A Survey of Programmable Networks

Andrew T. Campbell[1], Herman G. De Meer[2], Michael E. Kounavis[1], Kazuho Miki[3], John B. Vicente[4], and Daniel Villela[1]

[1]Center for Telecommunications Research,
Columbia University
[2] University of Hamburg, Germany
[3] Hitachi Limited
[4] Intel Corporation

genesis@comet.columbia.edu

Abstract

In this paper we present a programmable networking model that provides a common framework for understanding the state-of-the-art in programmable networks. A number of projects are reviewed and discussed against a set of programmable network characteristics. We believe that a number of important innovations are creating a paradigm shift in networking leading to higher levels of network programmability. These innovations include the separation between transmission hardware and control software, availability of open programmable network interfaces, accelerated virtualization of networking infrastructure, rapid creation and deployment of new network services and environments for resource partitioning and coexistence of multiple distinct network architectures. We present a simple qualitative comparison of the surveyed work and make a number of observations about the direction of the field.

1. INTRODUCTION

The ability to rapidly create, deploy and manage novel services in response to user demands is a key factor driving the programmable networking research community. Results from this field of research are likely to have a broad impact on customers, service providers and equipment vendors across a range of telecommunication sectors, including broadband, mobile and IP networking. Competition between existing and future Internet Service Providers (ISPs) could solely hinge on the speed at which one service provider can respond to new market demands over another. The introduction of new services is a challenging task and calls for major advances in methodologies and toolkits for service creation and enabling network technologies. A vast amount of service-specific computation, processing and switching must be handled and new network programming environments have to be engineered to enable future networking infrastructures to be open, extensible and programmable.

Before we can meet this challenge, we need to better understand the limitations of existing networks and the fundamentals for making networks more programmable. There is growing consensus that these network fundamentals are strongly associated with the deployment of new network programming environments, possibly based on "network-wide operating system support", that explicitly recognize service creation, deployment and management in the network infrastructure. For example a future programmable network operating system could be based on active network execution environments [42] operating on node operating system [40] or open signaling network kernels [30] supporting the coexistence of multiple control architectures [33]. Both of these proposals squarely address the same problem: how to open the network up and accelerate its programmability in a controlled and secure

manner for the deployment of new architectures, services and protocols.

The separation of communications hardware (i.e., switching fabrics, routing engines) from control software is fundamental to making the network more programmable. Such a separation is difficult to realize today. The reason for this is that switches and routers are vertically integrated - akin to mainframes of the 70s. Typically, service providers do not have access to switch/router control environments (e.g. Cisco's IOS operating system), algorithms (e.g. routing protocols) or states (e.g., routing tables, flow states). This makes the deployment of new network services, which may be many orders of magnitude more flexible than proprietary control systems, impossible due to the closed nature of network nodes. The question is, how do we go about 'opening up the boxes' for deployment of third party control software and services?

This paper examines the state of the art in programmable networks. In Section 2, we present and discuss two schools of thought on programmable networks advocated by the *Active Networks (AN)* [20] and *Open Signalling (Opensig)* [39] communities. The state-of-the-art in programmable networks is rather complex to analyze beyond historical differences. Recently, a number of programmable network toolkits have been implemented. By reviewing each contribution in turn, we arrive at a common set of features that govern the construction of these programmable networks. In Section 3, we present a generalized model and common set of characteristics to better understand the contributions found in the literature. Following this, in Section 4, we discuss a number of specific projects and characterize them in terms of a simple set of characteristics. In Section 5, we present a simple qualitative comparison of the surveyed work and make a number of observations about the direction of the field. We believe that a number of important innovations are creating a paradigm shift in networking leading to higher levels of network programmability. This leads us to the conclusion that the ultimate challenge facing the programmable networking community is the development of programmable virtual networking environments.

2. METHODOLOGIES

There has been an increasing demand to add new services to networks or to customize existing network services to match new application needs. Recent examples of this include the introduction of integrated and differentiated services to IP networks offering enhanced IP QOS. The introduction of new services into existing networks is usually a manual, time consuming and costly process. The goal of programmable networking is to simplify the deployment of new network services, leading to networks that explicitly support the process of service creation and deployment. There is general consensus that programmable network architectures can be customized, utilizing clearly defined open programmable interfaces (i.e., network APIs) and a range of service composition methodologies and toolkits.

Two schools of thought have emerged on how to make networks programmable. The first school is spearheaded by the Opensig community, which was established through a series of international workshops. The other school, established by DARPA, constitutes a large number of diverse AN projects. The Opensig community argues that by modeling communication hardware using a set of open programmable network interfaces, open access to switches and routers can be provided, thereby enabling third party software providers to enter the market for telecommunications software. The Opensig community argues that by "opening up" the switches in this manner, the development of new and distinct architectures and services (e.g., virtual networking [34]) can be realized. Open signaling as the name suggests takes a telecommunications approach to the problem of making the network programmable. Here, there is a clear distinction between transport, control and management that underpin programmable networks and an emphasis on service creation with QOS. Recently, the IEEE Project 1520 [9] on Programmable Interfaces for Networks is pursuing the Opensig approach in an attempt to standardize programming interfaces for ATM switches, IP routers and mobile telecommunications networks. Physical network devices are abstracted as distributed computing objects (e.g. virtual switches [15], switchlets [33], and virtual base stations [6]) with well-defined open programmable interfaces. These open interfaces

allow service providers to manipulate the states of the network using middleware toolkits (e.g., CORBA) in order to construct and manage new network services.

The AN community advocates the dynamic deployment of new services at runtime mainly within the confines of existing IP networks. The level of dynamic runtime support for new services goes far beyond that proposed by the Opensig community, especially when one considers the dispatch, execution and forwarding of packets based on the notion of "active packets". In one extreme case of active networking, "capsules" [42] comprise executable programs, consisting of code (for example Java code) and data. In active networks, code mobility represents the main vehicle for program delivery, control and service construction. The granularity of control can range from the packet and flow levels through the installation of completely new switchware [3]. The term 'granularity of control' [12] refers to the scope of switch/router behavior that can be modified by a received packet. At one extreme, a single packet could boot a complete software environment seen by all packets arriving at the node. At the other extreme, a single packet (e.g., a capsule) can modify the behavior seen only by that packet. Active networks allow the customization of network services at packet transport granularity, rather than through a programmable control plane. Active networks offer maximum flexibility in support of service creation but with the cost of adding more complexity to the programming model. The AN approach is, however, an order of magnitude more dynamic than Opensig's quasi-static network programming interfaces.

