NAMED-OBJECT BASED SERVICES IN THE FUTURE INTERNET ARCHITECTURE

by

FRANCESCO BRONZINO

A Dissertation submitted to the

Graduate School–New Brunswick

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Dipankar Raychaudhuri

And approved by

New Brunswick, New Jersey

January, 2017

ABSTRACT OF THE DISSERTATION

Named-Object Based Services in the Future Internet Architecture

By Francesco Bronzino

Dissertation Director:
Dipankar Raychaudhuri

This thesis presents the results of a study focused around the design and development of networking techniques aimed at the deployment and support of advanced services in the future Internet. After many years of constant evolution, the Internet has approached a historic inflection point where mobile platforms, applications and services are poised to replace the fixed-host/server model that has dominated the Internet since its inception. Driven by the strikingly different Internet population of mobile devices and services, new fundamental communication abstractions are required and the current IP based Internet fails to meet their requirement in a satisfying fashion. A top-down analysis of the requirements of such future mobile Internet services is provided, motivating a comprehensive set of solutions needed to meet them. Moreover, starting from the recognition that new core technologies will be a core enabling factor of the previously described evolution, driven by advances such as increased computing power and storage, as well as the trend towards software-based programmability and virtualization. This thesis not only aims to describe why such solutions are required, but also develops a bottom-up analysis of how these new technological advances could be employed to address the new requirements.

The first chapter of the thesis introduces the reader to the fundamental issues at stake, discussing the central architectural concept of *Named-Object* based networking and the power that lies behind it. Looking at the different architectures presented over the years, a set of fundamental abstractions are defined, providing a comprehensive analysis of their properties and how they could be met. This study leads to the presentation of the MobilityFirst architecture in which the "narrow waist" of the protocol stack is based on Named-Objects which enable a broad range of capabilities in the network. This is followed up with a specific set of network service APIs that provide full access to the proposed abstractions supported by MobilityFirst. Using performance benchmarks and the implementation of representative use cases it is shown that the abstractions enabled by the new API are flexible and can enable efficient and robust versions of present and future applications.

The second chapter of the thesis then moves to the set of services that will be required by the future mobile Internet and that due to different shortcomings are hardly supported by the current TCP/IP Internet architecture. These include: i) Multicast services, ii) Content services, iii) In-network compute, and finally iv) Context services. For each of these services, appropriate abstractions enabled by the Named-Object architecture are presented and a use case based prototype evaluation is provided. The results show the feasibility of providing a broad range of services with good performance and reasonable protocol overhead.

Starting from the above abstractions analysis and the newly introduced services, the third chapter of the work, focuses on how such new services are made available to the end-users of the network. Considering first the expected requirements for such systems, a new transport layer service is presented. The new designed protocol can seamlessly support a set of distinctive features based on use of names and in-network reliability techniques. Using the developed prototype components, experimental results show that for a few representative scenarios including mobile data delivery, web content retrieval, and disconnected/late binding service, the new systems can be exploited to reduce the impact of complex operations improving performance for the end users of the network.

The fourth chapter analyzes how advanced cloud services can be supported in the proposed Named-Object architecture. In particular, the concept of naming is extended to natively support virtual network identifiers. It is shown that the virtual network capability can be designed by introducing the concept of a “Virtual Network Identifier (VNID)” which is managed as a Named-Object. Further, the design supports the concept of Application Specific Routing (ASR) which enables network routing decisions to be made with awareness of application parameters such as cloud server workload. Experimental results show that the new framework provides a clean and simple logic for defining and managing virtual networks while limiting the performance impact produced by the additional overhead generated by running such system. Moreover, the potential of ASR is demonstrated through a based cloud service use case deployment.

The last chapter of the thesis aims to bring together the whole study and provide considerations on how the different components presented could be merged into a single end-to-end realization. The designed elements are used in combination to present an overview of how they could all be joined into a single experimental platform ready to be employed in various deployment scenarios. Specific prototyping details are given for several scenarios including advanced computing and context-aware services and how these have been deployed on a nation wide testbed.

Acknowledgements

Acknowledging people has always been an art I have failed miserably at. I have always been better at saying awkward "thank yous" than coming up with heartfelt speeches. But after more than four years of Ph.D. studies, this thesis could not be complete without mentioning a few people that made this work possible, either by guiding me throughout my work or by supporting me even during the worst moments. I will be brief and I am sure I am going to forget somebody. I am deeply sorry about it, but as always I am late in writing this.

First, I would like to thank the members of my defense and proposal committees, Professor Gruteser, Dr. Lakshman, Professor Trappe, and Professor Yates, for their advice and suggestions regarding this thesis. Having them approve the merits of my work is a personal and gratifying statement that these years were not wasted and hopefully produced some good work. And if I have to mention people that helped me shape my work to this form, I could not forgo thanking all my collaborators, starting from my colleague Kai, and finishing with all the faculty members I had the pleasure to work with, including Dr. Nagaraja and Dr. Nakauchi, among others.

These acknowledgements could never end without mentioning Ivan Seskar, who after four years must have grown tired of the sound of my fingers knocking at his door. But question after question, he never stopped answering me, guiding me through all the hours I spent working on ORBIT and GENI. You have not obtained a proper Ph.D. degree from WINLAB if you have not been debugging your ORBIT experiment with Ivan at 3 AM in the morning.

A special mention has to go to my advisor, Professor Dipankar Raychaudhuri. I could talk at length about how I grew as a researcher under his guidance; about how the rest of my career will be influenced by everything he has taught me. But there is one thing that Ray transmitted me above everything else: listen and understand the person that is in front of you. Ray's office was always open for me. I tried my best as a student, in every moment of my carrer, but without his guidance and his understanding I would probably not be writing these words. I feel this is the most important lesson an advisor could ever pass on to his students. If I will ever have my own students I will try my best to do the same.

Among the people that supported me, I will never be more thankful than in this moment to my family for being who they are. They followed me across the ocean, sometimes having no clue of what was happening, what I was working on; sometimes they had to deal with my moody interactions; but they never stopped believing in my abilities; they never stopped supporting me; I was always comforted by the idea that whatever happened they would have been there for me. They even watched the entirety of my thesis defense streamed live over the Internet, this is love.

Finally, talking about support, I would be a fool if I did not conclude by thanking the closest friends that have accompanied me over these years - Shreyasee, Shubham, Dragoslav, and Javier. This time would not have been the same without you. You have seen me go through all the up and downs. You have been there for the moments of joy and the moments of desperation. You have listened me all the times I thought I could not handle it anymore. I will never forget all this. I hope our paths cross for all the years to come.

Dedication

A tutti coloro che mi hanno strappato un sorriso nei momenti piú bui

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

The Internet today is very different from its original concept when the architecture and protocols were developed around the abstraction of communications between fixed end hosts. Growing levels of mobility characterize today's communications; mobile wireless devices have outnumbered fixed end hosts and even service end-points have different levels of mobility (not an uncommon scenario in data centers). Even though support for seamless mobility is a growing requirement for the Internet as a whole, past proposals and current solutions are either only applicable within limited environments (e.g., cellular [1]) or are inefficient when applied to the Internet (e.g., MobileIP [2]). A few recent scalable approaches to support mobility have been proposed, but these are not standalone and require changes to the routing plane and/or protocol stack defined in TCP/IP [3–6]. A few solutions may be applied by patching current systems, but benefits under fine-grained mobility are still unclear [7, 8].

A second serious shortcoming of the Internet is its host-centricity. Principals such content, services, and context have over the years gained at least as much importance as hosts. However, since they are not first-class network objects, they are not directly addressable. Direct addressability and location-independence, would enable seamless content/service mobility, and help with building efficient delivery networks without resorting to DNS-based tricks used by present-day CDNs. Direct addressability is also important for services offered by the core and edge of the network, not just end-points. While the clear boundaries of network pipes and compute end-points that were fundamental to end-to-end transport protocols (e.g., TCP) are starting to fade, CDNs and hosting platforms are starting to place storage and compute clouds closer to consumers [9] and this strategy is increasingly being co-opted by ISPs [10] eager to provide value-add services. Direct addressability for these resources and services would benefit both deployment and access strategies.

A third important consideration for the Internet is the need for supporting efficient one-to-many communication mechanisms. This need is foremost for the emerging context (e.g., location-aware messaging), collaborative (e.g., multi-player on-line games) and crowd-sourcing applications. The related aspect of device-level multihoming where devices may simultaneously attach to two or more networks for performance or robust connectivity will also benefit from native support for multi-point communication [3].

Finally, along with the exponential growth in size and economic importance of the Internet, the scale of security threats has grown too. Establishing network trustworthiness and avoiding spoofing and identity hijacking incidents are priorities today that are not completely met by patched solutions such as IPSEC or DNSSEC, mostly due to their partial deployment or adoption. A clean approach that ensures uniform deployment at a basic level for all network principals such as the PKI-based self-certified identifier proposal is a potential solution [11].

1.1 Evolving Networking Scenarios in the Future Internet

This predictable, yet fundamental, shift presents a unique opportunity to re-think how Internet and mobile networks work and to develop new technologies that could be the enabling factors of a new wave of advanced network services. Within this environment, this work focuses on understanding the mechanics behind this evolution and to design and develop new solutions to create better platforms for enhancing the potential of the new services that will come out in the future. By looking at how to improve single components of the protocol stack, but never forgetting the potential impact on the network as a whole, this study explores ways to simplify and improve the deployment of these scenarios through network assistance. In doing so, a particular attention is given to the identification of requirements and needed abstractions of applicable use cases.

In order to approach this problem a first step is necessary: define a set of basic network service abstractions that can support the needs of the future mobility-centric Internet applications. The identified abstractions are: name-based services, direct addressability for content and services, trusted identities, multi-point addressability and in-network hosting of storage and compute services. Starting from these abstractions new base and advanced services can be designed.

1.2 Using New Technologies for Quick Iteration

New technologies will be the key enabling factor of the previously described evolution, driven by advances such as increased computing power and storage, as well as the trend towards software-based programmability and virtualization. These advancements have stimulated the creation of new flexible and adaptable frameworks for networking platforms that allow researchers and industry players to advance the state of the art. An example of such technologies are:

- Network Virtualization and Software Defined Networks in Internet networks.
- Integration of computing and storage in the network routers.
- Software Defined Radios for access networks.

In this work, technologies such as those listed above will be used to create flexible platforms for the networking concepts under consideration. Further, the solutions developed will take advantage of the recent shift towards software defined routing and switching platforms for realizing new network functions.

At least as importantly toward the experimentation of our designs, a great attention will be given to the utilized evaluation environments. Our validation of the design of new tools and architectures spaces across different experimental techniques. This is due to the different requirements and goals that are part of the process. For performance verification at scale, simulated environments, such as NS3, have been employed. Moving toward prototype experimentation, tradeoffs with the scale of the experiments and their levels of realism will be analyzed. Testbeds of different scale and capabilities will then be employed, depending on the specific needs of the tool under evaluation.

1.3 Organization of the Thesis

This thesis starts with a top-down analysis on how an Internet architecture could evolve in order to support a variety of advanced services, and then provides specific network architecture and protocol solutions to realize the vision.

The first chapter of the thesis introduces the reader to the fundamental issues at stake, discussing the central architectural concept of *Named-Object* based networking and the power that lies behind it. Looking at the different architectures presented over the years, a set of fundamental abstractions are defined, providing a comprehensive analysis of their properties and how they could be met. This study leads to the presentation of the MobilityFirst architecture in which the "narrow waist" of the protocol stack is based on Named-Objects which enable a broad range of capabilities in the network. This is followed up with a specific set of network service APIs that provide full access to the proposed abstractions supported by MobilityFirst. Using performance benchmarks and the implementation of representative use cases it is shown that the abstractions enabled by the new API are flexible and can enable efficient and robust versions of present and future applications.

The second chapter of the thesis then moves to the set of services that will be required by the future mobile Internet and that due to different shortcomings are hardly supported by the current TCP/IP Internet architecture. These include: i) Multicast services, ii) Content services, iii) In-network compute, and finally iv) Context services. For each of these services, appropriate abstractions enabled by the Named-Object architecture are presented and a use case based prototype evaluation is provided. The results show the feasibility of providing a broad range of services with good performance and reasonable protocol overhead.

Starting from the above abstractions analysis and the newly introduced services, the third chapter of the work, focuses on how such new services are made available to the end-users of the network. Considering first the expected requirements for such systems, a new transport layer service is presented. The new designed protocol can seamlessly support a set of distinctive features based on use of names and in-network reliability techniques. Using the developed prototype components, experimental results show that for a few representative scenarios including mobile data delivery, web content retrieval, and disconnected/late binding service, the new systems can be exploited to reduce the impact of complex operations improving performance for the end users of the network.

The fourth chapter analyzes how advanced cloud services can be supported in the proposed Named-Object architecture. In particular, the concept of naming is extended to natively support virtual network identifiers. It is shown that the virtual network capability can be designed by introducing the concept of a "Virtual Network Identifier (VNID)" which is managed as a Named-Object. Further, the design supports the concept of Application Specific Routing (ASR) which enables network routing decisions to be made with awareness of application parameters such as cloud server workload. Experimental results show that the new framework provides a clean and simple logic for defining and managing virtual networks while limiting the performance impact produced by the additional overhead generated by running such system. Moreover, the potential of ASR is demonstrated through a based cloud service use case deployment.

The last chapter of the thesis aims to bring together the whole study and provide considerations on how the different components presented could be merged into a single end-to-end realization. The designed elements are used in combination to present an overview of how they could all be joined into a single experimental platform ready to be employed in various deployment scenarios. Specific prototyping details are given for several scenarios including advanced computing and context-aware services and how these have been deployed on a nation wide testbed.

Chapter 2

Name Based Architectures and Service Abstractions

The basic abstraction for communications over the Internet has always been very simple: *send* a message from an interface to a destination address and if an interface is listening for data on that address it will *receive* the message at the other end. From an end-point perspective, this simple concept creates the perception of transmitting data on top of a virtual link connecting the two interfaces. This is the base of the Berkeley IP socket layer, the most commonly used network interface. But after many years of using this simple IP services model, the Internet is now approaching a historic inflection point where mobile platforms, applications and services are poised to replace the fixed-host/server model that has dominated the Internet since its inception. While developers have still managed to create a rich ecosystem of services above the IP network layer, the reliance on this core interface forced them to develop ad-hoc and sometimes patched solutions to address even the most common service scenarios.

As the majority of end-points shifted to dynamic and mobile approaches, applications had to deal with the risk of potential communication failures. This is not only limited to wireless devices, that are now outnumbering the fixed end-hosts [12], but it also includes service end-points that are characterized by different levels of mobility - e.g. cloud migration. Even though support for seamless mobility is a growing requirement for the Internet as a whole, past proposals and current solutions are either only applicable within limited environments (e.g., cellular [1]) or are inefficient when applied to the Internet (e.g., MobileIP [2]). One-to-many communication applications have also grown in importance even if no widely available delivery mechanism, e.g. multicast, has been adopted in the Internet. Everyday example applications like multi-player on-line games and social networks come to mind. Even considering their popularity, they always had to rely on multi point-to-point connections to deploy such services, forcing huge overhead on the network due to excessive traffic and service development time. Moreover, as technology evolves, devices are beginning to incorporate multiple interfaces technologies such as WiFi and 4G/LTE. But even for common scenarios, e.g. phones with multiple wireless interfaces, it remains difficult to exploit this potential due to limitations in the underlying network protocols. Even for basic scenarios like VoIP calling no common solution has been found to avoid calls drop while transitioning across interfaces. Only exceptions for small and severely limited scenarios have been deployed to take advantage of this [13].

Acknowledging this markedly different Internet population of mobile devices and services, the research community has looked over the years at the possibility of defining new communication abstractions to address the limitations of the IP/TCP as it exists today. As an evolutionary step, the Host Identity Protocol (HIP) [3], proposed to introduce a naming layer through the use of a shim layer sitting between classical transport protocols - i.e. UDP/TCP - and the IP network layer. With the goal of exclusively requiring modifications in the end host network stack, HIP provides new tools and functions for future network needs, from supporting seamless host mobility and multi-homing, to the ability to securely identify previously unknown hosts and the ability to securely delegate signalling rights between hosts and from hosts to other nodes. In order to do so, HIP introduces a new name space made of Host Identifiers, that double as public cryptographic keys. Similar in spirit, SERVAL [5] implements a new Service Access Layer (SAL) that sits above an unmodified network layer, enabling applications to communicate directly on service names aimed at supporting Internet services located at multiple and different locations, while serving clients that are often mobile and multi-homed.

Using a different namespace to separate names from the addresses of the routing layer through the use of a name to address mapping service has also been advocated as a structural change within the network infrastructure itself [4, 14]. For example, the Locator/Identifier Separation Protocol (LISP) [4] proposed this separation as the decoupling in two types of addresses: EIDs that identify hosts, and RLOCs that identify network attachment points that are used as routing locators, i.e. are IP addresses. LISP, with such separation in place and the presence of the name mapping system, is able to offer native mobility through an extension of its protocol called LISP-MN.

A third clean slate approach is that of content centric networks [15] or named data networks [16], both belonging to the class of information centric networks (ICN). These architectures depart from the conventional point-to-point abstraction by having routers in the network directly operate on content labels making physical network addresses unnecessary. The Data-Oriented Network Architecture (DONA) [17], was perhaps the first comprehensive and detailed proposed architecture that relied on the use of self-certifying names. Content Centric Networking (CCN) [15] and its derivative designs [16, 18] evolved the concept and brought it to prominence within the networking community. The CCN architecture through an elegant mechanism shifts the networking paradigm from today's IP locators - *where* - to content descriptors - *what.* This paradigm shift not only enables efficient delivery of content, but also enables advanced services such as mobility and multi-homing which are relatively difficult to support in todays IP networks.

While all these different architectures have advantages and disadvantages, before looking at potential fundamental solutions, we believe it is important to make a step back and analyze how to define what abstractions a future Internet architecture should support.

2.1 Service Abstractions for the Future Internet

We think that a future Internet should support the set of service abstractions given below. The implication of each on the network architecture (NetArch) design is presented alongside.

A. Name-based Services. Communication with a mobile end-point should be no different than that with a fixed end-point. The current "practical" approach results in undesirable asymmetry, where mobile end-points are always responsible for re-establishing connections. The situation is doubly vexing when both end-points are mobile. A basic service abstraction that allows addressing a network end-point by its unique name and not its current location will establish a uniform approach to dealing with fixed and mobile end-points alike, enabling seamless mobility.

NetArch implications: To support a name-based network service that seamlessly handles mobility, the network requires native support for *dynamic and fine-grain location resolution.* Some have proposed protocol interposition approaches that dynamically substitute local addresses for end-points with dynamically resolved ones [3, 5]. Efficiency extensions in MobileIPv6 attempt to signal end-points with address updates [19] to avoid triangular routing through fixed home agents. However, we think that besides end-hosts, the network routing fabric must also be able to dynamically resolve and re-bind in-flight packets.

B. Direct Addressability for All Network Principals. Host-centric abstractions to network services were a solid building block during the conception and early advancement of the Internet. Though, other important principals have emerged since then. While content and services are two established principals (besides hosts), others such as sensors and actuators, as also more abstract ones such as context are quickly gaining traction. Since few foresaw today's usage of the Internet with any accuracy, allowing for a broader definition should be the path forward. In that vein, direct addressability for all principals eliminates any unnecessary bindings of one principal with another. For instance, content should be addressable both directly and independently of where it may be located physically.

NetArch implications: Allowing for direct addressability for not only content and services, but also other emerging first class entities requires that the *name space be practically inexhaustible.* For instance, 256 bits to encode the name would last us for a long time to come. To put it in perspective, 2^{270} is the ballpark for number of atoms in the observable universe. A larger namespace implies engineering challenges to implementing network elements - whenever forwarding engines inspect, lookup or classify on names - and scalability of network support services such as a name-to-address resolution service.

C. Trusted Identities. Stronger security and network trust is possible if self-certifying names were used for addressing principals, as shown in AIP [11]. Present-day approaches to authenticate and establishing trust is based on principals that maintain trust credentials (e.g., a PKI certificate) separate from their network identity (an IP address), and where the credential establishes the linkage of principal's identity to the network identifier. The linkage itself is certified by a mutually trusted entity (e.g., a certificate authority). The adoption today is far from pervasive and particularly challenging for mobile entities where the conflated network identifier may change. Using a public-key as the self-certifying identifier to address principals, can ensure both location independence and greater trust by preventing hijacking and spoofing problems.
NetArch implications: Embedding trust within the network names requires names to be longer and also to be flat. When using a PKI public key of a reasonable strength X, would require a minimum of 2X bits (= 256 bits for the current recommended 128-bit strength) as when using Elliptic Curve Cryptography. Also, since names form the basis of all network services, there must exist organizations or other easily accessible mechanisms by which names are produced and assigned to network principals in a reasonable manner. Finally, flat names imply they cannot be aggregated as is for IP addresses, creating engineering challenges when routing/forwarding operations need be performed on names.

D. Multi-Point Addressability. With group-based subscriptions (e.g., RSS, over-the-top video "broadcasts") and collaborative applications (e.g., teleconferencing, gaming) routine today, need for multi-point addressing is basic. Under this are multicast, multihoming, and also anycast, a 'one-of-many' abstraction important for a variety of reliability and load-sharing uses. Today's host-centric Internet essentially supports a point-to-point abstraction. IP-multicast is really an extension that's enabled by special interpretation of a small subset of destination addresses. Concerns of scalability means that multicast is commonly left disabled on network elements. Internet applications therefore regularly resort to multiple unicasts to address groups. In wireless environments, there is a desire to take advantage of inherent multicast/broadcast medium to enable efficient point to multi-point delivery services. Multi-homing, where entities have two or more active network attachments, deserves similar support with a few independent considerations that allow flexibility on how each attachment can be used separately or collectively for performance, reliability or other metrics.

NetArch implications: First, the network must have support for creating and managing groups of member identities. These must be dynamically available to the routing fabric dynamically. Be it as receiver driven multicast trees or some other means, a basic consideration would be limit per group state within network elements to minimize set-up procedures for easier and sustained adoption. Since names are flat and uniform, the service abstraction must provide a means for end-hosts to specify the requested delivery service - multicast, anycast, multihoming, broadcast, etc. - that's interpreted by the network elements. While network could participate in path choice and scheduling for multihoming, transport implementations may also be conceived at the end-host [20].

E. En-Route Storage and Compute. The traditional notion of keeping all data processing at end-points with the network as pipes with routing alone is beginning to dissolve. For instance, non-ISPs such as Akamai have independently placed compute/storage resources at the edges of the network to reduce access latency. It is not inconceivable, therefore, for ISPs to open up PoPs for placement of storage and compute services as well [10]. Routers in the request path may service content requests, or co-located compute resources may shoulder mobile offloaded security functions such as data encryption. In-network and en-route compute opportunity may hold particular appeal for mobile devices that are often on a limited resource budget [21] and may require non-trivial customization of data delivered to them.

NetArch implications: While certain services may be embedded transparently into the network [22] or be co-located without requiring a tight coupling with the routing fabric, network architectures with a extensible data plane in the form of a pluggable computer plane, would benefit from flexible service extensibility in the future.

2.2 Named-Objects

While all the solutions presented in the introduction section of this chapter were driven by one or more use cases, none of them has reached enough maturity to fully replace the original virtual link interface. This motivated us to consider the design of a more compelling abstraction which provides the benefits of ICN techniques while maintaining a higher degree of generality for service creation. We call this abstraction *Named-Objects*. At the base of our solution lies a fundamental rethinking of how hosts, devices and network elements in general are identified and communicate in the network. In contrast to the current IP based network, which tends to conflate both names and addresses, we create a new level of separation: names are flat globally unique identifiers (GUIDs) that are large enough to create a name space practically inexhaustible. The location of these objects is then resolved through a Name Resolution Service (NRS) through a dedicated API. While this idea is not completely new and relies on previous work on name/address separation [3, 4], we take it further down the line, proposing support for the so called "Named-Objects": Named-Objects are a new abstraction meant to represent any network entity that could be abstracted as an addressable network element. This should cover any possible abstraction: from the original host based abstraction of a virtual link bridging two interfaces, to recently introduced ones such as contents, to any potential future abstraction - e.g. context. While name based approaches have already been addressed in the past, they were mostly focused on either solving specific issues such as mobility [4] or security [3] or to shift the communication focus to new entities such as contents [15, 17]. Our target is a more comprehensive solution that can enable powerful abstractions and services to underpin the Internet architecture.

Figure 2.1 outlines our approach to defining the Named-Object abstraction through separation of names and addresses. Separating names (identities) from addresses has been advocated by the research community [3, 4] for quite some time and has inherent benefits in handling mobility and dynamism for one-to-one communications. If properly employed, names can also provide additional advantages to facilitate the creation

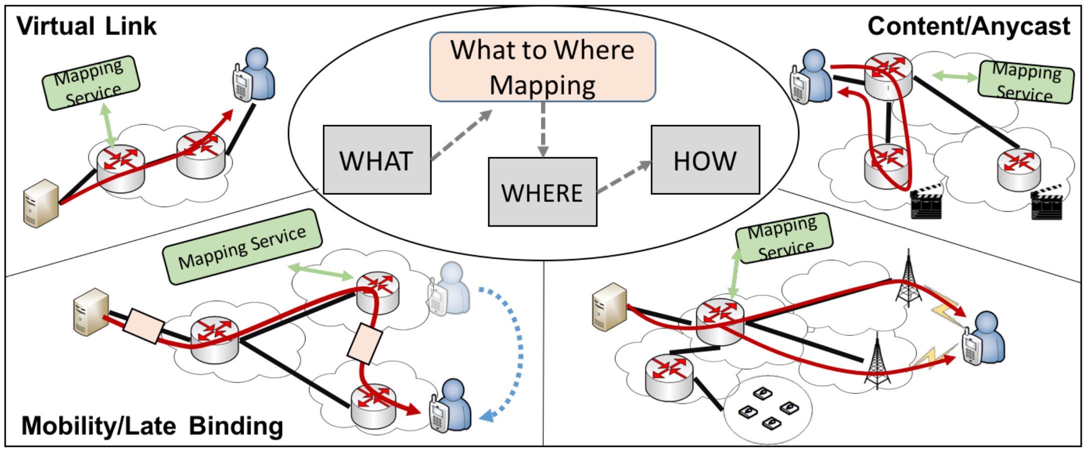


Figure 2.1: The Named-Object abstraction applied to core use cases.

of new service abstractions that can be used to support advanced applications. Our approach involves three steps: 1) First, *"what"* (or *"who"*) will take part in the communication has to be identified through a unique name that is understandable by all parties involved, e.g. end points, routing elements. 2) When forwarding is required, names are then resolved to *"where"* they are located. While different techniques can be employed for this purpose, previous proposals [4, 14, 23, 24] demonstrated that the use of a globally accessible Name Resolution Service is a suitable approach for this goal. 3) Finally, if the semantical value of such element is known it can be indicated through the use of a service identifier properly located in a packet header, giving an indication of *"how"* such packet should be treated.

Once we move away from the host-centric nature of the IP world, new and different delivery services can be supported, where routing decisions can be diversified based on the nature of the referenced object. In order to do so, network elements have to support hybrid routing schemes, where a service ID (SID in the Figure) can be located in the networking header to identify the required service. As we will show with the proposed solutions, we believe that this concept, together with appropriate support in the different network entities, can be the core at the center of the evolution of sustainable communication techniques for future networking scenarios. In particular we address how the name object solution can address the base abstractions presented.

Handling Mobility. Assigning permanent names to mobile objects and decoupling names from addresses has inherent benefits over IP avoiding the need of relying on triangular routing to solve the hierarchical nature of Mobile-IP routing schemes and support host mobility [2]. As end-points talk to each other through names, routers can map these names to current locators of the devices and route packets to them based on the locators avoiding the need to route packets to the previous location persistent element (the home agent). End-points are solely responsible for updating the resolution system with their current locator. Each packet is then routed to the most up-do-date location, as any other device can query the same system to obtain the current locator. When occasional inflight packets reach the previous location of a moving device, routing components can notice that this has moved and perform a new query to obtain the new location. We call this technique “late-binding”.

Group Based Delivery Devices can often be grouped into sets of devices that require to efficiently communicate to some or all of them at the same time. Consider for example IoT based messaging scenarios: a typical query involves sending short messages to hundreds or thousands of users or application agents, so that scalability becomes an issue, as multiple unicast messages through an overlay service can easily overload the network. Mobility of end-devices results in additional complexity, especially for dynamic environments such as vehicular communications. For example, if a single warning message needs to be pushed to hundreds of cars and pedestrians in a given area, multicast groups would need to be maintained across a large number of access networks in order to efficiently support such one-to-many communication. Using appropriate multicast routing solutions would help improving network efficiency, while reducing the complexity and cost of deploying such applications. However, existing network-layer multicast solutions (e.g., PIM-SM [25], MOSPF [26]) have not been widely adopted due to fundamental problems that are a by-product of the original Internet design geared toward static host-centric communication.

Security. A name or GUID assigned to a network object by one of several name certification services (NCS), is self-certifiable, i.e., end-points claiming a GUID can authenticate each other without the need for third-party certification. When the GUID is

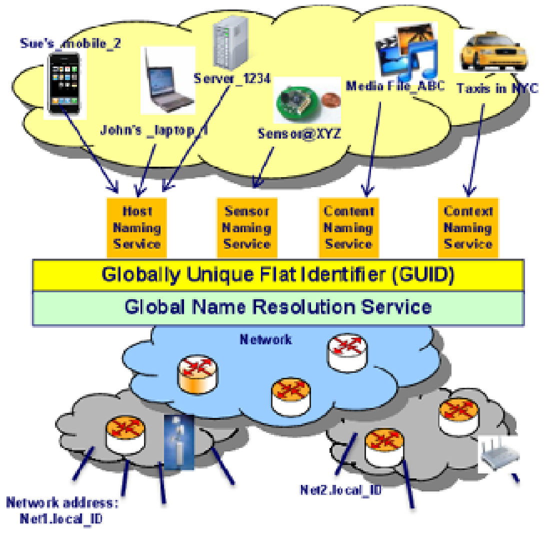


Figure 2.2: MobilityFirst architecture.

derived as a cryptographic hash of the object's public key, the authentication requires a simple, bilateral challenge response procedure to be executed between the communicating end-points. For content, the GUID may optionally be derived as a cryptographic hash of the bits of the content. For full non-repudiation, i.e., to verify origin of content, a signature may accompany the content which could authenticate the principal that originated the content.

Compute Support. We use name-based communications to implement this abstraction. Two core pieces of technology are introduced: first, the ability to aggregate multiple service instances under a single name. This is done by offloading the list of participant locations under a single name into the name resolution service. Second, the ability to make compute nodes participate in the routing protocol by sharing their application state. This could be either implemented through a new interface in the participating routers, requiring though the introduction of new schemes to identify participants of the protocol, or by offloading this information to the name resolutions service.

