Towards Self-Managing Networked Cyber-Physical Systems

Jan Janak

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2024

© 2024

Jan Janak

All Rights Reserved

Abstract

Towards Self-Managing Networked Cyber-Physical Systems

Jan Janak

Networked systems integrating software with the physical world are known as cyber-physical systems (CPSs). CPSs have been used in diverse sectors, including power generation and distribution, transportation, industrial systems, and building management. The diversity of applications and interdisciplinary nature make CPSs exciting to design and build but challenging to manage once deployed. Deployed CPSs must adapt to changes in the operating environment or the system's architecture, e.g., when outdated or malfunctioning components need to be replaced. Skilled human operators have traditionally performed such adaptations using centralized management protocols. As the CPS grows, management tasks become more complex, tedious, and error-prone.

This dissertation studies management challenges in deployed CPSs. It is based on practical research with CPSs of various sizes and diverse application domains, from the large geographically dispersed electrical grid to small-scale consumer Internet of Things (IoT) systems. We study the management challenges unique to each system and propose network services and protocols specifically designed to reduce the amount of management overhead, drawing inspiration from autonomic systems and networking research.

We first introduce PhoenixSEN, a self-managing ad hoc network designed to restore connectivity in the electrical grid after a large-scale outage. The electrical grid is a large, heterogeneous, geographically dispersed CPS. We analyze the U.S. electrical grid network subsystem, propose an ad hoc network to temporarily replace the network subsystem during a blackout, and discuss the experimental evaluation of the network on a one-of-a-kind physical electrical grid testbed. The novel aspects of PhoenixSEN lie in a combination of existing and new network technologies and manageability by power distribution industry operators.

Motivated by the challenges of running unmodified third-party applications in an ad hoc network like PhoenixSEN, we propose a geographic resource discovery and query processing service for federated CPSs called SenSQL. The service combines a resource discovery protocol inspired by the LoST protocol with a standard SQL-based query interface. SenSQL aims to simplify the development of applications for federated or administratively decoupled autonomous cyber-physical systems without a single administrative or technological point of failure. The SenSQL framework balances control over autonomous cyber-physical devices and their data with service federation, limiting the application's reliance on centralized infrastructures or services.

We conclude the first part of the dissertation by presenting the design and implementation of a testbed for usability experiments with mission-critical voice, a vital communication modality in PhoenixSEN, and during emergency scenarios in general. The testbed can be used to conduct human-subject studies under emulated network conditions to assess the influence of various network parameters on the end-user's quality of experience.

The second dissertation part focuses on network enrollment of IoT devices, a management process that is often complicated, frustrating, and error-prone, particularly in consumer-oriented systems. We motivate the work by reverse-engineering and analyzing Amazon Echo's network enrollment protocol. The Echo is one of the most widely deployed IoT devices and, thus, an excellent case study. We learn that the process is rather complicated and cumbersome.

We then present a systematic study of IoT network enrollment with a focus on consumer IoT devices in advanced deployment scenarios, e.g., third-party installations, shared physical spaces, or evolving IoT systems. We evaluate existing frameworks and their shortcoming and propose WIDE, a network-independent enrollment framework designed to minimize user interactions to enable advanced deployment scenarios. WIDE is designed for large-scale or heterogeneous IoT systems where multiple independent entities cooperate to set the system up. We also discuss the design of a human-subject study to compare and contrast the usability of network enrollment frameworks.

A secure network must authenticate a new device before it can be enrolled. The authentication step usually requires physical device access, which may be impossible in many advanced deployment scenarios, e.g., when IoT devices are installed by a specialist in physically unreachable locations. We propose Lighthouse, a visible-light authentication protocol for physically inaccessible IoT devices. We discuss the protocol's design, develop transmitter and receiver prototypes, and evaluate the system. Our measurements with off-the-shelf components over realistic distances indicate authentication times shorter or comparable with existing methods involving gaining physical access to the device. We also illustrate how the visible-light authentication protocol could be used as another authentication method in other network enrollment frameworks.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my gratitude to my advisor, Prof. Henning Schulzrinne, without whom this work would not have been possible. Prof. Schulzrinne taught me many valuable lessons, provided opportunities I never imagined would be available, and showed me how to think critically and independently. But above all, and for the first time in my life, he let me experience true academic freedom. He has shown me that a highly accomplished person can still be approachable, level-headed, and just generally nice. His sense of purpose, approach to impactful work, and ability to unconditionally motivate and nurture students is what I aspire to perpetuate in my own work.

I would like to thank my dissertation committee members, Prof. Vishal Misra, Prof. Dan Rubenstein, Prof. Xiaofan (Fred) Jiang, and Prof. Asaf Cidon. Their feedback, encouragement, and constructive comments have significantly improved my research and this dissertation. I hope they allow me to tap into their experience and wisdom in my future research.

I was fortunate to have the opportunity to work with other brilliant graduate students and researchers, particularly other members of the Internet Real-Time Laboratory. They have provided helpful comments, spirited discussions, and invaluable feedback. People who deserve special credit are Luoyao Hao, who supervised the essential research that inspired and grounded the work on network enrollment; Artiom Baloian, a co-developer of the software that enabled PhoenixSEN; Caspar Lant, who made invaluable contributions to the IoT infrastructure for the SenSQL research; and Kahlil Dozier who conducted experiments with the mission-critical voice testbed.

My sincere thanks also go to Jae Woo Lee, who has always been there when I needed it and whose friendship, support, and mentoring meant the world when the going got tough.

I had the privilege to supervise many brilliant students while working on this dissertation. The complete list of names would be far too long to fit here, but I at least acknowledged the key people in the respective chapters. To the rest, please know that I am truly grateful for all your help and contributions. You have made this work possible, and I consider myself truly lucky that I got to work with all of you.

The DARPA RADICS program financially supported my work on the PhoenixSEN network and provided the scope and direction for this work. I have greatly benefited from interactions with the colleagues from BAE Systems working on the project. In particular, Dr. Hema Retty deserves special credit for her support, the Herculean effort to keep the project going, and patience while answering my many questions about the U.S. electrical grid.

NIST's Public Safety Communications Research (PSCR) Division provided financial support for the work on the mission-critical voice testbed. I had the privilege to regularly meet with NIST PSRC team, whose members provided direction, feedback, and a fertile and productive discussion environment. Specifically, I would like to thank Jason Kahn for his feedback, helpful feedback and discussions, and valuable introductions to others in the field.

Last but not least, I would like to express my deepest gratitude to my parents, my sister, and her family, who have always encouraged and supported me and patiently endured my absence while I was away and busy with my research. There would be no dissertation without their patience, unwavering support, and understanding. This dissertation is dedicated to them.

To my family.

Chapter 1: Introduction

Networked computing systems interacting with the physical environment are known as cyber-physical systems (CPSs). Such systems have been used in diverse sectors, including power generation and distribution, smart cities, health care, agriculture, transportation, and building management. The diversity of applications makes CPSs exciting to design and build and challenging to operate and manage. Thus, they have received significant attention from the research community.

Despite the diversity of applications, CPSs share a common architectural and operational model, where embedded computing nodes equipped with physical-environment sensors and actuators communicate with one another to provide functionality that individual nodes could not provide. The network infrastructure and related services typically become critical components of the overall system, determining its reliability, scalability, and predictability.

The dependency of CPS devices on the network, often on the public internet or the cloud, may not always be apparent. Consider Amazon Echo, a popular Internet of Things (IoT) device sold by Amazon, which is unusable without an internet connection or access to the Amazon Cloud, even during intermittent network failures [1]. A similar dependency on network connectivity also exists in the United States (U.S.) electrical grid, a large geographically dispersed CPS, where a functioning communication network is required to bring the grid into an operational state, e.g., after large-scale power disruption [2].

The research community has paid considerable attention to the design of scalable network architectures, protocols, and services for CPSs. Significant effort has also been dedicated to optimizations for resource-constrained environments [3, 4, 5]. The architectural building blocks necessary to design truly large-scale networked CPSs exist. The availability of affordable general-purpose computational platforms (SoCs) combined with the inherent scalability of the internet architecture [6] paves the way for planetary-scale CPSs.

Scalable and resource-efficient system design, although essential, is generally insufficient for practically deployable systems and does not necessarily lead to CPSs that are also manageable once deployed. Deployable CPSs must be able to accommodate outdated components and changes in the operating environment or the system's architecture. Suppose the CPS becomes part of critical infrastructure. In that case, it should additionally be designed to support long-term evolution and organic growth, i.e., the ability of system operators to add or replace individual components or modules without disrupting the entire system. Such operations, collectively known as system or network management, are the job of the system or network administrator. The administrator typically has a variety of management protocols [7, 8, 9] and tools at their disposal.

However, due to the complex and heterogeneous nature of CPSs, management remains a manual activity requiring a "human in the loop". These tasks contribute to the management complexity of the system. In other words, management complexity represents operations that must be performed on a deployed CPS system to ensure that the system remains operational, efficient, or secure.

An exciting category is cyber-physical systems of systems (SoS), i.e., large-scale, heterogeneous, CPSs that span geographical or administrative boundaries. Cyber-physical systems of systems have found applications in defense, transportation, Industry 4.0, and healthcare domains. Individual components within a SoS are usually complex and managed by independent administrators. Suppose management decisions or operations need to span administrative boundaries. Management complexity can become a critical problem in such systems, primarily if the involved administrators are not motivated to cooperate or share critical information. Management complexity in a large-scale cyber-physical system of systems could potentially limit its reliability and future growth.

This dissertation studies management complexity in various classes of CPSs ranging from large, geographically dispersed, federated systems such as the U.S. electrical grid to small-scale consumer IoT systems. The focus is on federated systems, advanced deployments, and shared physical spaces. The grid is probably the largest deployed CPSs, presenting systems and network designers with many unique deployment and management challenges [10]. IoT systems deployed in shared physical spaces, such as a large university campus, face similar challenges, especially if the IoT system needs to support federated deployment scenarios. On a small scale, i.e., in consumer IoT systems, the dissertation focuses on the problem of IoT device network enrollment in advanced deployment scenarios. Collectively, these systems cover a significant portion of the hypothetical cyber-physical system complexity spectrum. The dissertation aims to compare and contrast the unique challenges in each system to draw broadly applicable conclusions.

We attempt to address some of the management complexity encountered in the systems mentioned earlier by applying selected autonomic principles and techniques [11]. We characterize the types and contexts of management complexity and attempt to identify the root causes. We use autonomic design patterns in the design of new protocols and network services to help the target CPS reduce management complexity in specific scenarios. We discuss how the protocols and network services help reduce or eliminate the need for manual intervention by the user or system administrator.

1.1 Problem Formulation

Many existing CPSs, large and small, exhibit symptoms of management complexity. Consider the Merrimack Valley incident, where a routine supervisory control and data acquisition (SCADA) system upgrade had catastrophic consequences on the environment [12]. Some deployed CPSs, such as the U.S. electrical grid, could be considered critical infrastructure, i.e., infrastructure vital to the nation's security or safety. Management complexity in critical CPSs could result in security issues or failures [13], affecting the safety and security of the nation or society that relies on it. This problem has been recognized as essential by the Defense Advanced Research Projects Agency (DARPA) and others [2, 14, 15].

In the absence of decentralized or automated management abstractions, some vendors have restored to implementing centralized architectures to curb management complexity. For example, many consumer IoT devices communicate exclusively with a manufacturer-operated cloud, which provides a unified management point to the manufacturer and the end user. Even SCADA systems that typically do not rely on the public Internet or the cloud are often highly centralized. Apart from introducing a single point of failure, centralized systems are challenging to compartmentalize or federate and can suffer from insider attacks or privilege abuse [16, 17].

Ad hoc attempts at reducing management complexity in cyber-physical systems have often resulted in systems that are:

- **highly centralized** with bottlenecks or single points of failure;
- **manually configured** with hardcoded addresses, names, and other parameters;
- **insecure** if the system is difficult to modify to apply fixes;
- **static**, that is, difficult to extend or expand to support new services or applications.

Management complexity is not a new problem. The long-term rapid growth of the Internet, which exhibits a similar problem, prompted the research community to explore autonomic techniques to help networked systems manage themselves. Autonomic techniques and principles are rarely applied to cyber-physical system design. Consequently, autonomic network protocols, services, and architectures are unavailable to CPS practitioners, leading to practical systems that are brittle, difficult to manage, or extend.

Network management includes many operations commonly performed on a deployed system, including enrollment, configuration, integration, fault detection and discovery, and coordination of the hardware and software that make up the system [18]. If these operations require significant involvement from the human system administrator, the resulting system could be unreliable and difficult to operate or extend.

1.2 Contributions

This dissertation studies management complexity in various cyber-physical systems, ranging from a large, geographically dispersed, federated electrical grid infrastructure to small-scale consumer IoT systems. We design, develop, and evaluate new network services and protocols based on selected autonomic principles to help mitigate unnecessary management complexity. The individual contributions made in the dissertation can be organized into two parts: (1) decentralization of CPS management (PhoenixSEN, SenSQL, MCV Testbed), and (2) improving the usability of IoT network enrollment (Echo Pairing, WIDE, Lighthouse). Figure 1.1 places the individual contributions in the context of a hypothetical large, federated cyber-physical system.

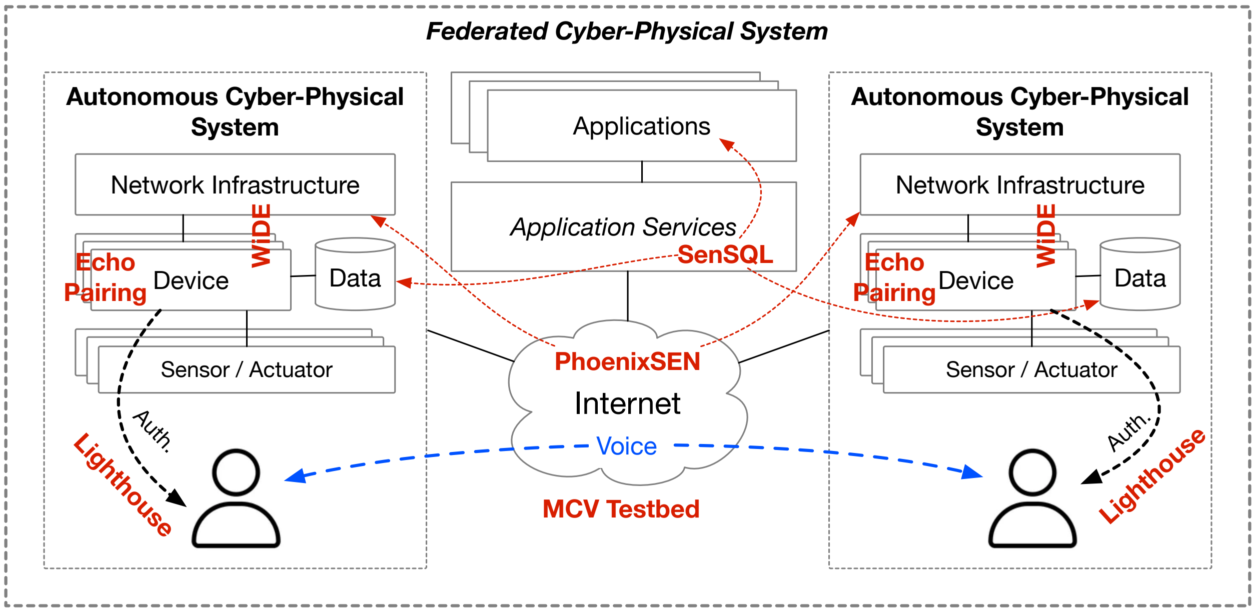


Figure 1.1: The context, positions, and relationships of individual thesis contributions within a hypothetical large, federated cyber-physical system.

The first part of the dissertation can be viewed as an effort to decentralize network operations, administration, and management in large-scale CPSs. Centralization in internet protocols, forces that encourage it, and ways to address undesired centralization have been recently discussed in the research community [19, 20].

Chapter 3 proposes **PhoenixSEN, a self-managing ad hoc backup network** for the U.S. electrical grid designed to restore connectivity in emergencies. We discuss the design of the network and associated services and describe the evaluation of PhoenixSEN on a one-of-a-kind electrical grid testbed by DARPA. The grid is a large CPS, which makes the PhoenixSEN approach broadly applicable to systems of similar size and complexity.

Obtaining a global view in a dynamic ad hoc network like PhoenixSEN proved challenging for applications. To address the problem, Chapter 4 proposes **SenSQL, a geographic resource discovery and query service** for federated cyber-physical systems. The SenSQL service combines a geographic discovery service with a standard SQL-based query interface in a novel way to help the application cope with the deployed system's dynamic or evolving nature. We show how SenSQL can be used to design portable applications spanning federated cyber-physical systems.

The DARPA RADICS program [21] stressed the importance of voice communications in emergencies. Chapter 5 presents the design and implementation of a **mission-critical voice (MCV) testbed** that can be used to design, conduct, and evaluate human-subject experiments to improve the quality of experience of first responder communication. Although initially developed as part of an unrelated NIST-funded project, the approach broadly applies to various human-in-the-loop communication scenarios over a degraded or impaired network infrastructure, such as PhoenixSEN.

The second part of the dissertation presents research aiming to improve the usability of IoT device network enrollment, a common management task that often results in unnecessary end-user confusion, frustration, and generally poor user experience with consumer IoT systems.

Chapter 6 presents **an analysis of Amazon Echo's pairing protocol**. The Echo is one of the most popular IoT devices and thus an excellent case study. To the best of our knowledge, this is the first attempt to decrypt, analyze, and document the Echo's network communication.

In Chapter 7, we revisit the IoT network enrollment problem. We analyze a variety of existing solutions from a usability perspective and propose **WIDE, a network enrollment framework for advanced deployment scenarios**. WIDE considers the entire lifecycle of IoT devices, supports shared physical spaces, and is link-layer agnostic. We also discuss the design of an IRB-approved human-subject usability study.

A secure network must authenticate a new IoT device before it can be enrolled. This step usually requires physical device access, which may be impossible to obtain in many deployment scenarios, e.g., when IoT devices are installed by a specialist in physically unreachable locations. Chapter 8 proposes **Lighthouse, a visible-light authentication protocol for physically inaccessible IoT devices**. We discuss the protocol's design, develop transmitter and receiver prototypes, and evaluate the system. We find that the approach is feasible and provides reasonably short authentication times with commodity hardware in realistic settings.

1.3 Intellectual Merit

This dissertation includes research on cyber-physical systems of varying scale and complexity in several application domains. The variety helps compare and contrast such systems regarding the specific measures that can be applied to reduce management complexity. The design patterns and principles used in designing network services and protocols across various application domains will hopefully yield broadly applicable network abstractions for future cyber-physical systems.

The research into the emergency backup networks for the U.S. electrical grid includes an architectural model and operational consideration for the grid's network subsystem. To the best of our knowledge, this is the first attempt to analyze and document the subsystem for the networking research community. We hope this knowledge might inspire future research into applied electrical grid networking.

1.4 Broader Impact

Networked cyber-physical and Internet of Things systems are becoming more pervasive and integrated into various aspects of everyday life. For that reason, making such systems easier to manage is becoming increasingly important. The literature provides many examples of the impacts of management complexity in such systems, ranging from mere inconvenience to catastrophic failures that endanger property or even people's lives. This dissertation aims to advance the ability of practitioners to design and build self-managing cyber-physical systems.

The reliability of the U.S. electrical grid infrastructure, for example, has seen renewed interest recently not only due to the danger of network-based cyber attacks [10, 14] but also due to the increased frequency with which such large-scale systems are disrupted during environmental disasters [22, 23]. Understanding how to curtail management complexity in such systems could improve their robustness in emergencies.

The backup network for the U.S. electrical grid might also be helpful in areas unrelated to the grid, such as disaster relief efforts. Such efforts often undergo the same fundamental problem of re-establishing connectivity in areas where the primary network infrastructure has been rendered inoperable due to a natural disaster.

The research conducted for the thesis produced several freely available open-source software tools. The service discovery subsystem developed for the DARPA RADICS project has been released as open source [24]. The entire mission-critical voice testbed is open-source [25].

1.5 Organization

This dissertation is organized as follows. We begin with an introduction, problem formulation, and summary of contributions in Chapter 1. Chapter 2 provides a broader perspective and reviews selected background material.

The rest of the dissertation is divided into thematic chapters. Each chapter explores cyber-physical system management complexity at a different scale, in a different application domain, or from a different perspective. Each chapter includes a literature review and a summary of findings.

Chapter 3 discusses the design, development, and experimental evaluation of PhoenixSEN, an emergency self-managing network for the electrical grid. Selected autonomic design principles have been used in the design of PhoenixSEN. This chapter also analyzes the architecture and operational considerations of the U.S. electrical grid network subsystem.

In Chapter 4, we propose SenSQL, a geographic resource discovery and query processing framework for federated autonomous cyber-physical systems. This research was partially inspired by the challenges of running monitoring and forensic applications on PhoenixSEN, the highly dynamic ad hoc network discussed in Chapter 3.

Chapter 5 describes designing and implementing a testbed for experimental human-subject research into mission-critical voice communications.

Chapter 6 analyses and documents Amazon Echo's network enrollment protocol. This research aimed to understand how the Echo enrolls in the user's Wi-Fi network and how the device associates with the user's Amazon cloud account.

Chapter 7 explores the design of a scalable and usable enrollment framework for advanced IoT deployments. This research addresses a long-standing issue shared by many commercially available IoT devices–the difficulty one experiences enrolling such devices to a secure Wi-Fi network.

In Chapter 8, we discuss the design, development, and evaluation of a visible-light authentication protocol for physically inaccessible IoT devices. This work complements the work in Chapter 7, enabling additional IoT deployment scenarios.

Finally, Chapter 9 concludes the dissertation and summarizes our findings.

Chapter 2: Background

2.1 Cyber-Physical Systems: A Broader Perspective

Cyber-physical systems are engineered networked computing systems interacting with the physical environment [26]. The research and practitioner communities have commonly used several names to refer to such systems or their components, depending on the community, technology focus, or application domain.

Cyber-physical system (CPS) is a somewhat broader term whose origin can be traced to the 2006 National Science Foundation (NSF) Workshop on Cyber-Physical Systems. CPS is the term most commonly used by the research community to refer to networked computational systems that interact with the physical world [27]. CPSs integrate embedded computing, networking, sensing, and actuation. The original scope of CPSs included safety-critical or mission-critical systems in application domains such as avionics, health care, and transportation. Such applications often depend on closed-loop control and are often developed using formal software development and verification techniques. However, the scope has expanded to include other types of systems that do not necessarily have such strict requirements, for example, the electrical grid control subsystem.

The phrase *internet of things* (IoT) is widely used to refer to networks of physical "smart" devices with an emphasis on the use of internet protocols and services in such systems [28]. IoT systems do not necessarily have to use the public internet infrastructure but typically incorporate internet protocols in some manner. The benefits of incorporating internet protocols and technology include scalability, simplified application development, and an extensive pool of developers familiar with the technology. IoT systems are most commonly deployed in consumer applications, including smart homes and home automation, wearable technology, connected health applications, and similar. The low cost of embedded connected platforms has resulted in an explosive growth of connected devices that could be monitored or controlled over the internet.

The phrase *wireless sensor networks* (WSN), whose origin can be traced to the early DARPA Distributed Sensors Networks project [29], has been used to refer to networks of spatially dispersed mostly uniform sensor nodes. Such networks have been proposed for environmental or area monitoring in agriculture, industrial applications, and health care [30]. The main characteristics of a wireless sensor network (WSN) include energy-constrained (battery-powered) nodes with a uniform hardware and software architecture, wireless ad hoc networking, an ability to cope with node failures, and a centralized data collection or processing architecture.

Finally, an *embedded system* is a computer-based system embedded in a larger device or system whose primary function is supporting the operation of the device [31]. In the context of CPSs, an embedded system typically refers to the individual computing node, disregarding its interactions over the network. The earliest applications of embedded computing can be traced back to the origins of the microprocessor and micro-controller technologies in the 1960s [32]. The embedded computer typically monitors or controls the device, replacing custom hardware with a general-purpose reprogrammable computer. This trend can be viewed as a shift from custom designs to general-purpose designs customized by software. Embedded systems range from general-purpose to specialized. The main characteristics of an embedded system include task-specific software, real-time control characteristics, lower performance requirements, and cost sensitivity.

Table 2.1 summarizes the key features of the four classes of systems. A heterogeneous system is made of architecturally diverse components. Critical systems are those deployed in mission, life, or safety-critical applications. A real-time system can respond within a deadline. Closed-loop systems support components that regulate themselves via federated control loops. Such components make decisions locally when the more extensive networked system is too large or slow.

As seen from the table, there is a considerable overlap between cyber-physical systems and the Internet of Things. National Institute of Standards and Technology (NIST) has discussed the overlap in a technical report [26], comparing and contrasting cyber-physical systems with the Internet of Things. For this thesis, we adopt the view that an IoT system is a system where a majority of nodes

Table 2.1: A summary of features of various types of systems

	Embedded	**WSN**	**IoT**	**CPS**
Heterogeneous	Yes	No	Yes	Yes
Critical	Yes	No	No	Yes
Networked	No	Yes	Yes	Yes
Real-time	Yes	No	No	Yes
Closed-loop	No	No	Maybe	Yes
Example	Home appliances	Environmental monitoring	Building automation	Electrical grid

are specialized devices, some of the devices interact with the physical world, and the system is often programmed by non-software engineers (e.g., mechanical, electrical, or civil engineers). The distinction between cyber-physical and embedded systems is based on [33].

The rest of the thesis will interchange the terms cyber-physical systems and the internet of things, depending on context.

Regardless of which name one chooses, all four classes of systems typically share a common architectural and operational model, where embedded computing nodes equipped with physical-environment sensors and actuators communicate with one another to provide functionality that individual nodes could not offer. Often, the network infrastructure becomes a critical component of the overall system, determining the reliability, scalability, and predictability of the system and the devices that rely on it. The dependency of devices on the network (and by proxy on the internet or the cloud) is not always apparent to the user. Consider Amazon Echo, a popular IoT device sold by Amazon, which is unusable without an internet connection and access to Amazon's cloud, even during an intermittent network failure.

2.2 Consequences of Failures in Cyber-Physical Systems

Management complexity in large-scale systems can lead to sub-optimal operation or system failures. The consequences can be especially severe if the system additionally controls physical processes, as in many cyber-physical systems. The consequences of failures in systems controlling physical processes are well documented in the literature. Modern cyber-physical systems are increasingly interconnected and require network communication to function. If the cyber-physical system is additionally part of critical infrastructure, for example, the electrical grid, it can be increasingly a target of network-based cyber attacks [14]. When successful, the cyber attack might trigger failures with consequences in the physical environment.

One of the most prominent examples of a cyber-physical system failure caused by management complexity was the Merrimack Valley incident of 2018 [12]. During an ordinary planned upgrade, a sensor-actuator mismatch caused a catastrophic failure where high-pressure natural gas was released into a low-pressure distribution system. The incident caused a series of explosions and fires in almost 40 homes. At the time of the incident, workers replacing parts of the low-pressure system failed to transfer a sensor from the old system to the new one. Thus, the corresponding sensors and actuators were connected to different systems, rendering the SCADA system unable to regulate pressure. A SCADA monitoring center received alarms but could not prevent the incident because it had no control over the valves. This incident highlights the difficulty and dangers of performing routine upgrades in deployed cyber-physical systems.

Management complexity can expose the cyber-physical system to insider attacks. Consider the Maroochy water breach incident of 2000 in which a former contractor took control of 150 sewage pumping stations and released millions of liters of untreated sewage into local waterways over a three-month period [16]. This incident is often cited in the literature as an example of the physical damage that could occur if the cyber-physical (SCADA) system is not adequately secured. This prolonged insider attack was possible due to the lack of security on the SCADA system's communication links and insufficient logging and monitoring. The root cause, however, can be attributed to the difficulty of upgrading and reconfiguring deployed SCADA systems.

The consequences of the Merrimack Valley and Maroochy incidents were immediately apparent. However, that is not necessarily always the case. Consider the Stuxnet malicious worm substantially damaging nuclear material separation centrifuges via programmable logic controllers (PLCs) [34].

The worm performed a man-in-the-middle attack on the sensors to prevent the system from detecting abnormal behavior. It then changed the rotational speed of the connected motors in a problematic way to physically observe, causing the centrifuges to tear themselves apart slowly. The Stuxnet incident illustrates that even air-gapped (disconnected) cyber-physical systems need multi-layered preventive measures such as network segmentation, access control policies, software upgrade and patch management, and system-wide monitoring and logging.

The Stuxnet incident additionally shows that merely disconnecting a cyber-physical system from the public internet may not be adequate if the adversary is sufficiently robust. An internally connected but externally disconnected cyber-physical system may still need protective measures and may need to be capable of adjusting its behavior automatically without direct human intervention.

A 2011 analysis by the Idaho National Laboratory found that applying traditional information technology (IT) system protective and management measures to real-time energy delivery control systems is inadequate and could lead to a power disruption [15]. A 2015 cyber attack caused a six-hour blackout for customers in and around Ukraine's capital city of Kiev [13]. This problem was recognized as important by DARPA and others [21]. The 2019 blackout in Argentina, Uruguay, and Paraguay, believed to have been caused by an operator error, left 48 million people without power [35]. For comparison, the Boston-Washington metropolitan area has 50 million inhabitants. In 2019, a large-scale power outage in England affected over a million homes and severely disrupted the public transit system [36, 37]. Electrical grids are some of the largest and most heterogeneous cyber-physical systems deployed. Managing such large and heterogeneous systems is a challenging problem, particularly during emergencies.

2.3 Network Management

2.3.1 Network Management Architecture

A typical network consists of many complex, interacting pieces of hardware and software, including links, switches, routers, hosts, and other devices. Keeping the network operational is the job of the network administrator, who has a variety of network management protocols [7, 8, 9] to control and coordinate the devices at their disposal. Network management includes the deployment, enrollment, configuration, integration, and coordination of the hardware and software that make up the networked system. Due to the complex and heterogeneous nature of most networked systems, keeping everything up can be a challenging task. An extensive collection of network monitoring and management tools [38] have been developed to help the network administrator monitor and manage the networked system. Network management remains primarily a manual, "human in the loop" type of activity.

The network management subsystem comprises the architecture, protocols, and data a network administrator uses to monitor and manage the networked system. Figure 2.1 illustrates the architecture of a managed network.

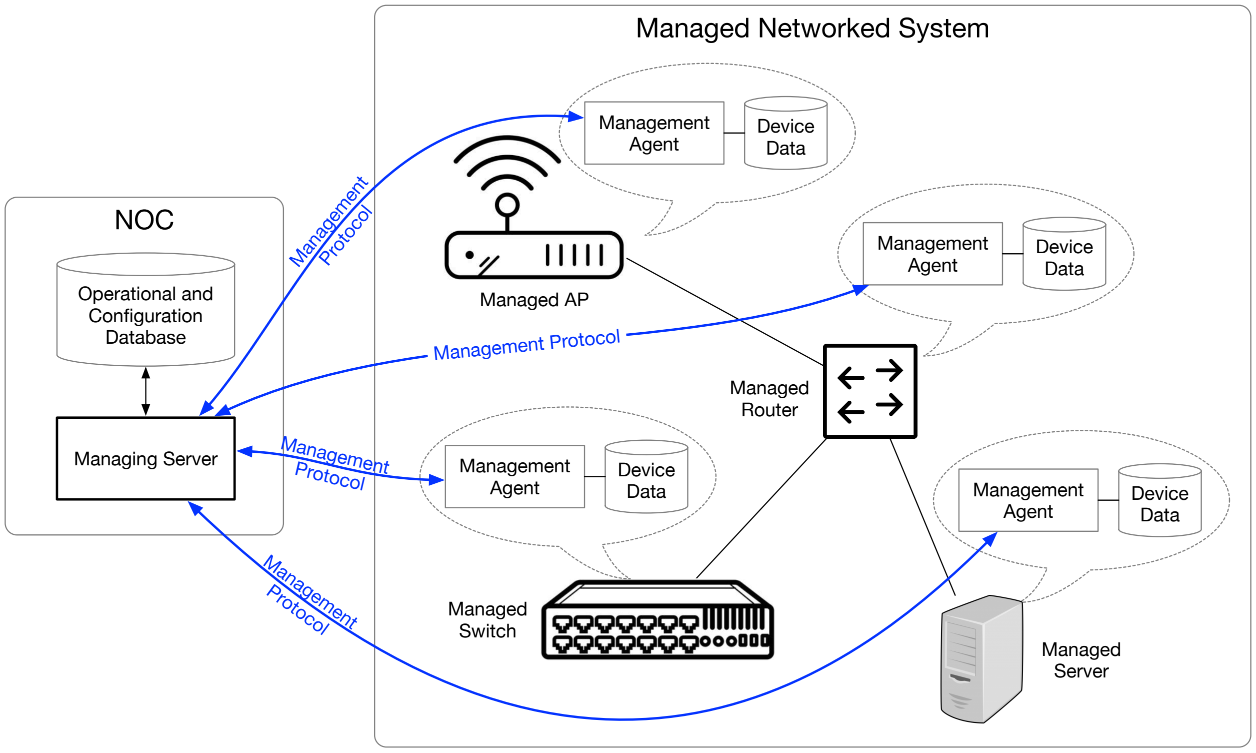


Figure 2.1: The conceptual diagram of components in a managed networked system.

The architecture has the following components:

- *Managing server*: an application, typically with a human in the loop, running on a centralized server or in a network operations center (NOC);
- *Managed devices*: network equipment in a managed network. Each device typically has

several manageable components and configuration parameters;

- *Data*: managed devices have data representing the device's state. The state includes operational data, configuration, and statistics;
- *Management agent*: a process running on the managed device that performs local actions based on requests from the managing server;
- *Management protocol*: a protocol that runs between the agent on the managed devices and the managing server. The protocol allows the managing server to query status, perform actions, and get notified of exceptions on the managed device.

There are three common approaches to network management: command line interfaces or vendor-specific protocols, Simple Network Monitoring Protocol (SNMP) and Management Information Base (MIB), and NETCONF/YANG. All three approaches fit into the general architecture shown in Figure 2.1 and require human input.

Managing networked devices via a command line interface, vendor-specific management protocol, or a web interface is the oldest approach. This approach requires extensive human involvement or complex management software to automate tedious and repetitive tasks. Consequently, this approach tends to be prone to errors and does not scale to large or heterogeneous networks.

SNMP [7] and MIB [39] is a network management protocol and device model standardized by the Internet Engineering Task Force (IETF) for Internet Protocol (IP) networks. SNMP allows the network operator to query or set data in the managed device and to subscribe to asynchronous notifications from the device. The managed device's configuration and state data are represented by the MIB. Some MIBs are device or vendor-specific, and others are device-agnostic, providing abstraction and generality. SNMP only allows managing devices individually and has scalability limitations.

NETCONF [8] and YANG [9] are network management protocols and data formats that provide a more abstract, network-wide, holistic approach to network management. NETCONF/YANG emphasizes configuration management and includes support for correctness constraints and atomic operations performed over multiple devices.

2.3.2 Autonomic Network and System Management

The growing complexity of networked computing systems, primarily driven by the rapid growth of the internet, is seen as a significant management problem by many in the research and industry communities. Current systems are managed mainly by highly skilled human operators who use the management protocols outlined in Section 2.3. However, reliance on highly skilled operators increases costs and limits future growth, heterogeneity, and usability. The increasing diversity of interconnected technology imposes an extra burden on existing networking infrastructure. Large systems often include mobile or embedded devices and use a combination of wired, fixed wireless (e.g., Wi-Fi), mobile wireless (e.g., cellular), and ad hoc networks. Mobile devices, the Internet of Things, and cyber-physical systems have primarily driven this trend. Managing such diverse, interconnected infrastructures via human intervention is becoming increasingly complex, time-consuming, and prone to errors.

Self-management has been suggested as a potential solution to the increasing management complexity in networked computing systems. Self-management is how a system manages its operation without direct human intervention. Self-managing systems adapt to changing conditions, constantly check and optimize operations, and make automated decisions based on high-level intents and policies provided by a human administrator, freeing the administrator from low-level management tasks [11]. The concept has been inspired by the autonomic nervous system, which unconsciously regulates critical bodily functions. The research community uses the terms autonomic computing and autonomic networking to refer to systems with self-managing characteristics.

An autonomic approach in the design of computing systems was first proposed by IBM in a 2001 manifesto, pointing out the difficulty of managing modern computing systems. As systems become more interconnected and heterogeneous, system architects can less anticipate interactions between the components, leaving more management tasks to be resolved by human operators once the system is deployed and operational [40]. IBM proposed the following four aspects of self-management:

- *self-configuration*: an ability of a system to automatically configure its components;
- *self-optimization*: an ability to monitor and optimize its operation and performance;

- *self-healing*: an ability automatically discover and correct local faults;
- *self-protection*: an ability to automatically detect and mitigate attacks and cascading faults.

The ultimate goal is for human operators to make less frequent, higher-level decisions for the underlying system to carry out automatically.

From an architectural perspective, an autonomic system is a collection of autonomic elements that interact with humans and each other. An autonomic element consists of ordinary managed components coupled with an autonomic manager. The manager governs the element's internal state, behavior, and interactions with other elements according to local knowledge and established high-level policies. Self-management can be viewed as an emergent property arising from interactions of autonomic elements.

Applying autonomic principles to network management has been an active research area for many years. Many abstract autonomic network architectures have been proposed in the literature [41]. In addition to the four abstract self-management aspects listed above, autonomic networking further applies the following principles:

- *compartmentalization*: division of the networking infrastructure into independent domains;
- *closed control loops*: eliminate external centralized systems from network's control loops;
- *rules, policies, intents*: network management via high-level primitives.

Recall that traditional network management relies on a centralized element (NOC) and human input, as shown in Figure 2.1. An autonomic network essentially distributes network management and attempts to eliminate the human factor as much as possible. Instead of a centralized configuration manager, the network contains a collection of communicating autonomic managers and associated managed components. An autonomic manager interacts with other managers to meet high-level network objectives. The manager monitors the managed components and executes commands to adjust their behavior. This behavior can be viewed as a closed control loop within the network. The inputs include signals from the management components and high-level policy-driven management rules. The outputs are commands to the managed components to adjust their behavior [42].

Figure 2.2 illustrates the difference between traditional and autonomic network management.

The diagram on the left represents traditional network management, where a human administrator manages individual network components using network-specific mechanisms such as SNMP, NETCONF, or vendor-specific command line tools. Traditional management is often automated, i.e., the administrator rarely configures managed components individually. However, even automated network management does not allow the network to respond to changes in its architecture or operating environment.

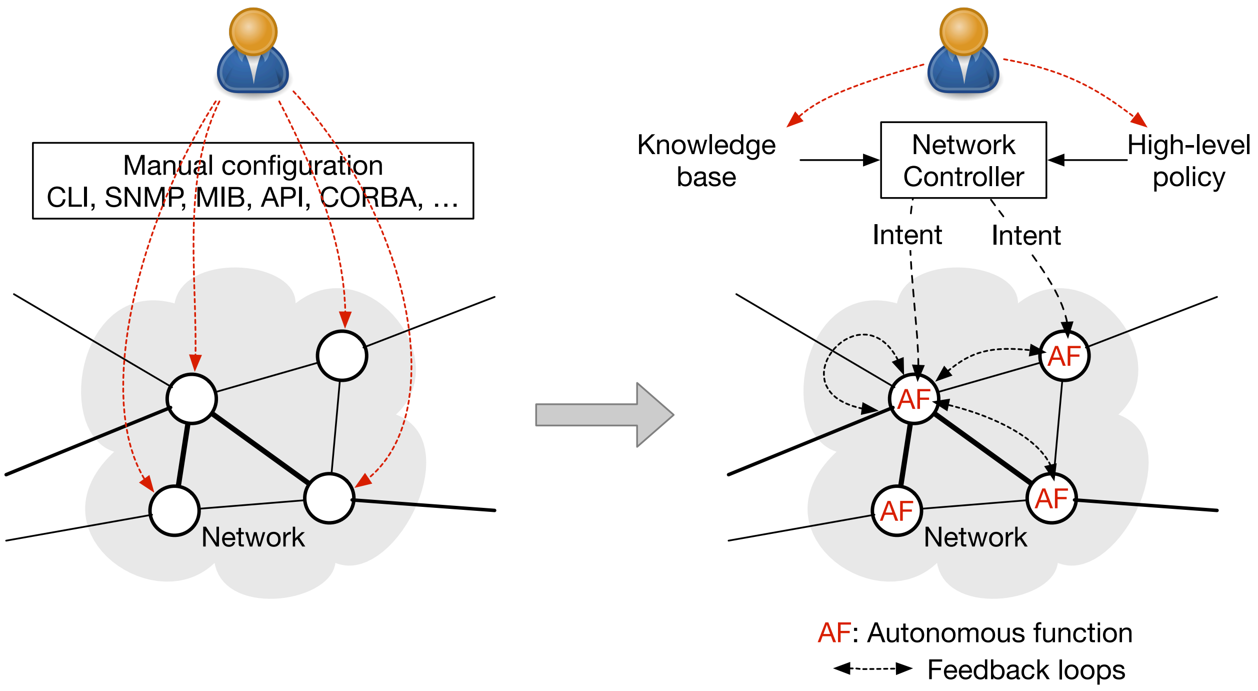


Figure 2.2: Evolution of network management techniques from manual to autonomic

The diagram on the right shows an abstract autonomic network management model where the administrator provides high-level input through local knowledge and high-level policies. An autonomic network control translates the input into high-level intents and distributes the intents to autonomic functions (AF) running on individual components within the network. The AFs establish feedback loops with one another to receive signals about the state of the network. The signals then trigger automatic state or behavior adjustments.

For the foreseeable future, autonomic networking control must coexist with traditional network management methods through centralized management systems. Most currently deployed networks benefit from a combined approach, where some functions are autonomic while others are centrally managed. Autonomic functions cannot always obtain all the necessary information on their own. For example, policy information is typically specified by humans and thus requires human input. Such information should be provided as intents, i.e., an abstract high-level policy determining network behavior. Ideally, intents do not contain network configuration or information applicable to a particular node.

The application of autonomic principles has been partially successful in some network architectures, for example, at the lower layers of the IP network architecture. Most internet protocols, however, depend on intelligence provided by external systems [11].

IP-based networks apply self-management, one of the autonomic principles, in routing protocols such as OSPF [43] and IS-IS [44], and could thus be considered autonomic. These protocols automatically build and manage a routing table on each node based on input (signals) from other nodes.

The Transmission Control Protocol (TCP) [45] is another example of a successful partial application of autonomic principles. TCP connections use internal closed control loops to regulate data transmission. However, the application must manually configure high-level parameters to establish and control the connection.

The Network Time Protocol (NTP) [46] manages time synchronization via closed control loops and could also be considered autonomic.

The research into autonomic network management has produced many mutually incompatible network architectures. The IETF started a working group called Autonomic Networking Integrated Model and Approach (ANIMA) [47] to develop specifications for interoperable protocols for autonomic network management of professionally-managed networks [48]. ANIMA has developed a reference model [49], an Autonomic Control Plane (ACP) [50], and the Generic Autonomic Signaling Protocol (GRASP) [51]. To our knowledge, the work produced through ANIMA has not been widely deployed yet.

Some of the work produced by the IETF ANIMA working group inspired parts of this thesis. The ANIMA reference model [49], specifically the configuration-less autonomic node and RFC

8368 [50], served as an inspiration for the design of configuration-less self-managing network nodes in Chapter 3. The "Decentralization and Distribution" design goal described in Section 3.4 of RFC 7575 [11] inspired the work in Chapter 4. The "Secure by Default" design goal inspired the work in Chapter 8. Finally, the "Full Life-cycle support" requirement in the same RFC inspired the work described in Chapter 7.

Chapter 3: Restoring Network Services in the Electrical Grid

This chapter presents the design, implementation, and experimental evaluation of PhoenixSEN, an ad hoc backup network for the U.S. electrical grid developed as part of the DARPA RADICS [21] project. Developing PhoenixSEN was a collaborative research project between DARPA, BAE Systems, and Columbia University.

As the thesis author, I was primarily responsible for designing and implementing various built-in network services, including virtualization, network-wide service discovery, VoIP, monitoring, and time synchronization. I also influenced the overall design of the IP network, including self-configuration, IP addressing, and routing.

3.1 Introduction

When imagining an extensive networked cyber-physical system, the electrical grid does not necessarily come to mind. Nevertheless, networking infrastructure plays an increasingly important role in the modern grid, which cannot function without coordination (communication) between producers and distributors of electric power. The networking subsystem in the electrical grid facilitates such communication. It coordinates devices producing and distributing electrical power and also enables real-time configuration and monitoring of devices, as well as communication between human operators in different parts of the grid. Thus, the electrical grid can be viewed as a large, geographically dispersed, and heterogeneous networked CPS.

Like all long-lived evolving infrastructures, the grid experiences outages. Most electric power outages are locally contained, and recovery can rely on the public or utility-owned communications infrastructure to coordinate restoration and energizing parts of the electrical grid. Large-scale electric power outages, a.k.a blackouts, are rare but do happen [52, 53, 35, 36]. Recovering from a large-scale outage typically follows a special procedure known as a black start. The procedure has been designed to allow the restoration of electricity supplies in a timely manner [54].

In the U.S., the black start procedure is usually managed by regional transmission organizations (RTOs) that coordinates several electric utilities. For example, PJM, a large RTO, designates specific generators and transmission infrastructure operators as critical for black start [55].

A successful black start requires coordination of electricity supply and demand, typically by incrementally adding both generating capacity and load [56]. Such coordination usually occurs via phone calls to substation[1] personnel with instructions about which grid branches to turn on or off or via real-time remote control of SCADA devices. Both cases require network connectivity.

Grid operators in the U.S. often rely on external internet service providers (ISPs) for network services [57]. If the ISPs are also impaired by the blackout, network connectivity may be difficult to guarantee. If the blackout is caused by a network-based cyber attack, the attacker may also attempt to actively thwart or delay power restoration, making a bad situation worse [13].

DARPA has recognized the danger network-based cyber attacks represent for the U.S. critical electrical grid infrastructure and launched the Rapid Attack Detection, Isolation, and Characterization Systems (RADICS) program [21]. The goal of the RADICS program was to design, develop, and evaluate a set of tools to aid the power generation and distribution industry in recovering from a hypothetical large-scale blackout. The toolset included a secured ad-hoc network, self-managing network services, and forensic and monitoring tools designed to incrementally restore grid connectivity while the primary ISP networks are unusable.

This chapter presents the design, prototype implementation, and experimental evaluation of Phoenix Secure Emergency Network (PhoenixSEN), an ad hoc network for electrical grid recovery. PhoenixSEN consists of a hybrid, isolated, self-forming network and network services designed to enable the coordination of power restoration. It combines existing and new technologies for rapid deployment into non-cooperative environments, can work with various link-layer protocols, and provides services for coordinating people and (SCADA) devices. PhoenixSEN is intended as a

[1]A substation is a component of an electrical grid. Substations transform voltage or perform other important functions. Electric power typically flows through several substations between the generator and consumer.

drop-in replacement for the grid's primary communication networks that may have been severely impaired during a large-scale blackout.

3.2 Motivation and Problem Statement

This work is primarily motivated by the need of the power distribution industry for backup network infrastructure that could be used to recover from a large-scale blackout caused by a network-based cyber attack [15]. A recent series of large-scale blackouts [35, 13, 36, 37] illustrates that such events are not merely hypothetical.

We assume that the grid control SCADA devices, as well as the network infrastructure itself maybe be compromised and could act maliciously. Therefore, the backup infrastructure should provide a means to reconnect healthy devices and keep those isolated from potentially compromised or malicious devices. This would allow for an incremental approach where devices are connected to the temporary (isolated) network only after they have been inspected and deemed healthy.

Like most complex networked systems today, the power grid relies, at least partially, on networks managed by external ISPs for remote command, control, and coordination of both machines (SCADA) and operators (voice). The network infrastructure itself will likely be severely affected by a large-scale blackout. This presents an interesting "chicken or the egg" dilemma. The network needs the power to connect the grid, but the grid cannot operate without the network. There needs to be infrastructure that will allow the grid to be temporarily self-sufficient, at least during the initial restart phase when the grid is not yet fully operational.

An ongoing cyber attack on power grid systems may attempt to thwart any restart attempts, leaving a large number of people without power for days or weeks [58]. Depending on the state of the power grid infrastructure, a full black-start recovery after a cyber attack may also take a long time, e.g., days or weeks, depending on the sophistication of the attack.

We envision a mostly self-configuring network infrastructure that can take advantage of various existing link layer technologies and can be deployed to geographically dispersed sites used by the power grid infrastructure after the blackout. We assume the person setting up the infrastructure will have a technical background but not necessarily in computer or network engineering. The network would provide the minimum services and bandwidth necessary to black-start the power grid securely.

3.3 U.S. Electrical Grid Architecture

The U.S. electrical grid is a complex, heterogeneous, geographically dispersed cyber-physical system that combines physical infrastructure for producing and delivering electric power with computer-based monitoring, management, and control. As the essential infrastructure that has been evolving for over a century and is subject to extensive government regulation, the grid has seen incremental upgrades and organic growth, resulting in considerable variability across geographical and political boundaries.

3.3.1 Interconnected Model

The grid's architecture is moving from a model with a small number of vertically-integrated[2] utility monopolies towards an interconnected model with a multitude of utility and non-utility companies coordinating to use shared transmission infrastructure, as shown in Figure 3.1. The interconnected model requires extensive data communications infrastructure. Thus, the networking subsystem is increasingly essential and critical for reliable grid operation.

In the U.S., hundreds of companies participate in producing and transmitting electric power. To ensure reliable grid operation, the Federal Energy Regulation Commission (FERC) has designated the North American Electric Reliability Corporation (NERC) to develop and enforce operational standards. Regional system operators coordinate electric power generation, transmission, and distribution. In some regions, the system operator is affiliated with a particular utility company. In other regions, the system operator is an independent entity known as the RTO (regional transmission operator) that coordinates multiple utilities. Some regions have an independent system operator (ISO) with a similar role. The differences are subtle and beyond the scope of this thesis. The

[2]Vertically-integrated utility controls all stages of the electric power supply chain: generation, transmission, and distribution to consumers.

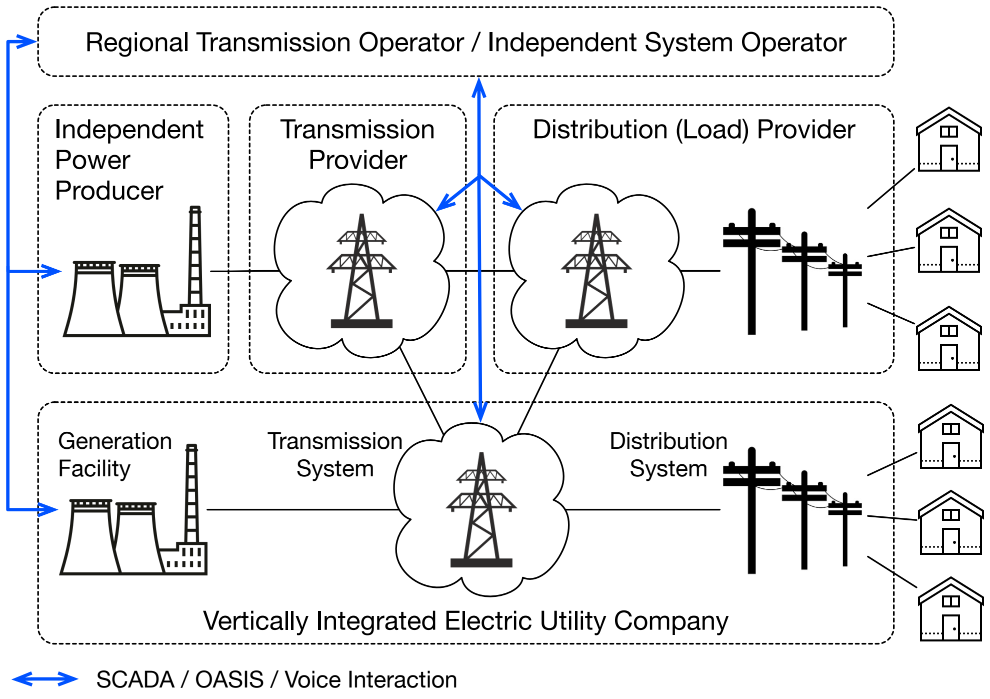

Regional Transmission Operator / Independent System Operator
Independent Power Producer
Transmission Provider
Distribution (Load) Provider
Generation Facility
Transmission System
Distribution System
Vertically Integrated Electric Utility Company
SCADA / OASIS / Voice Interaction

Figure 3.1: Distributed grid model where electricity market participants use shared transmission infrastructure coordinated by a regional transmission operator (RTO) or independent system operator (ISO).

RTO/ISO operates a wholesale electricity market, guarantees non-discriminatory access to shared grid infrastructure, and ensures reliable grid operation and compliance with NERC standards. The actual electric power generation, distribution, metering, and billing are provided by utility and non-utility companies coordinated by the RTO/ISO.

3.3.2 Network Subsystem

The major elements of an electric grid are the devices that produce and transmit electric power, IT, industrial control systems (ICSs), and the underlying network infrastructure. Since electric power is generated and consumed almost instantaneously, the grid must be coordinated to match power generation with demand in real time. Figure 3.2 provides an overview of the U.S. grid communications infrastructure.

NERC operational standards provide high-level guidance to ensure reliable grid operation. The exact implementation details of power grid systems are left to the RTOs/ISOs and utilities. As

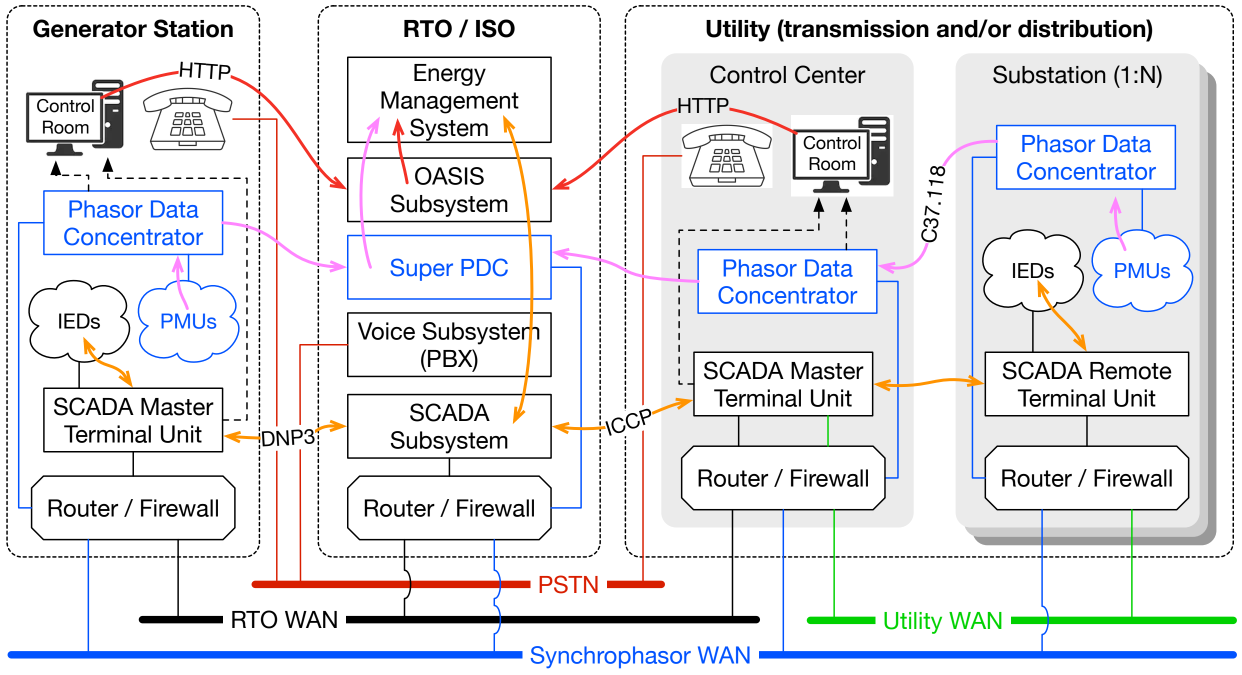


Figure 3.2: A detailed overview of the communications infrastructure required to keep the grid operational and reliable. The diagram shows a simplified architectural model. The existing grid exhibits considerable variety across geographic and political boundaries.

a result, existing grid systems are heterogeneous and often use many devices that communicate with mutually incompatible protocols. Early substation automation devices communicated using proprietary protocols over industry-specific buses or serial links. Modern-day substation systems use standardized process automation protocols such as Distributed Network Protocol 3 (DNP3) [59], the Inter-Control Center Communications Protocol (ICCN) [60], or IEC 61850 [61], and reuse existing wired and wireless network technologies.

The flow of electric power through the grid is managed by an energy management system (EMS) program. The primary purpose of the EMS is to keep the grid operational and reliable in response to varying conditions such as the available generator pool, transmission capacity, and instantaneous load. Ideally, a single instance of the EMS with the ability to remotely control critical grid devices would be provided by the RTO/ISO for the entire region. In practice, individual transmission operators typically run their EMS to monitor and protect their assets.

The EMS obtains the data about available generation resources and transmission capacity for its scheduling and planning algorithms from the Open Access Same-Time Information System (OASIS), a standardized web-based management system [62] that serves as an interface between electricity market participants, transmission providers, and the RTO/ISO. FERC requires that each RTO/ISO must provide an OASIS node. Clients typically interact with the OASIS system by invoking Hypertext Transfer Protocol (HTTP) application programming interfaces (APIs) over the internet.

The operators of infrastructure deemed critical for the reliability of the grid by the RTO/ISO are required to provide the RTO/ISO with remote access to selected components for monitoring and control purposes. This is accomplished by integrating the RTO/ISO's and the operator's SCADA and synchrophasor[3] subsystems over a redundant wide area network (WAN) provided for this purpose by the RTO/ISO. The data obtained from these subsystems are used by the EMS to build a global view of the state of the grid. Furthermore, the EMS can use SCADA to remotely control grid devices in the field, e.g., circuit breakers. Not all grid participants need to be SCADA-capable and connected to the RTO/ISO WAN. Smaller entities sometimes rely on the internet for all communication with the RTO/ISO.

The previous paragraphs show that many different types of data networks are involved in managing the flow of electricity through the grid. The RTO/ISO operates a redundant WAN used to connect to the control centers of grid infrastructure providers critical for overall system reliability. The typical RTO/ISO WAN is a redundant Multiprotocol Label Switching (MPLS) network based on links leased from several external ISPs. If deemed critical for future grid systems, the synchrophasor subsystem will likely use a dedicated WAN with stricter latency and bandwidth guarantees.

The use of the public switched telephone network (PSTN) for human-to-human voice communications between the RTO/ISO and utility personnel is required by NERC. Despite an overall increase in remote instrumentation capabilities across the electrical grid, human-to-human voice communication remains the most important communication modality in emergencies, e.g., after a blackout. The phone system typically uses cellular or satellite phones as a backup to meet NERC's

[3]Synchronized phasors (synchrophasors) are time-synchronized precise measurements of the electrical sinusoid's magnitude and phase angle performed by sensors distributed across the electrical grid.

strict reliability requirements.

Each utility also operates a dedicated WAN spanning its (sometimes large) service area that connects its control center with all substations. The typical utility WAN is IP-based and uses a combination of public (ISP-owned) and non-public (utility-owned) network infrastructure. A substation with connected devices (e.g., SCADA) must also provide a field area network (FAN). The FAN connects substation automation devices and any remote devices (metering, data collection) within the substation's service area, e.g., a neighborhood. Due to the large variety in deployed automation devices, the FAN is perhaps the most heterogeneous network type and is typically based on a combination of wired and wireless technologies. The public internet (not pictured) is typically used for other communication, e.g., to access the RTO/ISO's OASIS portal or to transfer metering or billing information between the utility and its customers.

SCADA interactions between the RTO/ISO and utility control centers typically use standardized protocols such as the DNP3 [59], the ICCN [60], or IEC 61850 [61] carried over TCP/IP. The SCADA subsystem is a hierarchically organized system where the RTO/ISO's master terminal unit (MTU) communicates with the MTUs at utility control centers, which in turn communicate with remote terminal units (RTUs) deployed at substations.

The synchrophasor subsystem is based on a hierarchy of phasor data concentrators (PDCs) that aggregate and process IEEE C37.118 [63] data streams coming from phasor measurement units (PMUs) deployed across the grid.

3.4 Phoenix Secure Emergency Network (PhoenixSEN)

As part of the RADICS project, we designed, built, and evaluated PhoenixSEN, a self-configuring ad hoc network designed as a drop-in replacement for the grid's primary networks. Developing PhoenixSEN was a collaborative effort between DARPA, BAE Systems, and Columbia University. The thesis author was primarily responsible for designing and implementing the built-in network services for service discovery, time synchronization, VoIP, and network monitoring.

The network can be deployed after a blackout or under a network-based cyber attack to quickly and securely restore connectivity to control centers (CCs) and substations. PhoenixSEN requires minimal deployment configuration and provides built-in services for voice and SCADA coordination. Uniform hardware and software architecture simplifies deployment to remote substations. The network is compatible with various link technologies, including radio, Ethernet, fiber, or powerline, and can work with links operated by third parties.

3.4.1 Deployment and Network Formation

Figure 3.3 shows one possible deployment configuration. The network is designed to be deployed into geographic areas served by multiple utility companies. A PhoenixSEN node is deployed to each substation, CC, or relay site. The nodes are connected with short-distance and long-distance links, either existing utility-owned links or temporary third-party links provided by, e.g., the National Guard. PhoenixSEN creates a secure, isolated virtual network for each utility spanning its CCs and substations.

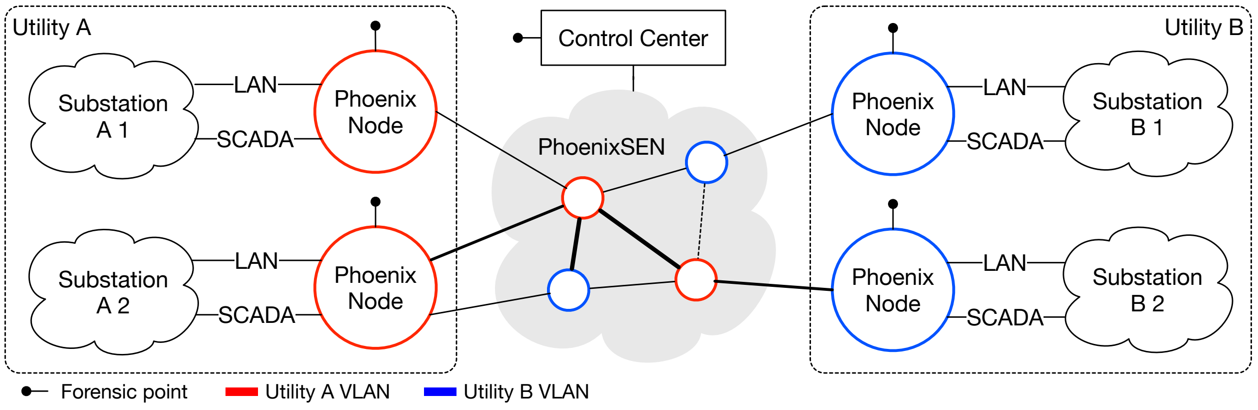


Figure 3.3: A PhoenixSEN node deployed at each substation connects the substation's VLANs to a virtual OLSR mesh network spanning all substations of the same utility. Per-utility virtual networks share a common physical infrastructure but are isolated from each other. Each PhoenixSEN node helps route packets for other utilities.

Each PhoenixSEN node provides a separate virtual LAN (VLAN) for voice, SCADA, and forensic traffic to its substation. All essential network services (DHCP, DNS, NTP, VoIP signaling) are provided locally to support communication within the substation even when its PhoenixSEN node is disconnected from the rest of the network. The addressing architecture of each VLAN is separately configurable. This allows the VLAN to match the original ISP network to minimize the need to reconfigure existing equipment.

A dedicated PhoenixSEN control center coordinates the deployment of PhoenixSEN. Without means to communicate with remote substation crews, the CC can use a one-way broadcast, e.g., a high-power radio, to transmit minimal PhoenixSEN node setup instructions. Once the substation is connected to PhoenixSEN, the CC crew can help configure its infrastructure remotely. PhoenixSEN hardware is designed for staff with technical background, but not necessarily in IT or networking. Detailed instructions are included to allow the substation crew to independently set up the PhoenixSEN node in a minimal operational configuration. Deployed nodes automatically form an ad hoc network based on the Optimized Link State Routing Protocol (OLSR) [64] that eventually connects all utility's sites and infrastructure.

When powered on, the PhoenixSEN node begins to search for other nodes on its link interfaces. Each interface has an instance of the OLSR daemon configured to discover other OLSR-enabled [64] nodes reachable over the interface. Gradually, PhoenixSEN nodes form an OLSR-based mobile ad hoc network (MANET) that eventually spans all substations and the CC. Once the network is formed, the CC can further configure and coordinate deployed PhoenixSEN nodes over the network.

While forming, PhoenixSEN provides connectivity in phases, gradually providing additional modes of communication as the system transitions from one phase to another:

1. Low-speed, broadcast-only communication from the CC to substations in the process of deploying a PhoenixSEN node.

2. Low-speed command and control connection between the CC and each substation. The connection provides just enough bandwidth for the CC crew to configure the PhoenixSEN node remotely.

3. The PhoenixSEN node is fully connected to the network, but the overlay may not have yet fully formed, or the node may be in the process of resolving an addressing conflict. The substation may not be able to reach all other substations of the same utility yet.

4. The node is fully connected and provides Voice over IP (VoIP), text, and SCADA communication between the CC and the substation crew.

3.4.2 Node Architecture

The PhoenixSEN node is designed to be deployed from a storage facility to substations by ground transportation or airlift shortly after a blackout. All PhoenixSEN nodes use a uniform hardware and software architecture to simplify deployment, installation, and configuration.

3.4.2.1 **Hardware**

All hardware comes in a weather-resistant enclosure that contains all essential components such as Intel Next Unit of Computing (NUC), Ethernet switches and cables, Global Positioning System (GPS), Wi-Fi access points, and VoIP clients. Figure 3.4 illustrates the hardware architecture.

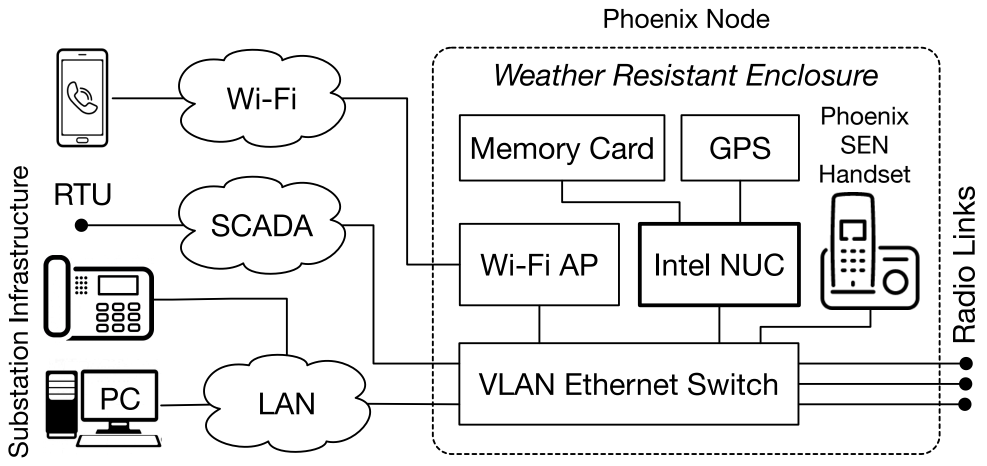


Figure 3.4: Hardware architecture of the PhoenixSEN node deployed to utility control centers and substations. The node provides wide area connectivity.

The PhoenixSEN node is based on the Intel NUC, a small form-factor computer, combined with a VLAN-capable Netgear Ethernet switch. The enclosure includes an Android smartphone, a pre-configured Yealink W52P cordless VoIP phone, a Wi-Fi access point, a Tripp Lite UPS, and assorted power and Ethernet cables. All devices within the enclosure are pre-configured and connected to Ethernet and power.

On the outside, each enclosure has four watertight Ethernet ports. These ports are pre-configured to be connected to external links, e.g., utility fiber, radio links provided by the National Guard, or Wi-Fi.

Figures 3.5 to 3.7 show the PhoenixSEN prototype built for evaluation in DARPA RADICS exercises. The complete prototype consisted of 21 PhoenixSEN nodes.

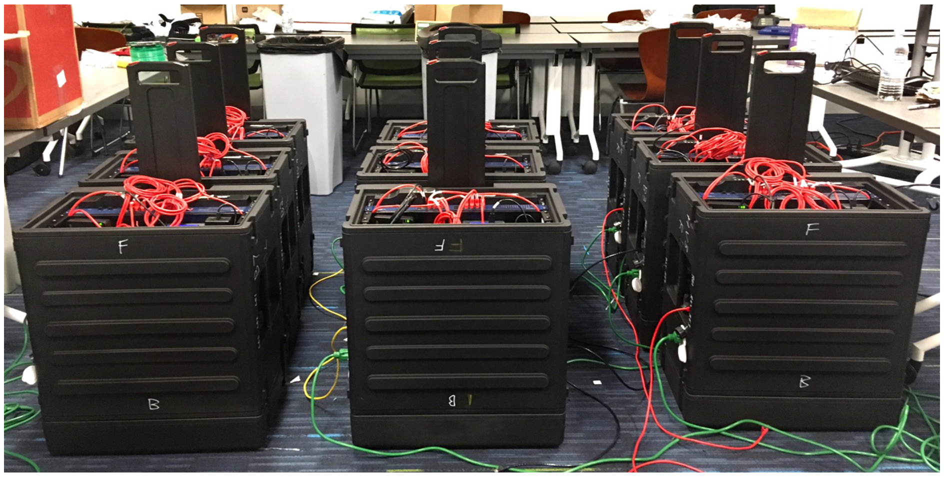


Figure 3.5: A portion of the PhoenixSEN prototype set up in a lab at the University of Illinois at Urbana-Champaign, September 2018. Each weather-resistant enclosure contains all the hardware for a single PhoenixSEN node. Photo author: Hema Retty. Used with permission.

3.4.2.2 Software

The PhoenixSEN node runs Ubuntu Linux 16.04 operating system (OS) in a minimal configuration. All custom software is pre-installed as Docker containers. The containers are built from source code and are specifically designed to support re-building in the field without internet access, e.g., after modifications to fix critical bugs or vulnerabilities.

Upon deployment, the substation crew provides the node with a utility company and substation identifiers, and the node connects to the corresponding per-utility PhoenixSEN virtual network. The node then creates isolated VLANs for voice, SCADA, and forensic traffic, each implemented with an isolated Linux networking namespace, and exposes the VLANs to the substation. The node also serves as a traffic router for other utilities in the same deployment area.

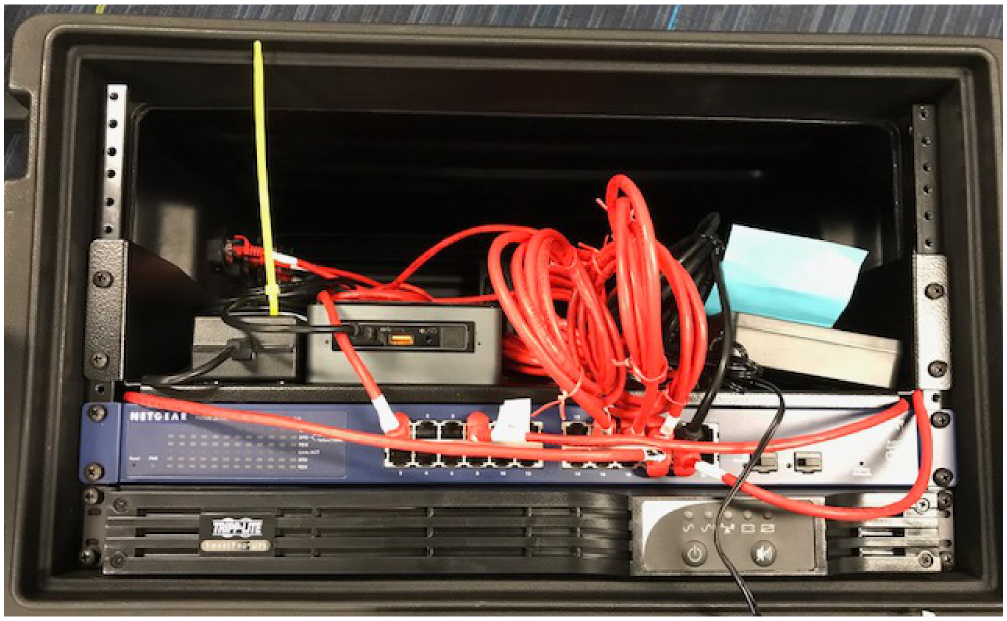


Figure 3.6: PhoenixSEN node in weather-resistant enclosure (Intel NUC, Yealink W52P phone, Netgear Ethernet switch, Tripp Lite UPS). September 2018. Photo author: Hema Retty. Used with permission.