Both communities share the common goal to go beyond existing approaches and technologies for construction, deployment and management of new services in telecommunication networks. Both movements include a broad spectrum of projects with diverse architectural approaches. For example, few AN projects consider every packet to be an active capsule and similarly few Opensig projects consider programmable network interfaces to be static. The Opensig approach, however, clearly separates network control from information transport and is primarily focused on programmable switches that provide some level of QOS support. In contrast, projects under the AN umbrella have historically been focused on IP networks, where the control and data paths are combined.

3. PROGRAMMABLE NETWORKING MODEL

3.1 Communications and Computation

A programmable network is distinguished from any other networking environment by the fact that it can be programmed from a minimal set of APIs from which one can ideally compose an infinite spectrum of higher level services. We present a generalized model for programmable networks as a three-dimensional model illustrated in Figure 1. This model shows the Internet reference model (viz. application, transport, network, link layers) augmented with transport[1], control and management planes. The division between transport, control and management allows our model to be generally applicable to telecommunications and Internet technologies. The notion of the separation between transport, control and management is evident in architectures. In the case of Internet there is a single data path but clearly one can visualize transport (e.g., video packets), control (e.g., RSVP) and management (e.g., SMNP) mechanisms. In the case of telecommunication networks there is typically support in the architecture for transport, control and management functions. This division is motivated by the different ways these networking functions utilize the underlying hardware and by the distinct time scales over which they operate. In both cases, the planes of our generalized model remain neutral supporting the design space of different networking technologies.

The programmability of network services is achieved by introducing computation inside the network, beyond the extent of the computation performed in existing routers and switches. To distinguish the notion of a "programmable network architecture" from a "network architecture", we have extended the communication model and augmented it with a computation model, explicitly acknowledging the programmability of network architectures. We can view the generalized model for programmable networks as comprising

[1] In this case planes collectively represent cross-layer services and protocols.

conventional communication, encompassing the transport, control and management planes, and computation as well, as illustrated in Figure 1. Collectively, the computation and communication models make up a programmable network. The computation model provides programmable support across the transport, control and management planes, allowing a network architect to program individual layers (viz. application, transport, network and link layers) in these planes. Another view is that programmable support is delivered to the transport, control and management planes through the computation model.

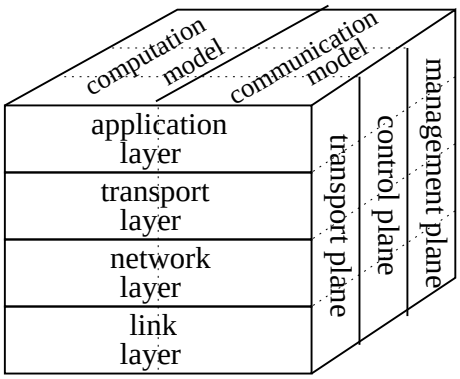

Figure 1: Computation and Communication Models

In Figure 2, an alternative view of the generalized model is shown. The key components of the computation model are represented as a distributed network programming environment and a set of "node kernels"[2]. Node kernels are node operating systems realizing resource management. Node kernels have local significance only, that is, they manage single node resources, potentially shared by multiple programmable network architectures. The network programming environment provides middleware support to distributed network programming services. Figure 2 illustrates the separation of switching hardware from programming and communication software. Two sets of interfaces are exposed. The first set of interfaces represents the network programming interfaces between network programming environments and programmable network architectures. The lower set of interfaces represents the node interfaces between node kernels and network programming environments. We believe that there needs to be some agreement or standardization of these interfaces to allow platform-independent network programming. This is likely to happen through a number of forums, e.g., IEEE Programmable Interfaces for Networks [9], DARPA Active Networks [20], Multiservice Switching Forum [38], OPENSIG [39] and IETF (e.g., the new work item on GSMP) or by the emerging new programmable network industries [48] [19].

Research on programmable networks is focused on all facets of this model. Different programming methodologies, levels of programmability, and communication technologies have been investigated. Some projects, especially from the Opensig community have placed more emphasis on API definitions. Others focus on issues related to code mobility or contribute to the application domain. Dynamic "plug-ins" have been investigated for the construction or potential extension of new protocols or applications. In what follows, we provide a more detailed overview of the components in our generalized model. It is our belief that independent contributions to the field are beginning to converge and it is our intention to indicate this convergence by way of survey.

3.2 Node Kernel

Many node vendors incorporate operating system support into their switches and routers to handle communication functions of network nodes, e.g. CISCO routers use the IOS environment and ATML ATM switches use the ATMOS micro-kernel. Typically, these node operating systems support a variety of communications activities, e.g., signaling, control and management processes, inter-process communication, forwarding functions, and downloading of new boot images. Currently, these node operating systems are closed to third party providers because of their proprietary nature, and they are limited in their support for evolving network programming environments. While the idea of introducing computation power into nodes is not new, there is a greater need for computation elements to abstract node functionality and allow it to be open and programmable. The computation

[2] We borrow the term node kernel from the work on NodeOS [40] and broadband kernels [30] by the active networking and Opensig communities, respectively.

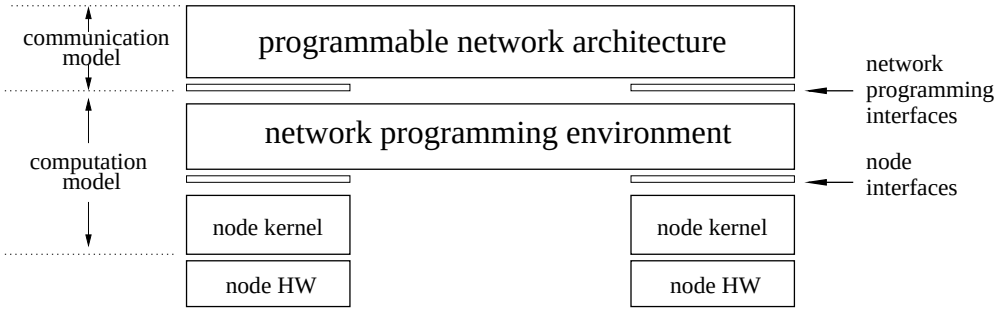

Figure 2: Generalized Model for Programmable Networks

model, introduced in the previous section, enables the programmability of the communication model and requires low-level programmable support for communication abstractions (e.g., packets, flows, tunnels, virtual paths), dynamic resource partitioning and security considerations.

We describe this low-level programming environment that runs on switch/routers as the node kernel. The node kernel represents the lowest level of programmability, providing a small set of node interfaces. These interfaces support the manipulation of the node state (e.g., accessing and controlling the node resources) and the invocation of communication services (e.g. communication abstractions and security). The node kernel is responsible for sharing node computational (e.g., sharing the CPU) and communication resources, (e.g., partitioning the capacity of a multiplexer), as well supporting core security services. A node kernel may operate on any type of network node, end-system or device, for example, IP router, ATM switch, or base station. It may also provide access to dedicated hardware offering fast packet processing services to network programming environments. A node kernel has local significance only, providing the network programming environment with a set of low-level programming interfaces, that are used by network architects to program network architectures in a systematic manner.

