DOI:10.1145/2699392

Future Internet testbeds permit experiments not possible in today's public Net or commercial cloud services.

BY MARK BERMAN, PIET DEMEESTER, JAE WOO LEE, KIRAN NAGARAJA, MICHAEL ZINK, DIDIER COLLE, DILIP KUMAR KRISHNAPPA, DIPANKAR RAYCHAUDHURI, HENNING SCHULZRINNE, IVAN SESKAR, AND SACHIN SHARMA

Future Internets Escape the Simulator

STANDARDIZATION OF BASIC underlying protocols such as the Internet Protocol (IP) has enabled rapid growth and widespread adoption of the global Internet. However, standardization carries the attendant risks of reducing variability and slowing the pace of progress. Validation and deployment of potential innovations by researchers in networking, distributed computing, and cloud computing are often hampered by *Internet ossification*, the inertia associated with the accumulated mass of hardware, software, and protocols that constitute the global, public Internet.[24] Researchers simply cannot develop, test, and deploy certain classes of important innovations into the Internet. In the best case, the experimental components and traffic would be ignored; in the worst case, they could disrupt the correct behavior of the Internet. Cloud computing researchers confront a similar dilemma. In order to maintain uniformity and efficiency in their data centers, commercial cloud providers generally do not provide "under the hood" controls that permit modification to the underlying network topology or protocols that comprise the cloud environment.

A clear example of the challenge is apparent to anyone tracking the pace of adoption of IPv6, a relatively modest revamping of IP. Because IPv6 deployment affects components throughout the Internet, years of extensive review, planning, and coordination have been required to ensure a smooth, if slow, transition. For researchers contemplating more fundamental innovations, such as non-IP protocols or new routing approaches, the barriers are correspondingly higher. Accordingly, researchers have been forced to employ compromise measures, such as validating their novel concepts only in simulation, or in modest, isolated laboratory configurations. These environments permit a wide range of experiments, but at the expense of the realism that comes with a large-scale physical deployment.

» key insights

- **By design, the Internet is inhospitable to many classes of experiments that could lead to major advances in networking, resulting in a problem known as *Internet ossification*.**
- **Researchers worldwide are increasingly turning to future Internet and distributed cloud (FIDC) testbeds, such as GENI in the U.S. and FIRE in the E.U., where they can conduct networking, distributed computing, and cloud computing experiments in a distributed laboratory setting.**
- **Although the technology is still in flux, FIDC testbeds are already supporting important research and education initiatives. As these testbeds join together in international federations, their benefits increase combinatorially.**

IMAGE BY OMELCHENKO

Figure 1. Current GENI deployment phase candidate sites.

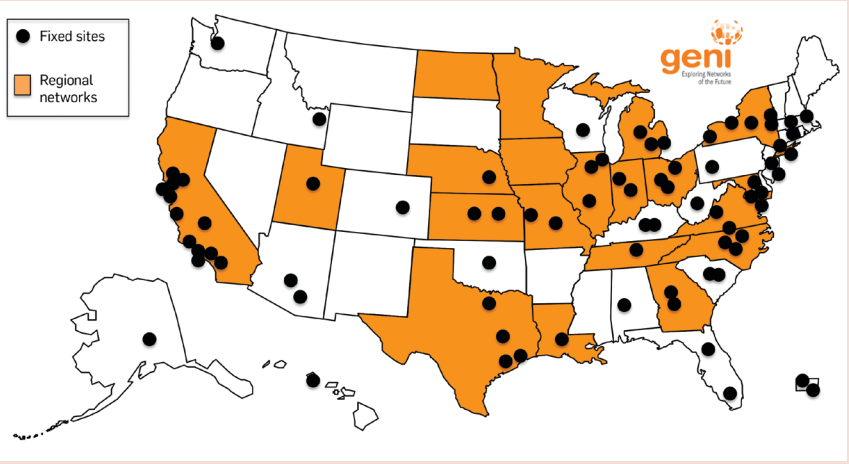
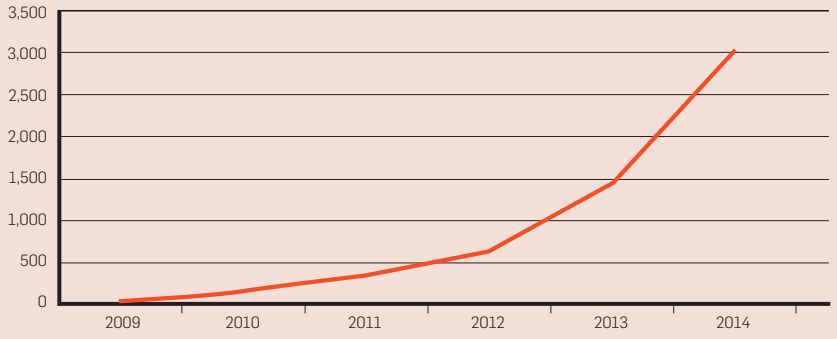


Figure 2. Cumulative unique GENI users.



FIDC Testbeds

Future Internet and distributed cloud testbeds are a promising response to these concerns, providing a virtualized environment where multiple experimental networks may be simultaneously deployed, tested, and validated at significant scale, within a shared platform. These testbeds, beginning with Global Environment for Networking Innovation (GENI) in the U.S.[2] and Future Internet Research & Experimentation (FIRE) in the E.U.,[4] are quickly gaining prevalence and scale. The GENI testbed, for instance, is currently completing its initial deployment phase to 50 sites, with a target of 100–200 sites (see Figure 1).

FIDC testbeds have proven to be successful and versatile in supporting a wide variety of work. For example, as shown in Figure 2, over 3,000 unique users to date have allocated GENI testbed resources for their research and educational work, with the pace of use rapidly increasing. This article is a survey, which introduces key FIDC testbed concepts and presents selected example applications. Interested readers are encouraged to pursue further details found in the referenced documents.

Future Internet testbeds are quickly becoming a global phenomenon supported by a growing international community. Key underlying technologies developed by GENI, FIRE, the University of Utah's Flux group, the OpenFlow and software-defined networking (SDN) communities, the VNode project in Japan, and others are rapidly being combined to form heterogeneous testbeds and interoperable federations. In addition to the U.S. and E.U., national-scale efforts are under way or in planning stages in Japan, Mexico, Canada, China, and South Korea. While these testbeds are built on a variety of underlying technologies, they share certain core capabilities.

Key concepts: Slicing and deep programmability. Establishing a FIDC testbed of meaningful scale requires a significant investment of money and effort. Exclusive use is generally not feasible, so virtualization quickly becomes a practical necessity to support a sizable research community. A key feature of FIDC testbeds is the ability simultaneously to virtualize both computing and networking resources and to assemble them into end-to-end configurations or *slices*. While virtualization of computational resources is reasonably well understood, adding network programmability and virtualization to the mix presents a challenge. However, it is the property of *deep programmability* that creates the key opportunities for innovation in a FIDC testbed. In a deeply programmable environment, the experimenter controls the behavior of computing, storage, routing, and forwarding components deep inside the network, not just at or near the network edge.