2.3 A Name Object Based Architecture: MobilityFirst

With the above stated network service abstractions in consideration, we have designed and prototyped a network architecture that addresses the principal goals of supporting at-scale and seamless *mobility*, along with *trustworthiness* in the future Internet. Figure 2.2 shows the main components of the MobilityFirst architecture which centers around the concept of self-certifying, Globally Unique IDentifiers (GUIDs) as names for all network principals. Below we present key details of the architecture that address the requirements outlined in the previous section.

Naming and Dynamic Resolution. At the crux of the MobilityFirst architecture is a new name-based service layer which serves as the 'narrow waist' of the protocol stack. The name-based service layer uses flat GUIDs for all principals or network-attached objects including hosts, content, and services, making each a first-class network object. Unlike IP addresses which conflate identity and location, addresses of objects in MobilityFirst are dynamically resolved using the object's GUID. This resolution is enabled by a globally accessible name resolution service (GNRS), which is used by objects to both announce their latest location/address and lookup end points they wish to communicate with. While a variety of incarnations of the GNRS are possible, we have validated 2 alternate designs that both meet our low resolution latency goals of less than 100ms on average for lookup operations [23, 24].

Trusted Communication. A GUID assigned to a network object by one of several name certification services (NCS), is self-certifiable, i.e., end-points claiming a GUID can authenticate each other without the need for third-party certification. When the GUID is derived as a cryptographic hash of the object's public key, the authentication requires a simple, bilateral challenge response procedure to be executed between the communicating end-points. For content, the GUID may optionally be derived as a cryptographic hash of the bits of the content. For full non-repudiation, i.e., to verify origin of content, a signature may accompany the content which could authenticate the principal that originated the content.

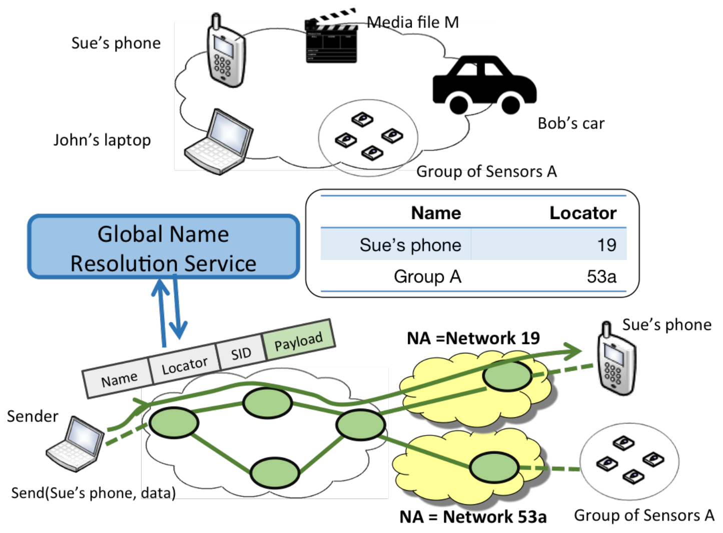


Figure 2.3: Named Object abstraction with clean separation of naming and addressing

Storage-Informed Segmented Transport, Edge-Aware Routing. In contrast to end-to-end transports which perform poorly in wireless conditions [27, 28], MobilityFirst employs a segmented transport to reliably progress data hop-by-hop. Data is segmented into large blocks that are cached at each hop, if storage is available, to enable in-network retransmission under losses. Experiments under a variety of wireless conditions have shown significantly better fairness, throughput, latency and robustness under the hop-by-hop transport, including an order of magnitude gains in median throughput [29, 30]. Within a domain, a generalized storage-aware routing (GSTAR) combines link-state routing with DTN elements, and flexibly expands connectivity across wired and wireless segments, as also occasionally connected partitions [31]. Conditions at the wireless-edge are further heeded by adopting an edge-aware inter-domain routing (EIR) approach that scalably gathers (using *telescoping* or aggregation of updates) and utilizes capacity and load conditions at edge networks to instrument effective multi-path and multi-home delivery.

Extensible In-Network Services. MobilityFirst proposes a network fabric with native support for multi-point and multi-path delivery services, with little to no set up control signaling between network elements and minimal state within them. For example, group memberships for multi-point delivery are maintained within the GNRS and retrieved dynamically by source hosts or en-route network elements to determine forwarding paths. The requested delivery type is specified by end-hosts and encoded as the *service identifier (SID)* field within the routing header, and includes multicast, anycast, multi-path, and multi-home delivery. To support future extensibility of network function, MobilityFirst proposes a pluggable "compute plane" for the network fabric. Compute services traditionally implemented by end-hosts may be plugged into the network at strategic points to provide en-route or local services such as content caching, encryption, VPN, or video transcoding. These instances register their name-address mappings at the GNRS, and require that end-hosts request their invocation by specifying a compute-plane SID and the GUID of the particular service. Furthermore, multiple SIDs may be specified simultaneously within the header invoking any sensible combination of services on a packet.

2.4 A New Network API

As we transition to the named based world defined by the MobilityFirst architecture and with it its novel abstractions, a new network network API has been developed to support advanced name based scenarios [32].

2.4.1 The MobilityFirst Network API

In this section we discuss the specific API we are developing with the goal of supporting the abstractions presented in Section 2.1. Table 2.4.1 shows the complete set of API designed to take advantage of the architecture. The parameters therein are a loose depiction mainly to simplify the presentation. We have divided the operations into three major groups: basic, content-centric and service-centric operations. By doing so we want to be able to take advantage of the inherent characteristics of the communication patterns of these three groups of abstractions.

Basic	**Content Centric**	**Service Centric**
open(profile, [profile-opts], [src-GUID])	get(content-GUID, request, buffer, [svc-opts])	exec(svc-GUID, request, buffer, [svc-opts])
send(dst-GUID, data, [svc-opts])	get_handle(handle, dst-GUID, request)	exec_handle(handle, dst-GUID, request)
recv(src-GUID, buffer, [GUID-set])	get_response(handle, data, [svc-opts])	exec_response(handle, data, [svc-opts])
attach(GUID-set)	post(dst-GUID, data, buffer, [svc-opts])	
detach(GUID-set)	post_handle(handle, dst-GUID, data)	
close()	send_response(handle, response, [svc-opts])	

Table 2.1: MobilityFirst API

Basic Operations

Basic operations are in charge of: create end points with default stack operations customized for applications, support name based message delivery and manage network presence for the set of application GUIDs.

Endpoint/Socket creation and customization: At the beginning of each communication session, an application initializes a MobilityFirst socket by invoking the *open* operation. During this initialization, the application provides the API layer the informations about the profile of the communication that will occur. With the *profile* parameter we allow user to specify the set of elements that characterize the session such as: communication patterns, resources needed and services required. Additional extensions to the profile can be provided through a set of optional profile options. The final parameter is also optional and represents the GUID that the application want to use as its default identity within the network.

Name-based Messaging: Once the session is initialized, *send* and *recv* are used for the exchange of data messages. While the baseline profile is common for the entire session, a per-message characterization is possible through the use of service options. These service options (*svc-opts*) are used to define the set of network services offered by the architecture. This set of features spans from the ability of exploiting the computing layer located at routers, to different delivery systems and security options. Additionally, GUIDs can be used to express intentional data receipt through the use of the optional *GUID-set* parameter in the *recv* operation. Messages from source GUIDs other than those specified are not delivered to the local end-point.

Management of network presence: To modify the set of GUIDs that an application wants to be responsible for *attach* and *detach* operations can be used. Through these operations network reachability for specified GUIDs can be announced if not already established. The utility of these operations can be easily identified in a content delivery scenario where availability of particular content may be in continuous flux (e.g., changing news items) and their availability needs to be updated both locally and remotely on the GNRS.

Content and Service Extensions to the API

As introduced before, GUIDs can represent any network object. While we do not present here details of how human readable names would get translated into GUIDs, we assume that from this transaction the semantic relationship between the nature of the GUID and the network object that it identifies is easily tracked by the programmer. Starting from this principle that additional information about the nature of the GUID could be known a-priori, we extended the API to support the following three specific operations as related to content and service entities: requests for transferring contents from a remote location (*get*), transfer of content to a remote location (*post*) and requests for a service to be invoked at a remote location (*exec*). The use of type specific operations translate into different advantages; on the host side this enables the network stack to select the best transport protocol and allocate in advance the adequate amount of resources; moreover it allows the user to choose between handling content transactions asynchronously or as atomic operations and use per operation security and data delivery options. On the network side this enables the usage of specific header SIDs to provide the network components additional information about the type of data flowing.

All three communication patterns are characterized by a three way transaction represented as: request (i.e., *get, post, exec*), handling of the request by the receiving host(i.e. *xxx_handle*) and final response (i.e., *xxx_response*). *Handle* objects are used to identify specific requests and characterize the responses that follow them. Additional informations for the end points involved in the communication can be passed through the use of the *request* and *response* parameters.

2.4.2 Implementation

To gain some experience with our design, we are developing a proof-of-concept prototype of MobilityFirst's API and end host protocol stack. The API is implemented as a system library and can be interfaced both from C/C++ and from JAVA; the stack implementation takes the form of a standalone, multi-threaded user-level process that uses *pcap* for low-level packet capture and injection. As all components are developed in C/C++ or JAVA with no major library dependencies they can be easily utilized on x86/ARM platforms running Linux or Android.

Micro-benchmarks

To ensure a reasonable implementation of the API and the protocol stack, we ran basic latency and throughput experiments to establish the combined API library and stack performance.

Chunk Size	**MobilityFirst**	**IP**
64B	5.62 ± 3.69	0.55 ± 0.42
256B	4.71 ± 0.75	0.65 ± 0.80
1KB	5.68 ± 0.88	0.81 ± 0.35
4KB	8.93 ± 3.75	3.31 ± 1.00
16KB	20.44 ± 1.78	12.72 ± 1.14

Table 2.2: Latency in ms on 54Mbps WiFi link

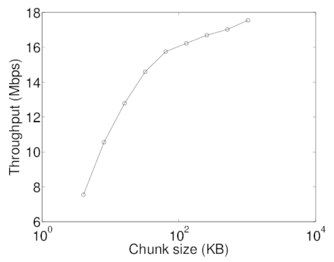


Figure 2.4: Throughput on a 54Mbps WiFi link

In Table 2.2 and Figure 2.4 are shown respectively the round trip times and maximum throughput achieved by two machines with Intel i7 K875 processors and 8GB of memory directly connected through an Intel 54 Mbps Wi-Fi interface. From the table is possible to notice that even though the software implementation generates additional computing overhead, the behavior of the average RTTs with increasing packet size follows the one obtained using ICMP control messages. The graph provides an idea of the difference of performances achieved by transferring a large file dividing it in chunks of different sizes. While the performance is not yet comparable to what we get from using *iperf* with UDP (21 Mbps on average) and TCP (17.1 Mbps on average), the behavior of the graph is consistent with what would be expected from the application of the wireless transport protocol Hop [29] where with additional protocol overhead better performances are achieved with big chunk sizes.

2.4.3 Use Cases

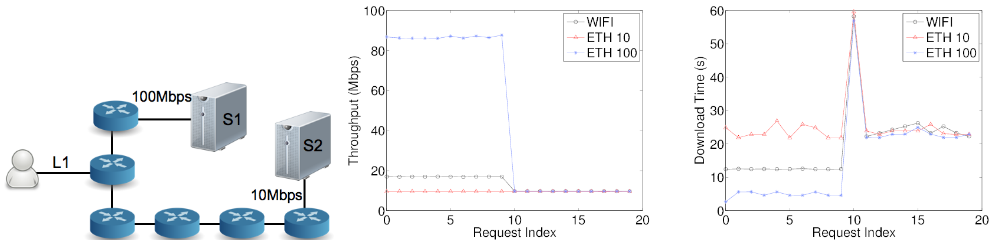


Figure 2.5: Anycast content retrieval experiment

We now show several examples of how the defined API would support: common Internet applications, difficult usage scenarios and MobilityFirst specific scenarios.

Content Retrieval. Among the services that would inherently benefit from name based routing there is content retrieval. The GUID abstraction nicely fits the needs of this context by allowing two different retrieval methods: by referencing contents with specific GUIDs or by contacting web servers through their GUIDs. Moreover delivery options could be exploited to achieve retrieval flexibility and to exploit in-network services.

	WIFI	**ETH 10**	**ETH 100**
Server 1	23.03 ± 10.52	14.74 ± 1.17	13.15 ± 1.02
Server 2	30.23 ± 8.66	17.58 ± 0.75	15.16 ± 0.81

Table 2.3: Latency in ms to content severs

As a supportive example of the second case we have implemented a small content retrieval scenario where two content servers are located at different distances, in terms of hops count, from a client. Figure 3 (left) shows the structure of the network. The core network is based on Ethernet links at 100Mbps or 10Mbps, while we change link *L1* from either a 54 Mbps Wi-Fi link, a 10 Mbps Ethernet link or a 100 Mbps Ethernet link. Table 2.4.3 shows that the latencies in the three cases do not differ much between the two servers. In this scenario, the end user sequentially requests 20 contents (replicated at both servers) of size 24 MB sending a request message addressed to a GUID representing the web service and try to retrieve them using anycast delivery, as reflected in the following listing:

```
int downloadFile(char *myGUID, char *sGUID){
    /* b is the request, opts is "anycast" */
    ...
    hdl = open(sGUID, myGUID, NULL);
    ret = send(hdl, b, 32, &opts);
    while(remSize) {
        ret = recv(hdl, b, MAX_CH,NULL);
        /* process 'ret' bytes of content */
        remSize -= ret;
    }
    ...
}
```

After the 10th transfer is completed a failure occurs on the link connecting to *S2*. The retrieval times in Figure 3 depict clearly how the routing paradigms embedded in the GUID abstraction enable the handling of the link failure. In this case the 11th request is delayed due to the adjusting period of the network. The effect of the failure would be additionally reduced in case of retrieval of larger contents. Figure 3 (center) shows the throughput change after the failure occurs.

This same example could be also represented exploiting the other approach where GUIDs are used to reference contents; in this case contents would be retrieved through *get* requests providing inherent support to content location services that nowadays are achieved through DNS tricks that would not be desirable otherwise [9].

Multihoming receive. While the concept of multihoming is inherently supported in name based routing systems, how to handle multiple interfaces from a network interface point of view is still an open issue as multiple options are available. We argue that there should be a two level policy to determine the usage of the available interfaces: one based on application specific needs and one defined by the user, with the latter having higher priority. In the proposed API, the user level policy could be then expressed during the context creation provided by the open operation while the application requirements would be inherently defined from the context defined. These policies would be then applied to provide additional information to other network elements by the usage of header SIDs or entries in the GNRS. These concepts could be applied to the sample network of Figure 3. While we have analyzed the network behavior in case of failure, the use of multiple interfaces to the network could be exploited to increase performances by exploiting content presence on both servers.

In-network services usage. Extensible In-Network Services are a key feature of MobilityFirst's architecture that enables en-route or local services such as content caching, encryption, VPN, or video transcoding. While we do not plan to provide full access to network services at the application level, we think that a set of content and service specific features should be addressable through the API. With the proposed API, these services could be activated on a per message level, allowing enough flexibility to the developer. As an example, we could consider the on route caching of contents. When servers or network caches respond to a content request (i.e., get (GUID, ...)) they set a content response SID in the header with cache options. Other routers in the path can then decide to cache the response for serving future requests.

Chapter 3

Named-Object Based Services

While the name object abstraction provides a perfect framework that can be relied upon, new networking solutions are required to fully exploit its potential. In this chapter we introduce three technologies that provide baseline services that we believe should be part of any evolving network architecture:

- A new multicast routing framework.
- A solution to support compute services in a name based architecture.
- The implementation of a context services framework that exploit the other two technologies.

3.1 Multicast Services

Internet applications like video streaming, online gaming and social networks, e.g. Twitter, often require dissemination of the same piece of information to multiple consumers at the same time. While multicast routing protocols have long been available, most of these applications rely on unicast based solutions that exploit overlay networks aimed at improving the efficiency of pushing the required data without support from the network. Recent increases in network traffic associated with the growth of mobile devices, Internet-of-Things (IoT) devices, smart wearables and connected vehicles, motivate the need for efficient push multicast, a service that is not well-addressed through overlay solutions. The new smart-objects category is particular interested in this service [33]. Consider for example IoT based messaging scenarios: a typical query involves sending short messages to hundreds or thousands of users or application agents, so that scalability becomes an issue, as multiple unicast messages through an overlay service can easily overload the network. Mobility of end-devices results in additional complexity, especially for dynamic environments such as vehicular communications. For example, if a single warning message needs to be pushed to hundreds of cars and pedestrians in a given area, multicast groups would need to be maintained across a large number of access networks in order to efficiently support such one-to-many communication.

Using appropriate multicast routing solutions would help solve these problems by improving network efficiency, while reducing the complexity and cost of deploying such applications. However, existing network-layer multicast solutions (e.g., PIM-SM [25], MOSPF [26]) have not been widely adopted due to fundamental problems that are a by-product of the original Internet design geared toward static host-centric communication. These solutions implicitly couple the forwarding path (*location*) with the multicast group (*name*). Whenever a receiver moves to a new location, it has to rejoin the multicast tree it was previously a part of and the network has to change the tree structure accordingly. This can cause packet loss during the process and large amount of distributed control traffic is generated to modify the tree structure. The problem becomes particularly acute for applications like Twitter where each receiver might have more than 100 groups to join each time it moves. Secondly, extending these protocols to inter-domain has achieved mixed results, with issues of scalability and coordination across domains [34]. For example IP multicast based on PIM-SM [35] relies on rendezvous points (RPs) as the shared root of a tree. However domains are often unwilling to have RPs for their local groups to be maintained in other domains. This leads to having RPs in every domain connected in a loose mesh, that require periodic flooding of control messages for coordination and management. Multicast group address assignment may require a separate protocol altogether, such as the Multicast Address-Set Claim (MASC) protocol used in conjunction with BGMP [36]. All of these problems have negative consequences for highly dynamic environments and emerging application scenarios. For example, in the vehicular use-case previously described, group membership changes rapidly with vehicular mobility. In addition, the context of data-delivery may change with time as well. An accident or traffic-alert push-notification to a group of cars in NJ Turnpike is such an example. Table 4.1 describes a sample set of application

Application	Multicast Type	Group Size	Group Flux	Group Longevity	Data Flow Size
IoT commands	Push	1000’s	Hours	Days	KB-MB
Accident notification	Push	100’s	Seconds	Minutes	KB
Twitter	Pull	100’s of 1000	Minutes	Months	KB-MB
IPTV	Pull	1000’s	Relatively static	Months	GB
Multiplayer games	Push/Pull	100’s	Hours	Hours	GB

Table 3.1: Emerging multicast application and their characteristics

scenarios that require efficient multicast primitives and their characteristics.

Application layer solutions for multicast have also been explored in this context; works like SCRIBE [37] and ZIGZAG [38] sought to find scalable and efficient solutions by building an overlay among the receivers in a tree or mesh structure. These solutions do address mobility and inter-domain management issues, but due to the lack of topology awareness, they may incur high levels of network traffic. In addition, forcing the end hosts to replicate packets, instead of dedicated routers results in heavy workload on the end hosts, which may have intrinsic power and computation constraints.

Based on the above considerations, a native network layer multicasting solution is identified as an important goal for future networks which are increasingly required to support many-to-many communication modes.

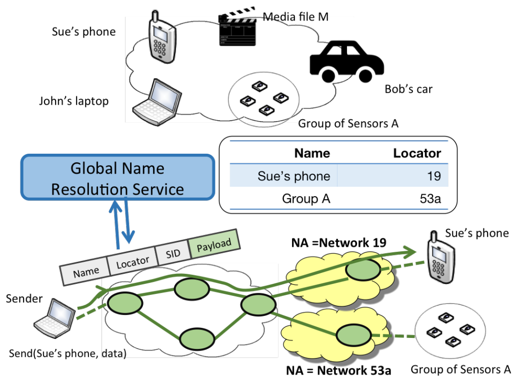
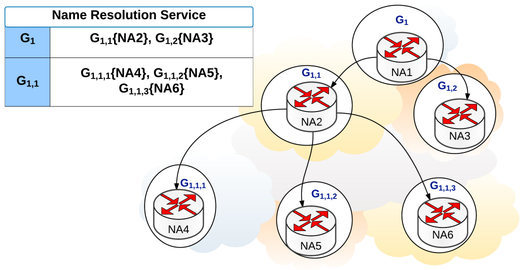


Figure 3.1: Named Object abstraction with clean separation of naming and addressing

We proposed a solution based on named objects and a dynamic name-resolution service for mapping names to routable network entities. Separating names (identities) from addresses has been advocated by the research community [3, 14, 39] for quite some time and has inherent benefits in handling mobility and dynamism for one-to-one communication. But they also provide additional advantages by facilitating creation of new service abstractions that can be used to design solutions for multicast services.

First, names can be used to represent many different Internet objects; for example, a cell-phone, a person, or a group of devices, as shown in Fig. 3.1; the latter perfectly applies in the context of multicast to define participation of end-hosts. Moreover, new entities can be integrated within these names, not being constrained to end points; through this, we gain the ability to directly refer to network entities that actively participate in the formation of a multicast tree, such as routers that implement the multicast routing protocols.



Name Resolution Service	
G_1	$G_{1,1}$\{NA2\}, $G_{1,2}$\{NA3\}
$G_{1,1}$	$G_{1,1,1}$\{NA4\}, $G_{1,1,2}$\{NA5\}, $G_{1,1,3}$\{NA6\}

Figure 3.2: Hierarchical tree structure maintained in a name resolution service, with names of tree nodes recursively mapping to routable addresses

We exploit names to design a Named Object Multicast (*NOMA*) [40] solution which relies on separation of names and addresses using a globally distributed, logically centralized name resolution service, similar in spirit to an evolved DNS. In *NOMA* each multicast group is identified by a unique name across all domains, thus separating routing logic from group management. *NOMA* takes advantage of the dynamic name resolution service to manage the tree, using name recursion, to store the tree topology. This is achieved by mapping unique names assigned to participating routers to their children nodes, as shown in Fig. 3.2. Data forwarding is then performed using tunnels between participating nodes; end-to-end information is preserved within the packet, while the information globally available in the name resolution service is used to identify next hops in the distribution path allowing for quick branching and replicating decisions. Finally, dynamicity of mobile environments is handled by decoupling the participants name from their location through the resolution service and periodically recomputing the multicast tree; the system first needs to translate the name into a list of host names participating in the multicast group. The routable address (locator) of each host (whether mobile or static) can then be identified by a subsequent query to the name resolution service.

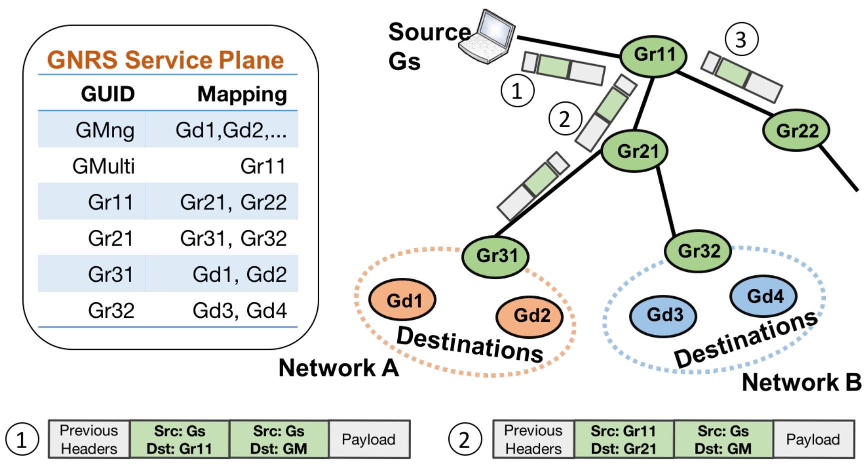
3.1.1 NOMA Design

NOMA aims to achieve efficient multicast communications through the employment of a logically centralized, globally distributed name resolution service associated with name based communications. In order to explain *NOMA*'s design, we utilize a Global Name Resolution Service (GNRS) as a network-wide entity that provides an API for inserting and querying mappings between unique name identifiers and a set of values which can include network addresses, other name identifiers and related parameters – e.g. node properties, past locations and more. In spirit, this service is very similar to current Internet's DNS, which has already been effectively applied for new service functions such as load balancing and service replication. Even more interesting services can be realized with the next generation of global name resolution services such as DMap [23] and Auspice [24] introduced recently. The key advantage of using a name resolution service is to achieve a clean separation of network names from addresses.

NOMA's design, as proposed here, is based on MobilityFirst (MF [39]), which is a clean-sate network architecture for the next-generation mobile network where DMap [23] is used to provide resolution of names, that are Globally Unique Identifiers (GUIDs), into routable network addresses (NAs). Moreover, MF incorporates a hybrid name-address forwarding scheme, in which routing components use availability of both names and addresses in packet headers to perform forwarding decisions. Note that even though *NOMA* is based on MF, the same design concept can be applied to IP extensions (such as HIP [3]), overlay protocols (such as SCRIBE [37]), or clean-slate ICN protocols such as NDN [41] and XIA [42] through the use of a similarly designed name resolution service.

Multicast Tree Management

Multicast management consists of two core operations: membership of destination nodes and building and management of multicast trees. Both operations can be streamlined by exploiting the logically centralized, globally distributed, name resolution service (GNRS); in particular by using two forms of name indirection. A first unique name ($GMng$ in Fig. 3.3) is assigned to perform the task of node membership; all entities interested in receiving data from the multicast flow, can request to join by inserting their own unique name into the corresponding mapping in the table. This information is then exploited close to the source by a multicast service manager, which builds an efficient tree based on the available resources and the size of the required tree. Recursive mappings are then used to express the tree graph: by assigning to each branching router a name that exclusively identifies it in the context of the given multicast flow, we recursively follow the tree structure. For example, in Fig. 3.3, the root of this tree is identified by the multicast flow unique name mapping to the first branching router ($GMulti \rightarrow Gr11$); this router then maps to its children in the tree ($Gr11 \rightarrow \{Gr21, Gr22\}$); this continues until the leaves of the tree are reached, where we identify the leaves as the destination nodes. As time progresses and destinations join or leave the multicast group, the service manager can rebuild the tree information contained in the GNRS to trigger the required update.



GUID	Mapping
GMng	Gd1,Gd2,...
GMulti	Gr11
Gr11	Gr21, Gr22
Gr21	Gr31, Gr32
Gr31	Gd1, Gd2
Gr32	Gd3, Gd4

Figure 3.3: Multicast architecture overview

One of the novelties of *NOMA* is that it can support push mode of multicast, where a source can send a single packet of multicast data, without the knowledge of the tree and this can happen even before the tree has been built. On receiving a multicast packet, for a group G_m, the gateway router at the source domain, acting as the multicast service manager, will do a membership query to the GNRS. GNRS supports recursive queries that return the host GUIDs along with the NAs of the domains they are currently connected to. Having the service manager on the gateway enables the tree computation to be topology-aware, as unicast path information of the NAs is available at the gateway, which is then used to build the tree. Once a tree is computed, it is updated in the GNRS such that downstream nodes do not need to recompute the tree again. This is quite different than distributed tree management techniques used in IP multicast since *NOMA* does not require flooding of multicast control messages (for example, source active (SA) or Join messages in PIM-SM and MSDP [35]) across domains, as shown in Fig. 3.4. The latter limits the scalability for traditional multicasting techniques to small to medium groups, as shown later in Sec. 3.1.2. Also, using unique names to represent a group, members of the group as well as the multicast tree eliminates the need of a separate address allocation protocol, similar to MASC required for BGMP [36]. For evaluation purposes, we focused on two categories of multicast tree computation algorithms, i.e. shortest path trees (SPTs) and Steiner trees. A constraint of having centralized computation of trees is complexity and hence we opted for SPT and its modifications, even though our design is not limited to any specific algorithm.

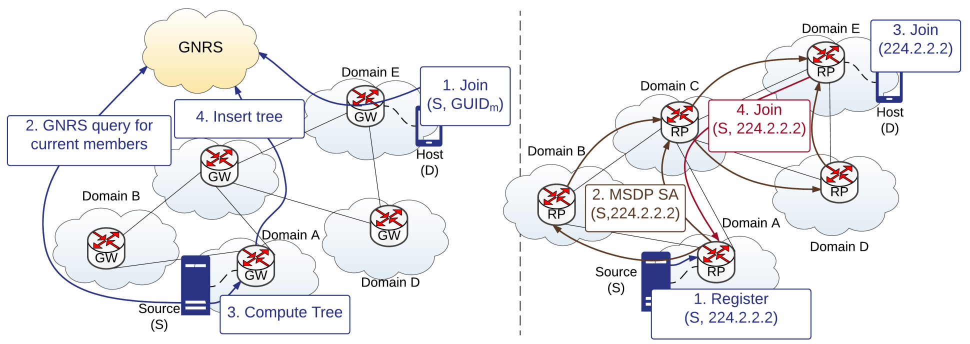


Figure 3.4: Tree building steps comparison of *NOMA* with IP multicast

Data Forwarding

Once the multicast tree is established, data forwarding can exploit the information contained in the GNRS to efficiently flow through edges between the nodes of the tree. In order to do so, we exploit address encapsulation, where two pieces of information are carried in data packets at the same time: Internally (i.e. second field in the green packets in Fig. 3.3), the encapsulated information carries the source and destination of the multicast flow, providing valuable information usable by all nodes along the path to easily identify data streams. Externally, routing information to perform hop-by-hop forwarding from one branching node to the next is placed. At each branching node participating in the multicast, forwarding decisions are performed by querying the GNRS to obtain information on how many next hops it has to forward to, generating required duplicates and replacing the external routing information with the new hop source and destination; this process is exemplified in the figure, where node $Gr21$ generates 2 duplicates for its two children, replacing headers accordingly. Intermediate nodes along the path forward encapsulated packets based on normal unicast rules. This reduces complexity of multicast packet processing to only a subset of nodes of the tree. To reduce the need of continuously involving the GNRS in the forwarding procedure, mappings can be cached at each hop, avoiding traffic and computational overhead. The tradeoff for this approach comes at the cost of slower reaction times to tree change events. More details on how to handle tree restructuring and end points mobility is provided in the following section.

3.1.2 Evaluation

In this section we present detailed performance evaluation based on a combination of large scale analytical modeling and fine-grained packet-level simulation on network simulator (NS3).