3.3 Network Programming Environment

Network programming environments support the construction of networks, enabling the dynamic deployment of network services and protocols. Network programming environments support different levels of programmability, programming methodologies, networking technologies and application domains. Network programming environments operate over a set of well-defined node kernel interfaces offering distributed toolkits for the realization of programmable network architectures through the deployment of distributed service components. In this sense, one can view network-programming environments as the "middleware glue" between executing network architectures and the node kernels themselves, as illustrated in Figure 2. Network programming environments provide network architect/designers with the necessary environment and tools for building distinct network architectures that run in a distributed fashion on multiple node kernels. In this sense network programming environments support the programming of network architectures in the same way that software development kits (SDKs) allow developers to build new applications that run on native operating systems.

This "middleware glue" can be constructed from scratch or be built on top of well-defined distributed object computing environments. For example, the xbind [15] and mobiware [6] toolkits address programmability of broadband and mobile networks, respectively, and are built using COBRA middleware technology. Other approaches use mobile code technology and virtual machines to dynamically program the network. For example, the Active Network Transport System (ANTS) incorporates capsule technology [45], leveraging the Java Virtual Machine for new protocol deployment. Both approaches result in toolkits that execute on node kernels offering a high level of programmability for service creation and deployment of distinct network architectures.

Network programming environments offer a set of open interfaces and services to network designers/architects to program distinct network

architectures. Network programming environments support the construction of network architectures through service composition, service control, and resource and state management. Services offered by network programming environments can range from simple Remote Procedure Calling (RPC) between distributed network objects to sophisticated dynamic loading of mobile code and fast compilation of intermediate machine-independent representation. Different types of network programming environments offer different levels of programmability to network architectures. For example, mobile code technologies offer the most radical solution to the development of services in programmable networks when compared to RPC-based object middleware. We identify the 'level of programmability' as an important characteristic of programmable networks.

3.4 Programmable Network Architecture

The goal of network programming environments is to provide the necessary support to dynamically program new network architectures. Network programming environments do not offer core network algorithms (e.g., routing, signaling) that define and differentiate network architecture in the same way that operating systems do not embed application specific algorithms in the kernel. Rather, a network programming environment offers a set of network programming interfaces for constructing network architectures. Philosophically this is similar to constructing new applications using software development kits. However in this case the application is the network architecture.

We broadly define network architecture as having the following attributes[3]:

- *network services*, which the network architecture realizes as a set of distributed network algorithms and offers to the end systems;
- *network algorithms*, which includes transport, signaling/control and management mechanisms;
- *multiple time scales*, which impact and influence the design of the network algorithms; and
- *network state management*, which includes the state that the network algorithms operate on (e.g., switching, routing, QOS state) to support consistent services.

Network programming environments offer creation and deployment tools and mechanisms that allow network architects to program and build new network architectures. Programmable network architectures are realized through the deployment of a set of network algorithms that take into account network state and reflect the time scales over which these algorithms operate. Network algorithms are potentially as diverse as the application base that exists in the end-systems today. Programmable network architectures may range from simple best-effort forwarding architectures to complex mobile protocols that respond dynamically to changes in wireless QOS and connectivity. Given this diversity, it is necessary that both network programming environments and node kernels are extensible and programmable to support a large variety of programmable network architectures.

4. PROGRAMMABLE NETWORKS

Following on from the discussion of the generalized model for programmable networks, we now survey a number of programmable networking projects that have emerged in the literature. We attempt to identify essential contributions of the various projects to the field in terms of a set of characteristics. The survey is not intended to represent an exhaustive review of the field[4]. Rather, we discuss a set of projects that are representative of each programmable network characteristic introduced, focusing on the pertinent and novel features of each project and then, in Section 5, we compare them to the generalized model introduced in the preceding section.

4.1 Characteristics

A number of research groups are actively designing and developing programmable network prototypes. Each group tends to use its own terminology.

[3] This is of course an over simplification of a complex system. Our goal here is to be illustrative in support of the generalized model and not definitive regarding a definition of network architecture.

[4] For a survey on active networks see [43].

However, on examination one can observe a common set of characteristics that govern the construction of these programmable networks. We use these characteristics to better understand the field:

- *networking technology*, which implicitly limits the programmability that can be delivered to higher levels. For example, some technologies are more QOS programmable (e.g., ATM), scalable (e.g., Internet) or limited in bandwidth availability (e.g., mobile networks);
- *level of programmability*, which indicates the method, granularity and time scale over which new services can be introduced into the network infrastructure. This in turn is strongly related to language support, programming methodology or middleware adopted. For example, distributed object technology can be based on RPC [46] or mobile code [45] methodologies resulting in quasi-static or dynamically composed network programming interfaces;
- *programmable communications abstractions*, which indicate the level of virtualization and programmability of networking infrastructure requiring different middleware and potentially network node support (e.g., switch/router, base station). For example, programmable communications abstractions include virtual switches [30], switchlets [33], active nodes [40], universal mobile channels [32] and virtual active networks [21]; and
- *architectural domain*, which indicates the targeted architectural or application domain (e.g., signaling, management, transport). This potentially dictates certain design choices and impacts the construction of architectures, and services offered, calling for a wide range of middleware support. Examples include, composing application services [4], programmable QOS control [30] and network management [41]).

4.2 Networking Technology

A number of programmable network prototypes have been targeted to specific networking technologies. The motivation behind these projects is to make the targeted networking technology more programmable in an attempt to overcome particular deficiencies associated with supporting communication services.

4.2.1 IP networks: Smart Packets

The University of Kansas has developed smart packets, a code-based specialized packet concept implemented in a programmable IP environment [29]. Smart packets represent elements of in-band or out-of-band mobile code based on Java classes. Smart packets propagate state information in the form of serialized objects and carry identifiers for authentication purposes. An active node architecture supports smart packets by exposing a set of resource abstractions and primitives made accessible to smart packets. Active nodes incorporate:

- resource controllers, which provide interfaces to node resources;
- node managers, which impose static limits on resource usage; and
- state managers, which control the amount of information smart packets may leave behind at an active node.

The active node supports a feedback-scheduling algorithm to allow partitioning of CPU cycles among competing tasks and a credit-based flow-control mechanism to regulate bandwidth usage. Each smart packet is allocated a single thread of CPU and some amount of node resources. Active nodes also include router managers that support both default routing schemes and alternative routing methods carried by smart packets. The smart packets testbed has been used to program enhanced HTTP and SMTP services that show some performance benefits over conventional HTTP and SMTP by reducing excessive ACK/NAK responses in the protocols. A beacon routing scheme supports the use of multiple routing algorithms within a common physical IP network based on smart packets.