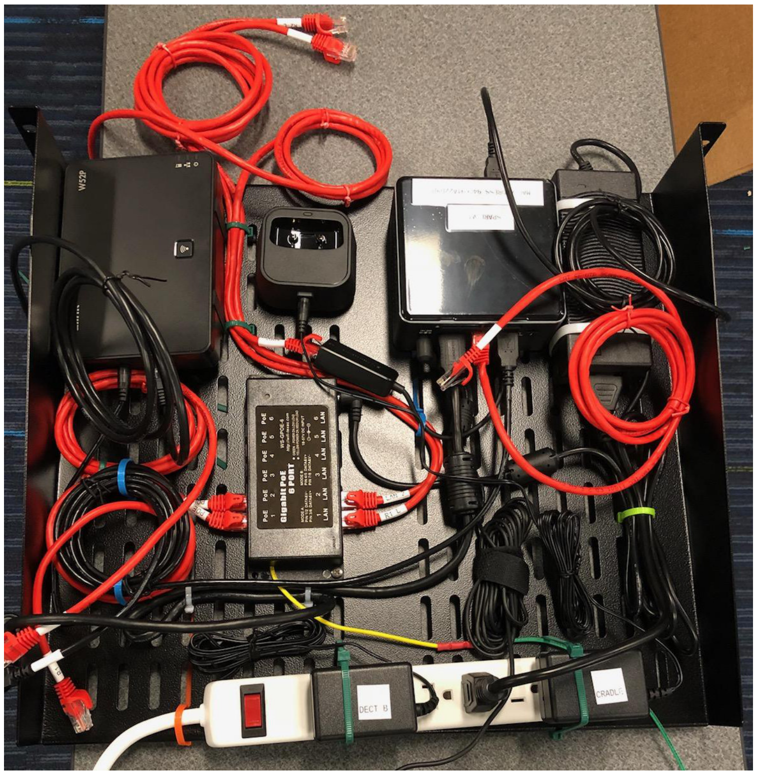


Figure 3.7: PhoenixSEN node components (Intel NUC, POE switch, Yealink W52P phone). September 2018. Photo author: Hema Retty. Used with permission.

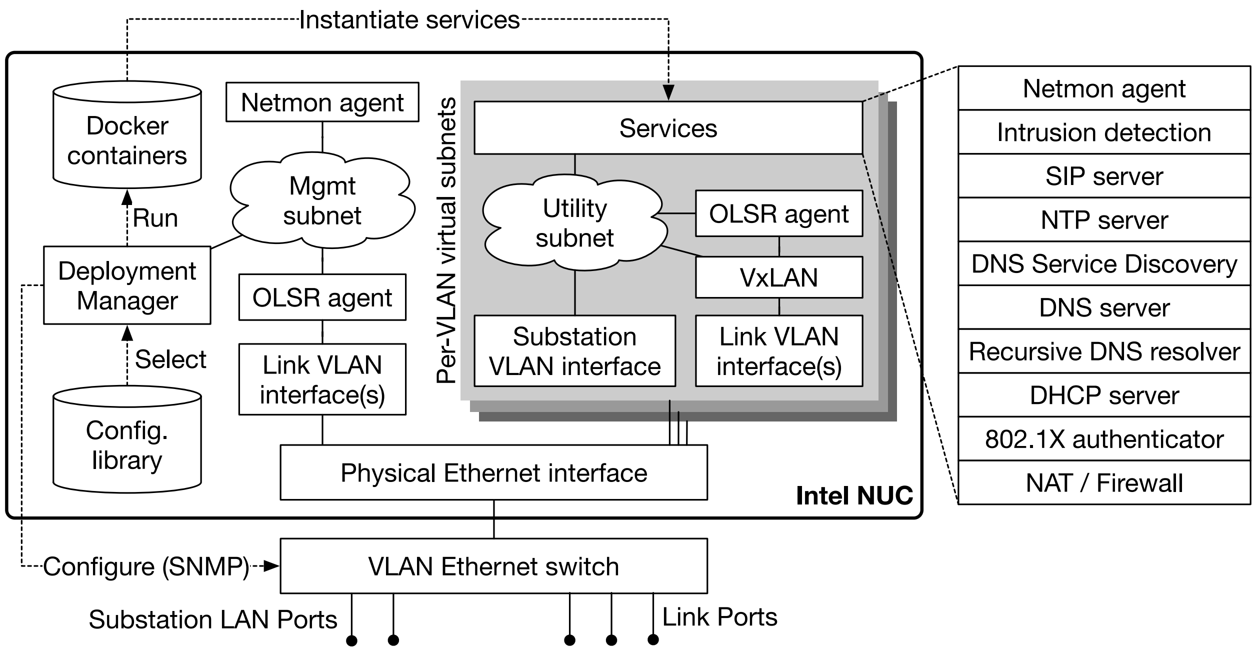


Figure 3.8: PhoenixSEN node software architecture. An isolated network environment (grey) with all required services is created for each substation LAN. The environments that belong to the same utility are connected across PhoenixSEN.

Each VLAN provides fully isolated network services to the substation, including network address translation (NAT), Dynamic Host Configuration Protocol (DHCP), Domain Name System (DNS), and NTP. Network service virtualization improves robustness and reliability under attack, e.g., if any of the substation SCADA devices have been compromised. The configuration for any of the services provided by PhoenixSEN within each VLAN (DHCP, DNS, and others) is generated by a configuration synthesis program based on a description of the configuration of the primary ISP networks. This step helps to eliminate or minimize the need to reconfigure substation devices as they are being reconnected to PhoenixSEN.

Each network environment runs its instance of the OLSR agent, which publishes the environment's IP subnet information to PhoenixSEN. This information is exchanged only with OLSR agents that belong to the same utility and environment type (e.g., SCADA). For example, utility A's SCADA environment on one Phoenix node connects to utility A's SCADA environments on all other PhoenixSEN nodes but not to utility B's SCADA environments.

The utilities served by the same PhoenixSEN can use conflicting or overlapping IP subnets (e.g., 192.168.x.y). Supporting such scenarios natively would require VLAN support on all PhoenixSEN links. To stay compatible with a wide variety of link layer technologies (including IP-only point-to-point links), the node does not require VLAN support on links. Instead, it passes all traffic through a Virtual Extensible LAN (VxLAN) [65] gateway, which encapsulates Ethernet frames in User Datagram Protocol (UDP) packets before transmission. A unique VxLAN virtual network identifier (VNI) is generated for each utility-VLAN combination during configuration synthesis.

3.4.3 Network Services

Each network environment provides fully isolated elementary network services to the corresponding substation VLAN such as DHCP, DNS, or NTP. In addition to the elementary services, each network environment also provides custom autonomic services for network-wide service discovery (Section 3.4.3.1), VoIP (Section 3.4.3.2), and network monitoring (Section 3.4.3.3). The following subsections describe the autonomic services.

The type of network environment determines what additional network services are created. For example, the VoIP environment provides a Session Initiation Protocol (SIP) [66] server on each PhoenixSEN node to keep the substation's phones operational and to provide a means of communication between the substation personnel and the CC. The SCADA environment offers additional services for intrusion detection and mitigating insider attacks by compromised devices.

3.4.3.1 Network-Wide Service Discovery

PhoenixSEN is a dynamic ad hoc network without a fixed structure. To help applications cope with changing network architecture, PhoenixSEN provides a built-in network-wide service discovery mechanism. Applications based on dynamic resource discovery tend to be more robust in case the network is impaired or partially formed. For maximum interoperability with existing applications, we designed PhoenixSEN service discovery around DNS service discovery (DNS-SD). A DNS-based service discovery provides backward compatibility with higher-layer protocols and services, e.g., TLS server certificate validation. The primary users are the voice subsystem (Section 3.4.3.2) and third-party network and SCADA forensic tools. Figure 3.9 illustrates the architecture of the service discovery subsystem.

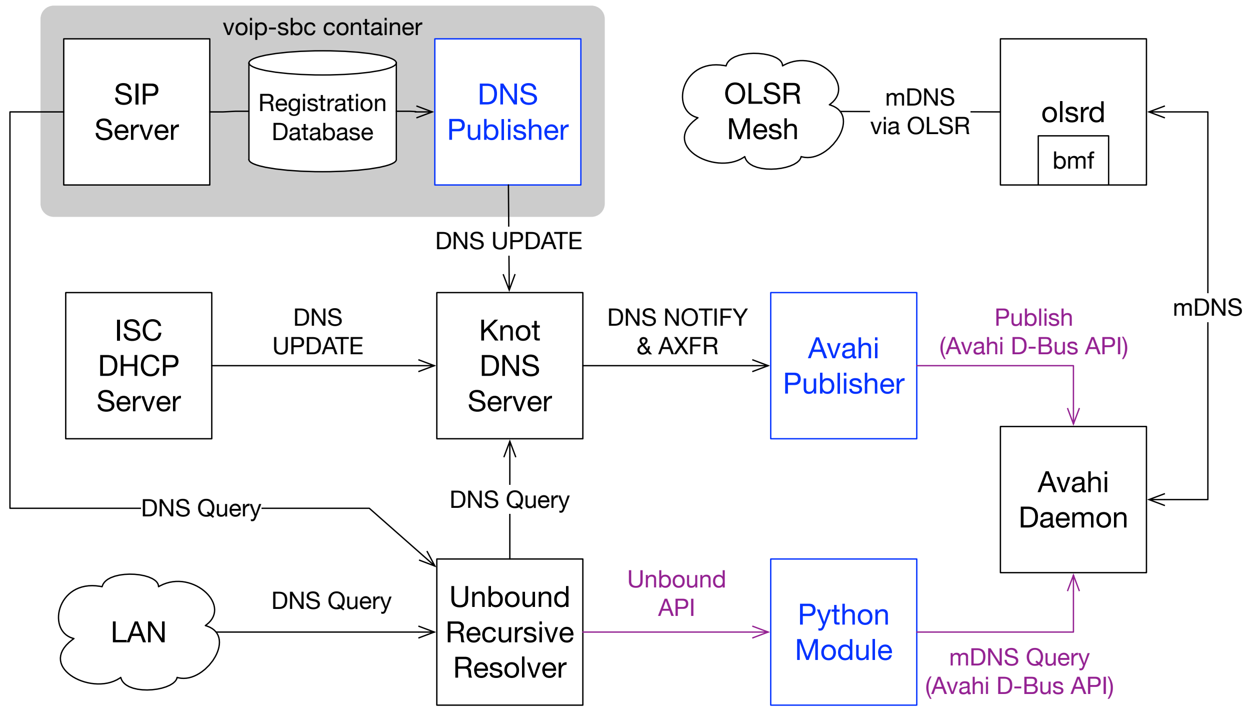


Figure 3.9: PhoenixSEN service discovery subsystem architecture. Every node runs the same services. Voice subsystem service discovery is shown in gray.

At the core of the service discovery subsystem is a peer-to-peer DNS service with no single authoritative master server. Every PhoenixSEN node runs a local DNS server (Knot) [67] that stores the DNS service description records published by local services and applications. A multicast DNS (mDNS) [68] publisher (Avahi) disseminates the local records across the whole PhoenixSEN network. Thus, DNS is always available on substation local area networks (LANs), even if the substation is isolated from the rest of the network. As more PhoenixSEN nodes join the network, DNS records gathered from other substations will become automatically available to clients.

Clients query the service description records via a recursive DNS resolver (Unbound) [69] also provided by every PhoenixSEN node. The resolver merges records from the local DNS server with the records obtained via mDNS from other PhoenixSEN nodes. This architecture provides an eventual consistency model. The client must resolve service description conflicts (rare in networks like PhoenixSEN).

Efficient network-wide IP multicast is implemented using a Basic Multicast Forwarding (BMF) plugin included with the OLSR agent (olsrd version 1) [70] running on each PhoenixSEN node. The plugin floods IP multicast packets to all nodes in the network along a spanning tree maintained by OLSR, using OLSR multi-point relays to optimize the flooding. We use the plugin to efficiently disseminate Avahi's mDNS packets across the entire PhoenixSEN.

The network-wide service discovery subsystem provides a simple and intuitive DNS-based interface to applications. An application on the PhoenixSEN node or in substation LAN can publish DNS-SD records via DNS UPDATE to the local DNS server to announce service availability. The application can query the local resolver for the corresponding DNS-SD records to discover remote services anywhere in the network.

To help future practitioners design and build similar systems, we published the network-wide service discovery subsystem as an open-source project at https://github.com/janakj/dns2avahi.

3.4.3.2 Voice and Chat

One of the goals of PhoenixSEN is to facilitate the coordination of people restoring grid operation in an emergency. Such coordination generally takes place via voice communications. PhoenixSEN provides built-in support for SIP [66] based real-time voice and text communication using the following modalities: two-party calls, multi-party conferencing, and text messaging (Figure 3.10). The included VoIP clients can place calls and send text messages to any site in any utility across the entire network. We designed the service to be autonomic, requiring no manual configuration.

Each PhoenixSEN node comes with a VoIP handset and an Android smartphone with a pre-installed SIP client. Both devices are pre-configured for immediate use. The PhoenixSEN node also provides a JavaScript WebRTC application that can turn any device with a compatible web browser into an additional VoIP and chat client. Every PhoenixSEN node runs a complete set of VoIP services. The clients register with the VoIP server on the PhoenixSEN node within the same substation. Local calls remain possible even if the substation is disconnected from PhoenixSEN.

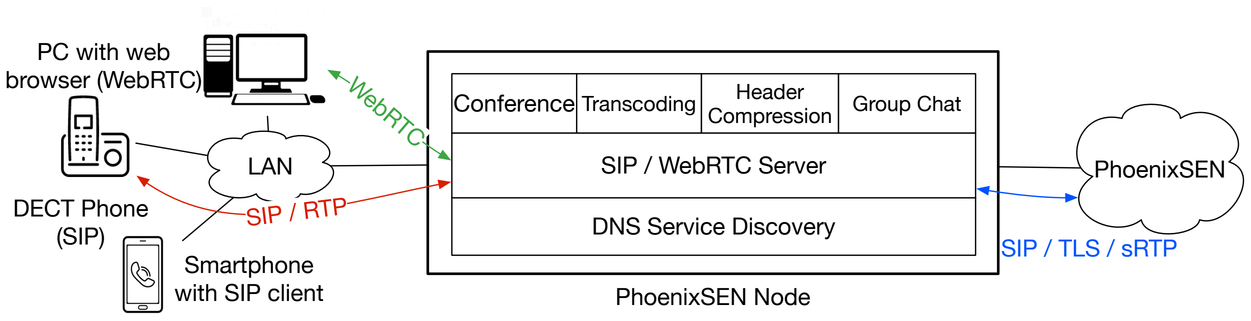


Figure 3.10: VoIP subsystem architecture with support for SIP and WebRTC. Network-wide DNS service discovery helps eliminate bottlenecks.

The VoIP server enforces transcoding, authentication, and encryption on all calls to PhoenixSEN.

The VoIP subsystem has a peer-to-peer architecture that requires no fixed dialing plan. VoIP clients and servers use network-wide service discovery (Section 3.4.3.1) to locate each other. Upon VoIP client registration, the VoIP server publishes a custom DNS service discovery record mapping the client's number to the name of the PhoenixSEN node where it has registered. For example,

```
4822._voip.phxnet.org. IN CNAME voip-phx23.phxnet.org.
```

where 4822 is a VoIP clients's phone number and `voip-phx23.phxnet.org` is the hostname of the PhoenixSEN node where the client is registered.

When a remote VoIP server receives a call for the client, it uses DNS service discovery to map the called number to the PhoenixSEN node where the client is registered and forwards the call there. The mDNS subsystem proactively disseminates DNS records across PhoenixSEN. Thus, calls to existing (registered) numbers are resolved from the local DNS cache in constant time. Calls to non-existing (unregistered) numbers take a few seconds to fail until multicast DNS reports that the corresponding record was not found.

The VoIP and chat subsystem uses a simple four-number dialing plan where each substation is allocated a fixed prefix. The substation's VoIP clients are assigned numbers from that prefix. Each PhoenixSEN node runs a full set of VoIP services (SIP and WebRTC servers, conference server, chat server). The clients register at the nearest SIP server, typically the one located on the PhoenixSEN node installed in the client's substation. Having a full set of VoIP services on each Phoenix node allows the node to function in isolation. Even if the node is disconnected from the network, calls and chats within its substation remain possible.

3.4.3.3 Network Monitoring

Monitoring and situation awareness are essential for successful deployment and operation in a geographically dispersed network such as PhoenixSEN. Netmon is a near real-time network monitoring service designed for PhoenixSEN. Through Netmon's web-based user interface (UI) (Figure 3.14), the CC crew can see the formed PhoenixSEN topology and any alerts generated by security events and incidents. Forensic teams can use Netmon to discover the devices at each substation and inspect their state while investigating a potential network-based cyber attack.

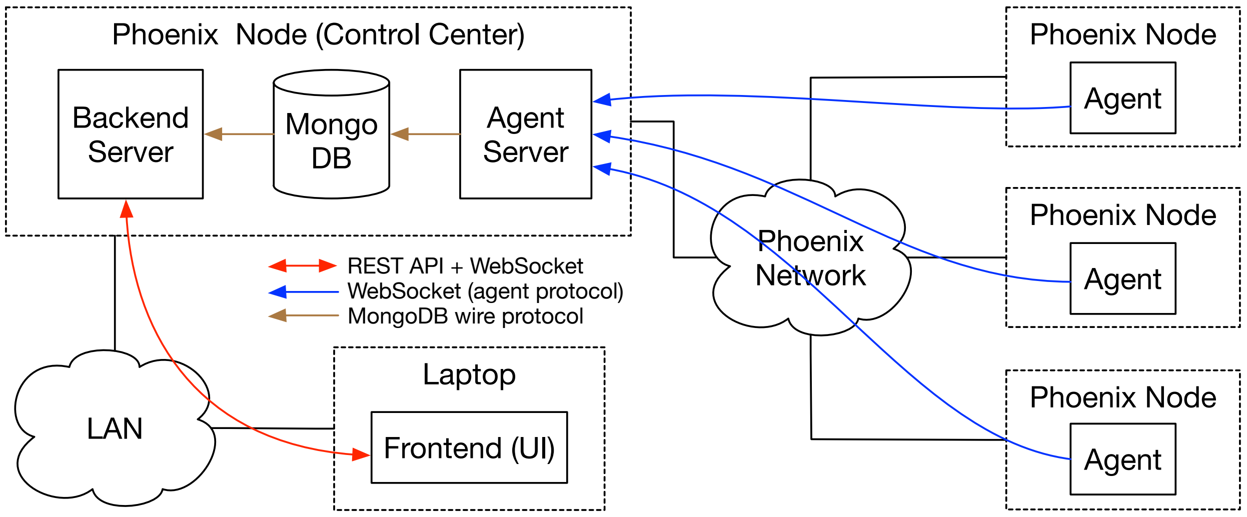


Figure 3.11: Architecture of the Netmon monitoring subsystem. PhoenixSEN nodes run agents that stream collected data to a backend server in the control center.

Figure 3.11 shows the architecture of Netmon. Every PhoenixSEN node runs a Netmon agent process that collects information about the node's state using OS-level instrumentation and any devices reachable via substation LANs using active network scanning. The collected data includes network interface statistics, the state of the node's OLSR links, and a list of discovered LAN devices. To discover LAN devices, the agent generates periodic gratuitous ARP requests on all LAN network interfaces. Once a device has been discovered, the agent probes the state of selected UDP and TCP ports on the device.

The collected data is streamed in near real-time over a persistent WebSocket [71] connection to a Netmon server in the PhoenixSEN CC. The communication occurs over an isolated management PhoenixSEN virtual network and is thus secure against passive and active attacks. When an agent gets disconnected from the server, it temporarily stores all collected data in a local database. The data will be uploaded to the server once the agent has reconnected.

The Netmon server correlates and persistently stores the data collected from all agents across PhoenixSEN. The data is indexed to allow time-based addressing and aggregation, e.g., to retrieve the state of the network at a particular time. This feature enables debugging or post-mortem analysis after an exercise when the physical network infrastructure is no longer operational. The server updates any connected UI clients as new data becomes available. This allows the UI to show the most current state of PhoenixSEN.

3.5 Experimental Evaluation

We experimentally evaluated the PhoenixSEN prototype on realistic physical grid infrastructure through a series of DARPA-led exercises on Plum Island, NY. Due to the difficulty of experimenting on real systems, the research community generally evaluates prototypes on artificial testbeds [72, 73, 74] that use co-simulation, i.e., they model the electrical and networking subsystems separately. The evaluation of PhoenixSEN on realistic physical grid infrastructure and the involvement of actual power distribution personnel during the exercises represent this project's unique and novel aspects.

3.5.1 DARPA RADICS Exercises

The DARPA RADICS program organized seven evaluation exercises from 2016 to 2020 to assess the effectiveness of black start grid restoration technology during a hypothetical large-scale grid cyber attack. The exercises occurred in a realistic operational environment with isolated physical grid infrastructure [10]. The following sections discuss the largest penultimate exercise (6), conducted over nine days in the fall of 2019.

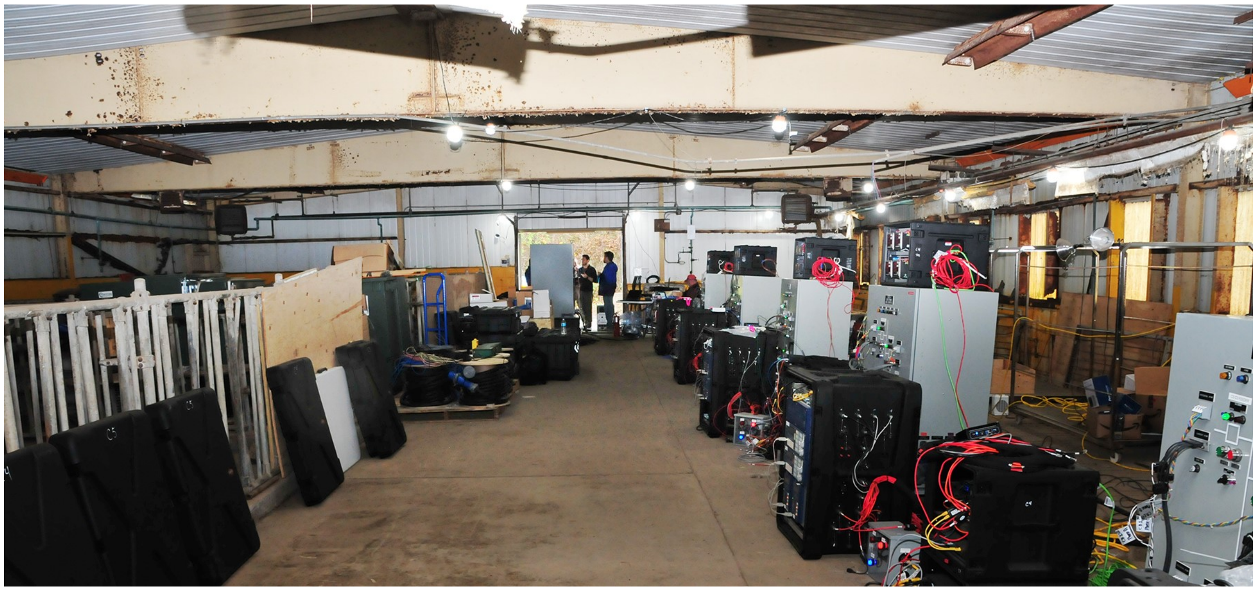


Figure 3.12: PhoenixSEN prototype (black boxes) being evaluated on an electrical grid testbed (gray boxes) in a DARPA RADICS exercise on Plum Island, NY, October 2019. Photo author: DARPA. Photo source: https://www.darpa.mil/news-events/2021-02-23. Used under the DARPA User Agreement terms.

3.5.2 Testbed Architecture

Figure 3.13 shows the configuration of physical infrastructure during exercise 6. Figure 3.14 shows the network's topology as seen by Netmon during exercise 6.

The testbed consisted of 21 sites spread across five geographic zones. The simulated RTO was located at Orient Point, NY (Long Island, NY). All remaining sites were located on Plum Island, NY, representing substations and control centers of three independent utilities: A, B, and C. The sites were connected with radio, Ethernet, and fiber links. Some links were temporarily provided by the U.S. National Guard. Others represented existing utility-owned connections. The following table summarizes the scale of the networking infrastructure during the largest exercise.

Each site or substation was equipped with realistic power grid infrastructure ("substation in a box") that included an RTU, real-time automation controller (RTAC), networking and SCADA switches, protective relays, GPS, and various sensors. All infrastructure was designed around commonly deployed U.S. systems to replicate real-world conditions. The substations were connected via power lines (not pictured in Figure 3.13) to form a multi-utility electrical grid.

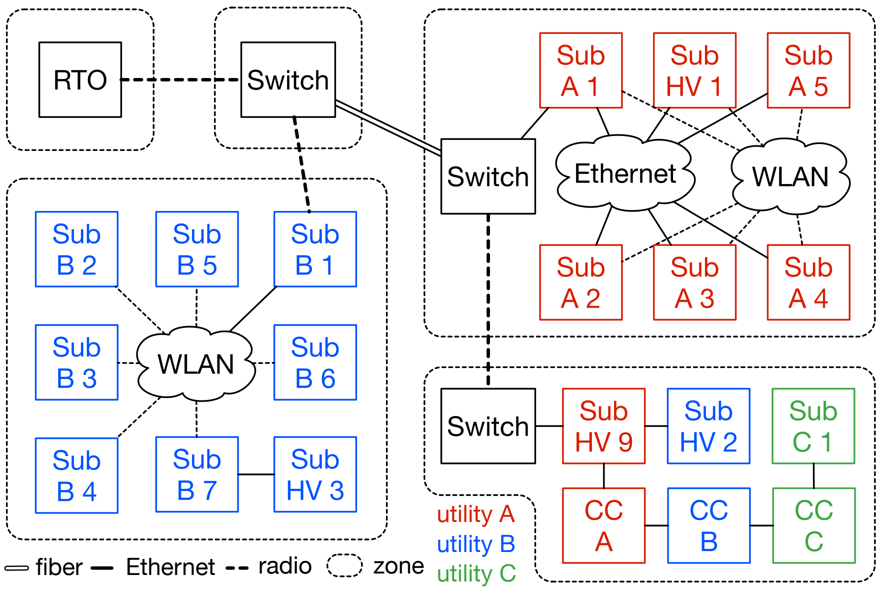
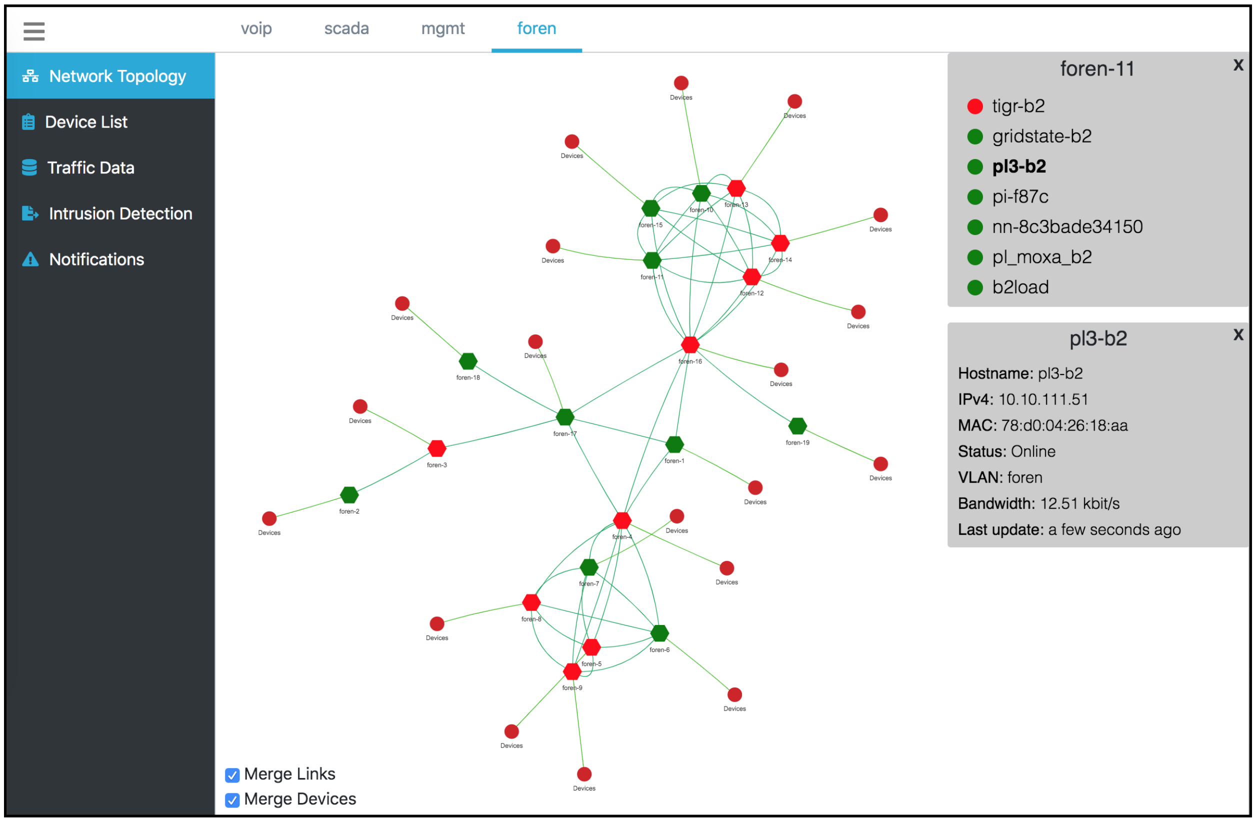


Figure 3.13: Physical configuration of the electric grid networking infrastructure during DARPA RADICS exercise 6. The infrastructure is assigned to three separate utilities: A, B, and C. Utilities are color-coded.

3.5.3 Reliability Evaluation

The DARPA exercise team repeatedly disrupted the whole system or its parts to simulate blackouts, device malfunctions, or loss of connectivity between sites. Exercise participants were organized into teams: RTO, utility personnel, connectivity, forensics, and security. Their goal was to cooperatively restore the system to an operational state as quickly as possible using only the technology and infrastructure available on Plum Island. Communication with the outside world was not permitted.

To see whether the OLSR overlay correctly spanned all substations, we analyzed the data collected by Netmon at all nodes during the exercise. Figure 3.15 shows the OLSR mesh topology as a graph. The graph was identical for all VLANs. The data shows the overlay network correctly spanning all sites. The diameter of the overlay network was 6, which is optimal for this physical configuration. PhoenixSEN correctly handled a variety of link types, including point-to-multipoint wireless, point-to-point radio and fiber links, Ethernet, and utility fiber connections. As long as all links were operational, PhoenixSEN converged to an optimal topology without network partitioning.



foren-11

- 🔴 tigr-b2
- 🟢 gridstate-b2
- 🟢 **pl3-b2**
- 🟢 pi-f87c
- 🟢 nn-8c3bade34150
- 🟢 pl_moxa_b2
- 🟢 b2load

pl3-b2

Hostname: pl3-b2
IPv4: 10.10.111.51
MAC: 78:d0:04:26:18:aa
Status: Online
VLAN: foren
Bandwidth: 12.51 kbit/s
Last update: a few seconds ago

Figure 3.14: Netmon showing the topology of the PhoenixSEN network during DARPA RADICS exercise 6. This view shows the VLAN dedicated to forensic activities. Elements that might require attention are highlighted in red. Hexagonal nodes represent PhoenixSEN nodes. Round nodes represent substation devices connected to the network.

Table 3.1: Network parameters during DARPA RADICS exercise 6

Grid parameters	Utility A	Utility B	Utility C
Control centers (CC)	1	1	1
Substations	5	7	1
High voltage (HV) substations	2	2	0
RTOs	1 (shared across utilities)		
Maximum RTO-substation distance	5 links		
PhoenixSEN parameters	**SCADA**	**VoIP**	**Forensic**
Connected devices	33	69	265
PhoenixSEN nodes	21 (shared across VLANs)		
Physical links per node	min: 1, max: 3		
OLSR Routes per node	min: 1, average: 5, max: 9		
Inter-utility relay nodes	3 (HV 9, CC A, CC B)		
OLSR network diameter	6 (all VLANs)		

The convergence time in OLSR is a function of the intervals of "hello" and "traffic control" messages periodically transmitted by nodes. In PhoenixSEN, we used 2s and 5s intervals, respectively. The exercise network had a network diameter of 6, giving a restart convergence time of 30 seconds or less when all nodes and links are operational. Thus, for networks of similar size, the recovery time will be dominated by factors other than OLSR convergence time.

As seen in Figure 3.13, the utility B control center and the entire utility C were connected to the rest of the system via the utility A control center. This configuration provided an opportunity to test inter-utility traffic relaying where a PhoenixSEN node at one utility relays traffic for other utilities in the area. Based on the Netmon data, this feature worked correctly during the entire exercise. Utility A substations were connected via redundant links (Figure 3.13). PhoenixSEN correctly handled that situation as well. The Ethernet connection received all traffic between the substations when it was operational. When the Ethernet connection disconnected, OLSR transparently re-routed all traffic to the backup wireless network.

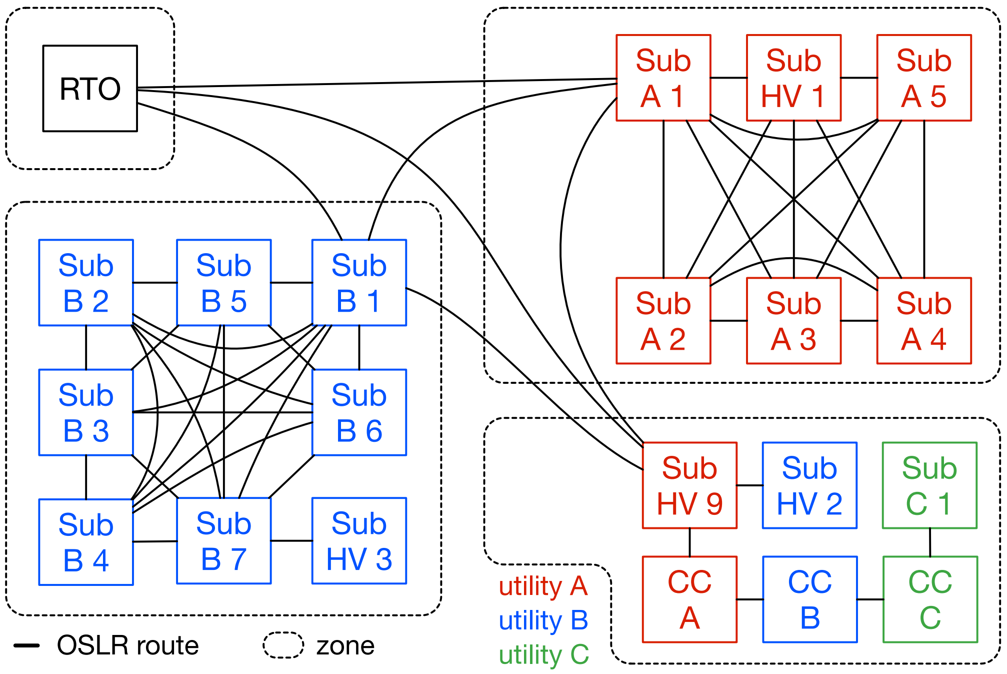


Figure 3.15: OLSR overlay network (routes) shown as a graph over physical network configuration. The overlay was identical across all three VLANs.

3.5.4 Scalability Evaluation

During DARPA RADICS exercises, PhoenixSEN provided connectivity within utility substations, between utilities, and to the RTO. A PhoenixSEN node was installed at each substation, connecting the substation infrastructure to the rest of the environment via external links. PhoenixSEN carried three kinds of traffic in isolated VLANs: VoIP, SCADA, and forensic. Other exercise participants generated forensic traffic to analyze and mitigate simulated problems in the power grid infrastructure.

Figure 3.16 shows daily traffic volumes flowing through PhoenixSEN as observed by the built-in monitoring system. A partially operational testbed caused the initial ramp-up period during the first days of the exercise. Forensic traffic generated by a large number of exercise participants dominates VoIP and SCADA traffic. Utilities planning backup networking infrastructure may need to consider forensic overhead.

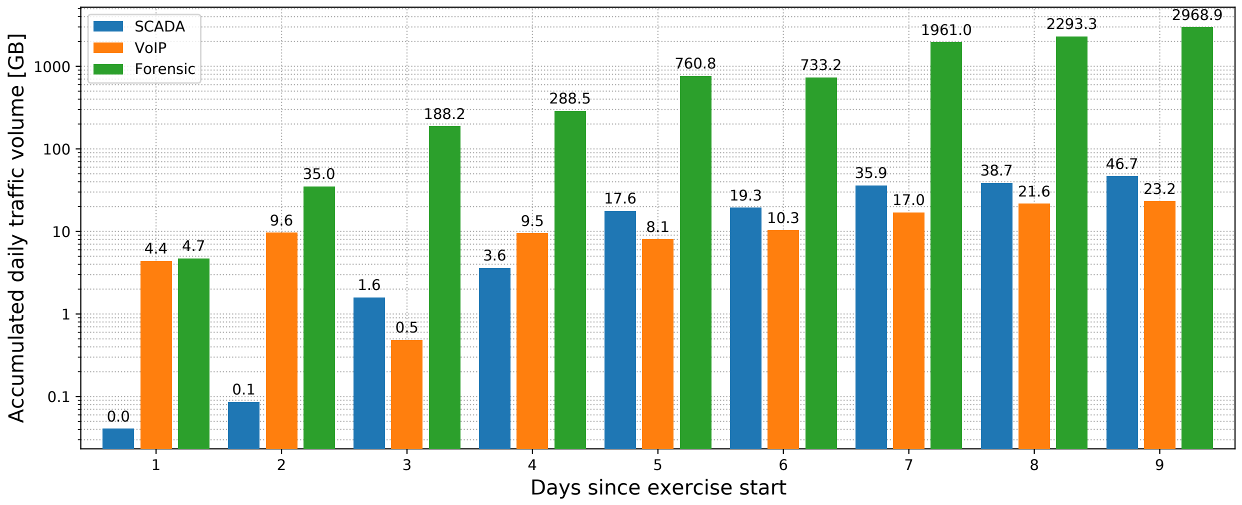


Figure 3.16: Daily traffic volumes in the three VLANs supported by PhoenixSEN during DARPA RADICS exercise 6.

3.6 Related Work

Modern critical industrial control systems, including the electrical grid, require communication. Such systems are increasingly targets of cyber attacks [75, 13]. The dangers cyber attacks represent for infrastructure controlling physical processes are well documented [34, 16]. Traditional IT system protective measures have been found inadequate for electrical grid infrastructure [15]. Redundancy and reliability of physical communication infrastructure for rapid black start power delivery restoration were found to be highly important [53].

The difficulty of experimenting on real systems prompted the research community to create a many testbeds [72, 73, 74, 76]. Many testbeds researched for this work appear to use co-simulation, modeling the electrical and networking subsystems separately. Most testbeds are software-based, and some use real hardware in a limited configuration. Networking subsystems beyond the substation SCADA are rarely emulated. Existing grid research mainly focused on securing and restoring physical grid subsystems. The RADICS project focused on restoring both the physical and networking grid subsystems.

Network architectures for emergency scenarios have been the subject of active research for decades. Many promising architectures based on ad hoc, peer-to-peer, mesh, mobile, delay-tolerant, and opportunistic networking have been proposed [77, 78]. Many architectures have been designed to be deployed in isolation without interfacing with legacy systems or link technology provided by third parties. Thus, they usually lack the necessary flexibility or services to support heterogeneous cyber-physical grid systems.

Observability was found to be an essential ingredient of rapid black start recovery. Networking infrastructure achieves observability (topology discovery, bandwidth estimation, intrusion detection) through monitoring. Existing network monitoring and measurement tools [38] can be classified into passive, active, and hybrid [79, 80]. We found that none of the freely available solutions provide sufficient flexibility to discover and monitor substation SCADA networks. Our solution, Netmon, is an integrated hybrid network monitor inspired by Moloch [81], MRTG [82], and OpenNMS [83]. Netmon integrates with OLSR, DHCP, and intrusion detection subsystems to improve SCADA observability.

Our work complements existing grid research focusing on securing and restoring physical grid subsystems. We focus primarily on the networking subsystems within and between substations and utilities. PhoenixSEN is designed to re-establish connectivity, and thus observability and coordination, during black start. The system supports traffic isolation and forensic activities to combat ongoing network-based cyber attacks. To the best of our knowledge, our work is the first attempt to create and evaluate rapidly deployable backup network infrastructure for the electrical grid.

3.7 Summary

This chapter presented the design, implementation, and experimental evaluation of PhoenixSEN, an ad hoc network for real-time coordination of people and SCADA devices. PhoenixSEN has been designed for the needs of the power distribution industry during black start. The network is flexible and can be used as a temporary replacement for third-party ISP networks. PhoenixSEN is designed to speed up electric grid restoration amidst a persistent or ongoing network-based cyber attack.

The grid networking subsystem faces many unique challenges due to the system heterogeneity caused by long-term evolution, security challenges resulting from the remote nature of some components, a large geographic service area, and a multi-stakeholder operational model. Selected autonomic design principles have been used in the design of PhoenixSEN. Similar principles also apply to the primary network subsystem of the U.S. electrical grid.

We tested and evaluated a PhoenixSEN prototype in a series of DARPA-led cyber security exercises on isolated realistic electrical grid infrastructure on Plum Island, NY [84]. During the exercises, PhoenixSEN provided connectivity for utility substations, control centers, the RTO, and also supported forensic activities that were part of the recovery process. At the peak of exercise activities, the network consisted of 21 PhoenixSEN nodes and connected over 350 devices.

The network performed well and correctly handled links with different kinds of technology, including radio point-to-point links, wireless point-to-multipoint connections, and utility fiber. The combined OLSR-VxLAN mesh network correctly handled simulated outages and disconnects and quickly converged to an optimal topology in all cases. Uniform hardware and software architecture proved to be very useful during deployment.

Although the design of PhoenixSEN was primarily motivated by the specific needs of the power distribution industry, we believe the resulting architecture is more general and flexible and might be suitable for other emergency scenarios as well, e.g., to restore connectivity in the aftermath of a hurricane [22]. Similar ad-hoc network technology based on off-the-shelf networking components is used by disaster relief organizations such as the Red Cross [85].

3.7.1 Future Work

One of our original goals was to design a self-configuring backup network for the electric grid that could be operated by utility personnel. We believe that the goal was partially reached. The current prototype requires deployment planning and minimal one-time configuration consisting mainly of the configuration profiles for the node's links. Both steps assume a background in computer networking. Simplifying the deployment process further is one of our future research goals.

As part of future work, PhoenixSEN could be extended with features and services for first responders, disaster recovery, and other emergency scenarios where communication and coordination are critical, even if the primary communication networks are not operational. We envision supporting a variety of typical emergency and disaster scenarios on top of general-purpose hardware and software architecture with interchangeable service profiles. Each profile would activate services designed to meet the needs of a particular emergency scenario, e.g., incident management, ticketing, or collaborative document editing.

3.7.2 Acknowledgments

We thank Clifton Lin, Charles Tao, Defu Li, Ranga Reddy, and James Dolan, all members of the BAE Systems team, for fruitful discussions and help in developing and testing the prototype. We are grateful to Frafos GmbH for a free license to use the ABC SBC platform and for help with configuration.

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views and conclusions contained in this section are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. government. Distribution statement A. Distribution approved for public release, distribution unlimited. Not export controlled per ES-FL-020821-0013.

Chapter 4: Geographic Resource Discovery in Federated Cyber-Physical Systems

This chapter describes SenSQL, a framework for geographic resource discovery in federated cyber-physical systems, i.e., systems that are geographically dispersed and where different parts of the system are managed by independent entities, such as the U.S. electrical grid. The framework combines a resource discovery protocol inspired by LoST [86] with a SQL-based query interface.

SenSQL aims to enable self-regulating and administratively independent cyber-physical systems, called autonomous cyber-physical systems (ACPS), that collectively form a federated system without a single administrative or technological point of failure. The SenSQL framework balances control over ACPS sensors and data with service federation, limiting the reliance of applications on centralized infrastructure.

4.1 Introduction

A key component of all CPS and IoT systems, both large and small, is the data acquisition infrastructure in the form of geographically distributed sensors [26]. Sensors perform measurements of the physical environment or engineered objects and produce data representing measured quantities. The data then informs computation, analysis, or control functions. Such systems' computational and physical components are tightly interconnected and coordinated to work effectively together [87].

Simple computational or control functions may only require a single measurement, for example, the most recent sensor measurements. More complex functions, however, often require a history of measurements from a particular sensor, e.g., to calculate aggregates, estimate trends, or train machine learning algorithms. The typical sensor device does not maintain a history of measurements on its own. This task is generally delegated to an external database management system (DBMS) that stores measurements from a collection of sensors.

Many DBMS architectures have been proposed in the literature. The DBMS can be either general-purpose or specialized for efficient storage and querying of sensor data in CPS systems. Specialized CPS databases often model sensor data as a time series of measurements and provide specialized functions for efficient storage and handling of such data. A particular limitation of most existing general-purpose DBMSs is that they assume that a common administrative entity tightly controls all storage nodes in a distributed DBMS.

We propose a distributed SQL-based storage and query processor architecture called SenSQL for federated autonomous cyber-physical systems. Our architecture provides the application programmer with a familiar declarative SQL query interface to spatiotemporal sensor data. Unlike other solutions, we assume the data is geographically partitioned and distributed across cyber-physical systems administered by multiple independent entities. System federation enables larger and more heterogeneous CPS systems where self-regulating and independent components can form a collaborative system.

The electrical grid scenario discussed in Chapter 3 involving geographically distributed substations and independent utilities sharing transmission infrastructure is a good example of a potential target system for this work on a larger scale.

The SenSQL service is designed to provide a familiar SQL interface to the application. SQL was chosen for pragmatic reasons. It is a standards-based, high-level declarative query language familiar to application developers. It is also widely supported by existing tools, e.g., Grafana (shown in Figure 4.3), Node-RED [88], Periscope Data [89], or Python pandas [90].

The rest of the chapter is organized as follows. Section 4.2 briefly overviews related protocols and technologies. We motivate and formulate the problem in Section 4.3. Section 4.4 provides a high-level overview of the SenSQL framework. In Section 4.5 we define the autonomous cyber-physical system notion. Section 4.6 discusses the naming architecture used in SenSQL. In Section 4.7 we describe the design and implementation of the geographic naming database. Section 4.8 describes the design of the SenSQL query coordinator component. Section 4.9 describes the evolved LoST service. Section 4.10 provides an overview of the implemented prototype and Section 4.11 discusses the design of the evaluation environment used to evaluate the prototype. Finally, we review related work in Section 4.12 and conclude in Section 4.13.

4.1.1 Challenges and Novelty

A key challenge in the design of the SenSQL service is locating all autonomous cyber-physical systems (ACPSs) related to a particular geographic area. Various forms of geographic routing, addressing, and discovery have been explored by the networking community [91, 92, 93, 94]. In our design, we have adopted an architectural and operational model based on the LoST protocol framework [95, 86]. Using LoST to locate storage nodes in a geographically dispersed federated DBMS is a novel aspect of the SenSQL service.

To alleviate the need for the LoST service to process complex spatial objects, the query coordinator runs a polygon simplification algorithm, similar to [96], on the geographic objects submitted to the service. This optimization shifts the burden of complex geographic object processing from the LoST service to the query processor and the application submitting the SQL query. This optimization and DE-9IM polygon matching implemented in the LoST server represent novel extensions of the LoST protocol framework.

4.2 Background

4.2.1 Database Management Systems

The need to separate an application's code from the data it operates on gave rise to general-purpose DBMSs. The separation allows application developers to focus on the access and manipulation of data. The details of how these operations are performed are left to the DBMS [97]. The database community has produced a wide variety of DBMS designs, both practical and academic. Existing DBMSs can be characterized by the data model (relational, document-oriented, graph), the type of supported workload (transaction or analytical), system architecture (single-node, distributed, cluster, federated), and elasticity (vertically or horizontally scalable, sharded).

First DBMSs based on the relational model of data [98] were developed in the 1970s. Around the same time, IBM developed a domain-specific language called SEQUEL (Structured English Query Language), designed to retrieve and manipulate data in IBM's System R database. The SEQUEL language was a direct predecessor of the Structured Query Language (SQL). In the 1980s, SQL was standardized by American National Standards Institute (ANSI) and ISO and became the dominant domain-specific language for accessing and manipulating data stored in relational DBMSs.

Despite standardization, significant variability exists between SQL dialects implemented and understood by various DBMSs. Furthermore, many extensions to standard SQL exist that offer additional functionality, e.g., various procedural languages, processing of data streams, geographic and spatial data types and operations, multi-media extensions, and much more. The variability makes developing applications portable across DBMSs challenging. Nevertheless, SQL remains the dominant application-layer interface to relational data. Some subset of standard SQL is supported by virtually all modern programming languages and frameworks.

SQL and the relational data model have been repeatedly criticized as complex to use or inadequate for a particular application programming style. The object-oriented programming style was dominant in the 1990s, and the criticism gave rise to object-oriented DBMSs designed to better support applications written in the object-oriented programming style. These DBMSs eschewed SQL in favor of more straightforward (and less powerful) APIs better suited for object-oriented applications. Object-oriented DBMSs did not see widespread adoption; however, they served as an essential early inspiration for document-oriented DBMSs.

The growing popularity of the World Wide Web (WWW) and internet applications in the 2000s resulted in a renewed interest in DBMS architectures. Internet applications can be characterized by the need to support many concurrent users, high availability (always online) requirements, and relatively simple database operations. Such requirements were difficult to meet for that era's general-purpose relational DBMSs.

The traditional method of vertically scaling up the relational DBMS using more powerful hardware did not achieve the desired results. Attempts at horizontal scalability (sharding), i.e., spreading the data over a cluster of nodes joined into one logical DBMS by custom middleware at the application level, were more successful. Notable examples of this approach were Google's and Facebook's sharded MySQL databases. Application-level sharding works well for simple operations. More complex operations, for example, transactions updating multiple records spread across different physical machines within the cluster, are challenging to support in such an architecture.

These challenges led to document-oriented DBMSs, also known as NoSQL or non-relational DBMSs. Examples include MongoDB [99], Amazon Dynamo [100], Google BigTable [101], and Facebook Cassandra [102]. NoSQL DBMSs trade consistency and correctness for performance and alternative data models [103]. Many NoSQL DBMSs implement an eventual consistency data model, which is often sufficient for internet applications. NoSQL databases developed in the 2000s provided a simpler API to the application without the full expressiveness of SQL. Such databases often served as simpler data stores, leaving much data processing logic to the application. It was believed these tradeoffs would allow developers to focus on the application rather than on how to scale the DBMS.

The ubiquity of Internet infrastructure prompted many of the original enterprise applications to move to the Internet, thus becoming Internet applications. Enterprise applications often have strong consistency and transactional requirements not easily fulfilled by NoSQL DBMSs. This problem brought about a new class of DBMSs called NewSQL, also known as distributed SQL. NewSQL databases are modern relational DBMSs with the scalability of NoSQL DBMSs while still maintaining ACID guarantees for transactions [97]. NewSQL DBMSs promises to enable internet-scale applications that can execute a large number of concurrent read-write transactions using SQL. Some of the NewSQL DBMSs were built from the ground up and used novel architectures; others evolved the sharding middleware approach popular in earlier DBMSs. Examples of NewSQL DBMSs include Google Spanner [104] and CockroachDB [105].

DBMSs can also be characterized by the type of workload or access patterns they are optimized for. Online transaction processing (OLTP) DBMSs are typically optimized for many concurrent short-lived predefined queries that access a relatively small portion of the data within the query. An OLTP database relies heavily on indexes when executing such queries to achieve high throughput and limited query processing time. Most SQL-based DBMSs are optimized for OLTP workloads.

In contrast, online analytical processing (OLAP) DBMSs are optimized for long-lasting complex queries typically used for analytical, reporting, and forecasting purposes. Such queries are often ad hoc and executed interactively. OLAP DBMSs typically use a multi-dimensional data model. The de-facto standard query language for OLAP systems is the Multidimensional Expressions (MDX) language [106] developed by Microsoft.

4.2.2 Spatial SQL

SQL/MM Spatial [107] is an extension to the SQL language that defines how to store, retrieve, and process geographic and spatial data using SQL. The extension has been standardized in ISO/IEC 13249-3 and is generally supported among major existing relational DBMSs. The extension defines interfaces, data types, geometries, standard geographic and spatial reference systems, and functions to convert, compare, and process spatial data.

4.2.2.1 Geometric Data Types

The spatial data model in SQL/MM Spatial was derived from the Simple Feature Access (SFA) standard [108] developed by the Open Geospatial Consortium (OGC). Figure 4.1 shows an abstract class hierarchy of the spatial data types defined in the SFA standard. The hierarchy inspired the spatial data type hierarchy in SQL/MM Spatial.

All spatial types derive from an abstract `Geometry` type. Geometry values are associated with a spatial reference system identified by a spatial reference system ID (SRID). A primary spatial data type is defined for each elementary geometric type: `Point`, `LineString`, and `Polygon`. The size and location of a geometry value are defined by its two-dimensional or three-dimensional coordinates relative to the associated spatial reference. More complex geometric values can be constructed from the basic types. The standard defines the following collection geometric types

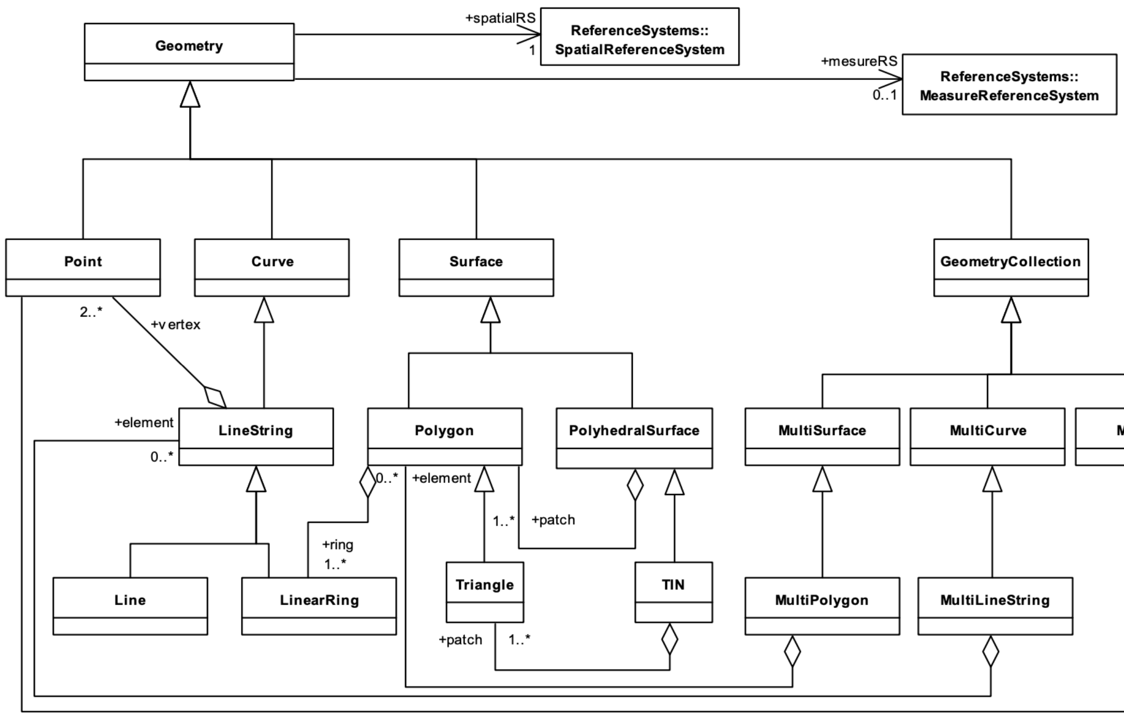


Figure 4.1: An abstract hierarchy of spatial data types as defined in the OGC Simple Feature Access standard. Source: [108]. Used under the terms of fair use.

for building complex geometric values: `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`. Unlike the first three collection types, the `GeometryCollection` can contain arbitrary geometry types.

A spatial data type's *dimension* is 0 for point types, 1 for linear types, and 2 for polygonal types. The dimension of a collection type is the maximum element dimension. The *bounding box* of a geometric value is a two or three-dimensional box that encloses all coordinates of the value.

In SQL, geometric values are most commonly represented in the well-known text (WKT) or well-known binary (WKB) formats. Both formats include information about the value types and the coordinates that define them. For example, the WKT format of a `Point` at coordinates (1, 1) is "`POINT(1 1)`". The WKB format of the same point is:

```
0101000000000000000000F03F000000000000F03
```

The GeoJSON format [109] is an alternative JSON-based representation for geometric values designed to directly support the spatial data types of SQL/MM Spatial.

4.2.2.2 Geometric Relationships

A spatial relationship determines how two geometries interact with each other. SQL/MM Spatial allows the application to index and query geometric values by their relationship with other geometric values.

The relationship between two geometric values is determined by comparing the intersections of their *interiors*, *boundaries*, and *exteriors*. The interior, boundary, and exterior represent a subset of the geometric value's coordinates (points). The comparison can be represented with an intersection matrix known as the Dimensionally Extended 9-Intersection Model (DE-9IM). The matrix provides a means to classify geometric value relations.

Let $I(g)$, $B(g)$, $E(g)$ denote the interior, boundary, exterior of geometric value g. Let $dim(s)$ denote the dimension of a set s in the domain of $\{0, 1, 2, F\}$, where 0 represents a point, 1 represents a line, 2 represents an area, and F represents an empty set. The DE-9IM intersection matrix for geometries a and b is illustrated in Table 4.1. Reading from left to right and from top to bottom, the DE-9IM intersection matrix can be represented with a text string of the form "`212101212`".

Table 4.1: DE-9IM intersection matrix

	Interior	**Boundary**	**Exterior**
Interior	$dim(I(a) \cap I(b))$	$dim(I(a) \cap B(b))$	$dim(I(a) \cap E(b))$
Boundary	$dim(B(a) \cap I(b))$	$dim(B(a) \cap B(b))$	$dim(B(a) \cap E(b))$
Exterior	$dim(E(a) \cap I(b))$	$dim(E(a) \cap B(b))$	$dim(E(a) \cap E(b))$

To test the relationship between two geometric values, the DBMS computes the intersection matrix, translates the matrix to a string, and then compares the string with pattern matching based on the relationship requested by the application. The OGC SFA standard defines predicated for common relationships. The SQL/MM Spatial standard provides the predicates as named functions, for example, `ST_Contains`, `ST_Crosses`, `ST_Disjoint`, `ST_Equals`, `ST_Intersects`, `ST_Overlaps`, `ST_Touches`, `ST_Within`. The predicate functions can be used in the WHERE clause of SQL queries as follows:

```sql
SELECT city.name, state.name, city.geom
FROM city JOIN state ON ST_Intersects(city.geom, state.geom);
```

4.2.3 LoST: A Location-to-Service Translation Framework

Many internet applications and protocols need to dynamically locate the entities providing a particular service. The Domain Name System (DNS) [110] is the most commonly used protocol for this purpose. DNS can map hierarchical service identifiers to service providers using a combination of resource records [111, 112, 113]. Thus, DNS provides a means to discover service implementations by the service's identity.

Some applications need to dynamically discover services by combining the service's identity and the requestor's location. The location information could be geographic coordinates or a civic address. Consider an emergency calling application where the caller dials 911 and expects to reach a nearby answering service that serves his or her current location.

Location-to-Service Translation Protocol (LoST) [86] is a protocol for mapping service identifiers and location in the form of geographic coordinates or a civic address to entities that provide the service for the client's location. Unlike DNS-based service discovery, LoST considers the client's location when selecting the appropriate mapping. The protocol was initially designed to determine the most appropriate emergency call answer point based on the caller's location. Incorporating geographic or civic location in the mapping process allows LoST to resolve services that are not necessarily closest to the client with respect to the network topology.

The LoST protocol uses HTTP to transport requests and responses and XML to encode data. Geographic and civic coordinates are encoded in a subset of the Presence Information Data Format Location Object (PIDF-LO) [114, 115, 116, 117]. The PIDF-LO format uses a subset of the Geography Markup Language (GML) [118] to convey geographic location. The civic address format was initially defined in [119]. The LoST protocol provides three primary functions: find service by location, obtain a service boundary, and list supported services at a server or location. The LoST protocol provides recursive and iterative resolving and support for caching.

The LoST specification only defines the syntax and semantics of requests and responses between a client and a LoST server. It does not specify how to design a scalable service supporting the LoST protocol. Two related RFCs [95, 120] propose a global, scalable, resilient, and administratively-distributed service architecture for the LoST protocol. In the architecture, LoST servers are organized in trees according to their coverage regions, where the coverage region of a server higher up in the tree covers the coverage regions of its children. A separate tree for each pair of service type and location profile is created. Designated servers known as forest guides keep track of all existing trees. The client then accesses such service through a resolver that keeps track and maintains trust with the forest guides.

The LoST protocol is extensible. Two extensions have been defined so far. RFC 6197 [121] specifies how a LoST server communicates the service boundary of a listed service to the client. RFC 6451 [122] defines additional mapping parameters to allow a LoST client to obtain N nearest services, services within a distance X, or list services that serve a particular location or area. This extension makes the LoST framework applicable in other application domains, e.g., discovering the nearest restaurants or restaurants that serve a particular area.

4.3 Motivation and Problem Statement

Consider PhoenixSEN, the electrical grid backup network discussed in Chapter 3. PhoenixSEN has been designed to restore connectivity in a large geographically dispersed CPS, where multiple administrative entities operate the shared infrastructure. Modern-day grid infrastructure includes a variety of sensors to monitor generators, transformers, protective devices, network infrastructure, and power transmission lines. Electrical grid sensors can produce large amounts of data (GB or TB). Phasor-measurement units (PMUs) generate up to 60 measurements per second. Upcoming continuous point-on-wave (CPOW) sensors generate thousands of measurements per second [123]. Figure 4.2 compares selected electrical grid sensors.

PMU and CPOW sensors are commonly deployed deep in the grid and dispersed across a large area. However, the applications that process sensor-generated measurements are usually centralized.

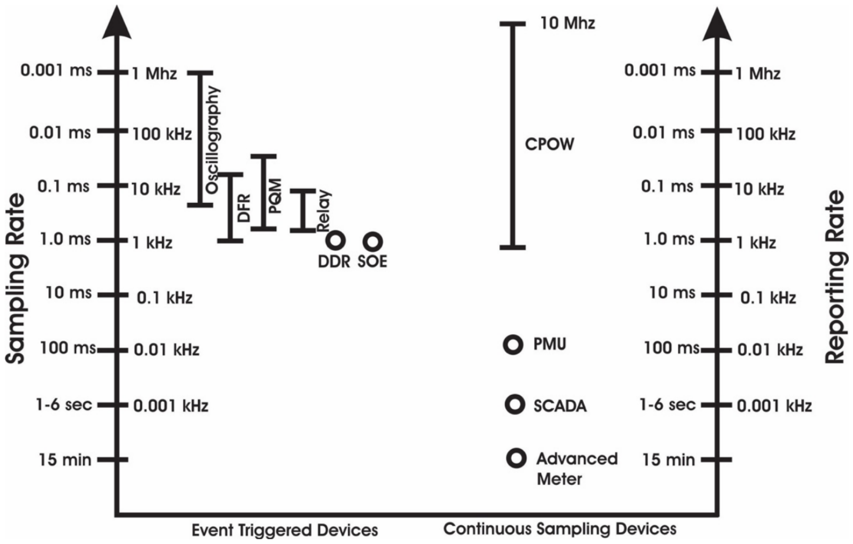


Figure 4.2: A comparison of electrical grid sensor measurement rates. Source: [123]. Used under the terms of fair use.

Measurements from many PMU or CPOW sensors are needed to build a global view of the electrical grid. Creating applications that manage such high volumes of data is challenging. The grid network subsystem lacks data storage and query services for PMU/CPOW measurements at the network edge. Such services would seem necessary for efficient data processing, e.g., anomaly-triggered push, query-selected pull, scalable data streaming, or data processing at the edge.

As a second example, consider the IoT infrastructure on a large university campus, e.g., Columbia University Morningside Heights campus. Organizationally, the University is divided into schools. Schools are divided into departments. Departments are assigned to floors, offices, and labs. Campus-wide organizational units such as Facilities or IT manage some infrastructure, but not all. Schools and departments have some degree of autonomy, e.g., to install and manage their own smart devices in labs and offices.

A typical large university campus may contain multiple IoT systems installed and operated by independent administrative entities. Some devices will be private, but not all. For example, temperature, humidity, or CO_2 sensors installed by departments or individuals could be helpful at the university level, e.g., for campus-wide environmental monitoring or anomaly detection. Nowadays, such devices must be associated with a single campus-wide system under a common administrative entity. Consequently, the device owner loses control over the device or its data. If the device communicates exclusively with a manufacturer cloud, determining who owns what or what data is stored where becomes even murkier.

The two examples highlight the need for a new approach to storing and discovering resources, i.e., sensors and data, in geographically dispersed or federated CPSs. A typical sensor device does not store the measurements and relies on an external DBMS. For small systems, for example, the infrastructure of an individual smart home, a single database node often suffices. Larger systems, such as the electrical grid or the campus example, need a decentralized architecture for storing and discovering sensors and their measurement data.

4.3.1 Requirements

The motivation examples and problem description from the previous section can be translated into a set of requirements the SenSQL framework must meet. We summarize the requirements in the following paragraphs.

Application Interface. Ideally, the service would be application-agnostic with a declarative query interface, allowing the application programmer to focus on what data to get rather than how or from where to get it. Furthermore, the service should support executing queries over the data at the network edge, i.e., where the data is stored. This feature helps to eliminate the need to stream large volumes of data from sensors to the application. We assume sending queries to the data, rather than the other way around, will be the dominant method of working with sensors that generate high volumes of data, such as some of the electrical grid sensor devices.

Federation. We assume the target CPS will be federated, i.e., with components operated by multiple administrative entities. Both examples mentioned in Section 4.3 illustrate federated scenarios. A federated architecture makes general-purpose distributed DBMS unsuitable. Such DBMSs generally assume that all storage nodes are tightly controlled and administered by a single administrative entity. That assumption does not hold in federated systems.

Resource Naming. We assume the target cyber-physical systems will be dynamic. Components (sensors or actuators) can come and go. One device can be replaced with another. For this reason, the application interface should provide a means to discover resources by names that could be reassigned if the underlying system changes. Ideally, the names would be formed by combining device or resource types with their relationship to a stable geographic feature.

Scalability. A scalable CPS would allow sensors to generate measurements frequently, store the data in a database near the sensor, and transfer the data only when required by the application. This approach could be further improved by data processing near the sensor, e.g., computing aggregates close to the source of the data.

Application Interoperability. The application interface to the service should be based on a technology widely supported by modern applications, frameworks, and programming libraries. The rationale behind this requirement originates from PhoenixSEN. Apart from restoring connectivity to the electrical grid, the network was also used by a forensic team to check SCADA devices for malicious activity. The forensic tools could only discover resources via well-established interfaces such as SQL.

4.4 High-Level System Overview

This section provides a high-level overview of the SenSQL architecture and its components.

4.4.1 System Architecture

We describe the SenSQL service using the high-level system architecture diagram shown in Figure 4.3. Suppose there are three independent, autonomous cyber-physical systems (ACPSs): A (blue), B (red), and C (green). A good example would be an electrical grid infrastructure in a geographic region divided among independent entities. The entities cooperate to operate the grid, but each entity has its physical infrastructure, e.g., generators, power lines, or substations. The internal structure of these ACPSs is not essential, but we assume each ACPS is willing to export some of its resources (sensors or measurement data) to trusted third parties via the SenSQL service.

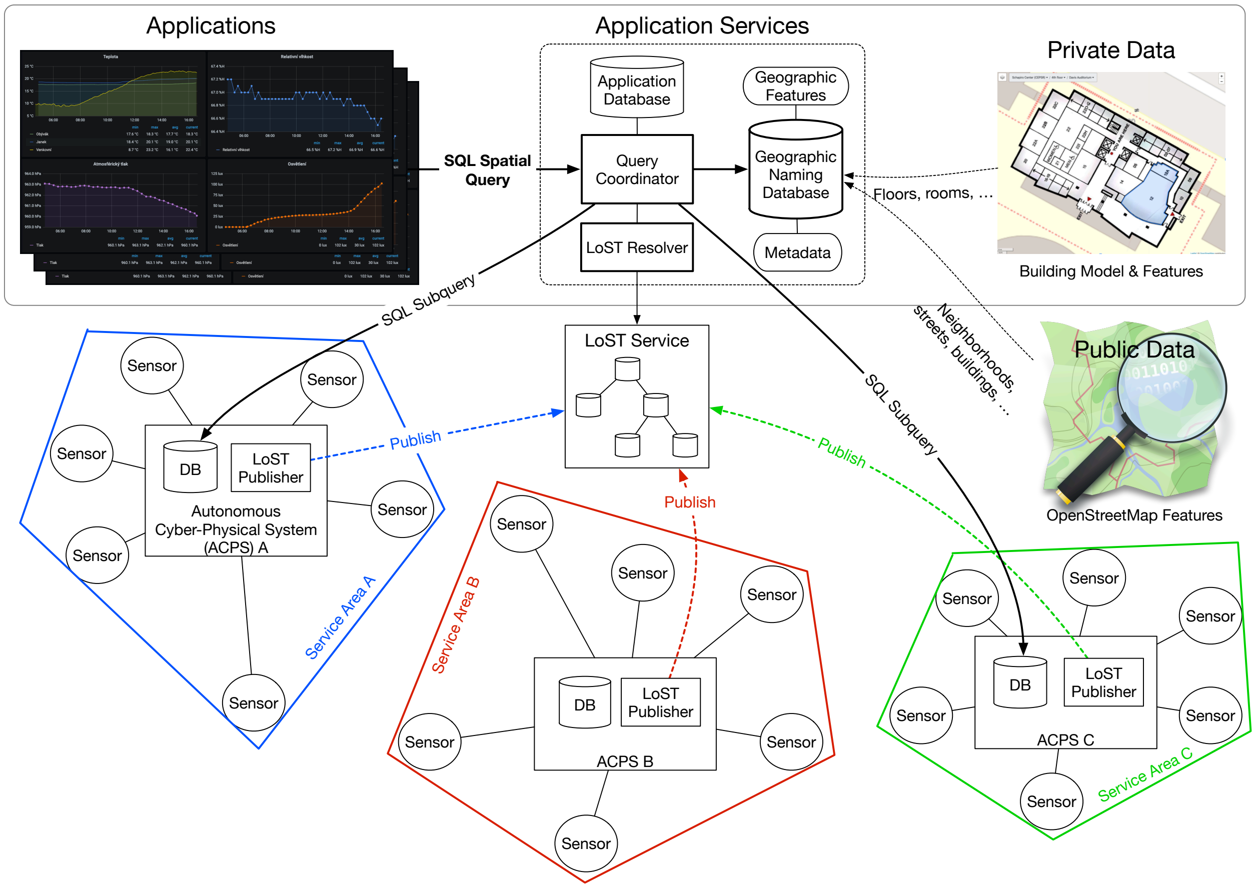


Figure 4.3: High-level system architecture of the SenSQL service. Applications use SenSQL application services to obtain a global view of a geographically dispersed federated system consisting of three autonomous cyber-physical systems.

Suppose another entity, e.g., a regional transmission operator (RTO) overseeing the operation of the grid in a geographic region, wishes to create an application that aggregates data from the sensors exported by individual ACPSs. The application could be a dashboard that summarizes data from geographically dispersed observation points to assess the overall health of the electrical grid. Recall that the systems providing the resources to the application are autonomous, i.e., administered independently with little or no coordination between the application and the system. In other words, the application needs to adapt to changes in the autonomous infrastructure, for example, as sensors are added, replaced, or removed.

Some sensor types can generate large volumes of data, as discussed in Section 4.3. Furthermore, since grid infrastructures tend to be dispersed, these sensors may be in remote areas or connected via low-bandwidth links. Thus, if the application needs to summarize data from many observation points, it should attempt to do so at the network edge where the data is produced or stored rather than by transferring high-volume data closer to the application.

The SenSQL service aims to solve both problems by providing the application with a standard SQL-based query interface with two features: a stable geographic resource naming and an ability to submit queries to the autonomous system where the resources reside.

Suppose each ACPS knows its exported sensors' approximate locations (WGS 84 coordinates). The ACPS determines its service area boundary in the form of a polygon that contains all its exported sensors. The boundary does not need to be minimal. It can be set administratively, for example, using a building boundary polygon, or computed automatically as the minimal polygon containing all exported sensors. The ACPS automatically updates the polygon to ensure that all exported sensors remain within the polygon across infrastructure updates.

The ACPS publishes its service area to an independently-administered Location-to-Service (Lost) mapping service [86]. The LoST service is a hierarchical scalable network service operated like the DNS. The ACPS also publishes a contact URI for its public SQL query interface alongside its service area. LoST resolvers use the LoST service to discover which ACPSs provide resources related to a geographic area of interest.

The application interacts with the ACPSs through SenSQL application services. The services include a SQL query coordinator, a LoST resolver for querying the LoST service, and a geographic naming database. The geographic naming database is local and unique to a particular instance of SenSQL application services. The database may contain a mix of public and private data. It contains a local copy of public geographic features obtained from, for example, OpenStreetMap [124]. It can also contain private geographic features derived from building information models, floor plans, or other sources.