Tree Generation Algorithms

NOMA provides a framework for managing and deploying multicast communications, independently from the tree generation algorithm employed. While this is a valuable feature of the design, it is necessary to study different algorithms and heuristics in the context of choosing one that can effectively utilize unicast routes, and is lightweight enough to be able to run at a single router. We looked at two main categories of algorithms for building multicast trees, namely shortest path trees (SPTs) and Steiner trees. Although Steiner trees provide an optimal solution in terms of overall network resource utilization, they are NP-hard to compute. Several Steiner heuristics have been proposed over the years to provide near-optimal solutions [43], with relatively high computation cost. However, computational complexity is a key constraint for our design, since the tree computation is *centralized.* We instead opt for the SPT algorithm that uses inter-domain unicast route information and require no further computation, but is less efficient compared with a Steiner tree. In SPT, packets are forwarded along the *longest-common path (LCP)* to all the destinations, as single copy, until the branching point is reached, where the packet is copied and delivered towards multiple destinations. This allows all destinations to receive multicast packets across the shortest path from the source. We also analyzed other heuristics that aimed to further minimize the overall network traffic with moderate computation. One of these heuristics is the *look-ahead longest-common path (LA-LCP)* algorithm. Unlike LCP, which branches whenever there a divergence of shortest paths to multiple destinations, LA-LCP, compares the overall network cost of branching from the current node and branching from each of the possible next hops, and decides to branch downstream if the cost is lower for the latter, thereby deviating from the SPT. This reduces the overall packet hops in the network, with slight increase in computation complexity.

Fig. 3.5 plots the CDF of total packet hops to reach 20 randomly placed destinations from a single source on a 100 node *Erdős-Rényi* random graph for each of these algorithms. As seen from the plot, all the multicast algorithms are much more efficient than unicast. Although Steiner provides the most efficient trees, it is computationally intensive. In comparison, LA-LCP provides *reasonable* performance with lower overall network overhead compared to traditional longest common path.

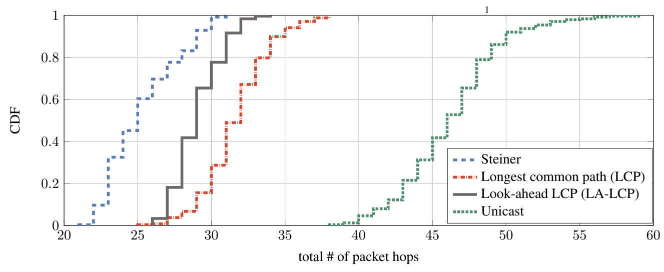


Figure 3.5: CDF of performance in terms of packet hops for different multicast tree generation algorithms, for 100 node random graph with 20 randomly chosen destination nodes.

Comparison to IP multicast

In this section we compare pull-based multicast of *NOMA* with IP based inter-domain multicast, namely, PIM-SM standard coupled with MSDP [35]. Through the results we highlight two key benefits of using *NOMA*, namely, 1) lower control overhead for maintaining a multicast group, and 2) better handling of mobility for data forwarding. Note that BGMP [36] is another prominent inter-domain IP multicast scheme, however, it is not well-suited for applications that involve dynamism and fast changes in the tree, and hence has not been a focus of our evaluation. BGMP allows multicast route updates to be carried along with inter-domain BGP messages and therefore tree changes occur at a much slower time-scale than PIM-SM/MSDP (typical BGP updates take about 100 seconds to propagate throughout the network [44]).

Control overhead: The advantage of using unicast routes to build the tree is that no multicast specific control overhead needs to be exchanged across networks. This is crucial for inter-domain settings where flooding periodic multicast tree update messages is not tractable. In Fig. 3.6 we plot the multicast specific messages exchanged for setting up a tree and forwarding packets for increasing graph sizes, with the topology being an *Erdős-Rényi* random graph, and 50% of the nodes being randomly chosen to have destination clients part of the multicast group. For *NOMA* this includes 1) the GNRS insert messages from each of the destination networks for joining a particular multicast group, 2) the GNRS insert from the gateway at the source domain to insert the generated multicast tree, and, 3) GNRS query and responses during data forwarding at the branching nodes. The GNRS is implemented as a distributed hashmap, following the DMap design [23], with the same mapping stored at multiple locations. For evaluation purposes, 3 GNRS instances were maintained, therefore each insert incurred 3 unicast messages to 3 specific nodes (determined by a hash function), whereas each query was *anycasted* to the nearest of the 3. In comparison, for PIM-SM+MSDP the overhead numbers comprise of, 1) the flooding of Source-Active (SA) messages from the source domain throughout the network, and, 2) the Join messages from the domains which have destinations nodes interested in receiving packet from that particular source. As seen from the plot, maintaining a multicast tree in the GNRS has higher overhead for smaller sized graphs (for example, for a 20 node topology, shown in the zoomed in section of Fig. 3.6), but it scales elegantly with size. Using PIM-SM+MSDP, on the other hand, becomes intractable as the number of nodes increases. With more than 40 thousand ASes in the Internet today, if every domain was multicast enabled, the cost becomes too high to maintain a distributed tree. Similar trends were observed by varying percentage of destination networks for fixed graph sizes and is not included here for brevity.

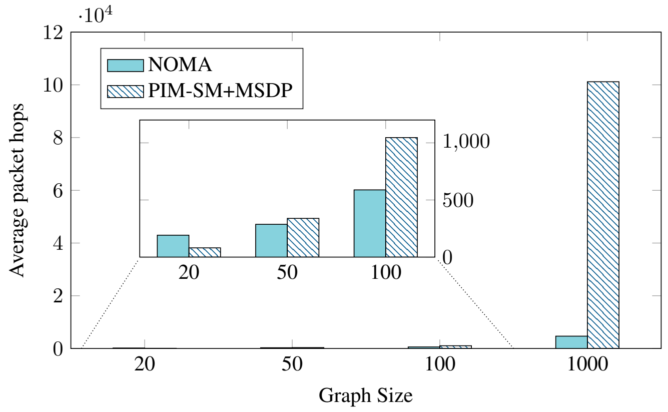


Figure 3.6: Control packet overhead for tree setup for varying graph sizes

Handling mobility: *NOMA* seamlessly handles client mobility and the dynamism in tree-changes thereof, by periodically recomputing the tree and updating the corresponding GNRS entries. In addition, to counter packet-loss due to mobility, *NOMA* supports unicast 'repair' packets to be sent from a previous edge node to the current point of attachment of a mobile client, until a tree update restructures the tree. We performed detailed packet level simulations in network-simulator (ns-3) on a 20 domain random topology with randomly placed mobile and static clients, for both *NOMA* and an IP multicast implementation of PIM-SM + MSDP. Fig. 3.7 plots the fluctuation in received throughput at a client receiving a multicast stream of 2Mbps on the event of mobility. A mobility event is characterized by disconnection of a client from its attachment point and re-association to another node, following a period of association (uniform random variable $U(0,1)$ seconds), as highlighted in the figure at $t =\sim 77$ seconds. *NOMA* periodically restructures the multicast tree every 10 seconds for this scenario, whereas, IP multicast restructures following the client explicitly joining the tree at the new point of association. Therefore, multicast traffic for *NOMA* falls to 0, until tree is restructured at $t = 80$ seconds. However, repair packets are delivered to counter packet loss and reordering, highlighted by the black trajectory in the figure. Note that *NOMA* is based on MobilityFirst (MF) transport, that uses reliable hop-by-hop delivery of large chunks, and the throughput received by the client is therefore in steps with the average being 2Mbps. In comparison, for IP multicast, data throughput falls following temporary disconnection and re-connection, as shown by the red dotted trajectory.

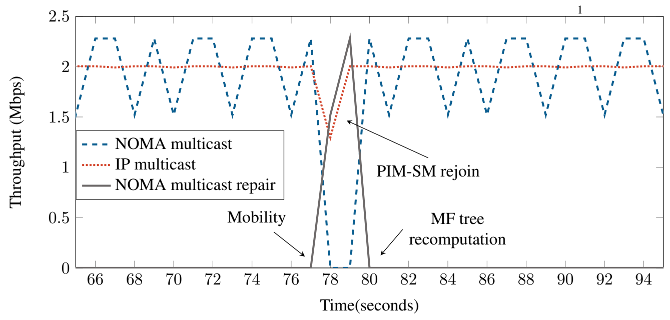


Figure 3.7: Comparison of average multicast throughput received at a client with mobility

Mobility not only affects the instantaneous throughput at a client, it also leads to loss of packets during the interval of disconnection, re-association of the client, re-joining and re-structuring of the multicast tree. Additionally, in a practical setting, for IP multicast, the mobile client will spend a significant amount of time for new IP address allocation through DHCP, which has not been taken into account for this evaluation. This packet loss and reduction in overall throughput is highlighted in Fig. 3.8 where we plot the aggregate throughput at a mobile client for increasing rates of mobility, that moves randomly with exponential random mean mobility interval of 50, 20 and 10 seconds. As seen from the plot, aggregate throughput for *NOMA* does not change with mobility, primarily due to native features of MF such as hop-by-hop reliable delivery and storage-capable routers to handle temporary disconnections. In comparison, IP multicast throughput significantly worsens with increasing mobility speeds.

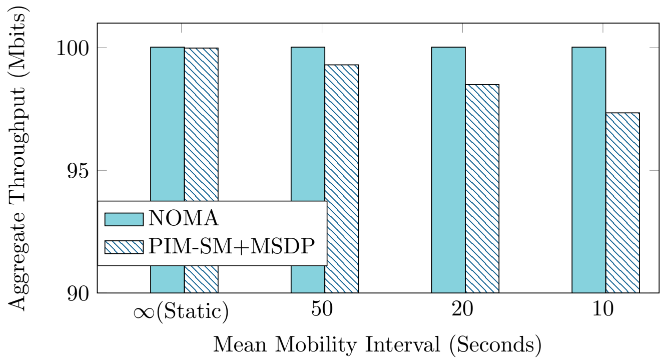


Figure 3.8: Aggregate throughput at a mobile client, with increasing mean mobility rates; mobility event is determined by an exponential random variable with the mean

3.2 In-Network Compute Services

Mobile devices such as smartphones and tablets have overtaken personal computers to become the primary platform for accessing the Internet. Internet reports caution, however, that video constitutes a majority of the traffic and will increasingly dominate mobile data in the future [45]. With this growing trend, service providers face significant challenges in providing quality service while maintaining high network efficiency. Mobility further complicates the user-experience aspect due to variable link quality, temporary disconnections, and seamlessness when crossing physical networks. Service providers commonly adopt short-term strategies of limiting user network activity with expensive data plans and bandwidth throttling, and end up with outright poor or patchy service performance. While better spectrum management and capacity improvements promise to help, we argue that innovation in network protocols and in-network services can provide large improvements over current data delivery efficiency, particularly in mobile wireless networks.

Widely deployed middlebox solutions such as content proxies have helped since the early days of the Internet to opportunistically improve data delivery performance. In recent years, content delivery networks and cloud platforms boast further gains through strategic placement and geo-colocation of content and services with end-user. However, these solutions have often aimed to improve raw server-client RTTs, but do not address specific challenges of wireless networks or mobile devices in any meaningful way. CDNs and cloud applications are generally built as overlay services that can react to long-term differentiation in path qualities. While this may suffice for certain applications and when end-hosts are on fixed wired networks, wireless networks aren't similarly benefited. A mobile device in a wireless network may have vastly different network experience in a short time span due to location, network load, or the particular access technology available(e.g., 3G, 4G or WiFi). For wireless networks, it is well understood that traditional content delivery methods, and in particular, connection-oriented transport protocols, have not provided satisfactory performance. We think that network-assisted approaches (e.g., storage and compute as integral components of the network) can provide a boost for content delivery in mobile wireless networks.

Streaming video, in particular, can benefit from an approach where the video bi-trate can be dynamically adapted to match available bandwidth at the client access link using in-network compute resources. It is well known that the last-mile is usually the bottleneck link, and for wireless clients, the network performance could be highly variable under mobility. Solutions that address network performance variations to-day either buffer excessively at the client, or require client-side estimation of available bandwidth. While over buffering could be wasteful, accurate client-side estimations of available bandwidth is known to be problematic [46]. Furthermore, recent studies have shown that a multiplicity of rate-adaptive flows at access networks can lead to unexpected degradation and instability in delivery performance [47]. To address these challenges and to enable a host of novel value-add services (e.g., embedding of ads, localized alerts, regional subtitles, etc.), we argue for a network-assisted solution for rate-adaption. The placement of such an adaptation service close to the edge would also enable the most accurate estimation of network performance as experienced by the client. As argued by others as well, we think the wireless service providers could make use of such in-network adaptation services as a way to effectively manage video flows at an aggregate level to handle traffic bursts, and to implement fairness, stability, and efficient delivery in the access network.

3.2.1 In-Network Compute Architecture

To enable future extensions to the network protocol, without expensive hardware replacements and disruption, MobilityFirst builds in an optional and dynamically pluggable *compute plane*. Examples of such need are additions of new service types, new principal types, new addressing structures, or extensions to the end-to-end security protocol. We envision service providers and network operators to be able to perform relatively simple upgrades in the form of software updates and addition/replacement of pluggable hardware modules to extend the data plane functionality. Furthermore, we also postulate that such extensibility can enable third party application service providers (via the ISPs) to deploy either service end-points or service adaptors that are both closely integrated with the delivery path and best located to improve client experience.

3.2.2 Overview

Consider an end-to-end application service S that requires content delivery to mobile end-points. For simplicity, let us assume a server-client interaction model with a fixed server and a mobile client. Under good conditions, when the mobile client has good connectivity to the network core, the server responds to the client request by delivering the highest fidelity content (C). Under these conditions MF provides hop-by-hop transport under either unicast or multihomed delivery (e.g., when connected to both 4G and WiFi) as specified by the delivery service type during a send operation. Short-lived disconnections or minor variability in the access link can effectively be handled through a combination of in-network store-forward buffers in MobilityFirst routers (MFR) and contingency buffering at client ahead of actual consumption rate (e.g, video playback). If access conditions change significantly, however, and remain so for extended periods of time, it will become hard if not impossible to deliver the original high-fidelity content to the client in a timely manner. These conditions may occur under a burst of traffic or when the client moves to outer/poorer coverage area. In these cases, the compute plane extensions allow an authorized *adaptor* A_S service, regionally co-located with the client's network, to intercept and modify original data C to subsequently deliver C' that is commensurate to existing bandwidth constraints. A second demonstrative scenario for an in-transit adaptor service is for delivery content that is context-sensitive (C_X - for context X). Client mobility may redefine context (X') thus requiring an adaptation to the delivered content($C_{X'}$).

Key Components

Enabling this dynamic adaptation requires compute services to be addressable, application providers to be able to deploy these services at appropriate points in the network, traffic to be correctly steered when beneficial, integrity of content and protocol continuity to be maintained post adaptation, and finally, applications should be provided a flexible service interface to invoke these services.

Service Addressing: Generally, the compute layer will host services that are directly addressable by information contained in the data packet, i.e., the GUID of the service is explicitly specified in the packet as an extension header. However, service providers may also invoke specific compute services on packets based on header signatures alone, e.g., source-destination GUID or NA fields. These *transparent* services are similar to middlebox approaches today, which are routinely used to support application-layer proxies, content caches, traffic steering, and security functions. We allow for both directly addressable and transparent services (which may also have internally known GUIDs) in the MF compute plane.

Hosting Platform and Registration: While routing fabric could potentially steer traffic to service endpoints located anywhere, packets will suffer least additional latency when the compute platform is co-located with the routing fabric. We envision both close hardware integrated solutions, where routers are embedded with a compute plane composed of general purpose CPUs, GPUs or pluggable accelerators, as well as co-located compute clusters or smart-sized clouds as possible platforms for hosting computer services. While the latter is less flexible and is challenging to scale, it presents the least overhead when the computations are limited and do not require access to other distributed state. Data compression or certain security checks are examples. When the compute service are more intensive, need to operate across a sequence of packets, or need access to distributed state, it appears that a co-located cloud is a better fit, as described in our PacketCloud instantiation [48]. In each case, MFR exports an interface for services to register a GUID addressable service with the forwarding plane. The attachment of the service instance will also be updated within the GNS to enable data packets that request the particular compute service to be routed correctly.

Geo-location and Routing: We expect that one or more instances of an in-network compute service will selectively be deployed at locations seen as beneficial to the application provider, primarily due to expenses incurred in running on the platform provider. Since application services may be consumed by clients across the globe, it implies along certain paths these in-network adaptation services may be unavailable. Such clients are either served through end-to-end adaptation only, or alternatively, packets may be routed through paths that do have registered service instances and do not cause too much path stretch. This can be done in MF through native tunneling, where packets are first tunneled to the network hosting the service (known to application provider, and also registered in the GNS), and following adaptation are then routed to the destination client. In the non-tunneled default case, the packets are routed first to the target network of the client, where the compute plane service, if available, will act on the packet. With these, application providers can plan placement of their services according to benefits to their end-users, so long as ISPs in those geo-regions provide an open hosting platform.

3.2.3 In-Network Rate Adaptation as an example

In this work [49] we focus our attention on a specific use case providing details on how the service and content APIs could be jointly used to offer an in-network service that does rate adaptation when delivering video streams to mobile devices that experience variable connection quality. Bitrate adaptive streaming protocols have been widely adopted in most commercial solutions thanks to their flexibility in providing the desired service given variable network conditions. In particular, Dynamic Adaptive Streaming over HTTP [50] has been increasingly chosen as the standard go to solution thanks to its easiness of implementation, as it relies on the already existing HTTP infrastructure of webservers, proxies and caches. While it is easy to reckon the simplicity behind these protocols, they all rely on the ability of the client to estimate the available bandwidth, a task arguably very difficult under normal network conditions [46], and even more complex under wireless and mobile environments due to the high dynamicity caused by time-varying fading, shadowing interference and hand-off delays [51, 52]. Introducing an in-network service to provide the adaptiveness of the video stream behind these protocols, would allow to off load the task of deciding on the adaptive stream to the components having the best knowledge about the available resources while maintaining most of the original good properties of these solutions; on the other hand, overly complicated services would discourage adoption.

To deploy an in-network service, the service developer, which for video streaming may either be the content distributor or the edge-network service provider, has to deploy, configure, and initialize the service instances at required network locations. To bring up an in-network service we refer the reader to the PacketCloud framework [48]. Once the service is up, the content server may pass further configuration such as listings of videos and available bitrates. Such metadata is transferred in the form of Media Presentation Description (MPD) files specified by the *MPEG-DASH* standard. To accommodate MF naming, we replace the content URLs in the MPD files with corresponding content GUIDs. Such config could be done out-of-band and apriori in preparation of handling the server-client flows. Though, the server may embed certain metadata, such as current bitrate, as parameters in the extension headers of the response chunk itself.

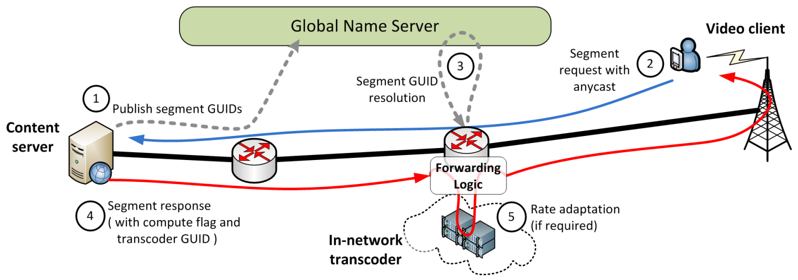


Figure 3.9: Overview of rate-adaption use-case showing MobilityFirst's support for direct addressing of content and in-network compute services

Leveraging and extending in-network functions: As previously introduced, MF offers native support for a variety of delivery services through the use of service type and service options. We have extended the basic set of name-based communication primitives to support the compute-layer service type and options, allowing application developers programmatic and in-band access to in-network services deployed on the compute plane. Figure 3.9 shows the flow of meta-data and data in a video streaming service implemented using the in-network extensions.

The first step involves the server establish network presence for the content (and hence direct addressability). It does so by attaching the GUIDs for the video segments to the network through the client interface (presented later in Section 2.4):

mfattach*(sock, GUID-set)*

where GUIDs in *GUID-set* GUIDs will be associated with NA(s) corresponding to the host's one or more network attachment points, resulting in GNS mappings.Following this, the streaming proceeds as sequence of content requests and responses between client and server, starting with a request for video meta-data, i.e., the MPD file:

mfget*(sock, GUID, data_buf, size, svc_flag, opts)*

where GUID is a segment's GUID retrieved from the MPD file, and the svc_flag and opts encode any special delivery requests such as 'ANYCAST'. When the server receives a request (via *mfrecv*), it replies with the segment using MF's response API that matches the response with the corresponding *get*. To enable in-network rate-adaptation, the server requests the compute-layer delivery service, specifying the GUID of the transcoder service and the current encoded bitrate:

mfget_response*(sock, getId, data_buf, size, svc_flag, opts)*

where getId (returned during preceding recv) is used to match the *get* request, and svc_flag ("COMPUTING") and *opts* (in key-value pairs) encode details of the compute layer service to be invoked on the payload. The server adds the GUID of the compute service and the current encoded bitrate to the options parameter. Note that the response chunk may not always be steered to the compute service before reaching the client, and the decision will be based on the bandwidth available to the client device at the time of delivery. This is enabled through a routing-layer service implemented on the edge router.

3.2.4 Implementation

The in-network compute architecture consists of: a new network stack and socket API for hosts that implements the service interface used by the end hosts of the system, a software router that implements the hybrid GUID and NA based forwarding and storage-aware routing protocols, and a computing engine/platform that presents an open API for configuring and running in-network services.

Host Stack and API: The host stack has been implemented on Linux and Android platforms as a user-level process built as an event-based data pipeline. The stack is composed of a minimal end-to-end transport to provide message level reliability, the name-based network protocol including the GUID service layer, a reliable link data transport layer, and a policy-driven interface manager to handle multiple concurrent interfaces. The device-level policies allow user to manage how data is multiplexed across one or more active interfaces. The previously introduced socket API is available as a linkable library and implements the name-based service API which include the primitives *send, recv, and get* and a set of meta-operations available for instance to bind or *attach* a GUID to one or more NAs, configure transport parameters in the stack, or to request custom delivery service types such as multicast, anycast, multihoming, or in-network compute. Additionally to the general aspects just presented, the host stack, in coordination with the API interfaces, provides the functionalities to interact with the in-network service; we introduced in the the previous sections different alternatives to provide tools for interacting with the network compute services; for this particular implementation, we exploited the option provided by the MF network protocol of flexibly introducing extension headers in the network header. The host stack is then in charge of accordingly fill in the fields of the extension header providing information regarding the carried video segments, such as bitrate and encoding information as passed from the application layer during the content operations invocations. A full description of the implementation will be later presented in Section 2.4 where the MobilityFirst network API is presented in detail.

Video Client and Server: We use the presented host stack and API to implement a modified DASH system that exploits the in-network service. In order to implement a DASH video client that can rely on the in-network adaptation instead of implementing bitrate adaptation locally, we took advantage of the *VLC-DASH* plugin presented in [53] and modified it to display received segments independently of the delivered representation. We've implemented a basic DASH webserver using MF network API that handles requests for content (i.e., video segments and meta files) addressed by their GUIDs. In order for the plugin to work with the MF network and the implemented server, HTTP requests are forwarded to a proxy co-located on the same machine that translates URL request to MF content GUID requests. Mappings from video segments URLs to GUIDs is implemented using a local data-base, but we plan on developing a name resolution service for future experiments; a single GUID is used to identify segments independently from their bitrate.

Router: The software router is implemented as a set of routing and forwarding elements within the Click modular router. The router implements dynamic-binding using GNRS, hop-by-hop transport, and storage-aware routing. It integrates a large storage – an in-memory *hold buffer* – to temporarily hold data blocks when destination endpoints during short-lived disconnections or poor access connections. For dynamic in-network binding of GUID to NA, the router is closely integrated with the in-network

GNRS. It attaches to a local instance of the distributed service, which is often co-resident on the physical device, but can also be hosted on separate co-located node. GUID-to-NA mappings once looked-up are cached by the router until TTL or expiry values established at the time the binding was created. The access routers implement a rate monitoring service that tracks the available bandwidth for each attached client. For the WiMAX network, the rate is obtained from the WiMAX base station which exports the most recent downlink bitrate allocated to each client by the scheduler based on a client's location, client offered traffic, and overall load on the BSS. We have implemented a similar rate monitoring capability for WiFi Access Points using standard 802.11 netlink configuration utilities. The monitored information is then used to select the eventual necessity invoking the transcoding service; this choice is implemented as a simple threshold logic where segments are forwarded if their required bitrate is not met by the available bandwidth. In the case segments need to be transcoded, the router forwards them to the transcoding service adding the available experienced performance to the chunk's extension header.

Cloudlet: Our hosting platform for compute services is based on the PacketCloud framework [48]. A cloudlet at a minimal is composed of a controller module, a pool of compute nodes, and a service interface that exposes management API for application providers to manage the lifecyle of their compute services. The controller can dynamically provision whole compute nodes per service or run them in virtual environments with clean isolation. For our Linux-based implementation, we use Linux Containers (lxc) as a light weight VM solution. The controller interfaces with the co-located router to register/de-register service GUIDs to control traffic steering between the router and the compute service. In our current implementation, traffic is steered over a TCP link set up between the software router and a basic TCP server running within the VM that hosts the service. The interaction protocol encodes the packet payload and some meta information such as input arguments supported by the compute extension headers. Lower overhead mechanisms to implement the interface are being considered.

Rate-Adaptor Service: The in-network rate-adaptor service combines both caching and transcoding functions. Video segments that require transcoding to a different bitrate can either be transcoded in real time, or returned from the cache if a version of the segment in the target bitrate exists. The cache entries can be populated from both in-transit video segments as well as those that emerge from a transcoding operation. Limitations on both storage and compute resources present interesting tradeoffs across optimal reuse and minimizing the latency of transcoding in real-time. The transcoding functions are based on the ffmpeg multimedia framework. Once received segments reach the adaptor they get transcoded based on the statistics obtained from the chunk extension header, selecting between the different potential encoding bitrates.

3.2.5 Prototype Evaluation

In order to evaluate the use case based on the in-network compute architecture, we deployed it on the GENI [54] nation wide testbed. The following subsections describe the experiments performed and the results collected.

Deployment on GENI Wide-Area Testbed

The GENI testbed supports deep programmability by allowing experimenters to run custom network and software stacks on testbed nodes, and through flexible interconnection specification (incl. layer-2 links) [54]. It provides scale and a large geographic footprint by stitching together several academic and other non-commercial testbeds using Internet2's 10/100 Gbit backbone network and a host of other regional networks that connect up the individual institutions. The resulting nation-wide testbed with a variety of wired and wireless segments, aims to emulate, albeit in a limited way, the heterogeneity of the real Internet.

Experiment Setup. We deployed MF prototypes at seven GENI sites as shown in Figure 6.4. The routers, naming servers, and applications run on Xen VMs (total 14, 2 VMs per site) each with 1 GB memory and one 2.09 GHz processor core. At the Rutgers site we also provision a raw node to run the transcoding server. Each MFR is configured with 1 or 2 interfaces depending on their role as core router or as an access/edge router, respectively. All routers have a core-facing interface connected to a layer-2 network that connects all seven sites. This was setup using a multi-point VLAN

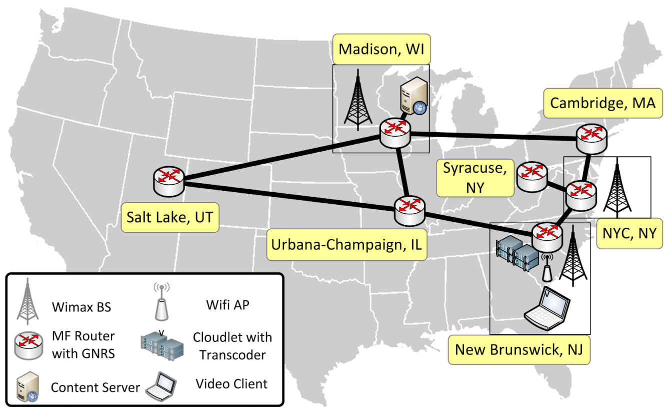


Figure 3.10: Prototype components deployed on the GENI testbed

feature provided by Internet2's Advanced Layer-2 Service (AL2S). Routers at three sites (viz. Wisconsin, Rutgers, NYU) are configured with a second interface connecting to the local wireless network (WiMAX). Mobile wireless or emulated clients connect to MF network through this interface. Routers are each configured with 500 MB of hold buffer space, and have access to a GNS service instance co-located on the same node. The GNS service runs at all seven sites using a replication factor of k=3, and achieves a 95th percentile lookup latency of under 80ms.

Evaluation of In-Network Rate-Adaptation

We used the above setup to evaluate the in-network rate-adaptation service for a DASH video streaming application. The MF-enabled DASH server ran at the Wisconsin site, while the VLC client at Rutgers connected over WiMAX. Though the Wisconsin site has a WiMAX client network, we connect the content server over Ethernet to the access router to reflect the high-bandwidth uplinks for server deployments. Table 3.2 shows the performance of links along the server-client path, showing clearly that the client link is the potential bottleneck.

DASH Video Dataset: We use the DASH video dataset generated by Lederer et al. [55] using their DashEncoder. In particular, we use the DASH-ized files they generated from a 1080p version of the *Big Buck Bunny* animated movie available from here [56]. The DASH version of this video is available in six different segment lengths (viz. 1, 2, 4, 6, 10 or 15 sec) and at 15 to 20 different bitrate encodings (from 50

Link	**Link Type**	**Bandwidth (forward / reverse)**	**Rtt (ms)**
Video Server (Wisconsin) — MFR (Wisconsin)	10G Ethernet (VMs shared)	4.86 / 2.58 Gbps	0.72 ± 0.11
MFR (Wisconsin) — MFR (Illinois)	Ethernet/Fiber	411 / 418 Mbps	8.9 ± 0.10
MFR (Illinois) — MFR (Rutgers)	Ethernet/Fiber	83.3 / 92.6 Mbps	78.7 ± 3.95
MFR (Rutgers) — Transcode Server (Rutgers)	10G Ethernet (VMs shared)	937 / 479 Mbps	0.51 ± 0.04
MFR (Rutgers) — Video Client (Rutgers)	WiMAX	9.49 / 1.05 Mbps	54.88 ± 2.87

Table 3.2: Performance of links on the path from server at Wisconsin to VLC client at Rutgers.

Kbps up to 8000 Kbps) for each segment length. In our experiments we use the fixed 480p resolution dataset with 2 second segment lengths. The fixed resolution allows fixed playback size and the shorter segment lengths allows for entire segment to be encapsulated in a single chunk for the hop-by-hop transfer.