4.2.2 ATM Networks: xbind

ATM technology provides connection-oriented communications and has been tailored towards QOS provisioning of multimedia networks. Although essential features of QOS provisioning, such as admission control and resource reservation, are inherently supported by the ATM technology, its

signaling component is unsuitable for practical usage due to its significant complexity. xbind [15] overcomes these service creation limitations by separating control algorithms from the telecommunications hardware. Emphasis is placed on the development of interfaces to provide open access to node resources and functions, using virtual switch and virtual link abstractions. The interfaces are designed to support the programmability of the management and control planes in ATM networks.

The xbind broadband kernel [47], which is based on the XRM model [15], incorporates three network models abstracting a broadband network, multimedia network and service network. The multimedia network supports programmable network management, network control, state management, connection management and media stream control. The xbind testbed incorporates multivendor ATM switches using open signaling and service creation to support a variety of broadband services, transport and signaling systems with QOS guarantees.

4.2.3 Mobile Networks: Mobiware

Mobiware [6] is a software-intensive open programmable mobile architecture extending the xbind model of programmability to packet based mobile networks for the delivery of adaptive mobile services over time-varying wireless links. Mobiware incorporates object-based, CORBA programmability for the control plane but also allows active transport objects (i.e., code plug-ins) based on Java byte code to be loaded into the data path. At the transport layer, an active transport environment injects algorithms into base stations providing value-added service support at strategic points inside the network. At the network layer, a set of distributed objects that run on mobile devices, access points and mobile-capable switches, interact with each other to support programmable handoff control and different styles of QOS adaptation. The MAC layer has also been made programmable.

The following mobile services have been programmed using the mobiware toolkit [37]:

- QOS-controlled handoff, which supports automatic media scaling and error control based on an adaptive-QOS API and wireless channel conditions;
- mobile soft-state, which provides mobile devices with the capability to respond to time varying QOS through a periodic reservation and renegotiation process; and
- flow bundling, which supports fast handoff in cellular access networks.

The mobiware testbed supports a variety of scalable audio and video services to mobile devices in addition to traditional web based data services.

4.3 Level of Programmability

The level of programmability expresses the granularity at which new services can be introduced into the network infrastructure. One can consider a spectrum of possible choices from highly dynamic to more conservative levels of programmability. At one end of this spectrum, capsules [42] carry code and data enabling the uncoordinated deployment of protocols. Capsules represent the most dynamic means of code and service deployment into the network. At the other end of the spectrum there are more conservative approaches to network programmability based on quasi-static network programming interfaces using RPCs between distributed controllers [46] to deploy new services. Between the two extremes lie a number of other methodologies combining dynamic plug-ins, active messaging and RPC. Different approaches have a direct bearing on the speed, flexibility, safety, security and performance at which new services can be introduced into the infrastructure.

4.3.1 Capsules: ANTS

ANTS [45], developed at MIT, enables the uncoordinated deployment of multiple communication protocols in active networks providing a set of core services including support for the transportation of mobile code, loading of code on demand and caching techniques. These core services allow network architects to introduce or extend existing network protocols. ANTS provides a network programming environment for building new capsule-based programmable network architectures. Examples of such programmed network services include enhanced multicast services, mobile IP routing and application-level filtering. The ANTS capsule-driven execution model provides a foundation for maximum network programmability

in comparison to other API approaches. Capsules serve as atomic units of network programmability supporting processing and forwarding interfaces. Incorporated features include node access, capsule manipulation, control operations and soft-state storage services on IP routers. Active nodes execute capsules and forwarding routines, maintain local state and support code distribution services for automating the deployment of new services. The ANTS toolkit also supports capsule processing quanta as a metric for node resource management.

4.3.2 Active Extensions: Switchware

Switchware [3], being developed at University of Pennsylvania, attempts to balance the flexibility of a programmable network against the safety and security requirements needed in a shared infrastructure such as the Internet. The Switchware toolkit allows the network architects to trade-off flexibility, safety, security, performance and usability when programming secure network architectures. At the operating system level, an active IP-router component is responsible for providing a secure foundation that guarantees system integrity. Active extensions can be dynamically loaded into secure active routers through a set of security mechanisms that include encryption, authentication and program verification. The correct behavior of active extensions can be verified off-line by applying 'heavyweight' methods, since the deployment of such extensions is done over slow time scales.

Active extensions provide interfaces for more dynamic network programming using active packets. Active packets can roam and customize the network in a similar way as capsules do. Active packets are written in functional languages (e.g., Caml and PLAN [28]) and carry lightweight programs that invoke node-resident service routines supported by active extensions. There is much less requirement for testing and verification in the case of active packets than for active extensions, given the confidence that lower level security checks have already been applied to active extensions. Active packets cannot explicitly leave state behind at nodes and they can access state only through clearly defined interfaces furnished by active extension software. Switchware applies heavyweight security checks on active extensions, which may represent major releases of switch code, and more lightweight security checks on active packets. This approach allows the network architect to balance security concerns against performance requirements. The security model of Switchware considers public, authenticated and verified facilities.

4.3.3 Composition Languages: CANEs

Capsules, active messages and active extensions promote the creation of new services through the composition of new building blocks or by adding components to existing services. The CANEs project led by researchers at University of Kentucky and Georgia Tech. aim to define and apply service composition rules as a general model for network programmability [14]. A composition method is used to construct composite network services from components. A composition method is specified as a programming language with enhanced language capabilities that operates on components to construct programmable network services. Attributes of a good composition method include high performance, scalability, security and ease of management. Features of well-structured composition methods combine:

- control on the sequence in which components are executed;
- control on shared data among components;
- binding times, which comprise composite creation and execution times;
- invocation methods, which are defined as events that cause a composite to be executed; and
- division of functionality among multiple components, which may either reside at an active node or be carried by packets.

PLAN, ANTS and Netscript [21] (described in Section 4.4.2) are examples of composition methods. LIANE is proposed within the CANEs project as a composition method that incorporates all the aforementioned features. The key idea of LIANE is that services are composed from basic underlying programs that contain processing slots. Users insert programs for customization in these slots. The CANEs definition of service composition encompasses the Opensig approach to network programmability indicating how different

approaches to programmable networking complement each other by addressing the same goal from different perspectives.

4.3.4 Network APIs: xbind

The xbind broadband kernel is based on a binding architecture and a collection of node interfaces referred to as Binding Interface Base (BIB) [2]. The BIB provides abstractions to the node state and network resources. Binding algorithms run on top of the BIB and bind QOS requirements to network resources via abstractions. The BIB is designed to support service creation through high-level programming languages. The interfaces are static while supporting universal programmability. The quasi-static nature of the BIB interfaces, allow for complete testing and verification of the correctness of new functions, on emulation platforms, before any service is deployed. The concept of active packets or capsules containing both programs and user data is not considered in the xbind approach to programmability. Rather, communication is performed using RPCs between distributed objects and controllers based on OMG's CORBA. The approach taken by xbind promotes interoperability between multi-vendor switch market supporting resource sharing and partitioning in a controlled manner.