As with any virtualized environment, FIDC testbeds present each user with the illusion of exclusive control over shared resources. In a FIDC testbed, the researcher's usual view of his or her resource suite is as a collection of general-purpose computers connected in experimenter-specified topologies by a programmable network, perhaps augmented with special-purpose devices (for example, sensors, high-performance computing resources, and cyberphysical systems). Because these resources are, at their core, real physical computers, networks, and storage devices, rather than simulations, FIDC testbeds can also create powerful opportunities for *end user opt-in*, the ability to run experimental configurations that offer advanced services to real end users.

The selection of virtualization and network programmability approaches represents a trade-off among performance, isolation, and cost. Different testbeds choose different operating points within this space. In fact, some offer multiple virtualization options, even for a single resource type. For example, in GENI a slice's computation resources may include bare metal, Xen, and Vserver hosts, while network programmability options may include modular software routers (for example,

Click), or hardware- or software-based OpenFlow software-defined networking. Similarly, Flare programmable switches in the Japanese VNode network virtualization testbed provide a wide variety of deep programmability options, including efficient native implementations of Click and OpenFlow.

Case Studies

The four efforts discussed here help to illustrate the broad applicability of FIDC testbeds in supporting advanced research and applications. These projects, recently conducted in the GENI testbed and FIRE's iLab.t,[8] apply a wide variety of technology innovations. These range from "clean slate," non-IP protocols to novel cloud computing paradigms and OpenFlow-based software-defined networking. The application domains are similarly diverse, including peer-to-peer message passing, weather forecasting, and video delivery. What these projects share in common is a need to program the underlying network infrastructure in ways not available in the current public Internet.

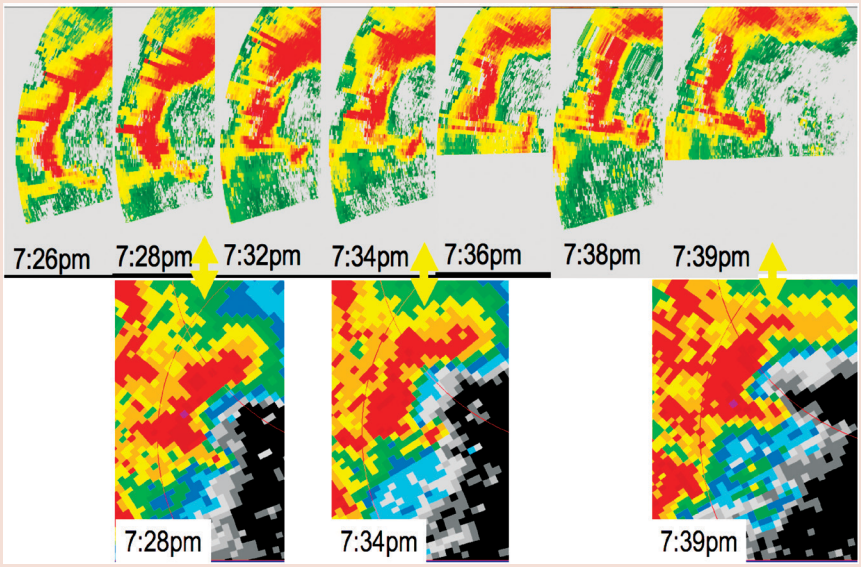
Cloud-based, near-term, localized weather forecasting. CloudCast[10] provides personalized short-term weather forecasts to clients based on their current location using cloud services, generating accurate forecasts tens of minutes in the future for small areas. These short-term "Nowcasts" have profound public safety implications for emergency response to dangerous weather (for example, tornados and severe thunderstorms). Researchers at the U.S. National Science Foundation (NSF) Engineering Research Center for Collaborative Adaptive Sensing of the Atmosphere (CASA) are evaluating the benefits and feasibility of providing a large-scale CloudCast service, based on improved weather observations from networks of small, low-cost X-band radars.

An example of the potential benefits of nowcasting is seen in Figure 3, which shows a comparison between data provided by a CASA radar network and data from the current National Weather Service (NWS) NEXRAD system. The upper series of images, from the CASA system, provides much finer temporal and spatial resolution, clearly showing a "hook echo," a rotational event that is a potential indicator of a tornado. CloudCast produces improved forecasts by employing a larger number of relatively short-range radars. These radar networks improve resolution and avoid low-altitude blind spots created over long distances by the curvature of the Earth. Exploiting the data from these sensor networks is computationally intensive, but highly bursty. For example, during a 75-day intensive operation period in the CASA Oklahoma testbed in the spring of 2011, the climatological peak season for severe weather, only 90 hours (or 5% of the period) featured ongoing convective precipitation.

For this reason, the infrastructure-as-a-service (IaaS) model offered by a cloud-computing environment is a very promising approach for a large-scale CloudCast implementation. Instead of acquiring dedicated computing hardware, which would sit idle most of the time, nowcasts could be computed in the cloud, on an as-needed basis. Clearly this economic benefit needs to be balanced with timeliness. Because nowcasts have a very short time horizon, they must be provided to the end users with as little delay as possible. For example, in the case of a 15-minute nowcast, a difference of just a minute or two in nowcast delivery time can be significant. Especially in severe weather situations, providing maximum lead-time can save lives and property.

Experiment design and results. Because timely nowcast generation and delivery relies on a well-engineered combination of network and computational resources, CASA researchers hypothesized that research cloud platforms coupled with control over network assets in an FIDC testbed might outperform commercial clouds. For this investigation, CASA researchers turned to the GENI testbed. They conducted a series of experiments comparing two commercial cloud platforms (Amazon AWS and Rackspace) and two research cloud platforms (GENICloud and ExoGENI), with a goal of improving overall delivery times by reducing

Figure 3. Data from CASA's Oklahoma radar network (top) shows a "hook echo" forming at the center of the image.



Timing results for nowcast algorithm.

Instances	Memory (GB)	Execution time (s)	Total time (s)
Amazon EC2	7.5	74.34	95.08
Rackspace	8	96.53	120.33
GENICloud	8	67.45	78.60
ExoGENI	8	56.83	72.07

data transmission time to and from the cloud computing resources.

Detailed experimental results from four cloud platforms are presented in Krishnappa et al[9] and summarized in the accompanying table. Similar computing resources are reserved on each cloud platform. Execution times are measured by replaying one hour of recorded weather data observed by the CASA radar network in southwestern Oklahoma. The cloud-based nowcast instance receives radar scans, generates 1-minute to 15-minute nowcasts, and stores them on the instance's storage. This continues for one hour, and the average execution time for the generation of each nowcast is measured.

As shown in the table, computation times range from 56.83 seconds on the ExoGENI platform to 96.53 seconds on a Rackspace instance. In comparison to their commercial counterparts, both research cloud instances take less time (67.45 seconds and 56.83 seconds respectively) to compute the nowcasts.

Live process measurement. As a proof of concept for the CloudCast application on cloud services, a live measurement experiment was carried out on each cloud instance to calculate the overall nowcast delivery time from the moment the radar generates the data. These live measurements used a prototype CASA radar located on the University of Massachusetts Amherst campus.[16] Overall results are shown in the last column of the table. These figures provide the most valid information on CloudCast feasibility, because they include the full product delivery chain: raw data transmission time to the instance executing the algorithm, computing time to generate 15-minute nowcast images, and time to transmit the resulting nowcast images to a central Web server for access by end users.