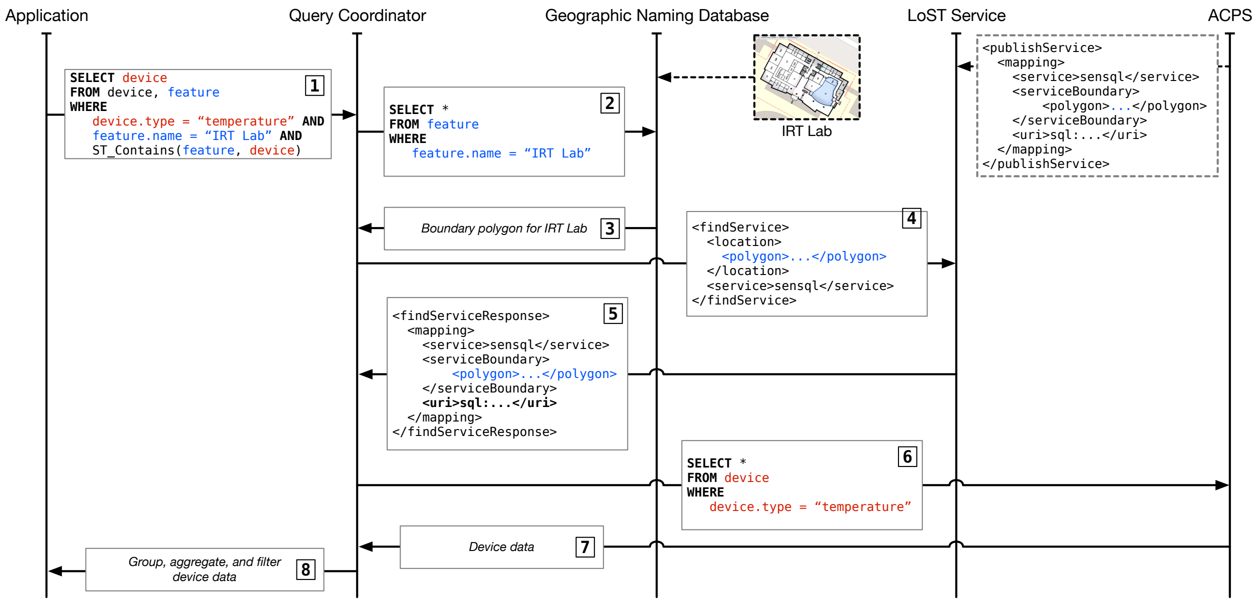
The application communicates with the services through a SQL interface provided by the query coordinator. The interface supports a subset of standard SQL [125] with SQL/MM Spatial extensions [107]. We chose standards-based SQL for compatibility with existing applications such as Grafana shown in Figure 4.3. Existing applications generally support SQL as a standard interface to the data. This design tradeoff makes it possible to use various existing applications on top of a dynamic, federated cyber-physical system managed by the SenSQL service.

The query coordinator provides the application with four virtual tables: *devices*, *measurements*, *features*, and *shapes*. The *devices* and *measurements* tables provide access to the data gathered from ACPSs. The application uses these tables to access the data collected by the SenSQL framework in response to SQL SELECT queries. The *features* and *shapes* tables give the application access to the geographic feature data from the geographic naming database. Other databases and tables may be mapped to an external application database if provided.

The application submits SQL SELECT queries to the query coordinator to query the federation of ACPSs for resources. The WHERE clause of those queries refers to the resources of interest by combining the resource type and its spatial relationship to selected geographic features. The query coordinator parses the WHERE clause for resource types and geographic features. It then uses the LoST service to resolve each geographic feature to downstream ACPSs intersecting it. The query coordinator constructs a SQL subquery without spatial matching for each downstream ACPS and forwards the subquery to its SQL query interface. The query coordinator aggregates all data returned by downstream ACPSs before the data is returned to the application.

4.4.2 Overview of Operation

Section 4.4 provides a high-level architectural overview of the SenSQL system. This section describes the high-level steps to process a simple SQL query. Individual steps are described in more detail in other sections of this chapter. Figure 4.4 shows component interactions and data flow while the SenSQL system processes a query. We assume the geographic naming database has been populated with data, and one or more ACPSs have published their service records to the LoST service before the SenSQL starts processing data. The two steps are shown in the upper right corner of the flow diagram.


Application
Query Coordinator
Geographic Naming Database
LoST Service
ACPS
IRT Lab
Boundary polygon for IRT Lab
Device data
Group, aggregate, and filter device data
1
2
3
4
5
6
7
8

```
SELECT device
FROM device, feature
WHERE
    device.type = “temperature” AND
    feature.name = “IRT Lab” AND
    ST_Contains(feature, device)
```

```
SELECT *
FROM feature
WHERE
    feature.name = “IRT Lab”
```

```
<publishService>
  <mapping>
    <service>sensql</service>
    <serviceBoundary>
        <polygon>...</polygon>
    </serviceBoundary>
    <uri>sql:...</uri>
  </mapping>
</publishService>
```

```
<findService>
  <location>
    <polygon>...</polygon>
  </location>
  <service>sensql</service>
</findService>
```

```
<findServiceResponse>
  <mapping>
    <service>sensql</service>
    <serviceBoundary>
        <polygon>...</polygon>
    </serviceBoundary>
    <uri>sql:...</uri>
  </mapping>
</findServiceResponse>
```

```
SELECT *
FROM device
WHERE
    device.type = “temperature”
```

Figure 4.4: A flow diagram showing component interaction while the SenSQL system processes a simple query.

Suppose an application wishes to discover all temperature sensors in the room named "IRT lab." The application creates a SQL SELECT query to search the tables *device* and *feature* and creates a WHERE clause that matches devices by their type and their relationship with the geographic feature named "IRT Lab." In this example, matching devices must be inside the "IRT Lab" boundary polygon. The resulting query is shown in box 1 in the flow diagram.

The query coordinator receives the query from the application, parses it, and extracts all references to the well-known *feature* table from the query's WHERE clause. The coordinator then submits another SQL query to the geographic naming database to fetch the data (boundary polygons) for all features referenced in the input query. Boxes 2 and 3 in the diagram illustrate these steps.

Having received the data for the geographic feature "IRT Lab," the query coordinator issues a *findService* request to the LoST service. The request includes the polygon obtained from the geographic naming database. It asks the LoST service for contact information of all ACPSs whose service areas (boundaries) intersect with the polygon included in the request. The LoST service returns the mappings for all such ACPSs. Each mapping contains a contact URI of the ACPS's public SQL query interface. These steps are shown in boxes 4 and 5 in the diagram.

Having received ACPS contact URIs from the LoST service, the query coordinator constructs a SQL subquery (shown in box 6) for every ACPS. The subquery is constructed from the input query, with all references to the geographic features removed from the WHERE clause. The query coordinator then sends the query to the ACPS (box 6). Any data received from the ACPSs is grouped, aggregated, and filtered by the query coordinator before it sends the data upstream to the application. This step is represented by boxes 7 and 8.

4.5 Autonomous Cyber-Physical System

An autonomous cyber-physical system (ACPS) is a cyber-physical system, a sensor network, or an Internet of Things (IoT) system under the management and operation of an independent administrative entity. The ACPS is a logical concept. We make very few assumptions (discussed below) about the system's underlying structure. Any collection of networked sensors or actuators and related computing and networking infrastructure could be viewed as an ACPS. Figure 4.5 shows a conceptual diagram of an ACPS.

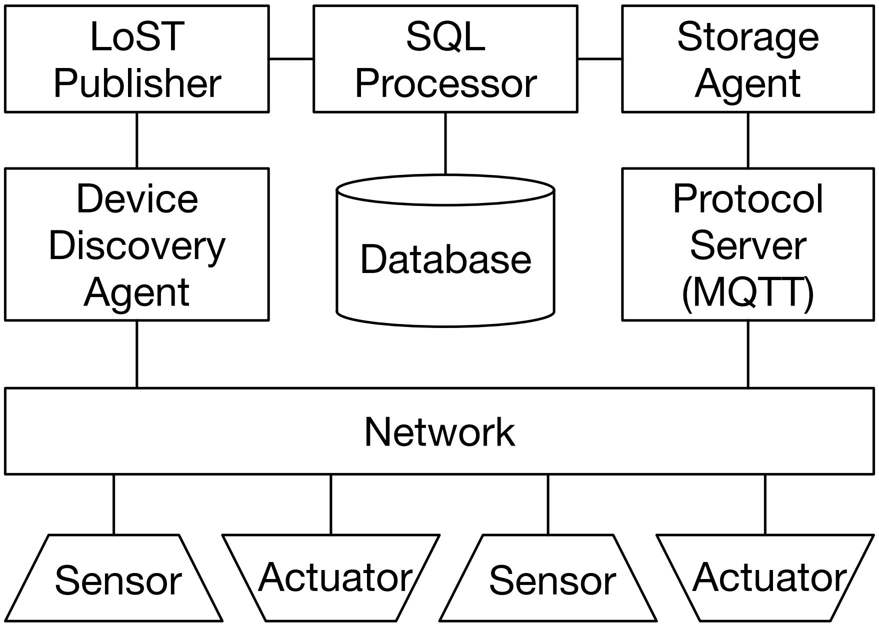


Figure 4.5: A conceptual diagram of an autonomous cyber-physical system (ACPS).

The internal structure of an ACPS is not significant to the SenSQL service. However, we make some assumptions to make it possible for the ACPS to become a part of a larger federated CPS cooperatively operated by multiple independent administrative entities.

We assume that each ACPS exports some of its resources, e.g., sensor devices or measurement data, to trusted third parties. Not all sensors or data have to be exported. For the ACPS to integrate into the SenSQL framework, it must provide a query interface based on a restricted subset of ANSI SQL [125]. Furthermore, the query interface must implement a data model the query coordinator component understands. We assume that all ACPSs participating in a SenSQL system provide the same data model for simplicity. Nevertheless, this is not strictly required as long as the query coordinator can understand and translate the data models of all involved ACPSs.

The key feature of an ACPS is that it operates under an independent administrative entity. The infrastructure of an intelligent home could become an ACPS. The IoT infrastructure on a large university campus could be divided into several independent ACPSs, e.g., according to buildings, schools, or departments. Even in small deployments, it may be advantageous to operate multiple related ACPSs to limit the scope of potential failures or to split the infrastructure into public and private components.

We assume the ACPS's exported sensors are administratively associated with a database that stores the sensors' measurements and provides a restricted public SQL query execution environment to query the data. The mechanism or protocol through which sensors push measurements to the database is out of this project's scope. This functionality could be implemented by a storage agent subscribing to measurement notifications over a protocol like MQTT.

While it might be advantageous to place the database close to the sensors, e.g., in the same LAN, it is not a requirement. The database could also be provided as a cloud-based service communicating with the sensors over a WAN. The former approach can be realized by placing the database on an IoT gateway. The latter accommodates sensors that maintain persistent connections to manufacturer-operated cloud infrastructure.

We also assume that the ACPS includes a device discovery process that can discover available sensors and actuators and their attributes, such as the type of each sensor and its approximate location or address. A subset of this information, i.e., the parameters of exported sensors, is made available to a LoST publisher component. The LoST publisher announces the ACPS via an independent LoST service shared by the federated SenSQL system.

4.5.1 Geographic Service Area

This section introduces the notion of an ACPS geographic service area. Each ACPS participating in the SenSQL framework announces its service area via a shared LoST service to allow third-party entities to discover the ACPS and its exported resources.

We assume the ACPS knows or can estimate the WGS 84 coordinates of its exported sensors. It maintains a *location estimate* for each device in the form of a $(latitude, longitude, uncertainty)$ triplet represented with a `geo` Uniform Resource Identifier (URI) [126]. The uncertainty component is expressed at a confidence of 95% or higher. Geometrically, a disk with the radius $uncertainty$ can represent the location estimate. A location estimate with large uncertainty (coarse location), as provided by, e.g., Wi-Fi or IP address positioning services, is sufficient for many applications. The process through which the ACPS obtains location estimates is out of this project's scope. In Figure 4.6, the location estimates of sensors are represented with blue discs.

The *geographic service area* of an ACPS is a geographic area that fully contains the location estimates of all exported sensors. The service area could be automatically computed by the ACPS or manually configured by an administrator. When the automatic method is used, the ACPS computes the boundary polygon of its service area as a convex envelope over the location estimates of all of its exported sensors. If the manual method is used, the service area can be determined using a building information model, a floor plan, or some other source of information.

The geographic service area for an ACPS does not necessarily have to be the smallest such area. If an ACPS serves an entire building but does not have devices in all building parts, setting the service area to the building's boundary polygon better represents the true extent of the ACPS. Smaller service areas and simpler polygons are advantageous because they help the LoST service

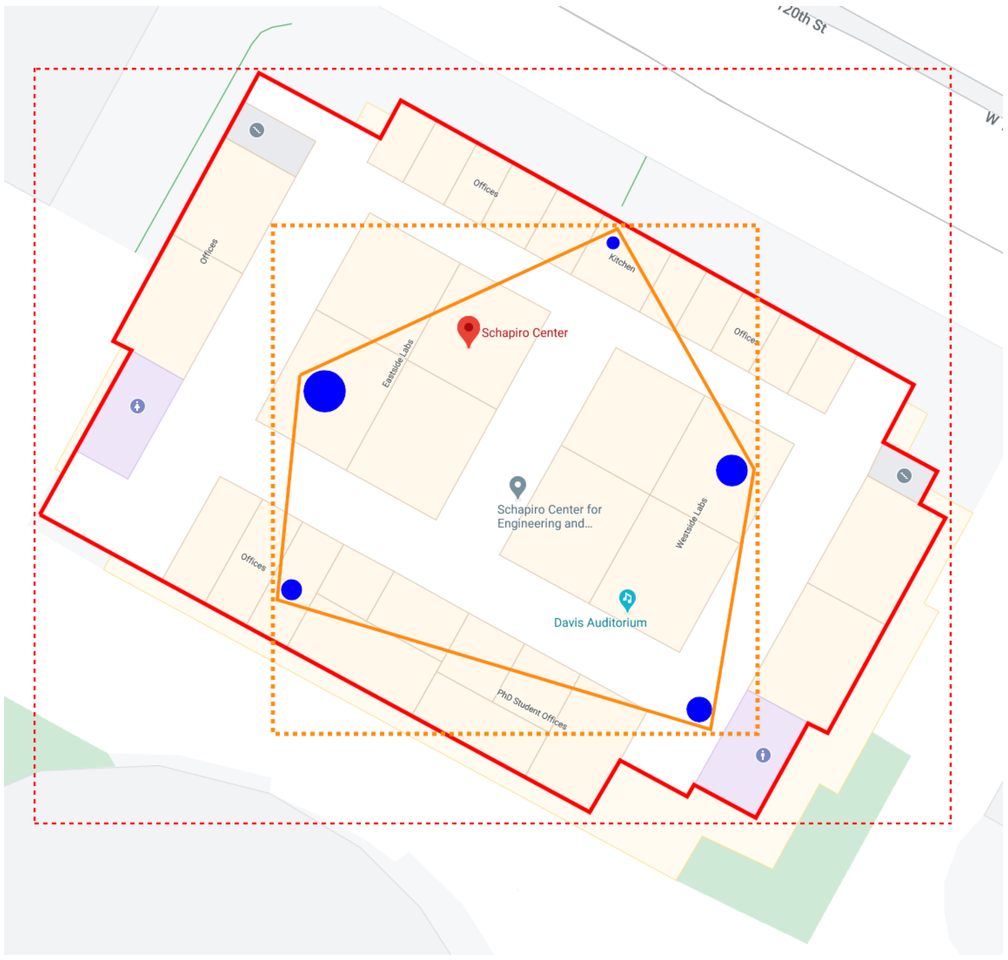


Figure 4.6: Device location estimates (blue), computed service area (orange), manually configured service area (red). Corresponding bounding boxes are shown with dotted lines.

remain scalable and performant.

Suppose the ACPS includes mobile components or its infrastructure changes over time. As sensors come and go, the ACPS needs to recompute its geographic service area and republish it via LoST if it has changed. The geographic service regions of multiple ACPSs may overlap. Consider, for example, ACPSs serving different floors of the same building.

Furthermore, we define the ACPS's *bounding box* as the minimum rectangle that fully contains the ACPS's geographic service area. If an ACPS's service area is represented with a GeoJSON format [109], the "bbox" property must be present and contain the bounding box for the ACPS's geographic service area.

4.5.2 LoST Mapping

The LoST protocol defines a <`mapping`> element that binds a service region with associated service URLs [86]. The element is an essential data structure for LoST servers. The server returns instances of the mapping element in response to `findService` requests.

The SenSQL framework uses the <`mapping`> element to map ACPS geographic service areas to contact URIs pointing to the public query interface provided by the ACPS service. The following example shows how the element is used in the SenSQL framework:

```
<mapping
    expires="2022-11-09T02:24:55"
    lastUpdated="2022-11-09T01:24:55Z"
    source="sensql.example"
    sourceId="7e3f40b098c711dbb6060800200c9a66">
    <displayName xml:lang="en">IRT Lab</displayName>
    <service>urn:sensql</service>
    <serviceBoundary profile="sensql">
        <gml:Polygon srsName="urn:ogc:def::crs:EPSG::4326">
            ...
        </gml:Polygon>
    </serviceBoundary>
    <uri>postgresql://user:password@hostname:port/db</uri>
</mapping>
```

The SenSQL framework sets the <`service`> element value to "urn:sensql" to indicate that the mapping element translates an ACPS's geographic service area to the ACPS's query interface. We also define a new profile "sensql" for the <`serviceBoundary`> element. The "sensql" profile indicates to the LoST server how to match polygons defined in the mapping. When the "sensql" profile is used, the LoST server matches the polygons from `findService` requests with the <`mapping`> polygons using the intersection predicate in the DE-9IM model described in Section 4.2.2.2.

With the SenSQL service, the <`serviceBoundary`> element can contain all geometries that are supported by the SQL/MM Spatial extension [107]. This includes polygons, line strings, points, and any combination of these elementary geometries. The geometries are stored in the GML [118] data format.

The `<uri>` element contains the URI of the ACPS's public query interface. This can be a database-specific URI, such as the one in the above example. A future revision of the SenSQL framework might define a database-agnostic transport mechanism.

The ACPS publishes its LoST `<mapping>` to the LoST service. The ACPS is expected to keep the mapping up-to-date and refresh it before it expires. SenSQL mapping elements are updated infrequently. We assume the LoST service provides a designated HTTP POST publication endpoint. The operation should use TLS and appropriate authentication and authorization mechanisms. The details of discovering the LoST service, its publication endpoint, or the credentials to be used by the ACPS are out of this project's scope.

4.5.3 Query Interface

Each ACPS provides a public SQL-based interface for querying exported sensors and stored data. The advantages of a SQL-based interface over just publishing the underlying data through a simpler API are: (1) clients can submit queries to data and get filtered or aggregated results only; (2) dynamic access control policies can be applied and enforced by the ACPS; and (3) information hiding can be supported. For simplicity, we assume all ACPSs use a common data model. At a minimum, the ACPS query interface must support the SELECT clause according to standard ANSI SQL [125].

An ACPS can also indicate support for the SQL/MM Spatial extension in its LoST service mapping. In that case, the query coordinator may send SQL queries with spatial predicates in the WHERE clause to the ACPS. The ACPS is expected to evaluate those predicates using supplied geospatial feature data. The query coordinator includes the features to be evaluated by value, i.e., inline, as part of the SELECT request. Spatial-compatible ACPSs are not expected to have access to the geographic naming database.

Alternatively, an ACPS supporting SQL/MM Spatial may contain its own private database with geographic features. The query coordinator could then provide a reference, e.g., a name or ID, to the features stored in the ACPS's private geographic database. The mechanism through which the query coordinator determines if the target ACPS contains private geographic features is left unspecified.

4.5.4 Data Model

For simplicity, we assume all ACPSs participating in a SenSQL system provide the same data model and export the same database schema through the query interface. This section briefly describes the data model.

Each ACPS provides a database called `sensql`. At a minimum, the database contains two mandatory tables *devices* and *measurements*. The tables can be provided as views over internal physical database tables using a different format. If the ACPS contains an internal geographic naming database, it can also export two optional tables *features* and *shapes*. The mandatory tables must conform to the format described below. The optional *features* and *shapes* tables, if present, must implement the same data model that is described in Section 4.7.1.

The following listing summarizes the format of the table *devices*. SenSQL query coordinators use the *devices* to discover devices according to the application's area of interest and the device's location. The columns `center` and `radius` represent the device's location uncertainty circle. The column `id` is a globally unique ID assigned to each device. The ID can be used as a key to the *measurements* table described later. The column `attrs` represents attribute-value key pairs that provide additional information (metadata) about the device.

```
 Column  |           Type           | Collation | Nullable | Default
---------+--------------------------+-----------+----------+---------
 id      | uuid                     |           | not null |
 network | text                     |           | not null |
 address | text                     |           | not null |
 port    | text                     |           |          |
 center  | geometry(Point)          |           |          |
 radius  | real                     |           |          |
 attrs   | jsonb                    |           | not null |
```

The following listing summarizes the format of the table *measurements*. The table is in PostgreSQL format. The *measurements* table provides access to the measurements taken by exported devices. Each row contains the data related to one measurement instance.

```
 Column  |           Type           | Collation | Nullable | Default
---------+--------------------------+-----------+----------+---------
 ctime   | timestamp with time zone |           | not null | now()
 id      | uuid                     |           | not null |
 network | text                     |           | not null |
 address | text                     |           | not null |
 port    | text                     |           |          |
 value   | jsonb                    |           | not null |
```

The column `ctime` represents the date and time when the measurement was taken. This column is used to order measurements and to obtain the measurement within a specified interval. The `ctime` column should be indexed.

The triplet (network, address, port) identifies the origin of the measurement data, i.e., a sensor that produced the measurement. On networked devices that include multiple sensors, the triplet identifies a particular sensor on the device. The network component is a logical name for a particular subsystem, e.g., "LoRaWAN," "MQTT," or something similar. The address component represents the device's address. The address must be unique within the subsystem. The port column then identifies a particular sensor (peripheral) on the device. When taken together, the three columns should form a globally unique identifier for the sensor that produced the measurement.

The column `value` encodes the measurement value in a well-known format. In the above example, the column has the general JSON type directly supported by DBMSs such as PostgreSQL. The actual format of the data is assumed to be in a well-recognized format such as SenML [127].

4.6 Stable Device Naming and Resolution

The importance of names for the usability of computer systems was first recognized by Saltzer [128, 129] in the context of operating and networked systems. A networked cyber-physical system consists of three fundamental entity types: physical components, computational processes, and data. Physical components represent sensors, actuators, computational devices, networking infrastructure, or storage devices. Computational processes are the programs running on computational devices. These could be the various processes implementing APIs, data collection and streaming, closed-loop control, and various auxiliary OS processes. Data entities refer to stored measurements, system configuration, or metadata describing the cyber-physical system.

A naming architecture consists of a naming system and the associated name resolver infrastructure. The naming system allows the application to assign general or application-specific identifiers to the fundamental entities. A string of characters typically represents the identifier. The resolver infrastructure allows the application to map the identifiers to locators. A locator is an identifier from a different namespace that allows the application to establish communication with the entity, e.g., to query the entity's current state.

Choosing an appropriate naming strategy is generally a complex problem. Ideally, the naming architecture should allow for stable applications that could be used unmodified across changes in the underlying system architecture. The names should also be short, contextual, and meaningful to the application programmer.

Since cyber-physical systems interact with the physical world via geographically dispersed sensors and actuators, the physical location of the system's components matters. This is unlike the Domain Name System (DNS), which is generally used to name abstract services irrespective of their topological or physical location[1].

The SenSQL framework allows the application to refer to physical components (sensors) with a combination of role (type) and geographic coordinates. We aim to provide stable names to the applications that do not change when the system changes, e.g., when individual devices are replaced during system evolution or upgrades. Geographic coordinates are typically represented with longitude and latitude pairs in the WGS 84 coordinate reference system. Such coordinates are globally unique but are difficult to remember and unintuitive to work with.

The SenSQL framework provides an indirect geographic naming system to avoid the need for the

[1]There are exceptions. DNS names increasingly include components representing a particular state, region, or area.

application programmer to deal with WGS 84 longitude-latitude pairs. The application programmer refers to physical devices (sensors) according to their relationship with a stable and well-known geographic feature, e.g., a building, street, or room name. This system resembles how humans refer to objects in the physical world, for example, “light bulb in the living room”, “temperature sensor in the garden”, and so on. Replacing the living room light bulb changes the device’s unique identifier but does not change the device’s relationship with the room (geographic feature). In other words, the newly installed device could be resolved under the same name, allowing the application to remain the same across infrastructure changes. Furthermore, we believe that names constructed from the device’s type and relative geographic location are intuitive for humans and might thus enable intuitive cyber-physical applications.

The process of resolving names to entities is called name resolution. The component or service performing name resolution is called a resolver. Before a resolver can resolve names, it must be configured with a “naming closure”. The closure provides a starting point for name resolution. For example, a DNS resolver must have a list of remote DNS servers. All that information is part of the DNS resolver closure. In the case of DNS, the operating system provides the closure mechanism. In the SenSQL resolver, the closure is provided by the Local Geographic Naming Database.

We illustrate the approach with a few examples. Consider an application interested in determining the current temperature near a geographic point. The application could submit a human-readable name, such as

```
temperature near 40.80782,-73.96213
```

to the SenSQL resolver. The above string identifies a collection of temperature sensors near the given position. The string contains a sensor type and a geographic position separated by the operator "near". The sensor type is chosen from a small set of well-known values representing common environmental sensor types. The operator is chosen from a small set of common geographic operators such as inside, nearest, along, around, within, inside, intersection, and others. The geographic position is represented by a longitude and latitude pair. The SenSQL resolver parses the natural language input. It converts the sensor type, operator, and geographic determiner into a SQL

WHERE clause. The SQL query is then submitted to the SenSQL SQL coordinator.

Because working with geographic longitude and latitude is unintuitive, the SenSQL resolver also allows substituting the coordinates with a feature name. For example, the longitude and latitude pair in the above string refers to the Alma Mater statue on the Columbia University campus. Thus, the above name could also be written as

```
temperature near "Alma Mater"
```

The SenSQL resolver treats the "Alma Mater" string as a geographic feature reference to the Geographic Naming Database described in Section 4.10.2. It resolves the reference to geographic coordinates via the database and then resolves the name.

The geographic feature reference does not need to refer to a point. It can refer to any zero (point), one (line string), or two (polygon) dimensional feature. Consider the following name, which refers to all temperature sensors inside a room:

```
temperature inside "Davis Auditorium"
```

The "Davis Auditorium" string refers to a boundary polygon representing a room within a building. The polygon is highlighted in blue in Figure 4.14. The geographic reference refers to features obtained from public databases like OpenStreetMap (buildings or streets) or privately imported features (floors, rooms, corridors).

The geographic feature can also be transformed using a small set of transformation functions inspired by the functions provided by SQL Spatial [107]. For example, the following name

```
temperature inside buffer("College Walk", 10)
```

resolves to all temperature sensors within the 10-meter buffer along a "College Walk" road. The above name can also be rewritten using the "along" operator, which automatically applies the buffer operation to the geographic feature reference:

```
temperature along "College Walk"
```

Finally, geographic references can be combined using logical expressions. For example, the following name resolves to all temperature sensors in either "IRT Lab" or "Davis Auditorium" rooms:

```
temperature inside ("IRT Lab" or "Davis Auditorium")
```

Regardless of how the name is constructed, the resolver must be able to translate the name to a SQL Spatial SELECT query. This requirement also determines the set of available feature transformation functions and combinations. These functions map one-to-one to the corresponding SQL Spatial functions. The following examples illustrate a few SQL queries generated by the resolver from the human-readable input strings:

```
SELECT device.id
FROM
    device,
    feature LEFT JOIN shape ON find_shape(ID)=shape.id
WHERE ST_Contains(shape.geometries, device.center)

select device.id
from
    device,
    feature left join shape on find_shape(ID)=shape.id
    order by st_distance(shape.geometries, device.center) asc
    limit 1

select distinct(device.id)
from
    device,
    feature left join shape on find_shape(ID)=shape.id
where
    st_contains(
        st_transform(
            st_buffer(
                st_transform(shape.geometries, 900913),
                10
            ), 4326),
        device.center)
```

4.7 Geographic Naming Database

The SenSQL service resolves dynamic CPS resources based on their relationships to well-known geographic features. The features are stored in a local geographic database. This section describes the design of the database and the process of populating the database with geographic data.

4.7.1 Data Model

Figure 4.7 describes the geographic data model as an entity-relationship (ER) diagram. At a high level, the database stores hierarchically organized geographic features and ancillary data. A geographic feature associates a geometric shape with attributes that describe the feature. The ancillary data enables defining features with respect to raster images (floor plans) and provides a means to orient such features in the world, i.e., to convert such features to the WGS 84 coordinate system. The application can query the features by their attribute values or relationships with other features in the DE-9IM intersection model. The following paragraphs describe individual entities.

The entity *Feature* represents geographic objects, both indoor and outdoor, such as buildings, floors, rooms, or roads. Features are hierarchically organized via their *parent* and *children* properties. This hierarchy is purely administrative and is not based on the geographic intersection relationship of the features. Each feature has metadata in the form of key-value attributes that describe the feature. This metadata can include address, room name or number, occupant name, and other information. The optional property *vertical range* represents, e.g., a floor by textual name or a pair of minimum-maximum altitudes.

The entity *Geometry* represents the horizontal physical shape of one or more features. Multiple features can refer to a single geometry entity. Geometry entities are spatially indexed and provide a means to evaluate the relationship of features according to the DE-9IM model. The coordinates

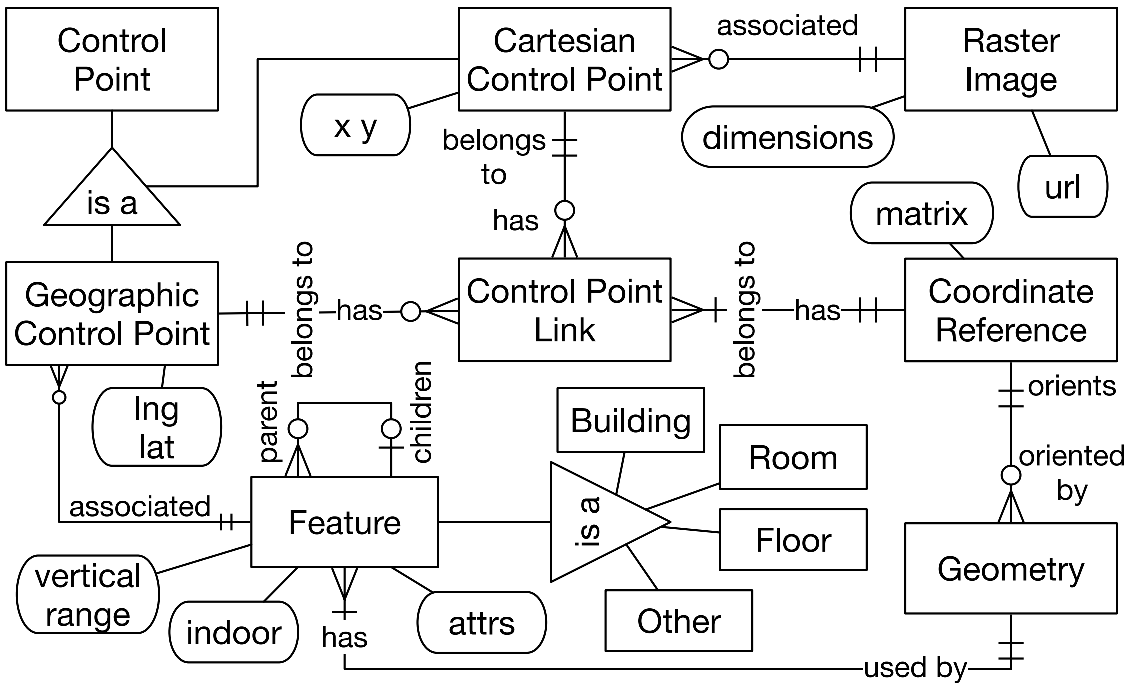


Figure 4.7: Entity-relationship model of the geographic naming database.

within a geometry are related to a coordinate reference. The reference can be a well-known reference, such as WGS 84, or a raster image, e.g., a floor plan. Geometries that use WGS 84 as a reference are geographic features oriented in the world, and their coordinates are pairs of longitude and latitude. Geometries that use a raster image as a reference represent points, lines, or polygons drawn over a raster image (e.g., a floor plan).

The entity *Raster Image* represents images uploaded to the database by the user, such as scanned floor plans.

The entity *Control Point* represents control points used to align raster images (floor plans) with a map. A *Geographic Control Point* has longitude and latitude coordinates and must be associated with a feature that uses the WGS 84 coordinate reference. A *Cartesian Control Point* has *x* and *y* coordinates and must be associated with a raster image which serves as its reference.

The entity *Control Point Link* binds a geographic control point with a cartesian control point. A link between two points indicates that the two points are colocated.

The entity *Coordinate Reference* represents a transformation from the cartesian coordinates in a raster image to WGS 84 coordinates and vice versa. The transformation is computed from a set of control point links and can be used to convert geometry from cartesian coordinates to WGS 84 coordinates.

4.7.2 Sources of Geographic Data

Table 4.2 summarizes potential geographic data sources considered for the SenSQL system.

Table 4.2: Sources of geographic data

	Google Maps	OpenStreetMap	BIM	Floor plans
User access	No	Yes	No	No
User editable	No	Yes	Yes	No
Accurate	Yes (?)	Yes and No	Yes (?)	No
Public areas	Yes	Yes	No	No
Private areas	?	No	Yes	Yes
Dimensionality	3D	2D	4D (time)	2D
Standardized	No	De facto	Yes	No
(Reverse) geocoding	Yes	Yes	No	No
Offline access	No	Yes	Yes	Yes
Aerial imagery	Yes	No	No	No

Google Maps is probably the most well-known and accurate source of geographic data. The database powering Google Maps contains information about publicly accessible geographic features such as buildings, streets, neighborhoods, parks, or landmarks. The Google Maps database also contains indoor information (rooms, floors) and complete 3D models for some buildings, but that information is not publicly accessible through an API. Restricted access to Google Maps data makes integrating the service into the SenSQL system challenging.

OpenStreetMap is a geographic and mapping internet service similar to Google Maps. OpenStreetMap provides a user-editable crowd-sourced geographic database. The accuracy of OpenStreetMap data depends on the region or area. Popular urban areas are generally represented accurately. Remote or rural areas may lack the accuracy of features. OpenStreetMap makes its database accessible under a permissive license.

Building information model (BIM) refers to a digital representation of physical infrastructure, for example, a building. The BIM concept is not new; proprietary BIM formats have existed since the 1970s under different names. Standardization began only recently; ISO released the BIM standard IEC ISO 81346 in 2019. Working with BIM data generally requires specialized tools. BIM models for existing buildings are generally non-existing or hard to obtain.

A BIM model for a building would be the ideal source of internal building geographic data for the SenSQL system. We have been looking for alternatives since such data is unavailable in most settings. One widely available alternative is emergency floor plans. Such floor plans are commonly available in shared or public buildings for emergencies. One often gets a floor plan when purchasing or renting a house or an apartment. Floor plans are a common data interchange format between building or apartment occupants and architects or interior design studios. Unfortunately, floor plans often have dubious accuracy and are quickly outdated. Emergency paper floor plans can be difficult to obtain and time-consuming to work with.

4.7.3 Acquiring OpenStreetMap Features

This section describes the process of acquiring geographic features from OpenStreetMap. The process is implemented in a web-based user interface. The input is a query string or objects selected on a map. The output is a metadata-augmented GeoJSON feature such as a building, street, intersection, neighborhood, park, or landmark.

OpenStreetMap (OSM) is a freely-editable geographic database of the World [124]. The project is collaborative and community-driven. OSM users crowd-source the underlying data from manual surveys, GPS, aerial photography, or local knowledge of an area. Unlike proprietary databases such as Google Maps, the geographic database powering OpenStreetMap is freely accessible under a permissive license.

The OSM database uses a relatively simple data model with only four core data types: nodes, ways, relations, and tags. Nodes represent features with no size (e.g., points of interest) and are also used in ways. In the database, nodes are represented with coordinates in the WGS 84 reference system. Ways represent linear features such as streets or rivers and areas such as building footprints or parks. The OSM database represents ways with line strings or polygons (if the way is closed). Relations represent relationships between existing nodes and ways. The OSM data model represents relations with ordered lists of nodes, ways, and other relations. Tags store arbitrary metadata in the form of key-value pairs. A tag is attached to a node, way, or relation.

The primary OSM database uses PostgreSQL DBMS, receives all user edits, and represents the whole world. The data only includes public information. For example, the OSM database contains information about buildings but not the building's floors or rooms. Each of the core data types is represented with a separate table. Rows represent individual nodes, ways, relations, and tags.

The SenSQL service provides a means to acquire geographic features from OpenStreetMap. We use two APIs to obtain the data: Overpass [130] and Nominatim [131]. The obtained features are converted to a GeoJSON format [109] and stored in the local geographic database.

The Overpass API allows the application to query the OSM database. The API is read-only and serves selected nodes, ways, relations, and tags. The client submits a query to the API, and the API returns the data corresponding to the query. The query is defined in the Overpass Query Language (QL). The data can be selected by criteria such as location, type of objects, properties, proximity, or a combination of those.

Nominatim is an OpenStreetMap API for geocoding and reverse geocoding. The API allows the application to search the OSM data by name and address (geocoding). It also provides a synthetic name and address, usually based on the nearest or enclosing OSM feature, for a pair of coordinates (reverse geocoding).

Please refer to Figure 4.8 for a block diagram of the OpenStreetMap feature acquisition process. The user navigates to a web-based UI to acquire a geographic feature from OpenStreetMap. The UI implements the whole acquisition process. The features to acquire can be interactively selected on a map or described with a query string. In the interactive map mode, the user clicks on or inside a feature to select it. The system computes the geographic coordinates of the point under the cursor.

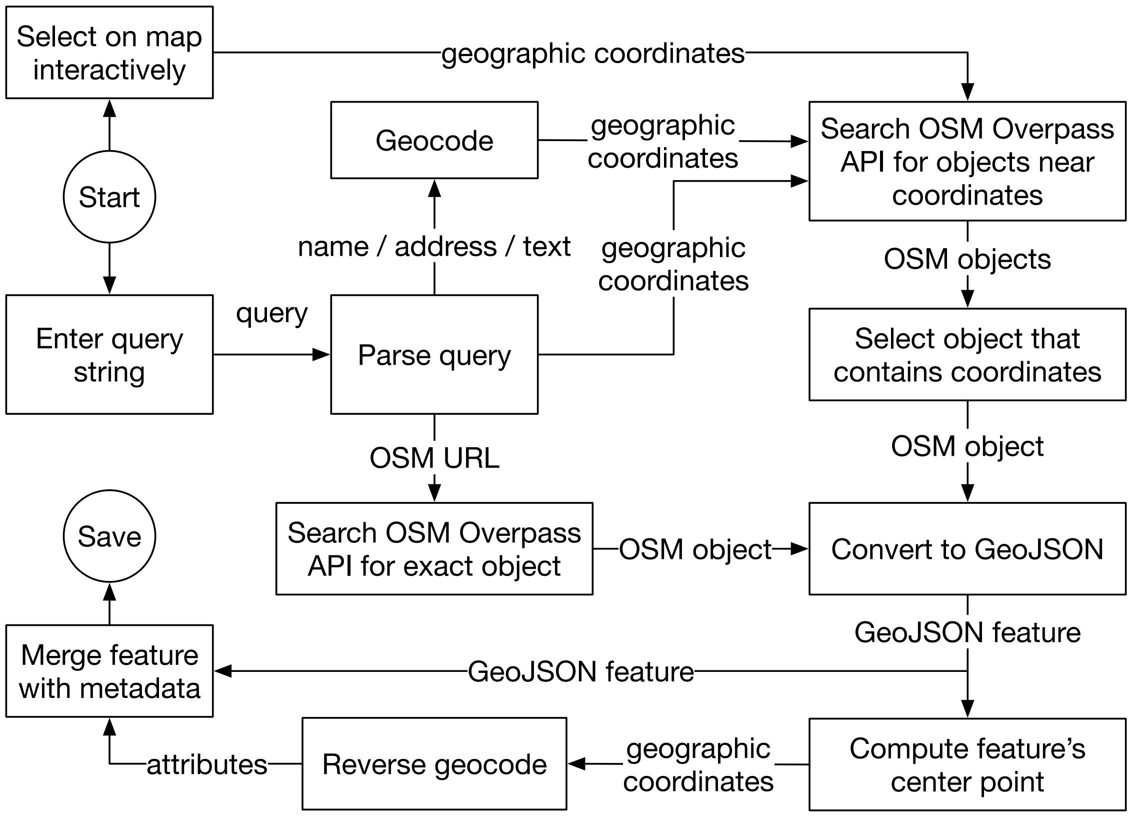


Figure 4.8: OpenStreetMap feature acquisition block diagram. The user selects the features to import in a web-based UI. The features can be selected interactively on a map or via a query string containing a name, address, OSM URL, or location.

If the user enters a query, the system parses it to determine if it is an OSM URL, geographic coordinates, or name or address. If the user enters a URL pointing to the OSM website, it is assumed to point to a particular object from the OSM database, and the acquisition process proceeds to convert the object to GeoJSON. Suppose the user enters a location string prefixed with "`loc:`"; it is assumed to be a longitude and latitude pair. Any other query is assumed to be a free-form name, civic address, or description text. The system geocodes[2] the text via the Nominatim API to convert it to geographic coordinates.

The coordinates are then converted into an OverpassQL query that searches the OSM database for objects representing buildings within a given radius from the coordinates. The system uses a default radius of 100 meters, but the query string can override the default radius or the type of the searched objects. Next, the system iterates through the OSM objects returned by the Overpass API,

[2]Geocoding is the process of converting a civic address to approximate location.

searching for an object that contains the given geographic coordinates. If such an object is found, it is returned. Otherwise, the system signals the user that the query is ambiguous, and the process restarts.

Currently, the system has a single OSM object without any helpful metadata. The system converts the OSM object to a GeoJSON feature and calculates the feature's center point. The system runs the center point coordinates through Nominatim's reverse geocoding API. If the API returns any known attributes such as feature name, address, or description, the attributes are merged into the GeoJSON feature. Finally, the system saves the GeoJSON feature in the local geographic database.

4.7.4 Acquiring Floor Plan Features

Section 4.7.3 describes acquiring geographic features from the OpenStreetMap database. That approach can only be used to obtain public geographic features such as streets or buildings. It cannot be used to obtain information about floors or rooms within a building because the OSM database does not contain such data. This section describes an interactive process for acquiring internal building geographic features via floor plans.

4.7.4.1 Digitizing Floor Plans

Suppose the user has access to a building floor plan. Such floor plans are commonly posted in buildings for emergency purposes. Figure 4.9 shows an emergency floor plan posted in the Shapiro Engineering Center (CEPSR) building on the Columbia University campus.

The user takes a photo of the floor plan with a smartphone. Dedicated text processing and image scanning applications exist on all major smartphone platforms. The application performs basic computer vision processing on the image. It crops, rotates, shears, straightens the floor plan, adjusts perspective, increases contrast, and optionally converts the image to black and white. The user then uploads the images to the geographic naming database via a web-based user interface. The images will be stored in the database as a *Raster Image* entity shown in Figure 4.7. Figure 4.10 shows two CEPSR floor plans scanned and processed with CamScanner [132], a free text and image scanning

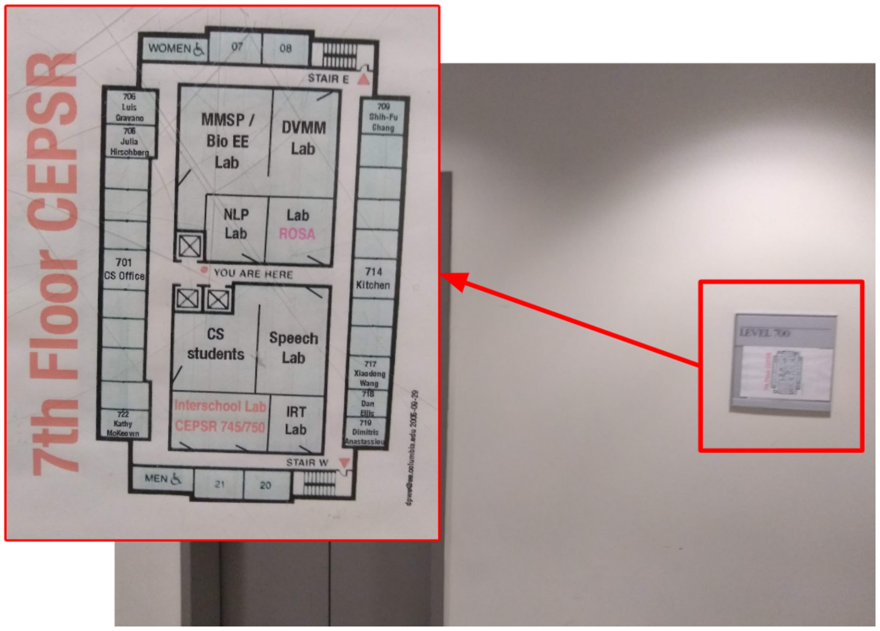
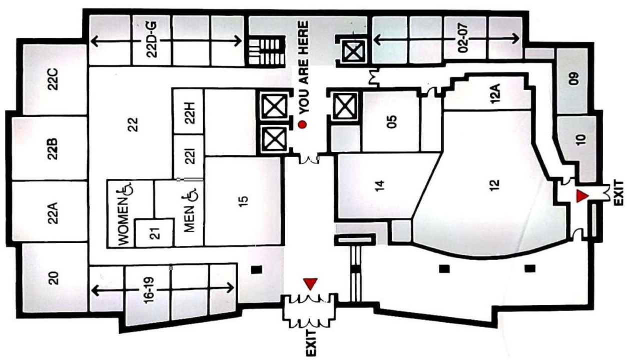
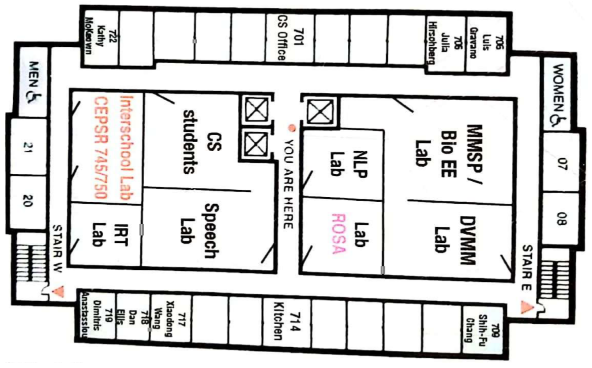


Figure 4.9: A floor plan posted on the 7th floor of CEPSR building on Columbia University campus.

application for the Android platform.



(a) CEPSR 4th floor



(b) CEPSR 7th floor

Figure 4.10: Two floor plans scanned and processed with CamScanner, a free image scanning application for Android.

4.7.4.2 Floor Plan Transformation

As the next step, the user performs floor plan orientation. The goal of the orientation step is to rotate, scale, and stretch the floor plan image to be matched (overlaid) over the corresponding building on a map. In other words, we want to find a transformation that maps the logical coordinates within the raster image to longitude and latitude, i.e., coordinates in the WGS 84 reference coordinate system.

The geographic naming database frontend uses machine learning to find the transformation based on minimal assistance from the user. Figure 4.11 shows the user interface. Suppose the user has acquired the geographic feature that describes the building from OpenStreetMap following the steps in Section 4.7.3. The front end shows the building polygon and the floor plan side by side.

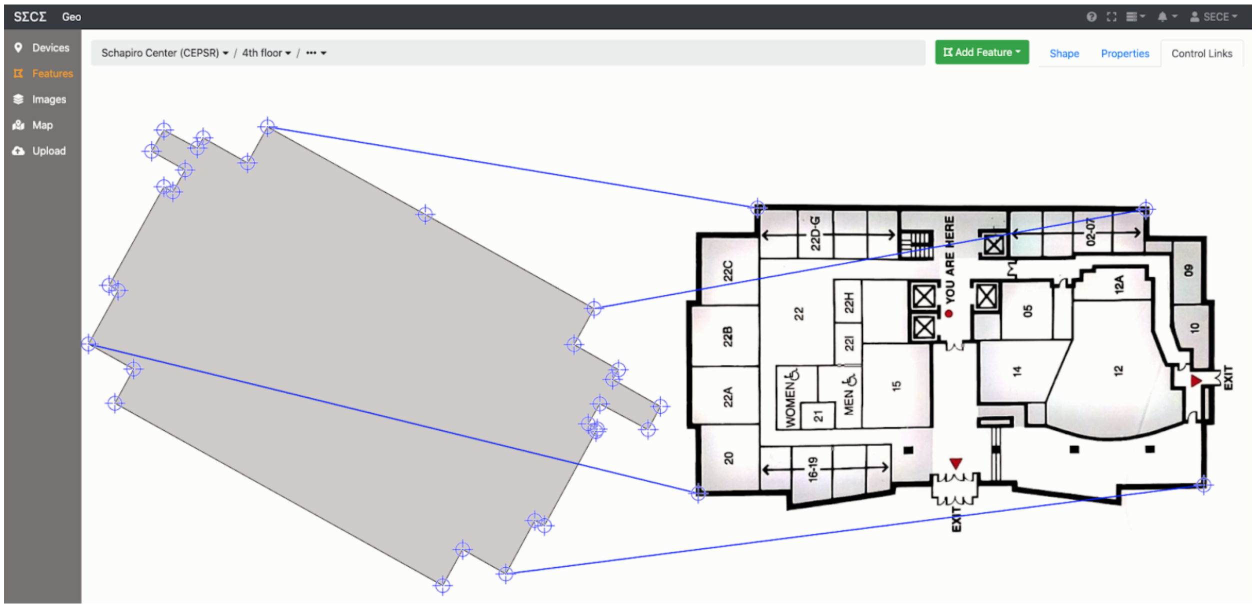


Figure 4.11: User interface to define control point links between an OpenStreetMap polygon and a raster image floor plan.

Crosshairs in Figure 4.11 represent draggable control points. The user interface automatically provides control points for all points in the building polygon, but the user can define new control points by double-clicking on the polygon. Similarly, the user can define control points on the floor plan image, which starts empty. The figure shows four floor-plan control points in the corners of the floor plan. The polygon control points and the floor plan can be connected via drag and drop. This is shown in the figure with lines connecting a subset of the control points in the building polygon to the control points in the floor plan.

After the user has defined and connected the control points, the system estimates a transformation that transforms the floor plan image to map the connected control points to the corresponding points in the building polygon.

A $3x3$ matrix describes the transformation. Depending on the number of connections, the transformation matrix describes similarity (two connections), affine (three connections), or projective (four or more connections) transformation. A similarity transformation only performs rotation, translation, isotropic scaling, and reflection of the source image. The transform preserves shape. An affine transformation can additionally shift the origin. It does not preserve shape, but it does preserve parallelism. A projective transformation is the most powerful of the three. It does not preserve parallelism or shape but preserves collinearity and incidence in the transformed image. More powerful transformations exist (e.g., polynomial transformation), but such methods require many control points and are sensitive to their placements. The projective transformation tends to give the best results with a relatively small number of control points interactively selected by the user.

The system then estimates the parameters for the transformation using the least-squares method described in [133]. The input to the algorithm is the connected control point pairs. The algorithm produces a $3x3$ matrix that transforms the floor plan image into the WGS 84 coordinate reference system. The system stores the resulting transformation matrix in the geographic naming database as *Coordinate Reference* entity. Figure 4.12 shows the result of the transformation graphically. The figure shows the building polygon (blue overlay) and the floor plan image overlaid over an OpenStreetMap map. The transformation was estimated using the four control point links shown in Figure 4.11.

The accuracy of the least-squares transformation estimation method depends on the number of control point links and their distribution across the source image. The astute reader might notice inaccuracies along the northern side of the building in Figure 4.11 where the corners of the building are not correctly aligned with the floor plan. The alignment can be improved by defining additional control points and connections in those areas. The alignment quality depends on the quality of the floor plan image and the number and locations of control points. The orientation process is iterative and requires skill and some training for effective control point placement.

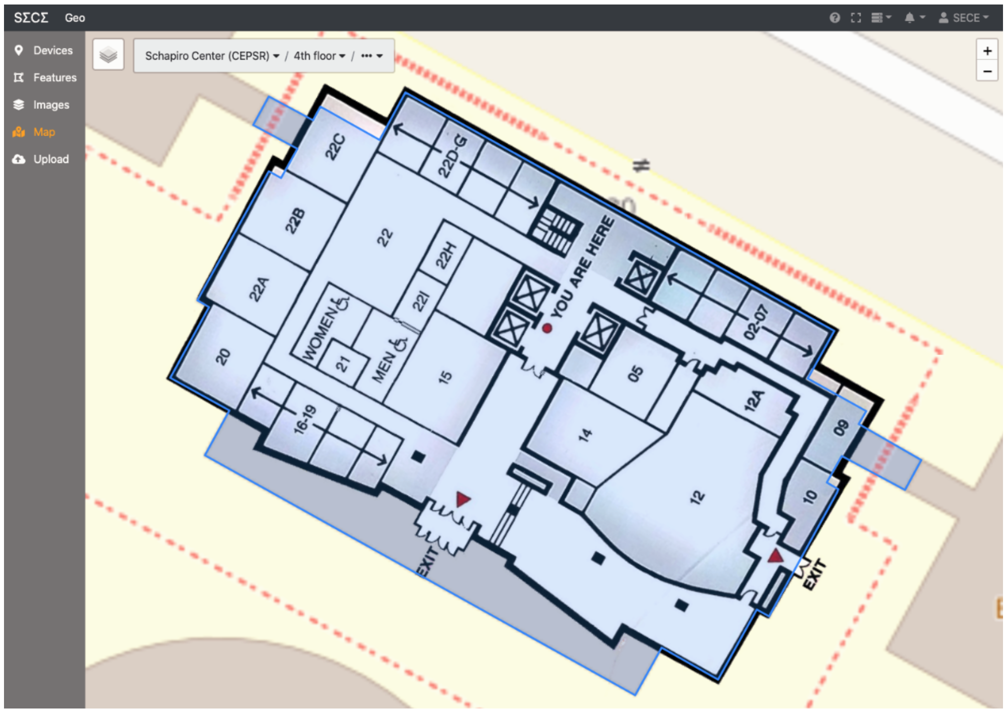


Figure 4.12: Floor plan image transformed and overlaid over the building polygon (blue) on an OpenStreetMap map. The building is Shapiro Engineering Center (CEPSR) on the Columbia University Campus. The floor plan shows the fourth floor.

4.7.4.3 Defining Polygons over a Floor Plan

The final step in the floor plan acquisition process is defining polygons representing rooms, hallways, labs, and other indoor partitions. This step is difficult to automate, given the poor quality and the lack of semantic information in most paper floor plans. Thus, we have designed a simple manual vector editor for our prototype where the user can manually define polygons for the rooms of interest. Figure 4.13 shows the component editing the polygon for Davis Auditorium room on the 4th floor of the Shapiro Engineering Center (CEPSR).

The editor lets users define and edit polygons over the selected floor plan image. The polygon is saved in the geographic naming database under the specified name. The polygon representing the entire floor (in this case) or building will be set as the entity's parent. The editor also lets the user define additional properties associated with the object. This feature is not shown in the figure.

When the system saves the manually generated feature in the geographic naming database, it

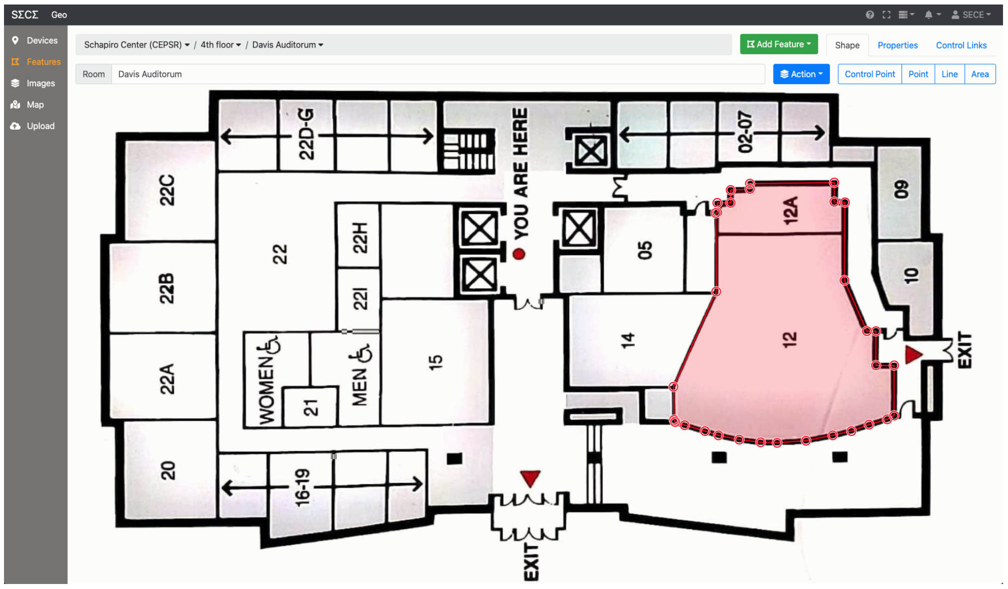


Figure 4.13: SenSQL polygon editor user interface

applies the earlier estimated transformation for the floor plan image. The transformation is applied to all points in the polygon. The result is a geographic feature whose shape (polygon) is in the WGS 84 coordinate reference frame and thus oriented in the world. Figure 4.14 shows the created geographic feature for Davis Auditorium oriented and overlaid over an OpenStreetMap map.

4.8 Query Coordinator

Applications submit their SQL queries for ACPS devices and measurements to a query coordinator. The coordinator is provided to the application as part of the application's runtime environment. This could be cloud infrastructure (Amazon AWS, Google Cloud Platform) or a locally running service. Like an SQL database, the query coordinator is not part of the application. As a service, it is intended to be shared by several applications running in the runtime environment, as an ordinary SQL database would.

The query coordinator provides the application with a subset of ANSI SQL [125]. Specifically, the interface supports WITH-SELECT-FROM-WHERE-GROUPBY-HAVING queries over the data gathered from ACPSs. Transactions or data modifying operations such as INSERT, UPDATE, or DELETE are not supported for simplicity. Supporting only a subset of the SQL language was a

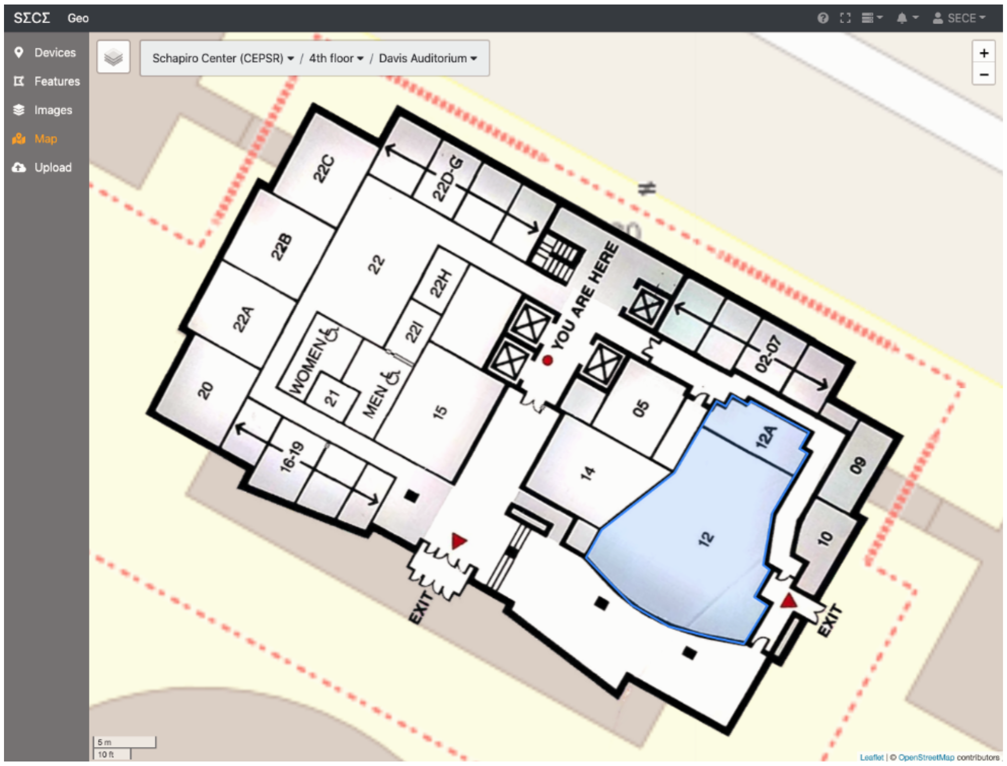


Figure 4.14: A geographic feature for Davis Auditorium in CEPSR shown on an OpenStreetMap map.

deliberate decision to simplify implementation. The primary goal of the SenSQL framework is to provide the application with read-only access to sensors and measurement data.

Upon receiving a SQL query from the application, the query coordinator parses the WHERE clause of the SQL query for references to geographic features. If the WHERE clause contains such references, the coordinator resolves the references to geographic objects via a locally configured geographic naming database.

The geographic naming database serves as a local cache for geographic data retrieved from external geospatial systems such as OpenStreetMap. It contains objects representing cities, neighborhoods, building footprints, and other information the application may want to query. The application can thus query for sensor data related to a particular geographic object, e.g., the data within a neighborhood, a building, or along a road or intersection.

To determine which autonomous cyber-physical systems might offer data relevant to the SQL query submitted by the application, the query coordinator maps the resolved geographic objects through a LoST service. The LoST service returns candidate ACPSs based on whether or not their service areas intersect the corresponding geographic feature.

Once a set of candidate ACPSs has been found, the query coordinator rewrites the SQL query to remove all resolved geographic features, effectively turning the application's SQL query into several SQL subqueries, one for each ACPS. The query coordinator then forwards the subqueries to the ACPSs using the contact URIs published by the ACPSs via the LoST service.

Upon receiving response data from the ACPSs, the query coordinator aggregates and filters the data before forwarding it to the application. Thus, the application is presented with a unified view, in other words, a virtual database table whose records span ACPSs distributed and federated via the SenSQL system. In database terminology, the SenSQL service implements a sharded database model with a geographic mapping algorithm.

4.9 LoST Service

A key challenge in the design of the SenSQL service discussed in this chapter is allowing SQL query coordinators to efficiently discover all autonomic cyber-physical systems (ACPSs) related to a geographic region. Various forms of geographic routing, addressing, and discovery have been explored by the networking community [91, 92, 93, 94]. This section discusses the design of an ACPS discovery service based on the LoST framework [95, 86]. Figure 4.15 illustrates the overall architecture of the LoST service used in SenSQL.

The LoST service is a network service that allows LoST seekers (query coordinators) to discover LoST publishers (ACPSs) by their service area published into a shared database. Specifically, a seeker can discover all publishers with coverage regions that intersect[3] the seeker's area of interest.

The LoST service is conceptually simple; it provides a rendezvous point between query coordinators and autonomous cyber-physical systems. ACPSs publish their service areas and other ancillary information to the service. Query coordinators look up published service areas in the LoST service based on areas of interest. A scalable and reliable design is essential since the LoST

[3]The term intersect here refers to the standard Dimensionally Extended 9-Intersection Model (DE-9IM) model.

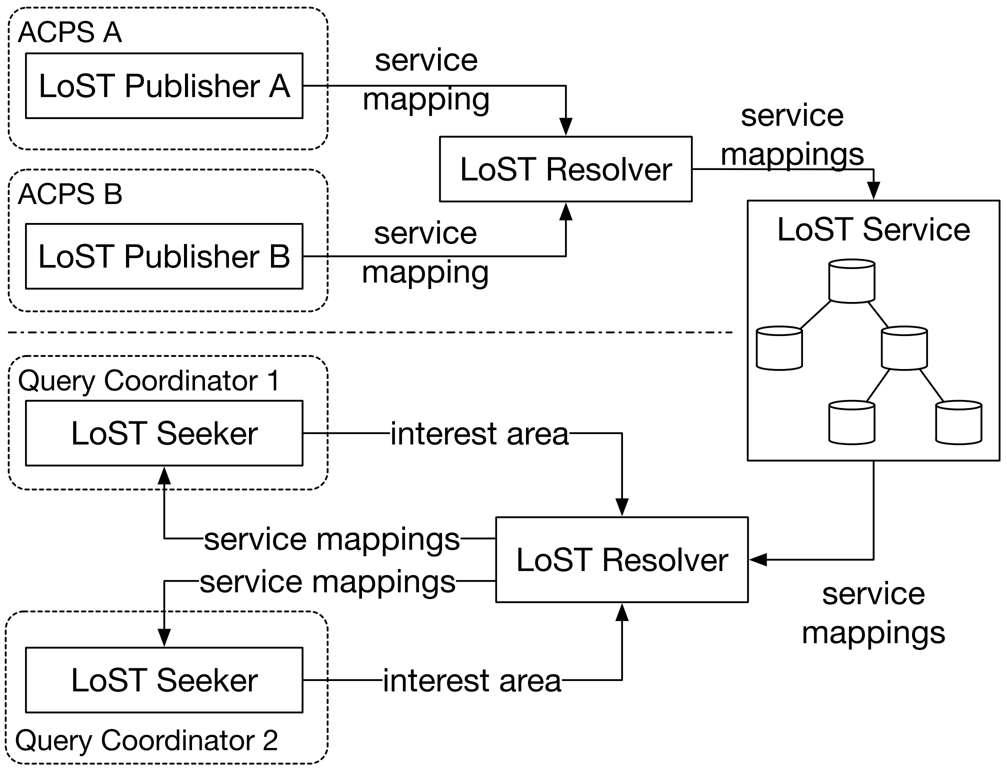


Figure 4.15: SenSQL LoST service architecture

service potentially represents a reliability or scalability bottleneck.

Support for authority delegation. The LoST service should support the delegation of authority over portions of the data. Specifically, it should allow delegating responsibility over the data to existing trustees for country-code top-level domains. This enables a multi-stakeholder model of registration policymaking.

Deployment considerations. The service should be easy to deploy and operate using existing network services (DNS, Amazon S3, cloud functions). Computationally expensive operations such as complex geographic object matching should be performed near the application, e.g., in the LoST resolver or seeker near the query coordinator. That way, the computationally expensive operations can be billed to the application. Furthermore, the LoST service should be efficient for large interest areas, e.g., an area covering Manhattan.

4.9.1 Server Hierarchy

The service consists of hierarchically organized, independently operated LoST servers. At the top of the hierarchy is a small set of well-known root servers. The root servers are similar to DNS root servers and must be synchronized with each other.

The root servers partition the World into large geographic regions and assign an authoritative LoST server to each region. The partitioning enables designated authoritative LoST servers in different jurisdictions.

Unlike in DNS, where delegation is based on domain labels, the delegation in the LoST service is based on geographic relationships. Each regional LoST server can delegate portions of its region to other LoST servers authoritative only for smaller sub-regions. Except in complicated scenarios that involve complex regional partitioning or complex interest or service area polygons, we envision a flat hierarchy of servers, provided that the hierarchy can handle the load generated by query coordinators and ACPSs.

The LoST service uses DDDS [113] to map larger geographic regions to LoST servers authoritative for the region. Note that a region can be resolved to multiple authoritative LoST servers, e.g., a region at the border of two countries. The application can query both or be configured with a country identifier that determines which one to pick.

4.9.2 Resolving

LoST seekers communicate with LoST servers via resolvers that can reach the root servers. This typically happens via static configuration, i.e., the resolver will be provided with root server addresses or some mechanism to discover those servers. A LoST resolver can traverse the LoST server tree iteratively or recursively.

LoST resolvers can be external to applications and may be shared by multiple applications. Resolvers are expected to cache and reuse results obtained from the LoST service.

4.9.3 Publishing

The LoST service is designed to make it easy for ACPSs to publish their service mappings. The process involves uploading a digitally signed XML document to the LoST service via HTTP. The upload takes place via an entity known as the LoST publisher. The publisher traverses the tree of authoritative LoST servers until it finds the server responsible for the area of the published mapping. The XML document is then inserted into the server's mapping database.

4.9.4 Mapping

Discovering published service mappings is a two-step process. First, the resolver finds a set of LoST servers authoritative for the area of interest received from a LoST seeker. Second, the resolver interrogates the authoritative servers for relevant service mappings.

Relevant service mappings are those service mappings in the LoST server's database that intersect with the interest area polygon. We define the intersection operation using the DE-9IM intersection model developed for the SQL spatial extension and operations [108].

4.9.5 Example

Consider a LoST seeker trying to discover all ACPSs within the Morningside Heights neighborhood in Manhattan. The seeker submits a geographic object representing the neighborhood's boundary polygon to a LoST resolver. The resolver begins traversing the tree of LoST servers to discover the authoritative server for the area. This will likely be a server authoritative for New York State, the East Coast, or the United States.

Once an authoritative LoST server has been found, the resolver submits a `<findService>` request to the server, including the boundary polygon for the Morningside Heights neighborhood. The server includes all service mappings entirely inside, partially inside, or at least touching the neighborhood's boundary polygon. Since the returned service mappings include those that only partially lie within the neighborhood, the LoST seeker needs to perform application-level filtering.

In other words, the seeker (or the application that uses it) should exclude all devices from the partially matched ACPS outside the neighborhood.

4.10 Prototype Implementation

We implemented functional prototypes for most of the components in the SenSQL framework to evaluate the feasibility of the approach. This section briefly describes the prototype. In all our experiments with the prototype, we used either `psql`, the command line SQL client distributed with PostgreSQL or Grafana. Grafana is a popular web-based user interface framework for building custom graphs, dashboards, and reporting functions. Grafana supports SQL, including SQL/MM Spatial. The programs that fetch data take the form of SQL SELECT statements.

4.10.1 Query Coordinator

We prototyped the query coordinator component using Facebook's Presto [134], an open-source distributed SQL query engine. We used version 0.260. Presto is a SQL proxy that can provide a generic SQL query interface for non-SQL or non-relational database backends. The program is implemented in Java and is extensible. It has a collection of Connector modules, one for each target backend database. New connectors can be created on top of a well-documented API. Presto also supports the SQL/MM Spatial extension out of the box.

Presto includes a generic SQL query parser, optimizer, planner, and scheduler implemented in Java. The scheduler breaks down an SQL query into multiple sub-queries to be forwarded to target backend databases based on the names of the tables they reference. On the way back, Presto can aggregate and filter data contributed by multiple backend databases before the data is forwarded upstream to the application. Thus, Presto presents a single logical SQL database to the application with contents potentially distributed (sharded) across multiple database backends. Not all of the backends have to be based on SQL.

For the SenSQL framework, we extended Presto with two features. We have modified Presto's SQL query planner and scheduler to search for references to the special tables *features* and *shapes*.

Presto would create a sub-query to fetch the corresponding geographic features from the geographic naming database for each reference to the two tables in the WHERE clause of the input SQL query. The references in the input queries would then be replaced with polygons representing the features.

The second Presto modification allows the application to discover target databases dynamically using the LoST service. Having replaced all named references to geographic features in the input WHERE clause with their definitions, the SQL query planner consults a pre-configured LoST service for each geographic feature. We implemented a custom Java module that submits a LoST <`findService`> request to the LoST service. The request includes one geographic feature in the form of its boundary polygon.

The LoST server returns the mappings to all databases that might provide data related to the boundary polygon. Presto dynamically adds those databases into its database catalogs and then issues a rewritten version of the input query to each database. The rewritten query is the input SQL query with all references to the geographic objects removed from the WHERE clause.

The prototype, as implemented, had several limitations. It assumed all target (ACPS) databases were based on PostgreSQL and spoke its wire protocol. We implemented no authentication or authorization between the query coordinator and ACPS databases. Finally, the prototype did not support geographic features submitted by value, i.e., in the form of a coordinate list, by the application. The application would first need to store the feature in the geographic naming database under a name that could be referenced from the WHERE clause. The query coordinator prototype also did not implement authentication or authorization when communicating with the LoST service.

4.10.2 Geographic Naming Database

We prototyped the geographic naming database using PostgreSQL 13.5 with the PostGIS extension. The PostGIS extension adds SQL/MM Spatial support to vanilla PostgreSQL. The data model provided by the database is described in detail in Section 4.7.1. The database contained a separate table for each main entity shown in the data model: features, shapes, raster images, control points, and coordinate transforms.

To populate the database with data (geographic features), we have designed and built a web-based user interface (UI). The front end was implemented in JavaScript and React. The backend was implemented in Python 3 on top of the Flask framework. Figures 4.16 to 4.18 and Section 4.7 show screenshots of the UI prototype. The UI backend ran on the same Linux host as the PostgreSQL database in all our experiments.