4.4 Programmable Communications Abstractions

Abstractions and partitioning of resources are essential concepts in programmable networking. Programmable communications abstractions may range from node resources to complete programmable virtual networks. Other programmable communications abstractions include programmable virtual routers, virtual links and mobile channels. Abstracting the network infrastructure through virtualization and making it programmable is a major contribution of the field that encompasses a number of different projects.

4.4.1 Active Node Abstractions: NodeOS

Members of the DARPA active network program [20] are developing an architectural framework for active networking [11]. A node operating system called NodeOS [40] represents the lowest level of the framework. NodeOS provides node kernel interfaces at routers utilized by multiple execution environments, which support communication abstractions such as threads, channels and flows. Development of an execution environment is a nontrivial task and it is anticipated [12] that the total number of execution environments will not be large. Encapsulation techniques based on an active network encapsulation protocol (ANEP) [5] support the deployment of multiple execution environments within a single active node. ANEP defines an encapsulation format allowing packets to be routed through multiple execution environments coexisting on the same physical nodes. Portability of execution environments across different types of physical nodes is accomplished by the NodeOS, by exposing a common, standard interface. This interface defines four programmable node abstractions: threads, memory, channels and flows. Threads, memory and channels abstract computation, storage, and communication capacity used by execution environments, whereas flows abstract user data-paths with security, authentication and admission control facilities. An execution environment uses the NodeOS interface to create threads and associate channels with flows. The NodeOS supports QOS using scheduling mechanisms that regulate the access to node computation and communication resources. The architectural framework for active networking is being implemented in the ABONE testbed [1] allowing researchers to prototype new active architectures.

4.4.2 Virtual Active Networks: Netscript

The Netscript project [49] at Columbia University takes a functional language-based approach to capture network programmability using universal language abstractions. Netscript is a strongly typed language that creates universal abstractions for programming network node functions. Unlike other active network projects that take a language-based approach Netscript is being developed to support Virtual Active Networks as a programmable abstraction. Virtual Active Network [21] abstractions can be systematically composed, provisioned and managed. In addition, Netscript automates management through language extensions that generate MIBs. Netscript leverages earlier work on decentralized management and agent technologies that automatically correlate and

analyze the behavior monitored by active MIB elements. A distinguishing feature of Netscript is that it seeks to provide a universal language for active networks in a manner that is analogous to postscript. Just as postscript captures the programmability of printer engines, Netscript captures the programmability of network node functions. Netscript communication abstractions include collections of nodes and virtual links that constitute virtual active networks.

4.4.3 Virtual ATM Networks: Tempest

The Tempest project at the University of Cambridge [34] has investigated the deployment of multiple coexisting control architectures in broadband ATM environments. Novel technological approaches include the usage of software mobile agents to customize network control and the consideration of control architectures dedicated to a single service. Tempest supports two levels of programmability and abstraction. First, switchlets, which are logical network elements that result from the partition of ATM switch resources, allow the introduction of alternative control architectures into an operational network. Second, services can be refined by dynamically loading programs that customize existing control architectures. Resources in an ATM network can be divided by using two software components: a switch control interface called ariel and a resource divider called prospero. Prospero communicates with an ariel server on an ATM switch, partitions the resources and exports a separate control interface for each switchlet created. A network builder creates, modifies and maintains control architectures.

4.5 Architectural Domains

Most programmable network projects are related to the introduction of services into networks. However, most projects are targeted to a particular architectural domain (e.g., QOS control, signaling, management, transport and applications). In what follows we discuss three projects that address the application, resource management and network management domains.

4.5.1 Application-Level: Active Services

In contrast to the main body of research in active networking, Amir et al. [4] call for the preservation of all routing and forwarding semantics of the Internet architecture by restricting the computation model to the application layer. The Active Services version 1 (AS1) programmable service architecture enables clients to download and run service agents at strategic locations inside the network. Service agents called "servents" are restricted from manipulating routing tables and forwarding functions that would contravene the IP-layer integrity. The AS1 architecture contains a number of architectural components:

- a service environment, which defines a programming model and a set of interfaces available to servents;
- a service-location facility, which allows clients to 'rendezvous' with the AS1 environment by obtaining bootstrapping and configuration mechanisms to instantiate servents[5];
- a service management system, which allocates clusters of resources to servents using admission control and load balancing of servents under high-load conditions;
- a service control system, which provides dynamic client control of servents once instantiated within an AS1 architecture;
- a service attachment facility, which provides mechanisms for clients that can not interact directly with the AS1 environment through soft-state gateways; and
- a service composition mechanism, which allows clients to contact multiple service clusters and interconnect servents running within and across clusters.

The AS1 architecture is programmable at the application layer supporting a range of application domains. In [4], the MeGa architecture is programmed using AS1 to support an active media gateway service. In this case, servents provide

[5] Servents are launched into the network by an active service control protocol (ASCP), which includes an announce-listen protocol for servers to manage session state consistency, soft-state to manage expiration due to timeouts and multicast damping to avoid flooding the environment with excessive servents.

support for application-level rate control and transcoding techniques.

4.5.2 Resource Management: Darwin

The Darwin Project [17] at Carnegie Mellon University is developing a middleware environment for the next generation IP networks with the goal of offering Internet users a platform for value-added and customizable services. The Darwin project is focused toward customizable resource management that supports QOS. Architecturally, the Darwin framework includes Xena, a service broker that maps user requirements to a set of local resources, resource managers that communicate with Xena using the Beagle signaling protocol, and hierarchical scheduling disciplines based on service profiles. The Xena architecture takes the view that the IP forwarding and routing functions should be left in tact and only allows restricted use of active packet technology in the system.

Alongside the IP stack, Darwin introduces a control plane that builds on similar concepts such as those leveraged by broadband kernels [30] and active services [4]. The Xena architecture is made programmable and incorporates active technologies in a restricted fashion. A set of service delegates provides support for active packets. Delegates can be dynamically injected into IP routers or servers to support application specific processing (e.g., sophisticated semantic dropping) and value-added services (e.g., transcoders). A distinguishing feature of the Darwin architectural approach is that mechanisms can be customized according to user specific service needs defined by space, organization and time constraints. While these architectural mechanisms are most effective when they work in unison each mechanism can also be combined with traditional QOS architecture components. For example, the Beagle signaling system could be programmed to support RSVP signaling for resource reservation, while the Xena resource brokers and hierarchical schedulers could support traffic control.