The average overall time for the whole nowcasting process was 95.08 seconds for the EC2 instance, of which 71.98 seconds are consumed in the computation of 15-minute nowcasts on the cloud instance. The remaining 23.10 seconds are used to send data from the radar to the receiving instance, and to transfer the predicted images back to the central server for access by end users. Similarly, the total time taken for executing the whole nowcasting process on Rackspace, GENICloud, and ExoGENI instances is 120.33, 78.60, 72.07 seconds, respectively. Significantly, the dedicated networking resources associated with the research cloud platforms markedly improved overall performance, even without specific tuning for the CloudCast application.

This series of experiments provides convincing evidence of the feasibility of performing short-term weather forecasts in a cloud, with only two minutes to generate and disseminate nowcasts. Additional detailed data on the network and computation components of the overall delivery timeline will provide useful guidance to a potential large-scale CloudCast implementation.

MobilityFirst future Internet architecture. MobilityFirst[20] is a future Internet architecture currently under development as part of the U.S. National Science Foundation (NSF) Future Internet Architecture (FIA) program. The architecture targets a broad set of performance, reliability, and security goals with particular focus on enabling seamless at-scale mobility and establishing trustworthiness as a basic ele-

Figure 4. MobilityFirst protocol stack with GUID service layer providing the new narrow waist.

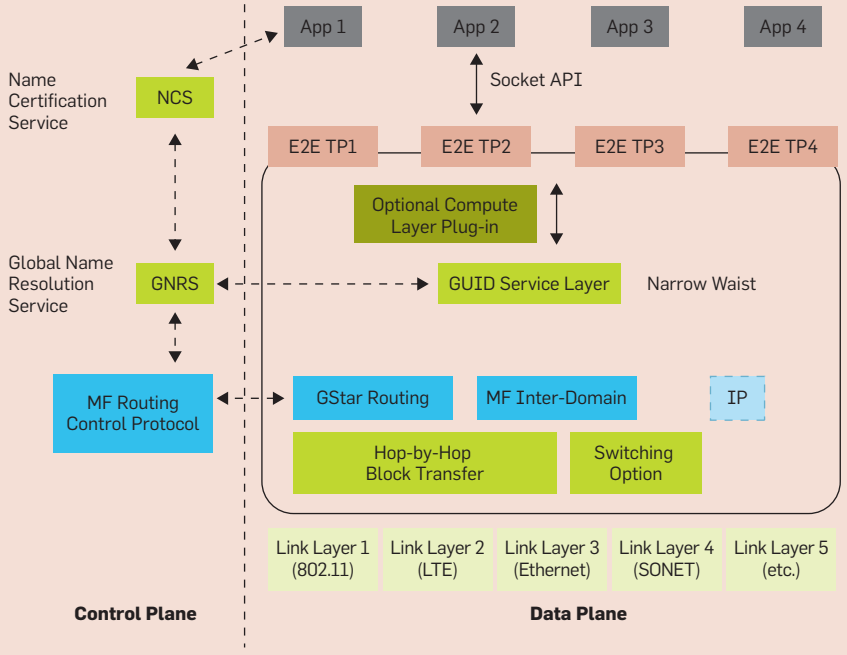
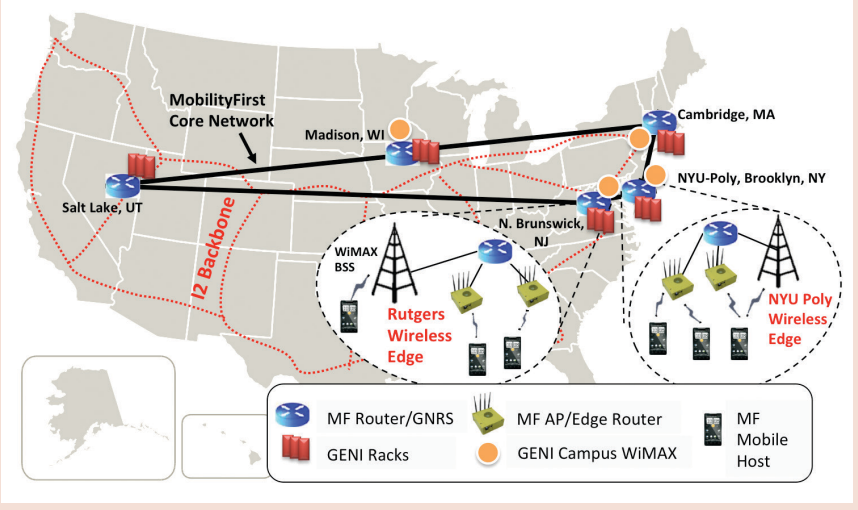


Figure 5. Wide-area deployment of MobilityFirst prototype network on GENI testbed.



ment of network communication in the future Internet. New protocols and in-network services will address wireless access challenges and will provide native support for emerging mobile Internet applications rich in content and context aspects.

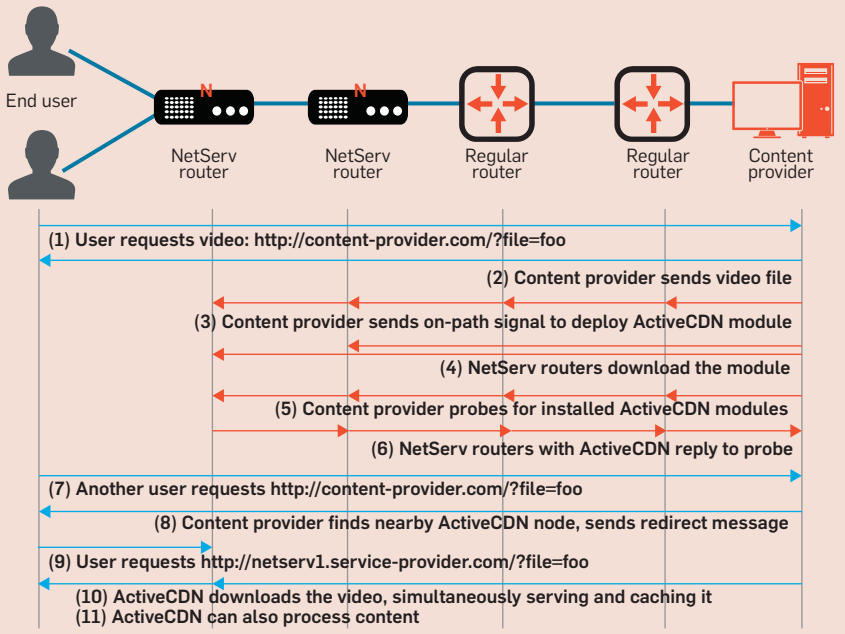
A key design choice made in realizing the preceding goals is to have a logically central naming service at the core of the architecture that is fast, highly scalable, and globally distributed. The naming service includes a name certification service (NCS) that translates human-readable names to unique endpoint identifiers (GUIDs) and a global name resolution service (GNRS) that maps GUIDs to one or more locators (that is, topological or network addresses) of a network-attached object. To enable decentralized trust, the GUID is derived from a public key of an asymmetric key-pair to provide self-certifiable property, that is, a trusted third party is not required to verify identity between two communicating endpoints.[1] Accordingly, the protocol stack (Figure 4) provides name-based networking abstractions primarily by introducing a GUID Service Layer. This new name-based *narrow waist* contrasts with the IP stack in which names and addresses are conflated making seamless mobility a difficult problem. Protocols for scalable inter-domain routing, for reliable data transport, and for scalable multipoint delivery (for example, multicast, anycast, and multihoming), all make use of mappings within GNRS to provide fast dynamic bindings under mobility.[20] *Late-binding*, where only coarse-grain locator information (for example, destination network) is resolved at the packet source, allows packets to be bound at the destination network to the latest address of a mobile endpoint. Data is transported as large blocks in a hop-by-hop manner using a segmented data transport[12] and storage-aware routing[17] protocols that leverage in-network storage to temporarily buffer data under transient problems in wireless access networks such as temporary disconnections under mobility and variable link quality. Finally, a new socket interface provides applications with name-based access to in-network services including direct operations on content (for example, *get* and *put*) and other more abstract objects such as context that can be named using a GUID.[3]