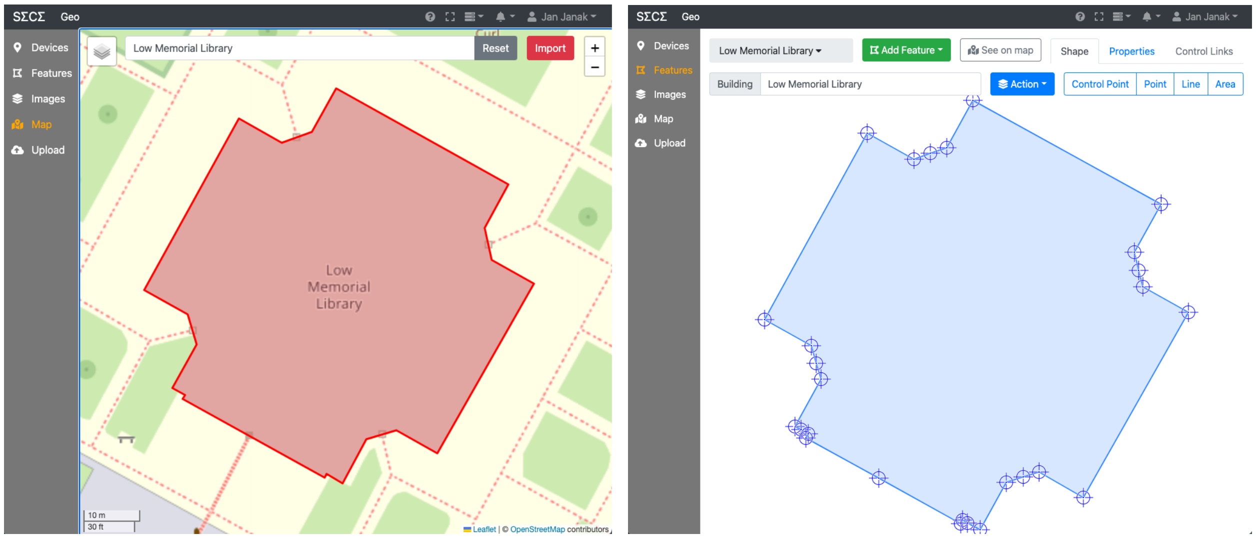
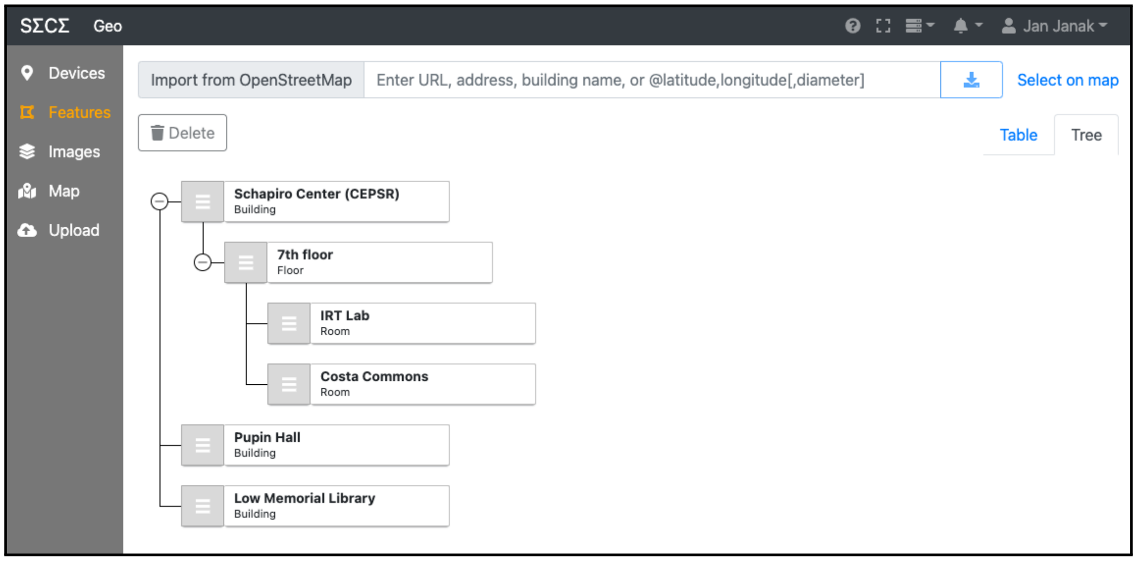


Figure 4.16: To interactively import a geographic object, the user selects the object on a map and clicks the import button. Crosshairs represent the object's control points.

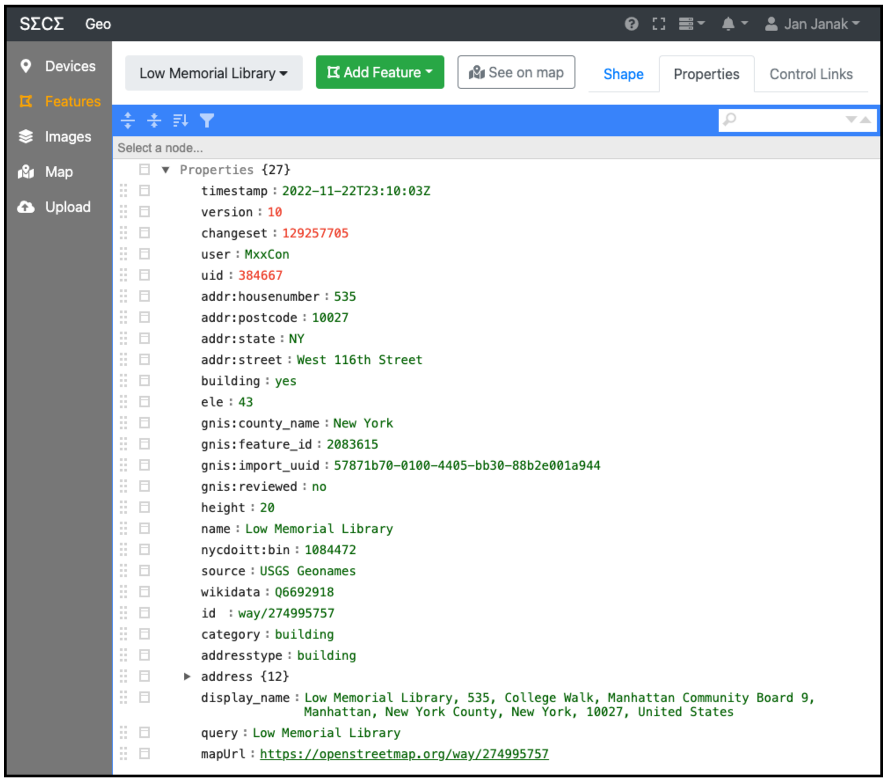

ΣΣΣ Geo
Jan Janak
Devices
Features
Images
Map
Upload
Import from OpenStreetMap
Enter URL, address, building name, or @latitude,longitude[,diameter]
Select on map
Delete
Table
Tree

Schapiro Center (CEPSR)
Building
- 7th floor
  Floor
  - IRT Lab
    Room
  - Costa Commons
    Room

Pupin Hall
Building

Low Memorial Library
Building

Figure 4.17: A hierarchical tree of imported geographic objects.



```
Properties {27}
    timestamp : 2022-11-22T23:10:03Z
    version : 10
    changeset : 129257705
    user : MxxCon
    uid : 384667
    addr:housenumber : 535
    addr:postcode : 10027
    addr:state : NY
    addr:street : West 116th Street
    building : yes
    ele : 43
    gnis:county_name : New York
    gnis:feature_id : 2083615
    gnis:import_uuid : 57871b70-0100-4405-bb30-88b2e001a944
    gnis:reviewed : no
    height : 20
    name : Low Memorial Library
    nycdoitt:bin : 1084472
    source : USGS Geonames
    wikidata : Q6692918
    id  : way/274995757
    category : building
    addresstype : building
  ▸ address {12}
    display_name : Low Memorial Library, 535, College Walk, Manhattan Community Board 9,
                   Manhattan, New York County, New York, 10027, United States
    query : Low Memorial Library
    mapUrl : https://openstreetmap.org/way/274995757
```

Figure 4.18: Attributes attached to the geographic object.

Apart from the tables listed above, the database also provided convenience functions in the form of stored procedures. Those functions are used in SQL examples scattered across this chapter. The functions were generally designed to simplify the custom functionality in the Presto SQL parser and planner.

4.10.3 LoST Service

We did not find a freely available LoST server implementation suitable for the SenSQL project, so we implemented a subset of the LoST server functionality. The LoST server interface and API were implemented in Python and Flask. All polygon matching, the core activity of a LoST server, was implemented with SQL/MM Spatial and delegated to a dedicated PostGIS database. Thus, the Python-based LoST server program only implemented simple conversions between LoST XML data formats and PostGIS SQL operations.

Our LoST server prototype only implemented the minimum subset of functionality needed for the SenSQL framework. It implemented the <`findService`> request and response and a custom HTTP API end-point for ACPSs to publish their mapping to.

The implemented prototype LoST service consisted of only one LoST server. This was sufficient for our small-scale experiments with few ACPSs with relatively simple service area polygons. A more distributed architecture with LoST forest guides and trees of servers might be necessary for deployments with larger numbers of ACPSs. This is left for future work.

4.11 Evaluation

We designed and developed a container-based evaluation environment for the SenSQL framework. Figure 4.19 illustrates the architecture of the environment. The emulator can launch a configurable number of emulated devices, autonomous cyber-physical systems, and other supporting infrastructure. The framework uses modern container-based techniques to isolate individual components and to make it possible to re-configure those to emulate the target scenario properly.

An Emulation Manager component controls the system and creates an emulation environment to support a particular evaluation scenario. The manager instantiates virtual (emulated) sensor devices and emulated database nodes representing ACPSs. Each emulated ACPS runs a Sensor Subscriber component that maintains MQTT [135] subscriptions with all emulated sensors associated with the ACPS. The subscriber periodically reads the sensors' measured values and stores them in the ACPS's database (PostgreSQL). The Emulator Manager manages the association between virtual sensors and ACPSs. The emulation environment allows the dynamic creation of emulated scenarios involving a configurable number of virtual sensors and autonomous cyber-physical systems.

To evaluate the SenSQL framework, we created an emulated environment covering the Columbia University Morningside Heights (main) campus. We launched 25 autonomous cyber-physical systems, one for each Morningside Heights campus building. The building's boundary polygon determined the service area of each ACPS. The polygons were obtained from OpenStreetMaps. Figure 4.20 some of the boundary polygons, which are stored in a local geographic database.

We then used the evaluation environment to generate a large number of sensors randomly

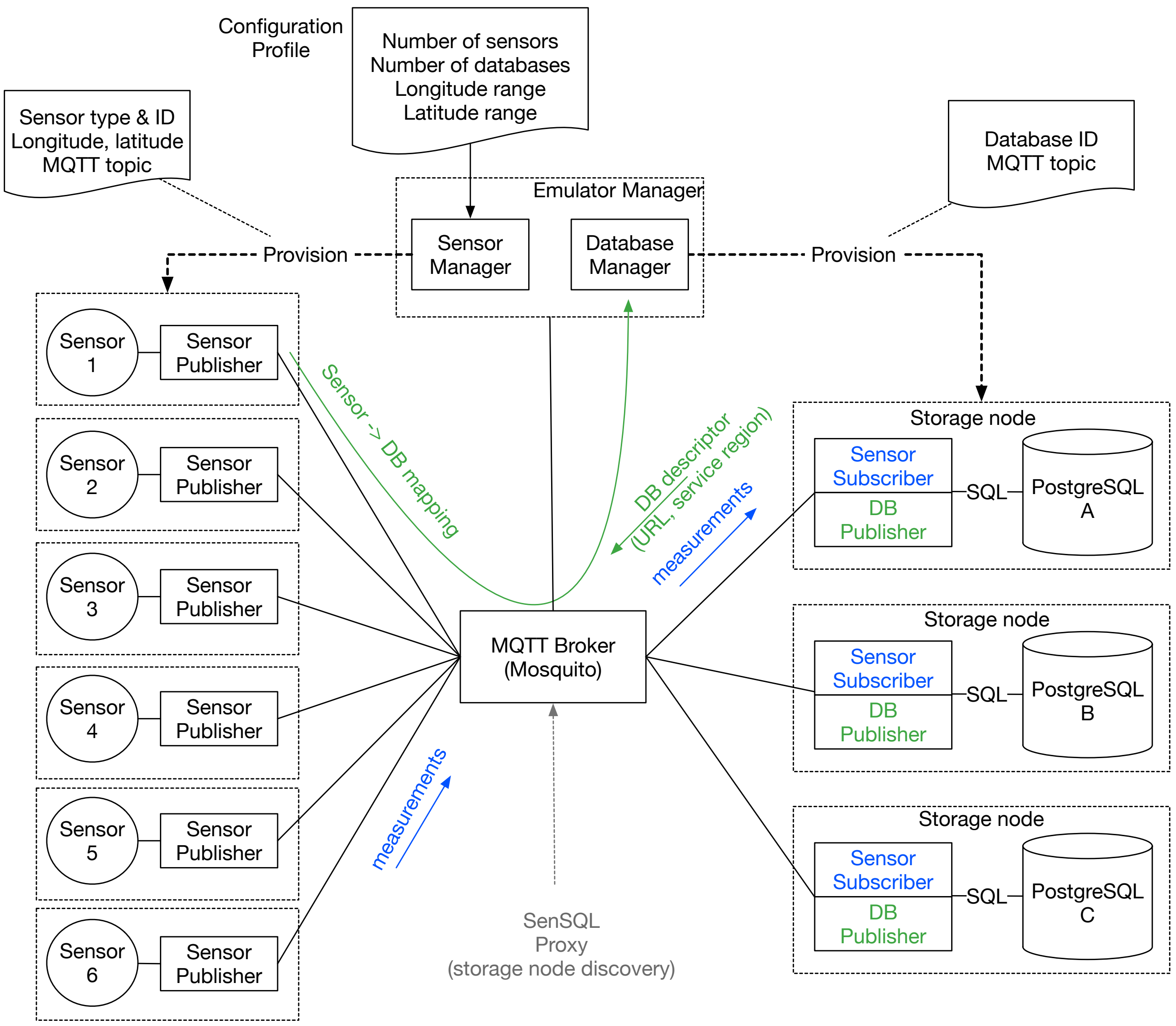
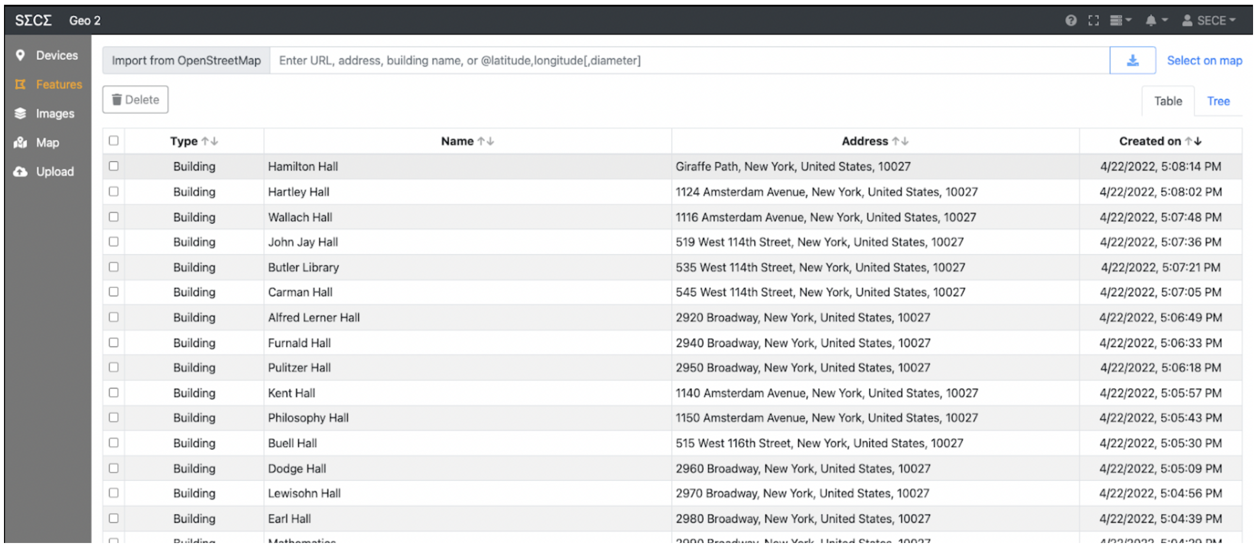


Figure 4.19: Evaluation environment for the SenSQL service

distributed across the campus. Figure 4.21 shows a configuration with thousands of sensors. The green discs represent the sensors' location uncertainty radius. The evaluation environment then associated each sensor with the enclosing building's autonomous cyber-physical system. The sensors associated with the ACPS for Shapiro Center are highlighted in red. Outdoor sensors were associated with a catch-all ACPS for the entire Morningside Heights campus.

With the emulation environment set up, we then run a number of SQL queries to test that the SenSQL framework correctly discovers all devices related to the query. Figure 4.22 shows the result of a SQL query to discover all sensors in the Davis Auditorium room of Shapiro Engineering Center


SΣCΣ
Geo 2
SECE
Devices
Features
Images
Map
Upload
Import from OpenStreetMap
Enter URL, address, building name, or @latitude,longitude[,diameter]
Select on map
Delete
Table
Tree

	Type	Name	Address	Created on
☐	Building	Hamilton Hall	Giraffe Path, New York, United States, 10027	4/22/2022, 5:08:14 PM
☐	Building	Hartley Hall	1124 Amsterdam Avenue, New York, United States, 10027	4/22/2022, 5:08:02 PM
☐	Building	Wallach Hall	1116 Amsterdam Avenue, New York, United States, 10027	4/22/2022, 5:07:48 PM
☐	Building	John Jay Hall	519 West 114th Street, New York, United States, 10027	4/22/2022, 5:07:36 PM
☐	Building	Butler Library	535 West 114th Street, New York, United States, 10027	4/22/2022, 5:07:21 PM
☐	Building	Carman Hall	545 West 114th Street, New York, United States, 10027	4/22/2022, 5:07:05 PM
☐	Building	Alfred Lerner Hall	2920 Broadway, New York, United States, 10027	4/22/2022, 5:06:49 PM
☐	Building	Furnald Hall	2940 Broadway, New York, United States, 10027	4/22/2022, 5:06:33 PM
☐	Building	Pulitzer Hall	2950 Broadway, New York, United States, 10027	4/22/2022, 5:06:18 PM
☐	Building	Kent Hall	1140 Amsterdam Avenue, New York, United States, 10027	4/22/2022, 5:05:57 PM
☐	Building	Philosophy Hall	1150 Amsterdam Avenue, New York, United States, 10027	4/22/2022, 5:05:43 PM
☐	Building	Buell Hall	515 West 116th Street, New York, United States, 10027	4/22/2022, 5:05:30 PM
☐	Building	Dodge Hall	2960 Broadway, New York, United States, 10027	4/22/2022, 5:05:09 PM
☐	Building	Lewisohn Hall	2970 Broadway, New York, United States, 10027	4/22/2022, 5:04:56 PM
☐	Building	Earl Hall	2980 Broadway, New York, United States, 10027	4/22/2022, 5:04:39 PM

Figure 4.20: Building polygons for Columbia University Morningside Heights main campus.

(CEPSR). We tested a variety of SQL queries involving many SQL Spatial functions and operators.

4.12 Related Work

Various data processing and query execution architectures have been explored for ad hoc and wireless sensor networks. Such networks are challenging due to their highly distributed nature, limited network bandwidth, and various constraints placed on individual nodes.

TinyDB [136] is a query processing system for wireless sensor networks. TinyDB provides a declarative SQL-like query interface to the application. The interface implements a subset of standard SQL with custom extensions to control sensor measurements, i.e., where, when, and how often data is acquired. TinyDB makes tradeoffs for efficient query execution on wireless energy-constrained sensor networks. Individual nodes (sensors) do not store data. Any data generated by the queries are communicated to a central (root) node where it can be stored and passed to the application. The root node is also responsible for all query planning and optimizations. On-demand (acquisitional) sensing helps the application use constrained resources efficiently. This model is, however, overly limiting in heterogeneous systems where the cost of acquiring data is of secondary concern.

Cougar [137] uses a similar distributed database architecture based on in-network aggregation

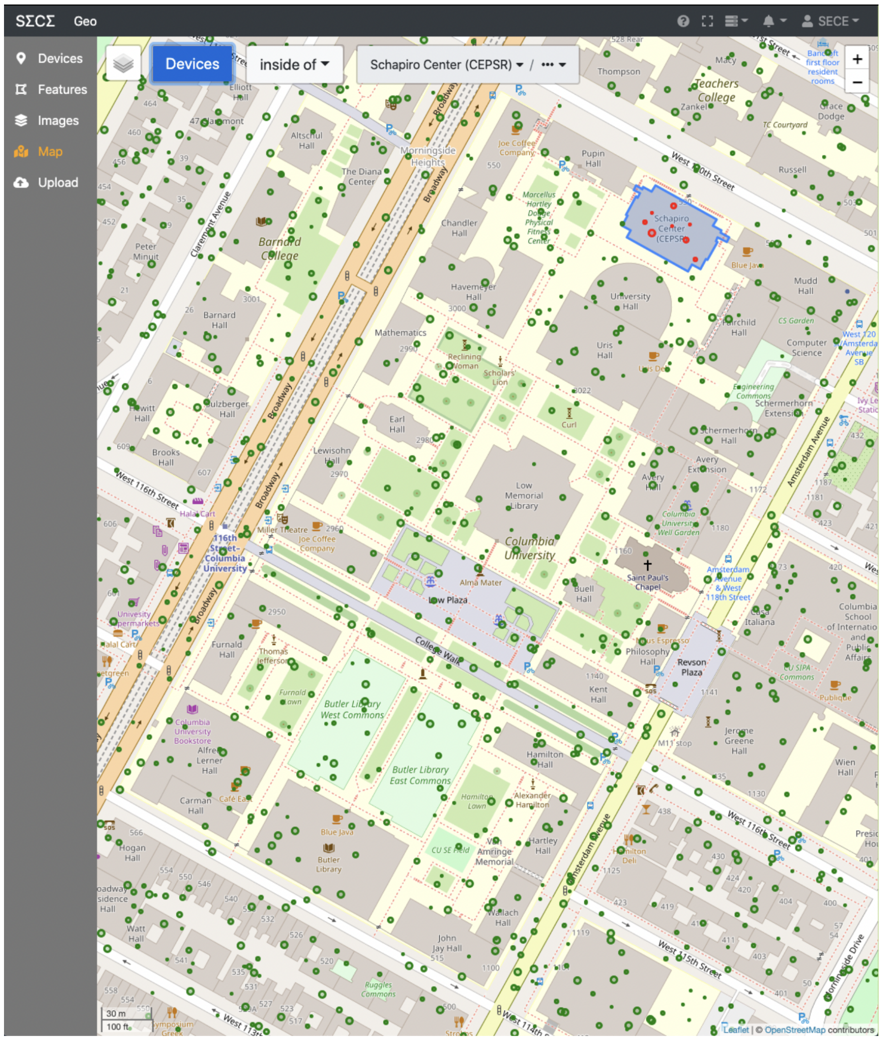


Figure 4.21: Sensors randomly distributed across Columbia University Morningside Heights campus.

and computation.

Sun and Zhou [138] discuss SQL extensions for ad hoc sensor networks. The extensions include operators motivated by the dynamic nature of ad hoc sensor networks.

Bacon et al. [139] discuss the evolution of Google Spanner—a globally distributed, replicated, consistent database—from a key-value store into a relational SQL-based database system. Similar to our work, this paper discusses breaking a SQL query into sub-queries that could be sent to shards.

Tsiftes and Dunkles [140] discuss the design of Antelope, a DBMS designed to run on a constrained device. Their approach complements TinyDB and Courage, as discussed above.

The Location-to-Service Translation (LoST) Protocol [86] and the associated architectural

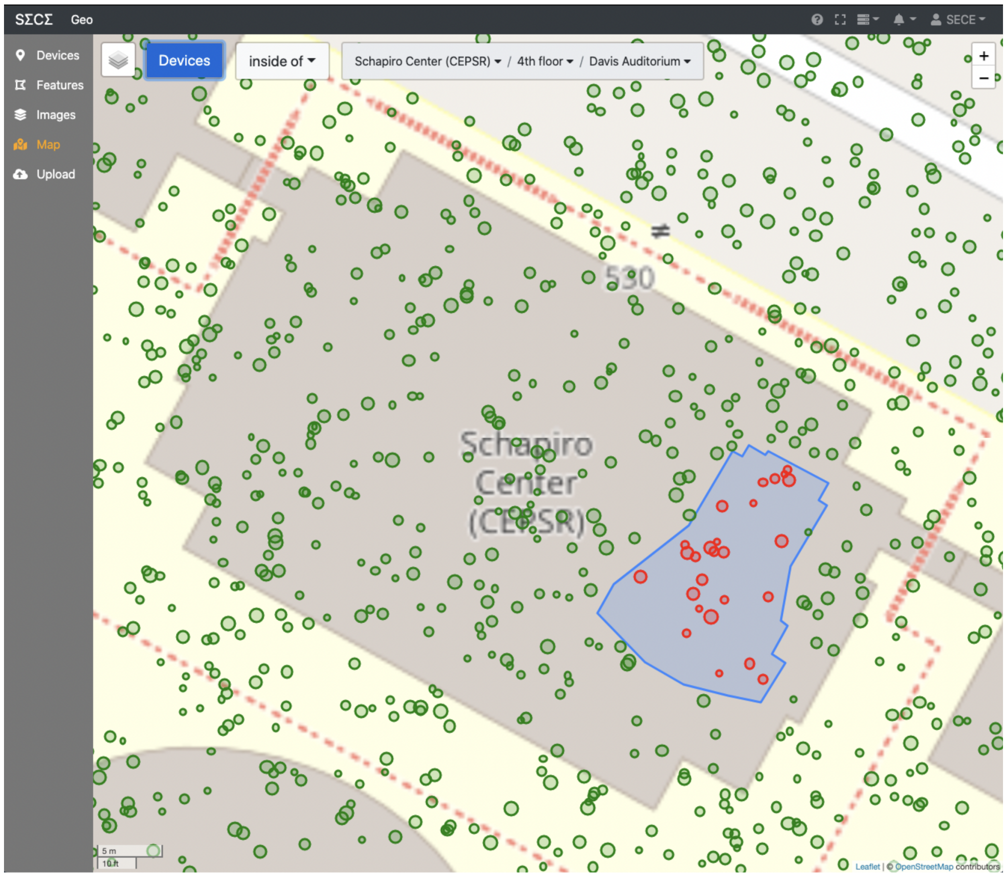

ΣΣCΣ Geo
Devices
Features
Images
Map
Upload
Devices
inside of
Schapiro Center (CEPSR) / 4th floor / Davis Auditorium
SECE
Schapiro Center (CEPSR)
530
5 m
10 ft
Leaflet | © OpenStreetMap contributors

Figure 4.22: The subset of the emulated sensors located in the Davis Auditorium of the Shapiro Engineering Center (CEPSR) on Columbia University campus.

framework [141, 95] served as a starting point for the design and implementation of the Geographic Discovery Service (GD). Sensor Markup Language (SenML) [127] provided inspiration for our spatiotemporal sensor measurement data model.

Cockroach Labs discusses spatial indexing and sharding on their blog [142]. Like our work, CockroachDB divides the target space (geography) among database nodes to provide scalability. Unlike our work, CockroachDB assumes the same administrator manages all nodes and provides no protocol to handle nodes joining or leaving the system.

LoST is an HTTP-XML protocol [86] for resolving (service name, location) pairs to a service instance that serves the location. The protocol was designed for emergency communications with (fixed and mobile) phones as clients but could be useful in cyber-physical systems, where programs often operate on geographically addressed devices. In LoST, the client includes its civic or geographic location in the request, and the server selects a service instance based on a locally configured policy. Modifications to the LoST protocol may be required for CPS applications. Cloud-based programs will likely use the location of the target service in the request. Mobile clients may need a mechanism to limit the rate of LoST requests (e.g., geo-fencing). The service instance selection algorithm, performed by LoST servers, would also need to be clarified.

Dynamic Delegation Discovery System (DDDS) [113, 143, 144, 145] is a DNS-based protocol to map application-specific names to sets of DNS resource records. DDDS is used in DNS ENUM [146] to map telephone numbers to URIs.

Several authors have proposed using flat, identity-based names for routing [147, 148], challenging the traditional wisdom that hierarchically structured names are needed for scalable routing. However, such names are usually random text or binary strings and not meaningful to users or programmers.

Many authors have discussed the shortcomings of DNS, which treats services and data as second-class network citizens [149].

To construct meaningful names for a device, the naming system may need to obtain some information from the device itself. While this is a complex problem in general, existing standardization efforts may help [150, 127].

4.13 Summary

Many cyber-physical applications need to work with a history of sensor data. In a federated and geographically dispersed cyber-physical system, such data may be scattered across large geographic areas in the network edge. A number of sensor database architectures with various sharding, scalability, and reliability tradeoffs have been proposed. We find that most of the available databases are only suitable for cyber-physical systems that are under common administrative control (are not federated) or that are not geographically distributed. Large geographically distributed and federated cyber-physical systems such as the U.S. electrical grid discussed in Chapter 3 require a different approach to prevent single points of failure and scalability bottlenecks.

Applications working with sensor data distributed across a dispersed cyber-physical system must adapt to system architecture changes. Such changes can result from component failures, planned upgrades [12], system evolution and extensions, or malicious activities (attacks) on critical infrastructure. Coping with dynamic system changes places an additional burden on applications. The need to deal with such complexity often results in bugs and errors with potentially dire consequences in systems that interact with physical processes.

In this chapter, we proposed SenSQL, a data storage and query processing network service for spatiotemporal sensor data. The service is specifically designed for large distributed and federated cyber-physical systems. The need for such a service arose during our work on PhoenixSEN [2]. The architecture and operational model of the service are inspired by earlier work on the LoST protocol [86] for mapping the geographic coordinates of emergency callers to the service center responsible for the caller's geographic area.

The SenSQL framework combines an evolved version of the LoST protocol with a SQL-based query processing API in a novel way. We introduced the notion of an autonomous cyber-physical system. Cyber-physical applications submit SQL queries to discover sensor devices and their measurements to an SQL coordinator entity that dynamically discovers subordinate databases at autonomous cyber-physical systems via the LoST protocol and forwards SQL subqueries to the databases.

We also described the data model of a local geographic naming database and the process of populating the database with data from public and private sources. To evaluate the system, we designed an emulated environment to evaluate the SenSQL framework. We described how the environment was used to evaluate the functionality of the SenSQL framework on systems the size and complexity of the Columbia University Morningside Heights campus.

4.13.1 Future Work

The evaluation environment described in Section 4.11 allows functional evaluation only. A more extensive hardware system would be necessary for quantitative scalability and performance evaluation. This is left for future work.

Further work will also be needed to specify a general model for spatiotemporal sensor data. Earlier IETF efforts such as SenML [127] might provide suitable early building blocks for this effort.

Finally, appropriate access control policies applied by autonomous cyber-physical systems will need to be designed, implemented, and tested. This specific task is left for future work beyond the scope of this dissertation.

4.13.2 Acknowledgments

We thank Jasmine Lou, Robert Kim, and Mike Michele Azure, all Columbia University students, for helping to implement the SenSQL framework prototype and evaluation environment.

Chapter 5: Evaluating Usability of Mission-Critical Voice Communications

5.1 Introduction

In the solicitation for the RADICS program discussed in Chapter 3, DARPA highlighted the importance of voice communications (VoIP) as a communication modality for electrical grid restoration during an emergency. Substation personnel use voice communication to coordinate the restoration of various parts of the electrical grid. Thus, the ability of the PhoenixSEN backup network to support voice communications when all other networks, including the public internet, are unavailable was deemed critical. PhoenixSEN may need to run over long-haul or temporary links with unclear bandwidth, delay, jitter, bit error, and packet loss guarantees. The influence of these parameters on the quality of experience (QoE) of the VoIP subsystem needs to be investigated.

NIST's Public Safety Communications Research (PSRC) division has also been investigating the quality of voice communications but in different scenarios and from a slightly different perspective. The PSCR was interested in evaluating QoE in mission critical voice (MCV) communications systems. MCV systems, also known as land-mobile radio (LMR), are used by first responders such as firefighters, disaster responders, police, or emergency services. As the MCV technology slowly transitions from analog FM systems via digital radio to systems based on cellular technology, NIST PSCR seeks to understand the influence of various MCV system parameters and impairments on the quality of experience of its users.

The traditional method to evaluate the usability or quality of experience of human users with technology involves human test subject studies performed in a controlled environment [151]. In such a study, human subject volunteers interact with a particular technology under the supervision of an experimenter. The study generates data that is then evaluated to answer research questions and prove or disprove hypotheses about the technology's suitability for a particular communication model or scenario.

This chapter presents the design and implementation of a testbed for human-subject experiments with mission-critical voice. The testbed was initially designed to support quality-of-experience human-subject studies with MCV in a research project funded by NIST. The testbed could also be used to analyze and optimize the VoIP subsystem in PhoenixSEN under various emulated scenarios.

The research performed with the testbed aims to understand the relationship between QoE and the communication channel quality. In the studies, trained first responders communicate through user terminals provided by the testbed. The testbed emulates various channel conditions by adjusting mouth-to-ear (MtE) delay, push-to-talk (PTT) button delay, codecs, bit error rate, and other communication channel parameters.

The rest of the chapter is organized as follows. Section 5.2 provides examples that motivate the need to design and develop the testbed to support human-subject studies with MCV. In Section 5.3 we cover some of the background material on mission-critical voice and quality of experience measures. Section 5.4 discusses the design of the testbed in detail. In Section 5.5 we describe two types of human-subject studies performed with the testbed. Finally, Section 5.6 summarized the chapter and potential future work.

5.2 Motivation

The critical role of communication in the aftermath of a large-scale disaster event became apparent after Hurricane Katrina in 2005 [152], Hurricane Irene in 2011, and during the 2017 Atlantic hurricane season [153]. Over 11 million people in Puerto Rico lost electricity for 11 days in the largest blackout in U.S. history [154]. These events rendered communication networks across large geographic areas unusable [23, 22].

Significant challenges caused by non-interoperable communication systems were reported [58], limiting situational awareness and preventing communication across jurisdictions and organizations [155, 156].

A 2016 NIST report [157] found communication failures to be a significant obstacle in all recent disaster recovery efforts.

5.3 Background

5.3.1 Quality of Experience Measures

When evaluating quality of experience (QoE) in experiments with volunteer human subjects, we assess the following measures based on the data collected by the testbed.

Comprehension Errors. A comprehension error is the inability of the human test subject to understand the message. The number of comprehension errors can be estimated from the number of retransmitted messages.

Task Errors. A task error occurs when the human test subject performs the wrong action in response to communication. That generally means the test subject records the wrong information in our testbed.

Usage Errors. A usage error represents incorrect use of the testbed during an experiment. In our experiments, usage errors are generally restricted to pressing the PTT button too early or too late.

Length and Latency of Responses. This measure represents the time between a request to perform transmission and the actual start of the transmission.

Subjective Ratings of User Experience. The subjective rating represents the frustration of the volunteer test subject with the radio (emulated by testbed user terminals) operation. This information is collected via a post-experiment form filled out by all study participants.

5.3.2 Half-Duplex PTT Communication Model

All communication in interactive human subject studies is half-duplex. Only one party can communicate at a time. To speak to the remote party, the participant must press and hold a physical button on their speaker-microphone, wait for a beep acknowledging the readiness of the communication infrastructure, and release the button at the end of their transmission. This basic procedure is repeated throughout the experiment. Figure 5.1 shows a flow diagram of all the events in a single half-duplex PTT transmission.

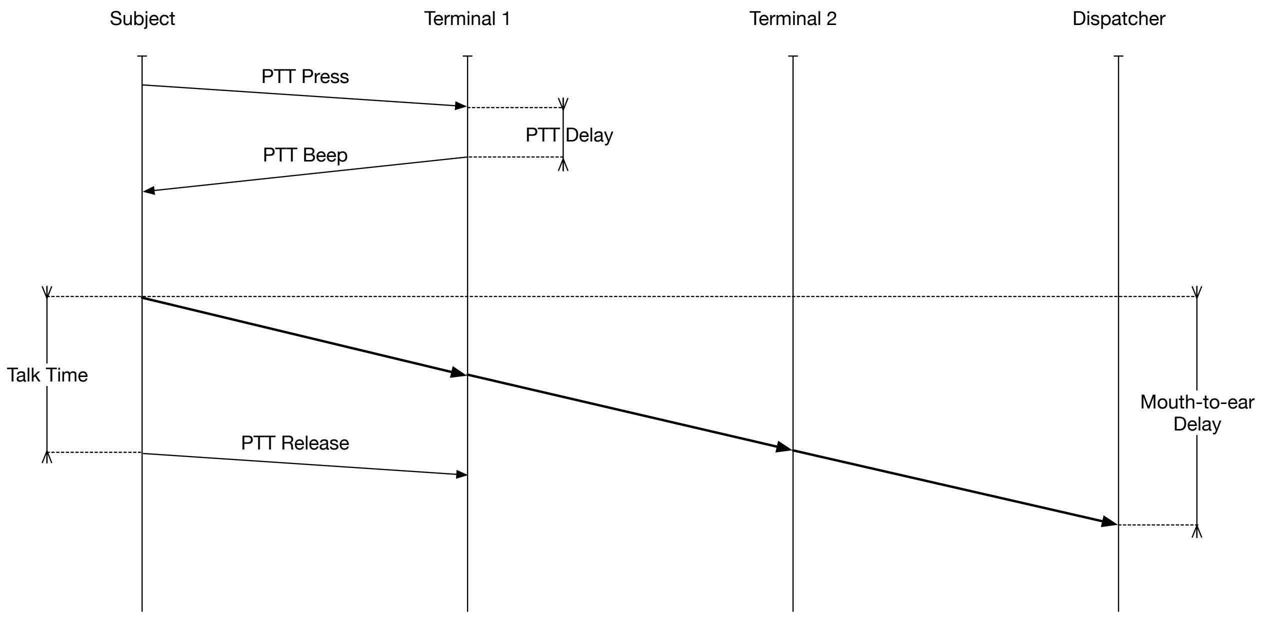


Figure 5.1: A flow diagram of the events within a single half-duplex PTT transmission

The delay between the PTT button press and the beep is known as the PTT delay. This delay represents the time the communication system needs to acquire and reserve the communication channel. In P.25 systems, this time will depend on the parameters of the communication subsystem, load (number of simultaneous sessions), and priority of the talker.

The voice travels from the speaker to the receiver with a mouth-to-ear delay. This delay consists of the time it takes for the audio to travel from the speaker's mouth to the microphone, the time it takes for digital data to travel from one terminal to another, and the time it takes for the audio to travel from the speaker to the receiver's ear. The time between terminals dominates the mouth-to-ear (MtE) delay in real systems.

The PTT and MtE delays are independent. If the talker starts speaking before the PTT beep, the beginning of the communication may be cut off. The total time it takes for the talker to complete the transmission is known as talk time.

5.4 Testbed Architecture

The testbed is designed to be easy to assemble and transport. We expect that there will be a need to move the hardware into a controlled environment designed to conduct human subjects studies, e.g., a quiet room or lab.

5.4.1 Hardware

Figure 5.2 illustrates the overall hardware and network architecture. Figure 5.3 shows an early prototype version built at Columbia University in 2019.

An Intel Next Unit of Computing (NUC), a small form-factor computer powerful enough to run all testbed services, manages the testbed. The NUC serves as a base station. The single Gigabit Ethernet port on the NUC is connected to a Gigabit Ethernet switch. Several user terminals, one per experiment participant, are connected to the Ethernet switch for maximum bandwidth and minimal delay and jitter. The software on the NUC provides a web-based management interface.

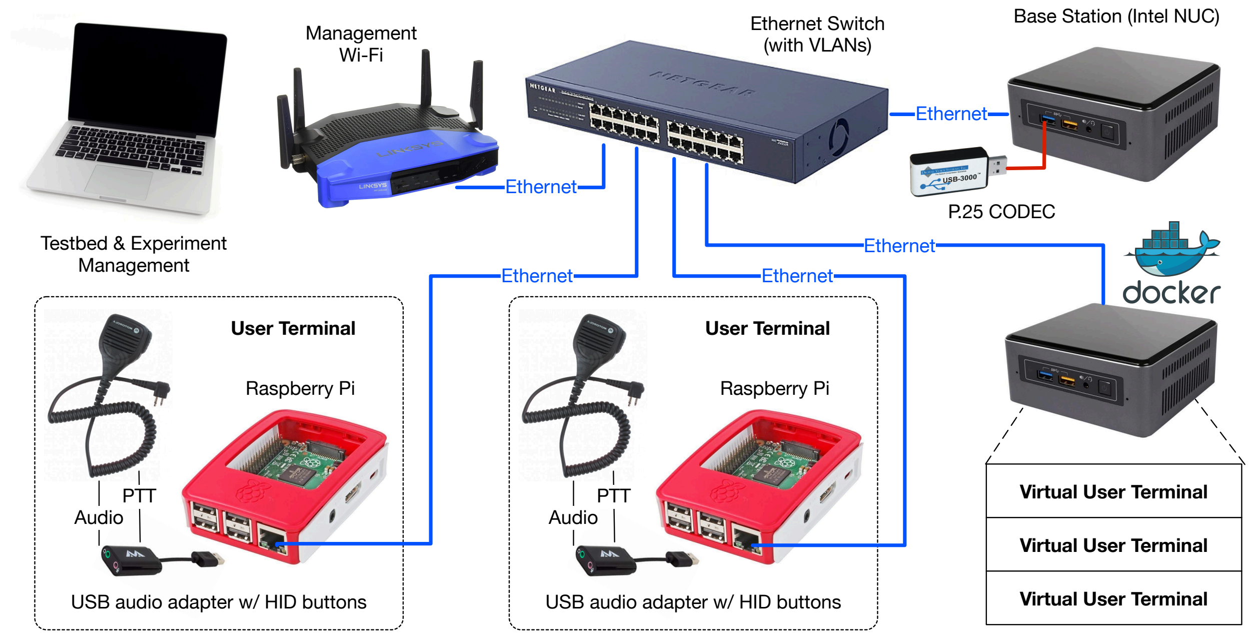


Figure 5.2: Hardware and network architecture of the testbed

The NUC's internal Wi-Fi interface is configured in access point (AP) mode to create a dedicated management Wi-Fi network. The experimenter connects a laptop computer to the management

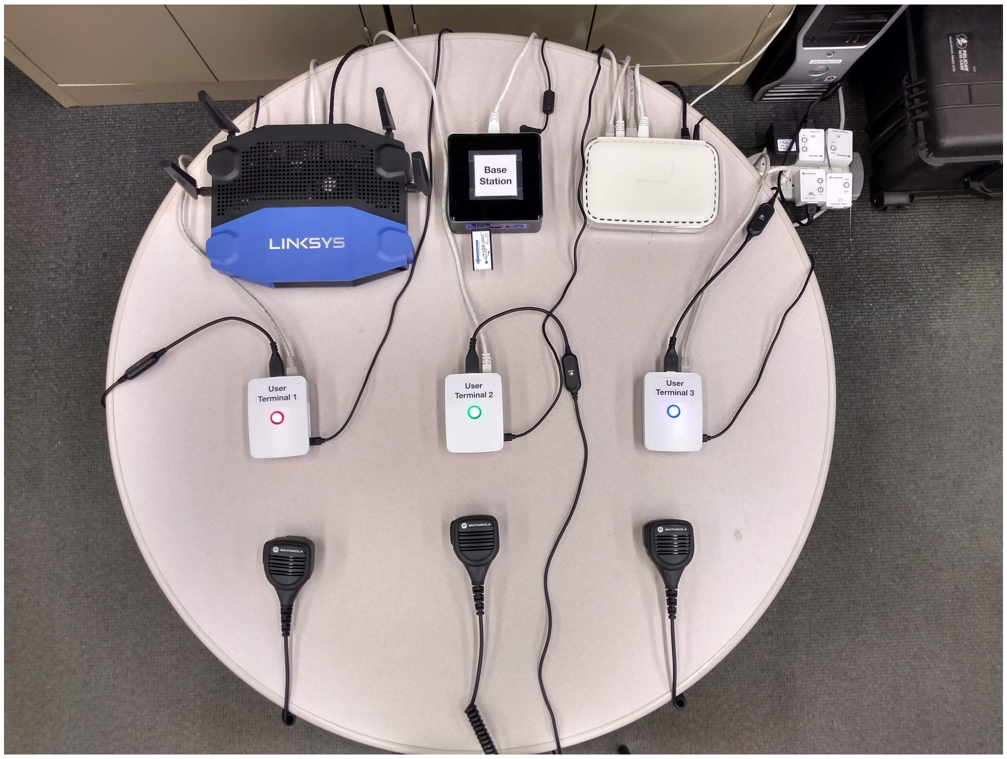


Figure 5.3: A photo of the mission-critical voice testbed built at Columbia University and used in all human-subject studies.

Wi-Fi network. All aspects of the ongoing experiment can be managed through a web-based user interface from the laptop computer. In more complicated scenarios, e.g., experiments conducted over a larger geographic area or multiple rooms, external APs can extend the management Wi-Fi network range.

Raspberry Pi-based user terminals emulate radio communications equipment, i.e., P.25 land-mobile radios. Each user terminal consists of a Raspberry Pi, a USB audio adapter, and an external Motorola speaker-microphone. The terminals communicate with the NUC and other terminals over standard IP-based protocols.

5.4.2 Software

The testbed software consists of components (modules) running on the base station (Intel NUC) and each Raspberry Pi user terminal. The software is a mix of C++, JavaScript, and Python.

5.4.2.1 Base Station

Figure 5.4 shows the software architecture of the base station. The base station is a SIP-based conferencing server with a custom channel emulator module, a web-based user interface, and extensive event logging and audio recording. The base station supports the notion of a virtual RF channel. Each virtual channel is backed by a room on the conference server, identified by a SIP URI with virtual channel “frequency” in the username portion of the URI. To “tune in” to a virtual RF channel, a user terminal calls the virtual channel’s conference room. The conference room mixes and broadcasts media streams from all user terminals in the room.

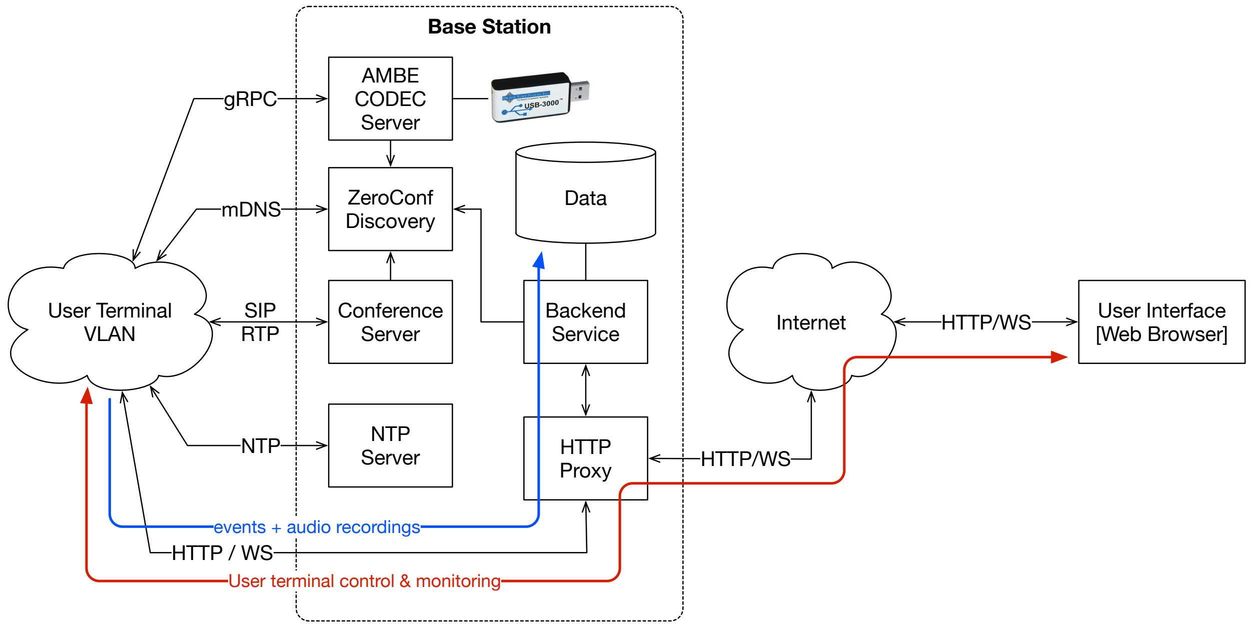
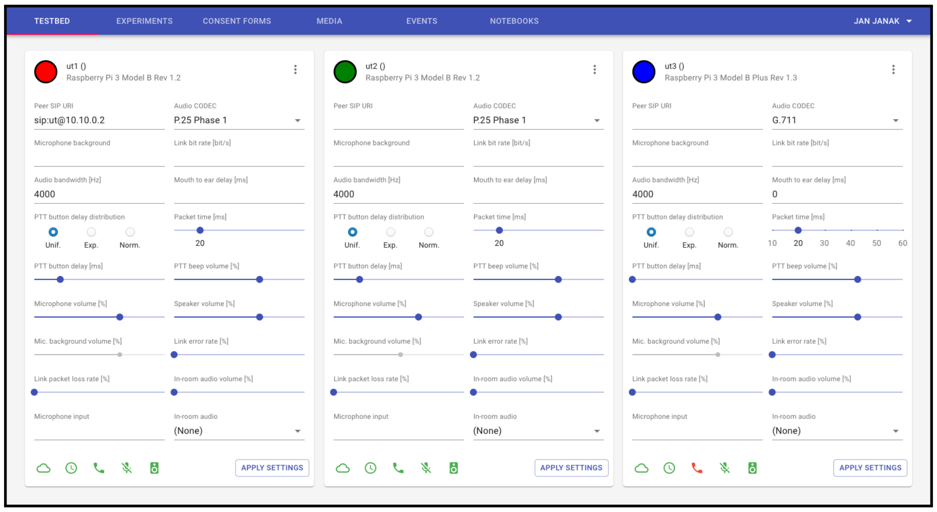


Figure 5.4: Base station software architecture and services

The experimenter manages all aspects of the system through a web-based user interface (UI) provided by the testbed. The UI allows controlling the user terminals in real-time, e.g., to change various parameters during an ongoing experiment. It can also be used to define, manage, and conduct experiments. Figure 5.5 shows a few selected UI components.

The base station provides extensive logging and recording facilities. The logging module records all significant events (terminals joining channels, PTT button presses, and others) that are important for later experiment evaluation. The recording module permanently records all audio streams from

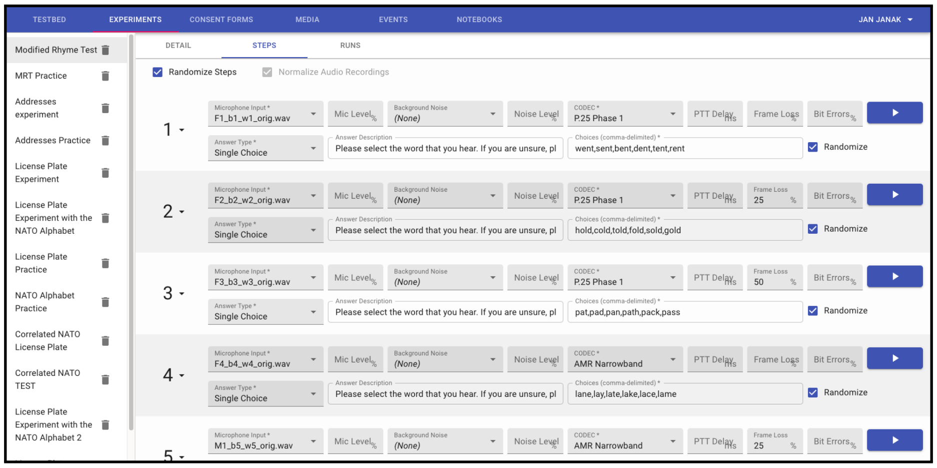


ut1 ()
Raspberry Pi 3 Model B Rev 1.2
Peer SIP URI: sip:ut@10.10.0.2
Audio CODEC: P.25 Phase 1
Microphone background
Link bit rate [bit/s]
Audio bandwidth [Hz]: 4000
Mouth to ear delay [ms]
PTT button delay distribution: [✓ Unif.] Exp. Norm.
Packet time [ms]: 20
PTT button delay [ms]
PTT beep volume [%]
Microphone volume [%]
Speaker volume [%]
Mic. background volume [%]
Link error rate [%]
Link packet loss rate [%]
In-room audio volume [%]
Microphone input
In-room audio: (None)

ut2 ()
Raspberry Pi 3 Model B Rev 1.2
Peer SIP URI
Audio CODEC: P.25 Phase 1
Microphone background
Link bit rate [bit/s]
Audio bandwidth [Hz]: 4000
Mouth to ear delay [ms]
PTT button delay distribution: [✓ Unif.] Exp. Norm.
Packet time [ms]: 20
PTT button delay [ms]
PTT beep volume [%]
Microphone volume [%]
Speaker volume [%]
Mic. background volume [%]
Link error rate [%]
Link packet loss rate [%]
In-room audio volume [%]
Microphone input
In-room audio: (None)

ut3 ()
Raspberry Pi 3 Model B Plus Rev 1.3
Peer SIP URI
Audio CODEC: G.711
Microphone background
Link bit rate [bit/s]
Audio bandwidth [Hz]: 4000
Mouth to ear delay [ms]: 0
PTT button delay distribution: [✓ Unif.] Exp. Norm.
Packet time [ms]: 10 [✓ 20] 30 40 50 60
PTT button delay [ms]
PTT beep volume [%]
Microphone volume [%]
Speaker volume [%]
Mic. background volume [%]
Link error rate [%]
Link packet loss rate [%]
In-room audio volume [%]
Microphone input
In-room audio: (None)

(a) UI for real-time user terminal control during interactive experiments

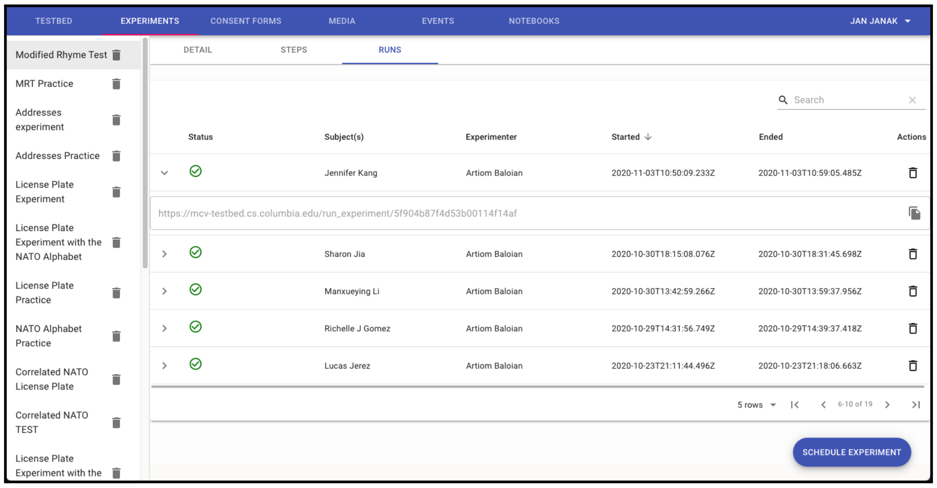


Modified Rhyme Test
MRT Practice
Addresses experiment
Addresses Practice
License Plate Experiment
License Plate Experiment with the NATO Alphabet
License Plate Practice
NATO Alphabet Practice
Correlated NATO License Plate
Correlated NATO TEST
License Plate Experiment with the NATO Alphabet 2

☑ Randomize Steps ☑ Normalize Audio Recordings

Step	Microphone Input *	Mic Level %	Background Noise	Noise Level %	CODEC *	PTT Delay ms	Frame Loss %	Bit Errors %	Answer Type *	Answer Description	Choices (comma-delimited) *	Randomize
1	F1_b1_w1_orig.wav		(None)		P.25 Phase 1				Single Choice	Please select the word that you hear. If you are unsure, pl	went,sent,bent,dent,tent,rent	✓
2	F2_b2_w2_orig.wav		(None)		P.25 Phase 1		25		Single Choice	Please select the word that you hear. If you are unsure, pl	hold,cold,told,fold,sold,gold	✓
3	F3_b3_w3_orig.wav		(None)		P.25 Phase 1		50		Single Choice	Please select the word that you hear. If you are unsure, pl	pat,pad,pan,path,pack,pass	✓
4	F4_b4_w4_orig.wav		(None)		AMR Narrowband				Single Choice	Please select the word that you hear. If you are unsure, pl	lane,lay,late,lake,lace,lame	✓
5	M1_b5_w5_orig.wav		(None)		AMR Narrowband		25					

(b) UI for managing listening experiments



Modified Rhyme Test
MRT Practice
Addresses experiment
Addresses Practice
License Plate Experiment
License Plate Experiment with the NATO Alphabet
License Plate Practice
NATO Alphabet Practice
Correlated NATO License Plate
Correlated NATO TEST
License Plate Experiment with the

Status	Subject(s)	Experimenter	Started	Ended	Actions
✓	Jennifer Kang	Artiom Baloian	2020-11-03T10:50:09.233Z	2020-11-03T10:59:05.485Z	
https://mcv-testbed.cs.columbia.edu/run_experiment/5f904b87f4d53b00114f14af					
✓	Sharon Jia	Artiom Baloian	2020-10-30T18:15:08.076Z	2020-10-30T18:31:45.698Z	
✓	Manxueying Li	Artiom Baloian	2020-10-30T13:42:59.266Z	2020-10-30T13:59:37.956Z	
✓	Richelle J Gomez	Artiom Baloian	2020-10-29T14:31:56.749Z	2020-10-29T14:39:37.418Z	
✓	Lucas Jerez	Artiom Baloian	2020-10-23T21:11:44.496Z	2020-10-23T21:18:06.663Z	

5 rows 6-10 of 19

(c) UI for managing human-subject studies

Figure 5.5: Examples of the web-based UI offered by the MCV testbed.

all conference rooms (virtual channels) and user terminals. The two modules allow replaying the activity on any given channel based on high-precision timestamps embedded in the stored data.

5.4.2.2 User Terminal

Figure 5.6 shows the essential building blocks of the user terminal software. The terminals are centrally managed from the web-based user interface provided by the base station.

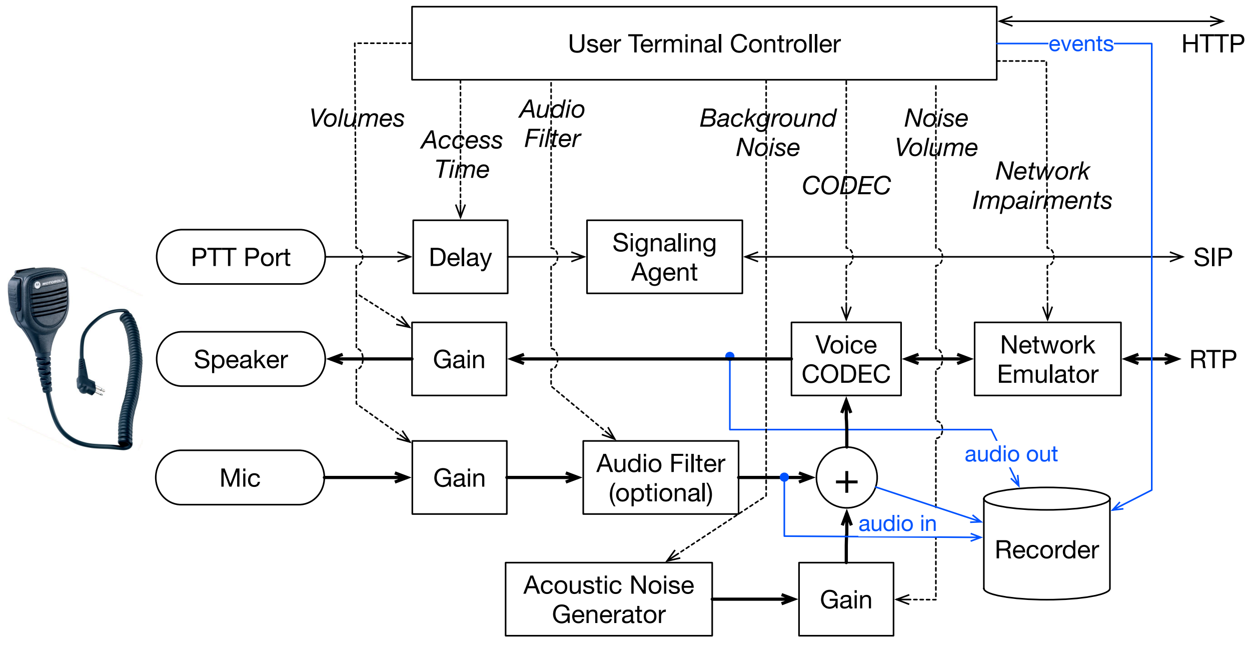


Figure 5.6: User terminal software architecture

All user terminal subsystems are managed by a user terminal controller program implemented in TypeScript compiled to JavaScript running in NodeJS. The controller provides an HTTP-based API to the user interface component running on the NUC. The API exposes internal variables that control the operation of the terminal. Upon startup, the controller configures a VoIP client to use a particular codec and to call a SIP URI representing a virtual channel. On configuration changes, the controller reconfigures and restarts the VoIP client.

The user terminal runs a modified VoIP client based on Baresip [158], a simple embedded SIP user agent. The client uses the speaker-microphone as its audio input-output. We made the following modifications to Baresip to make it suitable for the testbed:

- added support for push-to-talk (PTT) functionality with customizable PTT button delay;
- added a custom acoustic noise generator that can be mixed with microphone input;
- added manual control for audio gain;
- implemented support for the AMBE2+ codec (via a USB dongle) used in P.25 systems.

The user terminal communicates with the base station and other terminals via standards-based SIP [66] (signaling) and RTP [159] (media) protocols. Upon startup, the user terminal joins a virtual channel identified by a SIP URI, which maps to a room on the base station conference server. To "tune in" on a virtual channel, the user terminal sends a SIP INVITE message to the conference room URI.

Typically, SIP-based VoIP clients negotiate the codec to be used at the beginning of the communication session. The SIP UA in the user terminal is restricted to using a single codec. Eliminating codec selection helps emulate a particular type of system, e.g., AMBE2+ for P.25, G.711 for analog FM, or AMR for IP-based systems.

A delay block inserted between the PTT button and Baresip controls the time it takes for Baresip to register changes in the state of the PTT button. This block provides a means to evaluate the influence of variable PTT button delay on QoE. In real-world communication systems, activating the transmitter after a PTT button press can take a long time, negatively affecting the transmitted message's intelligibility. The input to the delay block is a signal representing the state of the PTT button over time. The output of the delay block is the input signal delayed by a configured time interval (e.g., 500 ms). The controller block configures the interval at run-time.

The user terminal implements a virtual microphone device that mixes microphone input with optional background noise read from a wave file. The wave file provides pre-recorded background noise, e.g., white or pink noise, street noise, ambulance sirens, and many others. Experimenters upload wave files with the background noise to the base station through its web-based user interface. User terminals download the wave file before the experiment and mix its contents into the microphone input.

5.4.2.3 Channel Emulator

The user terminals and the base station communicate over Gigabit Ethernet with very high bandwidth, low delay and jitter, and virtually no packet loss. The network is considerably over-provisioned compared to most mission-critical communications systems. We take advantage of the over-provisioning and provide a channel emulator module as part of the testbed for various RF environments commonly found in mission-critical communications systems. The emulator emulates the effects of varying RF channel conditions in software. That way, we can study the effects of various impairments on the quality of experience (QoE).

The channel emulator has built-in modes for the most common types of deployed MCV systems:

- P.25 system (4800 Baud, C4FM-modulated);
- Analog FM (12.5 kHz analog FM);
- Digital IP system (IP, RTP, RoHC, AMR codecs).

In addition to emulating the various parameters of the RF channel, the channel emulator also includes a configurable error model to generate bit errors at a given rate and distribution. The error model can emulate the effects of interference or channel fading. Figure 5.7 shows a possible configuration for P.25 systems.

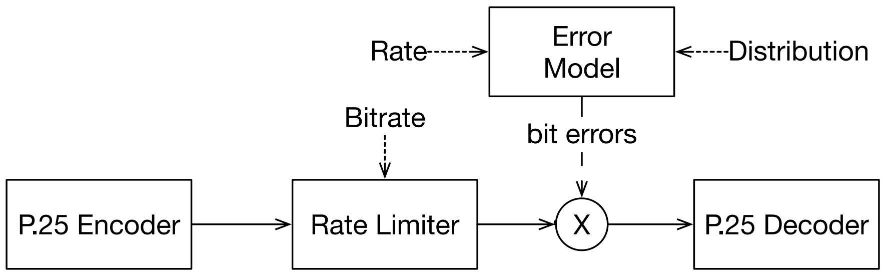


Figure 5.7: P.25 rate limiter and error injector

An RTP stream from a user terminal is fed into a P.25 frame encoder, which converts the RTP stream into a stream of P.25 frames. We assume the incoming RTP stream will be AMBE2+ encoded by the user terminal but will not include the P.25 frame header. The encoder adds the header and turns incoming RTP packets into proper P.25 frames. The stream of P.25 frames is then fed into a rate limiter.

The rate limiter provides a virtual serial link with parameters matching P.25, i.e., 4800 Baud, C4FM-modulated. The rate-limited bit stream is mixed with an error bit stream generated by the error model block before it is fed into a P.25 decoder. The P.25 decoder block generates another RTP stream (FEC error corrected) suitable for processing by the conference server (which supports AMBE2+).

There are several ways to implement the channel emulator block, with varying levels of fidelity. In the prototype, we focused on configuring various parameters of the RF channel. The parameters include bandwidth, delay, packet loss, and error rate. We aimed to have a flexible and configurable system that would not necessarily provide high-fidelity emulation. We implemented the emulator with Linux traffic control (tc) and the Netem network emulation tool. These subsystems only implement a subset of the functionality. The rest would need to be implemented as custom user-space processes.

5.4.3 Evaluation Environment

Each interactive or listening experiment run performed with the testbed produces a wealth of data. The data includes time-synchronized events, uncompressed audio recordings made by the user terminals, and any input entered by the test subjects into the web UI for listening experiments. At the end of an experiment, the base station collects the data from all devices and stores the time-synchronized events on a local MongoDB database and audio recordings in a designated directory on the local file system.

The testbed provides an environment based on JupyterHub [160] for the experimenters to analyze the data collected during the experiment. The experimenters can access the environment from their web browser. The environment is integrated with the Google OAuth2 authentication system. Authentication is necessary to restrict access to potentially sensitive data, subject to Institutional Review Board (IRB) conditions. Figure 5.8 illustrates the architecture of the environment.

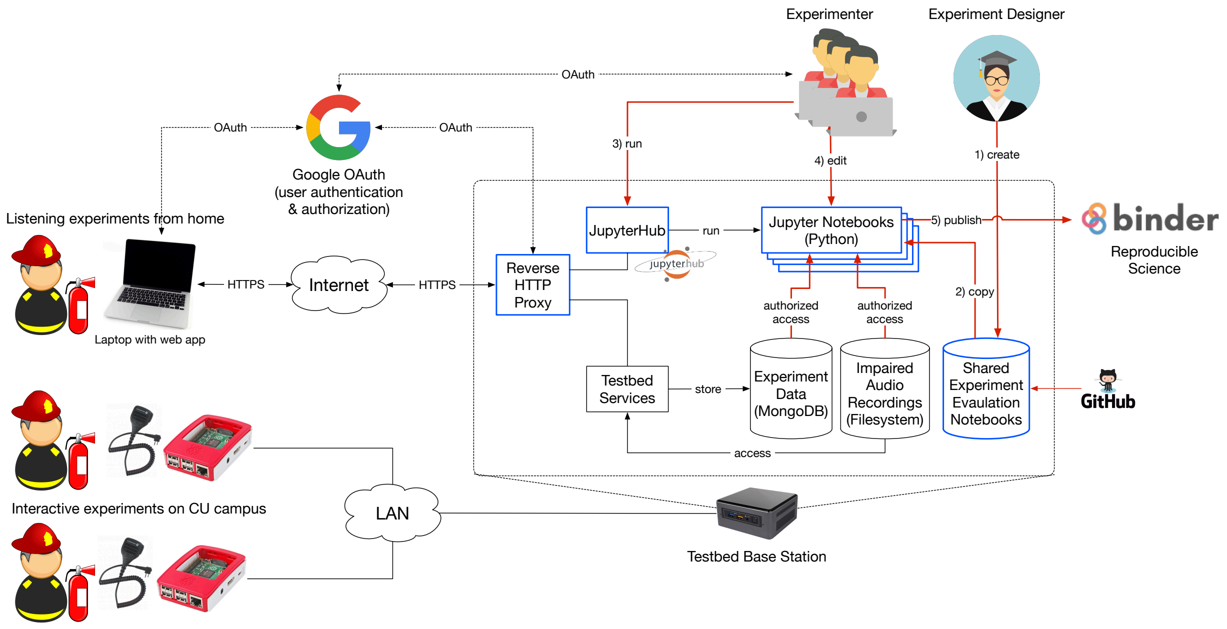


Figure 5.8: The testbed provides an evaluation environment for experiments based on JupyterHub

JupyterHub is a web-based environment for interactive computing in Python. JupyterHub programs are organized into notebooks. Each notebook consists of a series of steps that can be executed interactively. The ability to generate graphs directly within the notebook makes JupyterHub popular in the scientific community. The environment gives MCV experimenters the ability to:

- Interactively explore and analyze data collected during experiments;
- Run Python notebooks on an experiment repeatedly and iteratively, e.g., as more experiment runs are performed;
- Give the notebooks direct access to the sensitive data collected during the experiment, including time-synchronized events and audio recordings.

To integrate JupyterHub with the Google OAuth2 authentication system, we run it behind a reverse HTTP proxy and use an authentication middleware that authenticates the user through an HTTP header. Each experimenter is a verified JupyterHub user identified by the `X-Authorized-User` HTTP header value.

Testbed users (experimenters) do not necessarily have to have operating system accounts on the base station. The JupyterHub environment spawns notebooks in dynamically created Docker containers to isolate Python notebooks from one another and the rest of the system. The environment automatically provisions the Docker container to access the MongoDB database and the audio recordings stored in the local file system. Any notebooks created by the user within the Docker container are persistently saved to the local file system in the user's dedicated folder.

Figure 5.9 show an example Python notebook that analyzes a whole set of listening experiment runs of a particular experiment and produces a statistical analysis. Figure 5.10 shows an example Python notebook that analyzes audio recordings collected during an experiment.

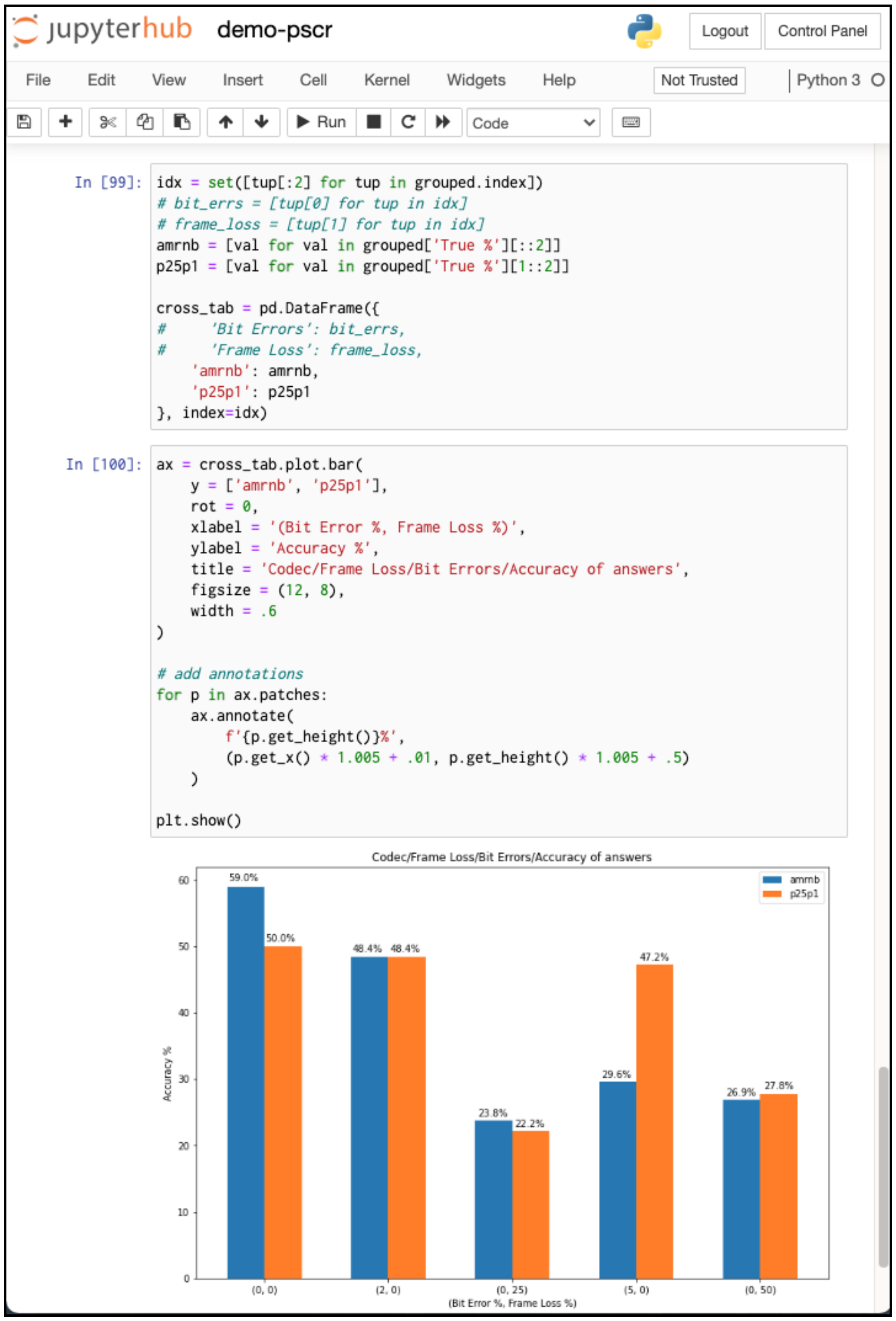
5.5 Human Subject Experiments

The MCV testbed was initially designed for interactive human-subject experiments where volunteer first-responder human subjects communicate through user terminals emulating MCV communication equipment. While developing the testbed, we also added support for listening experiments where human subjects listen to a series of impaired audio recordings in a web browser. This section discusses both experiment types and illustrates how the testbed was used for both.