4.5.3 Network Management: Smart Packets

The Smart Packets Project [41] (not to be confused with University of Kansas smart packets) at BBN aims to improve the performance of large and complex networks by leveraging active networking technology. Smart Packets are used to move management decision making points closer to the nodes being managed, target specific aspects of the node for management and abstract management concepts to language constructs. Management centers can send programs to managed nodes. Thus the management process can be tailored to the specific interests of the management center reducing the amount of back traffic and data requiring examination. A smart packet consists of a header and payload encapsulated using ANEP [5]. Smart packets may carry programs to be executed, results from execution, informational messages or reports on error conditions. Smart Packets are written in two programming languages:

- sprocket, which is a high-level C-like, language with security threatening constructs, and
- spanner, which is a low-level assembly-like language, that can result in tighter, optimized code.

Sprocket programs are compiled into spanner code, which in turn is assembled into a machine-independent binary encoding placed into smart packets. Meaningful programs perform networking functions and MIB information retrieval.

5. DISCUSSION

We have introduced a set of characteristics and a generalized model for programmable networks to help understand and differentiate the diverse set of programmable network projects discussed in this paper. In what follows we provide a brief comparison of these projects and other work in the field.

5.1 Comparison

In this section we present a simple qualitative comparison of the programmable networks surveyed in Section 4. Table 1 presents the comparison with respect to the characteristics and generalized model for programmable networks presented in Section 3 and 4, respectively.

5.2 Open Programmable Interfaces

The use of open programmable network interfaces is evident in many programmable network projects discussed in this survey. Open interfaces provide a

foundation for service programming and the introduction of new network architectures.

The xbind broadband kernel supports a comprehensive Binding Interface Base using CORBA/IDL to abstract network ATM devices, state and control. A number of other projects focussed on programming IP networks (e.g., ANTS, Switchware, CANEs) promote the use of open APIs that abstract node primitives, enabling network programmability and the composition of new services. Many network programming environments shown in Table 1 take fundamentally different approaches to providing open interfaces for service composition. The programming methodology adopted (e.g., distributed object technology based on RPC, mobile code or hybrid approaches) has a significant impact on an architecture's level of programmability; that is, the granularity, time scales and complexity incurred when introducing new APIs and algorithms into the network.

Two counter proposals include the xbind and ANTS APIs. While the ANTS approach to the deployment of new APIs in extremely flexible presenting a highly dynamic programming methodology it represents a complex programming model in comparison to the simple RPC model. In contrast, the xbind binding interfaces and programming paradigm is based on a set of CORBA IDL and RPC mechanisms. In comparison to capsule-based programmability the xbind approach is rather static in nature and the programming model less complex. These approaches represent two extremes of network programmability.

One could argue that quasi-static APIs based on RPC is a limited and restrictive approach. A counter argument is that the process of introducing and managing APIs is less complex than the capsule-based programming paradigm, representing a more manageable mechanism for service composition and service control. Similarly one could argue that active message and capsule-based technologies are more 'open' because of the inherent flexibility of their network programming models given that capsules can graft new APIs onto routers at runtime. The xbind approach lacks this dynamic nature at the cost of a simplified programming environment. Other projects adopt hybrid approaches. For example the mobiware toolkit combines the static APIs with the dynamic introduction of Java service plug-ins when needed [7]. A clear movement of the field is to open up the networks and present APIs for programming new architectures, services and protocols. As we discuss in the next section the field is arguing that the switches, routers and base stations should open up ultimately calling for open APIs everywhere.

5.3 Virtualization and Resource Partitioning

Many projects use virtualization techniques to support the programmability of different types of communication abstractions. The Tempest framework [33] presents a good example of the use of virtualization of the network infrastructure. Low-level physical switch interfaces are abstracted creating sets of interfaces to switch partitions called switchlets. Switchlets allow multiple control architectures to coexist and share the same physical switch resources (e.g., capacity, switching tables, name space, etc.). Typically, abstractions found in programmable networks are paired with safe resource partitioning strategies that enable multiple services, protocols and different programmable networking architectures to coexist. Virtualization of the network in this manner presents new levels of innovation in programmable networks that have not been considered before. All types of network components can be virtualized and made programmable from switches and links [15] to switchlets [33], active nodes [40], routelets [13] and virtual networks [21], [34], [13].

The NodeOS interface [40] provides a similar abstraction to node resources. The use of open interfaces allows multiple network programming environments (or execution environments using active networking terminology) to coexist within a common physical node architecture. In this case, the ANEP [5] protocol provides encapsulation as a mechanism for delivering packets to distinct execution environments.

Projects	Characteristics								Generalized Model for Programmable Networks															
									Node Kernels					Network Programming Environments				Programmable Network Architectures					
	architectural domain	networking technology	programmable communications abstractions	level of programmability	programming methodology				node interfaces and binding mechanisms	resource management support	support for multiple network architectures	support for multiple programming environments	security support	service composition	service control	resource management capability	security capability	application-level services	network management	network control	information transport	routing
					composition languages	distributed object technology	mobile code technology	encapsulation															
Active Services [4]	composing application level services	Internet	application services	dynamic	Tcl/oTcl		X		X					X	X	X		X				
Smart Packets, BBN [41]	network management	Internet	managed nodes	dynamic, discrete	Sprocket & Spanner		X	X	X					X	X		X		X			
NetScript [49]	composing network services and VANs	Internet	VANS	dynamic, discrete	NetScript		X	X	X		X			X	X				X			
ANTS [45]	composing network services	Internet	Internet protocols	dynamic, integrated	JAVA		X	X	X					X	X	X		X				X
CANEs [14]	composing services	Internet	composable services	dynamic	LIANE		X	X						X	X							
SwitchWare [3]	composing network services	Internet	Internet protocols	dynamic	PLAN & Caml		X	X	X				X	X	X		X					X
SmartPackets, U, Kansas [29]	composing network services	Internet	Internet protocols	dynamic	JAVA		X	X	X	X				X	X	X	X	X				X
Liquid Software [27]	investigating mobile code technology	Internet		dynamic	JAVA		X											X				
ANN [22]	composing network services	Internet	network node	dynamic, discrete	object code		X		X	X				X	X		X					X
NodeOS [40]	enabling network programmability	Internet	network node				X	X	X	X		X										
xbind [15]	enabling telecommunications service creation	ATM	multimedia networks	Static	CORBA/IDL	X			X	X				X	X			X		X		
DARWIN [17]	integrated resource management and value added services	Internet	flows	quasi-static		X	X		X	X				X	X			X	X	X		
Mobiware [6]	wireless QoS and mobile QoS control	Mobile	universal mobile channels	quasi-static	CORBA/IDL & JAVA	X	X		X	X				X	X			X		X	X	
Tempest [34]	enabling alternative control architectures	ATM	network control architectures	quasi-static	CORBA/IDL	X	X		X	X	X			X	X			X		X		
X-Bone [44]	automating the deployment of IP overlays	Internet	IP overlays					X			X					X	X					X
Supranet [23]	Virtual Network Services	Internet	Virtual Networks					X			X					X	X					X
Genesis [13]	Spawning Virtual Network Architectures	Internet	Spawning Networks	dynamic	Metabus/IDL	X	X		X	X	X	X		X	X	X						