In a deeply programmable environment, the experimenter controls the behavior of computing, storage, routing, and forwarding components deep inside network, not just at or near the network edge.

Evaluation and testbed considerations. To validate these key concepts, the MobilityFirst team chose to build, deploy, and evaluate a prototype MobilityFirst network in a realistic setting. This prototype enables them to characterize the performance and scalability of the GNRS, of routing and transport protocols, and also of end-to-end applications over the MobilityFirst protocol stack. Since protocol behavior and overall performance depend significantly on network properties, a local testbed with limited emulation capabilities alone will fall short. The following capabilities were deemed crucial for the testbed-based evaluations: flexible configurability of network with deep programmability of nodes and network elements; scale of hundreds or more programmable nodes; network with interconnects that are diverse in their latency and bandwidth characteristics; and finally, a broad choice of traditional and emerging network technologies, including wireless (for example, WiFi, 4G WiMAX, and LTE) and SDN (for example, OpenFlow switches).

While the protocol designs were first explored in simulation environments (for example, ns-3 and custom simulator), these were followed up with full-feature prototypes and evaluations in testbed (PlanetLab, GENI) and commercial cloud platforms (Amazon EC2). Prototypes for the key components of the architecture, that is, the naming, routing, and transport services have been implemented. A Click-based software router implements the storage-aware routing and transport services, and closely integrates a GNRS service instance for dynamic name bindings. For the GNRS service, two alternate designs were simultaneously prototyped. One design uses an in-network DHT to store GUID-address mappings (with multiple replicas for each mapping) where service instances are co-located with the routing fabric to minimize access latency.[28] The second design is a flexible overlay implementation, and optimizes service latencies by considering access locality and exerting fine-grained control over replica placement.[27] For end-hosts, the proto-

Figure 6. How ActiveCDN works.



col stack in Figure 4 was implemented for Linux and Android platforms, along with implementations of the new socket API.

Large-scale simulations and limited wide-area end-to-end experiments bear out the benefits of the architecture. Key results include: Both GNRS designs provided 100ms lookup latencies or better when evaluated under current Internet topology assumptions including topologies with up to 26K ASs and 90K links. These latencies could be even smaller (few 10s of ms) in the future if the Internet topology flattens with time (that is, ASs have more direct paths to the core).[28] Limited deployments of GNRS on the GENI testbed across seven rack sites showed 95th percentile latency under 80ms, which mostly reflect the inter-site RTTs; segmented data transport with storage-aware routing significantly improves end-to-end data transfers, particularly under transient wireless access conditions—in some cases an order of magnitude throughput improvements.[12,22] The Click-based router achieves a forwarding rate of approximately 750Mbps on GENI programmable nodes, while the SDN version of the router (OpenFlow/Floodlight with Pronto 3290) achieves close to the 1Gbps line rate.

Wide-area deployment on GENI testbed. Offering extensive heterogeneous resources, geographic diversity, and deep programmability, the GENI testbed was a clear choice for MobilityFirst's goal of at-scale realistic evaluation. Several of the early deployments on GENI were of standalone prototypes and demonstrated the working of key protocols including GNRS and GSTAR in the wide area. Recently, we are maintaining long-running deployment of the more complete prototype network to enable network-level evaluations and to provide an open platform for novel application development. The deployed components include our Click-based router, the in-network GNRS, along with end-hosts running the MobilityFirst stack. It is possible to connect all nodes on a single layer-2 network or establish domains using VLAN programming. The Click routers and GNRS servers are deployed on either bare-metal or VMs, with some as edge-routers having interfaces connected to wireless edge networks with WiFi or WiMAX access. End-hosts are run on GENI testbed nodes or can be user-carried devices (for example, on Android phones and tablets) at GENI campus sites, which can present natural mobile data traffic and mobility events. Experiments so far include validation of in-network DHT version of GNRS, evaluations of reliable content delivery to multihomed mobiles, and a recent demonstration at the 18th GENI Engineering Conference of a novel P2P messaging app that uses name-based networking to address contextual objects like location.

The messaging app—'Drop It'—uses GUIDs to name well-bounded locations such as a conference room or a campus bus stop, while mobile users either 'drop' messages at locations or pick up messages dropped by others. In this pure P2P version, messages remain on originating phones and are retrieved directly when picked up by another phone visiting the location. This is done by maintaining a GNRS mapping from location GUID to the set of phone-GUIDs (or their addresses) that dropped messages at that location, and sending a multicast request to the location GUID, which leverages the hybrid name-address routing in MobilityFirst. This application was demonstrated and evaluated using a multi-site GENI slice (shown in Figure 5) deployment with 10 MobilityFirst software routers (each with GNRS instance), five of which were edge routers providing WiFi and WiMAX client access for Android phones.

Current work and the next phase of the MobilityFirst project are focused on at-scale evaluations and real-world trials of the architecture, including end user opt-in. The plan for GENI-based deployments include footprint of few hundred network elements comprising a core network with several edge networks providing access to clients from participating campus networks or those deployed on third-party cloud platforms (for example, Amazon's EC2).

The GENI deployment will also serve as a crucial evaluation anchor and provide the foundation for three distinct real-world network environment trials in the next phase of the MobilityFirst project. These include a mobile data services trial with a wireless ISP (5Nines) in Madison, WI; a "content production and delivery network" trial involving several public broadcasting stations in Pennsylvania; and a weather emergency notification system based on the CASA radar network described here, with end users in

the Dallas/Fort Worth area. These trials are expected to provide a firm basis for validation of the protocol stack and its utility for advanced mobile, content, context, and cloud applications, while also advancing the technology to the field-deployment stage.

NetServ in-network services architecture. Computing devices connected to the Internet today largely fall into two categories: end systems and routers. The primary goal of a router is to move network packets as quickly as possible. To this end, router vendors typically employ custom operating systems that can take full advantage of the underlying hardware platform. This model limits the ability of a router operator who wishes to customize operation by installing a custom application inside the router—an application developed by a third-party vendor, for instance. So far, the inability to fully customize routers has not hampered the growth of the Internet. Consumer demands have been met by innovative applications running on end systems. The role of routers inside the network has remained the same: moving packets as quickly as possible.