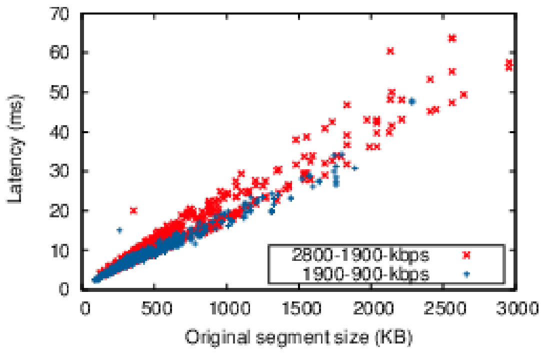


Figure 3.11: Latency for in-network transcoding vs. segment size

Microbenchmarks: We measured the latency overhead of steering video chunks to the in-network transcoding service as the time taken to packetize the chunk received at the router, transmit it to the service, wait time, time to receive the response, and to chunkify the payload so it can be forwarded-on to the destination. Figure 3.11 shows the response time in relation to the original chunk size. Since actual transcoding delays vary substantially with the transcoding profile, we benchmark here the “cached” transcode operation. Upon receiving a chunk and extracting the the encoded bitrate and the target bitrate parameters from the headers, the server responds with a pre-transcoded file of the requested bitrate. Therefore the overheads are primarily due to the messaging operations at the router and cloudlet server, and the cost to read the transcoded file from disk. In these benchmarks, we warm the cache sufficiently to enable files to be served from system buffers. As seen from the results for two sets of transcode operations (2800 to 1900 Kbps, 1900 to 900 Kbps), the overheads are in the order of few milliseconds to a few 10s of milliseconds for the larger segments. The big part of the delay is the transmission time, making a case for tighter integration of the compute service. These substantial delays also emphasize smart choice for segment sizes under variable client-access conditions, such as with a mobile client.

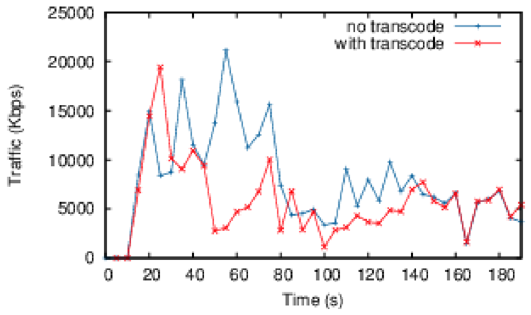


Figure 3.12: Downlink traffic at client's WiMAX interface

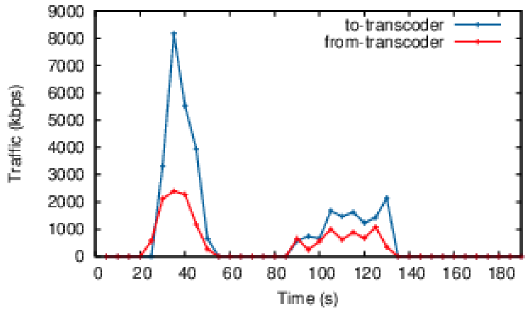


Figure 3.13: Traffic between Rutgers MFR and transcoder

Emulated Mobility Experiment: We had previously described how the client link is monitored at the access router via the information exported by the WiMAX base-station. Any variation in available bandwidth due to mobility or load is available almost instantaneously at the router. However, for the set of experiments we ran, the client is fixed and hence sees little variation in its access link properties. We therefore emulated the client mobility by intentionally modifying the bandwidth reported by the measurement service to drop below the rate required to support smooth video streaming at the original encoded bitrate. During playback, the VLC client keeps requesting the server for video segments of the animation video as long as it does not fill its buffer measured in terms of seconds of video. Under favorable conditions, the access bandwidth available to the client is sufficient for streaming the highest bitrate content made available to the user, i.e. 1900 kbps. This is the initial stretch (until about 30 sec) seen in the client-side traffic plot of Figure 3.12. During this part, the video segments traverse the server-client path without being steered for in-network transcoding. Then as we emulate client mobility at about t=30 sec, the lower client bandwidth triggers steering of traffic to the transcoder, tagged with a target bitrate of 900 Kbps. The steered traffic and the response traffic with transcoded segments at the router-transcoder link is seen in Figure 3.13. We emulate a second mobility event at about t=85 sec which once again reduces the traffic on client's downlink. Note that since the client's downlink is actually fine, the reduction is not an artifact of drop in available bandwidth. Interestingly, however, the reduction during the second event is less dramatic. The variable bitrate content happens to be of lower size during this stretch. To handle such cases, it would be help to have an estimate of the required over-the-air bitrate at a finer granularity, perhaps at the segment level, to enable more careful steering of segments for transcoding. While we did not quantitatively evaluate video quality at the VLC client besides bitrate (the quality transitions were noticeable of course), we can report that we didn't see any buffering pauses.

3.3 Context Services

Context-aware systems offer entirely new opportunities for application developers and for end users by gathering context data and adapting systems behaviour accordingly [57]. Especially in combination with mobile devices these mechanisms are of high value and are used to increase usability tremendously. Context data, usually identified as external factors from the network environment, can extend across a wide variety of different fields including for example: environmental conditions, time, location, available energy, network attachments points, channel conditions, and communicating sources and destinations. Moreover, human related social states can be part of the analyzed environment, for example, if a user is in meeting, busy, or free.

The MobilityFirst future Internet architecture project represents a clean-slate Internet architecture which provides the necessary abstractions for creating and managing context-aware services. In particular, the architecture enables dynamic identification of endpoints based on context attributes through the use of named-object identifiers and global name resolution [39]. The current Internet primarily supports a primitive to send data to a specific IP address, which limits applications to cast all communication intent in those terms. This primitive is inflexible when the network location of the destination (or even the principals constituting the destination) is not known a priori. For example, several mobile or Internet-of-Things applications can benefit from context-aware primitives such as send this message to all taxis in the Times Square area or request power consumption readings from devices in my living room, which are cumbersome to implement in IP. In contrast, MF enables context-aware communication primitives based on attributes more general than just the network location and dynamically associates a context identifier to its constituent principals.

Our strategy [58] is to develop an architecture where we can name environmental contexts that change where and how messages are routed and delivered. Application services could expect from such architecture improvements that span areas including security, communication efficiency, and energy management. In order to do so we identify three strategic mechanisms that are required to accomplish the set goals: an easy way to specify context based on human understandable techniques, i.e. use of names; an architecture supported management mechanism that allows both to conveniently deploy the service and efficiently provides management capabilities; and a native delivery system that reduces the tax on the network components and on the overhead cost of deploying such applications.

3.3.1 Context Service in MobilityFirst

While the MobilityFirst architecture provides the necessary abstractions for creating and managing context-aware services, new mechanisms are required in order to fully exploit them. Starting from the three strategic mechanisms defined, i.e. an easy way to specify context, an architecture supported management mechanism and a native multi-point delivery system, three fundamental technology components are exploited to design the MobilityFirst based context services framework.

Global Name Resolution Service: Recall that in MobilityFirst, the GUID represents an abstract endpoint that is independent of the network topology. We leverage this independence to build contextual networking; our approach is to overload the GUIDs at many levels of the network. A GUID could thus represent many abstractions; for example, a cell-phone, a person, or a group that is defined by context. In this sense, contextual communications are similar to *-cast types networking. For example, we could overload a GUID that names sending a message to a meeting. The system would have to first translate the GUID into a list of each person in the meeting. Further translation would be needed to identify the device each person is currently using. The Global Name Resolution Service is then the first key enabling technology that allows us to define context by collecting common set of network entities under a single name defining the context.

In-network computing capabilities: To enable future extensions to the network protocol without expensive hardware replacements and disruption, MobilityFirst builds in an optional and dynamically pluggable *compute plane*, as presented in Section 3.2. In-network computing capabilities are perfectly suited to deploy distributed and scalable management mechanisms to collect contextual information and manage membership. This last step is fundamental to the success of the architecture; for this purpose in-network deployed services interact with the GNRS to translate collected contextual information into context names. In order to best support the distributed nature of the in-network components, service addressing is of fundamental importance. Generally, the compute layer will be hosted at key locations, whereas network delivery mechanisms will allow to select the best located replica of the service.

Multicast Delivery: Multicast protocols have been long studied in the literature and different protocols and architectures have been proposed to support this type of delivery mechanisms. While these mechanisms are perfectly suited for static, tree based communications a different approach might be required for contextual applications. In particular, given the highly dynamic nature that defines context services, due to their multiple evolving factors, more flexible multicast delivery mechanisms are required. As importantly, in wireless environments, there is a desire to take advantage of inherent multicast/broadcast medium to enable efficient point to multi-point delivery services. The MobilityFirst architecture solves this problem by implement a lightweight multicast delivery system that, through name grouping in the GNRS, limits per group state within network elements taking per hop decisions on multicast splitting based on *Longest-common (LC)* lookahead techniques (see Section 3.1).

3.3.2 Emergency Service Demo

We use the developed MobilityFirst prototype to implement a context services framework designed around the one described in the previous section. The GNRS and the native delivery mechanisms, together with the newly implemented in-network context service, enable the MobilityFirst architecture to efficiently deploy context based services improving current architectures in a variety of ways. In order to demonstrate the mechanisms involved in implementing context based services and to showcase some of its benefits, we developed and deployed a contextual application that implements an alert system for vehicles assisting first responders. This service is aimed at providing ways to quickly and reliably transmit emergency messages to group of receivers identified by the conveniency of their geographical position or by the autoritative importance in the emergency matter. To better understand this consider an example where a car accident occurs on the roads of a metropolitan area; in this case, 3 potential candidates emerge for providing assistance: first, defined by a very small geographical area, other close passbyers could be informed in order to provide first assistance; second, based on a larger area, emergency vehicles such ambulances could be alerted; finally, given the importance of informing the central authorities, a police station could be included due to its relevance on emergency matters.

The system architecture deployed, as depicted in Figure 6.3.3, includes, together with the defining technologies of the MobilityFirst architecture, two core components: the in-network compute management framework and an Android based application.

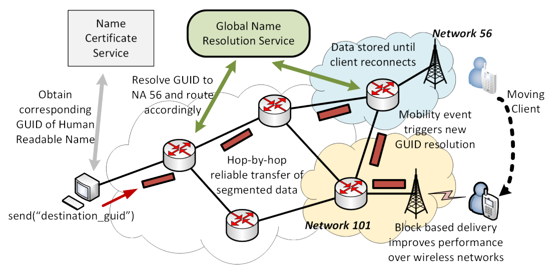

Name Certificate Service
Global Name Resolution Service
Network 56
Data stored until client reconnects
Obtain corresponding GUID of Human Readable Name
Resolve GUID to NA 56 and route accordingly
Moving Client
Mobility event triggers new GUID resolution
Hop-by-hop reliable transfer of segmented data
send("destination_guid")
Network 101
Block based delivery improves performance over wireless networks

Figure 3.14: MobilityFirst architecture design and late-binding example.

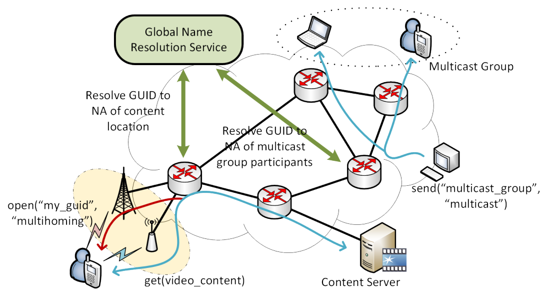

Global Name Resolution Service
Multicast Group
Resolve GUID to NA of content location
Resolve GUID to NA of multicast group participants
send("multicast_group", "multicast")
open("my_guid", "multihoming")
get(video_content)
Content Server

Figure 3.15: Service abstractions provided via the client API.

In-network compute management: A *Ruby* based context service is implemented to enable in-network computing capabilities that provide the distributed service described in the previous section. This service is then deployed in proximity of favorably located routers by directly connecting to the Routers API that provides access to the in-network compute capabilities. Context based operations are provided using web based APIs (i.e. using REST) and provide the core required management functionalities to create and manage context groups and report context information (e.g. users current location). Moreover, to implement the required management operations, the GNRS API is exploited to interact with the distributed service. A web interface is also developed for human based interactions to better understand the presented demo.

Contextual application: We exploit the available network host stack and API to deploy the context application components. In particular, an Android application has been developed providing two groups of functionalities: first, the core alert operations that allow for quickly broadcasting the alert message to the current context and to generate notifications for received emergency messages. Second, management operations are allowed, providing ways to interact with the service management framework and create, join and leave given contexts, by exploiting locally collected GPS coordinates.

The implemented prototype has been presented in the form of a demo [58]. The demo was be centered around three key operations: 1) creation and management of context groups based on geographical location of active users (Figure 3.3.2); 2) dynamic creation of routing information for multicast based delivery (Figure 3.3.2); 3) evaluation

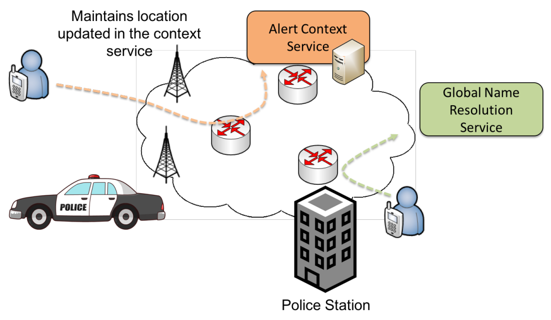

Maintains location updated in the context service
Alert Context Service
Global Name Resolution Service
POLICE
Police Station

Figure 3.16: Alert system architecture based on the MobilityFirst context services framework. Creation and management of context groups based on geographical location

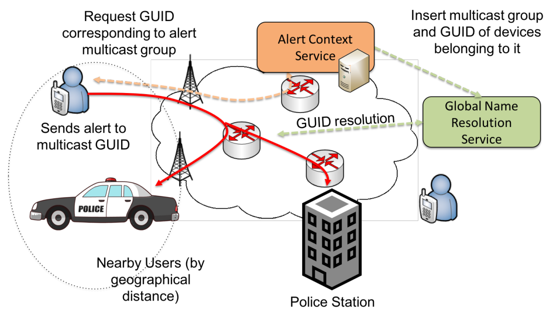

Request GUID corresponding to alert multicast group
Alert Context Service
Insert multicast group and GUID of devices belonging to it
Global Name Resolution Service
GUID resolution
Sends alert to multicast GUID
POLICE
Nearby Users (by geographical distance)
Police Station

Figure 3.17: Alert system architecture based on the MobilityFirst context services framework. Dynamic creation of routing information for multicast based delivery

of efficiency in providing the service. The demo will use the GENI nationwide testbed infrastructure [59] to deploy 3 different locations from where wireless nodes (i.e. Android phones) access via WiFi and WiMax the MobilityFirst architecture. Our current deployment spans 7 GENI sites across the US. 14 Xen VMs (2 VMs per site) each with 1 GB memory and one 2.09 GHz processor core provide us with the possibility to run one router per location and use the other node for application or services. All routers have a core-facing interface connected to a layer-2 network that connects all seven sites. This was setup using a multi-point VLAN feature provided by Internet2's Advanced Layer-2 Service (AL2S).

In order to better understand all the dynamics involved in the demo, a visualization system has been developed using web-based technologies (e.g. javascript, Google Maps, etc.). This visualization will provide information about different components state. In particular it will show the following information: location of mobile devices and their current status (idle, sending alert, alerted by other nodes), GNRS entries that enable context management and multicast routing, traffic crossing all the nodes in the system and finally important general events spanning all the networking layers of the architecture.

Chapter 4

Named-Object Based Services for Clients

As we transition to name based networking, new opportunities arise for end-hosts to evolve their role in the network. We then re-design two of the fundamental technologies that are used by these systems to participate in network communications:

- A network API that supports name based communications and advanced services.
- A design and implementation of a new transport protocol for an name based architectures that are able to exploit names to support new end-to-end services.

4.1 A Flexible Transport Layer

The TCP/IP architecture underpinning the current Internet is based on the end-to-end principle [60] of minimizing functionality in the network while handling service-specific requirements such as error and flow control at the end-points. In addition, the current Internet architecture is based on the concept of routing between IP addresses requiring a static one-to-one association between hosts and network locators. While the Internet works well for traditional kinds of communication, emerging mobile content and Internet-of-Things (IoT) services have motivated consideration of clean-slate *Information-Centric Network* (ICN) architectures [61, 62] which operate on names rather than addresses. Several distinct architectures for ICN have recently been proposed including MobilityFirst (MF) [39], Named Data Network (NDN) [63], and XIA [42]. While there are differences in detail, all the proposed ICN protocols share some common design elements that need to be considered in the design of transport protocols to be used for end-to-end services. Specific characteristics of ICN include: (a) use of names to identify sources and sinks of information; (b) storage of information at routers within the network in order to support content caching and disconnection; (c) multicasting and anycasting as integral network services; and in the MF case (d) hop-by-hop reliability protocols between routers in the network. These properties have significant implications for transport protocol design since the current protocols, TCP and UDP, were designed based on the end-to-end Internet principle, which typically assumes end-to-end connectivity during a transfer and uses address based routing with minimal functionality (i.e., no storage or reliability mechanisms) within the network.

Consider first the implications of name-based routing on transport protocol design. Communication with named objects, whether content files, devices, groups of devices or more complex context-based groups is different from conventional TCP connections in the sense that an object may have multiple end-points because the object may be multi-homed (i.e., multiple network interfaces to the same device) or multicast (to multiple devices, each with a different network interface) or multi-copy (i.e., multiple instances of the same information object can be found at different places in the network). This indicates that transport protocols need to be designed to provide appropriate service semantics for retrieving or delivering such named objects, for example, in multicast where the information object reaches all the named destinations or anycast where the object is fetched from the "nearest location" [64]. A second important property of ICN protocols is the fact that routers may store information objects such as content either for caching or for delay tolerant delivery. This implies the existence of in-network transport proxies which are in between the source and the destination, and the transport protocol should be designed to take advantage of the in-network copy to provide the desired service efficiently. For example, reliable delivery with an ICN transport would be able to utilize a copy of the information object stored at an intermediate router and avoid the need for end-to-end retransmission used in TCP. The third feature of ICN architectures is the fact that in-network storage can be associated with reliable hop-by-hop transmission of information objects between routers, thus alleviating the need for strong reliability mechanisms at the transport layer depending on the type of service desired.

Our contributions [65] in designing and implementing transport protocols for an

Service scenarios	Fragmentation & re-sequencing	Reliable delivery	Light transport	Flow/congestion control	In-network proxy
Large file retrieval	✓	✓		✓	✓
Web content retrieval	✓	✓		✓	
M2M communications	✓		✓		✓
Multicast	✓	✓	✓	✓	✓

Table 4.1: Different service scenarios' transport requirements

ICN architecture with explicit locators, such as MF, are two fold: (i) we examine a representative set of delivery service scenarios, and based on them, define the requirement space for transport protocol for any Information Centric architecture; (ii) unlike ICN architectures with no explicit locators, such as NDN, locator-based architectures enable richer end-to-end semantics, such as reliability delegation, and in-network retransmission. We show that such features are conducive to supporting mobility, and flexibly and robustly supporting different delivery patterns, through a prototype based validation with large content and web content deliveries through wireless, and content delivery with client mobility experiments.

4.1.1 Requirements for transport layer service for ICN

We first consider four common service scenarios that arise in information dissemination. These are large file transfer, web content retrieval, M2M communication and multicast. Through systematic analysis of these use cases, we identify the set of transport layer features for an ICN environment to support each of these scenarios (see the summary of requirements in Table 4.1).

Large file retrieval: A large file retrieval is abstracted as a *get*(content_name) socket call [32] in an ICN context. Clients inject a content request, independent of the content location, with such an operation, and the network will route the request to the location of the nearest copy of the content. After that, a flow with a large volume which carries the content is transferred reliably from the server to the requesting client. This is often referred as *anycast*.

Key TP functions required: Because of the large amount of data to be delivered, file transfer requires: (i) fragmentation and sequencing at the source, and reassembly at the sink; (ii) efficient usage of network resources with source rate control so as not to introduce congestion, both at the network layer along the path and at the destination. Reliable delivery, flow control, and congestion control becomes more complicated when the destination is connected to the Internet wirelessly, and especially when it is mobile. For instance, a wireless connection is susceptible to fading, may introduce random losses, and can typically provide a lower transmission rate than the nominal rate. Further, the imbalance of rates at different segments of an end-to-end path makes it difficult to perform end-to-end control at high speeds, with small amounts of buffering, and to deal with transient disruptions. This problem can be alleviated by providing additional in-network transport services, such as temporary storage for in-transit data (we call the en-route node with transport services a *transport proxy*).

Web content retrieval: In a web-browsing application, a sequence of content requests are sent by the client to the server. Each of the requests is for retrieving a constituent named object of a webpage. Two characteristics are inherent in web content retrieval: i) these requested objects are generally small in size, i.e. of tens or hundreds of kB; ii) user experience dictates that the objects must be received in a timely manner, preferably no more than several hundred milliseconds, thus making the transfer latency-sensitive. *Key TP functions required:* End-to-end error and congestion recovery need to be provided, but in a lightweight manner, because any significant setup overhead is not amortized easily. Flow control is not required because of the limited amount of data transmitted, in order to avoid unnecessary overhead contributing to increased latency.

M2M communications: In Machine-to-machine (M2M) communications, sensor data by nature is idempotent. That is, if the PDU is lost (due to bit errors or congestion) or it is delayed beyond the limits of latency for the data, the transport layer need not attempt to reliably deliver that PDU. This transfer paradigm is captured in a *send*(dst_name, content_name) API with no explicit reliability preference.
Key TP functions required: In such cases, the transport layer could simply resort to

stateless communication (e.g., lightweight transport with no error recovery and no flow/-congestion control) to minimize overheads. Moreover, due to power constraints in devices, a sensor node may not be on all the time. End-to-end control is not always possible in this case and delegation of transport service guarantees, such as reliability, need to be made to other en-route nodes. Thus in-network proxy support is desired.

Multicast: There are a number of popular applications that are based on multicast, such as group-based subscriptions (RSS), teleconferencing, online gaming, etc. In a name-based architecture, multicast can be realized with the same *send* (dst_name, content_name) API with the dst_name referring to a group of individual endpoints names. *Key TP functions required:* Guaranteeing 100% reliability in a multicast session is a well-known difficult problem. To achieve reliable transport, the source relies on negative acknowledgement (NACK) from clients to initiate retransmissions. With the number of subscribers increasing, retransmission has to be implemented in an efficient manner such that the ACK-implosion (see [66]) is avoided. This may require aggregation of retransmission requests in the network, and retransmission from within the network. Thus in-network proxies are desired to handle such aggregation and storage of pieces of contents for retransmission. On the other hand, when the multicast traffic has tight latency bounds, a lightweight transport protocol without end-to-end retransmission-based methods to recover from errors may be preferable, e.g. in a teleconferencing scenario, which may depend on application-level recovery and error-correction coding.

4.1.2 MFTP design

In this section, we introduce the design of the MFTP protocol. In particular, we explain how each key requirement, such as in-sequence delivery, end-to-end error and congestion control, in-network transport proxy and multicast delivery, is supported by the proposed transport layer components.

MFTP is based on the four characteristics of different ICN proposals mentioned in the introduction. At a more specific level, MFTP described in this paper has been designed to operate on top of the MobilityFirst networking stack [23, 31]. As described in [39], MobilityFirst is based on a clean separation of names and network addresses with a logically centralized but physically distributed global name resolution service (GNRS). The globally unique identifier (GUID) in MF is a flat public key identifier, i.e. a name, which can be used to represent any network attached object, including devices, people, groups, content, or context.

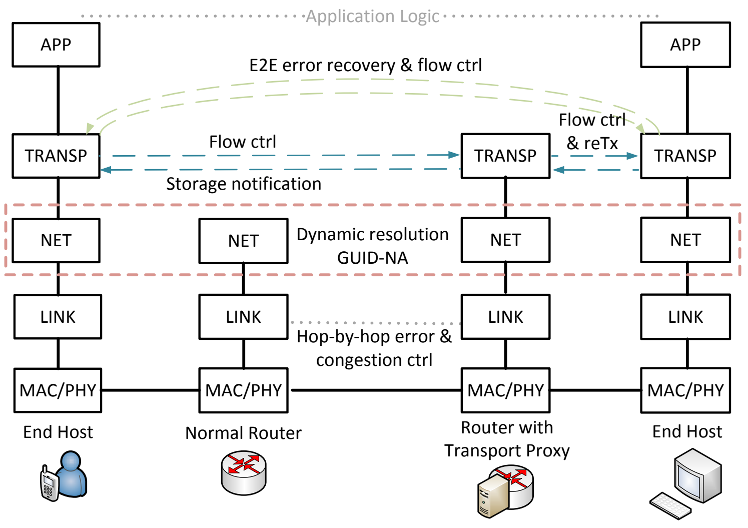


Figure 4.1: Protocol stack and transport layer functionalities

Fig. 4.1 shows the major layers in the MF protocol stack and the role of the MFTP transport layer above the named-object GUID based network layer which is supported by the GNRS [23, 24]. For additional details on MF, the reader is referred to the appendix of this paper, and [23, 31, 32, 39].

Segmentation, sequencing, and in-order delivery

A common characteristic across different ICN architectures is named network primitives, e.g. using names to identify a source, sink, content, or even a service. Typically in ICN, a request for a data is abstracted by an API, *get*(name, request, buffer, [opts])).

Depending on specific architecture, for instance, in NDN, each such request, namely interest, is marked with a relative sequence number, and solicits one segment of a content. In MF, requestor only sends one request for a piece of content; the server that handles the request segments the content and assign relative sequence numbers to the segments. In any case, sequence numbers are bound to the named content, rather than the two endpoints. This has significant implication for the hop-by-hop transfer and storage capability in ICN, as we shall see later when other design points are discussed. With data-centricity, such sequencing scheme works naturally for anycast and multicast. For example, in an anycast scenario, the forwarding plane decides where the content request should be handled. The transport layer is oblivious of the server location; rather, transport's functionality of providing ordering, and reliability can be fulfilled based on the knowledge of the data it concerns, using content names and sequence numbers.

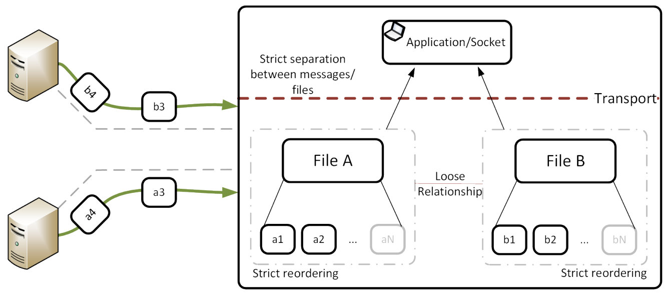


Figure 4.2: Illustration of transport layer fragmentation and sequencing. Transport layer fragments a content to be delivered into large blocks of data, i.e. chunks. Sequential delivery guaranteed for each content, but no strict ordering maintained for chunks belonging to different contents.

Consider Figure 4.2 as an illustrative example of this change; in this picture we show how the transport layer would support the reception of files from different data sources. As shown in the figure, in-order delivery is strictly enforced among the chunks of a single transported file. This suggests transport will buffer out-of-order chunk arrivals while waiting for the missing chunks. On the other hand, only a loose relationship is maintained across multiple files, because each file has a unique name and there is no need for strict ordering of delivering the received content based on the order of the requests. Applications can optionally specify the additional requirement of enforcing ordered delivery across the different files. In general, the only relationship maintained across different files is based on the available resources: memory management has to be coordinated to guarantee that buffers are not filled exclusively with partial data blocking the completion of any single object.

On the sender side, segmentation of the files is performed following the characteristics of one of the building blocks for the MF network design: the reliable hop-by-hop transfer of large blocks of data called "chunks". Given this guideline, the transport layer fragments the application data received across the API into large chunks of data, whose size can be negotiated by the two end-points based on a tradeoff between the overheads and the fair use of network resources across flows. We allow the chunk size to go up to the order of megabytes to take advantage of high-bandwidth communication channels. The link layer fragments a chunk to meet the link MTU requirement, but still logically maintains the semantics of a "chunk" at each hop. The same concepts of segmentation and sequential delivery can be applied to multiple semantically different objects such as application layer messages, contents or service requests.

Coordinated End-to-end error recovery and hop-by-hop reliable delivery

Most transport protocols including TCP, operate on an end-to-end basis. Congestion control and reliable delivery are between the two endpoints. Detection of loss (whether due to errors or congestion) or congestion at a link is communicated to the sender after a feedback delay, possibly quite a few end-to-end RTT's. After the detection, recovery mechanisms, such as congestion recovery or retransmission, can incur an unduly large penalty to the flow. Also, due to queuing at routers, and heterogeneous transmission technologies employed along the route to destination, spurious, or premature, retransmissions are not uncommon [67]. A more efficient way to recover from congestion or error happening at a particular link is through link level mechanisms. This yields two benefits: i) congestion and errors can be detected and reacted upon more quickly; ii) reduces the possibility of spurious retransmissions. Hop-by-hop transfer is suitable for ICN due in part to the fact that the segment of data being transferred is named; moreover, the transfer is not connection-oriented; routers have storage capabilities. In NDN, each Data is indeed transferred in a hop-by-hop manner: upon receiving a Data from the upstream, the router examines whether an interest for the data has been seen before, and whether it needs to cache the data. MF transport does this by relying on per-hop error recovery and congestion control whenever the problem can be resolved locally, and only invokes end-to-end mechanisms when it is absolutely necessary, e.g., a router fails and loses all the buffered data. On each hop, after every chunk that is transmitted, a corresponding control message called $CSYN$ is used to explicitly request acknowledgement from downstream, which then replies with a bitmap of reception status for every packet in that chunk. The transmission for this chunk finishes if there is no loss, otherwise the lost packets of that chunk are retransmitted locally following the same procedure until all packets are received.

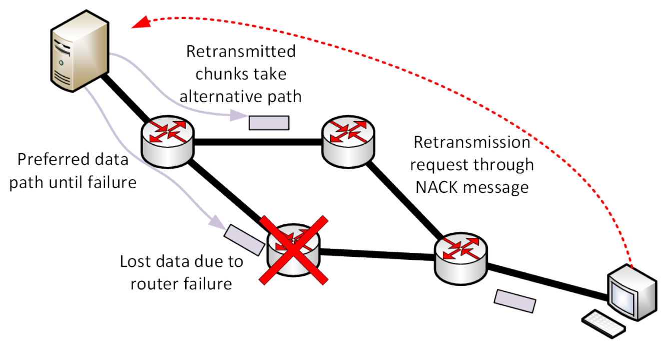

Retransmitted chunks take alternative path
Preferred data path until failure
Retransmission request through NACK message
Lost data due to router failure

Figure 4.3: End-to-end signaling to recover from in-network failure

Taking advantage of the hop-by-hop reliability of the network, we seek to have a parsimonious end-to-end mechanism that has minimal overhead (important in mobile wireless environments) while primarily seeking to recover from node and link failures. The end-end error recovery mechanism is built to be *flexible* to accommodate application and sender needs (including *don't care, NACK, ACK*). With a Negative-ACK, i.e. NACK, the transport reduces end-to-end message overhead, and the receiver provides notification only when a chunk is not delivered over a conservatively long period of time (as a result of a failure that causes the reliable hop-by-hop mechanism to lose an acknowledged chunk as shown in Figure 4.3). It is only for short-sessions (e.g., single PDU delivery) and for latency-sensitive interactions that the sender would enable the use of an end-to-end ACK option. With idempotent data transmissions (e.g, sensor data which the transport layer sends and forgets), the sender may choose to use the *don't care* option.