Table 1: Comparison of Programmable Networks

Using encapsulation in this manner allows for different overlay execution environments (e.g., ANTS, Switchware, or Netscript) to execute on the same router using a single, common node kernel. The notion of virtualization is not a new concept, however. Similar motivation in the Internet community has led to the advent of the Mbone. New directions in the virtualization of the Internet have prompted the proposal for X-bone [44], shown in Table 1, which will provide a network programming environment capable of dynamically deploying overlay networks. As Table 1 illustrates, other projects such as Supranet [23] advocate tunneling and encapsulation techniques for the separation and privacy among coexisting, collaborative environments.

5.4 Programmable Virtual Networking

The dynamic composition and deployment of new services can be extended to include the composition of complete network architectures as virtual networks. The Netscript project [49] supports the notion of Virtual Active Networks [21] over IP networks. Virtual network engines interconnect sets of virtual nodes and virtual links to form virtual active networks. The Tempest framework [34] supports the notion of virtual networks using safe partitioning over ATM hardware. Tempest offers two levels of programmability. First, network control architectures can be introduced over long time scales through a 'heavyweight' deployment process. Second, 'lightweight' application-specific customization of established control architectures take place over faster time scales. The abstraction of physical switch partitions within the Tempest framework has led to the implementation of multiple coexisting control architectures. The Tempest strategy aims to address QOS through connection-oriented ATM technology and investigates physical resource sharing techniques between alternative control architectures. Both Darwin [17] and Netscript [49] projects support the notion of sharing the underlying physical infrastructure in a customized way as well. As discussed in the previous section, the NodeOS [40] project also provides facilities for coexisting execution environments.

5.5 Spawning Networks

In [13] we describe spawning networks, a new class of programmable networks that automate the creation, deployment and management of distinct network architectures "on-the-fly". The term "spawning" finds a parallel with an operating system spawning a child process, typically operating over the same hardware. We envision programmable networks as having the capability to spawn not processes but complex network architectures [31]. The enabling technology behind spawning is the Genesis Kernel [13], a virtual network operating system that represents a next-generation approach to the development of network programming environments.

A key capability of Genesis is its ability to support a virtual network life cycle process for the creation and deployment of virtual networks through:

- profiling, which captures the "blueprint" of a virtual network architecture in terms of a comprehensive profiling script;
- spawning, which executes the profiling script to set-up network topology, and address space and bind transport control and management objects into the physical infrastructure; and
- management, which supports virtual network architecting and resource management.

Virtual networks, spawned by the Genesis Kernel operate in isolation with their traffic being carried securely and independently from other networks. Furthermore, "child" networks, created through spawning by "parent" networks inherit architectural components from their parent networks, including life cycle support. Thus a child virtual network can be a parent (i.e., provider) to its own child networks, creating a notion of "nested virtual networks" within a virtual network.

6. CONCLUSION

In this paper, we have discussed the state-of-the-art in programmable networks. We have presented a set of characteristics and generalized model for programmable networks, which has allowed us to better understand the relationship between the existing body of work on programmable networking. The generalized model comprises communication and computation models. By "grafting" a computation model to the communication model a network architecture can be made programmable. The generalized model includes node kernels to manage network node resources, and network

programming environments that provide tools for programming network architectures.

We believe that a number of important innovations are creating a paradigm shift in networking leading to higher levels of network programmability. These are:

- *separation* of hardware from software;
- *availability* of open programmable interfaces;
- *virtualization* of the networking infrastructure;
- *rapid creation* and *deployment* of new network services; and
- *safe resource partitioning and coexistence* of distinct network architectures over the same physical networking hardware.

Programmable networks provide a foundation for architecting, composing and deploying virtual network architectures through the availability of open programmable interfaces, resource partitioning and the virtualization of the networking infrastructure. We believe that a key challenge is the development of programmable virtual networking environments based on these foundations.

7. ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation (NSF) under CAREER Award ANI-9876299 and with support from COMET Group industrial sponsors. In particular, we would like to thank the Intel Corporation, Hitachi Limited and Nortel Networks for supporting the Genesis Project. John B. Vicente (Intel Corp) would like to thank the Intel Research Council for their support during his visit with the Center for Telecommunications Research, Columbia University. Miki Kazuho (Hitachi, Ltd) would like to express his thanks to Hitachi Ltd for their support of his work on Programmable Networks at Columbia University. Hermann G. De Meer is grateful to Deutsche Forschungsgemeinschaft (DFG) for providing his fellowship and research grant Me 1703/2-1. Daniel A. Villela would like to thank the National Council for Scientific and Technological Development (CNPq-Brazil) for sponsoring his scholarship at Columbia University (ref. 200168/98-3).

8. REFERENCES

[1] ABONE, Active network Backbone, http://www.csl.sri.com/ancors/abone/

[2] Adam, C.M., Lazar, A.A., Lim, K.-S., and Marconcini, F., "The Binding Interface Base Specification Revision 2.0", *OPENSIG Workshop on Open Signalling for ATM, Internet and Mobile Networks*, Cambridge, UK, April 1997.

[3] Alexander, D.S., Arbaugh, W.A., Hicks, M.A., Kakkar P., Keromytis A., Moore J.T., Nettles S.M., and Smith J.M., "The SwitchWare Active Network Architecture", *IEEE Network Special Issue on Active and Controllable Networks*, vol. 12 no. 3, 1998.

[4] Amir E., McCanne S., and Katz R., "An Active Service Framework and its Application to real-time Multimedia Transcoding", *Proceedings ACM SIGCOMM' 98*, Vancouver, Canada

[5] Alexander D.S., Braden B., Gunter C.A., Jackson W.A., Keromytis A.D., Milden G.A., and Wetherall D.A., "Active Network Encapsulation Protocol (ANEP)", *Active Networks Group Draft*, July 1997

[6] Angin, O., Campbell, A.T., Kounavis, M.E., and Liao, R.R.-F., "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking", *IEEE Personal Communications Magazine, Special Issue on Adaptive Mobile Systems*, August 1998.

[7] Balachandran, A., Campbell, A.T., and Kounavis, M.E, "Active Filters: Delivering Scalable Media to Mobile Devices" , *Proc. Seventh International Workshop on Network and Operating System Support for Digital Audio and Video*, St Louis, May, 1997.