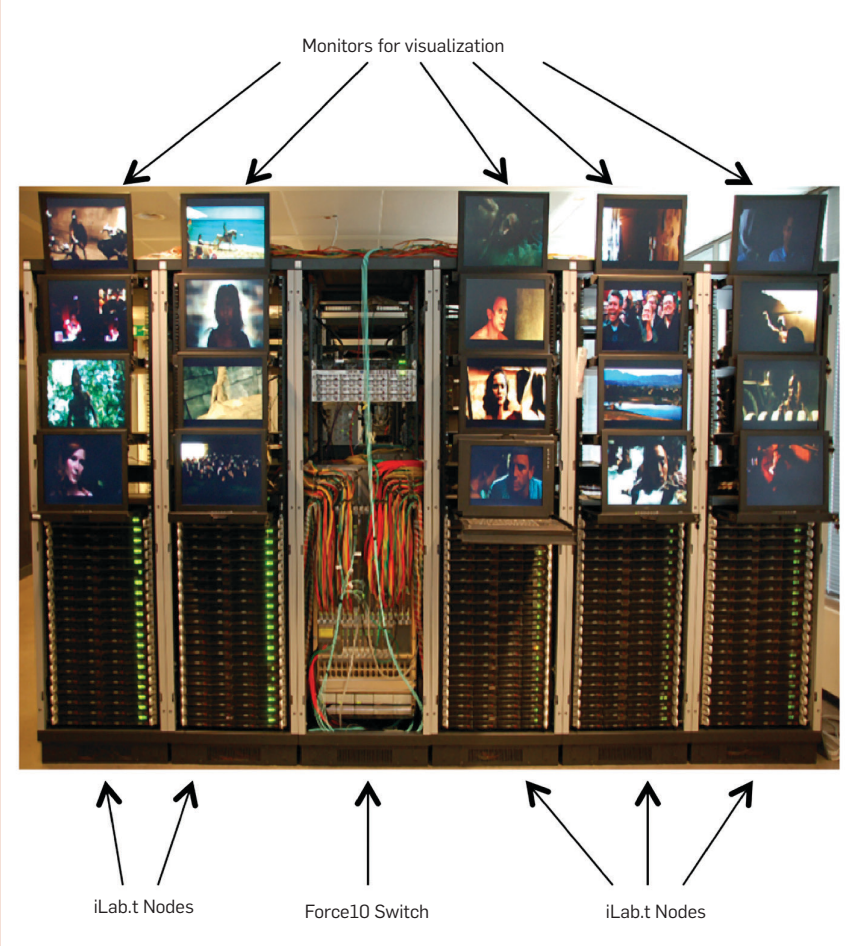
Many signs indicate, however, that the traditional dichotomy between end systems and routers no longer models the complexity of modern networks. With the advent of P2P networking, end systems now frequently assume the role of routers by relaying packets on behalf of other end systems. P2P applications must resort to such overlay networks because routers running fixed software are ill-equipped to form P2P networks, which are usually dynamic, ad hoc, and optimized for specific applications.

Router functionality has not stagnated during this period. Networks grow more complex, ISPs' need and desire for traffic engineering grow more sophisticated, and router vendors respond with new functions like QoS, firewall, VPN, IPsec, NAT, Web cache and rate limiting. Routers have become programmable, but not in the same way that end systems are programmable. Router functions are still limited to a predetermined set supplied by the router vendor itself or the third-party developers approved by the router vendor. In fact, the closed nature of mainstream routers brought the proliferation of middleboxes—network devices that are daisy-chained with routers to provide added functionality—adding to ISPs' network management problems.

Architecture goals. The NetServ team envisions a future Internet where all nodes support a common runtime environment, which eliminates the distinction between end systems and routers for the purpose of running network services.[5,11,14] Network services run on any node, from backbone routers to set-top boxes, and the services are dynamically installed, removed, and migrated to optimal locations on the Internet. As a first step toward that vision, NetServ provides an architectural framework for dynamically deploying in-network services on edge routers. Modules implementing various network functions can be installed at runtime on any NetServ-enabled router on the Internet.

NetServ has adopted five goals in designing a viable in-network service framework. First, NetServ must enable new classes of economically compelling applications that cannot be achieved with existing middleboxes and end systems. One such application is ActiveCDN, which has been demonstrated running on a set of NetServ nodes deployed in the GENI testbed. ActiveCDN is a use case that shows how NetServ can facilitate an economic alliance between ISPs and content providers. Second, NetServ provides a unified runtime environment. A NetServ module can act as an end system application engaged in client-server networking, as a router add-on performing deep packet inspection, or as both at the same time. Third, NetServ provides a wide-area deployment mechanism, using a standardized on-path signaling protocol.[6] A signaling packet is sent to a network destination and the packet is routed as

Figure 7. Setup for failure recovery experiment on ILab.t facility at iMinds.



usual by the regular IP routers. When the signaling packet passes through NetServ-enabled routers, however, the packet will trigger module installations on the routers. Fourth, NetServ provides a multi-user execution environment by running modules inside Java Virtual Machines (JVMs) allocated per user. Fifth, in order to address the performance overhead of running modules in JVM, NetServ has proposed a scalability solution using OpenFlow.

NetServ on GENI testbed. FIDC testbeds provide an ideal platform to deploy and test NetServ. NetServ requires deep programmability to deploy in-network packet processing modules. While emulation-based environments like Mininet and Emulab offer localized deep programmability, a sizable FIDC testbed creates additional experiment options, including measurements of signaling latencies arising from real geographic distances and network topologies. GENI has been an integral part of the NetServ project, in continual use for developing, testing and demonstrating NetServ functionality. The NetServ team has demonstrated two NetServ applications, ActiveCDN and Overload Control,[11] running on GENI at the 9th GENI Engineering Conference.[a]

Researchers have also investigated an autonomic management solution based on NetServ,[5,14] including applying NetServ's dynamic in-network service deployment capability to the problem of counteracting a DoS attack. In one example, an experiment performed on GENI showed a flow-based intrusion detection system was able to reconfigure itself and deploy protection modules quickly using NetServ and OpenFlow, effectively counteracting a DoS attack on a VoIP application server.

ActiveCDN. Figure 6 illustrates a NetServ module deployment scenario using ActiveCDN, a NetServ application that implements CDN functionality on edge routers. When a content provider's server receives a large number of requests for certain content from a particular network area, the server can deploy an ActiveCDN module in the NetServ-enabled routers near the area. The ActiveCDN module then handles subsequent requests for the same content from the network vicinity.

Unlike traditional CDNs, the content provider controls where an ActiveCDN module is deployed. The module can be redeployed to different locations depending on the current traffic. Moreover, the module can perform custom content processing, like inserting region-specific advertisements into video streams.

Many signs indicate the traditional dichotomy between end systems and routers no longer models the complexity of modern networks.