This NACK-based design also addresses possible ACK-implosion problems [66] that may arise in multipoint deliveries: if an ACK is used, then for each data multicast chunk, there will be as many acknowledgements as the number of receivers in the multicast group. This will be discussed in a later subsection.

In-network transport proxy

One of the challenges for conventional transport protocols is dealing with the delivery of large content to mobile devices, where mobility results in intermittent connectivity and the end-to-end connection experiences frequent disruptions. If the transport protocol has to re-establish the connection, then the transfer has to re-start and any data already in transit in the network will have to be discarded. In ICN, as each data transfer centers around the named data being requested, instead of the connection that delivers the data, client mobility is inherently supported. For instance, in NDN, each data is solicited by an Interest; in the case of client moves before obtaining the requested data, it can re-issue an Interest for the same data, which will be delivered to the new location.

To this end, we postulate having routers (or at least a subset of them) which provide in-network transport service such that the original source can delegate part of the end-to-end data transfer responsibility. The router, which we call an *in-network transport proxy*, would have substantial amounts of memory, e.g., several GB, to temporarily hold in-transit chunks when the destination is unreachable. This disruption may be due to: lack of connectivity to a mobile destination node, until connectivity is subsequently re-established; alternatively, in M2M communication, when a sensor node is only powered on intermittently, it may choose to deliver information chunks to the next hop and then power down.

The mechanisms implemented by such a node are shown in Figure 4.4: when faced with the impossibility of forwarding chunks with the information available at the network layer (i.e. the router detects that connectivity towards the destination of a chunk

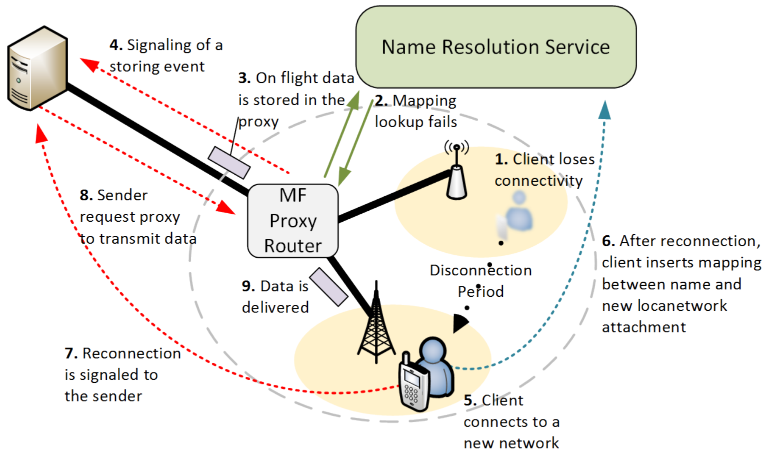

Name Resolution Service
4. Signaling of a storing event
3. On flight data is stored in the proxy
2. Mapping lookup fails
1. Client loses connectivity
MF Proxy Router
8. Sender request proxy to transmit data
9. Data is delivered
Disconnection Period
6. After reconnection, client inserts mapping between name and new locanetwork attachment
7. Reconnection is signaled to the sender
5. Client connects to a new network

Figure 4.4: Procedures involved to use in-network transport proxy to handle destination disconnection and retransmission: the proxy temporarily stores chunks when the destination disconnects, and transmits to the client when connectivity is restored as indicated by the name resolution service.

is disrupted), the router pushes up to the transport proxy layer the relative data chunks. Two reasons might generate this impossibility of forwarding chunks: (i) the destination does not have an active network address (NA) binding corresponding to its GUID entry in the GNRS; (ii) the chunk reaches the destination network given by its most recent binding, but either the destination has changed its point of attachment or it has disconnected from the network before the previous NA entry expires in the GNRS server. As a consequence, the link layer is not able to deliver the chunk despite several attempts, and corresponding CSYN timeouts. In these cases, the chunk is pushed up to the proxy layer to be temporarily stored. While this is similar to Delay-Tolerant Network protocols, the innovation here is the integration of these mechanisms with the support of dynamic mobility and ICN style named object services.

We limit the amount of content that can be stored for a flow. Each (source GUID, destination GUID) pair is limited to have buffered content up to size S. When a chunk for a new flow arrives, the chunk will be stored directly if sufficient space ($> S$) is available for the new flow; otherwise, a chunk for the oldest flow is replaced from the storage by the new chunk. When the chunk is stored, a timer is created to schedule future transmission. Further, a transport layer message, either *Store* or *Drop*, is transmitted back to the original source to notify it of the intermediate proxy storing or dropping the chunk. A stored chunk will be scheduled to retry a GNRS lookup to bind an updated NA to the destination GUID when its storage timer expires. The chunk will be pushed out if an NA is found, i.e. destination becomes connected again, otherwise will be kept in storage. On the other hand, rescheduling of the chunks can also be initiated by the original source of a chunk. As is shown in Fig. 4.4, when the source receives a NACK message identifying a chunk as missing, if it is aware that the corresponding chunk originally destined to the requesting destination is stored in the network, based on a previously received *Store* message, it utilizes this in-network copy and initiates the retransmission from inside the network. This is done by the source sending a *Push* message to the in-network proxy to trigger retransmission.

Flow control and congestion control

With the transport using hop-by-hop reliable delivery, we use a combination of hop-by-hop back-pressure for congestion control and end-to-end window-based flow control for the congestion and flow management components of the MFTP.

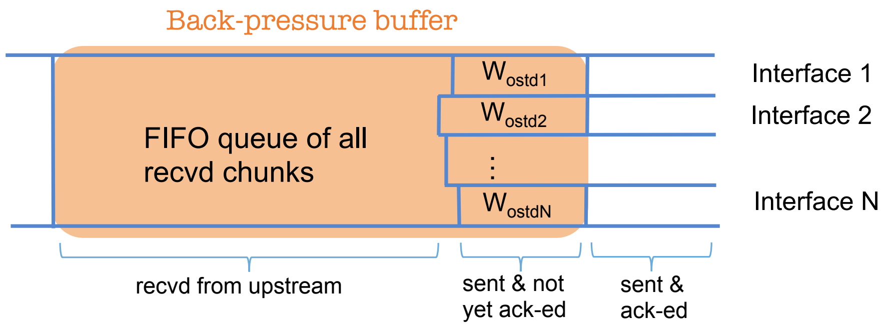


Figure 4.5: Back-pressure buffer and per-hop sending window.

Hop-by-hop congestion control: The hop-by-hop back-pressure scheme is built on top of a back-pressure buffer (of capacity B packets), that is at every MobilityFirst router. As illustrated in Fig. 4.5, the back-pressure buffer essentially has all the chunks that are received from the network and are queued to be transmitted. In addition, between two adjacent routers on a link, the sender maintains a sending window W_{ostd}, i.e., number of outstanding packets, that is bounded by the receiver's advertised window, W_{ad}. As mentioned before, following the transmission of a chunk of data, a CSYN message is sent, which the downstream node then acknowledges with a CACK message. The receiver advertised window is piggybacked in the CACK. The number of outstanding packets, W_{ostd}, is reduced based on the downstream node's acknowledgement. Also, whenever the router schedules to transmit on a particular outgoing interface, it attempts to transmit as many packets as W_{ad} allows.

When the occupancy of back-pressure buffer reaches its capacity, the router drops all incoming data packets and does not acknowledge reception. Furthermore, it throttles the advertised window to all of its upstream nodes. This "congestion signal" eventually propagates back to the original traffic sources in a hop-by-hop manner, thus eventually limiting the traffic injected into the network. When the congestion is released downstream, data already queued at the upstream router will immediately get transmitted.

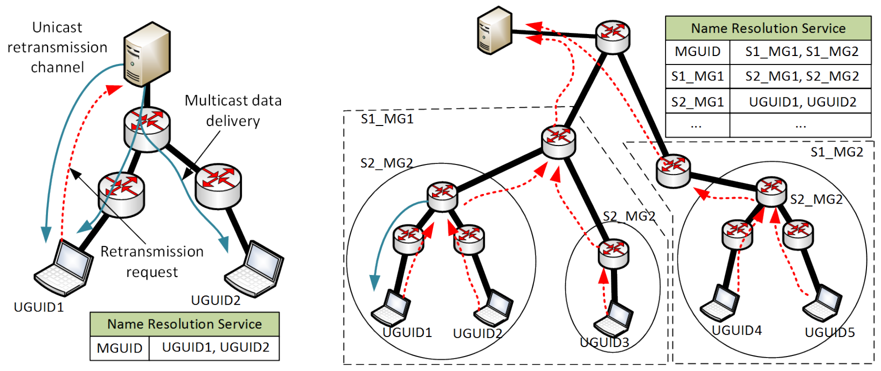
End-to-end flow control: Hop-by-hop back-pressure deals with congestion at each router, but is not sufficient to prevent the receiver's buffer from being overrun by the sender's data from an end-to-end perspective. Especially when the size of the transferred content is large, and the hop-by-hop transfer delivers data to the receiving node's transport at a high rate, end-to-end flow control is indispensable. Because MFTP does not require the receiving side to send frequent reception status update in the reverse path (it depends only on NACKs), the feedback from the receiver is both parsimonious and not timely for the sender to detect receiver buffer overflow. We therefore consider an explicit notification from the receiver. The sender starts at a initial end-to-end sending window W_e that is either provided by the receiver in the request, if the receiver proactively requests the data, or a predefined value that sender considers appropriate in a pub-sub scenario. For each window's worth of data chunks, the receiver then sends one window flow control message, to advise the sender to maintain, increase, or reduce the sending window to certain value based on the receiver's buffer occupancy. This message will be delivered reliably to the sender. Thus, the sender will not send more until a window feedback message is received. Note that the sending window is also the atomic unit for the end-to-end NACK message, thus the NACK and flow control are fulfilled by a single message (if a NACK has to be sent, i.e., some chunks are lost). This can be implemented by sender explicitly marking a chunk as the 'end-of-sending-window', to request a NACK and flow control feedback. In the event that this special chunk is lost due to a node failure, a NACK timeout at the receiver would trigger the receiver to proactively notify the sender of the reception status (NACK) and receiver buffer status (flow control).

Small content transfers are not subject to such end-to-end flow control, mainly because the content usually is composed of a single chunk and the transfer will be complete even before the flow control notification can be generated. However, small content transfers are still regulated by per-hop congestion control.

Alleviating head-of-line blocking due to hop-by-hop transfer: A drawback inherent with hop-by-hop back pressure is the unfairness caused by head-of-line (HOL) blocking with FIFO queueing [68]. Consider a chunk at the head of the queue blocked from being transmitted by a back-pressure signal from the downstream node. This can prevent chunks behind it in the queue that is destined to a different destination that is not experiencing congestion. An alternative to having HOL blocking is to drop the chunk being back-pressured, but this has undesirable consequences of requiring retransmissions when a temporary buffering could overcome the short-term congestion. Theoretically, per-flow queuing solves this problem, but scheduling with per-flow queues is difficult to scale and is impractical with large numbers of flows. However, the in-network transport proxy provides some relief to this situation and alleviates the short-term unfairness. If a back-pressure signal is received for the chunk at the head of the sending queue, the transfers of chunks destined to other nodes will thus not be blocked because chunk at the head of the queue will be removed and pushed up to the transport proxy layer for temporary storage. The transport proxy will then attempt to transmit that chunk when the storage timer expires (or is dropped if the chunk is replaced in the storage buffer because of the replacement policy we described above).

Multicast

Multicast is a delivery service naturally supported by name based architectures. In MobilityFirst, a dynamically formed multicast group is identified by a Globally Unique Identifier (GUID), which can be mapped into a set of individual clients' GUIDs or Network Addresses.



Name Resolution Service	
MGUID	UGUID1, UGUID2

Name Resolution Service	
MGUID	S1_MG1, S1_MG2
S1_MG1	S2_MG1, S2_MG2
S2_MG1	UGUID1, UGUID2
...	...

Figure 4.6: Multicast data delivery, small scale (left), large scale (right).

We first analyze a small scale scenario as depicted on the left side of Figure 4.6; in this case only a handful of nodes are receiving the multicast data. During the transmission the source of the multicast data marks outgoing chunks with a multicast service identifier and selects as destination GUID the one identifying the multicast group. This approach is the basis of the retransmission mechanism that allows us to achieve reliable multicast communications: multicast clients send NACK messages over a unicast channel and the multicast source can identify which multicast group a specific client belongs to. Further, the source aggregates retransmission requests for the transmitted chunks; it can, either employ multicast again for retransmission when the number of requestors exceeds a threshold; otherwise retransmitted data chunks can be sent using unicast destination GUIDs that identify the specific nodes that need the retransmitted data.

A similar concept can be used to scale this approach to a larger group of multicast destinations. While for the smaller scenario we used one level of *multicast group GUID to a set of GUIDs* mapping for retransmissions, as the number of participants increases we can exploit in-network transport proxies to build multiple levels of such mappings, in a recursive manner. This scenario is represented on the right side of Figure 4.6. Each participating transport proxy provides aggregation of retransmission requests that can be locally fulfilled using stored data. If the corresponding chunk is not available at the local storage, the request is propagated backward towards the source. In order to limit potential explosion of unfulfilled requests reaching the original source, transport proxies can be instructed through proper chunk marking, to discard retransmission requests that exceed a number of traversed proxies without encountering the missing chunks.

In a scenario where reliability is not demanded, the source just use the *don't care* option of the reliability preference as described in section 4.1.2 and the forwarding of multicast data can be carried out using similar recursive approach as the reliable delivery case.

4.1.3 Implementation

Our implementation of MFTP consists of two parts: end-system transport operations that are implemented on the MobilityFirst client stack [32], and an in-network transport proxy implemented as a pluggable module inside a MobilityFirst Click router implementation.

Host Stack and API: The client host stack has been implemented on Linux as a user-level process built as an event-based data pipeline. Apart from the MF transport protocol, the stack contains a name-based network layer and a reliable link layer with large chunk transfer. Applications interface with the host stack through socket APIs that are available as a linkable library and include the primitives *send, recv*, and *get*, and a set of meta-operations. Examples of meta-operations include those to bind or *attach* a GUID to one or more NAs. By specifying the options field in the API call, an application is able to configure transport parameters such as the i) desired chunk size; ii) end-to-end reliability preference; iii) NACK timeout; iv) willingness to use in-network proxy.

Router: The MobilityFirst software router is implemented as a set of routing and forwarding elements using Click [69]. The router implements MFTP transport proxy layer, MF network layer including intra-domain routing and dynamic binding using GNRS, and hop-by-hop reliable transfer. The transport layer (proxy) interacts with the intra-domain route look up component: if a lookup does not yield a valid next hop, the chunk is pushed up to the transport proxy. The transport proxy at the router will hold the data chunk for some time and attempt to rebind the name with one or more network addresses. When rebinding is successful, the chunk is pushed back down to the routing layer for forwarding.

Timers: There are two types of timers used in our implementation: one for triggering the transmission of an end-to-end NACK message, one for storage. For guaranteeing end-to-end reliability, timers are indispensable because a node has to learn about a remote node's failure impacting the end-end path. Previous experience with TCP end-to-end timers have taught us that timers need to be set loosely so as to reduce number of false alarms [70, 71], and not have a strict dependence of the transport protocol on timers for normal operations. In MFTP's design, it is possible because a NACK timer is associated with a chunk of data, rather than a single packet. The storage timer is only used to locally retry lookups, with minimal overhead.

4.1.4 Case studies and evaluations

In this section, we present how MFTP can be used in several different service scenarios, and quantitatively compare it with the performance of conventional HTTP and IP based protocols.

General experimental testbed setup: We use the ORBIT [72] wireless testbed for our experimental evaluation. Each machine in our experiment is equipped with Intel i7 2.93GHz processor and with 8GB RAM. In terms of networking capability, each node has one Gigabit-Ethernet interface and one WiFi interface with Atheros ath5k wireless

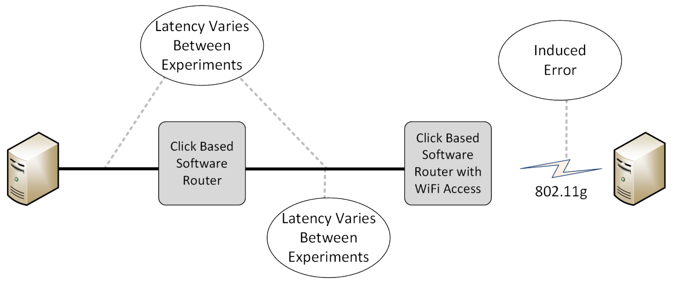

Latency Varies Between Experiments
Induced Error
Click Based Software Router
Click Based Software Router with WiFi Access
802.11g
Latency Varies Between Experiments

Figure 4.7: Experimental Setup

driver. Physically all the nodes are connected to a single layer-2 switch; we use VLAN tags to create desired topology to isolate Ethernet traffic. For wireless traffic, we use 802.11g with the data rate fixed at 54Mbps. Access routers are running hostapd [73] to operate as WiFi access points. We disable 802.11 authentication and use manual IP assignment (no DHCP), just to retain nearly the same amount of overhead with both MFTP and TCP for WiFi connection establishment. We considered a topology shown in Fig. 4.7, where a client, N_4 connects to a server N_1 through an access router N_3, which provides WiFi connectivity, and a regular router N_2.

Methodology: We evaluate three types of data delivery scenarios to compare MFTP with the current TCP/IP based architecture, in terms of the mechanisms employed, and their performance. We emulate the end-to-end RTT's of local, coast-to-coast and inter-continental communications, use the emulation tool *netem* [74] to add 10ms, 50ms, 100ms RTT between the two routers, respectively. To emulate loss in a controlled manner, we again use *netem* to introduce 1% loss. With MF, we run the MF Click router prototype (mentioned in section 4.1.3), and a local GNRS server on both N_2 and N_3. The MF client stack runs on N_1 and N_4. For specific use cases, we run corresponding applications that interface with the client stack through the MF API. In the case of TCP-based experiments, we run Click IP routers on node N_2 and N_3. TCP segmentation offloading is turned off as the basic Click IP router drops TCP packets with size larger than 1500 bytes. We enabled manual Ethernet header encapsulation on the Click IP router so no ARP message is triggered during routing. On the two end nodes, the default version of TCP, TCP Cubic, is used. We configured both nodes' TCP receiver buffer to be 2MB, so that it is not a bottleneck in a high delay-bandwidth path in any of the experiments.

Large content delivery over wireless

We first look into a large volume data transfer experiment. A 400MB file is requested and transferred. A simple file retrieval application in MF is running on the two end nodes. In the case of TCP, we used iperf to generate a flow of equal size with the maximum packet payload size of 1400 bytes. We repeated this experiment for a number of network conditions: RTT being 10ms, 50ms, or 100ms, and loss on WiFi link being 0 or 1%, to explore their effect on both architectures' goodput (i.e., application throughput).

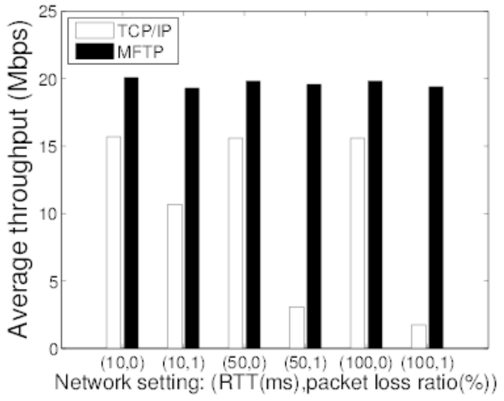


Figure 4.8: Throughput comparison. MFTP is robust in the presence of loss. Average throughput comparison for 6 different (RTT, loss rate) profiles.

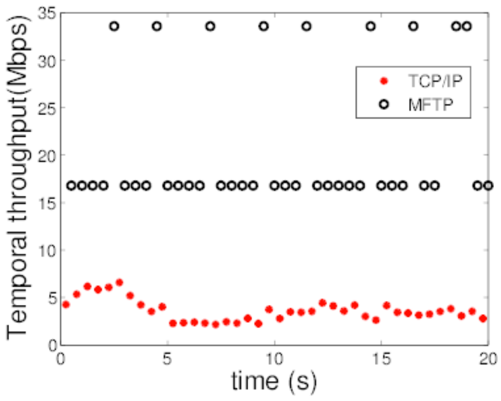


Figure 4.9: Throughput comparison. MFTP is robust in the presence of loss. Instantaneous throughput (per 500ms) for 50ms RTT and 1% loss.

Fig. 4.1.4 shows the average throughput comparisons for the six different network settings. Both MFTP and TCP' throughputs are consistently high when there is no loss, despite varying the end-to-end latency. MFTP is slightly higher in throughput in the lossless cases. MFTP is significantly more robust in the presence of loss, e.g., the throughput degrades by only 10% when there is 1% residual loss, with all 3 RTT profiles. On the other hand, TCP throughput drops significantly when there is loss.

For instance, with 50ms RTT, TCP throughput with loss drops to only a quarter of its throughput in the lossless case. Fig. 4.1.4 shows a plot of instantaneous throughput (averaged per 500ms) for 50ms latency and 1% loss. MF's PDU is a chunk of data, and in every 500ms, it receives at least one chunk (1MB), even in the presence of loss. With TCP, throughput fluctuate around 5Mbps. This is because the end-to-end congestion window is throttled whenever loss is detected. This misinterpretation of loss unrelated to congestion unnecessarily penalizes the flow. With MFTP, loss is not considered a signal for congestion, thus the sending rate is not throttled; moreover, loss happening at the last hop is recovered locally. Note in this experiment, the client suppresses the NACK messages because all the data has been successfully received.

Web content retrieval

Web applications are ubiquitous and used in many different contexts. Originally developed to reliably transfer Web pages, HTTP is now used for many different applications such as video streaming and content delivery. While originally intended as transport protocol agnostic, with the only specified requirement of reliability, the predominance of TCP has caused the HTTP protocol to evolve to version 1.1 with features that were designed specifically for TCP's characteristics, e.g., persistent connections. This tight coupling between the two protocols has caused different performance issues, especially with the recent increase of web access from mobile devices. These performance issues include: delays derived from connection establishment (still occurs despite persistent HTTP, with objects being located at various locations), and head of line blocking (happens when multiple objects are fulfilled by a single TCP connection).

We use the same topology as described before to compare the two alternatives. In addition to the routers, we run an Apache server (version 2.2.22) on node N_1, and a web browser emulator on node N_4 which requests webpages. We reuse the browser emulator, *epload*, presented in [75]. We also download the dataset introduced in [75] which consists of the real webpage objects of the 200 most accessed websites recorded by Alexa [76] in 2013. Among these we randomly select 40 pages and place them on N_1 to be hosted by the Apache server. In each run of the experiment, the browser emulator

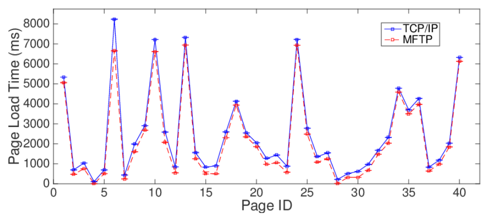


Figure 4.10: Page Load Times (min, average, and max of 5 runs) for 40 different webpages. 50ms RTT, no loss

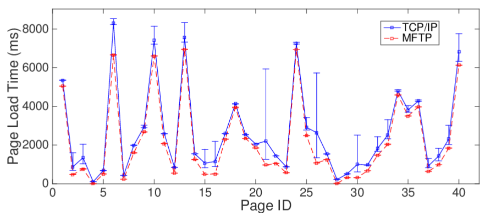


Figure 4.11: Page Load Times (min, average, and max of 5 runs) for 40 different webpages. 50ms RTT, 1% residual loss

opens up 6 concurrent TCP connections (default settings in most browsers [75]) and sequentially request the 40 webpages. For experiments with MFTP, in order to keep the modifications at the application end-hosts to a minimum, we developed an MF-HTTP-MF proxy whose main job is translating HTTP request and responses into MobilityFirst content requests and messages and vice-versa. We colocate 2 instances of these proxies with the HTTP components of the system, i.e., on the web client and on the server. For both MFTP and TCP, we performed 5 runs of the experiments with end-to-end RTT of 50ms, and 0 loss or 1% loss.

Figures 4.1.4 and 4.11 are the plots for average page load times (PLT), i.e. the time between emitting the first HTTP request to reception of the last byte of last object, for the experiments with 50ms RTT. Page load time with MFTP is consistently lower than TCP. In the case of no loss, when there is a smaller amount of data to be transferred, e.g. page 21, 22, and 23, with TCP the PLT is about 30% higher than with MFTP. The difference in PLT can be attributed to several features of MFTP: (1) MFTP is connectionless, and thus there is no overhead due to setting up a connection; (2) TCP identifies different requested objects by differences in sequence numbers of that connection, while MFTP differentiates each requested object by a unique name, therefore HOL blocking does not occur with MFTP; (3) each TCP connection "slow-starts", whereas with MFTP, short transfers, such as retrieving web objects, are not subject to flow control and are regulated only by per-hop back-pressure based congestion control, which allows sender to transmit at full rate as long as no congestion signal. As can be seen in Fig. 4.11, loss introduces a great amount of variability with TCP.

For instance, for page 21, the minimum PLT is around 1500ms with TCP, but the maximum is 6000ms, which is several orders of magnitude higher than with MFTP. For all the pages, MFTP maintains minimal variability in terms of page load time.

Late-binding and storage for disconnection

We evaluate the benefits of using in-network transport proxies for handling client disconnections in content retrieval. We consider the same topology as above. The end-to-end RTT is set to be 50ms, and no loss is added so that difference in performance would not be incurred by having different mechanisms for error recovery. We use *netem* to introduce 100% loss intermittently, so as to emulate client disconnections. In the experiment, WiFi connectivity is on for 10 seconds; then is turned off for d seconds; then the connection is restored. During the first 10 seconds of connection, the client requests a 10MB file at a random time. The experiment is repeated 30 times for both MF and TCP. We compare the distribution of file retrieval response times between MFTP and TCP.

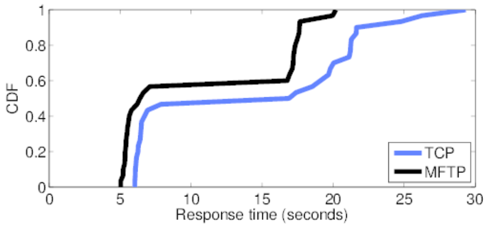


Figure 4.12: CDF of response times. With 10s disconnection

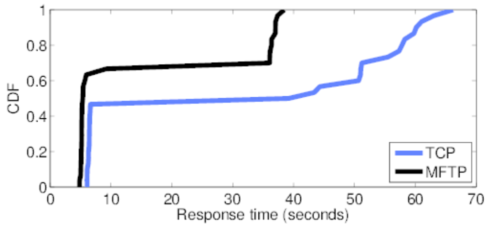


Figure 4.13: CDF of response times. With 30s disconnection

In Fig. 4.1.4, all the transfers having a response time of less than 10 seconds are completed before the disconnection. For the transfers that experience the disconnection, MFTP has at least 3 seconds lower response time (at 60th percentile). With 30 seconds disconnection, as shown in Fig. 4.13, the difference in response time is about 15 seconds at 70th percentile. It is worthwhile to understand the difference in the approaches taken by TCP and MFTP to dealing with disconnection. With TCP, the sender retransmits, based on a timer whose timeout value increases exponentially when the disconnection persists. In MFTP, the chunk in-transit is stored at the in-network proxy. A network address and next-hop lookup, rather than retransmission, is triggered when the storage timer for that chunk expires. Thus the transport proxy takes advantages of the global name resolution service in MF to learn whether there is a network address binding update for a client, and retransmits only when client is connected. This results in fewer retransmission attempts and more accuracy in the knowledge of end-to-end connectivity. Fig. 4.1.4 and Fig. 4.13 together suggest that MFTP's reduction in response time is nearly proportional to the length of the disconnection.

Chapter 5

Supporting Advanced Services Through Named-Object Based Network Virtualization

Network Virtualization has become over the years a fundamental technique can be exploited to implement advanced services and QoS oriented setups. Its value relies on four fundamental points: a) it allows for multiple logical networks to be deployed on top of a single physical network; b) it allows for use of commoditized hardware to overcome lack of resources, e.g. limited number of ports; c) drives experimental research via isolation of resources and providing different levels of abstraction on top of the common physical network; d) allows for Network Function Virtualization.

While these properties are very valuable and have driven the evolution of Virtual Networking technologies, we still have to live with a set of tradeoffs in designing these solutions. In particular, to support multi-domain network virtualization, the use of tunnels has been over-relied upon, providing different levels of abstraction, but introducing high levels of overhead. SDN solutions overcome this problem, but due to the fundamental technology they rely upon, they do not scale, especially when looking at inter-domain scenarios. For this reason they are usually mostly deploy within the constrains of a cloud network.

We propose a solution based on named objects and a dynamic name-resolution service for mapping names to routable network entities. Our system, exploits recursion in a name resolution service to implement a simple and clean design for Virtual Networks. Moreover, our system supports advanced routing algorithms that are able to exploit application state data to enhance advanced network applications. Finally, this solution can be integrated into larger systems that can support high performant future networking scenarios such as Cyber Physical Systems.

5.1 NOVN: A Named-Object Based Virtual Network Design

Virtual Networks (VNs) have been traditionally used over the years as a mean of connecting resources across the internet, creating the illusion of communicating on top of a dedicated network hiding the underlying physical infrastructure. Depending on the purpose, different techniques have been applied at different layers of the networking stack. Few examples come to mind: Virtual Private Networks (VPNs) have been extensively used to allow users to securely access a corporate intranet while located outside their offices or to securely connect geographically separated offices of an organization, creating one cohesive network. Layer 2 virtualization, enabled by 802.1Q VLAN tags, has been used to provide tools to simplify traffic engineering and differentiation in single domain networks. Even wireless network resources have been subject to virtualization, where collection of wireless access points behave as if they were only one - i.e. having the same MAC address and the same channel - so that wireless clients never need to do handovers.

Pushed by the ongoing advances in science and engineering that have largely improved computational capabilities allowing for a progressive softwarization of network technologies and equipment, Network Virtualization has been again adopted as one of the main enabling technologies across different scopes. Cloud networks have been one of its main adopters, whereas Virtual Networks techniques are used to abstract the distribution of resources - e.g. applications, data bases and more - across data centers, allowing for more flexible management techniques. For example, this approach was extensively exploited by NVP [77] to implement a network management system, within an enterprise data center. Attempts to extend similar concepts outside the limits of single data centers are ongoing, but have mostly focused on point to point connectivity between cloud locations [78].