5.5.1 Listening Experiments

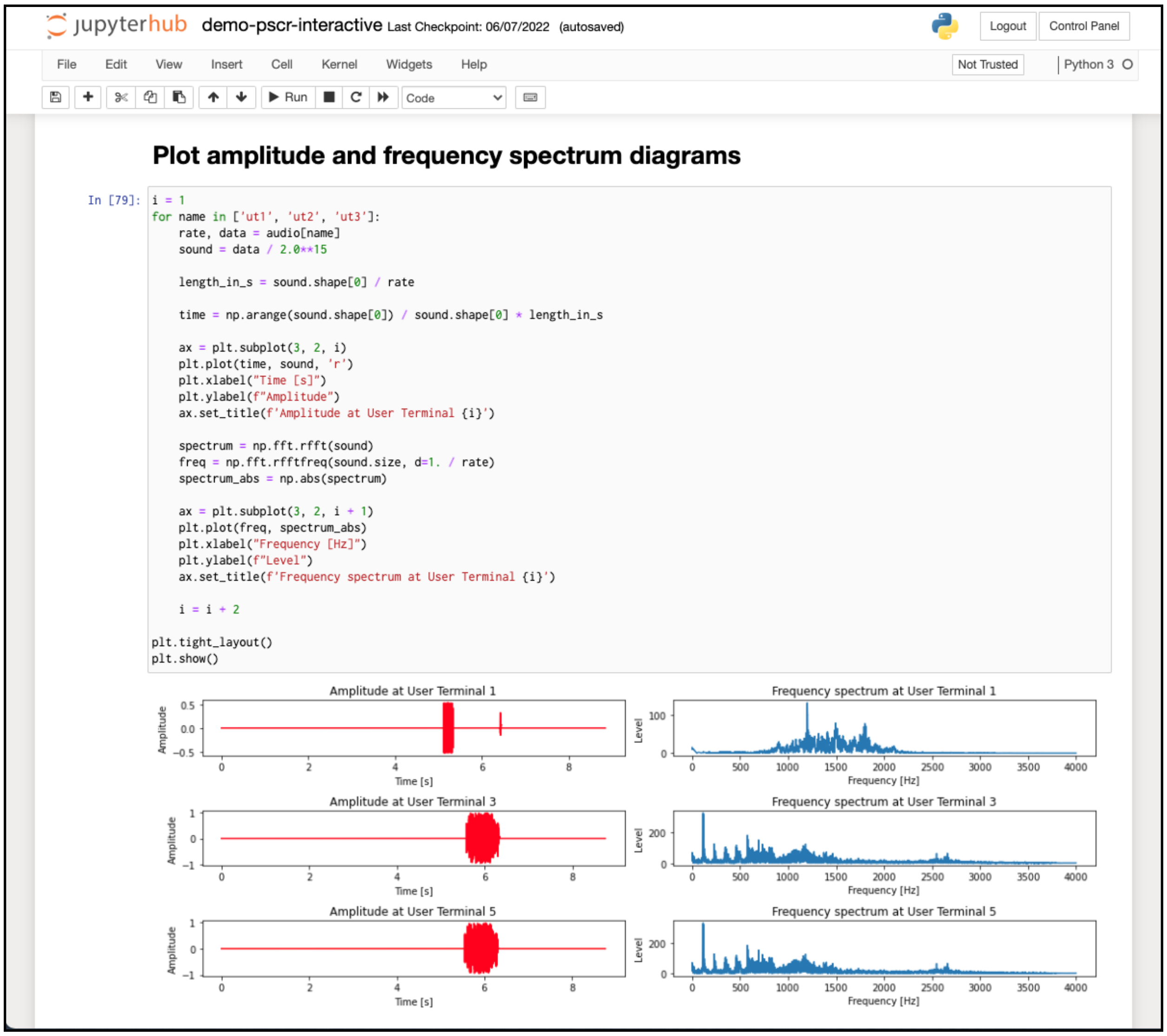
The testbed can be used to design, conduct, and evaluate listening experiments. In a listening experiment, the human test subject listens through a series of impaired audio recordings in a web browser. The audio recordings are transcoded with one of the codecs used in MCV communication systems. The data is impaired with bit errors, frame drops, background noise, and other types of impairments commonly found in deployed mission-critical voice communications systems. Figure 5.11 show the web-based user interface presented to test subjects.

The test subject is presented with a series of steps, one at a time. Pressing the Play button starts playing the impaired audio file. Each audio file can only be played once and cannot be restarted to prevent cheating. The test subject is expected to extract some information from the audio recording and enter the information as an answer into the field below the Play button. The answer can be a free-form text answer, single-choice, or multiple-choice. The Next button takes the test subject to



```
In [99]: idx = set([tup[:2] for tup in grouped.index])
         # bit_errs = [tup[0] for tup in idx]
         # frame_loss = [tup[1] for tup in idx]
         amrnb = [val for val in grouped['True %'][::2]]
         p25p1 = [val for val in grouped['True %'][1::2]]

         cross_tab = pd.DataFrame({
         #     'Bit Errors': bit_errs,
         #     'Frame Loss': frame_loss,
             'amrnb': amrnb,
             'p25p1': p25p1
         }, index=idx)

In [100]: ax = cross_tab.plot.bar(
              y = ['amrnb', 'p25p1'],
              rot = 0,
              xlabel = '(Bit Error %, Frame Loss %)',
              ylabel = 'Accuracy %',
              title = 'Codec/Frame Loss/Bit Errors/Accuracy of answers',
              figsize = (12, 8),
              width = .6
          )

          # add annotations
          for p in ax.patches:
              ax.annotate(
                  f'{p.get_height()}%',
                  (p.get_x() * 1.005 + .01, p.get_height() * 1.005 + .5)
              )

          plt.show()
```

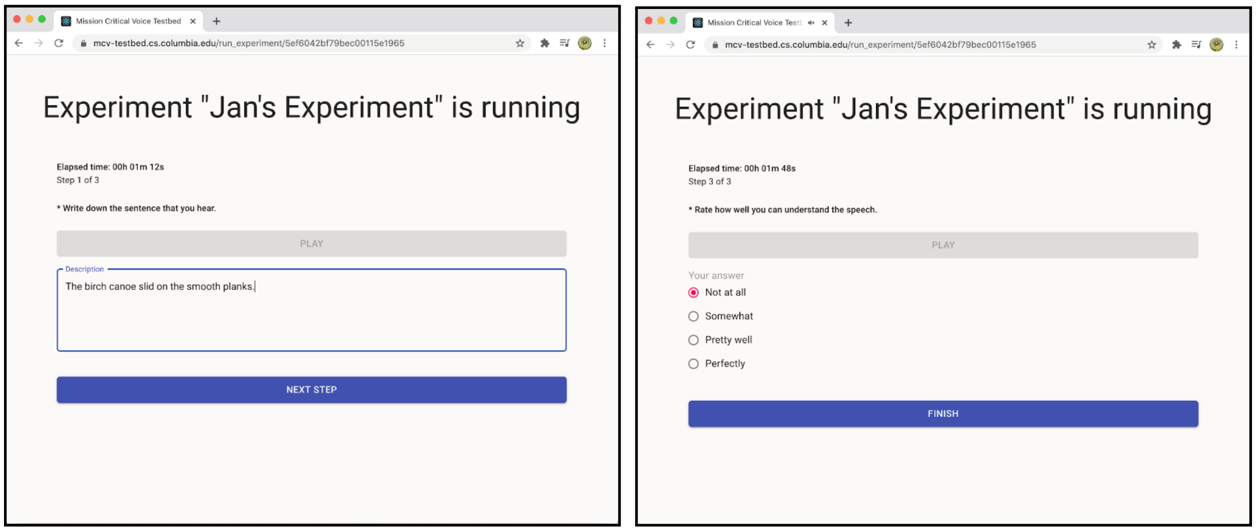
Figure 5.9: The MCV testbed offers an integrated JupyterLab (Python) environment for post-experiment evaluation. The environment (Jupyter notebook) has direct access to the data collected by user terminals and the base station during the experiment. This example shows a notebook analyzing a listening experiment. The plot shows the ratio of correct answers as a function of bit errors and frame drops for two codecs.


jupyterhub
demo-pscr-interactive Last Checkpoint: 06/07/2022 (autosaved)
Logout
Control Panel
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Not Trusted
Python 3
Run
Code
Amplitude at User Terminal 1
Frequency spectrum at User Terminal 1
Amplitude at User Terminal 3
Frequency spectrum at User Terminal 3
Amplitude at User Terminal 5
Frequency spectrum at User Terminal 5
Amplitude
Level
Time [s]
Frequency [Hz]

Plot amplitude and frequency spectrum diagrams

```
In [79]: i = 1
for name in ['ut1', 'ut2', 'ut3']:
    rate, data = audio[name]
    sound = data / 2.0**15

    length_in_s = sound.shape[0] / rate

    time = np.arange(sound.shape[0]) / sound.shape[0] * length_in_s

    ax = plt.subplot(3, 2, i)
    plt.plot(time, sound, 'r')
    plt.xlabel("Time [s]")
    plt.ylabel(f"Amplitude")
    ax.set_title(f'Amplitude at User Terminal {i}')

    spectrum = np.fft.rfft(sound)
    freq = np.fft.rfftfreq(sound.size, d=1. / rate)
    spectrum_abs = np.abs(spectrum)

    ax = plt.subplot(3, 2, i + 1)
    plt.plot(freq, spectrum_abs)
    plt.xlabel("Frequency [Hz]")
    plt.ylabel(f"Level")
    ax.set_title(f'Frequency spectrum at User Terminal {i}')

    i = i + 2

plt.tight_layout()
plt.show()
```

Figure 5.10: The MCV testbed offers an integrated JupyterLab (Python) environment for post-experiment evaluation. The environment (Jupyter notebook) has direct access to the data collected by user terminals and the base station during the experiment. The data includes time-synchronized events and audio recordings. Thus, the experimenter can analyze both events (e.g., PTT button activity) and audio. This example shows a notebook plotting volume and frequency distribution diagrams for microphone inputs on three user terminals.


Mission Critical Voice Testbed
mcv-testbed.cs.columbia.edu/run_experiment/5ef6042bf79bec00115e1965

Experiment "Jan's Experiment" is running

Elapsed time: 00h 01m 12s
Step 1 of 3

* Write down the sentence that you hear.

PLAY

Description
The birch canoe slid on the smooth planks.

NEXT STEP

Mission Critical Voice Testbed
mcv-testbed.cs.columbia.edu/run_experiment/5ef6042bf79bec00115e1965

Experiment "Jan's Experiment" is running

Elapsed time: 00h 01m 48s
Step 3 of 3

* Rate how well you can understand the speech.

PLAY

Your answer
[✓ Not at all]
Somewhat
Pretty well
Perfectly

FINISH

Figure 5.11: A web-based user interface for human subject listening experiments

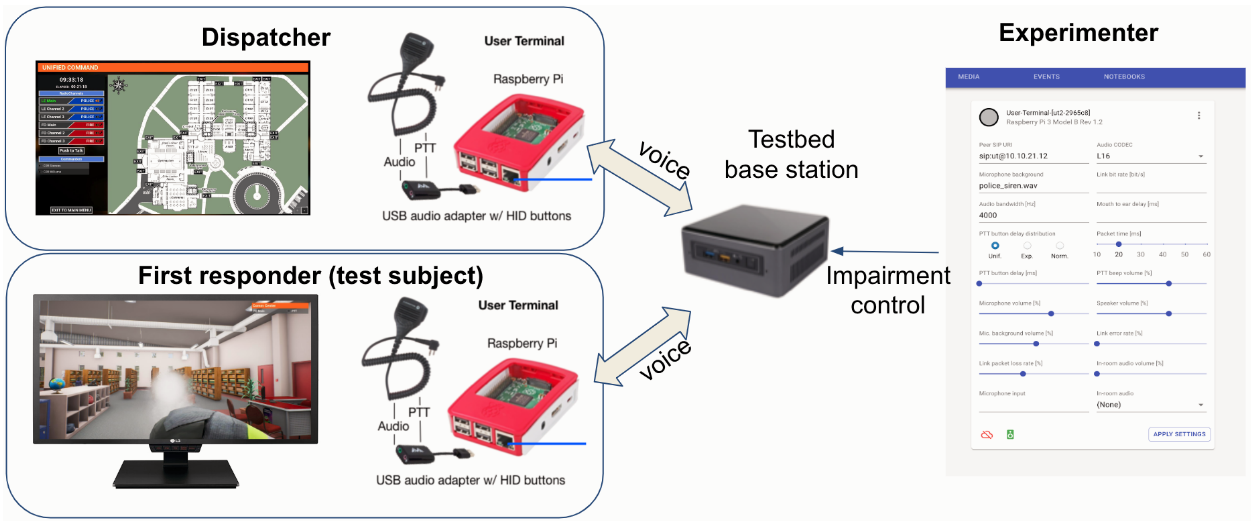
the following audio recording. This basic procedure is repeated for each audio recording until the end of the experiment run.

5.5.2 Interactive Experiments

The Enhanced Dynamic Geo-Social Environment (EDGE) [161] is a 3D virtual training environment for first responders built on the Unreal gaming engine. EDGE can be used to train a coordinated response to incidents in a simulated school scenario. Push-to-talk (PTT) communication is the primary means for coordinating the response among first responders.

To investigate the effect of audio impairments and delays on the quality of PTT communication, we paired the EDGE environment with the MCV testbed, as shown in Figure 5.12. The testbed takes over all communication between the first responder and the dispatcher as they complete a simulated navigation scenario in EDGE. The testbed was programmed to impair audio and introduce delays in the communication system. Figure 5.13 shows a team member during an experiment trial run, navigating the EDGE platform and communicating with an off-screen dispatcher.

Human test subjects recruited from the Columbia University student population assumed the role of a first responder responding to a threat in the school. Their goal was to locate a suspicious device (a pipe bomb) in one of the school rooms systematically and as quickly as possible while



User-Terminal-[ut2-2965c8]
Raspberry Pi 3 Model B Rev 1.2

Peer SIP URI: sip:ut@10.10.21.12
Audio CODEC: L16
Microphone background: police_siren.wav
Link bit rate [bit/s]
Audio bandwidth [Hz]: 4000
Mouth to ear delay [ms]
PTT button delay distribution: [✓ Unif.] Exp. Norm.
Packet time [ms]: 10 [✓ 20] 30 40 50 60
PTT button delay [ms]
PTT beep volume [%]
Microphone volume [%]
Speaker volume [%]
Mic. background volume [%]
Link error rate [%]
Link packet loss rate [%]
In-room audio volume [%]
Microphone input
In-room audio: (None)

Figure 5.12: A combination of the Enhanced Geo-Social Environment (EDGE) with the MCV testbed used in interactive human-subject experiments.

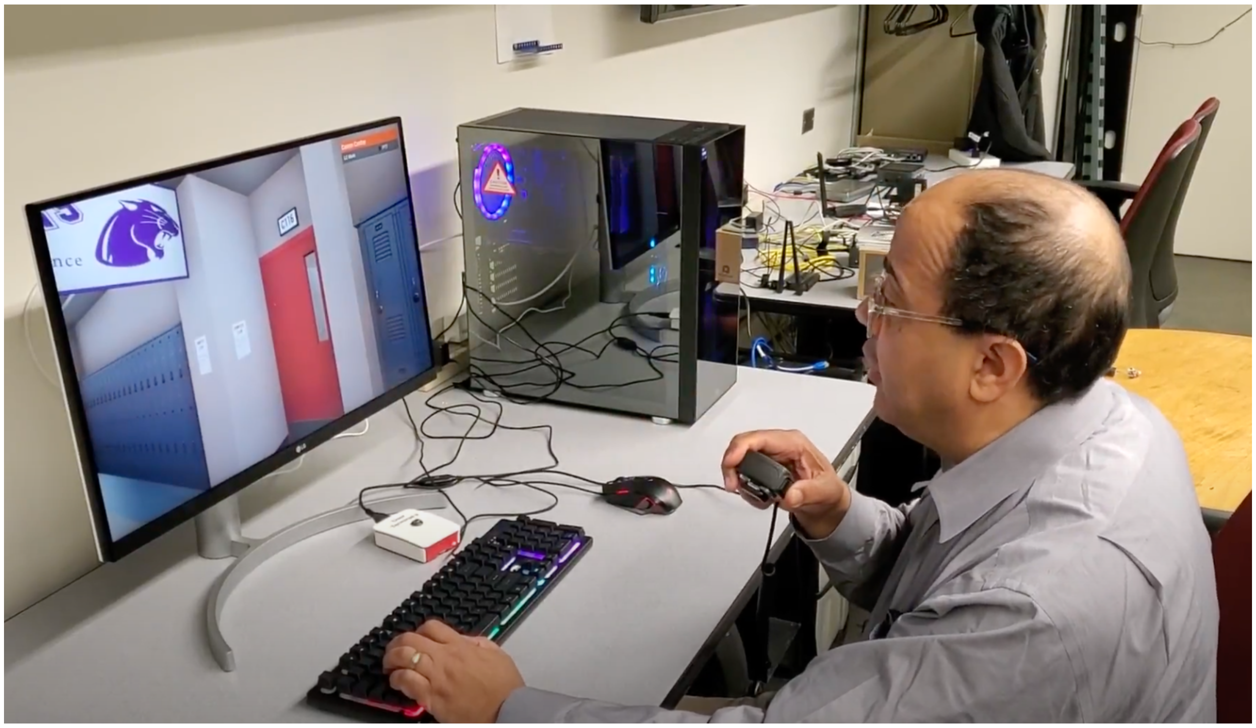


Figure 5.13: A principal investigator on our research team communicates through the testbed during a practice experiment run. In the experiment, the test subject navigates a virtual school environment implemented using the EDGE platform based on instructions from a remote dispatcher received over the emulated radio. The photo was taken on April 21, 2022. Used with permission.

coordinating with a remote dispatcher.

The experimenter uses the testbed's web-based console to set various communication parameters, impairing the communication between the test subject and the dispatcher. We focused on delay-related parameters such as mouth-to-ear (MtE) and push-to-talk (PTT) delays. We investigated the influence of MtE and PTT delays on the quality of the communication between the test subject and dispatcher by evaluating the completion time. During the experiment, the testbed recorded the precise timestamps of all interactions with the user terminals, including PTT button press, PTT beep start and end, and PTT release.

5.6 Summary

High-quality voice communication can be critical, especially in emergencies. This was recognized by DARPA, which stressed the need for high-quality VoIP communication in its requirements for a backup network for the U.S. electrical grid. In another context, NIST's Public Safety Communications Research division has also investigated the quality of experience in LMR mission-critical communications systems.

This chapter presented the design and implementation of a testbed for human-subject experimental research in MCV communications. It illustrated how the testbed was used to conduct listening and interactive experiments with human-subject volunteers at Columbia University.

The testbed uses open hardware and software and can easily be transported into a controlled environment. An integrated web-based interface provides experimenters with features to design listening and interactive experiments, manage individual devices, and evaluate the experiment in an integrated JupyterHub environment. The software developed for the testbed has been released as open source [25].

5.6.1 Future Work

In a future testbed version, we may consider implementing the channel emulator blocks using one of the existing software-defined radio (SDR) frameworks, e.g., GNU Radio. This approach has a significant advantage; frameworks like GNU Radio contain a large number of reusable building blocks that could be used to implement various blocks in the diagram shown in Figure 5.7. These blocks include P.25 encoders and decoders, signal generators, mixers, noise generators, and various other supporting blocks that could be used to create a more realistic (higher fidelity) RF channel emulator in software, including proper emulation of radio interference and fading.

5.6.2 Acknowledgments

We thank Artiom Baloian, Kahlil Dozier, Daniel Rubenstein, Lauren Berny, Charles Jennings, Shaun Maher, and Michael Paciullo for help with the design and implementation of the testbed. We also thank the members of the NIST PSCR team for many helpful suggestions and fruitful discussions.

Chapter 6: An Analysis of Amazon Echo's Enrollment Protocol

This chapter presents an analysis of Amazon Echo's enrollment (also known as network pairing) protocol. We chose the Echo for its privacy-sensitive nature and a substantial installed base. This work aims to illustrate the proprietary nature of IoT device configuration protocols. Furthermore, some of the shortcomings in the Echo's pairing protocol, namely the inability to properly authenticate the device, motivate the work discussed in Chapter 8.

6.1 Introduction

With over 20 million units sold since 2015 [162], Amazon Echo, the Alexa-enabled smart speaker developed and sold by Amazon, is probably one of the most widely deployed IoT consumer devices. The Echo has found its way to many homes [163], high school classrooms [164], and some hotels [165]. Despite the large installed base, surprisingly little is known about the device's network protocols and behavior.

The device's network enrollment protocol, also known as the Out-of-the-Box Experience (OOBE), is particularly interesting. The protocol is executed between a newly purchased Amazon Echo device, an Alexa application running on a smartphone, and an Amazon cloud. The enrollment protocol configures Wi-Fi network credentials into the device and associates the device with the user's Amazon account. Amazon Echo's enrollment protocol runs over a temporary unsecured Wi-Fi network. A large number of existing consumer IoT devices use similar enrollment protocols.

We obtained a first-generation Amazon Echo, modified its firmware to make it vulnerable to man-in-the-middle (MITM) attacks, and recorded, decrypted, and analyzed the network traffic between the device, the companion smartphone application, and Amazon cloud. We describe our hardware and firmware modifications, the methodology, and the design of the testbed. We then analyze the OOBE enrollment protocol between the Echo, the companion smartphone application, and the Amazon cloud. We document the proprietary nature of the protocol and discuss some of its shortcomings that motivate the work discussed in later chapters of the thesis.

The rest of the chapter is organized as follows. We describe the methodology and experimental setup in Section 6.2. Section 6.3 describes and analyzes the Echo's enrollment protocol. In Section 6.4 we discuss our approach and the design of the enrollment protocol. Section 6.5 summarizes related work. Finally, we present our conclusions in Section 6.6.

6.2 Methodology and Experimental Setup

We obtained a first-generation Amazon Echo and modified its firmware [166, 167] to make it vulnerable to MITM attacks. We then connected the device to a MITM-capable testbed depicted in Figure 6.1 where we could record, decrypt, and analyze the network traffic between the device, the companion smartphone application, and Amazon cloud.

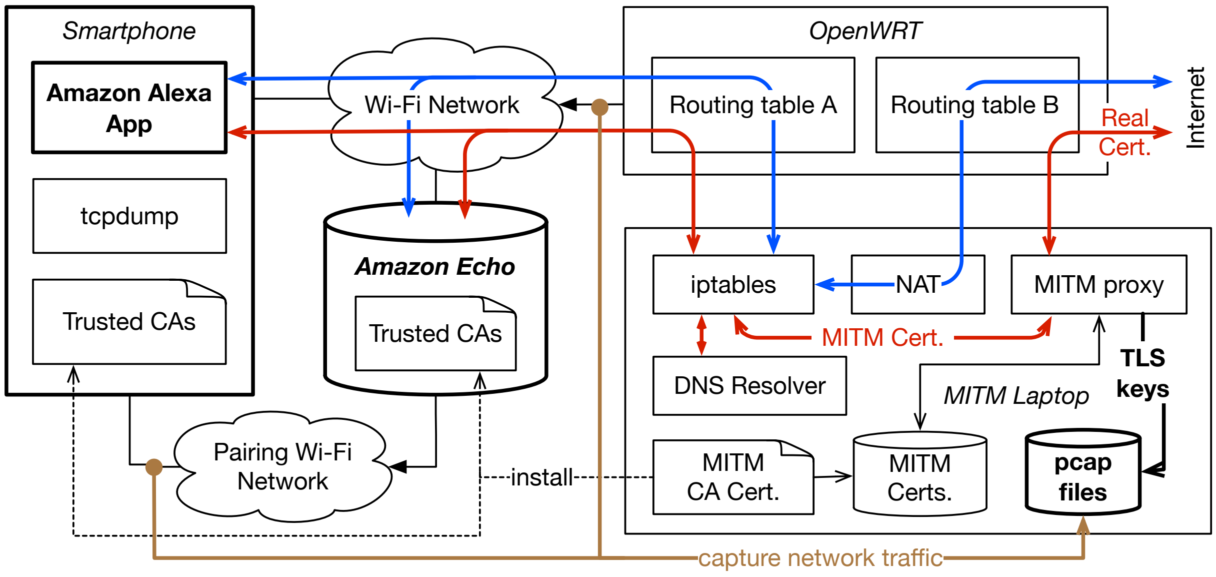


Figure 6.1: Experimental setup with Amazon Echo, Android smartphone with Alexa app, and a laptop with Mitmproxy. The Android device and the Echo have been modified to accept Mitmproxy's certificates.

We used a first-generation Amazon Echo with software version 618622220. The Alexa application ran on a rooted Huawei Honor 6X phone with Android 7.0. Our Wi-Fi access point (AP) was a

Linksys WRT1900AC with OpenWRT 17.01.4. Mitmproxy 4.0 (https://mitmproxy.org) was running on a Lenovo X60 laptop with Ubuntu Linux 18.04. We used tcpdump on the rooted smartphone to capture enrollment traffic between the Echo and the Alexa app. We also set Mitmproxy's certificate authority (CA) as trusted on the Android device.

We diverted all outgoing network traffic from the two devices to the laptop computer with iptables on the OpenWRT router to access encrypted network traffic between the Echo, Alexa app, and Amazon cloud. The laptop computer also served as a transparent network address translator (NAT) and router for all the network traffic we did not want to decrypt. To decrypt Transport Layer Security (TLS) traffic, we configured iptables on the laptop to forward such traffic to Mitmproxy. Mitmproxy was only applied to TLS connections between the Echo or Alexa app and Amazon cloud. Mitmproxy performed a MITM attack on the connections, replacing Amazon server certificates with locally generated ones. We used Wireshark to decrypt and analyze all captured network traffic.

We modified the Echo device to make it accept Mitmproxy server certificates. All our hardware modifications, shown in Figure 6.2, are based on the work done by the authors in [167, 166]. We exposed the debugging pads at the bottom of the device and connected a UART-USB converter to gain access to the serial console used by the boot-loader. We then connected an external Secure Digital (SD) card to the remaining pads.

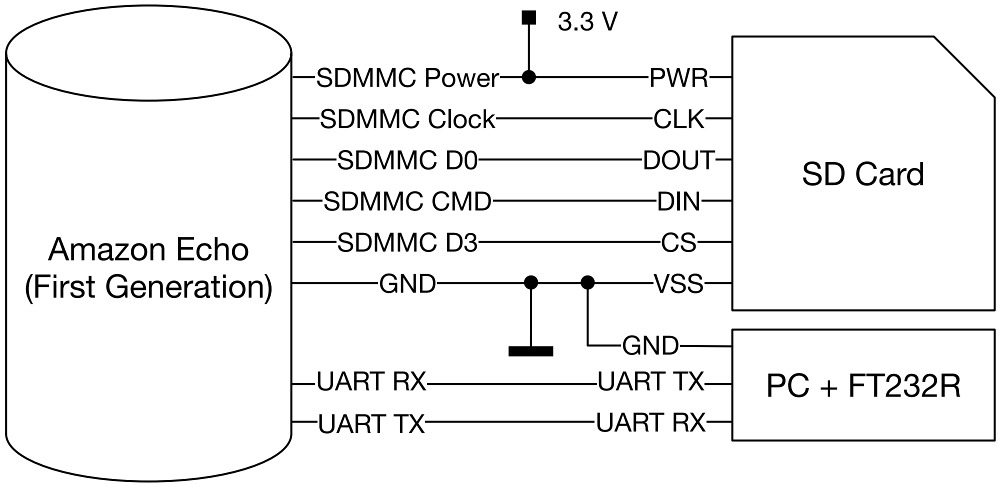


Figure 6.2: A modified Amazon Echo with an external SD card and a UART-USB interface attached to the device's debugging pads.

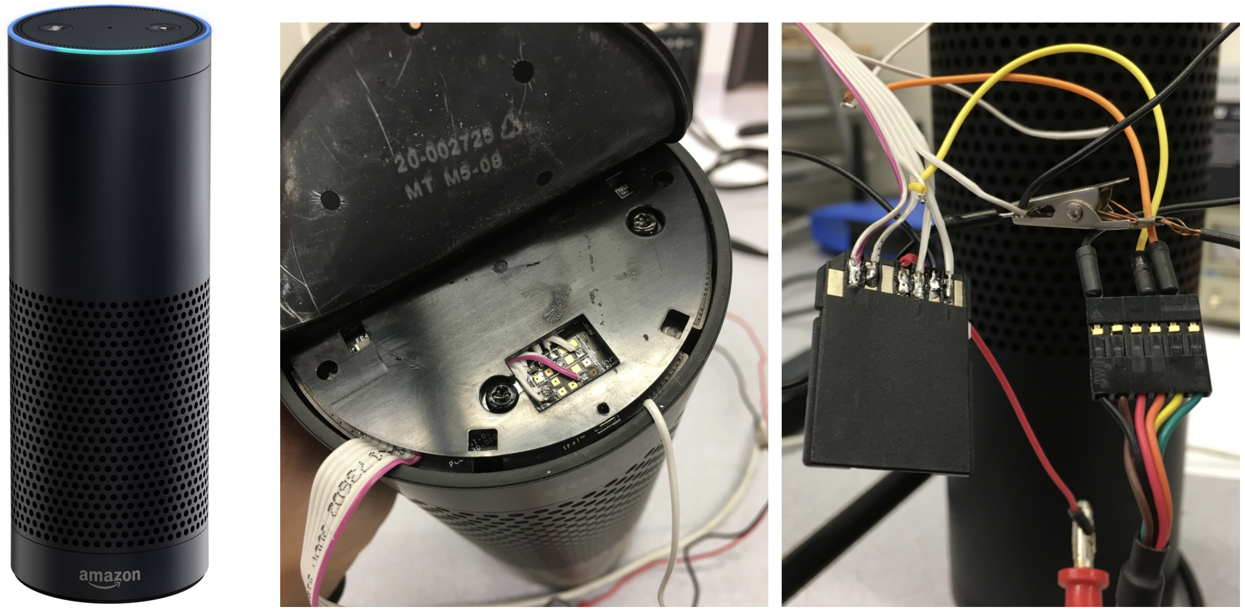


Figure 6.3: A photo of a modified Amazon Echo.

We then loaded the firmware image from [166] on the external SD card and rebooted the device. The custom firmware had a command line interface enabled, which allowed us to change the boot procedure. We followed the steps from [167] to obtain a root shell. We appended the Mitmproxy CA certificate in the root shell to file /etc/ssl/certs/ca-certificates.crt. After another reboot, the Echo would treat TLS connections modified by Mitmproxy as legitimate, allowing us to decrypt the communication.

6.3 Out-of-the-Box Experience (OOBE) Enrollment Protocol

6.3.1 System Architecture

Figure 6.4 illustrates the overall architecture of the ecosystem and the entities that participate in the OOBE protocol during device enrollment. The OOBE protocol is used to provision a network credential (Wi-Fi network name and password) into the Echo device and to associate the device with an Amazon cloud account. The protocol is executed between the Echo, an enrollment client in the form of an Amazon Alexa smartphone app or a web application, and backend services in the Amazon cloud.

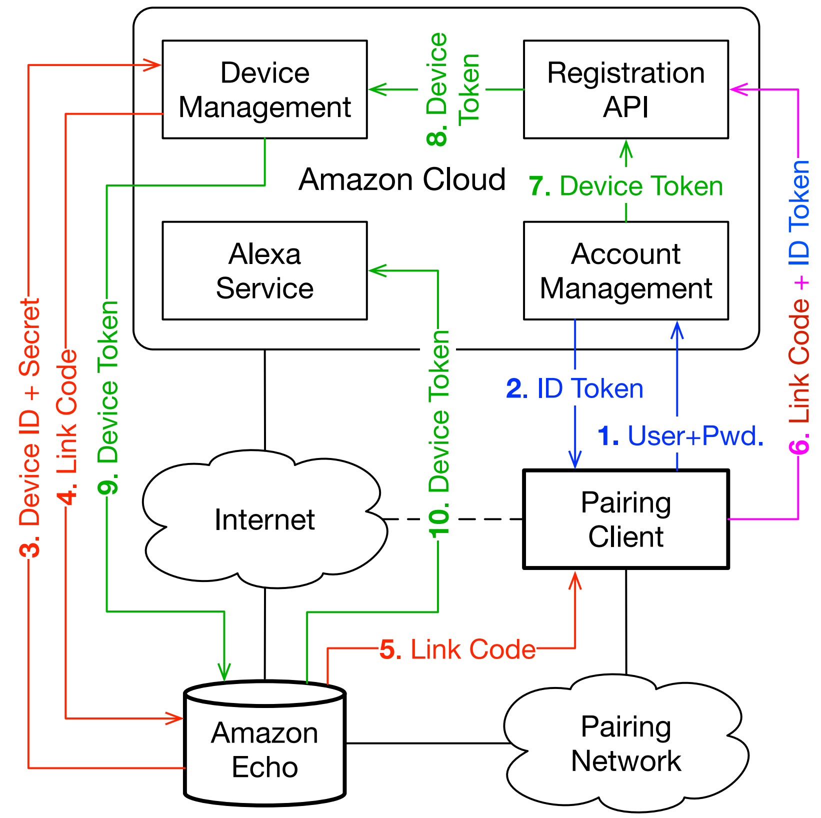


Figure 6.4: Data flow diagram of the Out-of-the-Box Experience (OOBE) enrollment protocol

The enrollment exchange occurs over an open temporary Wi-Fi network created by the Echo. The device creates a Wi-Fi P2P group and configures itself as the group owner. The SSID of the temporary network is "Amazon-XYZ" where XYZ represents three digits from the device's serial number. The Echo has two Wi-Fi interfaces and can create a temporary Wi-Fi network regardless of the state of its main Wi-Fi interface, i.e., even when connected as a client to another network on a different channel.

On the temporary network, the Echo configures itself with a fixed RFC 1918 IP address [168], starts DHCP and DNS servers, and configures iptables to redirect all DNS requests from connected clients to its internal DNS server. The internal DNS server has a few hard-wired, well-known Amazon hostnames and IP addresses, presumably to resolve those even when the device is not connected to the internet. If connected to the Internet during enrollment, the Echo forwards NAT-ed traffic from the enrollment client to the Internet via its main Wi-Fi interface. The enrollment client uses this to associate the device with the user's Amazon account. The Echo audibly announces only the first connected enrollment client with no identification. Multiple enrollment clients can be connected simultaneously.

6.3.2 Protocol Analysis

Figure 6.4 shows the data exchange between the various entities involved in the enrollment process. Figure 6.5 shows the out-of-band (OOB) protocol call flow between the enrollment client, Echo, and Amazon cloud. The Echo provides an enrollment API on TCP ports 8080 and 443. An HTTP server on port 8080 accepts JSON-serialized Thrift [169] messages[1] for the path `/OOBE`. Port 443 is a TCP proxy that forwards HTTPS connections from the enrollment client to the Amazon cloud.

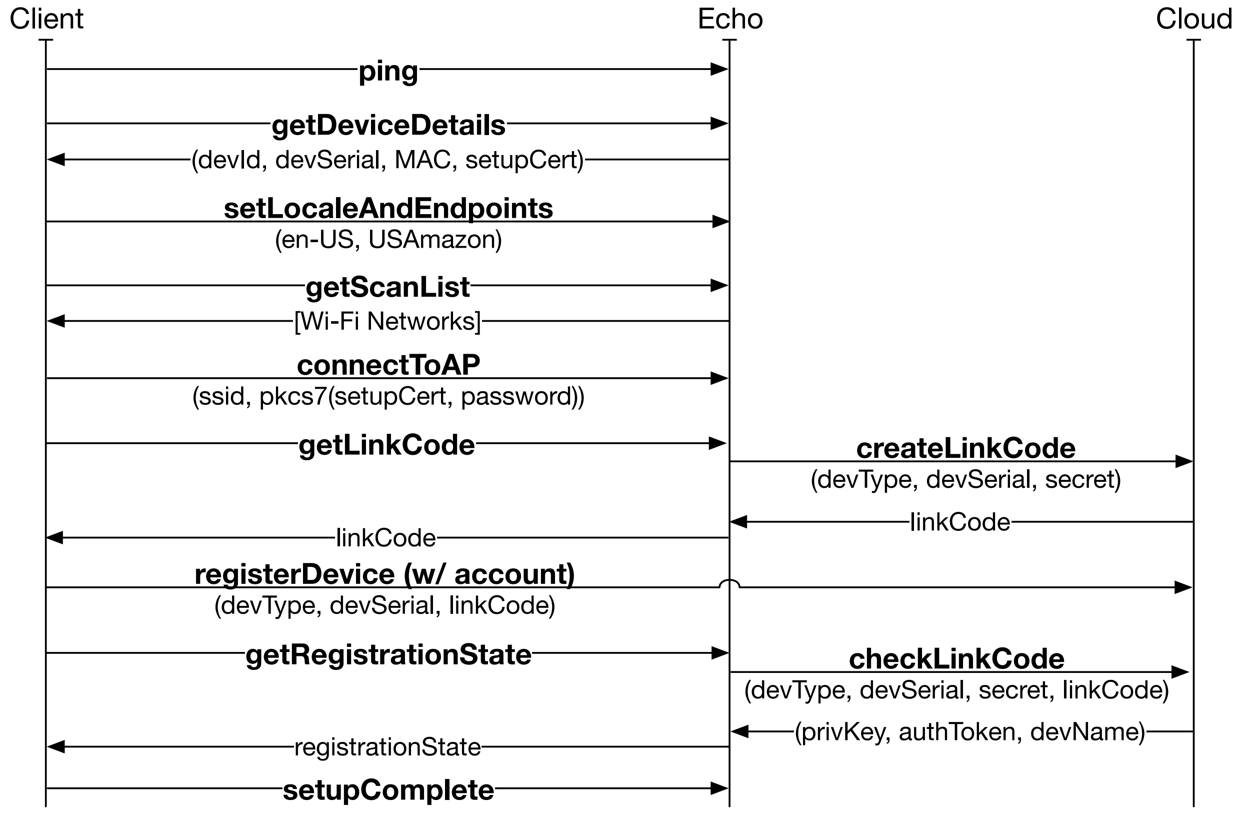

Client
Echo
Cloud
ping
getDeviceDetails
(devId, devSerial, MAC, setupCert)
setLocaleAndEndpoints
(en-US, USAmazon)
getScanList
[Wi-Fi Networks]
connectToAP
(ssid, pkcs7(setupCert, password))
getLinkCode
createLinkCode
(devType, devSerial, secret)
linkCode
linkCode
registerDevice (w/ account)
(devType, devSerial, linkCode)
getRegistrationState
checkLinkCode
(devType, devSerial, secret, linkCode)
(privKey, authToken, devName)
registrationState
setupComplete

Figure 6.5: Call flow diagram of the Out-of-the-Box Experience (OOBE) enrollment protocol

[1]Thrift is an interface definition language and a binary serialization protocol for cross-platform network services developed at Facebook.

The enrollment client periodically invokes the `ping` API method to determine if it is connected to the correct network on an Echo in enrollment mode. Since different Echo models may use different static IPs, the client sends the request to several pre-configured IPs in a round-robin manner. Pairing is initiated with the first device that correctly acknowledges the ping request. The ping method serves as a rudimentary service discovery mechanism. Unlike, e.g., ZeroConf, the ping method can be used by browser applications such as the enrollment application available at https://alexa.amazon.com.

After discovering an Echo in an enrollment mode (indicated by a spinning orange light), the client invokes the `getDeviceDetails` method. The method returns basic device information: device model, serial number, Wi-Fi hardware address, supported languages, and an enrollment X.509 public key certificate (used later). The client selects the language and configures Amazon cloud endpoints corresponding to the device's geographic location.

Next, the client determines whether the Echo has any Wi-Fi credentials configured already and obtains the list of scanned Wi-Fi networks. In Figure 6.5, these two steps are represented by the `getScanList` request. The client prompts the user to select a Wi-Fi network and invokes `connectToAP` to connect the Echo to the chosen network.

The enrollment client encrypts the Wi-Fi credential with AES-256 in CBC mode using a random secret key. The key is then encrypted with the public key from the X.509 certificate returned by `getDeviceDetails`. The encrypted key and Wi-Fi credential are transmitted in the Cryptographic Message Syntax (CMS) format [170]. The X.509 certificate provided by the Echo is self-signed, and this method is thus vulnerable to active MITM attacks.

The enrollment client then obtains a *link code* from the Amazon cloud via the Echo (steps 3–5 in Figure 6.4). The Echo first submits its device type, serial number, and a secret string to the `createLinkCode` API in the Amazon cloud. The returned link code (step 4 in Figure 6.4) consists of five alphanumeric characters and represents the device during registration with an Amazon user account. The format suggests that the code may have been designed for scenarios where the user is expected to transmit the string manually from the device to Amazon via the web interface. That API appears to be a part of a device rendezvous service, most likely connected to an inventory database that keeps track of the serial numbers and secrets for all manufactured devices. The secret string appears permanently set in each Echo device during manufacturing.

Having received the link code, the enrollment client registers the device with the user's Amazon account (step 6 in Figure 6.4). Since the client is connected to the Echo's enrollment Wi-Fi network, it uses the device as a proxy to reach the Amazon cloud to register the device. The client's request is end-to-end encrypted with TLS and authenticated with an HTTP cookie that authenticates the user (obtained in steps 1–2 in Figure 6.4). The client submits the device type, serial number, and link code identifying the device to be registered with the authenticated user's account. Amazon Cloud verifies the link code and associates the device with the user's account (steps 7–8 in Figure 6.4). These last steps are entirely implemented by Amazon Cloud, and we could not obtain more information about them.

The client invokes `getRegistrationState` on the Echo to determine its registration status. Using its secret, the Echo queries the rendezvous service for the state of the link code by periodically posting to the `checkLinkCode` API. Once registered, this API returns a private key, an authentication token (step 9 in Figure 6.4), and the human-friendly name assigned to the device by the user on the Amazon website. The Echo uses the private key and the authentication token to access the Amazon cloud on behalf of the user. The authentication token is included in every HTTP request to Amazon services (step 10 in Figure 6.4). It appears to contain data encrypted with a symmetric key cryptographically wrapped with a public-private key known only to Amazon and an initialization vector.

The client terminates the enrollment process by invoking the `setupComplete` method on the Echo, instructing the device to shut down the temporary Wi-Fi network.

6.4 Discussion

Our hardware modification is only effective with the first-generation Echo hardware. Modifying the firmware to inject a custom CA certificate on newer Echo models requires unlocking a secure boot process. Unlocking the secure boot process requires additional hardware equipment. Thus, the

MITM attack described earlier is much harder to perform on newer devices. However, the analyzed device enrollment protocol is compatible with newer Echo models and other ecosystem parts.

Neither the temporary Wi-Fi network nor the enrollment protocol are encrypted and are vulnerable to eavesdropping. An adversary observing the enrollment exchange could obtain the link code and might be able to associate a previously de-registered device with a different Amazon account. A newly purchased Echo comes pre-registered to the Amazon user account used to purchase the device or another account designated by the purchaser. This mechanism effectively prevents hijacking an Echo using the enrollment protocol if the device is registered in the Amazon cloud.

One of the supported enrollment clients is a web application implemented in JavaScript. For the client to communicate with the Echo device via unencrypted HTTP, the client itself must be served via unencrypted HTTP. This is necessary to work around the security limitations that prevent modern web browsers from loading mixed-security contexts (web resources). The web application is first loaded via HTTPS, and after the user has logged in, the browser is redirected to an HTTP URL, which downloads the JavaScript enrollment client. This design leaves the JavaScript enrollment client vulnerable to code injection by remote attackers.

6.5 Related Work

Clinton et al. [171] performed a hardware analysis of a first-generation Amazon Echo. Firmware extraction from newer Echo models is discussed in [172]. iFixit published a detailed teardown guide [173] for the device. We used some of these findings in our work: the description of debugging pads on the base of the devices and the ability of the Echo to boot from an external SD card. We used a firmware image obtained from [166] and followed the steps outlined in [167] to gain access to the Echo's embedded MultiMediaCard (eMMC) filesystem.

A security analysis of the Echo is presented in [174]. The authors performed a variety of attacks, including a sound-based attack, PIN brute-forcing attack, a replay attack on network traffic, and an attempt to exploit Amazon Alexa's API. While no significant vulnerabilities were found, the authors discuss potential weaknesses in the 4-digit PIN authentication method and the wake-word detection algorithm. If successfully exploited, it may be possible to activate the Echo using a highly distorted sound sample, which will not be recognized as a wake word by the user but will be recognized as such by the Echo.

Ford and Palmer analyzed [175] the network behavior of two Echo Dot devices over 21 days and created a network signature of the device from the captured traffic. The robustness of the signature will likely be limited due to a small number of devices and limited datasets.

The paper [176] analyzes the client and cloud components of the Amazon Alexa ecosystem from a digital forensics perspective. The authors analyzed available artifacts and developed a proof-of-concept digital forensic tool to collect and analyze the artifacts. The tool uses official and unofficial APIs and collects artifacts from the cloud, Amazon Echo, and smartphone devices.

While there have been several studies [177, 178] focusing on the Echo's hardware security and network behavior, little is known about the device's encrypted traffic. To the best of our knowledge, our study is the first attempt to analyze and document Echo's encrypted network communication.

6.6 Summary

Little is known about Amazon Echo's network behavior despite the large installed base. We modified the firmware to make the device vulnerable to MITM attacks. We then mounted a MITM attack against the device and decrypted the communication between the device, an enrollment smartphone application, and Amazon Cloud. We then analyzed and documented the OOBE enrollment protocol used by the first-generation Amazon Echo.

We found limitations in the OOBE enrollment protocol, which could be used to associate a de-registered Echo with a different Amazon account. This vulnerability is effectively mitigated by the pre-existing association of a new device to the purchasing Amazon account. Overall, the first-generation Amazon Echo is a well-designed device from the network communication perspective.

Our experiments were limited to the first-generation Amazon Echo. Later models make the MITM approach considerably more difficult. This limitation does not change the analysis of network communication, which is the primary contribution of this work.

6.6.1 Acknowledgments

Teresa Tseng and Aliza Isaacs, both undergraduate students at Barnard College, have been instrumental in decrypting, recording, and analyzing the encrypted traffic of Amazon Echo.

Chapter 7: On Usability of Network Enrollment in Cyber-Physical Systems

7.1 Introduction

Network enrollment[1] is the process of associating a new device to a networked system. The process must occur for the device to become operational, i.e., connected to the network, authorized to access a service or communicate with other devices. The task usually involves authenticating the device and provisioning (configuring) it to access the network infrastructure and its resources. Network enrollment can be viewed as a network management task (see Section 2.3) and, as such, is usually performed by the network administrator.

There is significant anecdotal evidence that network enrollment of IoT devices is a common source of confusion and frustration (H. Schulzrinne, personal communication, November 19, 2023) [179, 180, 181]. The enrollment process is often brittle, time-consuming, has a high failure rate, and does not scale to more extensive or heterogeneous networked systems, i.e., networks that include devices from several independent manufacturers and different form factors [182]. The problem is challenging, long-standing, and pervasive [183]. It is particularly apparent in the consumer IoT space where the end-user installing the devices does not necessarily have a networking background or expertise.

The literature abounds with network enrollment protocols for a variety of link-layer technologies [184, 185, 186, 187, 188] that are mutually incompatible, require extensive user interaction during pairing, and often have scalability limitations. Consumer Wi-Fi IoT devices often use a proprietary smartphone commissioning application that configures the device via either a temporary (SoftAP) Wi-Fi network created by the device or Bluetooth. The SoftAP approach is complex,

[1]Network enrollment is also known as pairing, association, registration, onboarding, provisioning, commissioning, initialization, or bootstrapping. While these terms generally refer to the same basic procedure, they often emphasize different aspects or are associated with a particular network technology. To use a common term, we use network enrollment to refer to all variants.

brittle, and scales poorly. In a heterogeneous IoT system with devices from various vendors, the end-user must use several commissioning applications that are often cumbersome and unreliable.

Practical experience with current IoT systems indicates that the problem is not solved and will only worsen as deployed IoT systems grow more extensive or heterogeneous. Keeping such systems operational will require an interoperable network enrollment framework with relaxed assumptions, e.g., no physical access to the device, support for various link types, and simultaneous enrollment of multiple devices. Developing a network enrollment framework to meet the criteria requires carefully considering advanced deployment scenarios and the lifecycle of deployed IoT devices.

We revisit the network enrollment problem in this chapter, focusing on large or heterogeneous deployed IoT systems. We systematically describe advanced IoT deployment scenarios to motivate the work. We specifically focus on the complete lifecycle of fixed IoT devices installed in shared physical spaces, e.g., rental spaces. We then describe the various entities involved in the general network enrollment process and develop an abstract network enrollment model. The model supports device heterogeneity, works across link-layer technologies, and can interoperate with existing network infrastructure (e.g., legacy Wi-Fi APs). We also discuss the design of an IRB-approved human-subject usability study that could be used to evaluate the model. The usability evaluation is left for future work.

Specifically, this chapter makes the following contributions:

- A systematic study of advanced deployment scenarios, assumptions, and requirements;
- The design of a general abstract network enrollment model;
- The design of an IRB-approved human-subject usability study.

The rest of the chapter is organized as follows. We begin by motivating the work and discussing the problem in Section 7.2. Section 7.3 discusses a few advanced network enrollment scenarios (focusing on IoT systems) that a usable and scalable network enrollment framework should support. In Section 7.4, we discuss the design challenges and requirements in detail. We review related enrollment protocols in Section 7.5. Section 7.6 discusses the design of the WIDE enrollment framework. We describe a prototype implementation in Section 7.7. Section 7.8 describes the design of an IRB-approved human-subject usability evaluation study. Finally, Section 7.9 summarizes the research and discusses future work.

7.2 Motivation and Problem Statement

Affordable system-on-a-chip (SoC) platforms with built-in connectivity have helped to create a consumer IoT market that is projected to significantly grow in the near future [189, 190]. Many consumer IoT devices are designed to be self-installable, i.e., the owner of the device (end-user) performs network enrollment and all the configuration for the device to become operational.

As IoT technology penetrates new areas and application domains, specialist installation, e.g., by a licensed technician, will be increasingly necessary [191]. Specialist installation is common in fixed infrastructures that "come with the space". It may be necessary for IoT devices installed in locked utility or electrical closets [192], security-oriented devices, devices installed behind walls [179, 193], or in building areas with restricted access. The end-user (and network administrator) may not have physical access to such devices, which may need to be installed before the operational network is operational. In such scenarios, the enrollment process must be split among several entities.

There are scenarios where the specialist and the commissioner (network administrator or end-user) cannot communicate directly. Consider a rental property where the superintendent hires one or more specialists to perform subsystem (e.g., lighting, communication, HVAC) installation. The specialists authenticate the devices during installation but can only communicate the authentication result to the superintendent. The superintendent will relay the information to the tenant upon signing a lease so that they can enroll the devices into their network. The existing network enrollment frameworks do not adequately address scenarios like these.

Shared physical spaces pose additional challenges for IoT infrastructure [194, 195, 196]. It is reasonable to expect that future homes, apartments, or office buildings will come with pre-installed smart (connected) infrastructure in the form of devices that belong to the space. These devices will likely outlive individual tenancies in shared homes, Airbnb rentals, or shared offices. A network enrollment protocol for shared spaces must support device enumeration, ownership transfer (e.g., during moving), enrollment revocation, and re-enrollment. Existing enrollment protocols generally cannot handle this kind of management complexity (Section 7.5).

Long-lived IoT infrastructure will inevitably need to grow organically and evolve [197], where a subset of the infrastructure is replaced with newer or improved components. Ideally, such evolution would have a minimal and controlled impact on the rest of the system. For example, replacing a Wi-Fi access point or network credentials should allow previously enrolled IoT devices to reconnect automatically. Additionally, some devices may come and go with visitors, workers, or specialists [183]. Existing network network enrollment frameworks rarely support such system evolution.

The lack of network enrollment standardization pushed IoT device vendors to roll out proprietary solutions. Vendor-specific solutions tend to use confusing user interface terminology, implement cumbersome enrollment processes, and are plagued with security [198] or reliability issues [180, 1]. Mutually incompatible vendor-specific enrollment protocols make managing heterogeneous IoT systems difficult and time-consuming.

Existing network enrollment frameworks (Section 7.5) cannot adequately handle the scenarios mentioned earlier. Different types of devices need to be managed separately using vendor-specific mobile applications. This approach does not scale, takes significant time and effort, and has a steep learning curve. Keeping network credentials across multiple applications could expose the credentials. Physical interaction with devices is not always possible. Transferring IoT device ownership is complex and cumbersome. Such scenarios require a more general enrollment model, both at the network and application level, where device authentication, authorization, and commissioning may need to be performed by different entities at different times, repeatedly, and with only a portion of the infrastructure operational. Scalability and usability are increasingly essential factors with the growing size and heterogeneity of deployed IoT systems.

7.3 Advanced Enrollment Scenarios

This section discusses selected advanced enrollment scenarios not adequately supported by existing network enrollment frameworks. The list is not exhaustive and is meant to inform the formulation of design challenges and requirements in Section 7.4.

First-Party Enrollment (Baseline). The most common and widely supported enrollment scenario is first-party enrollment. Most consumer IoT devices are designed with this scenario in mind. Thus, it is listed here as the baseline scenario.

In the first-party enrollment model, IoT devices are enrolled into the Wi-Fi network by the device's end-user, who is also the network owner (administrator) with full administrative privileges to the network (e.g., the AP). First-party enrollment is the most common scenario for small deployments, e.g., a single-family home with a pre-existing Wi-Fi network and several devices.

The first-party model assumes that the administrator has physical access to the enrolled device and can thus use authentication methods that rely on proximity-based out-of-band (OOB) communication channels, for example, NFC, QR codes, or Bluetooth. In consumer devices, network enrollment is usually automatically activated when the device has been powered on for the first time or upon losing access to the previously enrolled network. In other cases, network enrollment typically needs to be manually activated by the end-user, e.g., by pressing a button.

First-party enrollment is the scenario natively supported by enrollment frameworks such as Wi-Fi Easy Connect [184] or WPS [185].

Remote Enrollment. In the remote enrollment scenario, the network administrator performing the enrollment has physical access to neither the device being enrolled nor the Wi-FI AP at the time of enrollment. This scenario represents situations where the administrator is asked to help set the system up remotely, e.g., for family or friends, or where the system is physically large.

The remote enrollment scenario requires the following additional features from the network enrollment framework:

- A signaling protocol between an application used by the administrator and the network (AP);

- A device authentication mechanism that does not rely on proximity-based OOB channels;
- A pre-existing configuration or a network selection heuristic algorithm in the device;
- Pre-enrollment restricted connectivity for the device to communicate with the remote administrator.

A majority of the existing enrollment protocols do not support the remote scenario.

Fully-Automated Enrollment. This enrollment model assumes the network administrator cannot authenticate the device being enrolled directly and must rely on the information a third party provides, e.g., a certificate signed by the device manufacturer. This approach assumes the administrator and the manufacturer have a common root of trust, e.g., a certificate authority provided by the manufacturer and trusted by the network administrator.

A secure communication channel must exist to transfer the device authentication information from the manufacturer to the network administrator, either directly or via a chain of device distributors or resellers. For devices obtained offline, this could be accomplished with device authentication information included in the packaging. The authentication information could be provided to the administrator in a machine-readable format for devices obtained online.

When correctly implemented and executed, this model could provide a fully automated (zero-touch) network enrollment where the device automatically connects to the target network without needing manual intervention by the network administrator.

Transient Enrollment. The transient enrollment model covers scenarios where devices are enrolled to a network temporarily, for example, in rental apartments, rental offices, university campuses, or shared physical environments.

This model covers two sub-scenarios: (1) temporarily enrolling an own device to a foreign network, e.g., in an Airbnb, hotel, or office; (2) temporarily enrolling fixed devices to own a Wi-Fi network in rental spaces.

To support (1), the tenant's commissioner device must be given access to the host Wi-Fi network with limited privileges, e.g., without the privilege to remove or reconfigure existing (host's) devices.

Furthermore, the guest may only be able to enroll devices for a specific period.

Scenario (2) is designed for situations where the tenant provides internet connectivity, as is common in rented office spaces. The tenant enrolls a subset of their network's fixed building (office) infrastructure in this scenario. Upon leaving, the devices will be re-enrolled to another system, i.e., the owner or a new tenant. As a particular case, scenario (2) may also be helpful in long-term AirBnB or hotel rentals where the tenant brings their network connectivity. Enrolling existing devices in the tenant's network would also give them extra control over the devices' network communication, e.g., in privacy-sensitive applications.

Concurrent Enrollment. The concurrent enrollment model involves scenarios where many devices must be enrolled into the network. Consider a smart home that contains a large number of fixed infrastructure devices installed during construction. If the number of devices is large, enrolling the devices to the network individually could take a long time. A scalable and usable network enrollment framework should be capable of concurrently enrolling devices.

7.4 Design Challenges and Requirements

This section discusses the most important design challenges and requirements that should be met by a network enrollment framework usable in advanced deployment scenarios involving large-scale, heterogeneous, or federated IoT systems such as those listed in Section 7.3.

Interoperability. Like all long-lived infrastructures, future IoT systems will inevitably become heterogeneous[2] through component replacements, system evolution, and organic growth. Such infrastructures will require an interoperable network enrollment framework that does not depend on proprietary vendor-specific components. For example, if the framework requires a smartphone application, the application should be open and distributed with the smartphone platform rather than vendor-specific and downloaded from an app store. Ideally, the framework should also be

[2]A heterogeneous platform consists of different interoperable components produced by a variety of vendors or manufacturers.

interoperable with existing networking infrastructure, such as Wi-Fi access points, authentication, authorization and accounting (AAA) infrastructure, and smartphone platforms.

Scalability. Fixed home or office infrastructure (e.g., HVAC, heating, or lighting systems) can consist of tens if not hundreds of devices. Replacing the devices with their connected counterparts currently requires enrolling each device manually, one after another. Sequential enrollment can be a laborious and time-consuming process. A scalable enrollment framework should automate the enrollment of multiple devices as much as possible. Several traits could improve network enrollment scalability, such as device authentication reuse, concurrent device enrollment, or one-time enrollment across link-layer technologies (e.g., simultaneously enrolling into Wi-Fi and Bluetooth).

Role Separation. Contemporary consumer IoT systems generally assume self-installation by the end-user, who assumes the roles of an installation or configuration specialist, network administrator, and system integrator. Self-installation is uncommon in legacy fixed infrastructures where a certain skill, privileged access, or expertise might be required[3]. The enrollment framework should assume that the enrollment process will be distributed across multiple independent entities to support advanced deployment scenarios such as those listed in Section 7.3. For example, two independent users with limited privileges may perform device authentication and network commissioning. In larger deployments, e.g., on a university campus, the device end-user and the network administrator may also be independent entities.

Additionally, there might be scenarios where the specialist installing the device and the network administrator could only communicate via an intermediary. Consider a rental property where the superintendent hires one or more specialists to perform subsystem (e.g., lighting, communication, HVAC) installation. The superintendent collects device installation manifests and redistributes those to tenants (network administrators) upon lease signing. The tenants use the manifests to commission the devices to their network. Having never met the specialist, the tenant may need to rely on the

[3]Consider a lighting system where devices, e.g., connected light switches and fixtures, are installed by an industry specialist. The specialist configures and tests the devices but may not have access to the end-user's network, which may not yet be operational at the time of the installation.

superintendent to establish the manifest's authenticity (and its devices) using standard cryptographic techniques.

A network enrollment framework with support for role separation should assume that device installation, authentication, commissioning, and integration will be performed by independent entities, i.e., different people cooperating to set up complex IoT infrastructure. Following the principle of least privilege, each user might have privileged access only to a part of the system.

Link Abstraction. Modern IoT devices increasingly have built-in support for more than one link layer technology. For example, devices with simultaneous support for Wi-Fi (IEEE 802.11) and Bluetooth are increasingly common. Future devices will likely add IEEE 802.15.4-based interfaces (e.g., Thread, ZigBee, 6LowPAN). The established practice uses a dedicated (link-specific) enrollment or pairing protocol on each link. This approach is tedious and complicated. A better enrollment solution would enroll a device only once, reusing the enrollment outcome across all link types supported by the device. Similarly, a successful enrollment between a pair of devices could be extended to all devices associated with the same administrative account, as implemented in the Apple MagicPairing protocol [199].

Physical Access. Most existing network enrollment systems assume that the network administrator has (or can obtain) physical access to the enrolled device. Physical device access is often necessary to authenticate the device or activate network enrollment, e.g., by power-cycling the device or pushing a button. A network enrollment protocol for advanced deployment scenarios must not assume physical access to the enrolled device. Neither the device end-user nor the network administrator may be able to obtain physical access to the device. This could be the case for devices installed by licensed specialists in inaccessible locations or for devices where tampering is strictly prohibited. Not having physical access limits the choice of authentication protocols and requires careful management of previous device authentication results.

User Interface. The form factor of IoT devices often does not allow for the installation of a UI on the device. This is particularly challenging for authentication protocols. Even if the IoT device has a

UI, in some cases, it may be installed so that the UI is not physically reachable to the end-user. Not having a UI on a device also makes inputting network credentials impractical and inefficient. A network enrollment framework for advanced deployment scenarios should assume that at least some devices will be physically unreachable or may not provide a UI. Unique or identifying information physically shown on the device, e.g., a QR code or serial number, may not be visible.

Pair-Wise Connectivity. In large installations where different entities install and configure different parts of the IoT system independently, not every entity involved in setting up the infrastructure may be online simultaneously. For example, IoT devices may be installed by specialists, e.g., in a house or apartment being constructed, before the network infrastructure is available or configured. The commissioner device used by the specialist may not be able to connect to the operational network while it is connected to the device. Similarly, the commissioner device may be offline when the IoT device attempts to connect to the operational network later.

A network enrollment framework supporting such scenarios should assume pair-wise connectivity between the various entities. Pair-wise connectivity means that only a single connection between two entities may be available at any time, for example, device–commissioner, commissioner–network, device–network.

Device Lifecycle Management. Most existing network enrollment protocols only support the initial enrollment of a device into the network. The focus is on new devices being connected for the first time. Large deployed IoT infrastructures will inevitably need to evolve so that outdated or malfunctioning devices and components can be replaced. When the device being replaced is part of the network infrastructure, e.g., a Wi-Fi AP, care should be taken to keep all previously enrolled devices connected through the updated component.

Consider an apartment being sublet by its tenant to a subtenant temporarily. The subtenant may wish to temporarily enroll existing IoT devices into their system, e.g., cloud-based systems or mobile applications, for the duration of the sublet. Similarly, a mechanism must exist for the original tenant to remove such associations after the sublet period ends.

To support such scenarios, the network enrollment protocol should be designed to enroll new devices into the network and manage the association between the device and the network for the duration of the device's lifetime. The network enrollment framework should also allow selectively removing an individual device from the network without reconfiguring (rekeying) the entire network, for example, when a device is being retired or resold.

When the device detects a permanent connection problem to the operational network, e.g., due to invalid network credentials, the device should remain reachable from the commissioner so that its enrollment can be updated without resetting the device to factory defaults.

Privacy and Tracking. IoT devices can be deployed where physical security is difficult or impossible. The trust model should distinguish between devices susceptible to physical compromise and devices with some level of physical security [197].

We assume IoT devices can use public-key cryptography and keep private keys reasonably secure, for example, in a tamper-resistant secure element (SE). The SE should make obtaining or changing sensitive data reasonably difficult. Successfully mounting such an attack should require lab-grade hardware equipment.

We further assume the device can re-generate public-private keys on factory reset, for example, when the device is being resold. This could be done for privacy reasons, e.g., to prevent user or device tracking. Meeting this requirement requires a hardware-based random number generator in the device. Any identifying information that cannot be re-generated, for example, a certificate signed by the manufacturer, should only be used in a limited scope to prevent tracking.

Offline Device Acquisition. The device acquisition method determines the communication channel between the manufacturer and the network commissioner. Notably, it determines whether an automated (without user intervention) network enrollment could be possible.

If the end-user purchases a new IoT device online, i.e., directly from the manufacturer or via a distributor like Amazon, proof of ownership can be obtained automatically from the manufacturer and imported into the commissioner. Since the process is online, it could be fully automated. For example, the manufacturer or distributor could email the proof to the user.

If the user purchases the IoT device offline, for example, from a brick-and-mortar retail store, there is no online communication channel from the manufacturer to the commissioner. The manufacturer could include proof of ownership in the packaging. To transfer the proof to the commissioner, the user must scan a QR code, tap the device with an NFC reader (smartphone), or perform a similar interactive step.

7.5 Related Enrollment Protocols

Sethi et al. [200] surveyed enrollment protocols developed by the IETF [201], IEEE [202], FIDO Alliance [203], Open Connectivity Foundation (OCF) [204], and Open Mobile Alliance (OMA) [205]. This section complements their effort.

The Resurrecting Duckling Model. Many enrollment protocols implement a variant of a transient secure association, where a new device implicitly trusts the first entity that can communicate with it. This model was first described in [206, 207] and is known as the resurrecting duckling model. The resurrecting duckling model can be viewed as a variant of trust on first use (TOFU) authorization. The enrolled device (the duckling) implicitly authorizes the first entity (trusted device) that can communicate with it to enroll the device to the network. A notable weakness of the scheme is the weak authentication of the trusted device. The resurrecting duckling model is helpful in controlled environments where active adversary attacks, e.g., hijacking the device to a malicious network, are improbable.

IEEE 802.11 (Wi-Fi). Wi-Fi Alliance has standardized two enrollment protocols for IEEE 802.11 (Wi-Fi) networks: Wi-Fi Protected Setup (WPS) [185], and Wi-Fi Easy Connect (DPP) [184]. Many IoT devices use manufacturer-specific enrollment protocols running over a temporary Wi-Fi network created by the device (SoftAP).

Cisco initially developed WPS[4] and later had the protocol standardized by the Wi-Fi Alliance.

[4]Also known as Wi-Fi Simple Configuration Protocol.

The protocol simplifies adding new devices to a Wi-Fi network without entering lengthy passwords. To enroll a new device, the user presses a button on both the AP and the device simultaneously (within a two-minute interval known as the "walking distance"), or they enter a PIN from the AP's sticker into the device. Upon authenticating the device, the AP transmits the WPA pre-shared key (PSK) to the device. WPS is often disabled by default on modern APs due to its vulnerability to brute-force attacks [208], allowing the attacker to recover the password in a few hours. Another weakness of WPS is a limited form of automatic network selection. WPS does not guide what network to select if multiple actively enrolling networks are found.

The Wi-Fi Easy Connect (DPP[5]) framework [184] is a successor to WPS. In DPP, a device (enrollee) is enrolled into a secure Wi-Fi network by a trusted device (configurator), typically a smartphone, trusted by the AP. Exchanges in DPP are secured with elliptic curve (EC) public-key cryptography and use IEEE 802.11 Public Action frames [209] to exchange data. Each enrollee is identified by an EC public key, which is transferred to the configurator via a quick-response (QR) code, Bluetooth, or near-field communication (NFC). The configurator authenticates the enrollee's public key and issues a certificate to the device.

DPP requires the configurator to be within physical proximity of the enrollee (device) and the Wi-Fi AP, but not necessarily at the same time. This makes DPP suitable for intelligent home scenarios with a simple Wi-Fi network configuration. While the use of DPP in more advanced deployment scenarios, e.g., large campuses, hospitals, or manufacturing facilities, is possible, the usability of this method remains unknown. DPP is a Wi-Fi-only framework incompatible with existing authorization infrastructure based on RADIUS [210] or DIAMETER [211] protocols. This is a potential limitation in large-scale or professionally managed installations.

Many IoT devices use the so-called "SoftAP" approach, i.e., a vendor-specific enrollment protocol executed over a temporary Wi-Fi network created by the device. The device creates the temporary Wi-Fi network when it needs to be enrolled, i.e., after a factory reset or when it cannot connect to the operational network. Another device, e.g., a smartphone, connects to the temporary

[5]The framework is also known as the Device Provisioning Protocol (DPP).

Wi-Fi and configures the device with network credentials. This method is used by Amazon Echo [1], Google Chromecast [212], Google Home [213], the AllJoyn framework [214], and Matter [215].

The device selects a unique SSID that typically includes the device model and a short string derived from the device's serial number or link-layer address. The device configures itself as an AP in the temporary network, providing essential services such as DHCP and DNS. The configuration protocol is manufacturer-specific and commonly based on HTTP REST or XML-RPC.

The SoftAP approach requires an application running on the smartphone. Since SoftAP configuration protocols have not been standardized, a separate application for each manufacturer is needed. This can be a significant complication in heterogeneous IoT systems incorporating devices from many manufacturers. This approach also faces many limitations originating from the smartphone OS. The applications often could not automatically obtain Wi-Fi credentials from the smartphone platform due to restrictive security policies. The Wi-Fi credentials entered into one manufacturer-specific commissioning application could not be automatically shared with other commissioning applications. The SofAP approach is generally complex, brittle, and disruptive for the smartphone, allowing only one device to be enrolled at a time.

Matter. Matter [215] is a framework for interoperable smart home (IoT) devices, systems, and applications developed by the Connectivity Standards Alliance (CSA). Matter can be viewed as a successor to ZigBee [187]. The framework defines an IPv6-based network topology over three link-layer technologies (Ethernet, Wi-Fi, Thread) and the related network services and protocols, including a link-layer and network-layer enrollment protocol.

A commissioner (usually a smartphone) first obtains the device's passcode (secret) by scanning a QR code, tapping an NFC tag, or asking the user to enter the passcode manually. The commissioner then discovers the device via a link-layer-specific discovery protocol. Matter uses the SoftAP method described earlier on Wi-Fi, with link-local IPv6 addresses and DNS for service discovery (DNS-SD). Once connected, the device and the commissioner establish a mutually-authenticated session using SPAKE2+ [216]. The commissioner then configures the device for network access and prompts it to connect to the primary operational network.

Amazon Frustration-Free Setup. Amazon Frustration-Free Setup (FFS) [217] is Amazon's proprietary device enrollment framework that implements a peer-assisted enrollment model. A device already enrolled into the operational network (provisioner) assists a new device being enrolled (provisionee) by proxying its enrollment requests to the Amazon Device Setup cloud service. The service authenticates the provisionee, confirms its association with the provisioner's Amazon account, and sends network credentials previously saved by the user in the Amazon cloud to the provisionee.

The FFS framework has been adapted for Bluetooth, Wi-Fi, ZigBee, and Matter [215]. The FSS backend service in the Amazon cloud identifies devices using X.509 public key certificates signed by a CA certificate uploaded to Amazon by the manufacturer and unique to a particular device type. The framework depends on several proprietary Amazon services, including Alexa, Smart Home Wi-Fi locker, and the corresponding Android and iOS applications.

FSS supports a zero-touch mode of operation when the following preconditions are met: (1) the device is purchased online from Amazon, (2) it must be pre-registered to the correct Amazon account, (3) the user has Wi-Fi credentials stored in the Smart Home Wi-Fi locker, and (4) the user already has an FFS-enabled device. If the zero-touch mode fails or cannot be used, the user must scan a QR code with the device's public key certificate to enroll the device.

Table 7.1: A comparison of existing Wi-Fi enrollment protocols

	WPS[1]	DPP[2]	Matter	SoftAP	FFS[3]
Proprietary	No	No	No	Yes	Yes
Application-layer enrollment	No	No	Yes	Yes	Yes
Requires public internet	No	No	No	No	Yes
Participating entities	Enrollee, Registrar, AP	Enrollee, Configurator	?	?	?
Enrollee authentication	PIN, Button	QR, BTLE, NFC	N/A	N/A	CA
Initial enrollee assumptions		EC public key	Certificate		

[1] Wi-Fi Protected Setup
[2] Wi-Fi Easy Connect
[3] Amazon Frustration-Free Setup

Apple HomeKit. Apple HomeKit pairing is an application-level enrollment solution for iOS accessories [218]. It supports Wi-Fi and Bluetooth Low Energy (BTLE), assumes that connectivity has been established between the joiner and the commissioner, and requires that the joiner is equipped with an Apple-issued authentication coprocessor.

Mutual authentication in the HomeKit framework is established with the Secure Remote Password (SRP) protocol [219] with a randomly generated eight-digit code for the password. The devices exchange unique IDs and Ed25519 public keys over the secure session. The identity of the joiner (the unique ID and the Ed25519 key pair) is re-generated on factory reset. To establish a communication session, the devices generate ephemeral Curve25519 public-private keys and exchange those, protected with the identity keys saved during the enrollment phase. The communication session uses the station-to-station protocol [220].

EAP-NOOB. Nimble Out-of-Band Authentication for EAP (EAP-NOOB) [221] is a method for secure bootstrapping of IoT devices. The method assumes no preestablished relationship between the device and the network and requires a unidirectional user-assisted OOB channel. EAP-NOOB runs ephemeral ECDH key exchange over a sequence EAP exchanges [222]. The resulting session is authenticated with a QR code transmitted over the OOB channel. This method is usable in networks that support EAP, e.g., Wi-Fi in WPA-Enterprise mode or IEEE 802.1X [223] networks.

Bluetooth. The standard Bluetooth enrollment protocol is Secure Simple Pairing [186]. The most recent versions (Bluetooth 4.2 or newer) use the Elliptic Curve Diffie-Hellman (ECDH) protocol to establish a secure connection between the device (joiner) and the host (commissioner) over the Bluetooth interface. The connection is then authenticated using one of several methods: *Just Works*, *Out of Band*, *Numeric Comparison*, or *Passkey Entry*.

The *Out of Band* and *Numeric Comparison* methods are considered secure [224]. The *Numeric Comparison* method has a variant known as *Just Works*, where the actual comparison is not performed. This variant was designed for secure locations, but an active MITM attacker might still be able to force the devices to re-pair outside of the secure location [225, 226]. The *Passkey Entry* method is vulnerable to passive attacks when the same password is used to repair multiple times, as is often the case with static passwords [227, 225, 226].