Resilience in OpenFlow networks. Software-defined networking (SDN) decouples the control plane from the data/forwarding plane (switches or routers) of a network and embeds it into one or more external servers called controllers. OpenFlow is the reigning SDN technology, and research teams worldwide investigate many of the research challenges behind it.

The Split Architecture Carrier-Grade Networks (SPARC) project, funded through the European Seventh Framework Programme (FP7), investigated the potential applicability of OpenFlow in carrier-grade telecom networks. Carrier-grade telecom networks support hundreds of thousands of customers, assume failure recovery within 50ms (RFC 5654), and have to deliver high Quality of Experience (QoE) to their customers.

Two well-known recovery techniques, restoration and path protection,[26] are implemented in Openflow.[21] In case of restoration, the controller establishes an alternative path after a failure is detected in the network. In case of path protection, the controller establishes a disjoint alternate path together with the working path. After a failure is detected in the working path, the ingress switch redirects affected traffic to the alternate path using the fast-failover group table implementation of OpenFlow.[19]

Experiment design. The SPARC project has chosen a mixed-fidelity experimentation approach (accurately representing key parts of a system, while simplifying less important parts), using the FIRE testbed to investigate strategies and implementations for failure recovery in challenging network conditions. This approach, combining emulated topologies with realistic output devices, has two prime benefits. The first is the ability to evaluate the actual software stack, including Ericsson OpenFlow software,[18] using real video traffic and displays, simply by

a The 14-minute demo video is at http://vimeo.com/16474575.

observing the quality of video on the displays. Performing such experiments in simulation is very difficult, requiring simulation not only of network functions, but also the software stack of the video codec, the rendering pipeline, and post-processing functions of the video client. The second benefit is the opportunity to create a wide variety of pan-European topologies, controlled network traffic, and failure scenarios to evaluate thoroughly the performance requirements and implementation of switches and controller software. Experiments detailed in Sharma et al.[21] indicate a protection strategy is required to meet recovery time targets and validate the specific failure recovery approach. Although most critical components were tested in a realistic manner, further steps toward deployment will require further testing, for example, on real hardware switches.

Failure recovery experiments are conducted on the iLab.t virtual wall facility at iMinds[8] (see Figure 7). Currently, iLab.t has three virtual walls, each consisting of 100 nodes (multiprocessor, multi-core servers with up to eight 1Gb/s interfaces per server) interconnected by non-blocking Ethernet switches. iLab.t is based on Emulab software developed at the University of Utah, and researchers can build experiments by drawing a topology in a graphical user interface (GUI) or by defining it in scripts. Furthermore, a display wall (20 monitors) is present for easy visualization. iLab.t is part of the Fed4FIRE federation.[25]

Four kinds of failure recovery experiments are performed on the iLab.t facility: validation experiments in one of the pan-European topologies (shown in Figure 8) considering link failures; experiments considering link and node failures on different topologies; scalability experiments in terms of traffic load; and experiments for measuring subjective quality of video.

In the validation experiment, each client sends packets to all other clients at a constant 6ms interval. The experimenters intentionally break the London-Amsterdam link at time zero and observe the failure recovery time for both restoration and protection. Figure 9 shows the traffic destined to the London client, which is going through the Paris-London link, just before and

Figure 8. Emulated failure recovery experiment topology.

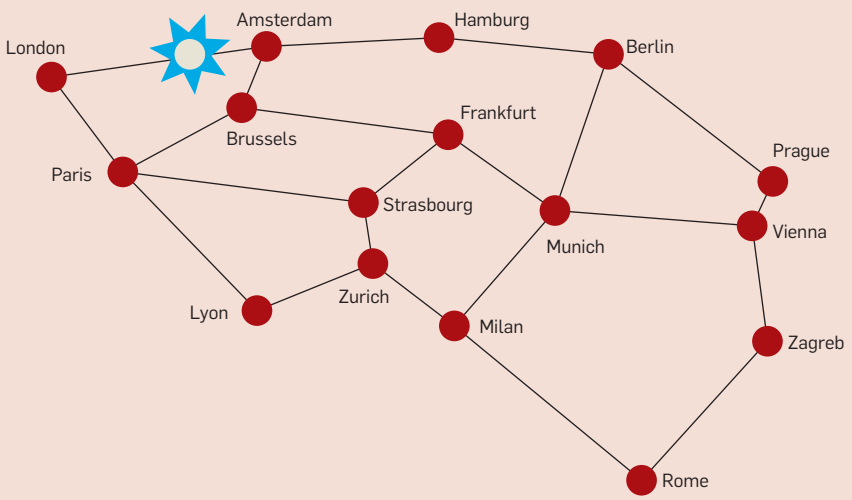


Figure 9. Failure recovery experiment results

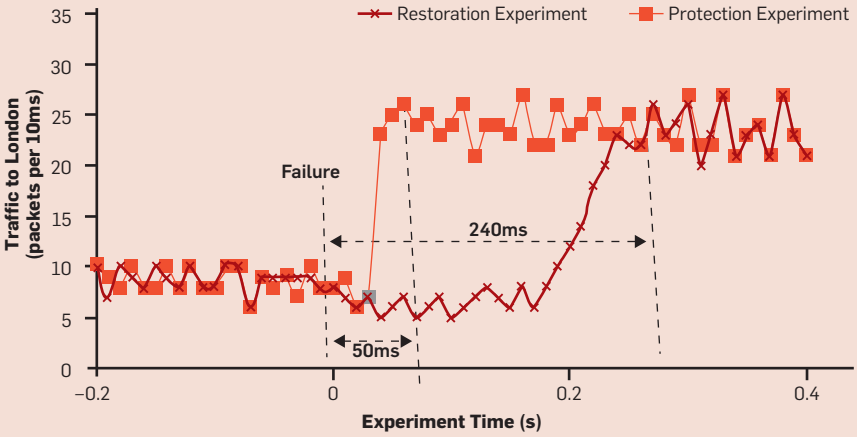
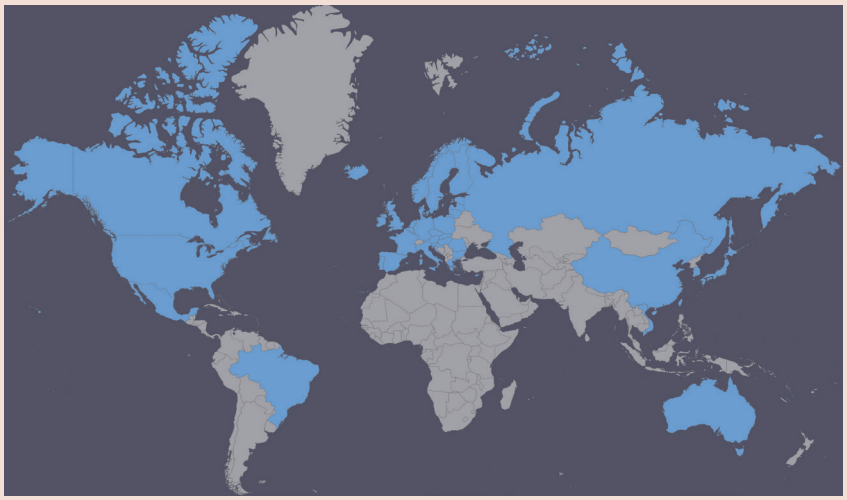


Figure 10. Ongoing FIDC testbed activity.



after the failure. After the failure (at time zero), this is the only link connecting London to the network. Therefore, after the failure all traffic to and from London traverses this link. Total restoration time is around 240ms and the total protection time around 50ms. This result shows that meeting the carrier-grade requirement of 50ms will be very difficult using a centralized controller. Protection is the only choice to meet this stringent requirement.