With a consistently similar logical approach, Internet Service providers are using VNs to instrument their access networks with Network Function Virtualization (NFV). NFV provides ways for network operators to perform network functionalities using cheaper technologies, e.g. commodity hardware, or that would be otherwise too expensive to perform in hardware, e.g. processing of higher layers headers [79]. Within this context, Virtual Networks are used to connect, or chain in the NFV gergo, these resources on top of the physical network infrastructure allowing switching/routing fabric to quickly forward packets to the Network Functions invoked. The underlying virtualization techniques are similar to the ones previously presented, scaled to the access network environment.

Moving away from single domain scenarios, Virtual Networks have been adopted as the main tool for enabling networking research on a single physical network infrastructure. GENI [80] is one of the most recent efforts toward this goal. GENI's testbed provides access to routing and computing resources on top of a single nation-wide Layer 2 network, that is sliced using L2 VLANS to provide the perception of accessing a dedicated network. Multi domain testbeds have also been extended for this purpose: starting from the foundation provided by the PlanetLab testbed [81], VINI [82] developed an experimental framework where software routers are deployed on top of an overlay - i.e. on top of UDP - based virtualized network.

Looking at these different use cases, four evident limitations arise: 1) Most virtualization techniques are limited to single domain scopes, e.g. a data center or an access network. 2) When extended to support larger networks, they either need full control of the network environment, e.g. GENI, or 3) they rely on overlay solutions that ar costly due to the generated overhead and lack any access to the underlying network environment, e.g. VINI. Finally 4) all these solutions provide limited exchange of information between the virtualized environment, the applications that run on top and the underlying network, limiting the attractiveness for service providers to exploit network virtualization to enhance their solutions, where applications might benefit from affecting routing decisions based on custom metrics and cross layer optimization [83,84].

Acknowledging these needs we present *NOVN*, a Virtual Network framework that exploits the concept of Named-Objects to implement a logically clean, easily deployable virtual networking framework. Named-Objects are a powerful abstraction achieved through the use of a dynamic Name Resolution Service (NRS) for mapping names to routable network entities. Separating names (identities) from addresses has been advocated by the research community [3, 4, 39] for quite some time and has inherent benefits in handling mobility and dynamism for one-to-one communication. We extend this base concept to achieve the additional advantage of facilitating creation the new service abstractions. First, names can be used to represent many different Internet objects; for example, a cell-phone, a person, or a group of devices; the latter perfectly applies in the context of network virtualization, allowing *NOVN*'s solution bases its to define participation of network elements to the logical network.

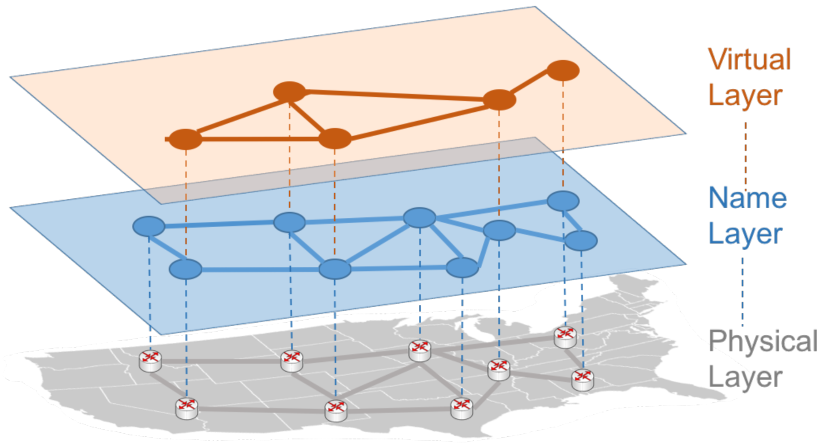


Figure 5.1: NOVN layers of abstraction.

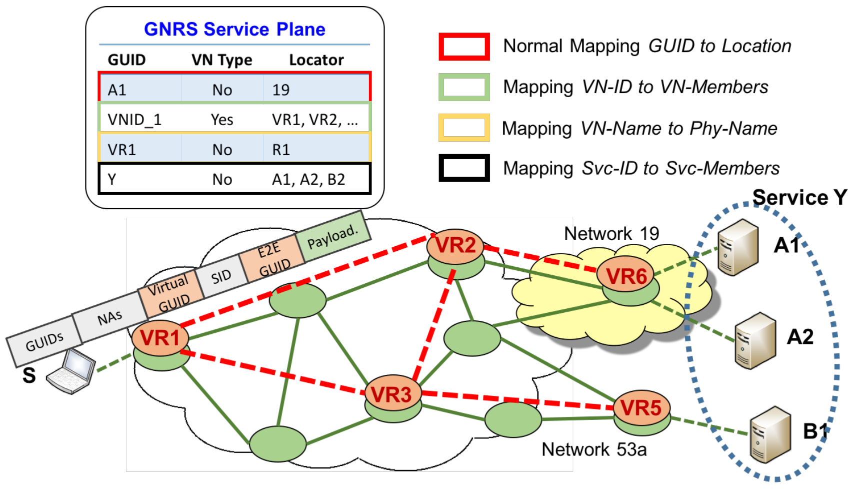
NOVN exploits Named-Objects to create clean partitions across logical layers, as shown in Figure 5.1. First, the NRS is used to map physical network resources to their names, eliminating the need of perpetually keeping track of routers addresses and possible configuration changes. A second layer of abstraction then maps network elements to the participants of the Virtual Network, creating a logical network on top of the infrastructure. As this network view is made available to all participants via access to the NRS, forwarding across domains is implemented through a form of tunnelling that uses names to address traversed routers on a hop-by-hop manner.

Building on top of this core framework, *NOVN* implements a novel technique that allows applications to push small snapshots of status data into the virtual routing fabric;

we call this concept Application Specific Routing. Imagine a mobile edge cloud scenario where the application goal is to connect mobile devices to the "best" cloud server. While in a normal networking environment "best" might correspond to the "nearest", in highly distributed environments varying computing loads might change this. Through ASR, *NOVN* supports this advanced *anycast* delivery service allowing virtualized routers to consider the application status and perform custom routing decisions. Inspired by active networks [85], ASR can be easily implemented exploiting software based fabric and the Named-Object abstraction.

5.1.1 NOVN General Design

NOVN addresses the fundamental issues of Virtual Network management and deployment support through the use of Named-Objects and the reliance on information offloading to the NRS. A set of core design operations are at base of the framework. We go through each one of them referring to Figure 5.2 for clarity. In this Section we keep into account three core assumptions: 1) the availability of a globally accessible NRS capable of storing mappings from *names* to *values* (in this case either network addresses or other names for indirection); 2) ;3) the flexibility of accessing names and addresses as part of a network header to enable hybrid routing, similar in spirit to the one employed in the MobilityFirst architecture [39].


GNRS Service Plane

GUID	VN Type	Locator
A1	No	19
VNID_1	Yes	VR1, VR2, ...
VR1	No	R1
Y	No	A1, A2, B2

🟥	Normal Mapping *GUID to Location*
🟩	Mapping *VN-ID to VN-Members*
🟨	Mapping *VN-Name to Phy-Name*
⬛	Mapping *Svc-ID to Svc-Members*

Figure 5.2: NOVN design.

Logically Define a VN Network Through Names. *NOVN* simplifies the definition of the virtualized logical layer through information offloading to the NRS. This is done as a three step process: 1) first, a unique identifier is assigned to the VN and a mapping from such name to all participating resources is stored in the naming service (red box in the Figure); reference resources are identified with a name that has meaning only within the limits of the Virtual Network Logic - i.e. they are unique and no shared across different VN instances; this provides the double function of simple access and distributed information recovery. 2) Each VN resource name, is then mapped into two values: a) the name identifying the resource the virtualized element is running on top and b) the list of its neighbors. 3) Finally, name resources are mapped into physical Network Addresses allowing for normal forwarding operations. Points 1 and 2 together define the higher abstraction level shown in Figure 5.1 and their mapping into the mid-layer, while point 3 provides the last translation to the bottom one.

Bootstrap Process & Management. As the topology information is made available at a global scale through the NRS and can be dynamically retrieved from participating resources, the scope of what information is required to share at each layer of the network infrastructure is limited in comparison to other solutions. This allows two core issues to be handled separately: the *local* problem of mapping virtual to physical resources, that could be handled in a network-by-network basis by a centralized authority, and the *global* problem of coordinating the virtualized logic across domains, offloaded to the NRS. To this end, the bootstrap process in *NOVN* is then limited to allocating on participating nodes instructions on how to retrieve the VN topology, i.e. the VN unique identifier used to query the NRS, and the information about the physical resources that are required. Similarly, management operations, e.g. migration, of resources can handled through NRS offloading too, whereas local changes are reflected into the globally accessible service and dynamically resolved at forward time.

Routing & Forwarding. Providing full flexibility for different routing configurations, *NOVN* does not constrain VN users to employ specific routing protocols. Routing information is exchanged across nodes exchanging control packets encapsulated to reach participating nodes. Similarly, data forwarding happens on a hop-by-hop manner across routers participating in the Virtual Network. When a data chunk reaches one of these routers and a routing decision is taken, the chunk is encapsulated as shown in Figure 5.2 where the external network header contains information to reach the next VN router. While crossing nodes not participating in the protocol, normal routing decisions are taken using the external network header. As names are used to identify hops, forwarding can happen independently from the physical network configuration.

Application Oriented Routing. Once the core abstraction layer of the *NOVN* framework has been established, virtualized routing resources can be exploited to easily deploy more advanced services aimed at supporting network applications. Two core pieces of technology are required and introduced into the framework: first, the ability to aggregate multiple service instances under a single name, a natural extension of the Named-Object abstraction. This is done by offloading the list of participant locations under a single name into the name resolution service. Second, the ability to make application nodes participate in the routing protocol by sharing their application state. This can be either implemented through a new interface in the participating routers, requiring though the introduction of new schemes to identify participants of the protocol, or by offloading this information to the name resolutions service. An example of how this application state information could be used in a routing algorithm is shown in Figure 5.2, where thresholds based on service load and distance are merged into a single decision process.

5.1.2 NOVN Protocol Details

The *NOVN* framework, as described in Section 5.1.1, provides a clean way to define a Virtual Network topology through the use of the Named-Object abstraction. While using this tool it is possible achieve the purpose of providing the high level mechanisms that characterize our system, additional details are required to provide a better sense of how *NOVN* can fully overcome the issues presented and how it could be deployed on top of the current IP/TCP Internet architecture.

A Better Virtualization Abstraction

Traditional network virtualization techniques have always suffered from the fundamental shortcomings of the IP architecture and address structure, limiting their flexibility and increasing deployment complexity. Consider the case of overlay based solutions (e.g. VINI [82]) where virtual router interfaces are assigned private IP addresses and then mapped to public ones that can be used to tunnel packets across participating resources. Due to the nature of IP addresses, any configuration change due to failure or resource migration requires the tunnel to be reconfigured, the new information to be propagated across all the participating resources, causing all ongoing traffic to be lost. This is due to packets not being able to carry the necessary information to self-correct temporary errors. Approaches to reduce this impact have been explored [86], but require the creation of dedicated control channel to maintain persistent traffic flow.

NOVN exploits Named-Objects to solve this issues by creating clean partitions across logical layers, as previously shown in Figure 5.1. This is obtained by recursively mapping from VN dedicated names, to network elements names and finally to the physical addresses. This different layers of abstraction are critical in allowing a separation of management issues. Consider for example the case of virtual routers migration. In *NOVN*, the process is simplified by limiting the impact of the migration to remapping resources between the top two layers. Once the required migration process is defined, the entry mapping the VN element to the network element is re-written to the new location. If on-flight packets are forwarded while the transfer process is occurring, names allow for simple recovery without need of end-to-end retransmission, by resolving the delivery location through the NRS. Similarly, if a physical machine needs to be replaced due to failure or an address change is required, a new can be instantiated and the state transfered.

One could argue that the employment of multiple layers of abstraction can introduce additional overhead due to the resolution costs of crossing the different logical layers through name resolution and due to the additional headers employed. The impact of these can be reduced by employing two separate techniques: 1) While name resolution

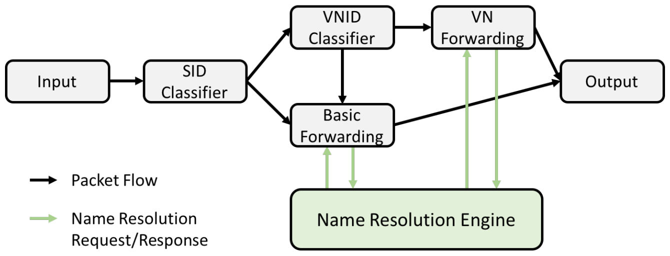


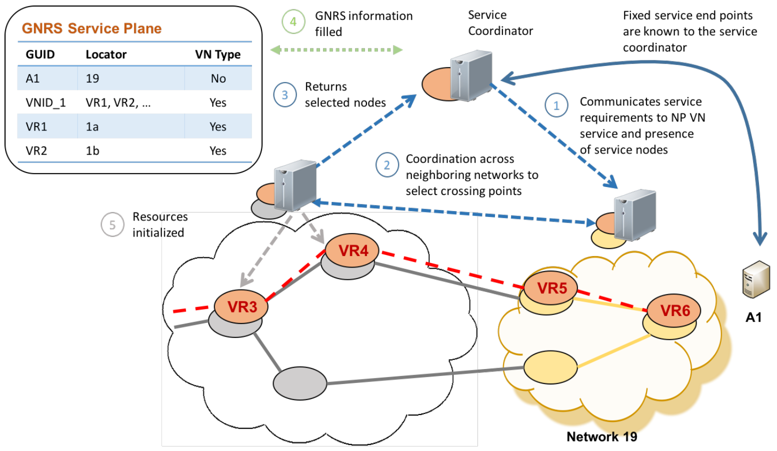
Figure 5.3: Packet flow inside a software router.

can become costly if performed for each forwarding decision, this is not required as for the majority of the time the resources do not change; hence, information can be pre-cached on the participating routers and only once notified of changes resources re-update their mappings by querying the NRS. 2) SDN based approaches [77] have shown that matching multiple fields in hardware is a feasible task and the presence of the full information. Figure 5.7, shows the simple logic employed where 3 matching fields (e.g. the SID, the VNID, and the destination name) can be mapped to easily take forwarding decisions at the virtualized layer.

Separating Local and Global Tasks

The management of resources in virtualized environments increases in complexity when extended to multiple domains. This is true for overlay approaches, where resources need to be coordinated and communicated potentially across multiple networks in order to coordinate, and it is mostly untreatable for tag based solutions that are mostly optimized for small domains, e.g. a data center or an access network. This is a consequence of the complexity of assigning coherent resources across multiple domains that can be managed by different commercial entities.

NOVN aims to approach the problem by creating a distinction between the *local* problem of assigning network and computing resources and the *global* problem of providing coordination mechanisms across domains. The NRS and the Named-Object abstraction are the key elements employed to provide ways for eliminating the complexity as they provide the globally accessible infrastructure to offload the sharing of the virtualized topology and the mapping to the underlying elements. With this, network administrators can then focus on: first, deploying techniques that and second, introduce and maintain update the mappings of requested resources to the available underlying network components.


GNRS Service Plane
4 GNRS information filled
Service Coordinator
Fixed service end points are known to the service coordinator
3 Returns selected nodes
1 Communicates service requirements to NP VN service and presence of service nodes
2 Coordination across neighboring networks to select crossing points
5 Resources initialized
VR4
VR3
VR5
VR6
A1
Network 19

GUID	Locator	VN Type
A1	19	No
VNID_1	VR1, VR2, ...	Yes
VR1	1a	Yes
VR2	1b	Yes

Figure 5.4: Separation of local and global scale problems through a distributed coordination plane.

Fig. 5.4 outlines the resource allocation process when a hierarchical set of service coordinators are employed. In this example, each network domain exposes an interface that services deploying a multi-network VN can invoke to allocate resources that span across the participating networks. While this example employs the concept of a single service interface per network with a centralized controller for requesting and coordinate resources across networks, the same tools can enable more distributed mechanisms for allocating and deploying Virtual Networks.

Network State Exchange

Similar in spirit to previous attempts of providing full control of the deployed routing protocols on top of the virtualized network [82], *NOVN* has been designed to offer a routing independent network abstractions. In other words, virtualized elements can independently choose which routing protocol better suits their needs as long as they have ways of learning the underlying network conditions, e.g. virtual links cost. This problem could be approached in multiple ways: a) recurring to overlay approaches where the in built measurements tools are used to extract the information, as done in VINI; b) by allowing routing information sharing across layers, through the use of APIs exposed by the underlying networking logic.

The current *NOVN* design uses the second approach: we exploit a software based router prototype implementation to support APIs, that are used by the virtual layer to extract link state information. The extracted information is then exchanged by employing a Link State like Protocol, where routers distribute through flooding the aggregated cost view of each virtualized link.

5.1.3 Application Specific Routing

In order to support advanced mobile edge cloud scenarios where the goal is to connect mobile devices to the "best" or "nearest" cloud server the *NOVN* framework was designed to further support *anycast* delivery services. While basic anycast services have been implemented over the years through the employment of DNS based techniques or through application based overlays, providing it as an integrate abstraction of the networking components would highly benefit system performance, by reducing experienced latency of the system. This is of fundamental importance in highly dynamic environments required for example by mobile edge clouds where the location of the service often migrates to closely follow moving users. Moreover, the concept of anycast could be further extended to allow different definitions of "best" destination. Recent studies [83, 84] demonstrate how allowing applications to affect routing decisions based on their own metric, e.g. server load for load balancing, can have a positive impact both in terms of application and network performance. A clean-slate design should aim at taking advantage of the opportunity and provide enough flexibility to support coordination across application and network logic.

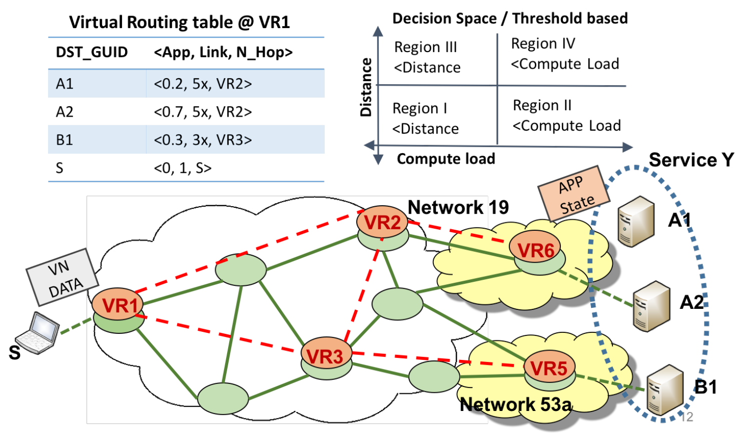
Again, in this scenario, the name-object abstraction allows network resources to take more informed decisions. Considering the example in Figure 5.2, the *black* mapping represents how a service name could be be bound to its participants allowing the forwarding fabric to obtain the potential desired destination. With the provided *anycast* service, the client solely needs to specify the name of the service and will be routed to the best destination site. While this abstraction is powerful. In most cases, the definition of the best service replica to select can vary, depending on the nature of the application. For this reason, we use our Virtual Network design to support Application Specific Routing (ASR). ASR allows the application components to proactively provide information regarding the status of the service to the routing components. This information, expressed in the form of a single or multiple metric values, can be integrated into the routing algorithms to take more informed decisions.

Introducing Application Performance Index into Service Anycast

While a name based architecture, like MobilityFirst, is well suited to provide the right abstractions for *Anycast* based services, when in need of meeting strict application performance requirements, additional control over routing decisions might be desired [83, 84]. In order to do so, we developed a concept called Application Specific Routing (ASR). ASR allows routing decisions to be based both on network and application metrics. Consider the scenario shown in Figure 5.5 where a service Y is distributed across three locations *A1*, *A2* and *B1*. When receiving a data packet from client S, classic routing protocols would forward such packet based solely on network level information, e.g. bandwidth, latency, hop counts, etc. Through ASR, we provide a framework that allows not only to consider classic L3 metrics, but also application layer ones, such as cloud workload/latency for the edge-cloud scenario.

ASR is implemented as an integrated function of the virtualization layer, whereas participating routers can be configured to support different metrics and forwarding logics. Current ASR's implementation is based on two core operations:

Application Metric Dissemination. When ASR is active, cloud nodes participate in the routing protocol by sending Application State Packets to the edge router they are attached to; the routers then re-distribute the state information across the other participant routers by inclusion of the metrics and related information (e.g. name of



Virtual Routing table @ VR1

DST_GUID	<App, Link, N_Hop>
A1	<0.2, 5x, VR2>
A2	<0.7, 5x, VR2>
B1	<0.3, 3x, VR3>
S	<0, 1, S>

Figure 5.5: Application Specific Routing (ASR) Concept

the server) into the routing packets exchanged within the Virtual Network.

Forwarding Decisions. The routing information exchanged is used to compute two tables: 1) the classic routing table used for normal networking decisions and 2) a service table that contains the current state (i.e. cloud server load) for the different cloud service locations. When receiving data packets with destination the service, i.e. Y, the Application Specific Routing logic is used to select the next hop. The current implementation supports a basic threshold based logic, as shown in Figure 5.5, where potential destinations are divided into a decision space in which different regions have higher priority based on desired behavior. For example *Region II* could be preferred over *Region III* if network metrics have higher importance over server load; they could be swapped otherwise. Our framework could be easily extended to support different decision logics.

While the current ASR implementation supports basic threshold based routing logic, our design allows for future extensions supporting more complex algorithms based on the collected state information.

5.1.4 Prototype and Benchmark Results

In order to understand the potential, the achievable performance and most importantly the feasibility of the proposed framework, we developed a full prototype of the proposed design. The components on top of which the framework is built, lay their foundation on the MobilityFirst future Internet architecture prototype [39, 87]. Thanks to the clean separation of names and addresses provided by the GNRS, MobilityFirst provides all the necessary components to natively deploy our design.

We deployed our prototype on the ORBIT testbed [72], a two-tier wireless network emulator/field trial designed to achieve reproducible experimentation. ORBIT's main facility is the radio grid testbed which uses a 20x20 two-dimensional grid of programmable radio nodes. While originally designed for wireless experimentation, ORBIT also provides a full wired 1Gbit ethernet connection between all nodes, creating a single L2 network. This provides experimenters with a fully customizable network to run wired experiments. Selecting 18 nodes out of the 200 available, we deployed the network shown in picture 5.6, using L2 VLANs to isolate traffic creating point to point connection between software routers. All selected nodes are equipped with Intel i7 processors, 16GB of memory and a single 1Gbps ethernet port, and run Ubuntu 14.04 LTS.

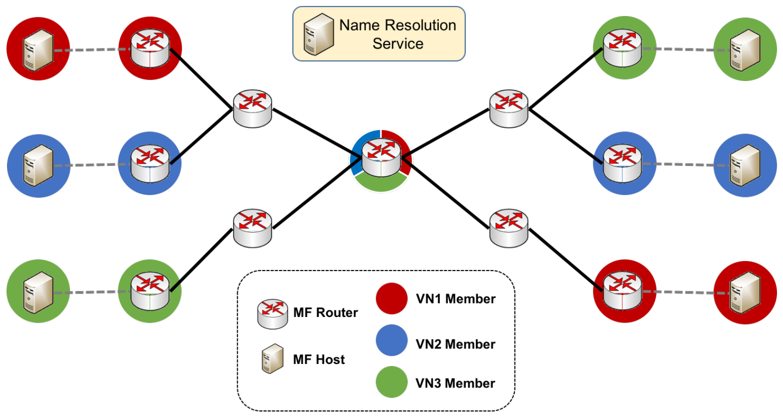


Figure 5.6: Experimental setup

After first introducing in the next subsection the design of the core components of the employed prototype, we will present initial benchmark results.

Click Based Virtual Routers

The MobilityFirst prototype is composed of three core components: a Java based Global Name Resolution Service that uses DMap's [23] log to implement entries distribution, a software router implementing MF's name based logic and a host API and network stack to run applications on the architecture; a more detailed description of the base MobilityFirst prototype has been provided; here we will focus on the changes made to support the *NOVN* design.

Routers: The software router is implemented as a set of routing and forwarding elements within the Click modular router [69]. The router implements dynamic-binding using GNRS, hop-by-hop transport, and storage-aware routing. It integrates a large storage, an in-memory *hold buffer*, to temporarily hold data blocks when destination endpoints during short-lived disconnections or poor access connections. A particular instance of this system, implements what we call a MobilityFirst access router, a router providing access connectivity to clients. We extend the base router to introduce the Virtual Network logic; three fundamental new logic blocks are introduced: 1) a VN manager that handles the initialization of the required classes and contains the required information for each VN instance; 2) the ASR service block, which collects application state date and compute the necessary algorithms to maintain it; 3) finally, the routing/forwarding that uses the information pulled from the routing layer and the ASR service block to compute forwarding tables. Multiplexing between normal traffic and the different VNs is handled based on Service IDs and Names available in the MF routing header as shown in Figure 5.7; this simple field base multiplexing exploits VN native concepts, i.e. integrated in the network header, minimizing processing overhead.

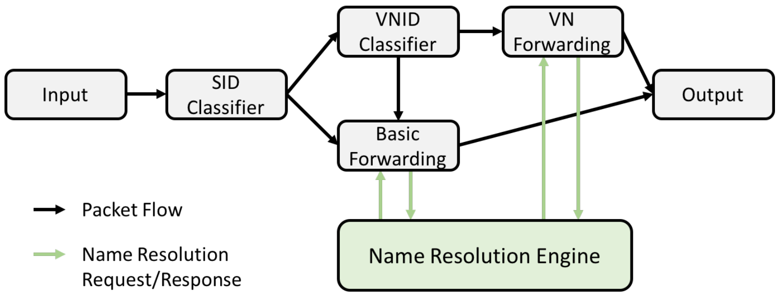


Figure 5.7: Packet flow inside the software router

Clients: The host stack has been implemented on Linux and Android platforms as a user-level process built as an event-based data pipeline. The stack is composed of a flexible end-to-end transport to provide message level reliability, the name-based network protocol including the GUID service layer, a reliable link data transport layer, and a policy-driven interface manager to handle multiple concurrent interfaces. The device-level policies allow users to manage how data is multiplexed across one or more active interfaces. A novel socket API [32] is available both as C/C++ and JAVA libraries and implements the name-based service API which include the primitives *send, recv, and get* and a set of meta-operations available for instance to bind or *attach* a GUID to one or more NAs, configure transport parameters in the stack, or to request custom delivery service types such as multicast, anycast, multihoming, or for our scenario, VN traffic.

5.1.5 Micro-Benchmarks

In order to understand the basic overhead introduced by running our Virtual Network logic on top of the base MobilityFirst prototype, we performed two sets of experiments on the ORBIT testbed, comparing both to the baseline MF implementation: first, a latency evaluation using a *ping*-like application that collects Round Trip Times (RTTs) for varying chunk sizes; second, using a port of *iPerf* [88] that uses our API and stack to transmit data, we estimate achievable bandwidth. For both scenarios we use the network shown in Figure 5.6, but we limit traffic generation to VN-2 (blue color).

Latency: We measure the average and standard deviation of RTT times between the client and servers belonging to VN-1, with varying chunk sizes. The collected results are shown in Figure 5.8 from which we can observe how the deviation from the baseline scenario is minimal in most cases.

Bandwidth: The same setup is used for the second case where *iperf* is used to evaluate the total achievable bandwidth. We run the experiment for 60 seconds for three different chunk sizes, 1500B (i.e. a single packet), 10KB and 100KB. Experienced bandwidth is collected every 10 seconds. Figure 5.1.5 shows the obtained results; as in the latency scenario, no evident differences are experienced in comparison to the baseline scenario.

We can conclude that, when a single Virtual Network is used with no additional

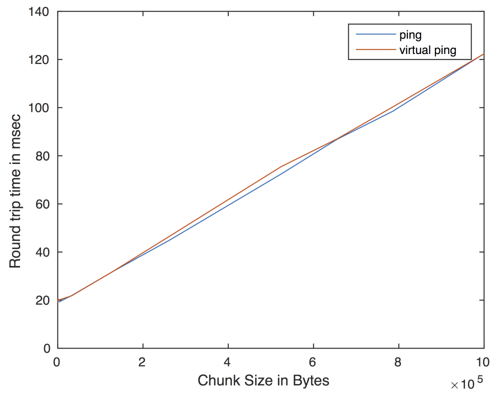


Figure 5.8: Ping based RTT for different chunk sizes

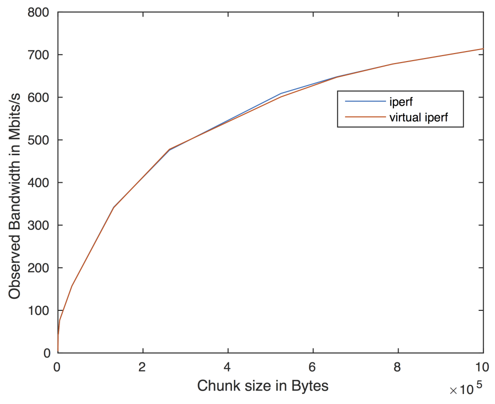


Figure 5.9: Iperf experiment

traffic, performance is not affected in comparison to the baseline MF implementation, confirming the minimal overhead of our implementation.

5.2 vMCN: Virtual Mobile Cloud Network

The emerging network services connecting and controlling machines, vehicles and other objects in the physical world often referred to as the Internet-of-Things (IoT) or cyber physical system (CPS), represent an important set of future requirements for the network research community. Unique technical challenges associated with the IoT and CPS scenarios include scaling the Internet architecture to support a very large number of objects including wireless/mobile devices, information, network addresses, virtual machines, etc., efficient integration of cloud computing services necessary to serve CPS systems, designing appropriate security and trust models, and achieving fast response for real-time/closed-loop applications. Fundamentally new architectural approaches to networking will be required to address the emerging needs of IoT and CPS scenarios.

In this study, we look at how to properly support mobile real-time CPS applications, such as Augmented Reality (AR) based navigation and self-driving cars. The first key challenge to realize real-time CPS is scaling to billions or trillions of objects, which are connected to networks or other devices, or whose meta data or statuses are handled in applications software. The second key challenge is low latency in applications. For example, Glass device based AR applications require less than 100 ms response time, while self-driving cars require 10 ms order of magnitude. Such extreme requirements cannot satisfied by the current mobile network systems.