Apple MagicPairing is a proprietary Bluetooth pairing protocol developed by Apple for the AirPods and other Apple Bluetooth peripherals [199]. MagicPairing replaces the long-term key derived through SSP with a key derived from a master key stored in the user's iCloud account. The primary benefit of this approach is that the peripheral paired with one device can automatically connect to all other devices associated with the same iCloud account. This is more convenient and eliminates the need to pair the device repeatedly, minimizing the risk of MITM attacks [228].

Google Fast Pair (FP) [229] is a proprietary Bluetooth Low Energy (LE) pairing protocol for the Nearby platform [230]. The FP protocol combines Bluetooth Low Energy discovery with Bluetooth Secure Simple Pairing. Upon discovering a nearby device advertising FP support, the device (smartphone) retrieves additional information about the device, such as the make, model, and device picture, from the Google Cloud platform. It displays that information to the user, who then pairs the device with a single click.

The Fast Pair Service is only available to pre-registered devices. The Google platform generates a private key unique to the device model during registration. The key is used in the discovery phase to prevent impersonation attacks. The FP protocol assumes the smartphone is connected to the internet during pairing. The actual pairing protocol is based on Bluetooth SSP.

BRSKI. Bootstrapping Remote Secure Key Infrastructure (BRSKI) [231] is an enrollment protocol for autonomic networks. The RFC discusses the authentication of devices, including sending authorizations to the device as to what network they should join and how to authenticate that network by specifying automated bootstrapping of an autonomic control plane (ACP). BRSKI combines manufacturer-installed X.509 certificates with the manufacturer's authorization service.

IEEE 802.15.4. The ZigBee commissioning architecture defines a set of reusable building blocks to be used by an application-specific commissioning framework [187, 232, 233]. ZigBee defines two commissioning variants: link-level and application-level. Link-level commissioning allows a

ZigBee device to join a secure ZigBee network. Application-level commissioning establishes secure application-level logical links, i.e., to “bind” compatible interfaces provided by different devices. The security of a ZigBee-based system depends on the safekeeping of symmetric keys. Little can be said about the security of ZigBee commissioning in general because it ultimately depends on the application (profile). Some existing profiles are known to be insecure [234].

Thread Commissioning [188] is a commissioning solution for IEEE 802.15.4 devices that use the Thread protocol architecture. Unlike other solutions, Thread commissioning is primarily based on standard protocols: DTLS 1.2 [235], PAKE (EC-JPAKE) [236], and CoAP [237].

Usability Studies. The ubiquitous computing and human-computer interaction (HCI) research communities have performed a large number of empirical user studies [179, 238, 181, 191, 239]. The studies indicate that even people with advanced computing knowledge struggle with home networking. Network configuration and management were consistently reported as a barrier to broader smart home technology adoption.

Thomas et al. proposed applying traditional HCI-inspired usability evaluation criteria such as flexibility, operability, learnability, and understandability to IoT systems [240].

Jewell et al. evaluated four configuration strategies for Wi-Fi devices with minimal user interfaces [241]. Their study involved 15 users testing three device pairing methods based on a flashing light, USB connection, and web-based UI. Each subject was provided with written instructions, pre-experiment and post-experiment questionnaires, and the experiment was filmed. The authors recorded each participant’s success rate, task completion time, and subjective ratings. This experiment inspired the design of our usability study.

Chong and Gellersen [242] tested 700 user-defined actions with 37 device combinations. The authors found that end-users chose proximity and touch-based methods when associating wireless devices.

Uzun et al. tested five secure device association methods [243]. The authors compared various pairing methods and compared their success/error rate. They found that methods that ask the user to compare and confirm information rather than enter it generate significantly higher success rates for non-technical users. The paper also makes specific recommendations for UI design to improve the success rate of device pairing.

Several researchers investigated the smart device trends and practices in shared physical spaces, i.e., rented places such as Airbnbs or multi-user homes. These studies focused primarily on privacy and security aspects.

Mare et al. [194] performed a large-scale study among Airbnb hosts and guests to learn about their practices, preferences, and privacy and security concerns regarding smart devices in rented places. Both hosts and guests indicated the need for smart devices but had different concerns and expectations. The study found that 90% of hosts share Wi-Fi passwords with guests and that the method determines how often the password is changed or how it is chosen. The authors identified a need for usable mechanisms to manage and share passwords for Wi-Fi networks, services, and smart devices with guests. Many hosts routinely perform manual management tasks such as revoking access for past hosts, creating access for future hosts, and ensuring all devices are connected and configured correctly. Wi-Fi provided by a third party trusted by hosts and guests, e.g., by Airbnb, was proposed to mitigate concerns and liability due to guests' internet activity.

Geeng and Roesner [195] conducted a study in multi-user smart homes where smart devices are shared among multiple people. The study highlighted occupant change, i.e., when smart home fixtures remain physically with a home when people move, as a design challenge. Handling occupant change correctly in the full range of possible circumstances is challenging and may raise security concerns. A similar study by Zeng and Roesner [196] found that current smart home technology is not designed for interaction by multiple people.

Brush et al. [238] conducted interviews in 14 homes with home automation to asses positives, negatives, and barriers to broader adoption.

Cerf discussed the problem of configuration of network-enabled devices [244] and proposed a continuous configuration mode as a means to cope with system changes as devices come and go with the residents, guests, workers, and others [183].

7.6 The WIDE Enrollment Framework

Network enrollment is the process of creating a security association[6] between a networked system and a previously unknown device. The device and the system authenticate each other, and the system authorizes the device to access its network services and resources.

The enrollment process can be performed at different network layers. Link-layer enrollment authorizes the device to communicate with other devices on the same link. A separate enrollment process is typically executed on each link if the device has multiple link interfaces (e.g., Wi-Fi, Bluetooth, IEEE 802.15.4, Ethernet). In the literature, the term "network enrollment" often refers to link-layer enrollment. Network-layer enrollment authorizes the device for secure communication beyond the local link, i.e., across a wide-area IP network. Network-layer enrollment can be used to create security associations for protocols such as IPSec [245] or Transport Layer Security (TLS) [246]. Application-layer enrollment associates the device to a user account, network service, or application.

While layer-specific enrollment protocols can be performed independently, this can be undesirable from a usability perspective. The enrollment process must authenticate the device. The authentication step often requires user interaction. Reusing the authentication outcome across links and layer-specific enrollment protocols could improve usability and simplify the process. Consider the Amazon Echo, which performs Wi-Fi and application-layer enrollment at the same time [1]. Another example is Apple's MagicPairing protocol, which allows devices associated with the same user account to share a single Bluetooth pairing [199].

The simplest network enrollment scenario involves a device and a network that neither authenticate nor authorize each other. We refer to this scenario as *implicit network enrollment*. An open Wi-Fi network and a device automatically connecting to it perform implicit network enrollment. If the network requires a confirmation, e.g., via a captive portal [247], then this is considered explicit (not implicit) network enrollment.

[6]A security association refers to the security parameters and other data used to establish a cryptographically secure, mutually authenticated connection over an insecure communication medium.

The network enrollment process is rarely executed directly between a device and a network. Usually, one or more intermediary entities, e.g., a mobile device equipped with a UI or a specific out-of-band (OOB) communication channel, must complete different enrollment stages. Thus, general network enrollment can be viewed as a distributed process among multiple entities cooperating to generate a security association between the device and the networked system.

7.6.1 Device Lifecycle Model

Let us first consider the model lifecycle of a generic IoT device as shown in Figure 7.1. The model shows which entities have physical (trusted) access to the device and various important events during the device's lifetime.

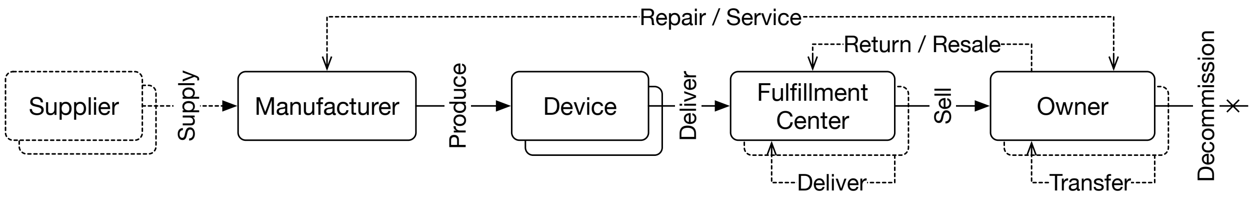


Figure 7.1: The model lifecycle of an IoT device. The device is manufactured, sold—optionally returned, resold, transferred from one owner to another, repaired or serviced—and decommissioned.

The life of a connected device begins at one or more suppliers (chipset manufacturers) who supply integrated circuits (ICs) such as processors, modems, or SoCs to the device manufacturer. The supplier has full access to the CPU, SoC, or secure element (SE) at this stage. This may include write access to information that becomes read-only later, including serial numbers, network interface hardware addresses, or security keys.

In the next stage, the manufacturer produces one or more devices. This step includes loading firmware into the device and initializing all components. The manufacturer has a secure MITM-resistant channel to the device at this stage and can thus produce a cryptographically protected device attestation certificate.

The manufacturer then ships the devices to a fulfillment center. The fulfillment center can be an online retailer such as Amazon, a brick-and-mortar store, or a combination of both. Some fulfillment centers might be able to personalize the device before the purchase. This step could include associating the device with a particular user account or pre-installing network credentials. Personalization is typically possible in online fulfillment centers with access to the purchaser's profile. Offline fulfillment centers usually do not have the personalization capability. Fulfillment centers are assumed not to have access to the physical device, which could be delivered in tamper-evident packaging. The center can access any device-identifying information accessible through the packaging, e.g., labels, stickers, or included NFC tags.

The device is eventually sold to a user (owner) for integration into their IoT system. The owner could optionally return the device for resale or transfer the ownership of the device to another owner. The owner could also send the device to the manufacturer for necessary servicing or repairs. Eventually, the owner decommissions the device when the device (a) is no longer needed, (b) stops working, (c) is stolen, or (d) is irrevocably physically damaged or compromised. There is no guarantee that the device will still be operational while decommissioning.

7.6.2 High-Level Protocol Overview

At a high level, the WIDE enrollment framework is similar to the existing link-layer-specific protocols reviewed in Section 7.5. The main differences in WIDE are the network-independent device registrar, protocols designed to minimize user interactions, and full device lifecycle support. Figure 7.2 provides a high-level overview of the architecture.

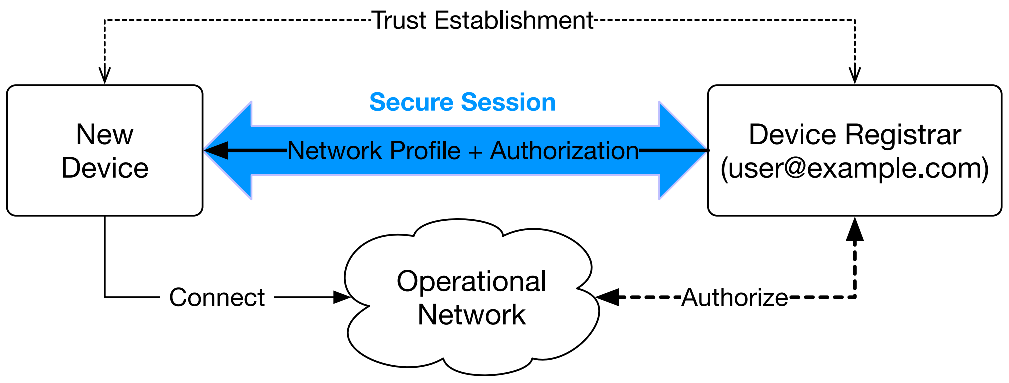


Figure 7.2: A high-level conceptual overview of the WIDE enrollment framework's architecture.

An *unregistered* device (without a security association to a registrar) automatically enters an enrollment mode and attempts to establish a secure session[7] with a device registrar. Once established, the device and the registrar create a mutual persistent security association. The security association binds the device with the registrar's user account. The device then transitions into a *registered* state.

The device requests a network profile and an authorization grant from the registrar over the secure session. The network profile represents all the information the device needs to discover and connect to the operational network. The authorization grant represents the user's (owner's) decision to grant the device access to the operational network.

Once the device has a network profile, it proceeds to discover the operational network using link-layer-specific discovery mechanisms. The device connects and submits its authorization grant to the network. The network verifies the grant and issues device-specific network access credentials. The device uses the credentials in all future connections to the network.

Once a device has been registered, it only communicates with its registrar for enrollment-related purposes. This communication can occur via any operational network available to the device or a temporary enrollment link or network. This allows the registrar to reconfigure the device if necessary, e.g., to provision a new network authorization grant into the device. It also ensures that the device remains reachable for provisioning at all times.

Upon reset to factory defaults or an explicit request from the device's registrar, the device deletes its registrar association and all network profiles and becomes unregistered again. If the device has been designed for deployments where physical security is hard or impossible, it may retain its registrar association across factory resets. In this case, the legitimate owner could delete the registration by submitting a secret revocation token to the device.

Previous paragraphs provide only a high-level conceptual overview, omitting many important details. How does the device discover the appropriate registrar to associate with? What networks does it use to connect to the registrar? How do the device and the registrar authenticate each other? How is the operational network associated with the registrar? We address these and other related questions in the following sections.

[7]A session that is mutually authenticated, confidential, and with integrity protection. The registrar strongly authenticates the device. The device weakly authorizes the device registrar.

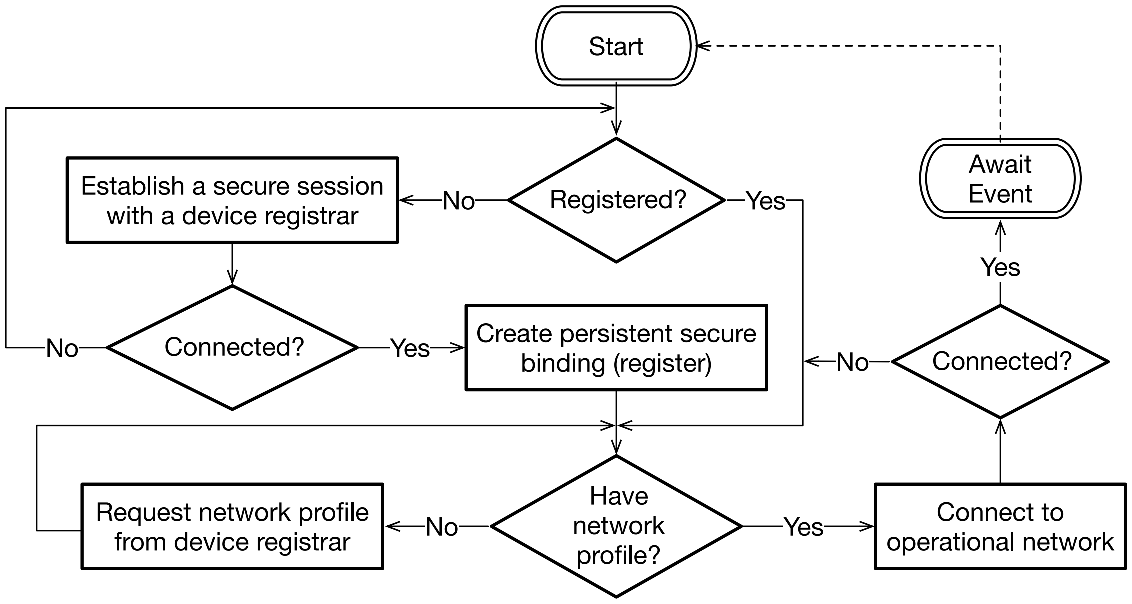

Start
Registered?
No
Yes
Establish a secure session with a device registrar
Await Event
Yes
Connected?
No
Yes
Create persistent secure binding (register)
No
Connected?
Request network profile from device registrar
No
Have network profile?
Yes
Connect to operational network

Figure 7.3: Device state transition diagram.

7.6.3 Trust Establishment

The purpose of trust establishment is for the device registrar to gather enough information to authenticate and identify devices securely. The device registrar maintains a database of trust anchors for that purpose. The trust anchors are ordinary X.509 self-signed certificate authority (CA) certificates [248]. The CA certificates represent various entities with physical access to the registrar's devices. We assume physical access to a device allows the entity to authenticate the device. Figure 7.4 shows a basic trust establishment scenario.

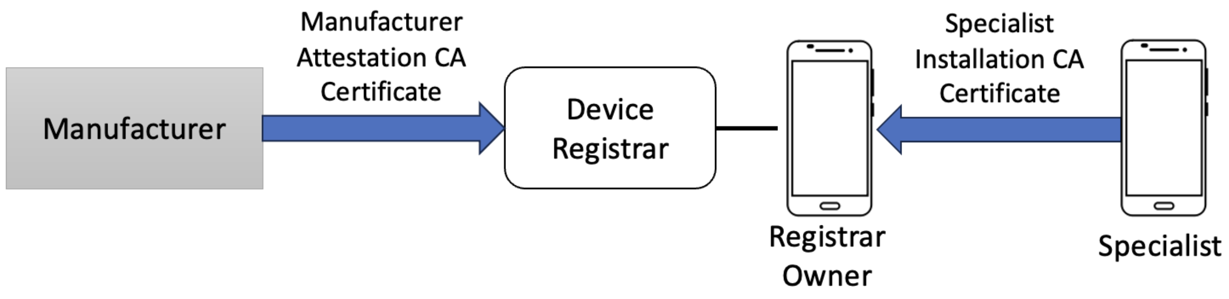

Manufacturer
Manufacturer Attestation CA Certificate
Device Registrar
Specialist Installation CA Certificate
Registrar Owner
Specialist

Figure 7.4: Establishing trust anchors in a device registrar

A device registrar could come preloaded with CA certificates of well-known manufacturers, similar to the CA lists embedded in modern operating systems or web browsers. The registrar's owner could also manually import or delete a trust anchor, for example, when purchasing a device from a new manufacturer.

The registrar keeps track of two CA certificate types: manufacturer attestation CA certificate and specialist installation CA certificate. The manufacturer uses the former to sign device attestation certificates. The specialist uses the latter during device installation to produce a signed device installation certificate. We assume the specialist CA certificates are manually imported by the registrar's owner, for example, via NFC.

7.6.4 Device Discovery

Consider the scenario shown in Figure 7.5. Suppose the new device has a registration authorization in the form of a public-private key pair. The public key represents the device's identity to register with a device registrar. The knowledge of the registration authorization authorizes the entity (device registrar) to register the device. The device does not disclose the public key on any insecure network interfaces.

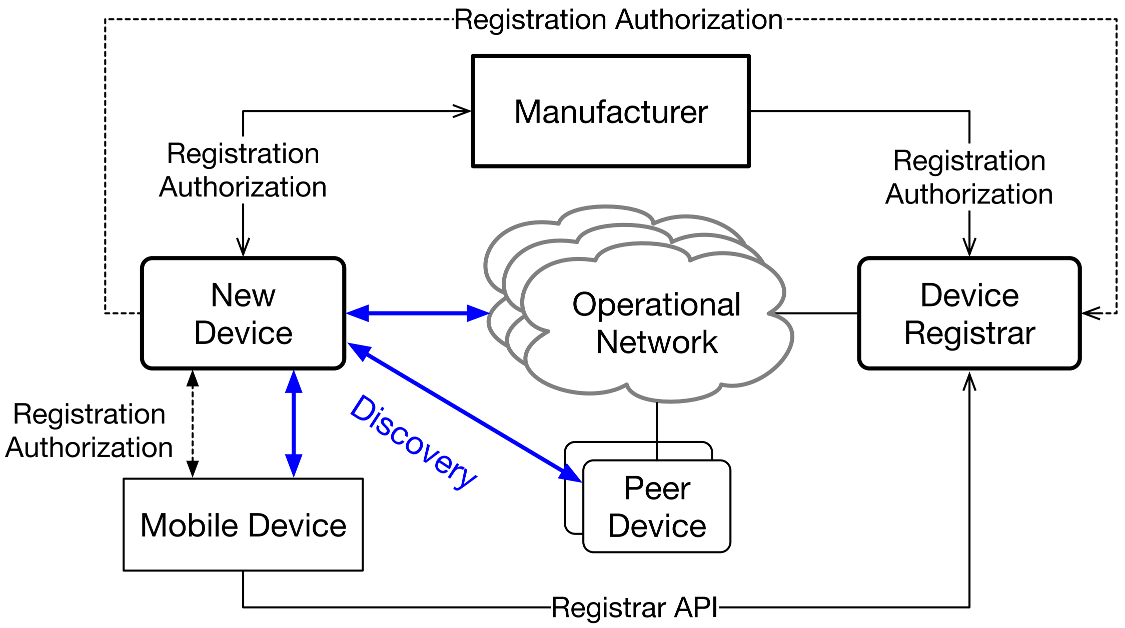

Registration Authorization
Manufacturer
Registration Authorization
Registration Authorization
New Device
Operational Network
Device Registrar
Registration Authorization
Discovery
Mobile Device
Peer Device
Registrar API

Figure 7.5: To discover each other, a new device and a device registrar rely on existing operational networks, an auxiliary mobile device, or peer devices associated with the same device registrar.

Suppose the device registrar has received the registration authorization for the new device. The registrar could receive the authorization using an automated mechanism from the manufacturer; for example, the authorization could be automatically sent to the owner as part of a purchase receipt and automatically imported into the registrar. If the device was purchased offline, the authorization is included in the packaging and manually imported by the user. The registration authorization could also be read by an auxiliary mobile device by the specialist installing the device, e.g., via NFC, QR code, or a similar approach.

The device registrar maintains a list of devices with known registration authorizations that have not been registered yet. If the list is not empty, the registrar advertises the devices via all of its known (authorized) operational networks. The advertisements use a link-layer-specific mechanism, e.g., on Wi-Fi, the devices the registrar is looking for could be advertised using vendor-specific Wi-Fi beacon elements.

To preserve the confidentiality of the registration authorization public key, the registrar advertises a hash of the public key constructed as follows:

```
advertisement = SHA256(registration_authorization | "wide:advertisement")
```

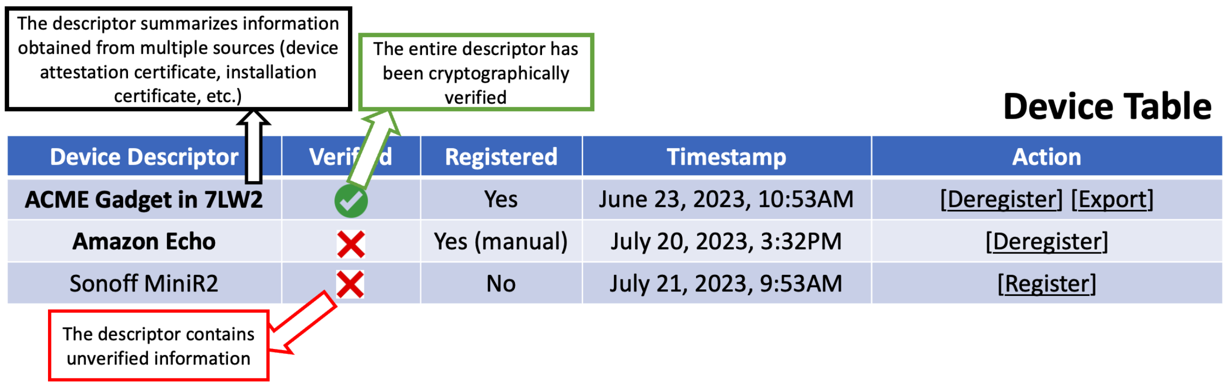
A new device seeking to register with a registrar uses link-layer-specific mechanisms to scan its network neighborhood for advertisements for its registration authorization from device registrars. If it finds an advertisement matching its registration authorization, it starts the device registration protocol with the registrar. Since multiple device registrars can advertise the same registration authorization, the device tries registering with each of them sequentially until the registration succeeds.

7.6.5 Device Registration

Upon discovering a device registrar advertising the possession of the device's registration authorization key, the device attempts to register with the device registrar. The device does not have access to any operational networks at this stage, which makes it challenging. The design of the device registration protocol impacts scalability and usability.

7.6.6 Device Authentication

Suppose the device registrar internally maintains a table of discovered, registered, and unregistered but known devices. Figure 7.6 shows a mockup of the device table.



The descriptor summarizes information obtained from multiple sources (device attestation certificate, installation certificate, etc.)

The entire descriptor has been cryptographically verified

Device Table

Device Descriptor	Verified	Registered	Timestamp	Action
ACME Gadget in 7LW2	✓	Yes	June 23, 2023, 10:53AM	[Deregister] [Export]
Amazon Echo	✗	Yes (manual)	July 20, 2023, 3:32PM	[Deregister]
Sonoff MiniR2	✗	No	July 21, 2023, 9:53AM	[Register]

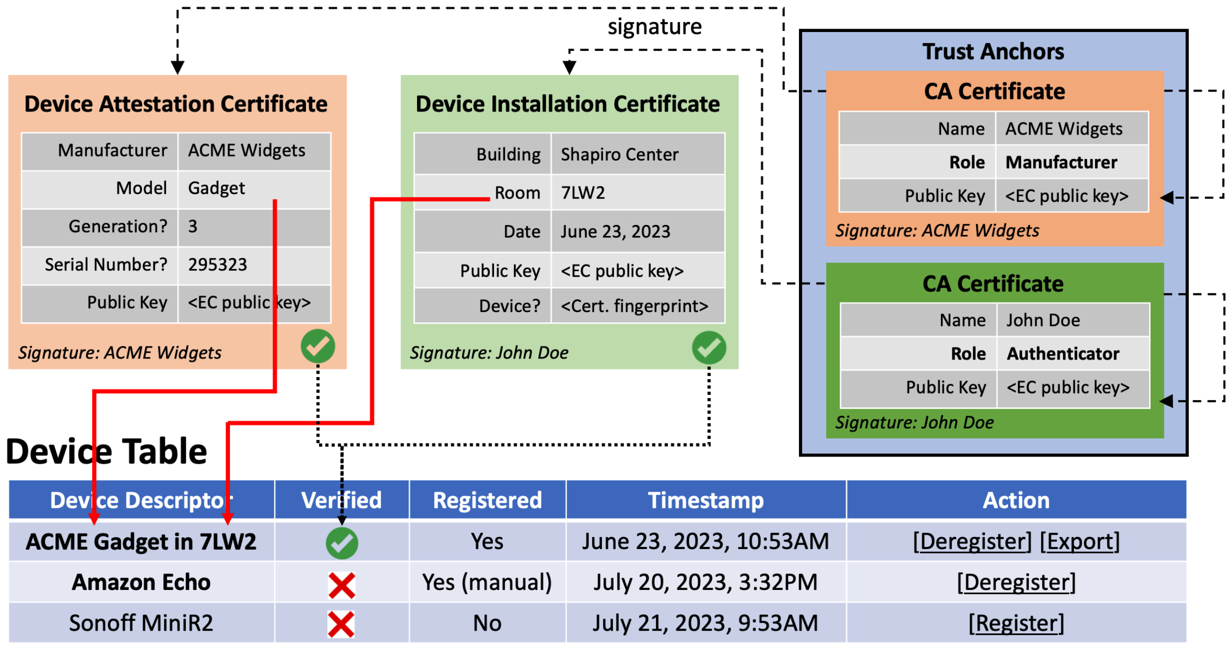
The descriptor contains unverified information

Figure 7.6: Device table maintained by the device registrar

The registrar derives a unique, human-readable device descriptor for each device (left-most column). The descriptor is intentionally human-readable so that it is easy to read and understand by the user and could be communicated to the user via other modalities such as audio (e.g., using text-to-speech (TTS)). The purpose of the device authentication phase is to authenticate all information (attributes) that is included in the device descriptor.

Consider the device descriptor “ACME Gadget in 7LW2” on the first row in Figure 7.6. The term “ACME” represents the manufacturer’s name, “Gadget” represents the name or ID of the product, and “7LW2” is the name of a location (in this case, a room) meaningful to the user. Thus, this particular descriptor contains three attributes: manufacturer name, product name, and location.

The device registrar obtains the values for these attributes from different sources. The manufacturer and product name values originate from a device attestation certificate (DAC) imprinted into the device by the manufacturer. The location attribute could originate from a device installation certificate (DIC) generated by an installation specialist trusted by the registrar. If the device registrar cryptographically verifies all attribute values in the descriptor, it assumes the device has been authenticated. Figure 7.7 illustrates this scenario.



Device Attestation Certificate

Manufacturer	ACME Widgets
Model	Gadget
Generation?	3
Serial Number?	295323
Public Key	<EC public key>

Signature: ACME Widgets ✓

Device Installation Certificate

Building	Shapiro Center
Room	7LW2
Date	June 23, 2023
Public Key	<EC public key>
Device?	<Cert. fingerprint>

Signature: John Doe ✓

Trust Anchors

CA Certificate

Name	ACME Widgets
Role	**Manufacturer**
Public Key	<EC public key>

Signature: ACME Widgets

CA Certificate

Name	John Doe
Role	**Authenticator**
Public Key	<EC public key>

Signature: John Doe

Device Table

Device Descriptor	Verified	Registered	Timestamp	Action
ACME Gadget in 7LW2	✓	Yes	June 23, 2023, 10:53AM	[Deregister] [Export]
Amazon Echo	✗	Yes (manual)	July 20, 2023, 3:32PM	[Deregister]
Sonoff MiniR2	✗	No	July 21, 2023, 9:53AM	[Register]

Figure 7.7: Device descriptor is computed from attributes obtained from device attestation and device installation certificates. The device becomes authenticated if both certificates are cryptographically bound to trust anchors.

The device registrar keeps a set of trust anchors, i.e., CA certificates that the registrar implicitly trusts. The CA certificates can be self-signed. The manufacturer CA certificate (orange background) represents the CA used by a device manufacturer to sign device attestation certificates. The authenticator CA certificate represents a third-party entity that installs devices, e.g., a specialist. Upon device installation and configuration or calibration, the specialist generates a device installation certificate signed by their authenticator CA certificate.

In Figure 7.6, an authenticated device is shown with a green check mark in the column “Verified”. If any attribute values cannot be cryptographically verified, e.g., if one of the values originated from an unsecured service discovery protocol, then the device is assumed to be unauthenticated.

The registrar keeps track of the authentication status for each known device. Authenticated devices are subject to less stringent policies; for example, they could be enrolled into operational networks automatically without the user’s explicit approval. An unauthenticated device would require explicit approval from the user (device registrar owner).

7.6.7 Operational Network Connection

Suppose the device has been provisioned with a network profile for each enrolled network. The profile aggregates all necessary information to discover, select, connect, and authenticate the network. Figure 7.8 shows one possible organization of network profiles and related data structures in the device.

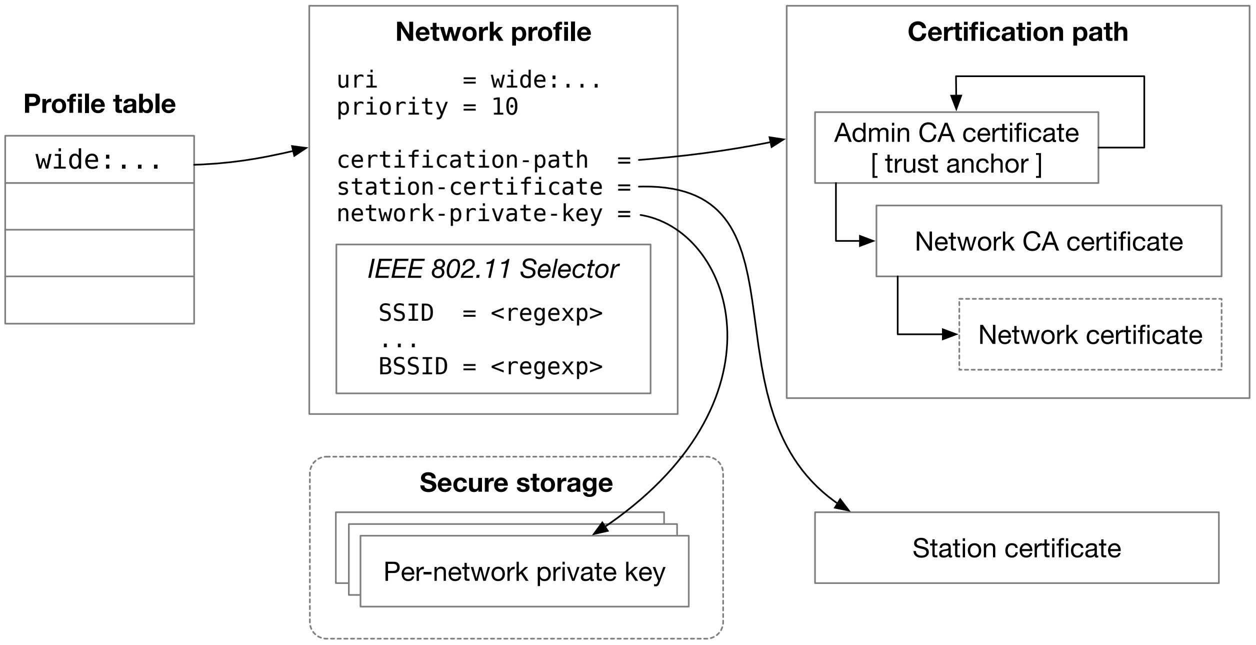


Figure 7.8: Network profile data maintained by an enrolled device

A uniform resource identifier (URI) uniquely identifies each network profile. The URI consists of an (optional) network name, mandatory CA public key identifier, and optional URI parameters. The network name and CA key identifier are cryptographically verified during network authentication. The URI parameters provide network selection hints and can also be used to override selected attributes from the stored network profile.

The following rules in the augmented Backus-Naur form (ABNF) [249, 250] describe the syntax of the URI in detail. The nonterminals `ESCAPED` and `ALPHANUM` originate from RFC 2396 [251].

```
WIDE-URI = %s"wide:" network-id uri-parameters


network-id = [ name "@" ] authority
```

```
name = 1*( unreserved / ESCAPED )
authority = ALPHANUM / "-" / "_"

uri-parameters = *( ";" parameter )
parameter = parameter-name [ "=" parameter-value ]
parameter-name = 1*( unreserved / ESCAPED )
parameter-value = 1*( unreserved / ESCAPED )

unreserved = ALPHANUM / "-" / "_" / "." / "!" / "~" / "*" / "'" / "(" / ")" / "["
    / "]" / "/" / ":" / "&" / "+" / "$" / "?" / "=" / "," / "`" / "#" / "^" / "{" / "}"
    / "|" / "\" / "<" / ">"
```

In the simplest form, the network profile URI contains only a public key identifier identifying the network. Additionally, the URI can contain parameters that inform the device's network selection process.

```
wide:n8MoU0fp6WNoSIFm3kV3xTcMqgtq3eFf-_gSfZtQx6A
wide:Columbia%20University@H1tOR3YI_CQgxGyYjoN5eDCWuKY0_Qy4nBq6EmOSO08
wide:CU%20Infrastructure@lQ8Q2W5G5kh9StSifcaBi_mbZqV6hsXohwe8cgsQ27c;band=2.4
    ;SSID=Columbia%20University
```

The network profile can also include a selector with expressions to match various network parameters. In the case of Wi-Fi, these could be SSID, BSSID, or other IEEE 802.11 attributes. The selector helps the device discover and prioritize the AP to associate with and authenticate. Other information could include the preferred RF band or channels. The selector has an advisory role only. If it yields multiple matching IEEE 802.11 networks, the device searches for the appropriate network by checking its certificate during authentication.

Each operational network has a certification path [248]. The certification path consists of the network's certificate and zero or more CA certificates. The simplest certification path consists of a self-signed network certificate. When the network certificate is not self-signed, the certification path includes any intermediate or trust anchor CA certificates needed to validate the network certificate. The certification path should be kept reasonably short [252].

The device has been provisioned with a client certificate, which includes the device's public key designated for the network. The client certificate is not necessarily signed by the same CA that signed the network certificate. It may be self-signed, or it could have been signed by some other CA used by the network to validate station certificates.

The device maintains a unique private key for each enrolled network. Ideally, the private key was generated locally and has never left the device. Private and corresponding public keys must not be reused across networks to prevent device tracking. We assume the device can store the private key securely, e.g., in a hardware security module (HSM). If no HSM is available, the device's firmware should use a least-privilege design to restrict access to the private key only to the software components that need it.

7.6.7.1 Network Authentication

The device authenticates to the operational network with its client certificate using the TLS protocol version 1.3 [253] or newer. We specifically choose TLS 1.3 for its privacy-preserving and forward-secrecy properties. TLS 1.3 transmits the client certificate encrypted to the server. Thus, using the client certificate for surveillance or tracking is difficult.

The device consults its local configuration to discover the target network to connect to. The network must advertise IEEE 802.1x [223] support. The device associates with the network and waits for the network's AAA server to initiate an EAP-TLS [254] authentication session. The device uses an anonymous network address identifier (NAI) [255] in the EAP Identity message. Since TLS 1.3 transmits the client certificate encrypted, there is no need to use a combined outer-inner EAP authentication scheme such as PEAP-EAP-TLS. This simplifies the authentication process and limits the number of required round trips.

The device validates the server certificate received from the network. The server certificate must be signed by the network's CA certificate previously provisioned into the device for this network.

The device associates with the operational network, which initiates a new EAP-TLS session to authenticate the device. The device verifies the server certificate presented by the network and sends its client certificate to the server for authentication. After successfully running the EAP-TLS protocols, the device and the network are mutually authenticated.

7.6.8 Device Deregistration

Deregistration is the process of securely removing the device from the networked system performed when the device (a) is no longer needed, (b) stops working, (c) is stolen, or (d) is irrevocably physically damaged or compromised. Once deregistered, the device is no longer associated with its registrar and cannot access any of its operational networks anymore. Deregistration can be triggered either from the device or the registrar (or both).

A reset to factory default or a user-directed action performed through the UI triggers a deregistration on the device. The device disconnects from any connected operational networks and securely erases all sensitive information, including the private keys corresponding to the device's registration and network certificates. If the device has any network grants, those must also be erased. De-registration through factory reset also supports scenarios where the original owner forgets to de-register the device while reselling or donating it.

If the device is no longer operational, the deregistration process must be performed through the device registrar, for example, triggered by the user in the registrar's UI. The registrar revokes the device's registration certificate and any network certificates but preserves the device's authentication information for potential reuse if the device re-registers later[8]. This step prevents the device from joining any operational networks without the user's explicit approval, but the device will remain known (authenticated) to the registrar.

7.7 Prototype Implementation

We implemented an early proof of concept prototype of the WIDE enrollment framework. The prototype used Wi-Fi as the operational network technology, Raspberry Pi with an external Wi-Fi card for the device, an Intel Next Unit of Computing (NUC) as a Wi-Fi access point and AAA

[8]Device authentication often requires user interaction, which can be tedious and time-consuming.

server, and a custom iPhone application. This section describes the prototype. Figure 7.9 shows the hardware configuration of the prototype.

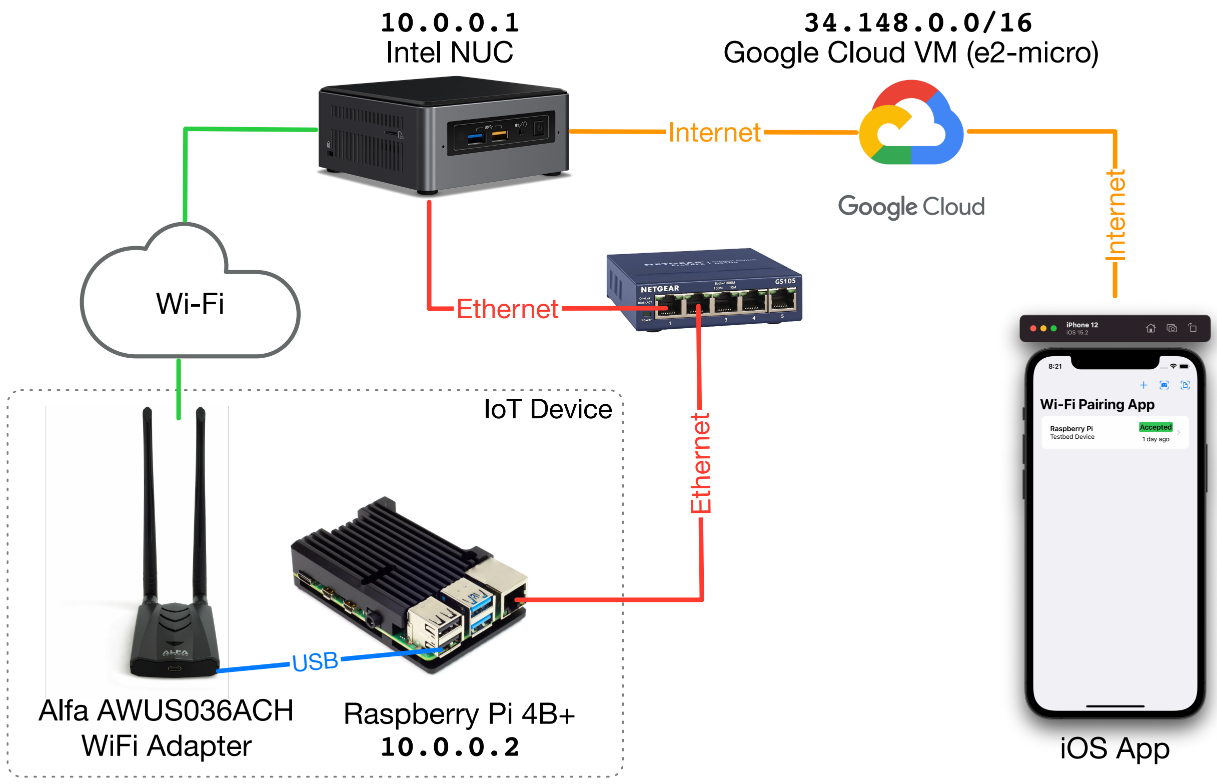
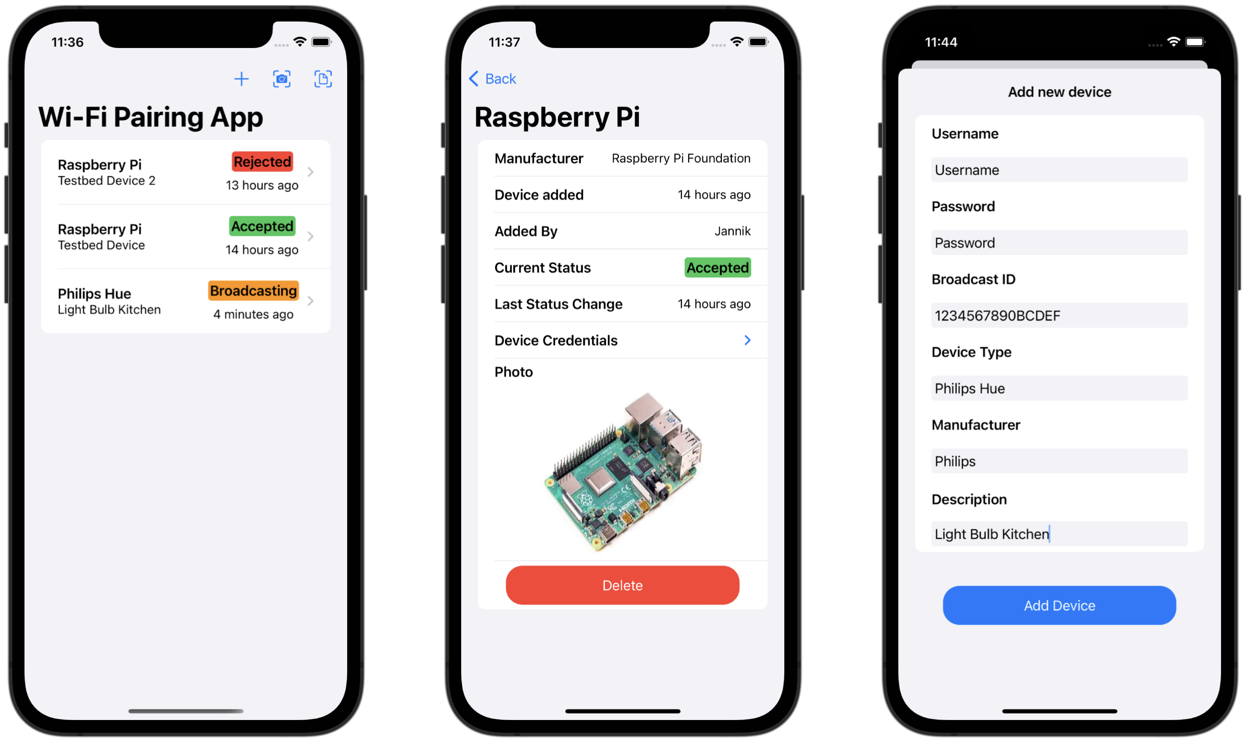


Figure 7.9: Early proof of concept prototype of the WIDE enrollment framework. Author: Jannik Wortmann. Used with permission.

We used a Raspberry Pi 4 Model B with built-in 2.4 GHz and 5.0 GHz IEEE 802.11ac Wi-Fi, Bluetooth 5.0, and an Ethernet port. We used an external Wi-Fi card with support for the monitor mode. This enables the device to eavesdrop on any traffic on a particular channel. The device requires this capability to monitor and analyze available Wi-Fi networks and vendor-specific advertisement payloads. Besides the AP and IoT device, the prototype included a device registrar running in the Google Cloud.

The authenticator and management UI was implemented as an iPhone app. Since the app only serves as a user interface, this could be implemented on almost any platform with an active internet connection. Figure 7.10 shows the user interface.

The diagram in Figure 7.11 then shows a detailed architecture of the whole prototype.



Raspberry Pi Testbed Device 2	Rejected	13 hours ago
Raspberry Pi Testbed Device	Accepted	14 hours ago
Philips Hue Light Bulb Kitchen	Broadcasting	4 minutes ago

Raspberry Pi

Manufacturer	Raspberry Pi Foundation
Device added	14 hours ago
Added By	Jannik
Current Status	Accepted
Last Status Change	14 hours ago
Device Credentials	
Photo	

Username	Username
Password	Password
Broadcast ID	1234567890BCDEF
Device Type	Philips Hue
Manufacturer	Philips
Description	Light Bulb Kitchen

Figure 7.10: WIDE enrollment framework iOS application. Author: Jannik Wortmann. Used with permission.

The device registrar and network server running in the Google Cloud provides a standard RADIUS service to the Wi-Fi network. The AP in the Wi-Fi network was configured in WPA-Enterprise (IEEE 802.1X) mode and forwarded EAP requests over RADIUS to the cloud component. We also implemented a custom protocol between the AP and the cloud that allows the AP to obtain a list of devices to search for.

The communication protocol between the iOS application and the cloud network server used an ordinary push service provided by Apple for the iOS platform. The push service triggered UI updates and notified users of new devices requiring authorization to be registered and enrolled.

7.8 Human-Subject Usability Study

The traditional method to evaluate the usability of a computer system is via a human-subject usability study in a controlled environment. This section describes our effort to design a study to

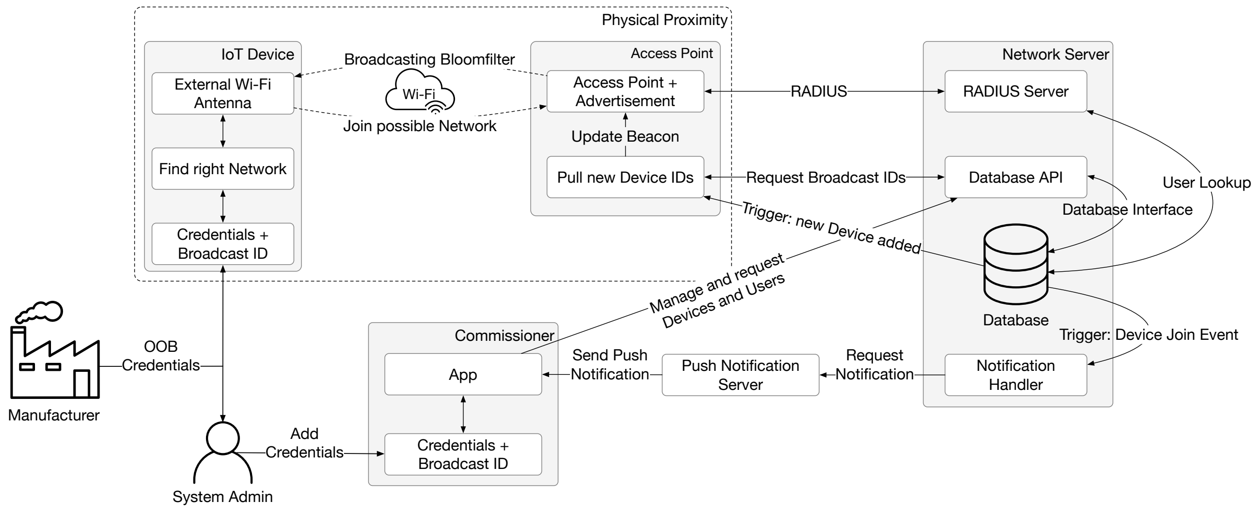


Figure 7.11: Proof of concept system architecture. Author: Jannik Wortmann. Used with permission.

evaluate the usability of various network enrollment solutions for consumer-oriented IoT systems. The primary research question being investigated is “How usable (difficult, error-prone) is the network enrollment process of consumer IoT devices for non-technical users.” This experiment seeks to answer the question by obtaining a usability evaluation through participant results and survey responses.

We describe the design of the study, our human-subject recruitment plan, and summarize our experience from a pilot trial with seven commercially-available IoT devices, cheap and more expensive, from a variety of vendors. Based on research and past experience, we hypothesize that the participants in the study will encounter difficulties while attempting to connect devices and report that they think the system could/should be improved.

7.8.1 Methodology

In the study, human-subject volunteers are given written instructions that ask them to perform network enrollment with a variety of commercial IoT devices. Each participant is randomly assigned three IoT devices to enroll in the campus Wi-Fi network. The maximum per participant is 30 minutes. The collected data would then be used to evaluate the usability of the network enrollment protocols involved in the study.

We recruit a mix of technical and non-technical human-subject volunteers. Participant age, technological experience, method of recruiting participants, and incentivizing participation are key factors. Age affects the amount of experience and comfort level with technology; young people who grow up with technology will probably have more experience with technology and have more technology intuition, even if they are unfamiliar with a specific type of technology. We aim to have some participants with very little to no smartphone experience, as the general Wi-Fi enrollment process is very smartphone-driven.

At the beginning of the experiment, each volunteer is asked to complete a pre-experiment self-evaluation survey (shown in Figure A.2). The survey asks about their background, general experience and skill level with technology, the type of technology they regularly use, and whether they have ever used the devices assigned to them in the study. We also ask them to assess beforehand how well they expect to perform in the experiment. The data collected via the pre-experiment survey helps establish a baseline of information about the participant.

The main experiment procedure is the same for each device being enrolled. Each participant attempts to connect up to three devices in the allotted time. For each device, the participants are provided the device and a description of what it does, any necessary equipment (outlets, power cords, monitors, speakers), and a device documentation sheet containing either written instructions for network enrollment or a website link to the instructions. We record the entire main experiment procedure and ask each subject to "think aloud" through the experiment, mentioning any steps that seem counter-intuitive or confusing.

Once the participants have finished the network enrollment procedure with all three devices, they take a post-experiment survey (similar to the pre-experiment survey). The post-experiment survey includes questions about the experiment process for each device they enrolled. This survey aims to collect information indicating which and when errors (if any) occurred, how well the participants performed, and how well they think they performed. The questions are written so that the answers provide both the quantitative and qualitative information needed. The post-experiment survey form can be seen in Figure A.3.

The participants will be recruited from the Columbia University community with varying technical experience, trending towards less experience, and an anticipated age range of 25 to 60. Sex, race, gender, ethnicity, and class will be diversely represented to gain an accurate picture. The subject population is designed to represent all people who use internet-connected devices accurately. The population will be focusing on those who are older and/or have less technical experience, as they are likely to be the ones who have the most trouble connecting the devices.

Based on similar studies performed in the literature, we expect to hire approximately 20 to 30 participants in each round of experiments. Each volunteer tests three devices. This gives us 60–90 data points, which is large enough for a sample population.

During the study, we collect the following data:

- **Completion time**: How long it takes for the test subject to complete enrollment;
- **User mistakes**: The number of mistakes made, including incorrect passwords, steps in the wrong order, and restarting the process.
- **Non-user mistakes**: errors that are not caused by the user. Directions that do not match the application UI or documented process, Wi-Fi connectivity not working, application crashing.
- **Correctness**: Whether the test subject successfully completes the entire enrollment process in the allotted time.

The collected data will be anonymized. Furthermore, we would ask each test subject to rate the following criteria on a scale from 1 to 10:

- Overall process;
- Clarity of instructions;
- Ease of the enrollment process;
- Complexity.

7.8.2 Pilot Trial

This section summarizes the experience from a pilot trial of the usability study. The experiment was performed by a Columbia University student volunteer who enrolled seven devices to a Wi-Fi network. The student was unfamiliar with any of the devices before the experiment. We first summarize the experience, followed by a description in narrative form.

Table 7.2: Devices used in the pilot study

Manufacturer	**Model**
Google	Chromecast
Google	Chromecast Audio
Smart Life	Light Bulb
Amazon	Echo (Alexa)
VOCOLink	Light Bulb
EZVIZ	Camera (Mini O)
Google	OnHub Wi-Fi AP

- Every device required a separate app to download, which took up too much space on the smartphone.
- Each app interface was different, and it took too much time to figure out how to navigate the UI for each app.
- Every app requires a separate username and password that the user would have to remember.
- All apps except one required manual entry of Wi-Fi credentials inside the app (this is a security restriction on the Android platform).
- Several apps had issues discovering the device (Alexa, VOCOlinc light bulb, Google Home). This seems to be a recurring problem with vendor-specific enrollment protocols.

Each tested device required a separate smartphone application to enroll the device to the Wi-Fi network. With seven tested devices, the seven apps took up so much space on the iPhone that storage settings had to be reconfigured. If someone wanted to connect devices across an entire smart home, they would likely have too many apps to fit on their phone.

As each app interface was different, it took a significant amount of time to figure out how to navigate the UI for each app. This would likely present a challenge for non-technical users. Several apps also had a UI that looked different from the pictures shown in the enclosed documentation, which was confusing. Every app requires a username and password to log in at some point during the process. Some apps allowed a pre-existing Google or Amazon account, but the rest required a new account specific to the app. This means a heterogeneous IoT system user would have too many usernames and passwords to remember.

Attempting to pair Alexa with a light bulb seemed unnecessarily complex. The process required both devices to be paired with separate apps on the same phone. The user had to log into the light bulb app and choose a tab to add Amazon Alexa, which opened the Alexa app. Inside the Alexa app, the user had to search for the light bulb brand inside a “skills” tab and choose the light bulb from the results, clicking “enable.” The user then had to sign into their account for the light bulb again.

7.9 Summary

Experience with existing consumer IoT devices suggests that reliable network enrollment is still an open problem that deserves research. Existing solutions generally lack the scalability, flexibility, or security to support large-scale or heterogeneous IoT systems. The deployment model of vendor-specific solutions is commonly limited to self-installation scenarios where a single entity, usually the network administrator, performs all network enrollment steps.

We reviewed many existing enrollment frameworks for various link-layer technologies and discussed several advanced deployment scenarios poorly served by the existing protocols. Our focus was explicitly on consumer IoT systems. We then developed a collection of assumptions and requirements that must be met in the advanced deployment scenarios.

We developed WIDE, a link-layer independent device enrollment framework for cyber-physical systems. Unlike existing solutions, WIDE supports role separation, devices installed by third parties, multiple link-layer technologies, and considers the device’s entire lifecycle. We implemented an early proof-of-concept prototype and described its architecture.

We designed an IRB-approved human-subject usability study to evaluate the usability of several common vendor-specific network enrollment protocols. We conducted a pilot test of the study with a small number of devices.

Together with the work described in Chapters 6 and 8, this work can be viewed as an attempt to understand, simplify, and automate network management in large-scale or heterogeneous cyber-physical systems.

7.9.1 Future Work

Due to pandemic-related restrictions and the general logistical difficulty of managing human-subject experiments on campus, we have not yet been able to conduct the usability study described in Section 7.8. This is left for future work. We also plan to develop a more comprehensive prototype of the WIDE enrollment framework and compare it with other enrollment frameworks in the usability study.

7.9.2 Acknowledgments

We thank Jannik Wortmann, a visiting student from RWTH Aachen University, for helping with the implementation and evaluation of the prototype. We thank Rosie Torola and J.J. Shaw, both Columbia University students, for their help with the usability study design.

Chapter 8: Authenticating Physically Inaccessible Devices via Visible Light

This chapter presents the design, implementation, and experimental evaluation of a visible light communication-based (VLC) authentication protocol for physically inaccessible IoT devices. The protocol complements existing authentication protocols based on QR codes, NFC, or shared secrets. The authentication protocol was developed for Wi-Fi (IEEE 802.11) IoT devices and can be used as an additional bootstrapping method in the Device Provisioning Protocol (DPP) framework [184]. The protocol could be adapted to other link layer technologies with minimal effort.

8.1 Introduction

Network enrollment is one of the first steps in the lifecycle of any IoT device. Network enrollment usually involves authenticating and authorizing the new device before it can connect to the network infrastructure. The exact mechanisms and protocols used in the enrollment process depend on the characteristics of the underlying network technology, device form factor, and application domain. A critical step in the network enrollment process is device authentication. The device's authenticity must be established before an IoT device can be connected to a secure network. Device authentication helps to prevent setup mistakes (e.g., the wrong device being assigned to a role) and eliminates certain types of network-based attacks, such as device impersonation. The authentication step is generally interactive and involves the user or a third party setting up the device.

Existing device authentication methods generally rely on the presence of the so-called MITM-resistant out-of-band (OOB) communication channel[1] between the device and an entity trusted by the secure network. The trusted entity is often another device under the control of the end-user,

[1]The term out-of-band (OOB) channel refers to a communication channel other than the primary (Wi-Fi) communication network. The OOB channel could use a variety of mediums, such as ultrasound, low-range RF communication, or visible light.

commonly a smartphone, that is already authorized to the network. The device being authenticated reveals a secret to the trusted entity over the OOB channel. If the channel is resistant to active (impersonation) and passive (eavesdropping) MITM attacks, an adversary cannot obtain access to the secret. If the trusted entity wishes to authenticate the device over the main (insecure) network interface, it performs a key agreement authenticated with the secret revealed to it by the device. If successful, the device participating in the key agreement knows the secret, and its authenticity has been established.

Many secure MITM-resistant OOB channels and protocols have been proposed and analyzed in the literature. Existing OOB channels rely on limited signal propagation, cryptography, or a combination of both to minimize the risk of a remote adversary gaining access to the communication (secret). An OOB channel based on limited signal propagation forces the adversary to be physically close to the device, thus revealing themselves. Such channels can rely on sound, light, low-power RF, or selected properties of the physical environment for communication. Cryptographic OOB channels use established cryptographic protocols (e.g., encryption) to conceal the secret.

Existing authentication mechanisms implemented by off-the-shelf IoT devices generally assume that the end-user has physical access to the device. The authentication step could involve reading the device's PIN or serial number, scanning a QR code, tapping an NFC or RFID reader, pressing buttons simultaneously, or connecting to the device with a USB cable.

Many of these methods (e.g., NFC) require relatively expensive additional hardware. This may not be economical for many types of devices, especially those installed in bulk, such as smart light bulbs. Methods based on Bluetooth or ultrasound communication trade convenience (authentication over a distance) for security and may only be usable in controlled environments where the risk of a MITM attack could be characterized.

Future IoT systems will likely include devices that are physically inaccessible to the user. Yet, the end-user may need to be able to authenticate such devices. Examples include Wi-Fi-connected smoke detectors mounted under a high ceiling, light bulbs, fans, air conditioning devices, and many other types of devices commonly installed as part of a fixed building or apartment infrastructure. Existing authentication methods based on PINs, buttons, QR codes, or NFC will likely be inconvenient or unusable with such infrastructure.

Regardless of the form factor, virtually all modern IoT devices are already equipped with an LED status indicator. All modern smartphones are equipped with a camera. We propose an authentication protocol for physically inaccessible IoT devices based on VLC. To authenticate the device, the user points his or her smartphone camera at the device. The device transmits a secret by modulating its LED indicator. The user keeps the smartphone pointed at the device until enough information has been received to reconstruct the secret successfully. The smartphone then uses the secret in a password-authenticated key agreement performed over the primary network interface, in our case, Wi-Fi.

The rest of the chapter is organized as follows. Section 8.2 motivates our work and describes the problem being addressed in more detail. In Section 8.3, we list the various requirements the protocol must meet to be practically helpful. Section 8.4 illustrates the overall system architecture and the roles of various subsystems. Section 8.5 describes the protocols running over the Wi-Fi and visible-light channels. The design of the prototype implementation is described in Section 8.6. Section 8.8 presents an experimental evaluation of the visible-light component (protocol). In Section 8.9, we summarize the related work. Finally, Section 8.10 summarizes the results.

8.2 Motivation and Problem Description

As deployed IoT systems grow more complex over time, they will likely integrate a wider variety of devices. Most of the IoT devices on the market as of 2022 are designed for self-installation by the end-user (consumer). Extending the application domain of future IoT systems will necessitate professionally installed devices. Such devices could include smoke detectors, lighting controllers, air conditioning, and, in general, any devices that are part of a fixed house or apartment infrastructure.

Professionally installed IoT devices may still require integration into an IoT system operated by the end-user (consumer), which may require connecting the device to the end-user Wi-Fi network, i.e., to an AP provided by the end-user. If the device is installed in a remote or otherwise inaccessible location, the end-user may be unable to use conventional (proximity-based) device authentication methods.

We wish to design an authentication protocol for remote (physically inaccessible) IoT devices to support such scenarios. The protocol should allow authenticating a device over a short distance (e.g., a large room) using readily available components (LEDs) and a handheld smartphone. Ideally, the authentication process would be as fast or faster than the steps necessary to obtain physical access to the device, for example, by using a ladder or other tools.

The VLC authentication protocol is designed to be used in addition, not instead of other authentication methods. For example, if the device manufacturer provides a digital certificate to the end-user to assist with device authentication, that method is generally preferable to VLC authentication. However, we assume this method may not always be available, for example, if the device was resold or rented or the digital certificate was lost. The VLC-based authentication protocol is designed as a viable alternative for such cases.

8.3 Assumptions and Requirements

Camera Hardware Diversity. We assume the smartphone is equipped with an ordinary CMOS camera. We require no particular frame rate, resolution, or camera sensitivity. Low frame rate or camera sensitivity would merely slow the authentication process down but not entirely prohibit it.

End-user Assistance. In most situations, the end-user will hold the smartphone during authentication. Thus, the authentication application needs to either automatically locate the transmitting (blinking) device or provide a viewfinder-like functionality so that the user can correctly position the smartphone.

Range. The target range is “across the room”. The VLC protocol should adapt to distance or suboptimal lighting conditions. If the device is too far away or if there is too much background noise (ambient light), the authentication process should slow down but not completely fail.

Transmitter LED. The transmitter (IoT device) is equipped with an LED, either monochromatic or (preferably) tri-color (RGB). The form factor of the device or its location may not allow for any other information to be shown (labels, displays, etc.). The LED may also serve as a status indicator. It must be controllable from the application (firmware) and must transmit in the visible-light spectrum (i.e., no infrared).

Network Interface. A high-bandwidth (potentially insecure) network interface exists between the device and the smartphone. The device and the smartphone can directly communicate over the network interface. In our prototype, we assume both devices support Wi-Fi (IEEE 802.11) and the Device Provisioning Protocol (DPP) framework [184]. The idea is generalizable to any network technology with sufficient bandwidth for an authenticated key agreement based on Elliptic curve cryptography.

Physical Access. The authentication process must not require physical access to the device from the user. The device may be in an unreachable location, e.g., permanently mounted under the ceiling. Special tools or privileges may be required to physically access the device in some cases.

Authentication Completion Time. Authentication running times on the order of tens of seconds are acceptable. Maximizing the bandwidth of the VLC channel is a non-goal. Assuming reaching the device takes some time, the goal is to develop a comparatively faster system.

8.4 Architecture

Figure 8.1 illustrates the overall system architecture. The Lighthouse system consists of one or more IoT devices and a trusted entity (authenticator) in the form of a camera-equipped smartphone. In the figure, the IoT devices are represented by a smoke detector. The IoT device may be physically unreachable (e.g., a smoke detector fixed under a high ceiling), is equipped with a Wi-Fi network interface, and has a visible tri-color (RGB) LED indicator visible to the authenticator. We assume a line-of-sight exists between each device and the authenticator.

Each device is uniquely identified by an elliptic curve public key. The device generates the corresponding private key locally and keeps the key reasonably secure from disclosure. Suppose the device has other identifying information, e.g., a serial number, unique code, or PIN printed on the

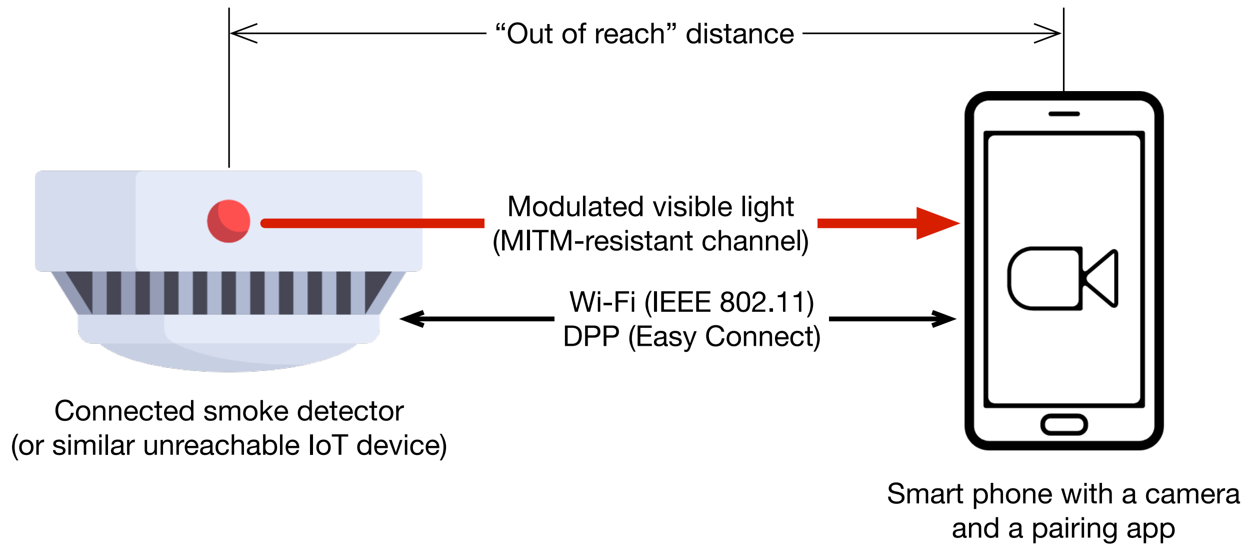


Figure 8.1: Lighthouse system architecture. A smartphone (authenticator) and a remote IoT device participate in an authenticated key agreement protocol over Wi-Fi. The smartphone authenticates the key agreement with a unique secret received from the device over an MITM-resistant visible-light communication channel.

enclosure. In that case, it is assumed to be inaccessible to the end-user (authenticator). Thus, the device's public key serves as its primary identifier.

The Lighthouse system uses public key cryptography to authenticate devices. The authentication protocol runs over two communication channels: Wi-Fi (RF) and visible light. Most protocol exchange occurs over the (insecure) Wi-Fi channel. The visible-light channel is used to transmit a short authentication string (secret) from the device to the authenticator in a manner resistant to MITM attacks.

The authenticator associates a public key with the authenticated device at the end of a successful protocol run. The Lighthouse system does not provide mutual authentication. A device is authenticated to the authenticator but not vice versa. The system's overall security depends on the device's ability to secure its private key, the cryptographic protocols running over Wi-Fi, and the visible-light channel's physical properties (narrow beam, restricted light propagation).

Once an IoT device has been authenticated, the authenticator can create a secure association with the device. The authenticator can then securely communicate with the device, e.g., to provision the device for network access. Network provisioning and other use cases for the secure association are out of the scope of this work.

For compatibility reasons, the Lighthouse system is based on a subset of the DPP framework developed by Wi-Fi Alliance [184]. It follows the DPP recommendations for elliptic curve cryptography parameters, reuses device discovery and peer-to-peer Wi-Fi communication mechanisms, and uses a combination of the DPP Authentication and Public Key Exchange (PKEX) protocols [256]. The Lighthouse visible-light authentication method could also be considered a complementary bootstrapping method for the DPP framework.

8.4.1 Wi-Fi Subsystem

The primary RF network interface connecting the IoT device and the smartphone (authenticator) is Wi-Fi (IEEE 802.11). There is no need for the IoT device and the authenticator to be connected to the same infrastructure network or to be connected to any network at all. Both devices, however, need to support the same frequency band for them to be able to communicate directly.

The Lighthouse system relies on the Device Provisioning Protocol (DPP) (Wi-Fi Easy Connect) framework [184] for direct (device-to-device) Wi-Fi communication. Thus, both devices need to support DPP. Android and Linux-based devices satisfy this requirement since the DPP framework is integrated into those platforms.

8.4.2 Visible-Light Channel

A unidirectional visible-light communication channel from the IoT device to the smartphone (authenticator) exists in the Lighthouse system. This assumes a reasonable distance and a line of sight between the device and the authenticator. The IoT device assumes the role of a transmitter. The smartphone assumes the role of a receiver.

The IoT device has a monochromatic or (preferably) tri-color (RGB) LED. Since the Lighthouse system only uses the LED to transmit occasionally, the LED can also serve as a status indicator. We assume the LED is sufficiently large and bright for the authenticator to be able to pick up the emitted light. If a device is equipped with multiple LEDs, those could all be used simultaneously. Since a vast majority of IoT devices already have a status indicator LED, implementing the visible-light channel requires no additional hardware.

The authenticator uses a smartphone camera as a receiver for the visible-light communication channel. The user keeps the camera pointed at the IoT device while the device is modulating its LED. A custom application running on the smartphone receives a video stream from the camera and decodes the modulated visible light back into the transmitted data. The Lighthouse system places no special requirements on the smartphone camera. An ordinary CMOS camera with a frame rate of at least 25 frames per second is sufficient. Virtually all modern smartphone cameras easily meet these requirements.

The sole purpose of the visible-light channel is to transmit a short randomly generated authentication code from the IoT device to the authenticator. The code must be only known to the device and authenticator. It should be reasonably hard for a remote adversary to observe or tamper with visible-light communication.

The visible-light channel is unidirectional. The receiver (authenticator) has no feedback to the transmitter and cannot require retransmission if data is lost. The Lighthouse visible-light protocol uses a combination of forward error correction (FEC) and error detection techniques to protect the integrity of the transmitted data. A rateless FEC scheme provides adaptation to varying visible-light channel conditions. Rateless FEC allows the receiver to adapt for transmission errors by extending the reception time.

8.4.2.1 Link Budget

The maximum frame rate of the smartphone camera determines the maximum bandwidth of the visible light channel. Most smartphone cameras can capture 30 frames per second (FPS). Selected smartphone cameras support 60 FPS or even 120 FPS in a constrained mode. The Android camera API provides limited control over the frame rate. The application can select a “best effort” frame rate. The camera occasionally drops to a lower frame rate, e.g., when the system is overloaded or when the camera is adjusting exposure. The camera API provides a timestamp with each frame, but it is unclear at what point the timestamp is taken or how precise it is.

The maximum camera frame rate, subject to the Nyquist-Shannon sampling theorem, determines the maximum transmission rate. For a function $x(t)$ that contains no frequencies higher than M Hz, the function is entirely determined by a series of samples spaced no more than $\frac{1}{2M}$ seconds apart. If we approximate the LED signal with a sinusoidal, one period consists of two changes, and the maximum frequency is half the sampling period. Thus, if the camera samples with 60 frames per second, the transmitter can modulate the LED no more than 60 times per second.

The visible-light channel will transfer a short, fixed-length authentication string. The string is used to authenticate an elliptic curve public key. The maximum length of the string is the length of the elliptic curve public key in the compressed form or the length of the key's SHA-256 fingerprint. A public key on the NIST P-256 curve is 33 bytes long in the compressed format. A SHA-256 fingerprint is 32 bytes long. Thus, the maximum length of the authentication code is roughly 256 bits. The minimum length is deployment-specific and is determined by the desired security level, i.e., the probability that an attacker could guess the string. With a 20-bit string, the attacker has roughly a one-in-a-million chance to guess the authentication code.

Assuming a simple Manchester encoding, which encodes each bit into a signal change (i.e., doubling the modulation rate) and 2-level FSK modulation, Table 8.1 summarizes the minimum transmission times for various authentication string lengths and camera frame rates. These are the shortest transmission times, i.e., lower bounds, achievable only on a perfect error-free visible-light channel.

Table 8.1: Minimal transmission times based on string lengths and camera frame rate

	30 FPS	60 FPS	120 FPS
20 bits	$1.\overline{3}$ s	$0.\overline{6}$ s	$0.\overline{3}$ s
64 bits	$4.2\overline{6}$ s	$2.1\overline{3}$ s	1.1 s
128 bits	$8.5\overline{3}$ s	$4.2\overline{6}$ s	$2.1\overline{3}$ s
256 bits	17.1 s	$8.5\overline{3}$ s	$4.2\overline{6}$ s
512 bits	34.1 s	17.1 s	$8.5\overline{3}$ s

8.4.3 Overview of Operation

Consider a system consisting of one IoT device and a smartphone authenticator, as shown in Figure 8.1. Assume the end-user can see the device and wishes to authenticate it to add (enroll) it to a secure Wi-Fi network. The following paragraphs summarize the steps to authenticate the IoT device.