Failure recovery results. To evaluate the impact of the topology on the recovery time, different pan-European topologies with varying numbers of nodes and node degree are evaluated. The restoration time increases with the number of nodes because the path calculation time grows as $O(n^2)$, where n is the number of nodes. When node degree is reduced, restoration time increases because more hops are used for the restoration path and the controller needs to configure more switches. Protection does not require controller intervention and is therefore far less dependent on the network topology.

The scalability experiment shows an approximately linear relationship between the restoration time and the number of affected flows. In the protection case, the group table implementation effectively mitigates the dependency on the number of affected flows. In all experiments, protection achieves the carrier-grade recovery requirement.

In the video experiment, a video is streamed over the Real-Time Transport Protocol (RTP) and the effect of failure recovery on the video quality is assessed on the display wall. The video is sent using different maximum transmission unit (MTU) sizes. The restoration experiments show clear errors in the video, however, without a clear influence of the MTU. In contrast, protection is so fast that no observable artifacts are noticed.

FIDC Testbeds in Education

An increasing number of universities are turning to FIDC testbeds to support classroom exercises and research projects in their computer science curricula. Perhaps the most straightforward application is to adopt a FIDC testbed as a virtual laboratory for classroom networking experiments. Laboratory-based instruction is already a popular approach for introducing basic networking concepts. A leading example is Liebeherr and El Zarki's classroom networking laboratory design, with companion exercises.[13] A networking student following this approach is quickly exposed to important concepts and tools, such as datagrams, address resolution, configuring a basic IP network and debugging with tools like ping, tcpdump, and wireshark.

Instructors employing a laboratory approach find the transition to classroom experimentation in a FIDC testbed relatively straightforward. Using a virtual laboratory brings many of the same benefits as a physical networking laboratory, with the obvious exception of familiarization with physical networking components. Instructors are attracted to a virtual laboratory because it has greater elasticity for varying class sizes and eliminates the equipment cost (Liebeherr and El Zarki estimate U.S. $1,000 per student) and administrative burden to configure and maintain a classroom laboratory. In addition, FIDC testbeds by their very nature are well suited to manipulation of network settings and configurations that are generally not available to users of standard cloud computing services. Thus, instructors can easily offer basic exercises such as configuring IP addressing and routing for a simple network configuration.

FIDC testbeds have also gained popularity in non-networking courses One example is the FORGE initiative,[15] which is implementing an educational layer over the FIRE testbed facilities, in support of a broad suite of CS laboratory exercises. Another is Williams College Professor Jeannie Albrecht's undergraduate distributed systems class, where students use the GENI testbed to build distributed applications like a Web server, an online bookstore, and a P2P file-sharing system. In addition to simplifying lab setup for the course staff, students see a benefit of using FIDC testbeds for these assignments (compared to local resources), because alternative network topologies with varying conditions can be easily created. Students are then better able to observe the impact of the network on distributed applications.

Educational applications range from relatively basic exercises (for example, configuring IP networking in a small network or implementing a rudimentary Web server) to advanced experiments with novel protocols and individual research projects. Increased exposure to FIDC testbeds in university curricula is raising the comfort level of the next generation of researchers and practitioners who may benefit from these capabilities.

Future Trends and Challenges

FIDC testbeds are already creating additional opportunities for experimental investigations that are not possible in Internet-based and commercial cloud environments. Although the contributions of FIDC testbeds are clear, there remain important unanswered questions in designing and managing these community resources. Many of these questions are fundamental design tensions, such as slicability vs. fidelity, that have been anticipated since the first discussions of GENI's design.[7] Such questions do not have correct answers, but rather describe a spectrum of design options actively being sampled by various testbed developers. As a result, FIDC testbeds such as FIRE and GENI are generally heterogeneous federations of collaborating resource providers, providing experimenters with a variety of options. For example, one experiment may choose to emphasize reproducibility, while another seeks exposure to "in the wild" network traffic.

Other challenges do not arise from deep philosophical conundrums, but simply represent the relatively immature state of these testbeds and their associated tools. While most FIDC researchers are successful in conducting experiments they could not run elsewhere, the user experience can be challenging. There is much good work still to be done to provide both novice and experiment researchers with appropriate tools to design and manage their experiments, particularly those that are long-lived or conducted at large scale. Similarly, tools and processes supporting FIDC infrastructure owners' monitoring and maintenance are often relatively basic.

The trend toward heterogeneous federation of research infrastructure is likely to continue, as testbed developers and their researchers seek greater

scale, flexibility, and variety by joining together in global federations. Figure 10 highlights (in blue) countries and regions worldwide with active FIDC testbed communities. While this trend is exciting, it also creates new challenges in federated policy management and enforcement.

Perhaps the leading example of an emerging global FIDC federation is evident from a collaboration that began in the summer of 2013. Participating testbeds are jointly developing a federation compatibility interface for "clearinghouse" functions, such as user credentialing and project membership, and adopting the GENI "aggregate manager" API for resource discovery and provisioning functions. Federation testing and demonstration began in late 2013, with initial participants including GENI in the U.S., FIRE in the E.U., FIBRE in Brazil and Europe, VNode in Japan, and NICTA in Australia. A monitoring capability, developed at iMinds, continually assesses and reports the status and availability of federation resources, for use by participating experimenters.

FIDC testbeds are gaining global traction, with support from a growing international community. For a variety of practical reasons, testbed developers are increasingly turning to federated designs as a strategy to achieve scale while controlling cost and administrative effort. Federated testbeds rely on a web of trust relationships, uniting the key testbed stakeholders: infrastructure providers, testbed developers, and research/educational users. Several national-scale FIDC testbeds are employing a federation approach to their development and deployment. In addition, a number of these national testbeds are entering into collaborative efforts to create a worldwide, federated infrastructure that facilitates transcontinental research.

Acknowledgments

GENI and CASA are funded by the NSF under cooperative agreements CNS-0737890 and EEC-0313747, respectively. Authors from Columbia, University of Massachusetts, and Rutgers acknowledge support under NSF grants CNS-0831912, CNS-1238485, and CNS-1040735 and CNS-1345295, respectively. Authors Zink and Krishnappa acknowledge support from the GENI project office. Any opinions, findings, conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect the views of the NSF.

Part of this work was carried out with the support of the Fed4FIRE and SPARC projects, funded by the European Commission through the 7th ICT Framework Programme. It does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.