In order to meet the described requirements, we propose a virtual mobile cloud network (vMCN) architecture for emerging scalable, real-time CPS applications. The idea is to create a single virtual network (or "slice") for a given cloud service, which provides the illusion of a local network with uniform authentication of devices and services, seamless mobility of devices/users, dynamic migration of BS and cloud resource and state, coordinated with some level of managed wireless resource allocation. Novel network virtualization techniques are introduced that exploit the "named-object" abstraction provided by a fast and scalable global name service. The first challenge of scaling to billions or trillions of objects is addressed by the presence of the global name resolution service (GNRS), a key stone of the MobilityFirst (MF) architecture [39] that serves as the networking foundation for vMCN, which is designed to provide fast dynamic binding 100B-1T object names (globally unique identifiers known as GUIDs) and their current network locators. The second key challenge of achieving low latency is achieved by the two following novel mechanisms: first, cloud service addressability and anycast capabilities enabled by the name base routing available in MF and enhanced through virtualization techniques deployed into the network; second, the prioritization of a specific service in wireless access enabled by dynamic assignment and migration of virtualized BS resources supported by the foundation of the MF GUID service layer.

The design of vMCN architecture lays its foundations on the three core technology components and coordination techniques among them: a) GNRS, b) *NOVN*, presented in the previous Sections; finally c) vBS, a virtual network framework in WiFi networks [89, 90]. The initial proof-of-concept prototype of the vMCN was developed based on the integration of the *NOVN* prototype on the ORBIT testbed, and the vBS prototype system developed by NICT. Through a set of experiments on the vMCN prototype, the impact on the reduction of CPS response time was evaluated. The experimental results reveal the vMCN can support up to about 94% CPS cycles under the set goal of 100 ms, outperforming the baseline system by almost two times.

5.2.1 vMCN Design

The designed architecture exploits two fundamental technologies developed at Rutgers and NICT: a Virtual Network designed on top of the name based Future Internet architecture MobilityFirst (*NOVN*) [39] for inter domain support of the applications and the virtual Base Station (vBS) for local domain support [89]. The two components are merged into an architecture that is aimed at supporting a variety of services, including the trillion-order scalability CPS applications with less than 100 ms response time. The comprehensive architecture, as shown in Figure 5.10, relies on four main characteristics: 1) a fast and scalable global name resolution for user mobility through a Global Name Resolution Service (GNRS), 2) a virtual network with dynamic configuration of wired & wireless resources and inter-domain migration, and 3) a network-assisted service anycast routing service for supporting edge cloud services. As the first two components have already been presented in this work, the following paragraphs will focus on the second components and on how *NOVN* integrates vBS to implement the vMCN architecture.

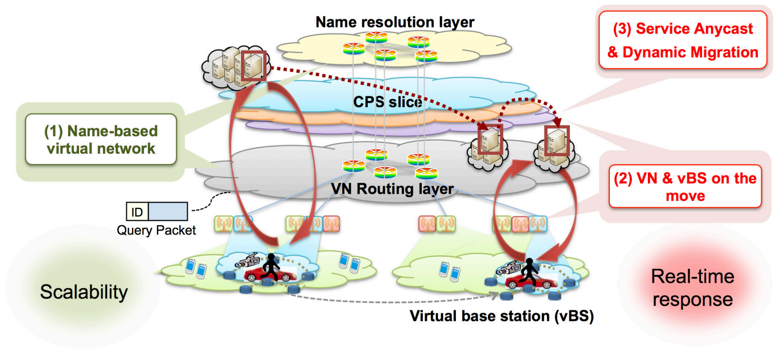


Figure 5.10: vMCN Architecture Design

vBS: WiFi Network Virtualization

Figure 5.11 shows a basic concept of WiFi network virtualization [89]. WiFi network virtualization is a technique in which physical WiFi network infrastructure resources and physical radio resources can be abstracted and shared by multiple independent and customizable logical (virtual) WiFi networks through isolating each other, and can be considered as an example of wireless network virtualization [89, 91–95]. We call a set of physical WiFi network infrastructure resources and physical radio resources as a BS resource for simplicity. In Figure 5.11, a set of isolated BS resources is presented as a vBS. A vBS behaves as a logical multi-channel BS organized by multiple physical BS resources. We define a vBS that dedicates all its own BS resources to a target service as a *service-specific* vBS [96, 97]. In the current design, intra-domain migration of the vBS is supported [90].

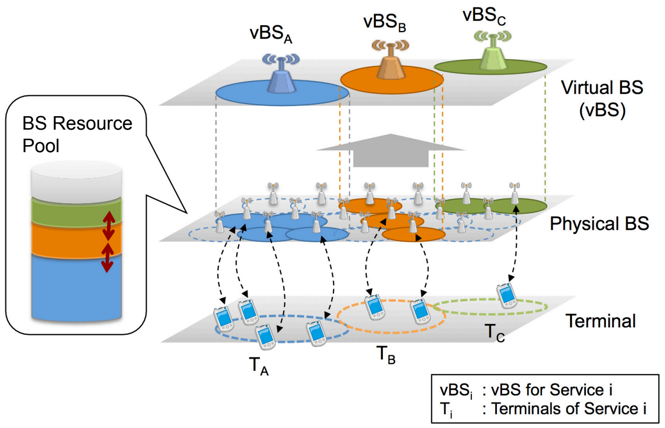


Figure 5.11: Concept of WiFi Network Virtualization (vBS)

To make a vBS logically behave as a single BS, a logical layer-2 network spanning across physical BSs that organize the vBS is configured in the backhaul. In addition, all the BSs are configured with the same MAC address in a wireless interface, and hence with the same BSSID. In the same way, the same ESSID are configured at all the BSs. These configurations make it possible to separate BS selection and handover decisions from BSs and terminals and put them together into a centralized controller.

The distinct advantage of the architecture is that all the decisions on BS selection and handover can be fully managed regardless of the differences in the vendor-specific BS selection and handover algorithms implemented in terminals. The association and handover procedures naturally go together with layer-2 routing (re)configurations in the backhaul for the terminals, so that OpenFlow [98] is exploited for this purpose. Reducing handover latency and avoiding packet drops during handover can be also achieved by cooperative and fast layer-2 path reconfiguration.

5.3 Protocol Details for Components Coordination

In order to implement the comprehensive vMCN architecture, *NOVN* and vBS technologies have to be integrated to dynamically coordinate resources to create a single virtual network (or "slice") for a given cloud service.

5.3.1 Dynamically Configuring a Name-based Virtual Network

The three core technology components previously described represent the foundations required to achieve a scalable architecture to support the real-time CPS. In order to achieve a fully functional architecture a series of coordination techniques are required among those components. The key integration point is a mechanism for building an integrated virtual network by bridging a MF-based virtual network (MF VN) and a corresponding vBS. In this paper we propose a mechanism to coordinate the MF's control plane and the vBS's one for that purpose.

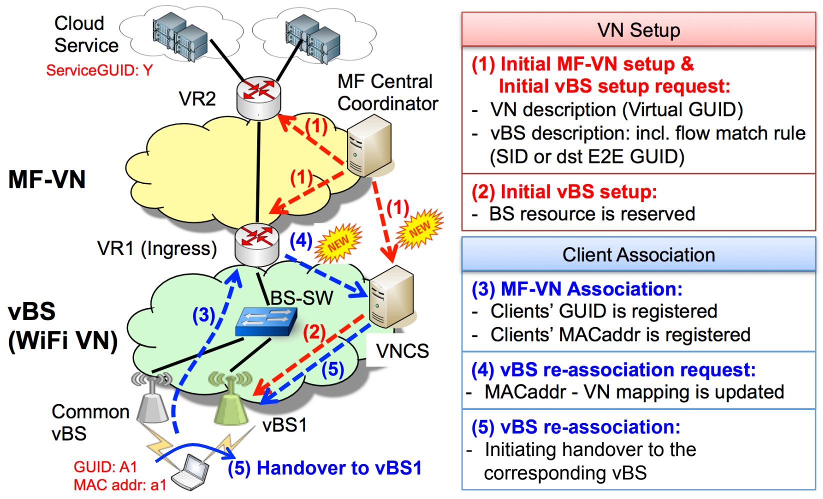

Cloud Service
ServiceGUID: Y
VR2
MF Central Coordinator
MF-VN
(1)
(1)
(1)
VR1 (Ingress)
(4)
NEW
NEW
vBS (WiFi VN)
(3)
BS-SW
(2)
(5)
VNCS
Common vBS
vBS1
GUID: A1
MAC addr: a1
(5) Handover to vBS1
VN Setup
(1) Initial MF-VN setup & Initial vBS setup request:
- VN description (Virtual GUID)
- vBS description: incl. flow match rule (SID or dst E2E GUID)
(2) Initial vBS setup:
- BS resource is reserved
Client Association
(3) MF-VN Association:
- Clients' GUID is registered
- Clients' MACaddr is registered
(4) vBS re-association request:
- MACaddr - VN mapping is updated
(5) vBS re-association:
- Initiating handover to the corresponding vBS

Figure 5.12: The control planes of MF-VN and vBS are coordinated for building an integrated virtual network

Figure 5.12 shows a procedure of building an integrated virtual network based on the coordination between the MF resource manager (called central controller in the picture) and the vBS's controller, namely VNCS. The procedure can be divided into two parts: the first part is the VN setup phase (step (1) and (2)) and the latter one is the client association phase (Step (3) – (5)).

In the VN setup phase, the MF resource manager initiates the MF-VN setup by pushing to the participating nodes the definition of requested resources and the unique identifier that characterize the Virtual Network(Step (1)). Note that an original process is added that the coordinator requests the VNCS to create a corresponding vBS using the northbound VNCS API [89]. A unique tag, called Service ID, or alternatively the E2E communication GUIDs, corresponding to the MF-VN is also notified to enable the VNCS to identify the GUID packets of the VN. Then, the VNCS creates the requested service-specific vBS (vBS1) by configuring the physical BSs and the BS switch (BS-SW) in the BS backhaul (Step (2)).

In the client association phase, a MF client is associated with the MF-VN first, and then associated with the service-specific vBS. While the client WiFi interface is active, the client periodically looks for a MF ingress router using broadcast messages. The MF-VN association is completed when the client receives the acknowledgement from the ingress router (Step (3)). The GUID and MAC address of the client are registered in the ingress router. Note that at this step, the MF client is associated with a common vBS instead of the service-specific vBS.

Then the ingress router notifies the VNCS of MAC address – VN mapping information (Step (4)). This information is required for the VNCS to identify which MAC address should be bound with which vBS. The Step (4) is also original in the proposed integration mechanism. Finally, the VNCS initiates the handover of the target client to the service specific vBS (Step (5)).

5.4 Prototyping

The initial proof-of-concept vMCN prototype system was developed by integrating the two prototypes of vBS and *NOVN*. The MF-VN prototype was developed by implementing the *NOVN* logic described in Section 5.1.4 on the MobilityFirst foundation in

MF nodes.

The vBS prototype consists of (1) VNCS (Virtual Network Control Server) that mainly manages vBS creation, radio resource assignment, client association, and handover, (2) virtualization-capable BS, where 22 IEEE 802.11a/b/g/n WiFi modules are equipped and at most 22 WiFi BSs can be run independently at a time, and (3) BS-SW, which is an OpenFlow switch. The BS-SW was implemented using Open vSwitch in this integrated prototype.

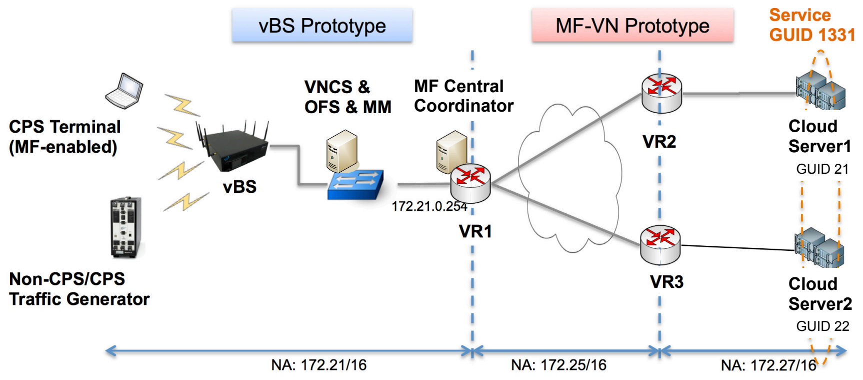


Figure 5.13: Physical Structure of the vMCN Prototype

Figure 5.13 shows the physical structure of the vMCN prototype. The MF-VN part consists of three MF routers and two cloud servers. These five servers as well as a GNRS node were built on the ORBIT testbed in the Rutgers University. Then the BS-SW (OFS) is connected to the MF ingress router (VR1). The VNCS and the MF central coordinator were run on the BS-SW and VR1, respectively.

For integrating these two prototypes, the function of dynamically configuring a virtual network among vBS and MF-VN was implemented. Specifically, the function of calling an initial vBS setup request (Step (1) in Figure 5.12) was implemented in the MF central coordinator. This function is embedded into the existing initial MF-VN setup procedure. The function of calling a vBS re-association request (Step (4) in Figure 5.12) was implemented in the MF ingress router. This function is invoked just after the MF-VN association procedure.

The above new functions require communications with the VNCS, and a software

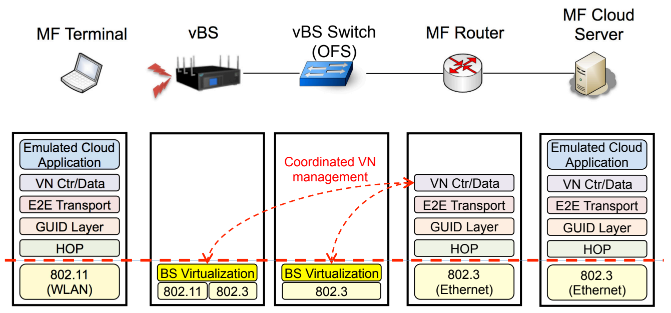


Figure 5.14: Protocol Stack

module for calling the northbound API of the VNCS, namely VNCS API [89], was installed in the MF central coordinator and the ingress router. The VNCS API provides a series of methods, such as SliceAdd, vBSAdd, and vBSAttachBS for creating a vBS. These methods are used in Step (1), and the EnvHandover method is used in Step (4).

Figure 5.14 shows the protocol stack implemented on the vMCN prototype system. The vBS stack is working as a layer-2 protocol, and a MobilityFirst GUID packet encapsulated by the IEEE 802.3 or IEEE 802.11 header can be routed on the vBS and BS-SW as it is. The interaction between the BS virtualization layer and the VN control/data layer denotes the process of dynamically configuring a virtual network.

5.5 Reduction of the CPS Response Time

5.5.1 Experimental Setup

A first batch of results showcasing the potential of the vMCN architecture has been obtained by experiments on the prototype system. Figure 5.15 shows the basic experimental scenario, which can evaluate both the individual effects of the vBS on vWiFi and ASR on MF-VN and the marginal one.

We generated loosed-loop (round-trip) UDP traffic and one-way UDP traffic at terminals to emulate a real-time CPS service and a non-CPS best-effort one, respectively. The CPS-specific vBS and the best-effort common vBS on the virtualization capable BS (vcBS), respectively, and a physical BS configured with IEEE 802.11 n/a mode and 65 Mbps transmission rate is assigned to each vBS. The channel 36 and 48 in the 5 GHz

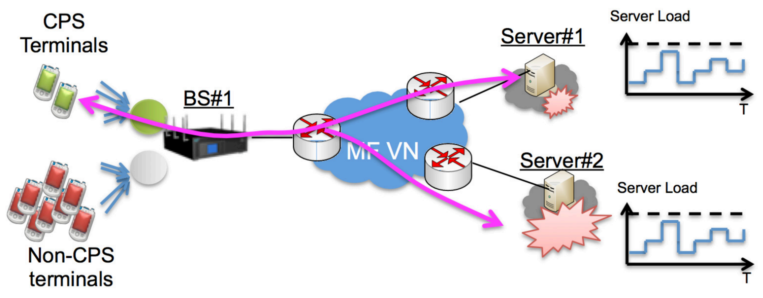


Figure 5.15: Experimental Scenario

band are assigned to these vBS, respectively. When these vBSs are activated, CPS and non-CPS traffic are completely isolated, and the CPS response time can be reduced. On the other hand, without the vBSs, i.e. in the normal WiFi mode, the interference causes the increase of the response time.

In the could servers, dynamic server load are configured. Every 10 seconds, each server randomly chooses the server load from the preconfigured parameter set {0.2, 0.4, 0.6, 0.8}, and linearly increased latency of {20, 40, 60, 80} ms is injected before responding to the received CPS data unit. The server load is announced every 2 seconds to the MF routers, and then the ASR routing table is updated accordingly. When the ASR is activated on the MF-VN and servers, the ASR routes the CPS traffic to the less-loaded server, and the CPS response time can be reduced. On the other hand, without the ASR, i.e. in the normal GNRS mode, the CPS traffic are always routed to Server#1, and the response time is significantly affected during the high server load period.

To evaluate the application-level CPS performance instead of the packet-level one, we generate a large size data unit assuming a picture frame. For example, in the case of 50 KB data unit and 1500 Byte MTU, a packet train including 37 MF-formatted packets is generated. We generated a CPS data unit every second at each CPS terminal. The response time is defined as the time from the generation of a data unit to the reception of the last packet of the packet train. If any packet in the train is lost, we consider the data unit is lost.

We setup three CPS terminals and 12 non-CPS ones per physical BS. We configure two physical BSs, and 30 terminals are configured in total. This means that when vBSs are activated, the CPS-specific vBS accommodates 6 CPS terminals and the remaining 24 non-CPS terminals are accommodated by the non-CPS vBS, when vBSs are activated. On the other hand, without vBSs, each physical BS accommodates 3 CPS terminals and 12 non-CPS ones. We generated the non-CPS traffic with 100KB data unit every 1 second at each non-CPS terminal, and the load offered by 12 non-CPS terminals is 9.65 Mbps. We implemented the CPS traffic generator, named ßmfping, on a Linux laptop to measure the CPS response time. The other CPS and non-CPS terminals are configured on the IXIA WiFi terminal emulator. All the terminals are connected to the vcBS using coaxial cables to eliminate the interference from the other wireless systems using the ISM band.

5.5.2 Experimental Results

Figure 5.16 shows the cumulative distribution function (CDF) of CPS response time.

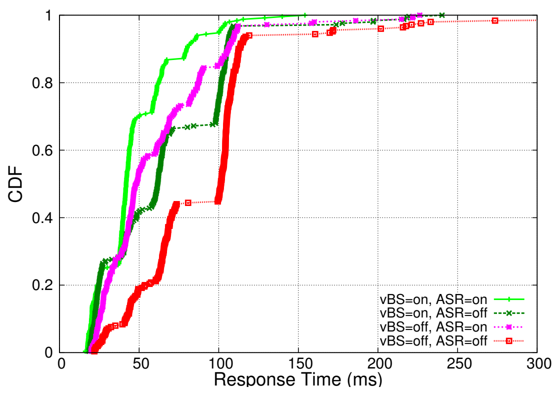


Figure 5.16: CDF of CPS Response Time (Data Unit Size = 25KB)

300 data units with 25KB size are generated in this experiment. Apparently the combination of vBS and ASR outperforms the other three cases. As we target the response time less than 100ms, we focus on how many data units meet the requirement. In the case of vMCN, i.e. (vBS, ASR) = (on, on), 94% data units achieves less than 100ms response time. On the other hand, in the case of (vBS, ASR) = (off, on),

(on, off), and (off, off), the value decreases to 85%, 74%, and 46%, respectively. These results show ASR has larger impact to lift up the CDF line, We can conclude the vMCN can support up to 94% CPS cycles under the set goal of 100 msec, and outperforms the baseline system by almost two times.

The specific percentile response time is also an important performance index to evaluate a real-time system. We evaluate the performance in the congested WiFi environment, we determines to evaluate 90 percentile response time. In the case of (vBS, ASR) = (on, on), the 90 percentile response time is 80 ms. On the other hand, in the case of (vBS, ASR) = (off, on), (on, off), and (off, off), the value increases to 106 ms, 105 ms, and 113 ms, respectively. These results show vBS and ASR have the same level of impact. We can conclude the vMCN can achieve less than 100 msec for 90 percentile response time for 25KB CPS data units.

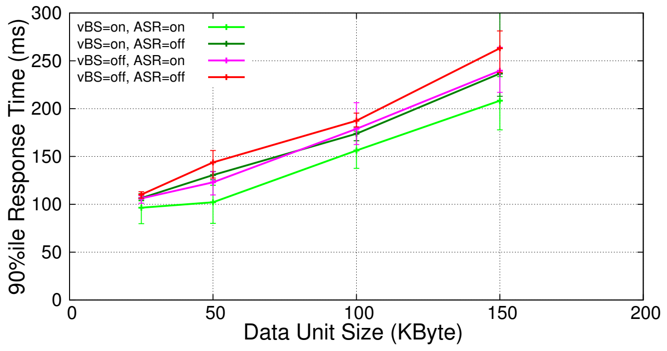


Figure 5.17: 90 Percentile Response Time

Figure 5.17 shows the impact of data unit size on the 90 percentile response time. We measured 10 times (obtained 10 sets of 300 samples) for each parameter set. The average values with the range between the minimum and maximum ones are plotted in the figure. In all the cased, the larger data unit size has larger impact on the response time. The slope on each graph is almost the same, and the effects are common for all the cases. We can conclude the it is difficult for vMCN to keep the 90 percentile response time less than 100 ms for data units with larger than 50KB. Note that the non-CPS traffic is static, and the ratio of CPS traffic becomes larger for larger CPS data unit size. For example, the load offered by three CPS terminals is 4.0 Mbps when the CPS data unit size is 150 KB.

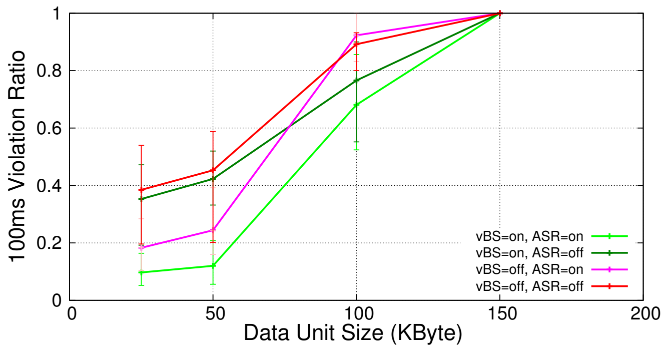


Figure 5.18: 100ms Delay Violation Ratio

Figure 5.18 shows the impact of data unit size on the 100 ms violation ratio. The 100 ms violation ratio is expressed as $1 - \frac{N_{success}}{N_{all}}$, where N_{all} and $N_{success}$ denote the total number of generated data units and the total number of data units that meet the response time of less than 100 ms, respectively. In the case of (vBS, ASR) = (on, on), the 100 ms violation ratio is 0.10 and 0.12 for 25 KB and 50 KB data units, respectively. When the unit size is 50 KB, the 100 ms violation ratio in the case of (vBS, ASR) = (off, on), (on, off), and (off, off) is 0.24, 0.42, and 0.45, respectively. When the unit size is larger than 100 KB, the ratio significantly increases. We can conclude the it is difficult for vMCN to keep the 100 ms violation ratio less than 0.1 for data units with larger than 50KB.

Chapter 6

An End-to-End Service Realization

Throughout this thesis, different solutions and technology components have been presented, targeted at solving piece by piece the requirements of each new problem presented. After starting from a top-down analysis of the network requirements, experimental results have been presented to support the case for our solutions. This chapter aimes to complete this discussion, presenting a comprehensive analysis of all the technologies implemented throughout this study and show how they can merge together to realize a comprehensive experimental end-to-end realization of name based services for the future Internet.

Key aspects that have characterized this experimental work are presented:

- First, a description of the approach taken at designing and implementing the software components that are part of the architecture prototype used to evaluate and distribute the presented concepts.
- Second, the base technologies developed, giving an overview of how they can be used as the base for more advanced services and applications.
- Third, an overview of the experimental testbeds used for architecture evaluation, including considerations on different approaches for incremental deployment to simplify adoptability of the platform.
- Finally, two examples of how such technologies could be deployed on a nation wide testbed to analyze an end-to-end service built on top of the architecture features.

6.1 Architecture Validation Approach

Validating the design of a novel architecture such as MobilityFirst requires a comprehensive effort that spaces across different experimental techniques. This is due to the different requirements and goals that are part of the process. In order to understand the value of alternative experimental testbeds it is indeed important to identify the particular functional aspects (scale, performance, protocol validity, etc.) that need to be evaluated. In many cases, performance verification at scale is still best suited for simulated environments, such as NS3. This approach applies well to classical network problems such as aggregate routing evaluations looking at different metrics such as protocol overhead and achievable throughput. A tradeoff with the scale of the experiments can be desirable in order to obtain higher levels of realism. Performance critical systems and elements of the architecture might require this approach in order to validate their feasibility. Finally, even higher levels of realism can be obtained by means of deploying the architecture with real end-users that can interact with the network and the deployed services through specific application.

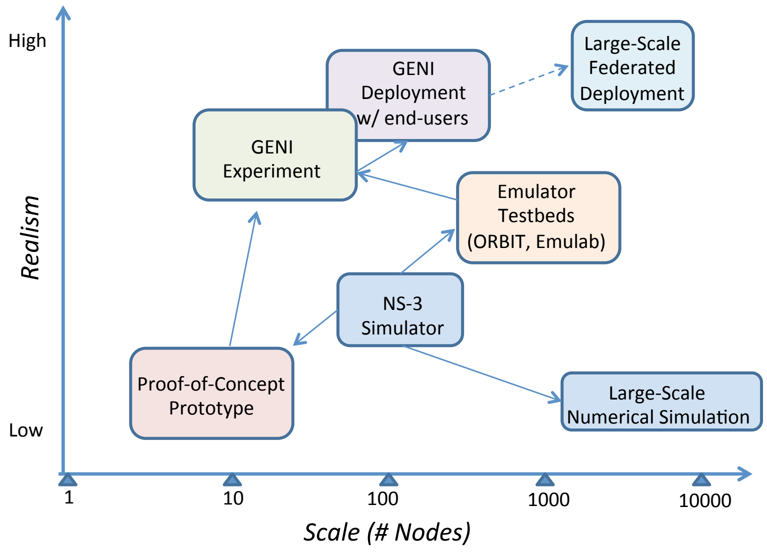

High
Low
Realism
Large-Scale Federated Deployment
GENI Deployment w/ end-users
GENI Experiment
Emulator Testbeds (ORBIT, Emulab)
NS-3 Simulator
Proof-of-Concept Prototype
Large-Scale Numerical Simulation
1
10
100
1000
10000
Scale (# Nodes)

Figure 6.1: Realism vs scale provided by different network evaluation methods.

Figure 6.1 summarizes realism and scale achieved by different evaluation methods/testbeds, and the typical sequence of simulation to testbed evaluation to large-scale user trials. The focus of this works falls in the right side of such graph. Starting from the initial validation of the architecture routing protocols that were performed through simulation models [23, 31], we developed the main components that constituted our protocol stack implementation that has been the basis of experiments performed on two different testbeds: ORBIT and GENI.

A set of key guidelines have been followed throughout the development of the technologies that will be presented:

1. **Develop from scratch**: in order to maximize the adaptability of the software to different deployment scenarios, it was decided to avoid reliance on current IP infrastructure and support for both independent and overlay architecture deployments.

2. **High portability**: as technologies and operating systems quickly evolve, it was aimed to achieve the lowest level of dependency from specific components, minimizing dependence from system specific software libraries.

3. **Adaptability to multiple environments**: supporting the widest possible set of devices (e.g. laptops, smartphones) and communication technologies (e.g. WiFi, LTE) to provide deployment flexibility.

4. **Easy control and results collection**: implement a cohesive control framework integrated in all components, that could enable architecture-wide collection of networking data and statistics and enable real-time visualization of system events.

6.2 Key Developed Technologies

In order to move towards testbed based experimentation we needed to develop a prototype that included the main components that are part of the designed architecture. As the MobilityFirst project addresses the feasibility of building systems and networks in a clean-slate design, it requires the development of such components from scratch. The result of this efforts consisted in three main tools: a GNRS implementation based on DMap's design [23], a Click [69] based software router and a multiplatform protocol stack and network API for clients. Applications and network services can be implemented as extensions of these basic elements. Moreover, we developed the necessary support to automate experimentation using the OMF [99] framework and provide statistic collections through OML [100].

Global Name Resolution Service. A GNRS implementation has been written in Java to provide a hardware and operating system agnostic implementation. Wherever possible, standard libraries are utilized to provide the required functionality, and only the application logic needed to be written by hand. The server is organized into several individual modules: network access, GUID mapping, persistent storage, and application logic. The application logic serves as a central point of coordination within the framework of the GNRS server daemon. The network access component ensures that the GNRS server is able to operate over any networking layer/technology without changes to the core code. This replaceable component currently supports IPv4 and MF routing. The GUID mapping module, relying partly on a networking implementation, enables the server to determine the remote GNRS hosts responsible for maintaining the current bindings of GUID values. Persistent storage is handled independently from the rest of the server and exposes only a very simple interface, mapping to the application messages available in the protocol. A BerkeleyDB provides both in-memory and on-disk storage for GUID bindings.

Routers. Software routers are implemented as a set of routing and forwarding elements within the Click [69] modular router. The router implements dynamic-binding using GNRS, hop-by-hop transport, and storage-aware routing. It integrates a large storage, an in-memory *hold buffer*, to temporarily hold data blocks when destination endpoints during short-lived disconnections or poor access connections. For dynamic in-network binding of names (GUIDs) to network addresses (NA), the router is closely integrated with the in-network GNRS by attaching to a local instance of the distributed service, which is often co-resident on the physical device, but can also be hosted on separate co-located node. A particular instance of this system, implements what we call a MobilityFirst access router, a router providing access connectivity to clients. Access routers also implement a rate monitoring service that tracks the available bandwidth for each attached client. For WiMAX networks, the rate is obtained by querying the

WiMAX base station when possible which exports the most recent downlink bitrate allocated to each client by the scheduler based on a client's location, client offered traffic, and overall load on the BSS. A similar rate monitoring capability is implemented for WiFi Access Points using standard 802.11 netlink configuration utilities. Thanks to the modular structure of Click, we are able to extend the software implementation with additional logic modules to support programmable network services. The router software also collects statistics at different layers of the protocol stack that can be reported through the use of an OML-based monitor that runs within the router and periodically transmit data to a remote (or local) server. Figure 6.2 provides an abstract representation of the data processing pipeline inside the router.

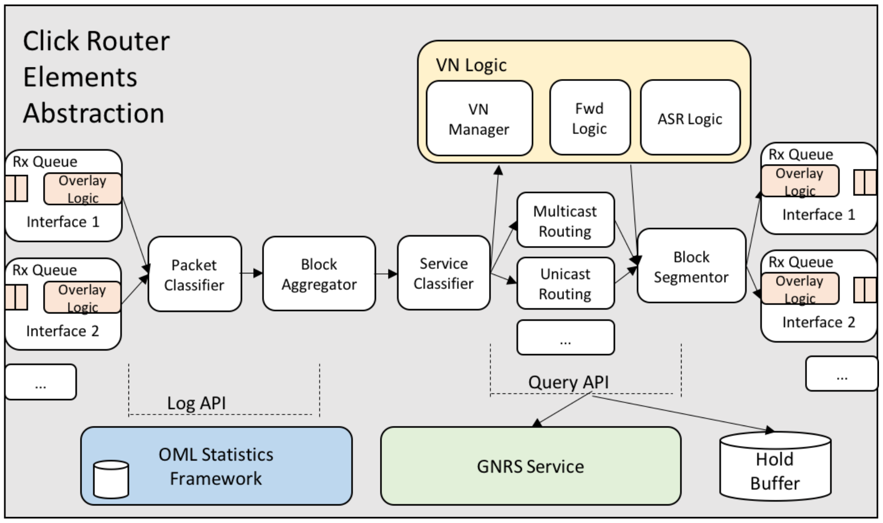


Figure 6.2: Click router based block diagram.