The user opens the Lighthouse app on their smartphone. The app activates the DPP framework [184] to discover nearby IoT devices. After a while, the app shows a list of discovered devices. The app also indicates whether the device supports the Lighthouse authentication protocol for each device.

The user selects a device to authenticate from the list. The app connects to the device over Wi-Fi and activates Lighthouse authentication. The device generates a new random secret and starts transmitting the secret via visible light by modulating its LED indicator. The transmission is visible to the naked eye. The user can observe which device has activated authentication by observing the LED indicator rapidly changing light intensity and color.

The app activates a viewfinder screen, as shown in Figure 8.6. The user points the camera at the device so that the device's flashing LED indicator remains within the yellow circle. As the app decodes the blinking LED and receives data, the yellow progress indicator changes from yellow to red. Once the entire circle is red, the app has received enough data to reconstruct the secret. Under favorable conditions, the entire process should take less than 20 seconds.

The app and the device exchange their public keys over Wi-Fi. The app uses the secret received over the visible-light channel to authenticate the key exchange. A successful authentication indicates that the public key was sent by the device that transmitted the secret via the visible-light channel and not by an adversary.

As the last step, the app verifies that the device possesses the corresponding private key. The authenticator and the device perform an elliptic curve Diffie-Hellman key exchange over Wi-Fi. This step establishes a shared security context between the device and the authenticator. The authenticator persistently associates the public key with the device.

The device stops transmitting on the visible-light channel when the authentication protocol completes, either successfully or with an error.

8.5 Protocols

All communication between the IoT device and the authenticator occurs over two separate communication media: radio (Wi-Fi) and visible light. Most of the protocol exchanges run over Wi-Fi. The visible-light channel only transmits an authentication code from the device to the authenticator. Section 8.5.1 describes protocol exchanges running over Wi-Fi. Section 8.5.2 describes the visible-light protocol.

8.5.1 Wi-Fi

The direct device-authenticator Wi-Fi communication relies on IEEE 802.11 Public Action frames [209]. The Action frame is a type of IEEE 802.11 management frame used for spectrum management, radio measurements, quality of service, and others. The Public Action frame is a type of Action frame used for communication between APs, from an AP to an unassociated station, and between unassociated stations.

The Lighthouse system transmits Public Action frames compatible with the DPP framework [184], i.e., vendor-specific Public Action frames. This is on purpose to simplify the development of the smartphone authenticator application. Existing smartphone platforms generally prohibit application access to link-layer frames. Thus, only existing APIs and frameworks can be used.

Unless the device and the configurator have agreed on mutual channels to use, the following "social" channels are used for all communication between the device and the authenticator: 2.437 GHz (Channel 6), 5.22 GHz (Channel 44) or 5.745 GHz (Channel 149). Only the channels supported by the device and permitted by local regulations are used.

The Wi-Fi communication between the device and the authenticator consists of three protocols: Device Discovery, Authentication Request, and Authenticated Key Exchange. The protocols are described in the following subsections.

8.5.1.1 Device Discovery

A device willing to engage in Lighthouse authentication periodically broadcasts a DPP Presence Announcement frame on social channels. The device transmits the frame on a selected social channel, listens for incoming key exchange requests for two seconds, and then repeats the same procedure on the next social channel. The entire procedure repeats when the list of social channels has been exhausted.

If the device is associated with an AP, it also transmits the presence announcements on the channel used by the AP.

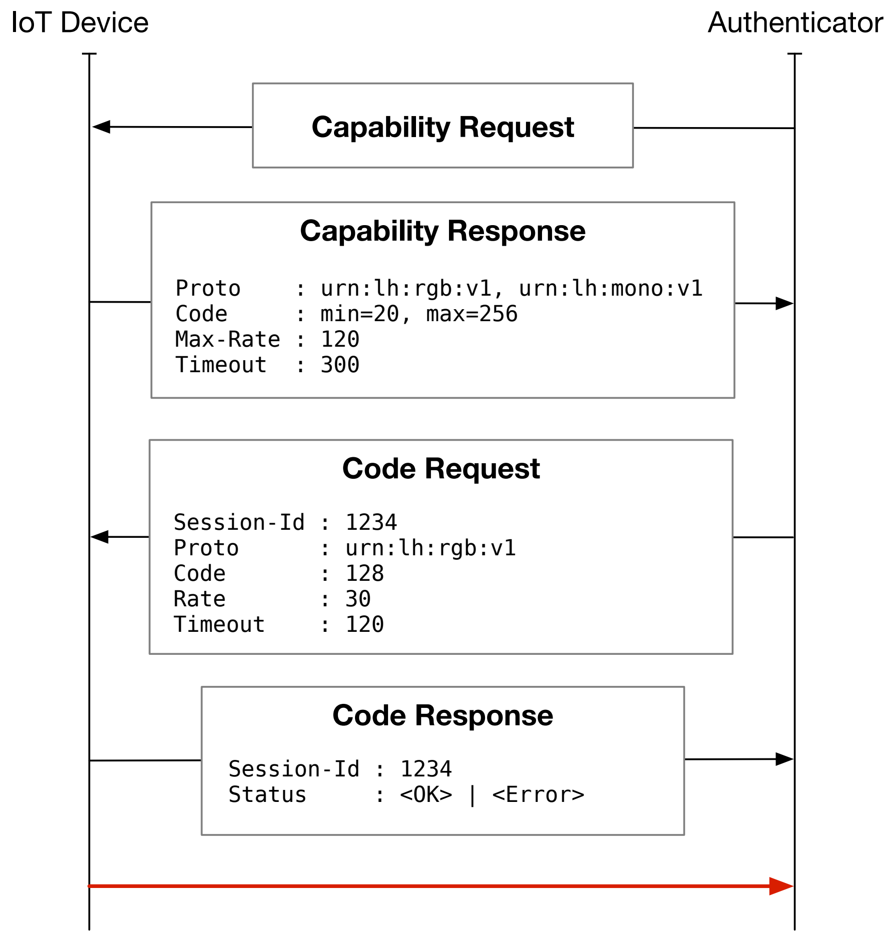
The DPP Presence Announcement frame includes a hash of the device's public key. The authenticator compares the hash with the hashes of the public keys of previously authenticated devices. This allows the authenticator to determine whether a particular device has been previously authenticated.

8.5.1.2 Authentication Code Request

In the authentication code request, the authenticator prompts the selected IoT device to generate a new authentication code and to start transmitting the code via the visible light channel. The authenticator (smartphone) must obtain the authentication code before starting an authenticated key exchange. Figure 8.2 shows a flow diagram of the authentication code request protocol.

The authentication code request protocol consists of two phases. In the first phase, the authenticator sends a capability request to the IoT device to learn about supported visible-light protocols and other parameters. In the second phase, the authenticator sends a code request message to the IoT device to request the generation and transmission of an authentication code.

IoT devices can come in various types and form factors, and their visible-light communication capabilities can vary accordingly. Furthermore, the authenticator (smartphone) must determine whether its hardware (camera) is compatible with the device. The authenticator sends the device an `Capability Request` to learn about supported modulation parameters. The request includes a SHA256 of the device's bootstrapping certificate learned through the device discovery protocol



```
Proto    : urn:lh:rgb:v1, urn:lh:mono:v1
Code     : min=20, max=256
Max-Rate : 120
Timeout  : 300
```

```
Session-Id : 1234
Proto      : urn:lh:rgb:v1
Code       : 128
Rate       : 30
Timeout    : 120
```

```
Session-Id : 1234
Status     : <OK> | <Error>
```

Figure 8.2: Authentication code request flow diagram

described in Section 8.5.1.1.

The IoT device responds with a `Capability Response` message. The message carries attribute-value pairs describing the capabilities of the device. The `Proto` attribute lists all visible-light communication protocols the device supports. The individual protocols are identified by URNs. We envision a registry, e.g., created by IANA, that maps URNs to visible-light modulation parameters. Each URN would be mapped to a combination of FEC, error correction, line coding, and modulation parameters. The example in Figure 8.2 shows two VLC protocols supported by the device, one for an RGB LED and the other one for a monochromatic LED.

The attribute `Code` specifies the minimum and maximum length of the authentication code that the device supports. The authenticator can only request an authentication code whose length falls within this interval.

The `Max-Rate` attribute specifies the maximum modulation rate reliably supported by the device. The authenticator must select a modulation rate smaller or equal to the value provided in this attribute. The value is in modulation frames per second.

The `Timeout` attribute determines the maximum duration of an authentication session in seconds. The authenticator is expected to complete device authentication within this interval. The interval includes visible-light communication and authenticated key exchange. The IoT device will only keep transmitting over the visible-light channel until the timeout value expires or authentication succeeds. Any ephemeral information, such as the generated authentication code or pair-wise elliptic curve keys for an authentication key exchange, will be irrevocably deleted by the device after the timeout interval.

Upon receiving the `Capability Response` message from the device, the authenticator checks if the received parameters are compatible with its capabilities, e.g., the frame rate of its camera. It selects a protocol from the list in `Proto` that it wishes to use and sends a `Code Request` message to the IoT device. The message includes a globally unique `Session-Id` attribute that uniquely identifies the pair-wise authentication session between the IoT device and the authenticator. The message also includes the desired authentication code length, modulation rate, and timeout.

If the attributes included in the `Code Request` message are acceptable to the IoT device and if the device is accepting the authentication request and is not currently engaged in another authentication session, it sends a `Code Response` message back to the authenticator to indicate that it is starting transmitting. In other cases, the device indicates to the authenticator that the request failed via an error descriptor included in the message. An optional `Retry-After` attribute may be included if the device engages in another authentication session.

8.5.1.3 Authenticated Key Exchange

The device and the authenticator engage in authenticated key exchange protocol over Wi-Fi to obtain each other's public keys. At the end of a successful protocol run, the authenticator could trust that the received public key comes from the device and not an adversary attempting to subvert the exchange. As a by-product, the device and the authenticator will also gain a session key to secure future communication.

Since the Lighthouse system is based on the DPP, we assume the public key obtained through the authenticated key exchange may be used as a bootstrapping key in DPP. The authentication key exchange should thus keep the device's public key confidential. Only an entity possessing the correct secret key should be able to reveal the public key. Should the authenticator use the protocol, this will help mitigate denial-of-service attacks in the DPP Authentication Protocol. The confidentiality of the device's bootstrapping public key is used in the DPP framework for weak authentication and authorization of the Configurator (the smartphone).

In the following steps, we assume the authenticator has obtained an authentication code from the device via the steps described in Section 8.5.1.2 or out-of-band, e.g., through a QR code or some other protocol.

The authenticator generates a new ephemeral elliptic curve key pair to be used only with the authenticated device. The authenticator and the device then use the PKEX protocol [256], a balanced password-authenticated key exchange (PAKE) protocol, to exchange their public keys. Using PKEX with DPP is described in Section 5.6.2 of the DPP specification [184]. The Lighthouse system follows the specification. At the end of a successful run of the PKEX protocol, both the device and the authenticator trust that the exchanged public keys come from the remote party and not an adversary attempting to subvert the exchange.

At the end of a successful PKEX protocol run, the authenticator associates the public key obtained through the PKEX protocol with the device. The PKEX protocol authenticated the public key with the authentication code. As long as an adversary cannot obtain or guess the code, the public key originated from the IoT device, and the device has access to the corresponding private key.

8.5.2 Visible Light

Figure 8.3 shows a detailed block diagram of the visible-light communication subsystem, with transmitter blocks on the left and receiver blocks on the right. Various configuration parameters that need to be synchronized between the transmitter and the receiver can be negotiated over Wi-Fi at the start of authentication.

The data transmitted via the visible-light channel is a short, randomly generated authentication code generated by the IoT device. The device generates a new authentication code for each authentication transaction.

8.5.2.1 Transmitting

The transmitter is designed to be flexible and supports several transmission schemes and two kinds of LEDs (monochromatic or tri-color). Several transmission parameters can be influenced by the receiver (authenticator) over Wi-Fi when it requests a new authentication code to be transmitted. The receiver can control the length of the authentication code (within bounds), FEC parameters, and the modulation rate or algorithm.

The transmitter feeds the authentication code into a FEC encoder. The encoder implements the RaptorQ code [257], a systematic rateless fountain code that generates a limitless sequence of encoding symbols from the source data until the transmission stops. The receiver can reconstruct the authentication code with a probability approaching 1 after it has received a certain number of encoding symbols. A rateless code, such as RaptorQ, allows the system to automatically adapt to varying visible-light channel conditions without relying on feedback (acknowledgments or retransmission requests) from the receiver to the transmitter.

Since the authentication code is short (20 to 512 bits), the entire code is processed as a single source block by the RaptorQ encoder. The encoder produces encoding symbols of a fixed size that fit into a single frame.

Fountain codes are designed to operate over an erasure channel, i.e., a channel that only delivers an intact packet with a high probability. Thus, the encoding symbols generated by the RaptorQ encoder need to be protected with checksum or parity symbols. The transmitter feeds each encoding symbol into a CRC-8 checksum generator. Based on the analysis in [258], we selected the CRC-8 polynomial $0x97 = x^8 + x^5 + x^3 + x^2 + x + 1$, which achieves a Hamming distance of 4 for payloads

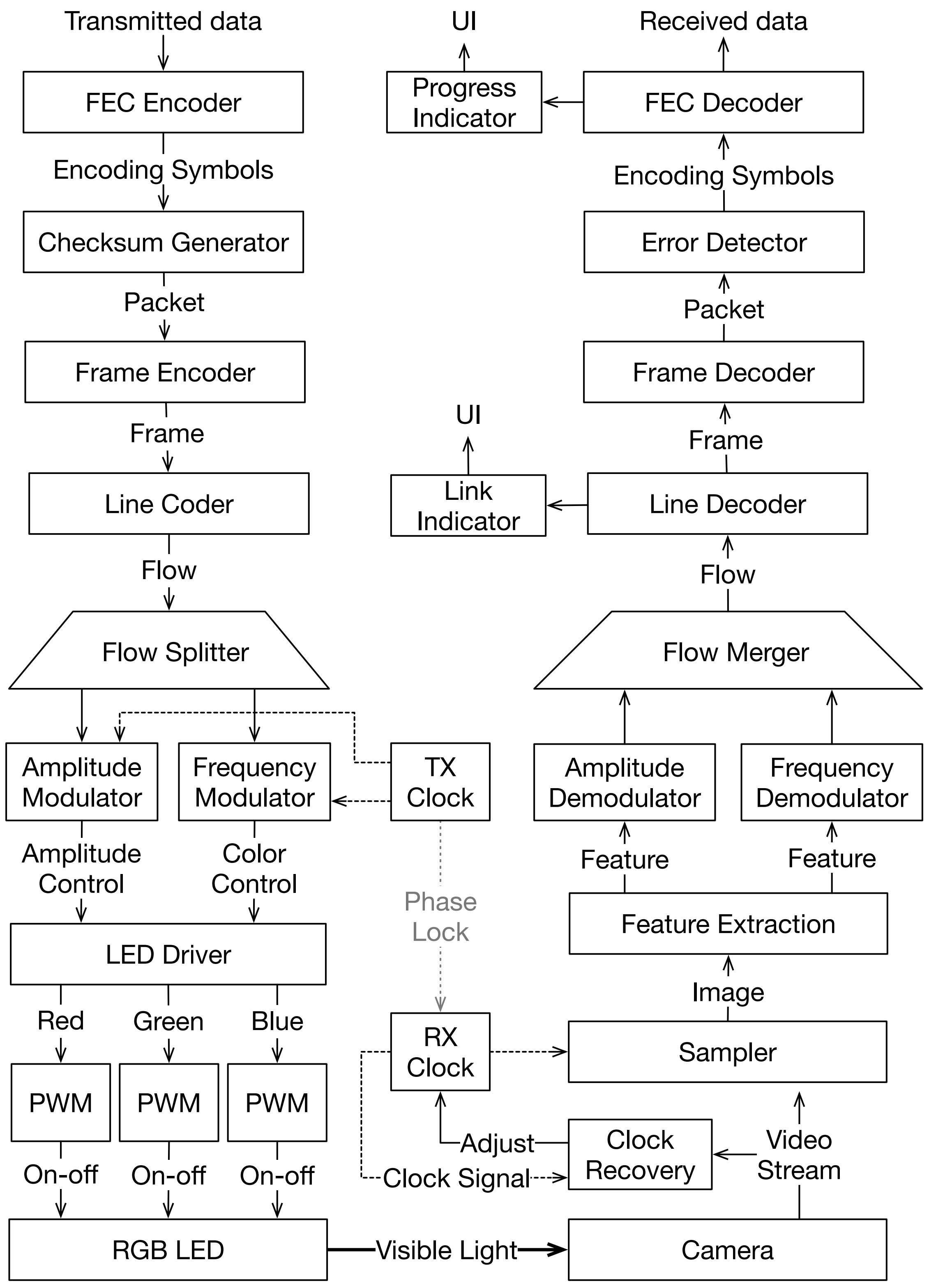


Figure 8.3: A block diagram of the Lighthouse visible-light communication subsystem. Transmitter (IoT device) blocks are on the left. Receiver (authenticator) blocks are on the right.

of up to 119 bits. With this polynomial, there will be no undetected 1, 2, or 3-bit flip errors in a payload of up to 119 bits.

The checksum, the encoding symbol data, and the ID of the encoding symbol form the payload of a single frame, as shown in Figure 8.4.

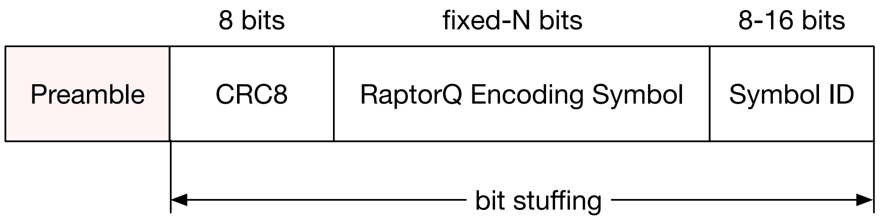


Figure 8.4: Structure of a visible light channel frame

The encoding symbol's ID uses a variable-length encoding and is placed at the end of the frame after all fixed-size fields. If the visible light channel is not too noisy, the transmission can stop after less than 127 frames, and the ID field only uses a single byte. Placing the variable-length field at the end of the frame makes decoding easier.

We set the maximum payload length to 14 bytes (112 bits) for CRC-8 to detect all 3-bit errors. In practice, the payload length will be much shorter and depends on the transmission rate and the rate at which we want the UI to provide feedback about correctly received data. The longer the payload, the longer it takes the receiver to detect whether the packet was correctly received, thus requiring the user to steadily hold the smartphone for longer periods of time.

A frame encoder block adds a preamble at the beginning of the frame. The preamble is a group of symbols guaranteed never to occur in the middle of a frame. The encoder encodes the contents with a bit-stuffing algorithm to guarantee that the preamble does not occur in the middle of the frame. Since the user can point the camera at the LED at any time, the receiver needs to be able to guarantee that it has received an entire frame and not just a portion of it. The encoder escapes occurrences of the marker within a frame (using bit-stuffing) to guarantee that the marker never occurs in the middle of a frame. If there is no data to be transmitted, the encoder keeps transmitting a sequence of preambles.

The frame is then fed into an optional line coder block. Line coding transforms the frame in a manner that helps the receiver synchronize its clock with the transmitter clock. This generally involves encoding the data stream to include frequent or periodic state changes. There are several well-known line coding schemes to choose from, for example, the Manchester code. The line code depends on the selected modulation algorithm. We use the Manchester code for on-off keying (monochromatic LED) and no line code for 4-FSK modulation with tri-color (RGB) LEDs.

Assuming an RGB LED is used, the transmitter modulates the LED's intensity and color (frequency). In that case, the data stream would first pass through a demultiplexer that splits it into two sub-streams. One sub-stream modulates the intensity (amplitude), and the other modulates color (frequency). The amplitude and frequency modulator blocks generate a series of commands for the LED driver to control the amplitude and color of the emitted light. The intensity of each individual LED is controlled with pulse-width modulation (PWM). Both modulator blocks are synchronous and are driven by the TX clock.

When a monochromatic LED is used, the transmitter encodes the data with the Manchester code and modulates the LED with on-off keying (OOK).

8.5.2.2 Receiving

In the receiver (an Android application), a video stream from the camera is first fed into a clock recovery subsystem, which synchronizes the receiver's internal RX clock with the transmitter's TX clock. The clock recovery subsystem runs at the camera's frame rate and analyzes each frame, looking for changes (transitions) in the transmitted signal. If the detected rate is within a range of the TX rate value obtained from the transmitter, the clock recovery block adjusts the internal RX clock. Otherwise (if no usable signal was detected), the clock is left running freely.

The video stream is then submitted to a sampler block, which selects individual frames to be processed based on the RX clock signal. The sampler effectively reduces the frame rate of the video stream to the rate (modulation frequency) of the transmitted signal.

Next, the sampled video stream is passed to a computer vision-based feature extraction component. This component locates the source of the signal in the frame. It performs cropping, color adjustments, and other operations to generate inputs with momentary amplitude and frequency suitable for amplitude and frequency detectors (demodulators).

The rest of the processing is analogous to the initial processing blocks in the transmitter. The output from the demodulator passes through a line decoder, frame decoder, error detector, and RaptorQ FEC decoder. The FEC decoder is the last processing block. The decoder keeps track of how many encoding symbols have been received and calculates the probability of having correct data. Once above a threshold, the data is returned to the application.

The line decoder also outputs a boolean link indicator signal. This signal is represented in the application's UI and helps the user aim the camera. The RaptorQ FEC decoder also outputs a progress indicator signal that helps the user determine how much more data needs to be gathered before the transmitted data (secret) can be reconstructed.

8.6 Implementation

This section describes the prototype implementation of the Lighthouse authentication system. We designed and built a prototype consisting of a Raspberry Pi transmitter (IoT device) and an Android-based receiver. Section 8.6.1 describes the Raspberry Pi transmitter. Section 8.6.2 describes the Android-based receiver.

Both the transmitter and the receiver are implemented in Java. Platform-specific functionality is implemented in C. A considerable portion of the code base, particularly at higher layers, is shared between the transmitter and the receiver in the form of a JAR library. The JAR file is linked to the transmitter application and included in the Android receiver APK package.

We used the prototype implementation to measure the parameters of the visible-light channel, i.e., its bit error rate (BER) under various conditions and over specific distances. These measurements informed the design of the modulation scheme, data encoding, and packet format discussed in Section 8.5.

We also used the prototype to evaluate the system's feasibility under various lighting conditions and over typical distances. The evaluation is discussed in Section 8.8. The transmitter was mounted under a ceiling in a lab on the Columbia University campus for these experiments. For initial functional tests, the smartphone was mounted on a tripod. In later experiments, the smartphone was hand-held.

8.6.1 Transmitter

Figure 8.5 illustrates the overall architecture of the prototype transmitter. We implemented the prototype using a Raspberry Pi model 3 B, as shown in Figure 8.6. The device used the built-in Wi-Fi interface for network connectivity during all experiments. The Raspberry Pi was powered via USB from mains power. The device was running the default installation of Raspbian (Debian 9.4), the Linux-based OS for Raspberry Pi.

A triple-color white diffused LED with a diameter of 5 mm, model YSL-R596CR3G4B5W-F12, manufactured by China Young Sun LED Technology Co. Ltd., is connected to three 3.3 V GPIO pins on the Raspberry Pi. Each color channel (red, green, blue) is connected to a separate GPIO pin (pins 22, 27, 17). The transmitter uses PWM with a frequency of 40 kHz to control the light intensity of each LED channel individually. We extended the range of the LED with a reflector. In retrospect, a diffuser or a larger diameter LED might have been a better choice (the LED with the reflector was often too bright for the smartphone camera).

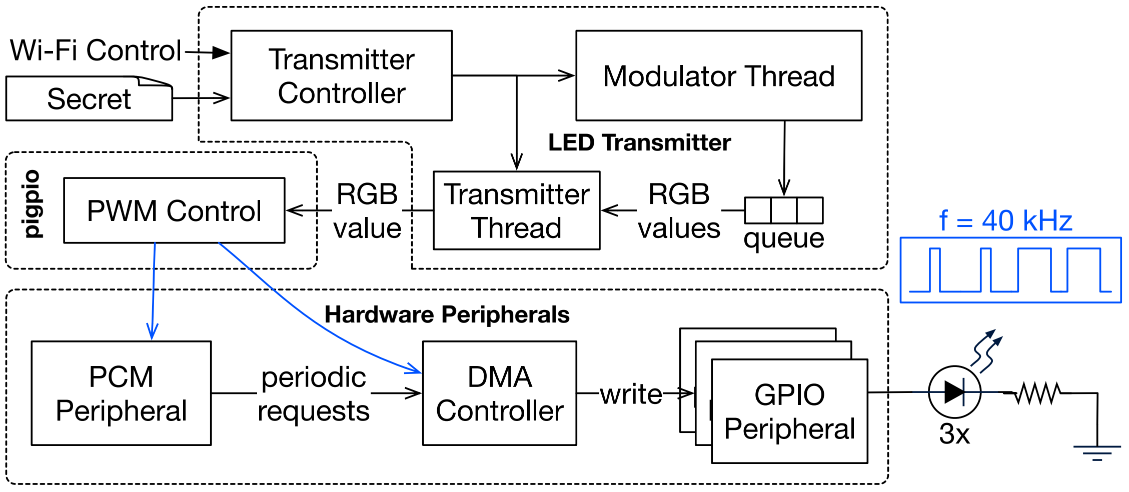


Figure 8.5: A block diagram of the prototype transmitter (IoT device). The transmitter was built using a Raspberry Pi 3 with an RGB LED connected to its GPIO ports. The LED intensity was controlled with pulse-width modulation (PWM).

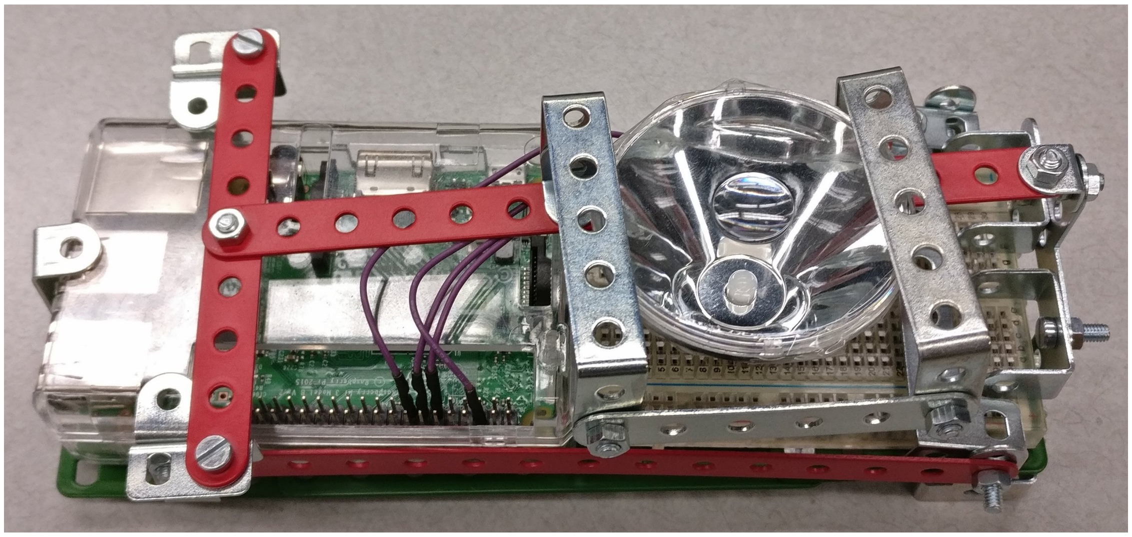


Figure 8.6: A prototype Lighthouse transmitter based on a Raspberry Pi 3. An RGB LED was fitted inside a reflector and connected to GPIO ports on the Raspberry Pi. The device was mounted under a ceiling in a lab at Columbia University during experiments.

The Linux operating system running on the Raspberry Pi is not real-time capable. This makes PWM control on GPIO pins challenging. If PWM is controlled from software, an interrupt or context switch at the wrong time might generate an unacceptable delay. This issue can be mitigated with hardware-based PWM control. Unfortunately, the Raspberry Pi only has two GPIO pins with the hardware PWM capability, but our application needs to control three LEDs. We use the `pigpio` library [259] to precisely control PWM on three GPIOs using direct memory access (DMA). The library programs the PCM (sound) peripheral to generate the desired data stream to control PWM and the DMA controller to automatically transfer the data from the PCM peripheral to GPIO control registers. This approach allows us to precisely control all three PWM outputs without worrying about operating system interruptions.

The transmitter manages the Wi-Fi interface using the venerable open-source `wpa_supplicant` program. We compiled the supplicant from the source code to enable support for the DPP framework. The supplicant communicates with the main transmitter application over a UNIX domain socket.

The timing APIs in standard Java did not provide sufficient accuracy for our purposes. We use a combination of standard Java `sleep` and busy-waiting for precise timing in the transmitter application thread. The transmitting process would sleep slightly less than the desired time to allow the OS to switch context. Upon waking up, the process would calculate the remaining time to the deadline and use busy-waiting to suspend the program for the remainder of the time.

The transmitter application is a multi-threaded Java 8 application running on OpenJDK. A dedicated transmitter thread implements a tightly controlled transmission loop based on the busy-waiting technique described in the previous paragraph. A separate modulator thread performs all data encoding and formatting in the background and feeds the transmitter thread with pre-computed data (RGB values) via a shared blocking queue. A transmitter-controller thread controls the entire application and handles API requests received over Wi-Fi. This thread performs platform initialization and configuration and generates random authentication codes whenever an authenticator requests the start of visible-light transmission.

The remainder of the transmitter application implements data processing as described in Section 8.5.2. These processing blocks turn an authentication code (secret) into a sequence of control commands for the three PWM controllers. The architecture of the data processing subsystem is shown in detail in Figure 8.3. The transmitter uses the OpenRQ Java library [260] for the FEC encoder.

We used the transmitter in lab experiments designed to select the most suitable combination of modulation, framing, data error detection, and correction algorithms. To facilitate such experimentation, the transmitter software is designed to be configurable and easy to extend by overriding existing Java classes. Most visible-light communication subsystem parameters can be configured at startup time without modifying the source code. The authenticator can configure selected parameters at run-time via the Wi-Fi API. The transmitter software is implemented as pluggable and supports different kinds of LEDs (monochromatic or tri-color), several modulation techniques (on-off keying, 2/4/8-level FSK), a small set of CRC polynomials, and calibration parameters for the LED.

8.6.2 Receiver

The receiver is implemented in the form of an Android application. The application uses the smartphone camera, accessed through the Android Camera2 API, to receive modulated visible light from the IoT device and decodes the signal into data. The app uses the Android port of OpenCV [261], the free computer vision library for image processing. For Wi-Fi and DPP framework access, the app uses built-in APIs of the Android platform. Figure 8.7 shows the architecture of the application.

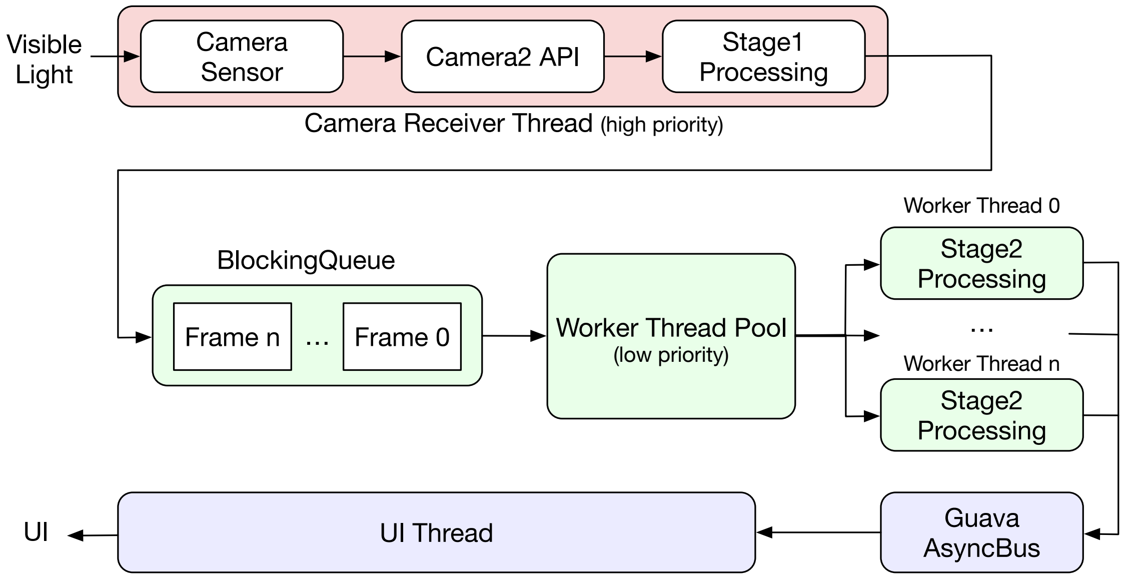
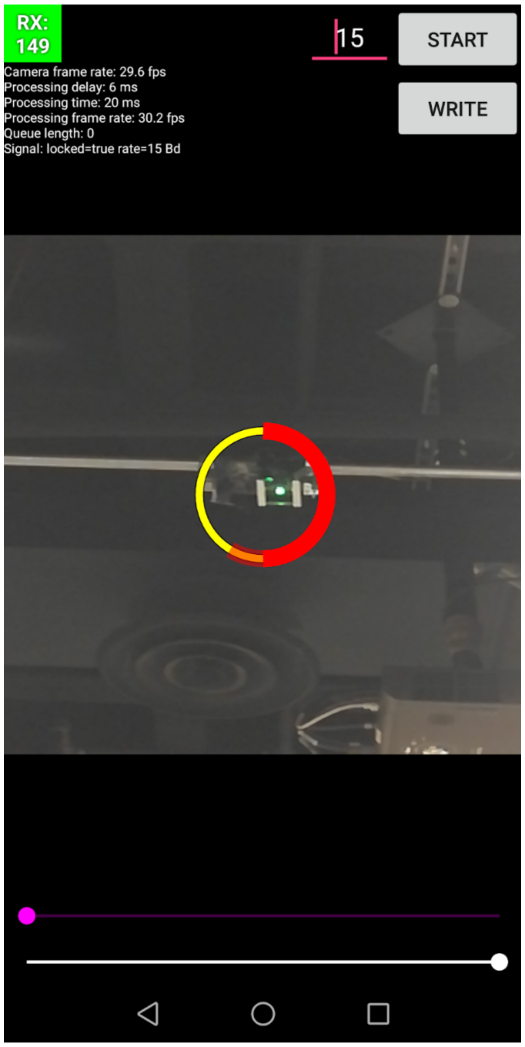


Figure 8.7: The internal architecture of the Android authenticator application

The application employs a multi-threaded architecture to move most of the work away from the main UI thread and to take advantage of multiple CPU cores, if available. All processing performed by the application is divided into two stages: stage 1 and stage 2.

Stage 1 processing is performed at the camera's frame rate, i.e., for every frame received from the camera. Stage 1 processing takes place on a dedicated high-priority thread and only includes operations that run in constant time. Such operations include frame rate calculation, region-of-interest extraction, and frame cropping. The output of stage 1 processing is posted to a shared blocking queue.

Stage 2 processing includes all time-consuming processing tasks. These tasks are executed on a



Camera frame rate: 29.6 fps
Processing delay: 6 ms
Processing time: 20 ms
Processing frame rate: 30.2 fps
Queue length: 0
Signal: locked=true rate=15 Bd

Figure 8.8: Android app viewfinder being used to authenticate an IoT device (green light in the center). The yellow circle is a progress indicator. The red portion indicates the amount of data received so far. The app can reconstruct the data once the entire circle has turned red. The two sliders at the bottom could manually adjust the camera's zoom and exposure if necessary.

low-priority thread pool. Whenever a thread in the pool becomes idle, it checks the shared blocking queue for work to do. If a frame is in the queue, the thread picks it up and starts processing it. Typically, the thread pool would contain one thread per CPU core.

Some data must be passed to the main (UI) thread to show visual feedback to the user. The figure in Figure 8.8 shows the UI implemented by the application. The application must remain responsive to prevent the Android activity manager from shutting it down. Thus, all time-consuming operations need to be performed on the thread pool and not in the main UI thread. Worker threads communicate with the main UI thread via an asynchronous event bus implemented using the Guava library.

Stage 2 processing consists of six sequentially executed blocks: color detector, demodulator, sampler, line decoder, packet decoder, and RaptorQ decoder. The output of a block becomes the input of the next block on the list. Each block is implemented by a separate Java class that implements an abstract interface. All blocks are sequentially executed on the same thread from the low-priority thread pool.

The following paragraphs describe the color detector, demodulator, and sampler block. The operation of the remaining blocks is described in detail in Section 8.5.2.

Region of Interest. The input to the first block on the list (color detector) is a region of interest (RoI). The RoI is a small area cropped from the original (large) camera frame. The RoI is the area inside the yellow circle in the screenshot in Figure 8.6. The end-user is responsible for aiming the camera so that the LED of the transmitting device remains within the circle (and thus the RoI). A more sophisticated application could try to automatically locate the RoI within the larger frame using computer vision techniques. That would also allow the application to receive multiple transmissions simultaneously. We have not implemented these techniques due to a lack of time.

Color Detector. The first processing block reduces the RoI into an average hue. The block first converts the RoI data from a YUV420_888 camera format to a hue-saturation-value (HSV) format using OpenCV. The HSV format represents luminance and chromaticity values separately. Thus, the relative brightness of the color does not influence its hue. Next, the color detector eliminates very bright and very dark pixels, i.e., pixels that have been either underexposed or overexposed. The elimination is based on predefined thresholds for the saturation and brightness components. Thus, the average hue will only be computed over correctly exposed pixels. The block then computes the average hue for the RoI using the following formula for calculating the average of an angle:

$$\overline{\alpha} = \operatorname{atan2}\left(\frac{1}{n}\sum_{j=1}^{n}\sin\alpha_j, \frac{1}{n}\sum_{j=1}^{n}\cos\alpha_j\right)$$

Demodulator. The demodulator converts the average RoI hue computed in the previous step into a modulation symbol. The Lighthouse visible-light link uses 4-level FSK, which maps three symbols to the primary colors (red, green, blue) and the fourth symbol to black (LED is off). This modulator scheme is easy to detect and simple to implement. Please refer to Section 8.5.2.1 for details. The demodulator uses the nearest neighbor algorithm to map the average hue to the nearest of the four modulator colors in a three-dimensional Euclidean space.

Sampler. The sampler block synchronizes the receiver with the transmitter. This block selects a subset of the RoI frames, with rate r, for further processing such that each selected frame represents one transition of the signal, regardless of the camera frame rate c. The camera interval c must be smaller than r to ensure at least one frame sample per each interval. In the Android Camera2 API, the camera's frame rate is not constant. In the prototype, we chose to run the camera at half the maximum sample rate for stable results.

8.7 Security Considerations

The Lighthouse system can be viewed as an extension of the DPP (Wi-Fi Easy Connect) framework. Specifically, Lighthouse establishes an MITM-resistant visible-light communication channel from the device (DPP Enrollee) to the authenticator (DPP Configurator). The authentication code is then used to authenticate a DPP PKEX exchange. The overall security of a Lighthouse-based authentication system thus depends on the security of the DPP framework and specifically of the PKEX bootstrapping method.

The Lighthouse system assumes that a remote adversary could not perform an active attack on the visible-light communication channel and cannot compromise the authenticator itself. Active MITM attacks are extremely difficult to perform against a visual communication channel without the user detecting that such an attack is taking place. Since the end-user has to aim his or her camera at the device being authenticated and sees the device on the smartphone's display, the adversary's light source would be immediately apparent. Thus, the communication channel is resistant to active

MITM attacks, and the user quickly detects such attacks. The channel provides the end-user with demonstrative identification of the device being authenticated.

The authentication code communicated through the visible-light channel must be randomly generated with a cryptographically strong random number generator. The authentication code must be uniquely generated for each authentication session and must not be reused. The device should irrevocably delete the authentication code as soon as it is no longer needed.

Since the Lighthouse system is based on a password-authenticated key exchange (PKEX), it is resistant to both offline and online dictionary attacks. An adversary who does not know the authentication code could try to guess the code by tampering with the key exchange. The probability of a successful guess is proportional to the length of the length of the authentication code in bits. The Lighthouse system recommends 20 bits as the minimum length of an authentication code. This length gives an adversary one chance in a million to guess the code.

Since the visual communication channel is only used to communicate a one-time secret that authenticates a key exchange performed over the RF channel, eavesdropping on the visual channel would give the adversary no advantage as long as the RF cryptographic protocols are secure and the devices have not been compromised.

The Wi-Fi-based Authentication Code Request protocol is potentially vulnerable to denial-of-service attacks. Upon discovering a device willing to participate in Lighthouse authentication, an adversary could repeatedly issue authentication requests to the device, blocking the device's visible light transmitter (LED) for an extended time. We expect most IoT devices will only engage in Lighthouse pairing for a limited time, i.e., if the device detects that it has not been associated with any other device or network yet. Thus, the adversary could only launch this attack on newly unboxed or factory-reset devices. The gain from such an attack is minimal.

8.8 Evaluation

This section discusses an experimental evaluation of the Lighthouse prototype system. We used two transmitter configurations with the parameters summarized in Table 8.2. We used three smartphones as authenticators; their key parameters are summarized in Table 8.3.

Table 8.2: Evaluated Lighthouse transmitter configurations

Configuration	**1**	**2**
Platform	Raspberry Pi 3 B, rev. 1.2	Raspberry Pi 3 B, rev. 1.2
OS	Raspbian (Debian 9.4)	Raspbian (Debian 9.4)
Kernel	Linux 4.14.52-v7+	Linux 4.14.50-v7+
LED	Common-cathode diff. 5mm RGB	Common-anode diff. 10mm RGB
LED current	20 mA	20 mA

Table 8.3: Evaluated Lighthouse receiver configurations

Configuration	**1**	**2**	**3**
Device	Huawei Honor 7X	Samsung Galaxy S7	OnePlus 3T
Android version	7.0	8.0	8.0
Maximum frame rate	30 FPS	30 FPS	60 FPS

8.8.1 Hue Detection Accuracy

In this experiment, we tested the color detection accuracy of the three smartphones when used with the two transmitters in a lab with fluorescent overhead lights on the Columbia University campus.

First, we modified the transmitter to iterate over a sequence of hue values. Each hue was transmitted for one second. We modified the Android application to record received hue values, timestamps, and other camera-related parameters.

Next, we calibrated the receiver (smartphone camera) in a completely black environment, i.e., with no background light. Figure 8.9 shows a black box we built for this purpose. The black box contains one of the transmitters and the receiver (smartphone). The values obtained through the black box calibration process were used to correct the hue value detected by each smartphone model across the spectrum. This step allowed us to compensate for hue detection deviations in the three smartphone models.

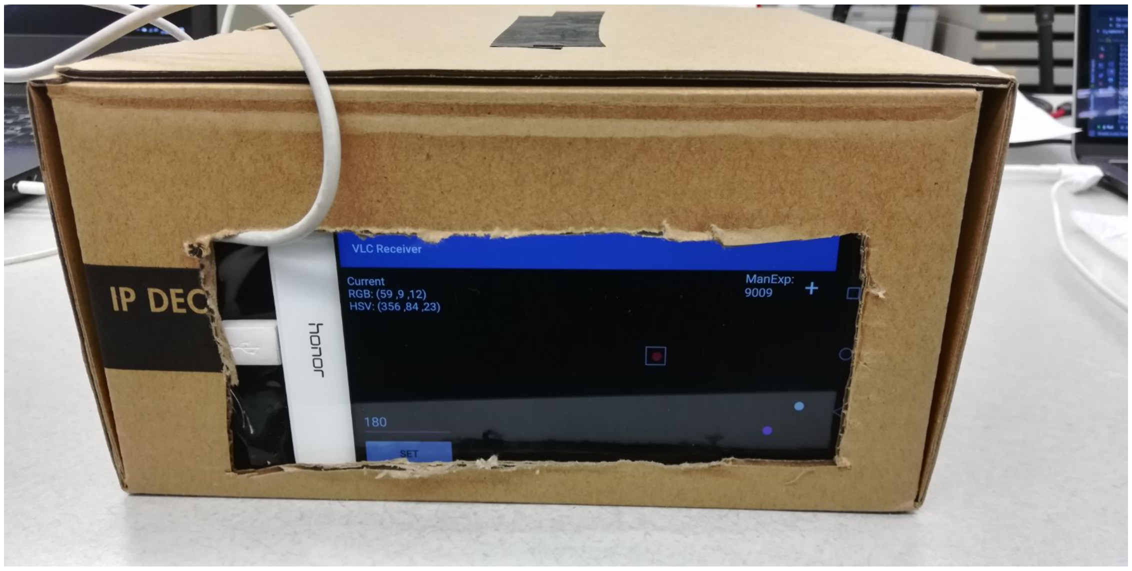
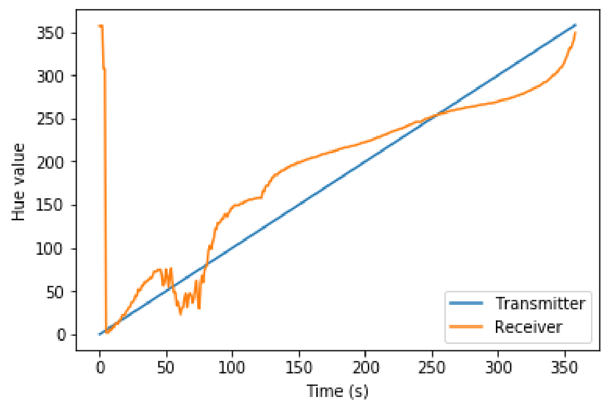


Figure 8.9: Receiver hue detection calibration. We placed each evaluated smartphone into a black box with a remotely-controlled RGB LED to measure the camera's response to the emitted light.

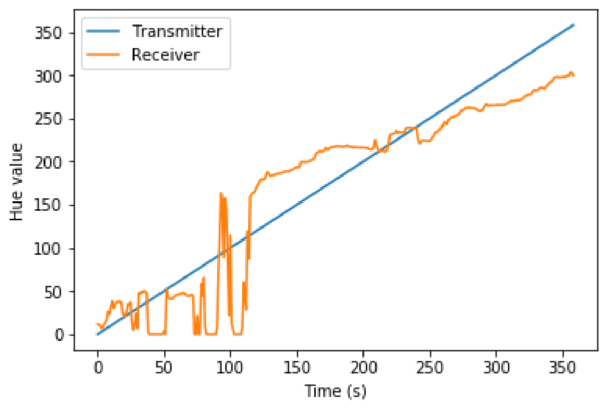
We then ran the hue detection experiment over two distances: two meters (near) and four meters (far). In all experiments, the transmitter was mounted under a ceiling, and the smartphone was mounted on a fixed tripod in the same room. The room was lit with fluorescent overhead lights. Figures 8.10 and 8.11 contain all hue accuracy measurement results.

Both transmitters showed a significant offset from the ideal value in the green portion of the hue spectrum (values 80–180) over the two-meter distance. Out of the three tested smartphones, the Samsung Galaxy S7 seems to be showing more significant errors in this hue range than the Huawei Honor 7X or OnePlus 3T. The same test performed over a distance of four meters shows similar offsets for both transmitters in the green segment. Since all measurements exhibit a similar offset, the error is likely systemic. One possible explanation is that the smartphone camera uses a filter that enhances green hues, as the human eye is known to be the most sensitive in this portion of the spectrum. Another explanation might be that the camera enhances this color range so that the human skin looks natural.

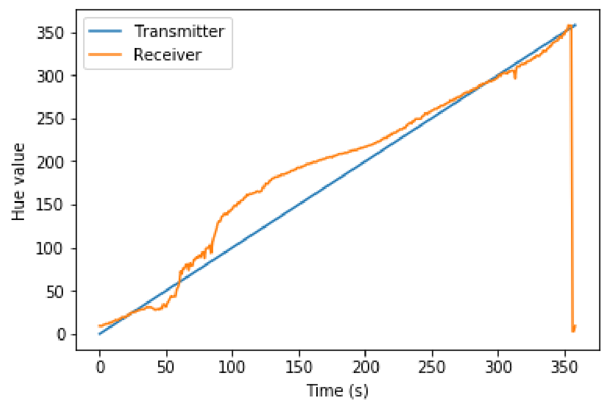
Hue measurements with transmitter 2 (Figure 8.11) additionally show significant deviations in



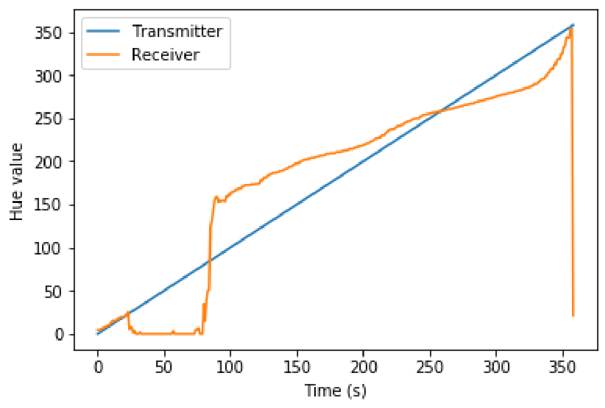
(a) Huawei Honor 7X, 2m



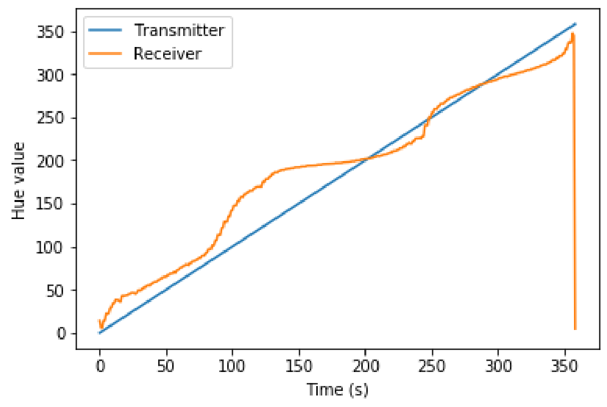
(b) Huawei Honor 7X 4m



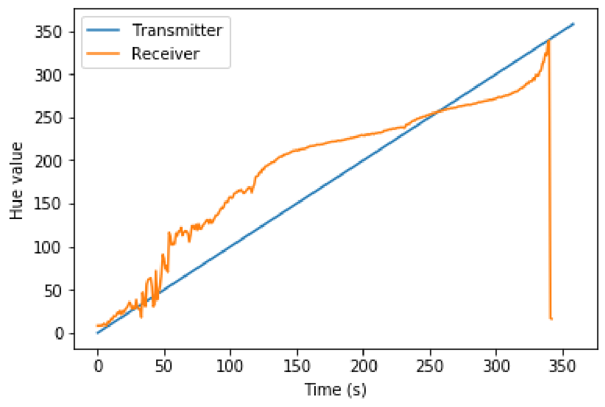
(c) Samsung Galaxy S7, 2m



(d) Samsung Galaxy S7, 4m

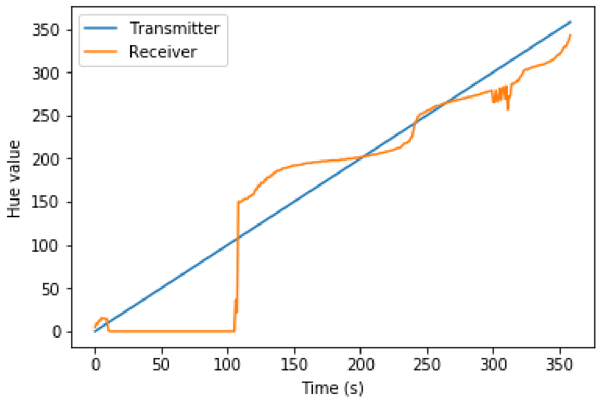


(e) OnePlus 3T, 2m

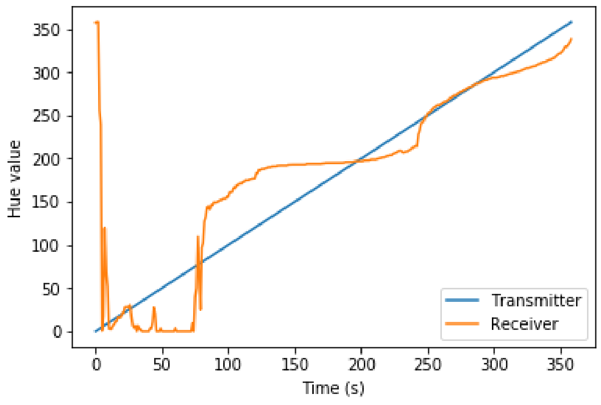


(f) OnePlus 3T, 4m

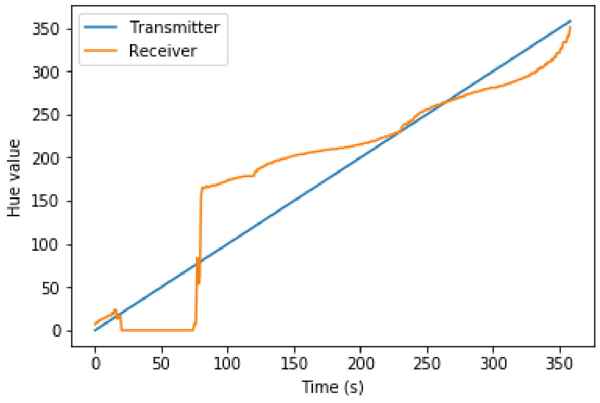
Figure 8.10: Hue detection accuracy measurement results for transmitter 1



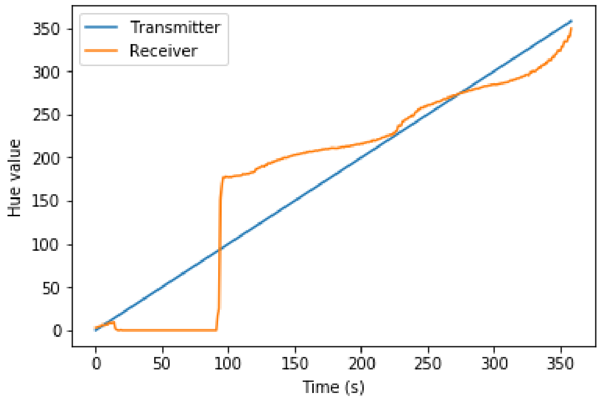
(a) Huawei Honor 7X, 2m



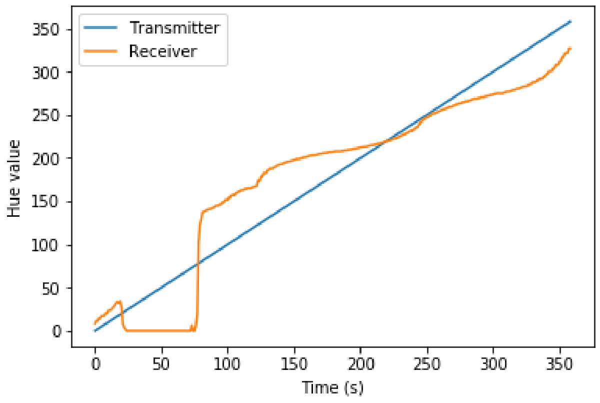
(b) Huawei Honor 7X, 4m



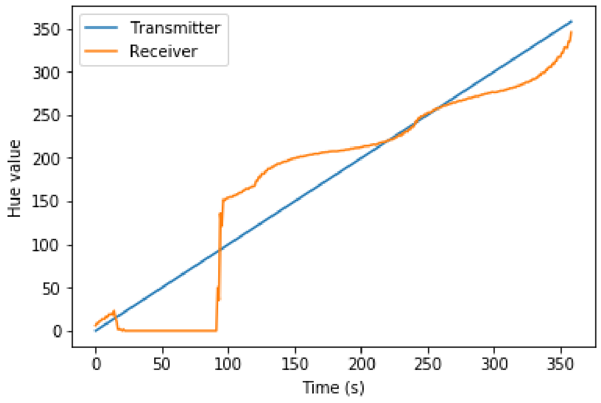
(c) Samsung Galaxy S7, 2m



(d) Samsung Galaxy S7, 4m



(e) OnePlus 3T, 2m



(f) OnePlus 3T, 4m

Figure 8.11: Hue detection accuracy measurement results for transmitter 2

the yellow portion of the spectrum (values 0–80). The detected hue drops to 0 in that portion of the spectrum. The color detection algorithm in the Lighthouse app uses saturation and brightness thresholds to eliminate gray colors. Thus, one explanation is that the LED in transmitter 2 is too bright to detect yellow hues accurately.

The graphs for Huawei Honor 7X show less accurate results overall over both distances. This smartphone's camera provides less accurate color detection than the Samsung Galaxy S7 or OnePlus 3T.

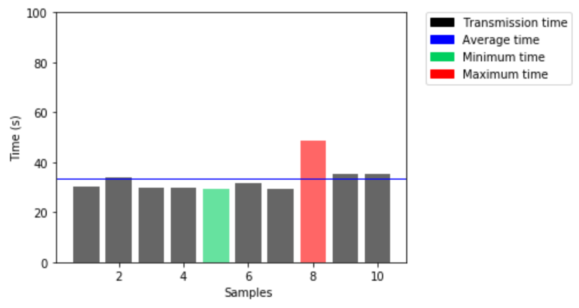
Based on these observations, we conclude that the three primary colors (red, green, and blue) might be best for modulation. In all cases, the distance of the perceived color from the actual color is less than the distance from one of the other primary colors. This is an important precondition for the nearest neighbor detection algorithm. Thus, with a three-color LED, 4-level FSK modulation is an option (with the fourth symbol encoded as dark).

8.8.2 Transmission Time

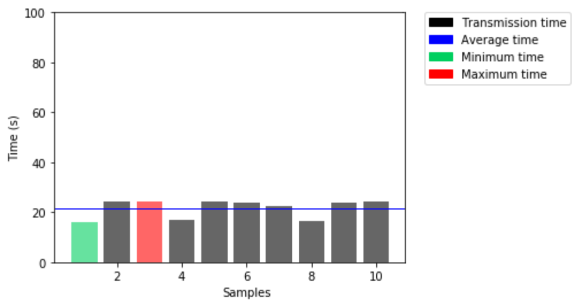
In this experiment, we measured the transmission times of a 128-bit authentication code. The measurements were performed over two distances of two and four meters. The setup was identical to the previous experiment, i.e., the transmitter was mounted under a ceiling, and the smartphone was mounted on a tripod. For an extra margin, we configured the smartphone camera with half the maximum frame rate supported on the device. Samsung Galaxy S7 and Honor 7X used 15 frames per second. OnePlus 3T used 30 frames per second.

Figures 8.12 and 8.13 show all measurement data. Each measurement consisted of ten consecutive transmissions. A unique authentication code was generated for each transmission. In the graphs, red bars indicate the longest transmission, green bars indicate the shortest transmission, and the blue line represents the average transmission time over ten transmissions.

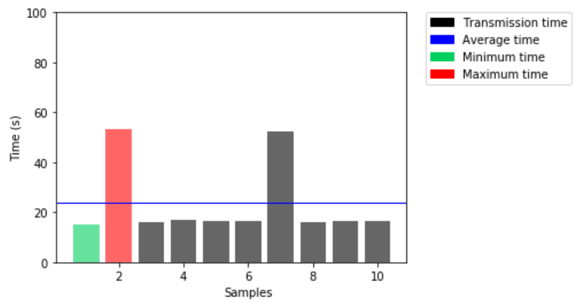
The measurements performed over two meters with transmitter 1 show the expected performance of the Samsung Galaxy S7 and OnePlus 3T. Eight out of ten transmissions lasted 15 to 16 seconds with Samsung and approximately 10 seconds with OnePlus. The remaining two measurements



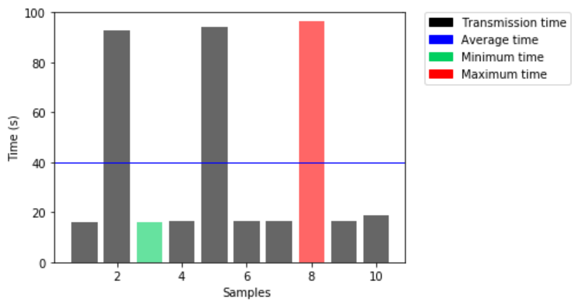
(a) Huawei Honor 7X, 2m



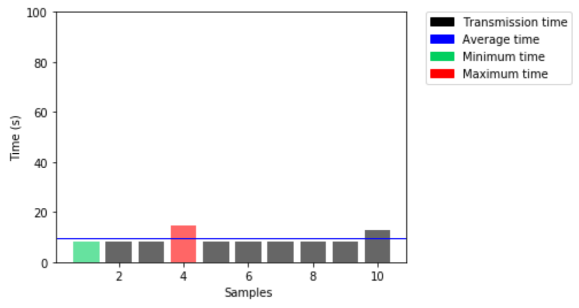
(b) Huawei Honor 7X, 4m



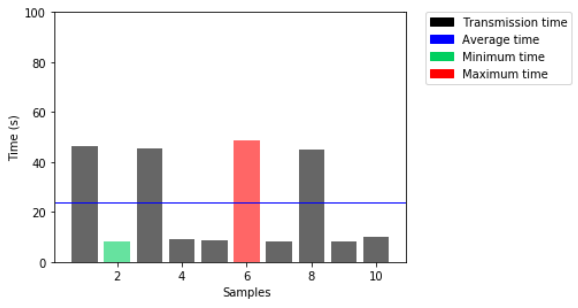
(c) Samsung Galaxy S7, 2m



(d) Samsung Galaxy S7, 4m

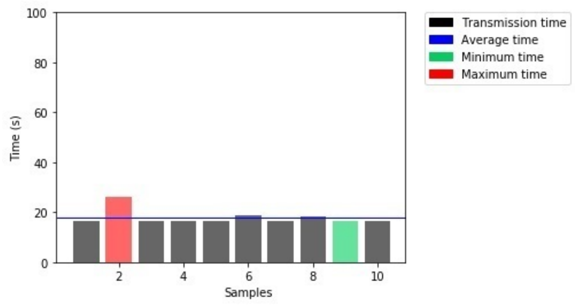


(e) OnePlus 3T, 2m

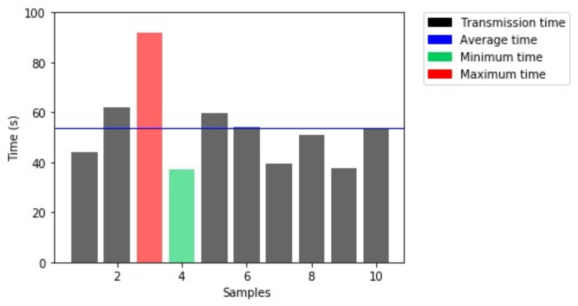


(f) OnePlus 3T, 4m

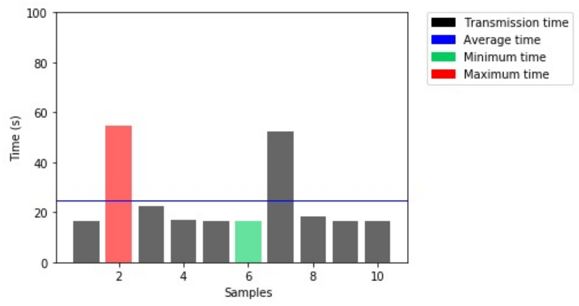
Figure 8.12: Transmission time measurement results for transmitter 1



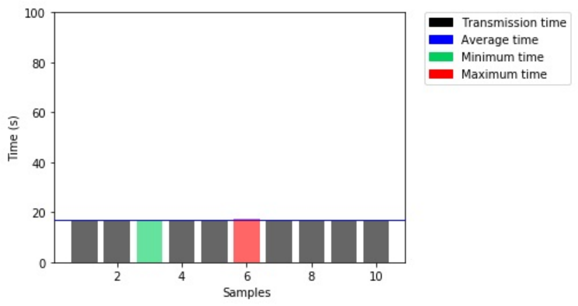
(a) Huawei Honor 7X, 2m



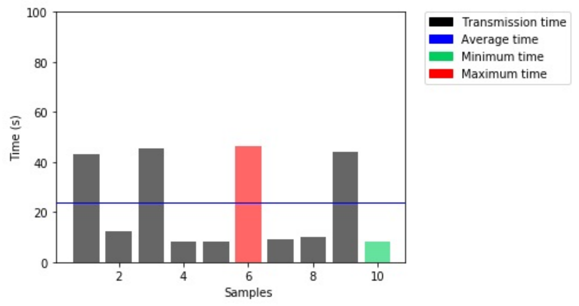
(b) Huawe Honor 7X, 4m



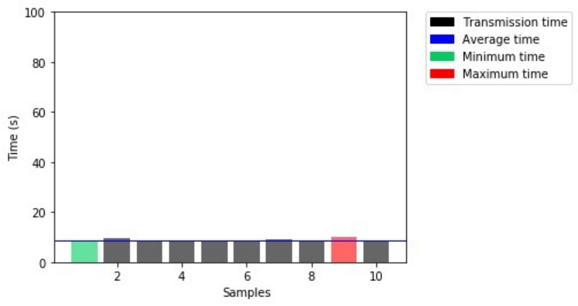
(c) Samsung Galaxy S7, 2m



(d) Samsung Galaxy S7, 4m



(e) OnePlus 3T, 2m



(f) OnePlus 3T, 4m

Figure 8.13: Transmission time measurement results for transmitter 2

show much longer durations of 52 to 53 seconds. The long transmission times can be explained by delayed synchronization, i.e., the phone started receiving a previous frame in the middle. The frame got dropped, but its transmission time was added to the transmission time of the next successfully received frame. The Huawei phone is showing much longer transmission times overall. This indicates a higher error rate.

When the distance increased from two to four meters, the number of long transmissions and their duration increased for both Samsung and OpenPlus phones. As expected, this indicates a higher error rate on the visible light channel. The baseline transmission times remain 15 seconds for Samsung and 10 seconds for OnePlus over this distance. Given the modulation rate, these are the shortest possible transmission times and represent an error-free transmission.

Interestingly, the Huawei phone performed better on average over a distance of four meters than it did over two meters. This could indicate a problem with the setup or an inadequate automatic exposure algorithm in the phone.

The results were comparable with transmitter 2. In these experiments, the Huawei phone again showed inconsistent performance, but this time, it performed worse over the long distance than it did over the short distance. Samsung and OnePlus phones exhibited marginally more transmission errors over short distances but performed very well over long distances. One potential explanation for this difference is that transmitter 2 has a higher luminosity that the cameras cannot correctly compensate for, leading to overexposed camera frames that the application cannot properly decode.

In conclusion, these experiments show that Samsung Galaxy S7 and OnePlus 3T perform consistently overall, with transmission times roughly in the expected range with occasional long transmissions due to missed synchronization. The performance of the Huawei Honor 7X was inconsistent across both transmitters and distances. We attribute the inconsistency to a low-performing camera on the Huawei phone.

8.8.3 Usability

In this experiment, we let two members of our research lab unfamiliar with the project perform visible-light IoT device authentication using the Lighthouse prototype. We aimed to observe their performance as they hand-held the phone and note the obtained transmission times. Each test subject performed the transmission twice on one of the Samsung Galaxy S7 or OnePlus 3T smartphones. The distance between the transmitter and the test subject was between two and three meters.

The first test subject experimented on the Samsung Galaxy S7. The first run took approximately 56 seconds. During the run, the test subject adjusted auto exposure compensation to match the ambient light levels. The test subject also tried different zoom levels. Consequently, the camera was not always pointed at the transmitter correctly, and some frames got dropped, resulting in a longer transmission time. In the second run, the test subject adjusted the camera before the run, and the transmission itself took only 18 seconds.

The second test subject used the OnePlus 3T. This test subject did not adjust the exposure settings or the zoom level. The first transmission took approximately 10 seconds. The second transmission took approximately 10 seconds.

Our test subject population consisted of only two people. This is not sufficient for any generalizable results. However, the usability experiment shows that upon getting familiar with the application, our two human test subjects were able to obtain transmission times estimated in Section 8.4.2.1. Automatic exposure and zoom settings could additionally improve user experience. Moreover, both test subjects rated the viewfinder UI in the Android app as intuitive and user-friendly.

8.9 Related Work

The literature contains a significant body of prior work on VLC, also known as optical wireless communications. As incandescent light bulbs get replaced by LED technology, VLC is increasingly being viewed as a compelling alternative physical layer to RF for IoT devices. Visible light communication is inherently MITM-resistant due to the spatial confinement of light beams [262].

VLC research generally focuses on maximizing bandwidth and minimizing the number of visual artifacts. In other words, VLC research aims to design systems that transfer the maximum amount of data in the visible-light channel in a manner that is imperceivable to humans. The requirements of a visible-light authentication protocol are somewhat complementary. It is generally sufficient to transfer a small amount of data noticeably to make an ongoing authentication process visible to humans.

BlinkUp [263] is a visible-light communication technology developed by The Electric Imp. The BlinkUp protocol can transfer network configuration from a smartphone into an Electric Imp embedded device. Unlike our protocol, the BlinkUp protocol is designed for configuration (not authentication), requires a photo-sensitive component in the IoT device, and works only over short distances.

VL-Auth [264] is a visible-light-based framework for Wi-Fi authentication. The framework authenticates a station to connect to an open Wi-Fi network based on the visible-light signature of the station's location, for example, different frequency domain signatures of different fluorescent lights. The framework is based on the IEEE 802.1x authentication protocol. The framework requires training, i.e., all areas that should be authenticated must be pre-measured.

The authors in [265] a visible-light authentication system called LISA, which uses a combination of RF and visible-light communication, similar to the Lighthouse system. Unlike our project, the LISA system relies on smartphones and tablets as visible-light transmitters.

Seeing-is-Believing [266] is a system that utilizes 2D barcodes and smartphone cameras to pair two devices with no prior shared security context. The authenticator (a smartphone) scans a 2D barcode (QR code) presented by the device on its screen. A static barcode must be affixed to the device's housing if the device has no screen. The barcode consists of a hash of the device's public key.

8.10 Summary

This chapter presented the design, implementation, and experimental evaluation of the Lighthouse system, a visible-light communication protocol for physically unreachable IoT devices. The mechanism complements existing device mechanisms based on QR codes, PINS or passwords, NFC, or Bluetooth. The ability to authenticate remote IoT devices significantly expands their deployability and paves the way for zero-touch IoT device network enrollment.

We built a prototype system based on the Raspberry Pi and an Android smartphone. Unlike other similar IoT device authentication mechanisms, visible-light authentication works with various form factors and does not require expensive additional hardware.

The Lighthouse system is only applicable in scenarios where a line of sight exists between the IoT device being authenticated and the smartphone. A different authentication mechanism would need to be used for devices installed in cabinets or walls. Visible-light authentication could also be negatively influenced by intense ambient light. For best results (short authentication time), the area surrounding the transmitting LED, in most cases the enclosure of the IoT device, should be white or gray. Finally, the smartphone camera's maximum frame rate determines the visible-light channel's modulation rate. The camera's maximum frame rate ultimately determines the time it takes to authenticate a device.

8.10.1 Acknowledgments

We thank Alexander Linssen and Hagen Odenthal, both visiting students from Universität der Bundeswehr München, for help with the prototype implementation and evaluation. Alexander implemented the Android authenticator application. Hagen implemented both Raspberry Pi transmitters.

Chapter 9: Conclusion

This dissertation studied management complexity in cyber-physical systems spanning geographical or administrative boundaries. The study is based on practical research with different classes of systems ranging from the geographically dispersed U.S. electrical grid to small-scale consumer IoT systems in shared physical environments. Designing and operating such systems presents many unique scalability, reliability, security, and management complexity challenges. Collectively, the main contributions of this dissertation are a new network architecture, network services, and protocols designed to help reduce management complexity in large, evolving, heterogeneous, or federated cyber-physical systems. We find that selectively applying autonomic principles to new protocols and services can help simplify the management of deployed cyber-physical systems. However, the effectiveness of this approach depends on the specific deployment scenario and, most importantly, the portion of legacy infrastructure within the deployed system.

We presented an analysis of the U.S. electrical grid networking subsystem, designed PhoenixSEN, an ad hoc backup network to replace the primary operational networks in emergencies, and discussed the experimental evaluation of PhoenixSEN on a realistic physical grid testbed in exercises conducted by DARPA. The novelty of PhoenixSEN lies in the combination of existing and new network technologies and built-in services for rapid deployment after a blackout, when the primary networks or the public Internet may not be available.

PhoenixSEN is designed to be deployable by the power distribution industry, i.e., operators who do not necessarily have a networking background. This made the design, selection of network services, and their configuration particularly challenging. The network is designed as self-configuring to a large extent and requires only minimal input from the operator during deployment. This was achieved with automated network configuration, network-wide service discovery, and network services capable of operating when disconnected from the network. We believe a similar network design might be helpful during regular cyber-physical system operations, not just in emergencies.

During DARPA RADICS exercises, the PhoenixSEN network was also used by forensic teams to identify and isolate potentially compromised SCADA devices in the electrical grid. Running third-party applications, e.g., existing forensic tools, on a highly dynamic ad hoc network like PhoenixSEN, without public Internet access, is challenging. This was also the case for our custom-built network monitoring application.

To address the problem, we presented SenSQL, a geographic resource discovery and query processing service designed to help applications discover sensors and their stored measurement data in federated cyber-physical systems operated by multiple independent administrative entities. The SenSQL service provides a familiar declarative-style SQL interface to applications. The novel aspects of the service are the use of the LoST protocol to discover SQL databases at the network edge, a stable geographic resource naming architecture, and the ability for the application to submit queries to the data, i.e., to the SQL database that store sensor measurements at the network edge, near where the data was produced.