References

1. Anderson, D.G. et al. Accountable Internet Protocol (AIP). *ACM SIGCOMM Computer Communication Review 38*, 4 (2008), 339–350.
2. Berman, M. et al. GENI: A federated testbed for innovative network experiments. *Computer Networks 61* (2014), 5–23.
3. Bronzino, F., Nagaraja, K., Seskar, I. and Raychaudhuri, D. Network service abstractions for a mobility-centric future internet architecture. In *Proceedings of the International Workshop on Mobility in the Evolving Internet Architecture*, (2013), 5–10.
4. Fdida, S. et al. FIRE Roadmap Report 1 – Part II. *Future Internet Research and Experimentation* (FIRE), 2011.
5. Femminella, M., Francescangeli, R., Reali, G., Lee, J.W. and Schulzrinne, H. An enabling platform for autonomic management of the future internet. *IEEE Network 25*, 6 (2011), 24–32.
6. Fu, X., Schulzrinne, H., Bader, A., Hogrefe, D., Kappler, C. and Karagiannis, G. et al. NSIS: A new extensible IP signaling protocol suite. *IEEE Communications 43*, 10 (2005), 133–141.
7. GENI design principles. *Computer 39*, 9 (Sept. 2009), 102–105.
8. iLab.t at iMinds; *http://www.iminds.be/en/develop-test/ilab-t*.
9. Krishnappa, D.K., Lyons, E., Irwin, D. and Zink, M. Network capabilities of cloud services for a real time scientific application. In *Proceedings of the 2012 IEEE 37th Conference on Local Computer Networks*, (2012), 487–495.
10. Krishnappa, D., Irwin, D., Lyons, E. and Zink, M. CloudCast: Cloud computing for short-term weather forecasts. *Computing in Science & Engineering 15*, 4 (2013), 30–37.
11. Lee, J.W. Towards a Common System Architecture for Dynamically Deploying Network Services in Routers and End Hosts.
12. Li, M., Agrawal, D., Ganesan, D., Venkataramani, A. and Agrawal, H. Block-switched networks: A new paradigm for wireless transport. In *Proceedings of Symposium on Networked Systems Design and Implementation 9*, (2009), 423–436.
13. Liebeherr, J. and El Zarki, M. *Mastering Networks: An Internet Lab Manual.* Addison-Wesley, Reading, PA, 2003.
14. Maccherani, E., Femminella, M., Lee, J., Francescangeli, R., Janak, J., Reali, G., et al. Extending the NetServ autonomic management capabilities using OpenFlow. *Network Operations and Management Symposium* (2012), 582–585.
15. Marquez-Barja, J.M., Jourjon, G., Mikroyannidis, A., Tranoris, C., Domingue, J. and DaSilva, L.A. FORGE: Enhancing elearning and research in ICT through remote experimentation. In *Proceedings of the IEEE Global Engineering Education Conference*. Istanbul, Turkey, 2014.
16. McLaughlin, D., Pepyne, D., Chandrasekar, V., Brenda Philips, J. K., Zink, M., Droegemeier, K., et al. Short wavelength technology and the potential for distributed networks of small radar system. *Bulletin of the American Meteorological Society 90*, 12 (2009), 1797-1817.
17. Nelson, S., Bhanage, G., & Raychaudhuri, D. (2011). GSTAR: Generalized storage-aware routing for MobilityFirst in the future mobile Internet. *Proceedings of the 6th ACM International Workshop on Mobility in the Evolving Internet Architecture (MobiArch)* ACM, New York, 19–24.
18. OpenFlow Ericsson implementation; https://github.com/TrafficLab/of11softswitch .
19. OpenFlow Specification 1.1; http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf.
20. Raychaudhuri, D., Nagaraja, K. and Venkataramani, A. MobilityFirst: A robust and trustworthy mobility-centric architecture for the future Internet. *ACM Mobile Computing and Communications Review 16*, 3 (2012), 2–13.
21. Sharma, S., Staessens, D., Colle, D., Pickavet, M. and Demeester, P. OpenFlow: Meeting carrier-grade recovery requirements. *Computer Communications 36*, 6 (2013), 656–665.
22. Somani, N., Chanda, A., Nelson, S.C. and Raychaudhuri, D. Storage aware routing protocol for robust and efficient services in the future mobile Internet. In *Proceedings of the IEEE International Communications Conference, FutureNet V Workshop* (2012).
23. Staelens, N., Deschrijver, D., Vladislavleva, E., Vermeulen, B., Dhaene, T. and Demeester, P. Constructing a no-reference H.264/AVC bitstream-based video quality metric using genetic programming-based symbolic regression. *IEEE Transactions on Circuits and Systems for Video Technology 23*, 8 (2013), 1322–1333.
24. Turner, J. and Taylor, D. Diversifying the Internet. In *Proceedings of the Global Telecommunications Conference* (2005). IEEE.
25. Vandenberghe, W., Vermeulen, B., Demeester, P., Willner, A., Papavassiliou, S. and Gavras, A. et al. Architecture for the heterogeneous federation of future Internet experimentation facilities. In *Proceedings of the Future Network and Mobile Summit 2013 Conference*.
26. Vasseur, J.P., Pickavet, M. and Demeester, P. *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS.* Morgan Kaufmann, 2004.
27. Venkataramani, A., Sharma, A., Tie, X., Uppal, H., Westbrook, D. and Kurose, J. et al. Design requirements of a global name service for a mobility-centric, trustworthy internetwork. *Communication Systems and Networks* (2013) 1–9.
28. Vu, T., Baid, A., Zhang, Y., Nguyen, T.D., Fukuyama, J., Martin, R.P. et al. DMap: A shared hosting scheme for dynamic identifier to locator mappings in the global internet. *Distributed Computing Systems* (2012), 698–707.

Mark Berman (mberman@acm.org) is Vice President for Technology Development and GENI Project Director at Raytheon BBN Technologies, Cambridge, MA.

Piet Demeester (piet.demeester@intec.ugent.be) is a professor at Ghent University–iMinds, Gent, Belgium.

Jae Woo Lee (jae@cs.columbia.edu) is a lecturer in computer science at Columbia University, New York, NY.

Kiran Nagaraja (kiran.nagaraja@ericsson.com) is a senior researcher at Ericsson, Inc. This work was done while he was a senior research associate at WINLAB, Rutgers University, New Brunswick, NJ.

Michael Zink (zink@ecs.umass.edu) is an assistant professor at the University of Massachusetts, Amherst, MA.

Didier Colle (Didier.Colle@UGent.be) is a professor at Ghent University–iMinds, Gent, Belgium.

Dilip Kumar Krishnappa (dilipkuk@akamai.com) is a senior performance engineer, Akami Technologies. This work was done while he was a student at the University of Massachusetts, Amherst, MA.

Dipankar Raychaudhuri (ray@winlab.rutgers.edu) is Distinguished Professr of Electrical and Computer Engineering and Director of WINLAB at Rutgers University, New Brunswick, NJ.

Henning Schulzrinne (hgs@cs.columbia.edu) is Julian Clarence Levi Professor of Computer Science at Columbia University, New York, NY.

Ivan Seskar (seskar@winlab.rutgers.edu) is associate director of WINLAB at Rutgers University, New Brunswick, NJ.

Sachin Sharma (sachin.sharma@intec.ugent.be) is a Ph.D. student at Ghent University–iMinds, Gent, Belgium.