[8] Bershad,B.N., et al., "Extensibility, Safety and Performance in the SPIN Operating System", *Fifth ACM Symposium on Operating Systems* Principles, Copper Mountain, December 1995.

[9] Biswas, J., et al., " The IEEE P1520 Standards Initiative for Programmable Network Interfaces" *IEEE Communications Magazine, Special Issue on Programmable Networks*, October, 1998.

[10] Braden,B., "Active Signaling Protocols", *Active Networks Workshop*, Tucson AZ, March 1998.

[11] Calvert, K. et al, “Architectural Framework for Active Networks”, *Active Networks Working Group Draft*, July 1998.

[12] Calvert, K. et. al, “Directions in Active networks”, *IEEE Communications Magazine, Special Issue on Programmable Networks*, October 1998.

[13] Campbell A.T., De Meer H.G., Kounavis M.E., Miki K., Vicente J.B., and Villela D., “The Genesis Kernel: A Virtual Network Operating System for Spawning Network Architectures”, *Second International Conference on Open Architectures and Network Programming (OPENARCH)*, New York, 1999.

[14] "CANEs: Composable Active Network Elements", http://www.cc.gatech.edu/ projects/canes/

[15] Chan, M.-C., Huard, J.-F., Lazar, A.A., and Lim, K.-S., “On Realizing a Broadband Kernel for Multimedia Networks”, *3rd COST 237 Workshop on Multimedia Telecommunications and Applications*, Barcelona, Spain, November 25-27, 1996.

[16] Chen and Jackson, Editorial, IEEE Network Magazine, *Special Issue on Programmable and Active Networks*, May 1998

[17] Chandra, P. et al., “Darwin: Customizable Resource Management for Value-added Network Services”, *Sixth IEEE International Conference on Network Protocols (ICNP'98*), Austin, October 1998.

[18] Coulson, G., et al., “The Design of a QOS-Controlled ATM-Based Communications System in Chorus”, *IEEE Journal of Selected Areas in Communications*, vol.13, no.4, May 1995.

[19] Cplane Inc., www.cplane.com

[20] DARPA Active Network Program, http://www.darpa.mil/ito/research/anets/projects.html, 1996.

[21] Da Silva, S., Florissi, D. and Yemini, Y., “NetScript: A Language-Based Approach to Active Networks”, *Technical Report, Computer Science Dept., Columbia University* January 27, 1998.

[22] Decasper, D., Parulkar, G., Plattner, B., “A Scalable, High Performance Active Network Node”, *IEEE Network*, January 1999.

[23] Delgrossi, L. and Ferrari D., “A Virtual Network Service for Integrated-Services Internetworks”, *7th International Workshop on Network and Operating System Support for Digital Audio and Video*, St. Louis, May 1997.

[24] Engler, D.R., Kaashoek, M.F. and O’Toole ,J., “Exokernel: An Operating System Architecture for Application-Level Resource Management”, *Fifth ACM Symposium on Operating Systems* Principles, Copper Mountain, December 1995.

[25] Feldmeier, D.C., at al. “Protocol Boosters”, *IEEE Journal on Selected Areas in Communications, Special Issue on Protocol Architectures for the 21st Century*, 1998.

[26] Ferguson, P. and Huston, G., “What is a VPN?”, *OPENSIG'98 Workshop on Open Signalling for ATM, Internet and Mobile Networks*, Toronto, October 1998.

[27] Hartman, J., et al., “Liquid Software: A New Paradigm for Networked Systems”, *Technical Report 96-11, Dept. of Computer Science, Univ. of Arizona*, 1996.

[28] Hicks, M., et al., “PLAN: A Programming Language for Active Networks”, *Proc ICFP'98*, 1998.

[29] Kulkarni, A.B. Minden G.J., Hill, R., Wijata, Y., Gopinath, A., Sheth, S., Wahhab, F., Pindi, H., and Nagarajan, A., “Implementation of a Prototype Active Network”, *First International Conference on Open Architectures and Network Programming (OPENARCH)*, San Francisco, 1998.

[30] Lazar, A.A.,“Programming Telecommunication Networks”, *IEEE Network*, vol.11, no.5, September/October 1997.

[31] Lazar, A.A., and A.T Campbell, “Spawning Network Architectures”, *Technical Report, Center for Telecommunications Research*, Columbia University, 1997.

[32] Liao, R.-F. and Campbell, A.T., “On Programmable Universal Mobile Channels in a Cellular Internet”, *4th ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98)* , Dallas, October, 1998

[33] Van der Merwe, J.E., and Leslie, I.M., “Switchlets and Dynamic Virtual ATM

Networks", *Proc Integrated Network Management V*, May 1997.

[34] Van der Merwe, J.E., Rooney, S., Leslie, I.M. and Crosby, S.A., "The Tempest - A Practical Framework for Network Programmability", *IEEE Network*, November 1997.

[35] DARPA Active Network Mail List Archives, 1996. http://www.ittc.ukans.edu/Projects/Activenets

[36] Montz, A.B., et al., "Scout: A Communications-Oriented Operating System", *Technical Report 94-20, University of Arizona, Dept. of Computer Science*, June 1994.

[37] Mobiware Toolkit v1.0 source code distribution http://www.comet.columbia.edu/mobiware

[38] Multiservice Switching Forum (MSF) , www.msforum.org

[39] Open Signaling Working Group comet.columbia.edu/opensig/

[40] Peterson L., "NodeOS Interface Specification", *Technical Report, Active Networks NodeOS Working Group*, February 2, 1999

[41] Schwartz, B., Jackson, W.A., Strayer W.T., Zhou, W., Rockwell, R.D., and Partridge, C., "Smart Packets for Active Networks", *Second International Conference on Open Architectures and Network Programming (OPENARCH)*, New York, 1999.

[42] Tennenhouse, D., and Wetherall, D., "Towards an Active Network Architecture", *Proceedings, Multimedia Computing and Networking*, San Jose, CA, 1996.

[43] Tennenhouse, D., et al., "A Survey of Active Network Research", *IEEE Communications* Magazine, January 1997.

[44] Touch, J. and Hotz, S., "The X-Bone", Third *Global Internet Mini-Conference in conjunction with Globecom '98* Sydney, Australia, November 1998.

[45] Wetherall, D., Guttag, J. and Tennenhouse, D., "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols", *Proc. IEEE OPENARCH'98*, San Francisco, CA, April 1998.

[46] Vinoski, S.,"CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", *IEEE Communications Magazine*, Vol. 14, No. 2, February, 1997.

[47] xbind code http://comet.columbia.edu/xbind

[48] Xbind Inc., www.xbind.com

[49] Yemini, Y., and Da Silva, S, "Towards Programmable Networks", *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, L'Aquila, Italy, October, 1996.