SenSQL is a network service we wished PhoenixSEN could have provided to third-party applications during DARPA RADICS exercises.

DARPA stressed the importance of interpersonal voice communications for the success of the grid recovery process. PhoenixSEN has built-in VoIP service to meet that requirement with a self-managing design that provides graceful service degradation in the case of network partitions. NIST has also recognized the importance of high-quality voice communications in emergencies and funded research to design the tools to evaluate the quality of experience of voice communications for first responders. We presented the design and implementation of a mission-critical voice testbed for experimental research into mission-critical voice communications. We used the testbed in human-subject studies with trained first responders in a controlled environment.

Although initially designed for the NIST project, the testbed would have helped evaluate the operation of the VoIP service in PhoenixSEN under various emulated scenarios and conditions.

The second part of the dissertation focused on the intricacies of IoT device network enrollment.

Network enrollment is typically the first step in the life cycle of a new IoT device. That process is often complicated, frustrating, and error-prone, particularly with consumer-oriented IoT devices without a user interface.

To better understand the challenges associated with IoT network enrollment, we reverse-engineered, analyzed, and documented the Wi-Fi pairing process of Amazon Echo, one of the most popular consumer IoT devices. We analyzed the protocol between the Echo and a pairing smartphone application and decrypted and analyzed all communication between the Echo and Amazon cloud. We learned that the pairing process is rather complicated, may disrupt internet connectivity on the smartphone, and that the smartphone uses the Echo as an internet proxy to associate the device with Amazon Cloud. To our surprise, we learned that a newly purchased Echo is pre-registered with the buyer's Amazon account.

Prompted by practical experience with enrolling the Amazon Echo [1] and other consumer-grade IoT devices [182], we analyzed a variety of existing network enrollment frameworks and protocols from a usability perspective. We proposed WIDE, a network enrollment framework for advanced deployment scenarios. WIDE is an abstract network enrollment framework designed to enable advanced deployment scenarios unsupported by existing network enrollment protocols. Such deployment scenarios involve shared physical spaces (e.g., Airbnbs), installations by specialists, or IoT systems that "come with the environment", i.e., fixed connected infrastructures. WIDE considers the entire lifecycle of IoT devices and is link-layer agnostic. We also discuss the design of an IRB-approved human-subject usability study to compare and contrast WIDE with other network enrollment frameworks.

A secure network must authenticate a new IoT device before it can be enrolled. Existing authentication methods rely on physical access, proximity, or a pre-shared secret. Obtaining physical access may be impossible in many advanced deployment scenarios, e.g., when IoT devices are installed by a specialist in physically unreachable locations. We proposed Lighthouse, a visible-light authentication protocol for physically inaccessible IoT devices. We discussed the protocol's design, developed transmitter and receiver prototypes, and evaluated the system in a controlled environment.

We found that designing a visible-light authentication protocol for IoT devices and smartphone cameras is possible. Through realistic measurements, we obtained authentication time estimates and found them reasonable compared to authentication methods that involve gaining physical access to the device. We also illustrated how the visible-light authentication protocol could be used as an additional authentication method in the Wi-Fi Easy Connect framework [184].

9.1 Funding Acknowledgments

The research presented in Chapter 3 was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views and conclusions contained in this section are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. government. Distribution statement A. Distribution approved for public release, distribution unlimited. Not export controlled per ES-FL-020821-0013.

The work discussed in Chapters 4 and 7 was developed with funding from the National Science Foundation (NSF) Cyber-Physical Systems (CPS) Program, Award # 1932418.

The work presented in Chapter 5 was funded by the National Institute of Standards and Technology (NIST) under the Federal Award ID 70NANB19H003.

References

[1] J. Janak, T. Tseng, A. Isaacs, and H. Schulzrinne, “An analysis of Amazon Echo’s network behavior,” in *2021 IEEE Global Communications Conference: IoT and Sensor Networks (Globecom2021 IoTSN)*, Madrid, Spain, Dec. 2021.

[2] J. Janak, H. Retty, D. A. Chee, A. Baloian, and H. Schulzrinne, “Talking after lights out: An ad hoc network for electric grid recovery,” in *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm) (IEEE SmartGridComm’21)*, Aachen, Germany, Oct. 2021.

[3] G. Montenegro, C. Schumacher, and N. Kushalnagar, “IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals,” RFC Editor, RFC 4919, Aug. 2007, 12 pp.

[4] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, “RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks,” RFC 6550, Mar. 2012, 157 pp.

[5] C. Bormann, M. Ersue, and A. Keränen, “Terminology for Constrained-Node Networks,” RFC Editor, RFC 7228, May 2014.

[6] J. H. Saltzer, D. P. Reed, and D. D. Clark, “End-to-end arguments in system design,” *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 4, pp. 277–288, 1984.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin, “Simple Network Management Protocol (SNMP),” RFC Editor, RFC 1157, May 1990.

[8] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network Configuration Protocol (NETCONF),” RFC Editor, RFC 6241, Jun. 2011.

[9] M. Björklund, “The YANG 1.1 Data Modeling Language,” RFC Editor, RFC 7950, Aug. 2016.

[10] DARPA. “Technologies to rapidly restore the electrical grid after cyber attack come online.” (2021), [Online]. Available: https://www.darpa.mil/news-events/2021-02-23 (visited on 06/01/2020).

[11] M. H. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. E. Carpenter, S. Jiang, and L. Ciavaglia, “Autonomic Networking: Definitions and Design Goals,” RFC Editor, RFC 7575, Jun. 2015.

[12] National Transportation Safety Board, "Pipeline Over-pressure of a Columbia Gas of Massachusetts Low-pressure Natural Gas Distribution System: Merrimack Valley, Massachusetts," Tech. Rep. PLD18MR003, Sep. 2018.

[13] J. E. Sullivan and D. Kamensky, "How cyber-attacks in Ukraine show the vulnerability of the U.S. power grid," *The Electricity Journal*, vol. 30, no. 3, pp. 30–35, 2017.

[14] The White House. "Remarks by the President on securing our Nation's cyber infrastructure." (2009), [Online]. Available: https://obamawhitehouse.archives.gov/the-press-office/remarks-president-securing-our-nations-cyber-infrastructure (visited on 05/05/2020).

[15] Idaho National Laboratory, "Vulnerability analysis of energy delivery control systems," Tech. Rep. INL/EXT-10-18381, Sep. 2011.

[16] J. Slay and M. Miller, "Lessons learned from the Maroochy water breach," in *International Conference on Critical Infrastructure Protection*, Springer, 2007, pp. 73–82.

[17] S. Gallagher. "IoT garage door opener maker bricks customer's product after bad review." (2017), [Online]. Available: https://arstechnica.com/information-technology/2017/04/iot-garage-door-opener-maker-bricks-customers-product-after-bad-review (visited on 11/17/2022).

[18] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th ed. Boston, MA: Pearson, 2016, ISBN: 978-0-13-359414-0.

[19] *IWCI'21: Proceedings of the Interdisciplinary Workshop on (de) Centralization in the Internet*, Virtual Event, Germany: Association for Computing Machinery, 2021, ISBN: 9781450391382.

[20] M. Nottingham, "Centralization, Decentralization, and Internet Standards," RFC Editor, RFC 9518, Dec. 2023, 22 pp.

[21] Defense Advanced Research Projects Agency (DARPA). "Rapid Attack Detection, Isolation, and Characterization Systems (RADICS) program." (2019), [Online]. Available: https://www.darpa.mil/program/rapid-attack-detection-isolation-and-characterization-systems (visited on 08/30/2019).

[22] FCC. "2017 Atlantic hurricane season impact on communications report and recommendations public safety docket no. 17-344." (Aug. 2018), [Online]. Available: https://docs.fcc.gov/public/attachments/DOC-353805A1.pdf (visited on 05/28/2019).

[23] Federal Government of the United States. "The federal response to hurricane Katrina: Lessons learned." (2006), [Online]. Available: https://georgewbush-whitehouse.archives.gov/reports/katrina-lessons-learned/ (visited on 05/04/2020).

[24] J. Janak. “DNS to Avahi gateway.” (2021), [Online]. Available: https://github.com/janakj/dns2avahi (visited on 09/20/2021).

[25] J. Janak, A. Baloian, D. Rubenstein, and H. Schulzrinne. “A platform for experimental research in mission-critical voice.” (2023), [Online]. Available: https://gitlab.com/irtlab/mcv-testbed (visited on 05/02/2023).

[26] C. Greer, M. Burns, D. Wollman, and E. Griffor, “Cyber-physical systems and Internet of Things,” National Institute of Standards and Technology (NIST), Tech. Rep. NIST.SP.1900-202, Mar. 2019.

[27] R. Alur, *Principles of cyber-physical systems*. MIT press, 2015, ISBN: 978-0-2620-2911-7.

[28] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[29] MIT Lincoln Laboratory, “Distributed sensor networks,” Massachusetts Institute of Technology (MIT), Tech. Rep. AD-A204719, Sep. 1986.

[30] J. Yick, B. Mukherjee, and D. Ghosal, “Wireless sensor network survey,” *Computer networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[31] S. Heath, *Embedded Systems Design*. Elsevier, 2002, ISBN: 978-0750655460.

[32] Wikipedia. “Embedded Systems.” (2023), [Online]. Available: https://en.wikipedia.org/wiki/Embedded_system (visited on 10/16/2023).

[33] E. A. Lee and S. A. Seshia, *Introduction to embedded systems: A cyber-physical systems approach*. MIT press, 2016.

[34] S. Karnouskos, “Stuxnet worm impact on industrial cyber-physical system security,” in *IECON 2011 - 37th Annual Conference of the IEEE Industrial Electronics Society*, Nov. 2011, pp. 4490–4494.

[35] CNN. “Massive failure leaves Argentina, Paraguay and Uruguay with no power.” (2019), [Online]. Available: https://edition.cnn.com/2019/06/16/world/power-outage-argentina-uruguay-paraguay (visited on 08/17/2019).

[36] BBC. “Major power failure affects homes and transport.” (2019), [Online]. Available: https://www.bbc.com/news/uk-49300025 (visited on 08/17/2019).

[37] National Grid ESO. “Interim report into the low frequency demand disconnection (LFDD) following generator trips and frequency excursion on 9 aug 2019.” (2019), [Online]. Available: https://www.nationalgrideso.com/document/151081/download (visited on 08/17/2019).

[38] L. Cottrell. “Network monitoring tools.” (2020), [Online]. Available: https://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html (visited on 05/08/2020).

[39] R. Presuhn, “Management Information Base (MIB) for the Simple Network Management Protocol (SNMP),” RFC Editor, RFC 3418, Dec. 2002.

[40] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[41] Z. Movahedi, M. Ayari, R. Langar, and G. Pujolle, “A survey of autonomic network architectures and evaluation criteria,” *IEEE Communications Surveys & Tutorials*, vol. 14, no. 2, pp. 464–490, 2011.

[42] D. Raymer, S. van der Meer, and J. Strassner, “From autonomic computing to autonomic networking: An architectural perspective,” in *Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems (EASE 2008)*, IEEE, 2008, pp. 174–183.

[43] J. Moy, “OSPF Version 2,” RFC Editor, RFC 2328, Apr. 1998.

[44] D. Oran, “OSI IS-IS Intra-domain Routing Protocol,” RFC Editor, RFC 1142, Feb. 1990.

[45] W. Eddy, “Transmission Control Protocol (TCP),” RFC Editor, RFC 9293, Aug. 2022, 98 pp.

[46] D. Mills, J. Martin, J. Burbank, and W. Kasch, “Network Time Protocol Version 4: Protocol and Algorithms Specification,” RFC Editor, RFC 5905, Jun. 2010.

[47] Internet Engineering Task Force (IETF). “Autonomic Networking Integrated Model and Approach (anima).” (2022), [Online]. Available: https://datatracker.ietf.org/wg/anima (visited on 11/19/2022).

[48] M. Behringer, C. Bormann, B. E. Carpenter, T. Eckert, J. Campos Nobre, S. Jiang, Y. Li, and M. C. Richardson, “Autonomic networking gets serious,” in *The Internet Protocol Journal*, The Internet Protocol Journal, vol. 24, Oct. 2021, pp. 2–18.

[49] M. H. Behringer, B. E. Carpenter, T. Eckert, L. Ciavaglia, and J. C. Nobre, “A Reference Model for Autonomic Networking,” RFC Editor, RFC 8993, May 2021, 26 pp.

[50] T. Eckert and M. H. Behringer, “Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM),” RFC Editor, RFC 8368, May 2018, 24 pp.

[51] C. Bormann, B. E. Carpenter, and B. Liu, “GeneRic Autonomic Signaling Protocol (GRASP),” RFC Editor, RFC 8990, May 2021, 55 pp.

[52] The Wall Street Journal. “The Texas grid came close to an even bigger disaster during February freeze.” (2020), [Online]. Available: https://www.wsj.com/articles/texas-electrical-grid-bigger-disaster-february-freeze-black-starts-11622124896 (visited on 05/13/2020).

[53] G. Andersson, P. Donalek, R. Farmer, N. Hatziargyriou, I. Kamwa, P. Kundur, N. Martins, J. Paserba, P. Pourbeik, J. Sanchez-Gasca, *et al.*, “Causes of the 2003 major grid blackouts in North America and Europe, and recommended means to improve system dynamic performance,” *IEEE transactions on Power Systems*, vol. 20, no. 4, pp. 1922–1928, 2005.

[54] National Grid ESO. “Black start.” (2019), [Online]. Available: https://www.nationalgrideso.com/balancing-services/system-security-services/black-start (visited on 09/13/2019).

[55] PJM. “PJM manual 12: Balancing operations.” (2019), [Online]. Available: https://www.pjm.com/~/media/documents/manuals/m12.ashx (visited on 08/13/2019).

[56] M. Adibi and L. Fink, “Power system restoration planning,” *IEEE Transactions on Power Systems*, vol. 9, no. 1, pp. 22–28, 1994.

[57] National Institute of Standards and Technology (NIST), “NIST framework and roadmap for smart grid interoperability standards, release 3.0,” Tech. Rep. NIST.SP.1108r3, Sep. 2014.

[58] Applied Technology Council, “Critical assessment of lifeline system performance: Understanding societal needs in disaster recovery,” National Institute of Standards and Technology (NIST), Tech. Rep. NIST.GCR.16-917039, Apr. 2016.

[59] IEEE, “IEEE standard for electric power systems communications-Distributed Network Protocol (DNP3),” *IEEE Std 1815-2012 (Revision of IEEE Std 1815-2010)*, pp. 1–821, 2012.

[60] International Electrotechnical Commission, “Telecontrol equipment and systems - part 6-503: Telecontrol protocols compatible with ISO standards and ITU-T recommendations - TASE.2 services and protocol,” *IEC Std 60870-6-503 TASE.2 Services and protocol*, 2014.

[61] International Electrotechnical Commission, “Communication networks and systems for power utility automation,” *IEC Std 61850*, 2013.

[62] Federal Energy Regulatory Commission. “Order no. 889: Open access same-time information system and standards of conduct.” (1996), [Online]. Available: https://www.ferc.gov/legal/maj-ord-reg/land-docs/order889.asp (visited on 05/04/2019).

[63] IEEE, “IEEE standard for synchrophasor measurements for power systems,” Tech. Rep. IEEE Std C37.118.1-2011, 2011.

[64] T. Clausen and P. Jacquet, “Optimized Link State Routing Protocol (OLSR),” RFC Editor, RFC 3626, Oct. 2003.

[65] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (VXLAN): A framework for overlaying virtualized layer 2 networks over layer 3 networks," RFC Editor, RFC 7348, Aug. 2014.

[66] J. Rosenberg, H. Schulzrinne, A. Johnston, G. Camarillo, J. Peterson, R. Sparks, M. J. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC Editor, RFC 3261, Jul. 2002, 269 pp.

[67] cz.nic. "Knot DNS." (2023), [Online]. Available: https://www.knot-dns.cz/ (visited on 10/08/2023).

[68] S. Cheshire and M. Krochmal, "Multicast DNS," RFC Editor, RFC 6762, Feb. 2013.

[69] NLNet Labs. "Unbound." (2023), [Online]. Available: https://nlnetlabs.nl/projects/unbound (visited on 10/08/2023).

[70] OLSR Project. "Olsrd GitHub Repository." (2023), [Online]. Available: https://github.com/OLSR/olsrd (visited on 10/08/2023).

[71] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC Editor, RFC 6455, Dec. 2011.

[72] M. McDonald, J. Mulder, B. Richardson, R. Cassidy, A. Chavez, N. Pattengale, G. Pollock, J. Urrea, M. Schwartz, W. Atkins, *et al.*, "Modeling and simulation for cyber-physical system security research, development and applications," Sandia National Laboratories, Tech. Rep. SAND2010-0568, 2010.

[73] M. M. Roomi, P. P. Biswas, D. Mashima, Y. Fan, and E.-C. Chang, "False data injection cyber range of modernized substation system," in *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, IEEE, 2020, pp. 1–7.

[74] A. Ashok, S. Krishnaswamy, and M. Govindarasu, "PowerCyber: A remotely accessible testbed for cyber physical security of the smart grid," in *2016 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, IEEE, 2016, pp. 1–5.

[75] National Electric Sector Cybersecurity Organization Resource (NESCOR). "Analysis of selected electric sector high risk failure scenarios." (2019), [Online]. Available: https://smartgrid.epri.com/NESCOR.aspx (visited on 10/18/2019).

[76] P. Gunathilaka, D. Mashima, and B. Chen, "Softgrid: A software-based smart grid testbed for evaluating substation cybersecurity solutions," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, 2016, pp. 113–124.

[77] F. Legendre, T. Hossmann, F. Sutton, and B. Plattner, "30 years of wireless ad hoc networking research: What about humanitarian and disaster relief solutions? What are we

still missing?" In *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief (ACWR'11)*, 2011, pp. 217–217.

[78] G. V. Kumar, Y. V. Reddyr, and D. M. Nagendra, "Current research work on routing protocols for MANET: A literature survey," *International Journal on Computer Science and Engineering*, vol. 2, no. 03, pp. 706–713, 2010.

[79] B. B. Lowekamp, "Combining active and passive network measurements to build scalable monitoring systems on the grid," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 4, pp. 19–26, Mar. 2003.

[80] W. John, S. Tafvelin, and T. Olovsson, "Passive Internet measurement: Overview and guidelines based on experiences," *Computer Communications*, vol. 33, no. 5, pp. 533–550, 2010.

[81] "Moloch full packet capture." (2020), [Online]. Available: https://molo.ch (visited on 05/08/2020).

[82] T. Oetiker. "The Multi Router Traffic Grapher." (2020), [Online]. Available: https://oss.oetiker.ch/mrtg (visited on 05/08/2020).

[83] The OpenNMS Group. "The OpenNMS platform." (2020), [Online]. Available: https://www.opennms.com (visited on 05/08/2020).

[84] Lily Hay Newman. "The Hail Mary plan to restart a hacked US electric grid." (Nov. 2018), [Online]. Available: https://www.wired.com/story/black-start-power-grid-darpa-plum-island (visited on 08/30/2019).

[85] Red Cross. "Red Cross disaster services technology." (Aug. 2019), [Online]. Available: http://www.tucsoncert.info/Red_Cross_DST_slides.pdf (visited on 08/19/2019).

[86] T. Hardie, A. Newton, H. Schulzrinne, and H. Tschofenig, "LoST: A Location-to-Service Translation Protocol," RFC Editor, RFC 5222, Aug. 2008.

[87] National Institute of Standards and Technology (NIST). "Report from the executive roundtable on cyber-physical systems: Strategic vision and business drivers for 21st century cyber-physical systems." (Jan. 2013), [Online]. Available: https://www.nist.gov/system/files/documents/el/Exec-Roundtable-SumReport-Final-1-30-13.pdf (visited on 12/11/2022).

[88] OpenJS Foundation. "Node-RED." (2022), [Online]. Available: https://nodered.org (visited on 01/13/2022).

[89] sisense. "Periscope data." (2022), [Online]. Available: https://www.sisense.com (visited on 01/13/2022).

[90] Python pandas Developers. “pandas.” (2022), [Online]. Available: https://pandas.pydata.org (visited on 01/16/2022).

[91] C. Davis, P. Vixie, T. Goodwin, and I. Dickinson, “A Means for Expressing Location Information in the Domain Name System,” RFC Editor, RFC 1876, Jan. 1996.

[92] T. Fioreze and G. Heijenk, “Extending the Domain Name System (DNS) to provide geographical addressing towards vehicular ad-hoc networks (VANETs),” in *2011 IEEE Vehicular Networking Conference (VNC)*, IEEE, 2011, pp. 70–77.

[93] B. Meijerink, M. Baratchi, and G. Heijenk, “An efficient geographical addressing scheme for the Internet,” in *International Conference on Wired/Wireless Internet Communication*, Springer, 2016, pp. 78–90.

[94] SRI International. “The proposed .geo top-level domain name.” (2000), [Online]. Available: http://www.ai.sri.com/dotgeo (visited on 12/30/2020).

[95] H. Schulzrinne, “Location-to-URL Mapping Architecture and Framework,” RFC Editor, RFC 5582, Sep. 2009.

[96] W. Song, J. W. Lee, and H. Schulzrinne, “Polygon simplification for location-based services using population density,” in *2011 IEEE International Conference on Communications (ICC)*, IEEE, 2011, pp. 1–6.

[97] A. Pavlo and M. Aslett, “What’s really new with NewSQL?” *ACM Sigmod Record*, vol. 45, no. 2, pp. 45–55, 2016.

[98] E. F. Codd, “A relational model of data for large shared data banks,” *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[99] S. G. Edward and N. Sabharwal, “Mongodb architecture,” in *Practical MongoDB*, Springer, 2015, pp. 95–157.

[100] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: Amazon’s highly available key-value store,” *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

[101] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, “Bigtable: A distributed storage system for structured data,” *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, pp. 1–26, 2008.

[102] A. Lakshman and P. Malik, “Cassandra: A decentralized structured storage system,” *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[103] R. Cattell, “Scalable SQL and NoSQL data stores,” *SIGMOD Rec.*, vol. 39, no. 4, pp. 12–27, May 2011.

[104] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, *et al.*, “Spanner: Google’s globally distributed database,” *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 3, pp. 1–22, 2013.

[105] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss, *et al.*, “CockroachDB: The resilient geo-distributed SQL database,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 1493–1509.

[106] Microsoft. “Multidimensional Expressions (MDX) language reference.” (2022), [Online]. Available: https://learn.microsoft.com/en-us/sql/mdx/multidimensional-expressions-mdx-reference (visited on 11/01/2022).

[107] ISO/IEC JTC 1/SC 32, “Information technology — Database languages — SQL multimedia and application packages — Part 3: Spatial,” International Organization for Standardization (ISO), Standard ISO/IEC 13249-3:2016, 2016.

[108] ISO/TC 211, “Geographic information — Simple feature access — Part 1: Common architecture,” International Organization for Standardization (ISO), Standard ISO 19125-1:2004, 2004.

[109] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub, “The GeoJSON Format,” RFC Editor, RFC 7946, Aug. 2016.

[110] P. Mockapetris, “Domain names - implementation and specification,” RFC Editor, RFC 1035, Nov. 1987.

[111] A. Gulbrandsen, P. Vixie, and L. Esibov, “A DNS RR for specifying the location of services (DNS SRV),” RFC Editor, RFC 2782, Feb. 2000.

[112] M. Mealling and R. Daniel, “The Naming Authority Pointer (NAPTR) DNS Resource Record,” RFC Editor, RFC 2915, Sep. 2000.

[113] M. Mealling, “Dynamic Delegation Discovery System (DDDS) Part One: The Comprehensive DDDS,” RFC Editor, RFC 3401, Oct. 2002.

[114] J. Peterson, “A Presence-based GEOPRIV Location Object Format,” RFC Editor, RFC 4119, Dec. 2005.

[115] M. Thomson and J. Winterbottom, “Revised Civic Location Format for Presence Information Data Format Location Object (PIDF-LO),” RFC Editor, RFC 5139, Feb. 2008.

[116] J. Winterbottom, M. Thomson, and H. Tschofenig, “GEOPRIV Presence Information Data Format Location Object (PIDF-LO) Usage Clarification, Considerations, and Recommendations,” RFC Editor, RFC 5491, Mar. 2009.

[117] M. Thomson and J. Winterbottom, “Representation of Uncertainty and Confidence in the Presence Information Data Format Location Object (PIDF-LO),” RFC Editor, RFC 7459, Feb. 2015.

[118] International Organization for Standardization (ISO), “Geographic information — Geography Markup Language (GML),” ISO, Standard ISO 19136, 2020.

[119] H. Schulzrinne, “Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information,” RFC Editor, RFC 4776, Nov. 2006.

[120] H. Schulzrinne and H. Tschofenig, “Synchronizing Service Boundaries and <mapping> Elements Based on the Location-to-Service Translation (LoST) Protocol,” RFC Editor, RFC 6739, Oct. 2012.

[121] K. Wolf, “Location-to-Service Translation (LoST) Service List Boundary Extension,” RFC Editor, RFC 6197, Apr. 2011.

[122] A. Forte and H. Schulzrinne, “Location-to-Service Translation (LoST) Protocol Extensions,” RFC Editor, RFC 6451, Dec. 2011.

[123] A. Silverstein and J. Follum, “High-resolution, time-synchronized grid monitoring devices,” North American SynchroPhasor Initiative (NASPI), Tech. Rep. NASPI-2020-TR-004, Mar. 2020.

[124] OpenStreetMap Foundation. “OpenStreetMap.” (2022), [Online]. Available: https://openstreetmap.org (visited on 11/26/2022).

[125] ISO/IEC JTC 1/SC 32, “Information technology - Database languages — SQL,” International Organization for Standardization (ISO), Standard ISO/IEC 9075, 2016.

[126] A. Mayrhofer and C. Spanring, “A Uniform Resource Identifier for Geographic Locations (’geo’ URI),” RFC Editor, RFC 5870, Jun. 2010.

[127] C. Jennings, Z. Shelby, J. Arkko, A. Keranen, and C. Bormann, “Sensor Measurement Lists (SenML),” Tech. Rep. 8428, Aug. 2018.

[128] J. Saltzer, “On the Naming and Binding of Network Destinations,” RFC Editor, RFC 1498, Aug. 1993.

[129] J. H. Saltzer, “Naming and binding of objects,” in *Operating Systems*, Springer, 1978, pp. 99–208.

[130] Overpass Authors. "Overpass API." (2023), [Online]. Available: https://overpass-api.de (visited on 11/01/2023).

[131] The OpenStreetMap Project. "OpenStreetMap Nominatim." (2023), [Online]. Available: https://nominatim.openstreetmap.org (visited on 11/01/2023).

[132] INTSIG. "CamScanner." (2022), [Online]. Available: https://www.camscanner.com/ (visited on 11/25/2022).

[133] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991.

[134] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, *et al.*, "Presto: SQL on everything," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, IEEE, 2019, pp. 1802–1813.

[135] OASIS. "MQTT Protocol." (2023), [Online]. Available: https://mqtt.org (visited on 11/08/2018).

[136] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," *ACM Transactions on database systems (TODS)*, vol. 30, no. 1, pp. 122–173, 2005.

[137] Y. Yao and J. Gehrke, "The Cougar approach to in-network query processing in sensor networks," *ACM Sigmod record*, vol. 31, no. 3, pp. 9–18, 2002.

[138] J.-Z. Sun and J. Zhou, "Querying Sensor Networks With Extended SQL," in *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*, IEEE, 2010, pp. 634–638.

[139] D. F. Bacon, N. Bales, N. Bruno, B. F. Cooper, A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi, E. Kogan, *et al.*, "Spanner: Becoming a SQL System," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 331–343.

[140] N. Tsiftes and A. Dunkels, "A database in every sensor," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '11, Seattle, Washington: ACM, 2011, pp. 316–332, ISBN: 978-1-4503-0718-5.

[141] H. Schulzrinne and R. Marshall, "Requirements for Emergency Context Resolution with Internet Technologies," RFC Editor, RFC 5012, Jan. 2008.

[142] Sumeer Bhola. "How We Built Scalable Spatial Data & Spatial Indexing in CockroachDB." (Dec. 2020), [Online]. Available: https://www.cockroachlabs.com/blog/how-we-built-spatial-indexing/ (visited on 12/18/2020).

[143] M. Mealling, “Dynamic Delegation Discovery System (DDDS) Part Two: The Algorithm,” RFC Editor, RFC 3402, Oct. 2002.

[144] M. Mealling, “Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database,” RFC Editor, RFC 3403, Oct. 2002.

[145] M. Mealling, “Dynamic Delegation Discovery System (DDDS) Part Four: The Uniform Resource Identifiers (URI),” RFC Editor, RFC 3404, Oct. 2002.

[146] S. Bradner, L. Conroy, and K. Fujiwara, “The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM),” RFC Editor, RFC 6116, Mar. 2011.

[147] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker, “ROFL: Routing on Flat Labels,” *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 363–374, 2006.

[148] B. Ford, “Unmanaged Internet Protocol: Taming the edge network management crisis,” *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 1, pp. 93–98, 2004.

[149] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, “A Layered Naming Architecture for the Internet,” in *Proceedings of the 2004 conference on applications, technologies, architectures, and protocols for computer communications - SIGCOMM ’04*, New York, New York, USA: ACM Press, 2004, p. 343, ISBN: 1581138628.

[150] W3C. “Web of Things.” (2018), [Online]. Available: https://www.w3.org/WoT/WG (visited on 11/20/2018).

[151] J. S. Dumas and J. C. Redish, *A practical guide to usability testing*. Intellect Books, 1999, ISBN: 978-1841500201.

[152] U.S. House of Representatives. “A failure of initiative: Final report of the select bipartisan committee to investigate the preparation for and response to hurricane Katrina.” (2006), [Online]. Available: https://www.congress.gov/congressional-report/109th-congress/house-report/377/1 (visited on 05/04/2020).

[153] Fitzpatrick, Leo and Scurato, Carmen and Torres, Joseph. “Connecting the dots. The telecommanications crisis in Puerto Rico,” Free Press. (May 2019), [Online]. Available: https://www.freepress.net/sites/default/files/2019-05/connecting_the_dots_the_telecom_crisis_in_puerto_rico_free_press.pdf (visited on 05/23/2019).

[154] Campbel, Alexia Fernandez. “It took 11 months to restore power to Puerto Rico,” Vox. (2018), [Online]. Available: https://www.vox.com/identities/2018/8/15/17692414/puerto-rico-power-electricity-restored-hurricane-maria (visited on 05/28/2019).

[155] M. Stute, M. Maass, T. Schons, M.-A. Kaufhold, C. Reuter, and M. Hollick, "Empirical insights for designing information and communication technology for international disaster response," *International Journal of Disaster Risk Reduction*, pp. 101–598, 2020.

[156] L. K. Comfort and T. W. Haase, "Communication, coherence, and collective action: The impact of hurricane Katrina on communications infrastructure," *Public Works Management & Policy*, vol. 10, no. 4, pp. 328–343, 2006.

[157] L. Johnson, T. D. O'Rourke, S. Chang, C. A. Davis, L. D.-O. I. Robertson, H. Schulzrinne, and K. Tierney, "Critical assessment of lifeline system performance: Understanding societal needs in disaster recovery," National Institute of Standards and Technology (NIST), Tech. Rep. NIST CGR 16-917-39, Apr. 2016.

[158] Baresip contributors. "Baresip." (2022), [Online]. Available: https://github.com/baresip/baresip (visited on 11/25/2022).

[159] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC Editor, RFC 3550, Jul. 2003.

[160] Project Jupyter. "JupyterHub." (2022), [Online]. Available: https://jupyter.org/hub (visited on 12/01/2022).

[161] U.S. Department of Homeland Security. "Enhanced Dynamic Geo-Social Environment (EDGE)." (2022), [Online]. Available: https://www.dhs.gov/science-and-technology/EDGE (visited on 11/01/2022).

[162] Amazon. "Amazon quarterly results." (2018), [Online]. Available: https://ir.aboutamazon.com/quarterly-results (visited on 11/26/2018).

[163] NPR and Edison Research. "The smart audio report." (2018), [Online]. Available: https://www.nationalpublicmedia.com/smart-audio-report/latest-report (visited on 11/26/2018).

[164] Amazon. "Alexa in Higher Education." (2018), [Online]. Available: https://developer.amazon.com/alexa/education (visited on 11/26/2018).

[165] Amazon. "Alexa for hospitality." (2018), [Online]. Available: https://www.amazon.com/alexahospitality (visited on 11/26/2018).

[166] A. Vanderpot. "Echohacking wiki." (2018), [Online]. Available: https://github.com/echohacking/wiki (visited on 11/20/2018).

[167] M. Barnes. "Alexa, are you listening?" (2017), [Online]. Available: https://labs.mwrinfosecurity.com/blog/alexa-are-you-listening (visited on 11/20/2018).

[168] R. Moskowitz, D. Karrenberg, Y. Rekhter, E. Lear, and G. J. de Groot, “Address Allocation for Private Internets,” RFC Editor, RFC 1918, Feb. 1996, 9 pp.

[169] Apache Software Foundation. “Apache Thrift.” (2018), [Online]. Available: https://thrift.apache.org/ (visited on 11/23/2018).

[170] R. Housley, “Cryptographic Message Syntax (CMS),” RFC Editor, RFC 5652, Sep. 2009.

[171] I. Clinton, L. Cook, and S. Banik. “A Survey of Various Methods for Analyzing the Amazon Echo.” (2016), [Online]. Available: https://vanderpot.com/Clinton_Cook_Paper.pdf (visited on 05/04/2021).

[172] S. Vasile, D. Oswald, and T. Chothia, “Breaking all the things—a systematic survey of firmware extraction techniques for iot devices,” in *International Conference on Smart Card Research and Advanced Applications*, Springer, 2018, pp. 171–185.

[173] iFixit. “Amazon Echo teardown.” (2018), [Online]. Available: https://www.ifixit.com/Teardown/Amazon+Echo+Teardown/33953 (visited on 11/21/2018).

[174] W. Haack, M. Severance, M. Wallace, and J. Wohlwend. “Security Analysis of the Amazon Echo.” (2017), [Online]. Available: https://courses.csail.mit.edu/6.857/2017/project/8.pdf (visited on 11/20/2018).

[175] M. Ford and W. Palmer, “Alexa, are you listening to me? An analysis of Alexa voice service network traffic,” *Personal and Ubiquitous Computing*, pp. 1–13, 2018.

[176] H. Chung, J. Park, and S. Lee, “Digital Forensic Approaches for Amazon Alexa Ecosystem,” *Digital Investigation*, vol. 22, S15–S25, 2017.

[177] D. Weinstein. “Amazon Echo Dot Explorations (Part 1).” (2018), [Online]. Available: https://medium.com/@dweinstein/amazon-echo-dot-explorations-part-1-5f5ae38ffeab (visited on 11/21/2018).

[178] N. Apthorpe, D. Reisman, and N. Feamster, “A smart home is no castle: Privacy vulnerabilities of encrypted IoT traffic,” *arXiv preprint 1705.06805*, 2017.

[179] M. Chetty, J.-Y. Sung, and R. E. Grinter, “How Smart Homes Learn: The Evolution of the Networked Home and Household,” in *9th International Conference on Ubiquitous Computing (UbiComp 2007), Innsbruck, Austria*, Springer, 2007, pp. 127–144.

[180] L. Hao, V. Chanda, and H. Schulzrinne. “IoT Home Device Usability Evaluation.” (2022), [Online]. Available: https://www.cs.columbia.edu/~lyhao/paper/iotusability.pdf (visited on 10/02/2023).

[181] R. E. Grinter, W. K. Edwards, M. Chetty, E. S. Poole, J.-Y. Sung, J. Yang, A. Crabtree, P. Tolmie, T. Rodden, C. Greenhalgh, *et al.*, “The Ins and Outs of Home Networking: The Case for Useful and Usable Domestic Networking,” *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 16, no. 2, pp. 1–28, 2009.

[182] V. Chanda, L. Hao, and H. Schulzrinne, “From Frustration to Function: A Study on Usability Challenges in Smart Home IoT Devices,” in *Third International Workshop on IoT Interoperability and the Web of Things (IEEE IIWOT 2024)*, IEEE, 2024.

[183] V. G. Cerf, “A revised view of the IoT ecosystem,” *IEEE Internet Computing*, 2017.

[184] Wi-Fi Alliance. “Wi-Fi Easy Connect.” (2018), [Online]. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-easy-connect (visited on 11/20/2018).

[185] Wi-Fi Alliance. “Wi-Fi Protected Setup.” (2020), [Online]. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-protected-setup (visited on 11/20/2022).

[186] Bluetooth SIG, *Bluetooth core specification*, https://www.bluetooth.com/specifications/adopted-specifications, 2017.

[187] The ZigBee Alliance, *ZigBee Specification revision 21*, http://www.zigbee.org, Aug. 2015.

[188] Thread Group. “Thread commissioning.” (2016), [Online]. Available: http://threadgroup.org (visited on 11/09/2016).

[189] Statista. “Market Insights: Internet of Things - Worldwide.” (2023), [Online]. Available: https://www.statista.com/outlook/tmo/internet-of-things/worldwide (visited on 11/11/2023).

[190] Allied Market Research. “Consumer IoT Market Outlook - 2030.” (2023), [Online]. Available: https://www.alliedmarketresearch.com/consumer-iot-market-A12703 (visited on 11/11/2023).

[191] S. Mennicken and E. M. Huang, “Hacking the Natural Habitat: An in-the-wild study of smart homes, their development, and the people who live in them,” in *Pervasive Computing: 10th International Conference, Pervasive 2012, Newcastle, UK, June 18-22, 2012. Proceedings 10*, Springer, 2012, pp. 143–160.

[192] Sonoff. “Sonoff Smart Stackable Power Meter.” (2023), [Online]. Available: https://sonoff.tech/product/diy-smart-switches/spm-main-spm-4relay (visited on 11/08/2018).

[193] Sonoff. “Sonoff MINIR2.” (2022), [Online]. Available: https://sonoff.tech/product/diy-smart-switch/minir2 (visited on 01/13/2022).

[194] S. Mare, F. Roesner, and T. Kohno, “Smart devices in airbnbs: Considering privacy and security for both guests and hosts.,” *Proc. Priv. Enhancing Technol.*, vol. 2020, no. 2, pp. 436–458, 2020.

[195] C. Geeng and F. Roesner, “Who’s In Control? Interactions In Multi-User Smart Homes,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–13.

[196] E. Zeng and F. Roesner, “Understanding and Improving Security and Privacy in Multi-User Smart Homes: A Design Exploration and In-Home User Study,” in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 159–176.

[197] H. Tschofenig, J. Arkko, D. Thaler, and D. R. McPherson, “Architectural Considerations in Smart Object Networking,” RFC Editor, RFC 7452, Mar. 2015, 24 pp.

[198] C. Haschek. “The curious case of the Raspberry Pi in the network closet.” (2018), [Online]. Available: https://blog.haschek.at/2018/the-curious-case-of-the-RasPi-in-our-network.html (visited on 02/26/2019).

[199] D. Heinze, J. Classen, and F. Rohrbach, “MagicPairing: Apple’s Take on Securing Bluetooth Peripherals,” in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, pp. 111–121.

[200] M. Sethi, B. Sarikaya, and D. Garcia-Carillo, “Terminology and processes for initial security setup of IoT devices,” IETF Secretariat, Internet-Draft draft-irtf-t2trg-secure-bootstrapping-03.txt, Nov. 2022.

[201] Internet Engineering Task Force (IETF). “IETF.” (2023), [Online]. Available: https://www.ietf.org/ (visited on 11/09/2018).

[202] IEEE. “IEEE.” (2023), [Online]. Available: https://www.ieee.org/ (visited on 11/09/2018).

[203] FIDO Alliance. “FIDO Alliance.” (2023), [Online]. Available: https://fidoalliance.org/ (visited on 11/09/2018).

[204] Open Connectivity Foundation. “Open Connectivity Foundation.” (2023), [Online]. Available: https://openconnectivity.org/ (visited on 11/09/2018).

[205] Open Mobile Alliance. “Open Mobile Alliance.” (2023), [Online]. Available: https://omaspecworks.org/ (visited on 11/09/2018).

[206] F. Stajano, “The resurrecting duckling,” in *International workshop on security protocols*, Springer, 1999, pp. 183–194.

[207] F. Stajano, “The resurrecting duckling—what next?” In *International Workshop on Security Protocols*, Springer, 2000, pp. 204–214.

[208] S. Viehboeck. “Brute forcing Wi-Fi Protected Setup – when poor design meets poor implementation.” (Dec. 2011), [Online]. Available: https://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf (visited on 07/08/2017).

[209] *IEEE 802.11: Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, 2012.

[210] C. Rigney, S. Willens, A. Rubens, and W. Simpson, “Remote Authentication Dial In User Service (RADIUS),” RFC Editor, RFC 2865, Jun. 2000.

[211] V. Fajardo, J. Arkko, J. A. Loughney, and G. Zorn, “Diameter Base Protocol,” RFC Editor, RFC 6733, Oct. 2012, 152 pp.

[212] Google. “Google Chromecast.” (2023), [Online]. Available: https://store.google.com/us/product/chromecast_google_tv (visited on 11/01/2023).

[213] Google. “Google Home.” (2023), [Online]. Available: https://home.google.com (visited on 11/01/2023).

[214] AllSeen Alliance. “AllJoyn onboarding framework.” (2017), [Online]. Available: https://allseenalliance.org/framework/documentation/learn/base-services/onboarding (visited on 07/13/2017).

[215] Connectivity Standards Alliance. “Matter.” (2022), [Online]. Available: https://csa-iot.org/all-solutions/matter/ (visited on 02/03/2022).

[216] T. Taubert and C. A. Wood, “SPAKE2+, an Augmented Password-Authenticated Key Exchange (PAKE) Protocol,” RFC Editor, RFC 9383, Sep. 2023, 25 pp.

[217] Amazon. “Frustration-Free Setup.” (2022), [Online]. Available: https://developer.amazon.com/docs/frustration-free-setup/understanding-ffs.html (visited on 12/01/2022).

[218] Apple. “HomeKit accessory protocol specification.” (2021), [Online]. Available: https://developer.apple.com/homekit (visited on 01/08/2021).

[219] T. Wu, “The SRP Authentication and Key Exchange System,” RFC Editor, RFC 2945, Sep. 2000.

[220] Y. Desmedt, “Station-to-station protocol,” in *Encyclopedia of Cryptography and Security*, Springer, 2011, pp. 1256–1256.

[221] T. Aura, M. Sethi, and A. Peltonen, “Nimble Out-of-Band Authentication for EAP (EAP-NOOB),” RFC Editor, RFC 9140, Dec. 2021, 51 pp.

[222] J. Vollbrecht, J. D. Carlson, L. Blunk, D. B. D. Aboba, and H. Levkowetz, “Extensible Authentication Protocol (EAP),” RFC Editor, RFC 3748, Jun. 2004, 67 pp.

[223] *IEEE 802.1X: Standard for Local and metropolitan area networks–Port-Based Network Access Control*, 2010.

[224] A. Y. Lindell, “Comparison-based key exchange and the security of the numeric comparison mode in Bluetooth v2.1,” in *Topics in Cryptology–CT-RSA 2009*, Springer, 2009, pp. 66–83.

[225] Y. Shaked and A. Wool, “Cracking the Bluetooth PIN,” in *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, ACM, 2005, pp. 39–50.

[226] M. Ryan *et al.*, “Bluetooth: With low energy comes low security,” *7th USENIX Workshop on Offensive Technologies (WOOT)*, vol. 13, pp. 4–4, 2013.

[227] A. Y. Lindell, “Attacks on the pairing protocol of Bluetooth v2.1,” *Black Hat USA, Las Vegas, Nevada*, 2008.

[228] J. Classen and M. Hollick, “Happy MitM: Fun and Toys in Every Bluetooth Device,” in *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021, pp. 72–77.

[229] Google. “Google Fast Pair Service.” (2023), [Online]. Available: https://developers.google.com/nearby/fast-pair (visited on 11/09/2018).

[230] Google. “Nearby Platform.” (2023), [Online]. Available: https://developers.google.com/nearby (visited on 10/10/2023).

[231] M. Pritikin, M. Richardson, T. Eckert, M. H. Behringer, and K. Watsen, “Bootstrapping Remote Secure Key Infrastructure (BRSKI),” RFC Editor, RFC 8995, May 2021, 116 pp.

[232] The ZigBee Alliance, *ZigBee Base Device Behavior Specification Version 1.0*, http://www.zigbee.org, Feb. 2016.

[233] *IEEE 802.15.4: Standard for Local and metropolitan area networks–Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*, 2011.

[234] The ZigBee Alliance, *ZigBee Light Link Standard Version 1.0*, http://www.zigbee.org, Apr. 2012.

[235] E. Rescorla and N. Modadugu, “Datagram Transport Layer Security,” RFC Editor, RFC 4347, Apr. 2006.

[236] F. Hao, "J-PAKE: Password-Authenticated Key Exchange by Juggling," RFC Editor, RFC 8236, Sep. 2017.

[237] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC Editor, RFC 7252, Jun. 2014.

[238] A. B. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home Automation in the Wild: Challenges and Opportunities," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 2115–2124.

[239] R. E. Grinter, W. K. Edwards, M. W. Newman, and N. Ducheneaut, "The work to make a home network work," in *ECSCW 2005: Proceedings of the Ninth European Conference on Computer-Supported Cooperative Work, 18–22 September 2005, Paris, France*, Springer, 2005, pp. 469–488.

[240] M. O. Thomas, B. A. Onyimbo, and R. Logeswaran, "Usability Evaluation Criteria for Internet of Things," *International Journal of Information Technology and Computer Science*, vol. 8, pp. 10–18, 2016.

[241] M. O. Jewell, E. Costanza, and J. Kittley-Davies, "Connecting the Things to the Internet: An Evaluation of Four Configuration Strategies for Wi-Fi Devices with Minimal User Interfaces," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, pp. 767–778.

[242] M. K. Chong and H. Gellersen, "How users associate wireless devices," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 1909–1918.

[243] E. Uzun, K. Karvonen, and N. Asokan, "Usability analysis of secure pairing methods," in *International Conference on Financial Cryptography and Data Security*, Springer, 2007, pp. 307–324.

[244] V. G. Cerf, "Things and the Net," *IEEE Internet Computing*, vol. 16, no. 6, pp. 96–96, 2012.

[245] R. Atkinson and S. Kent, *Security Architecture for the Internet Protocol*, RFC 2401, RFC, Nov. 1998.

[246] E. Rescorla and T. Dierks, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246, RFC, Aug. 2008.

[247] Wikipedia. "Captive Portal." (2023), [Online]. Available: https://en.wikipedia.org/wiki/Captive_portal (visited on 11/01/2023).

[248] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC Editor, RFC 5280, May 2008.

[249] D. Crocker and P. Overell, “Augmented BNF for Syntax Specifications: ABNF,” RFC Editor, RFC 5234, Jan. 2008.

[250] P. Kyzivat, “Case-Sensitive String Support in ABNF,” RFC Editor, RFC 7405, Dec. 2014.

[251] T. Berners-Lee, R. Fielding, and L. Masinter, “Uniform Resource Identifiers (URI): Generic Syntax,” RFC Editor, RFC 2396, Aug. 1998.

[252] M. Sethi, J. P. Mattsson, and S. Turner, “Handling Large Certificates and Long Certificate Chains in TLS-Based EAP Methods,” RFC Editor, RFC 9191, Feb. 2022, 12 pp.

[253] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC Editor, RFC 8446, Aug. 2018, 160 pp.

[254] J. P. Mattsson and M. Sethi, “EAP-TLS 1.3: Using the Extensible Authentication Protocol with TLS 1.3,” RFC Editor, RFC 9190, Feb. 2022, 31 pp.

[255] A. DeKok, “The Network Access Identifier,” RFC Editor, RFC 7542, May 2015, 30 pp.

[256] D. Harkins, “Public Key Exchange (PKEX),” IETF Secretariat, Internet-Draft draft-harkins-pkex-06.txt, Aug. 2018.

[257] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, “RaptorQ Forward Error Correction Scheme for Object Delivery,” RFC Editor, RFC 6330, Aug. 2011.

[258] P. Koopman and T. Chakravarty, “Cyclic redundancy code (CRC) polynomial selection for embedded networks,” in *International Conference on Dependable Systems and Networks, 2004*, IEEE, 2004, pp. 145–154.

[259] Pigpio Contributors. “The Pigpio library.” (2018), [Online]. Available: http://abyz.me.uk/rpi/pigpio (visited on 12/03/2018).

[260] The OpenRQ Project. “OpenRQ: an open-source RaptorQ implementation.” (2014), [Online]. Available: https://openrq-team.github.io/openrq/ (visited on 11/16/2022).

[261] The OpenCV team. “OpenCV: a real-time computer vision library.” (2022), [Online]. Available: https://opencv.org/ (visited on 11/17/2022).

[262] C. Rohner, S. Raza, D. Puccinelli, and T. Voigt, “Security in visible light communication: Novel challenges and opportunities,” *Sensors & Transducers Journal*, vol. 192, no. 9, pp. 9–15, 2015.

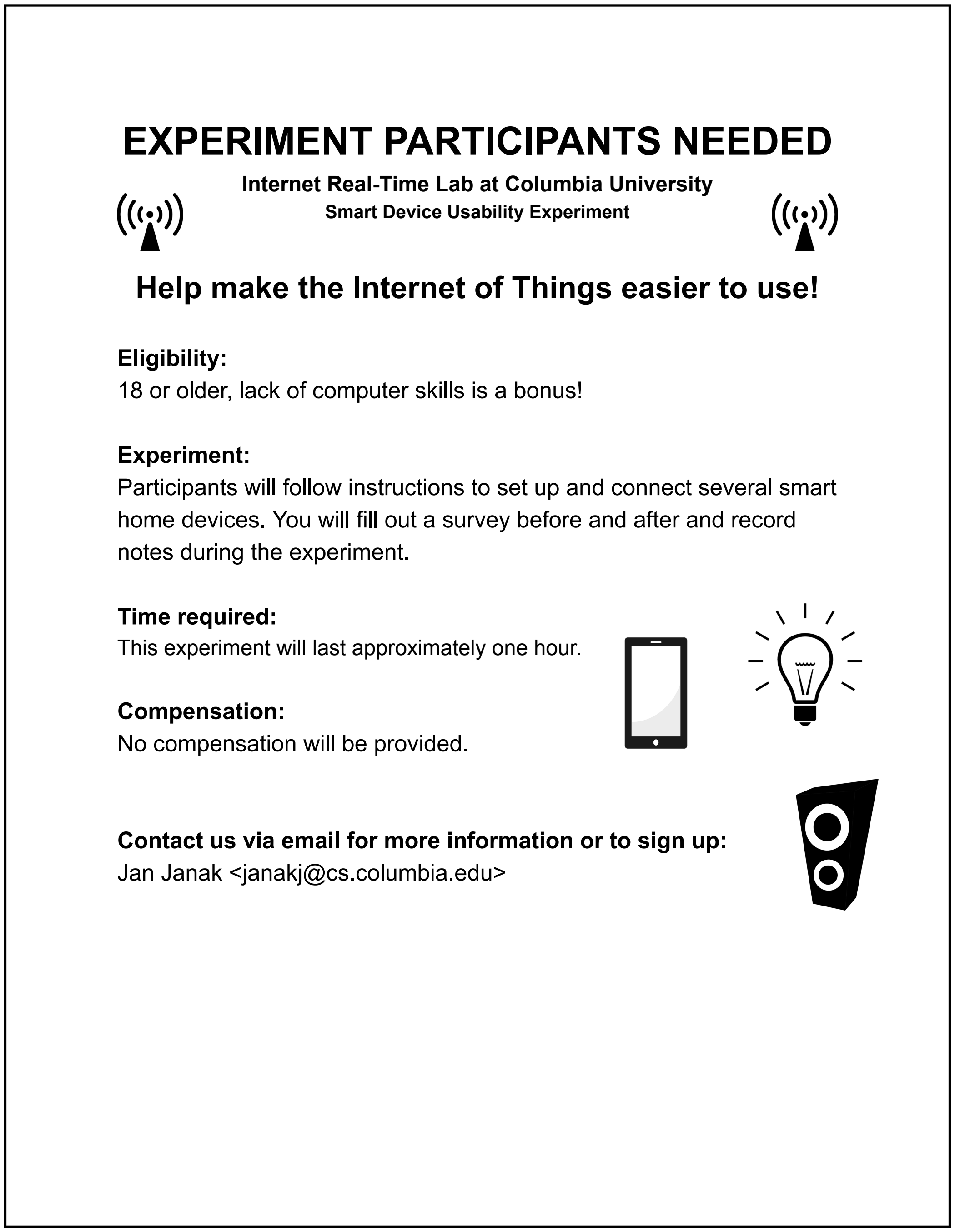
[263] The Electric Imp. “Electric Imp’s BlinkUp Technology.” (2022), [Online]. Available: https://developer.electricimp.com/manufacturing/sdkdocs/whatisblinkup (visited on 11/16/2022).

[264] Z. Zhao, G. Min, Y. Pang, W. Gao, and J. Lv, "Towards fast and reliable WiFi authentication by utilizing visible light diversity," in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, IEEE, 2019, pp. 1–9.

[265] T. Perković, M. Čagalj, and T. Kovačević, "Lisa: Visible light based initialization and SMS based authentication of constrained IoT devices," *Future Generation Computer Systems*, vol. 97, pp. 105–118, 2019.

[266] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication," in *2005 IEEE Symposium on Security and Privacy (S&P'05)*, IEEE, 2005, pp. 110–124.

Appendix A: On Usability of Network Enrollment in Cyber-Physical Systems



EXPERIMENT PARTICIPANTS NEEDED

Internet Real-Time Lab at Columbia University

Smart Device Usability Experiment

Help make the Internet of Things easier to use!

Eligibility:
18 or older, lack of computer skills is a bonus!

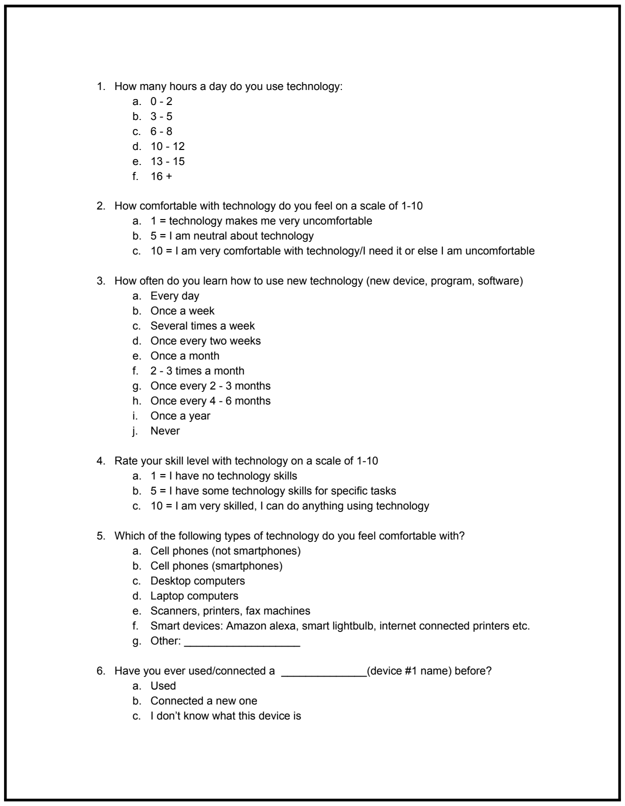
Experiment:
Participants will follow instructions to set up and connect several smart home devices. You will fill out a survey before and after and record notes during the experiment.

Time required:
This experiment will last approximately one hour.

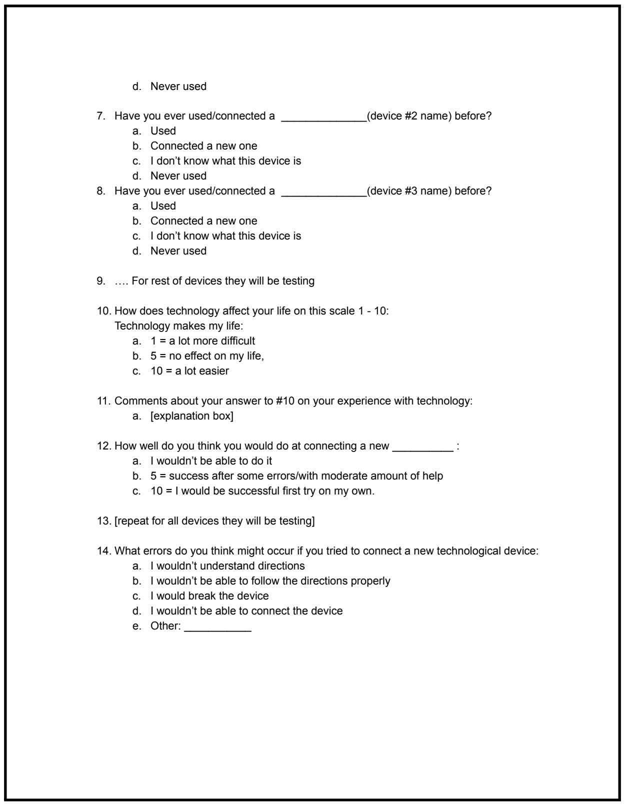
Compensation:
No compensation will be provided.

Contact us via email for more information or to sign up:
Jan Janak <janakj@cs.columbia.edu>

Figure A.1: Recruitment flyer



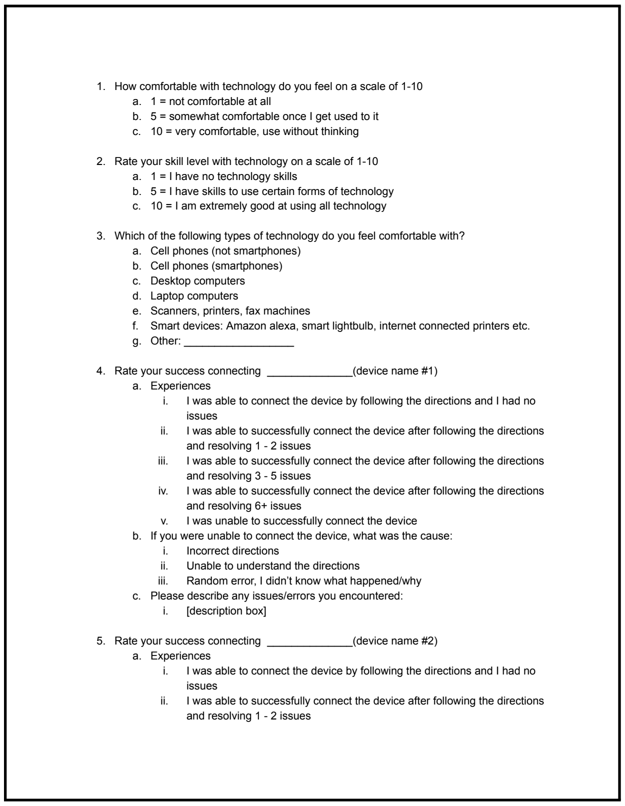
1. How many hours a day do you use technology:
    a. 0 - 2
    b. 3 - 5
    c. 6 - 8
    d. 10 - 12
    e. 13 - 15
    f. 16 +
2. How comfortable with technology do you feel on a scale of 1-10
    a. 1 = technology makes me very uncomfortable
    b. 5 = I am neutral about technology
    c. 10 = I am very comfortable with technology/I need it or else I am uncomfortable
3. How often do you learn how to use new technology (new device, program, software)
    a. Every day
    b. Once a week
    c. Several times a week
    d. Once every two weeks
    e. Once a month
    f. 2 - 3 times a month
    g. Once every 2 - 3 months
    h. Once every 4 - 6 months
    i. Once a year
    j. Never
4. Rate your skill level with technology on a scale of 1-10
    a. 1 = I have no technology skills
    b. 5 = I have some technology skills for specific tasks
    c. 10 = I am very skilled, I can do anything using technology
5. Which of the following types of technology do you feel comfortable with?
    a. Cell phones (not smartphones)
    b. Cell phones (smartphones)
    c. Desktop computers
    d. Laptop computers
    e. Scanners, printers, fax machines
    f. Smart devices: Amazon alexa, smart lightbulb, internet connected printers etc.
    g. Other: ___________________
6. Have you ever used/connected a ______________(device #1 name) before?
    a. Used
    b. Connected a new one
    c. I don't know what this device is



    d. Never used

7. Have you ever used/connected a ______________(device #2 name) before?
    a. Used
    b. Connected a new one
    c. I don't know what this device is
    d. Never used
8. Have you ever used/connected a ______________(device #3 name) before?
    a. Used
    b. Connected a new one
    c. I don't know what this device is
    d. Never used
9. …. For rest of devices they will be testing
10. How does technology affect your life on this scale 1 - 10:
    Technology makes my life:
    a. 1 = a lot more difficult
    b. 5 = no effect on my life,
    c. 10 = a lot easier
11. Comments about your answer to #10 on your experience with technology:
    a. [explanation box]
12. How well do you think you would do at connecting a new __________ :
    a. I wouldn't be able to do it
    b. 5 = success after some errors/with moderate amount of help
    c. 10 = I would be successful first try on my own.
13. [repeat for all devices they will be testing]
14. What errors do you think might occur if you tried to connect a new technological device:
    a. I wouldn't understand directions
    b. I wouldn't be able to follow the directions properly
    c. I would break the device
    d. I wouldn't be able to connect the device
    e. Other: ___________

Figure A.2: Pre-experiment survey form



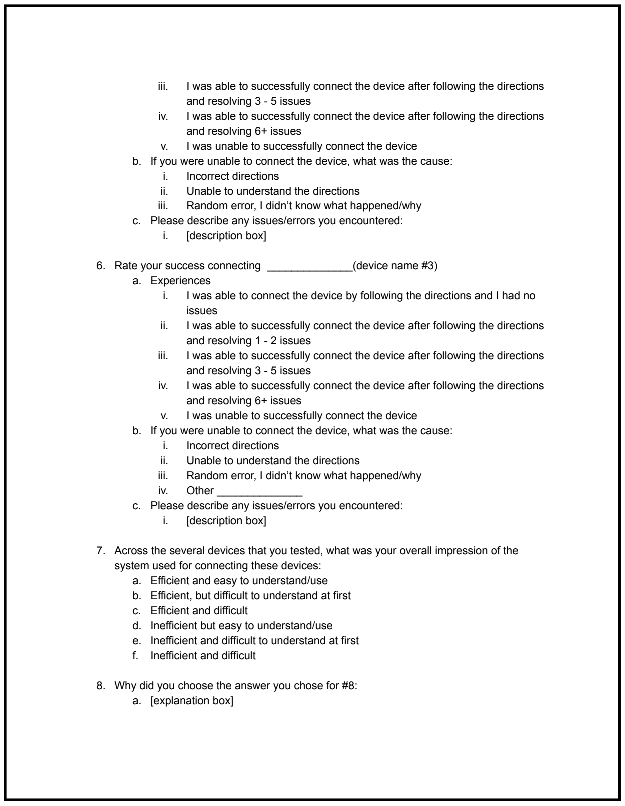
1. How comfortable with technology do you feel on a scale of 1-10
    a. 1 = not comfortable at all
    b. 5 = somewhat comfortable once I get used to it
    c. 10 = very comfortable, use without thinking

2. Rate your skill level with technology on a scale of 1-10
    a. 1 = I have no technology skills
    b. 5 = I have skills to use certain forms of technology
    c. 10 = I am extremely good at using all technology

3. Which of the following types of technology do you feel comfortable with?
    a. Cell phones (not smartphones)
    b. Cell phones (smartphones)
    c. Desktop computers
    d. Laptop computers
    e. Scanners, printers, fax machines
    f. Smart devices: Amazon alexa, smart lightbulb, internet connected printers etc.
    g. Other: ___________________

4. Rate your success connecting _______________(device name #1)
    a. Experiences
        i. I was able to connect the device by following the directions and I had no issues
        ii. I was able to successfully connect the device after following the directions and resolving 1 - 2 issues
        iii. I was able to successfully connect the device after following the directions and resolving 3 - 5 issues
        iv. I was able to successfully connect the device after following the directions and resolving 6+ issues
        v. I was unable to successfully connect the device
    b. If you were unable to connect the device, what was the cause:
        i. Incorrect directions
        ii. Unable to understand the directions
        iii. Random error, I didn't know what happened/why
    c. Please describe any issues/errors you encountered:
        i. [description box]

5. Rate your success connecting _______________(device name #2)
    a. Experiences
        i. I was able to connect the device by following the directions and I had no issues
        ii. I was able to successfully connect the device after following the directions and resolving 1 - 2 issues

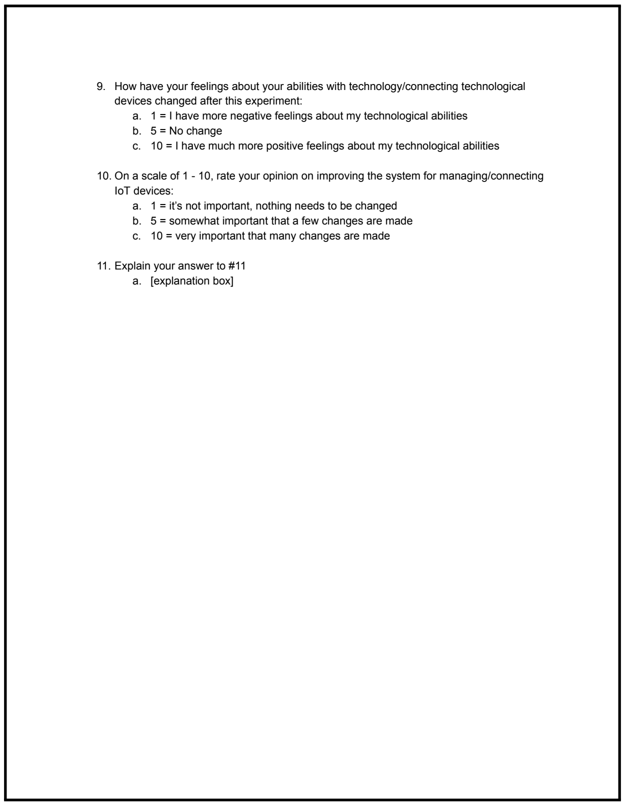


        iii. I was able to successfully connect the device after following the directions and resolving 3 - 5 issues
        iv. I was able to successfully connect the device after following the directions and resolving 6+ issues
        v. I was unable to successfully connect the device
    b. If you were unable to connect the device, what was the cause:
        i. Incorrect directions
        ii. Unable to understand the directions
        iii. Random error, I didn't know what happened/why
    c. Please describe any issues/errors you encountered:
        i. [description box]

6. Rate your success connecting _______________(device name #3)
    a. Experiences
        i. I was able to connect the device by following the directions and I had no issues
        ii. I was able to successfully connect the device after following the directions and resolving 1 - 2 issues
        iii. I was able to successfully connect the device after following the directions and resolving 3 - 5 issues
        iv. I was able to successfully connect the device after following the directions and resolving 6+ issues
        v. I was unable to successfully connect the device
    b. If you were unable to connect the device, what was the cause:
        i. Incorrect directions
        ii. Unable to understand the directions
        iii. Random error, I didn't know what happened/why
        iv. Other _______________
    c. Please describe any issues/errors you encountered:
        i. [description box]

7. Across the several devices that you tested, what was your overall impression of the system used for connecting these devices:
    a. Efficient and easy to understand/use
    b. Efficient, but difficult to understand at first
    c. Efficient and difficult
    d. Inefficient but easy to understand/use
    e. Inefficient and difficult to understand at first
    f. Inefficient and difficult

8. Why did you choose the answer you chose for #8:
    a. [explanation box]

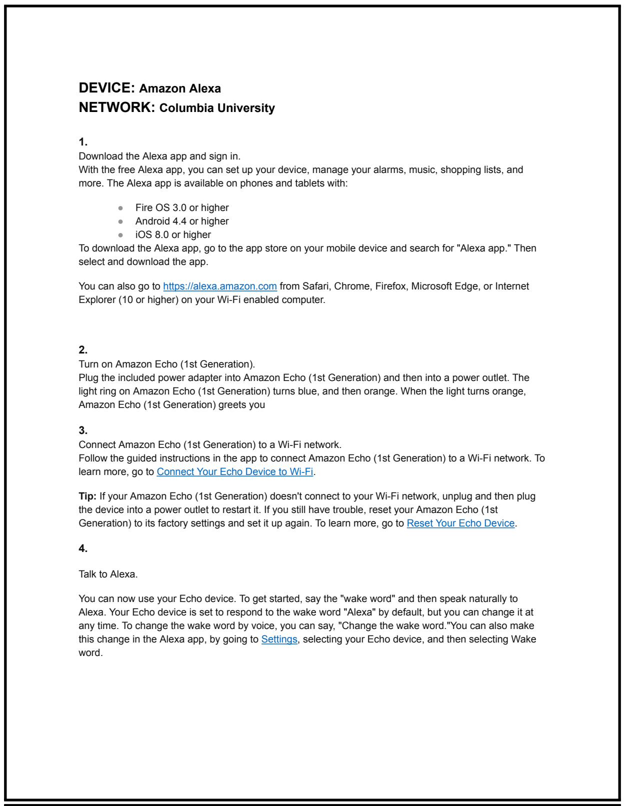


9. How have your feelings about your abilities with technology/connecting technological devices changed after this experiment:
    a. 1 = I have more negative feelings about my technological abilities
    b. 5 = No change
    c. 10 = I have much more positive feelings about my technological abilities

10. On a scale of 1 - 10, rate your opinion on improving the system for managing/connecting IoT devices:
    a. 1 = it's not important, nothing needs to be changed
    b. 5 = somewhat important that a few changes are made
    c. 10 = very important that many changes are made

11. Explain your answer to #11
    a. [explanation box]

Figure A.3: Post-experiment survey form



DEVICE: Amazon Alexa
NETWORK: Columbia University

1.
Download the Alexa app and sign in.
With the free Alexa app, you can set up your device, manage your alarms, music, shopping lists, and more. The Alexa app is available on phones and tablets with:

- Fire OS 3.0 or higher
- Android 4.4 or higher
- iOS 8.0 or higher

To download the Alexa app, go to the app store on your mobile device and search for "Alexa app." Then select and download the app.

You can also go to https://alexa.amazon.com from Safari, Chrome, Firefox, Microsoft Edge, or Internet Explorer (10 or higher) on your Wi-Fi enabled computer.

2.
Turn on Amazon Echo (1st Generation).
Plug the included power adapter into Amazon Echo (1st Generation) and then into a power outlet. The light ring on Amazon Echo (1st Generation) turns blue, and then orange. When the light turns orange, Amazon Echo (1st Generation) greets you

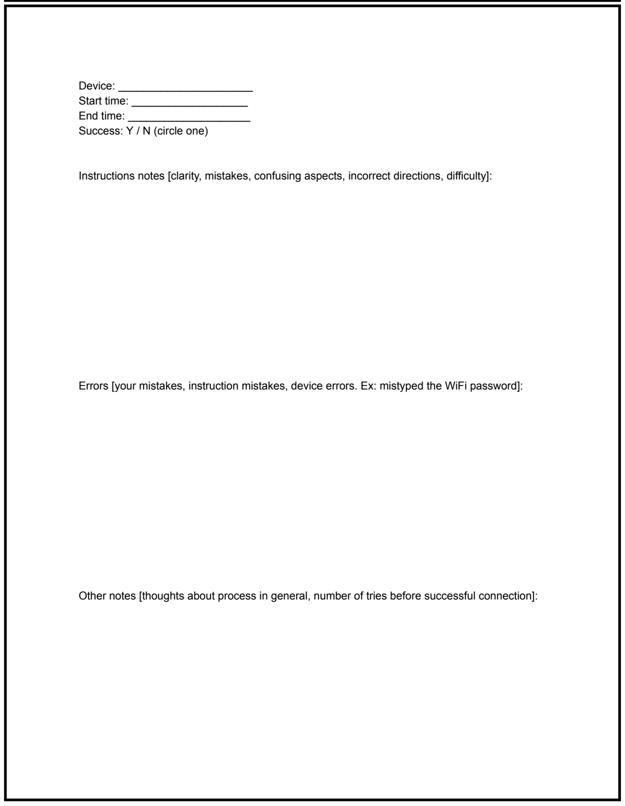
3.
Connect Amazon Echo (1st Generation) to a Wi-Fi network.
Follow the guided instructions in the app to connect Amazon Echo (1st Generation) to a Wi-Fi network. To learn more, go to Connect Your Echo Device to Wi-Fi.

Tip: If your Amazon Echo (1st Generation) doesn't connect to your Wi-Fi network, unplug and then plug the device into a power outlet to restart it. If you still have trouble, reset your Amazon Echo (1st Generation) to its factory settings and set it up again. To learn more, go to Reset Your Echo Device.

4.

Talk to Alexa.

You can now use your Echo device. To get started, say the "wake word" and then speak naturally to Alexa. Your Echo device is set to respond to the wake word "Alexa" by default, but you can change it at any time. To change the wake word by voice, you can say, "Change the wake word."You can also make this change in the Alexa app, by going to Settings, selecting your Echo device, and then selecting Wake word.



Device: ______________________
Start time: ___________________
End time: ____________________
Success: Y / N (circle one)

Instructions notes [clarity, mistakes, confusing aspects, incorrect directions, difficulty]:

Errors [your mistakes, instruction mistakes, device errors. Ex: mistyped the WiFi password]:

Other notes [thoughts about process in general, number of tries before successful connection]:

Figure A.4: Sample device instructions and experiment data record sheet