Host Stack and API. The host network protocol stack has been implemented on Linux and Android platforms as a user-level process built as an event-based data pipeline. The stack is composed of a flexible end-to-end transport to provide message level reliability, the name-based network protocol including the GUID service layer, a reliable link data transport layer, and a policy-driven interface manager to handle multiple concurrent interfaces. The device-level policies allow the user to manage how data is multiplexed across one or more active interfaces. The previously introduced socket API [32] is available both as C/C++ and JAVA libraries and implements the name-based service API which include the primitives *send, recv, and get* and a set of meta-operations available for instance to bind or *attach* a GUID to one or more NAs, configure transport parameters in the stack, or to request custom delivery service types such as multicast, anycast, multihoming, or in-network compute. Similarly to the router implementation, the protocol stack collects and optionally reports traffic and resource statistics to a OML backend data repository.

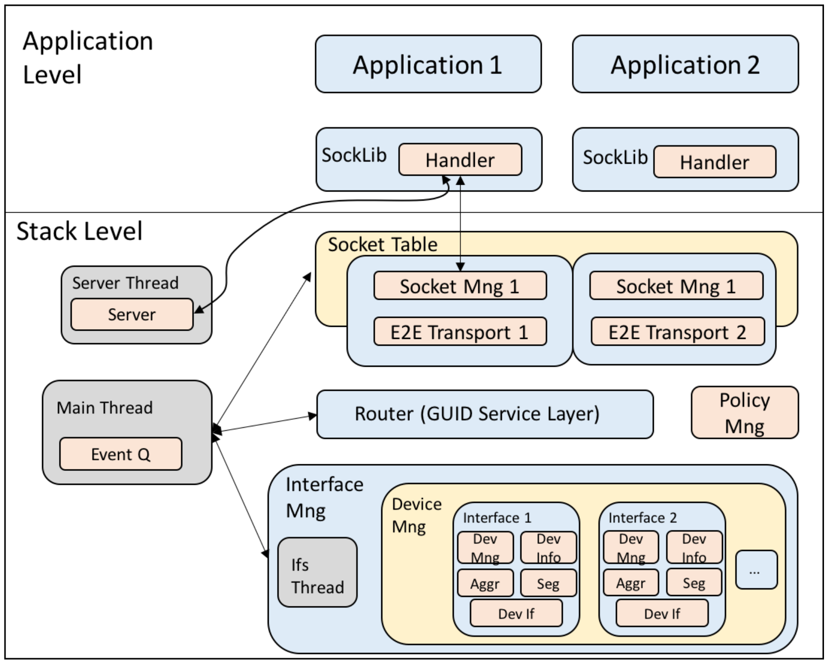


Figure 6.3: Client host stack block diagram.

Other Services. All the other services described throughout this work, have been implemented as extensions of the these core three components. A few key examples are: 1) the *NOVN* framework has been implemented as a set of Click elements that naturally fit into our router prototype; allowing for quick processing of virtual network control and data packets. 2) The same applies for the *NOMA* multicast design, that has now been integrated into the Click based software. 3) The API and network stack were used as the based for implementing a variety of applications including but not limited to a port of the *iperf* tool, content and context applications and a variety of demos [58, 101–106].

6.3 Deployment Scenarios

While Section 6.1 outlined how all these components have been designed with flexibility in mind trying to reduce dependency from specific systems to a minimum, it is important to understand how they could be used in a variety of scenarios. Throughout the development of the described technologies, two large scale testbeds, i.e. ORBIT [72] and GENI [59], have been used to deploy the different components and provide a thorough evaluation. This Section will describe the deployment process on such testbeds, together with some considerations on how the same software pieces could be incrementally deployed on top of the current infrastructure.

6.3.1 Support at Scale Testbeds and Experimental Research

In an effort to provide openly available tools for researchers to experiment with the prototype of the MobilityFirst architecture, we organized all the developed material over the project years under a unique prototyping framework. This framework includes the following components: a) all the prototype source code available both as open source software both as ready-to-run Debian packages; b) documentation organized in the form of wiki to support the understanding of the available material in connection to the concepts at the base of the architecture; c) automated tools to deploy and perform experiments in multiple research testbeds, including ORBIT and GENI; d) a web based tool to be used to track experiments through live monitoring of important events or post-processing statistics experienced during prototype runs.

The two aforementioned testbeds, i.e. ORBIT and GENI, provide a perfect platform for experimenters to have high control on different possible deployment setups for the presented software.

ORBIT. The Orbit testbed is a two-tier wireless network emulator/field trial designed to achieve reproducible experimentation. Its main facility is the radio grid testbed which uses a 20x20 two-dimensional grid of programmable radio nodes which can be interconnected into specified topologies both using reproducible wireless channel models allowing fine grained control over connectivity resources, both using a fully connected

1Gbit ethernet based layer 2 network. Thanks to its large set of resources it provides a perfect environment to support realistic evaluation of protocols and applications up to medium scale.

GENI. The GENI nationwide testbed offers an infrastructure with Internet2 and NLR backbones connecting several university campuses. The wide area hosts, interconnected by a Layer 2 1/10 Gbit core network allows for a realistic deployment and evaluation of MobilityFirst architecture and protocols. The GENI network has been used extensively for validation and evaluation of the MobilityFirst protocol stack. GENI provides the necessary scale and geographic distribution necessary to test key design features such as name resolution and inter-domain routing. GENI also makes it possible to transition from technical experiments to service trials by bringing in opt-in users in different campus networks across the country. Several GENI sites, including the site at Rutgers, have wireless deployments which are equipped to support real-world mobility experiments over a variety of access technologies including WiFi, WiMax and LTE.

While the two testbeds provide different capabilities, both testbeds played a key role in evaluating the presented components. ORBIT was fundamental for evaluating different scenarios that required fine grained control over the wireless resources; GENI provided the right at scale environment for more realistic deployment scenarios. In an effort to provide details on how end-to-end services could be deployed, the focus of the remaining part of the chapter will be given to how the presented components have been deployed on top of the GENI infrastructure.

6.3.2 Considerations for Future Adoptability

Throughout this work, it has been presented how the different components design could be employed to deploy a Named-Object based architecture, starting from the assumption that each packet network header provide space for accessing the names, addresses and service identifiers that enable hybrid routing. While this simplifies the framework design, and allows operating without the need for any additional overlay protocols, a few considerations have to be taken on how this could be done on current networks.

Three different approaches that could be used to incrementally deploy the framework on top of the existing infrastructure have been considered.

Overlay. Fully overlay approaches represent a flexible way for deploying experimental networks and protocols on top of the existing infrastructure. Through encapsulation of network packets on top of UDP packets and tunneling across participating nodes, they allow for the quickest solution to implement experimental protocols on top of the existing infrastructure. With this solution, flexibility and simplicity come at the cost of additional overhead. The named-object abstraction helps taking advantage of the abstraction layers presented to simplify the implementation requirements for such solutions. First, as the network address that represent tunnels are overloaded into the GNRS, no need to define a-priori tunnels is required, leaving forwarding decisions to be dynamically resolved at running time, as needed. This approach is similar in spirit to LISP [4], where multiple encapsulated headers can be used to traver networks and reach participating routers, but extends the base name space to provide the more advanced named-object abstraction.

Header Tags. Tag based approaches exploit flat unique identifiers placed at different layers of the network stack to uniquely identify packet flows. Example of this are VLANs and MPLS. The core issue with these solutions is the limited scope of application in which they can be exploited, as the employed tags are limited in size and have validity only within a single network. For this reason they can mostly be used to support single domain solutions. Even with this limitation in mind, VLAN like solutions, for example, could be used to implement cut through switching to implement fast traversing of single domain networks [107].

SDN Based. Originally presented as a tool for network experimentation [98], Software Defined Networking can be a technology enabler for a native solution to the Mobility-First architecture deployment. Current OpenFlow [108] based solutions are still strictly binded to the TCP/IP protocol stack and would require approaches similar to the ones just presented in the previous paragraphs (e.g. overlay). But as new SDN solutions as [109] become commercially available or as the OpenFlow protocol is extended to support different classes of network, MobilityFirst could be incrementally deployed, starting from single SDN enabled islands interconnected using overlay solutions.

Following the spirit of flexible deployability on top of multiple experimental scenarios, the implemented technologies have been enabled with interface abstractions that can smartly adapt to different networking environments. Recalling Figures 6.2 and 6.3, both technologies have been designed to provide interchangeable *Interface* classes that can implement different deployment scenarios. These include: a) native support of the MobilityFirst protocols on top of a L2 network and overlay support both on top of b) barebone IP network or c) a ful overlay solution on top UPD. Thanks to this support for a wide range of overlay modes, it was possible to deploy the components on top of different access and network technologies, such as the GENI WiMax and LTE infrastructure.

6.3.3 An End-to-End Deployment Realization

The GENI nationwide testbed offers an infrastructure with Internet2 and NLR backbones connecting several university campuses. The wide area hosts, interconnected by a 1/10 Gbit core network allows for a realistic deployment and evaluation of MobilityFirst architecture and protocols. Participation from access networks and mobile clients at collaborating campuses when combined with deployments at the GENI core can establish reasonably large size networks of the order 10s to a few hundred nodes with realistic wide-area network conditions.

Experimental Setup. A long-running deployment of the architecture was set up on a GENI slice (virtual network) starting in 2013 and is still being used for evaluations at the time of this writing. As part of this effort, the prototype components were deployed at seven GENI sites as shown in Figure 6.4. The routers, naming servers, and applications run on Xen VMs (total 14, 2 VMs per site) each with 1 GB memory and one 2.09 GHz processor core. At the Rutgers site we also provision a raw node to run the transcoding server. Each router was configured with 1 or 2 interfaces depending on their role as core router or as an access/edge router, respectively. All routers had a core-facing interface connected to a layer-2 network that connected all seven sites. This was setup using a multi-point VLAN feature provided by Internet2's Advanced Layer-2 Service (AL2S). Routers at three sites (viz. Wisconsin, Rutgers, NYU) were configured with a second interface connecting to the local wireless network (WiMAX). Mobile wireless or emulated clients connected to network through this interface. Routers were each configured with 500 MB of hold buffer space, and had access to a GNRS service instance co-located on the same node. The GNRS service run at all seven sites using a replication factor of k=3, achieving a 95th percentile lookup latency of under 80ms.

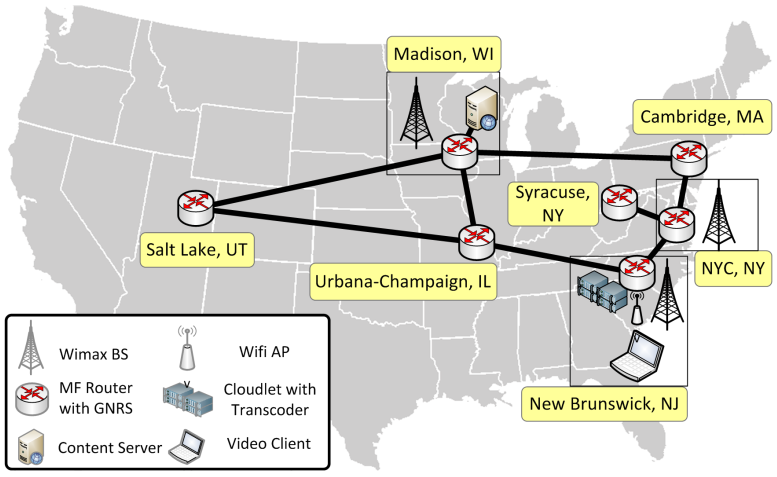


Figure 6.4: Prototype components deployed on the GENI testbed

Exploiting the deployed network, different experimental setups have been deployed over the years to showcase the features of the architecture [58,101–106]. As part of this work, two representative application deployments are presented.

In-Network Transcoding as a Compute Service. Traffic from mobile wireless networks has been growing at a fast pace in recent years and poses significant challenges to service providers in scale and efficiency of data delivery. Streaming video, in particular, is a perfect example of a service that could benefit from our approach. At the current state, most used protocols (e.g. DASH) rely on the ability of a client to estimate the available bandwidth, a task arguably very difficult, especially under wireless and mobile environments. By having an in-network service that would dynamically adapt the encoded bitrate of delivered content according to available bandwidth at the access link would provide a system to move the adaptation logic where that information is more easily accessible. Section 3.2 introduced our solution to natively support in-network computing resources using the named-object abstraction. A realization of such service was designed to implement an in-network service that does rate adaptation when delivering video streams to mobile devices that experience variable connection quality. The in-network compute architecture consisted of: a new network stack and socket API for hosts that implements the service interface used by the end hosts of the system, a software router that implements the hybrid GUID and NA based forwarding and storage-aware routing protocols, and a computing engine/platform that presents an open API for configuring and running in-network services.

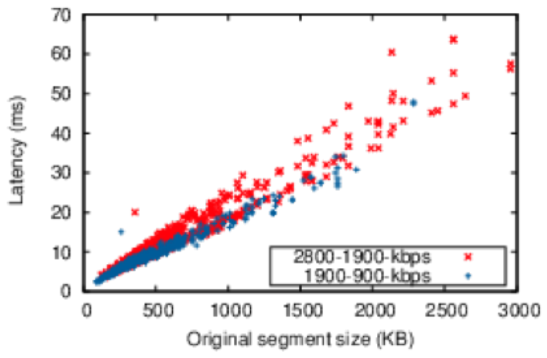


Figure 6.5: Transcoder response time

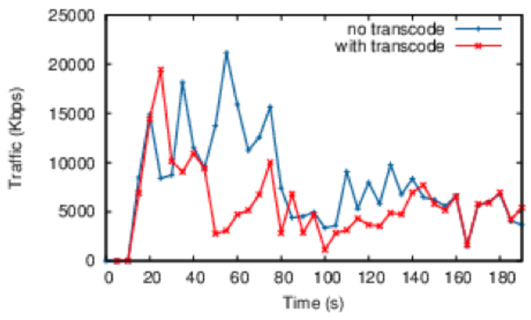


Figure 6.6: Client traffic reduction with transcode

To evaluate our solution, we used the long-running GENI deployment described above to run the in-network rate-adaptation service for a DASH video streaming application, as shown in Figure 6.4. In order to do so, we modified the VLC DASH plugin to use the novel network API presented. The DASH-enabled content server was run at the Wisconsin site, and the client ran at the Rutgers WiMAX network. The rate-adaptation service run within a cloudlet co-located with the Rutgers edge router and was instantiated using the PacketCloud [110] framework. Figure 6.5 shows the overhead in introducing in-network processing for a video stream under 2 different rate-adaptations. While the overhead can be reduced by making right hosting choices for the compute layer, it can also be traded off against the edge bandwidth conditions to dynamically decide benefit of rate-adaptation. In our scheme, the clients access link is monitored using a routing layer service at the access router and the rate-adaptation is dynamically invoked if the access bandwidth drops below the server encoded rate. For the demonstrations, the drops in access bandwidth were emulated by adjusting bandwidth reports, to simulate link quality variation from client mobility. Figure 6.3.3

shows the reduction in client traffic with transcoding.

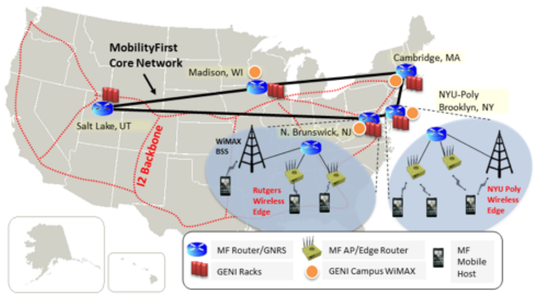


Figure 6.7: Deployment at five rack sites on the GENI wide-area and edge testbeds at Rutgers and NYU Poly (shown expanded).

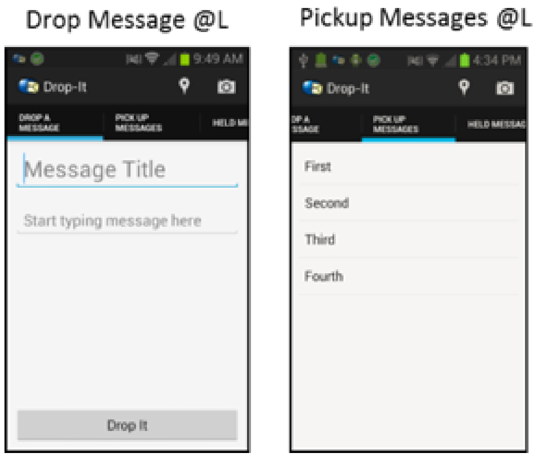


Figure 6.8: The GUI for the Drop It application showing the message drop and pickup screens

Context Based Communications. A contextual messaging application – Drop It – was developed using name-based networking abstractions provided by MobilityFirst, which allows users to drop messages at particular locations, and to pick up messages left by others at the same location. Our context services solution allows locations (contexts, in general) to be assigned unique names (a GUID globally unique ID) which help identify them for network operations such as send, recv or get (for named content retrieval). Locations in physical space can be defined (or fenced) by a set of GPS coordinates, for example, and a persistent GUID can be assigned to them by a well-known service. Next, by maintaining meaningful address mappings for a location GUID in the GNRS, endpoints can send and receive messages to/from this context. For instance, a mapping of location GUID to the set of all phones that dropped messages at that particular location can enable a pure peer-to-peer realization of the contextual messaging service, where the “pick-up” can implemented as an efficient multicast request to each of the phones by using MobilityFirsts get API. It is also possible to realize alternate approaches to pure p2p, where in-network message caches could enable a more robust operation when phones go offline.

The demonstration was run across five of the seven sites within the long-running network deployment (shown in Figure 6.7). The two edge sites at NYU Poly and Rutgers WINLAB, hosted both WiFi and GENI WiMAX access networks that were connected back to the GENI core. Ten Android phones (some with dual WiFi/WiMAX interfaces), each running the network protocol stack and the "Drop It" application (shown in Figure 6.3.3) were carried around by volunteers (except two which were static at Rutgers and remotely accessible) who performed message drop and pick-up operations at the several preset locations on the demo floor. Each location was marked with a QR-code tag that encoded the locations GUID and was directly scanned by the app to retrieve the context GUID. We used QR codes to identify locations primarily due to the difficulties of using GPS indoors at the demo floor.

Chapter 7

Conclusions

The aim of this thesis work has been to design and develop new networking techniques aimed at the deployment and support of advanced services in the future Internet. Much of the effort has been dedicated to the development of different protocol components that enhance and simplify service creation in the future Internet, starting from the introduction of the central architectural concept of *Named-Object* based networking and the power that lies behind it. Looking at the different architectures presented over the years, a set of fundamental abstractions have been defined, providing a comprehensive analysis of their properties and how they could be met. This was centered around the MobilityFirst architecture in which the "narrow waist" of the protocol stack is based on Named-Objects which enable a broad range of capabilities in the network. This was followed up with a specific set of network service APIs that provide full access to the proposed abstractions supported by MobilityFirst. Using performance benchmarks and the implementation of representative use cases it was shown that the abstractions enabled by the new API are flexible and can enable efficient and robust versions of present and future applications.

The work then moved to the set of services that will be required by the future mobile Internet and that due to different shortcomings are hardly supported by the current TCP/IP Internet architecture. These included: i) Multicast services, ii) Content services, iii) In-network compute, and finally iv) Context services. For each of these services, appropriate abstractions enabled by the Named-Object architecture have been presented and a use case based prototype evaluation has been provided. The results show the feasibility of providing a broad range of services with good performance and reasonable protocol overhead.

Starting from the above abstractions analysis and the newly introduced services, the third chapter of the work, focused on how such new services can be made available to the end-users of the network. Considering first the expected requirements for such systems, a new transport layer service has been presented. The new designed protocol can seamlessly support a set of distinctive features based on use of names and in-network reliability techniques. Using the developed prototype components, experimental results show that for a few representative scenarios including mobile data delivery, web content retrieval, and disconnected/late binding service, the new systems can be exploited to reduce the impact of complex operations improving performance for the end users of the network.

It was then analyzed how advanced advanced cloud services can be supported in the proposed Named-Object architecture. In particular, the concept of named-objects has been extended to natively support virtual network identifiers. It has been shown that the virtual network capability can be designed by introducing the concept of a “Virtual Network Identifier (VNID)” which is managed as a Named-Object. Further, the presented design supports the concept of Application Specific Routing (ASR) which enables network routing decisions to be made with awareness of application parameters such as cloud server workload. Experimental results show that the new framework provides a clean and simple logic for defining and managing virtual networks while limiting the performance impact produced by the additional overhead generated by running such system. Moreover, the potential of ASR was demonstrated through a based cloud service use case deployment.

The last chapter of the work bridged together the whole study and provided considerations on how the different components presented could be merged into a single end-to-end realization. As part of this effort, several scenarios including advanced computing and context-aware services have been deployed on a nation wide testbed. While this led to demonstrate that the developed prototype and the protocols that it implements are capable of providing a sustainable framework, future research will be key to further analyze how the named-object abstraction could be the innovative factor that will push the future Internet. In particular, incremental deployment scenarios will have to be identified to provide an approachable release timeline for the architecture. A potential solution will be to focus on edge access networks, where services such as in-network computing and context and multicast services for IoT will have a fundamental role in the future of network communications, providing the perfect environment for island deployments.

References

[1] A. G. Valkó, "Cellular ip: a new approach to internet host mobility," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 1, pp. 50–65, 1999.

[2] "Ip mobility support for ipv4," http://tools.ietf.org/html/rfc3344.

[3] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson, "Host identity protocol," *RFC 5201, April*, 2008.

[4] D. Farinacci, D. Lewis, D. Meyer, and V. Fuller, "The locator/id separation protocol (lisp)," 2013.

[5] E. Nordstrom *et al.*, "Serval: An end-host stack for service-centric networking," *Proc. 9th USENIX NSDI*, 2012.

[6] J. Su *et al.*, "Haggle: Seamless networking for mobile applications," in *UbiComp 2007: Ubiquitous Computing.* Springer, 2007, pp. 391–408.

[7] A. C. Snoeren and H. Balakrishnan, "An end-to-end approach to host mobility," in *Proceedings of the 6th annual international conference on Mobile computing and networking.* ACM, 2000, pp. 155–166.

[8] B. Y. Kimura and H. C. Guardia, "Tips: wrapping the sockets api for seamless ip mobility," in *Proc. of Applied computing.* ACM, 2008.

[9] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: a platform for high-performance internet applications," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 3, pp. 2–19, 2010.

[10] A. V. Abhigyan Sharma and R. Sitaraman, "Distributing content simplifies isp traffic engineering," in *Proc. of ACM SIGMETRICS*, 2013.

[11] D. G. Andersen *et al.*, "Accountable Internet Protocol (AIP)," in *Proc. ACM SIGCOMM*, August 2008.

[12] C. V. Mobile, "Cisco visual networking index: global mobile data traffic forecast update, 2015–2020," *San Jose, CA*, vol. 1, 2016.

[13] B. Hesmans, H. Tran-Viet, R. Sadre, and O. Bonaventure, "A first look at real multipath tcp traffic," in *International Workshop on Traffic Monitoring and Analysis.* Springer, 2015, pp. 233–246.

[14] J. Pan, R. Jain, S. Paul, and C. So-In, "Milsa: A new evolutionary architecture for scalability, mobility, and multihoming in the future internet," *Selected Areas in Communications, IEEE Journal on*, vol. 28, no. 8, pp. 1344–1362, 2010.

[15] V. Jacobson *et al.*, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies.* ACM, 2009, pp. 1–12.

[16] L. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[17] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 181–192.

[18] M. Mosko, I. Solis, E. Uzun, and C. Wood, "Ccnx 1.0 protocol architecture," http:// www. ccnx. org/ pubs/ CCNxProtocolArch itecture. pdf, Tech. Rep., 2015.

[19] "Mobility support in ipv6," http://www.ietf.org/rfc/rfc3775.txt.

[20] R. Stewart and C. Metz, "Sctp: new transport protocol for tcp/ip," *Internet Computing, IEEE*, vol. 5, no. 6, pp. 64–69, 2001.

[21] M. S. Gordon *et al.*, "Comet: Code offload by migrating execution transparently," in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation.* USENIX Association, 2012, pp. 93–106.

[22] J. Erman *et al.*, "Network-aware forward caching," in *Proceedings of the 18th international conference on World wide web.* ACM, 2009, pp. 291–300.

[23] T. Vu *et al.*, "Dmap: a shared hosting scheme for dynamic identifier to locator mappings in the global internet," in *Distributed Computing Systems (ICDCS), 2012.* IEEE, 2012, pp. 698–707.

[24] A. Sharma, X. Tie, H. Uppal, A. Venkataramani, D. Westbrook, and A. Yadav, "A global name service for a highly mobile internetwork," in *ACM SIGCOMM*, 2014.

[25] D. Farinacci, C. Liu, S. Deering, D. Estrin, M. Handley, V. Jacobson, L. Wei, P. Sharma, D. Thaler, and A. Helmy, "Protocol independent multicast-sparse mode (pim-sm): Protocol specification," 1998.

[26] J. Moy, "Rfc 1584–multicast extensions to ospf," *SRI Network Information Center*, 1994.

[27] S. Farrell *et al.*, "When tcp breaks: Delay- and disruption- tolerant networking," *IEEE Internet Computing*, vol. 10, no. 4, pp. 72–78, 2006.

[28] M. C. Chan and R. Ramjee, "Tcp/ip performance over 3g wireless links with rate and delay variation." *Wireless Networks*, pp. 81–97, 2005.

[29] M. Li, D. Agrawal, D. Ganesan, and A. Venkataramani, "Block-switched networks: a new paradigm for wireless transport," in *Proc. of NSDI*, 2009.

[30] S. Gopinath, S. Jain, S. Makharia, and D. Raychaudhuri, "An experimental study of the cache-and-forward network architecture in multi-hop wireless scenarios," in *Proc. of LANMAN*, 2010.

[31] S. C. Nelson, G. Bhanage, and D. Raychaudhuri, "GSTAR: generalized storage-aware routing for mobilityfirst in the future mobile internet," in *Proc. of MobiArch*. ACM, 2011, pp. 19–24.

[32] F. Bronzino, K. Nagaraja, I. Seskar, and D. Raychaudhuri, "Network service abstractions for a mobility-centric future internet architecture," in *Proceedings of the eighth ACM international workshop on Mobility in the evolving internet architecture*. ACM, 2013, pp. 5–10.

[33] F. Bronzino and D. Raychaudhuri, "Abstractions and solutions to support smart-objects in the future internet," in *SmartObjects 2016*. ACM, 2016.

[34] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the ip multicast service and architecture," *Network, IEEE*, vol. 14, no. 1, pp. 78–88, 2000.

[35] D. Meyer and B. Fenner, "Multicast source discovery protocol (msdp)," 2003.

[36] P. Radoslavov, D. Estrin, R. Govindan, M. Handley, S. Kumar, and D. Thaler, "The multicast address-set claim (masc) protocol, rfc-2909," Tech. Rep., 2000.

[37] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron, "SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," *JSAC*, pp. 1489–1499, 2002.

[38] D. A. Tran, K. Hua, and T. Do, "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming," in *INFOCOM*, 2003.

[39] D. Raychaudhuri, K. Nagaraja, and A. Venkataramani, "Mobilityfirst: a robust and trustworthy mobility-centric architecture for the future internet," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 16, no. 3, pp. 2–13, 2012.

[40] S. Mukherjee, F. Bronzino, S. Srinivasan, J. Chen, and D. Raychaudhuri, "Achieving scalable push multicast services using global name resolution," IEEE, 2016.

[41] V. Jacobson *et al.*, "Networking Named Content," in *CoNEXT*, 2009.

[42] D. Han, A. Anand, F. R. Dogar, and Others, "Xia: Efficient support for evolvable internetworking." in *USENIX NSDI*, 2012.

[43] K. Bharath-Kumar and J. M. Jaffe, "Routing to multiple destinations in computer networks," *Communications, IEEE Transactions on*, 1983.

[44] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed internet routing convergence," *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, pp. 175–187, 2000.

[45] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013-2018," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf, 2013.

[46] T.-Y. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari, "Confused, timid, and unstable: picking a video streaming rate is hard," in *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM, 2012, pp. 225–238.

[47] J. Chen, R. Mahindra, M. A. Khojastepour, S. Rangarajan, and M. Chiang, "A scheduling framework for adaptive video delivery over cellular networks," in *Proceedings of the 19th annual international conference on Mobile computing & networking*. ACM, 2013, pp. 389–400.

[48] Y. Chen, B. Liu, Y. Chen, A. Li, X. Yang, and J. Bi, "Packetcloud: an open platform for elastic in-network services," in *Proceedings of the eighth ACM international workshop on Mobility in the evolving internet architecture*. ACM, 2013, pp. 17–22.

[49] F. Bronzino, C. Han, Y. Chen, K. Nagaraja, X. Yang, I. Seskar, and D. Raychaudhuri, "In-network compute extensions for rate-adaptive content delivery in mobile networks," in *International Workshop on Computer and Networking Experimental Research using Testbeds (CNERT 2014)4*.

[50] T. Stockhammer, "Dynamic adaptive streaming over http: standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 133–144.

[51] C. Müller, S. Lederer, and C. Timmerer, "An evaluation of dynamic adaptive streaming over http in vehicular environments," in *Proceedings of the 4th Workshop on Mobile Video*. ACM, 2012, pp. 37–42.

[52] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz, "A comparison of quality scheduling in commercial adaptive http streaming solutions on a 3g network," in *Proceedings of the 4th Workshop on Mobile Video*. ACM, 2012, pp. 25–30.

[53] C. Müller and C. Timmerer, "A vlc media player plugin enabling dynamic adaptive streaming over http," in *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 2011, pp. 723–726.

[54] "Global environment for networking innovations (GENI)," http://www.geni.net.

[55] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over http dataset," in *Proceedings of the 3rd Multimedia Systems Conference*. ACM, 2012, pp. 89–94.

[56] "Big Buck Bunny movie," http://bigbuckbunnymovie.org/.

[57] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[58] F. Bronzino, D. Raychaudhuri, and I. Seskar, "Demonstrating context-aware services in the mobilityfirst future internet architecture," in *Proceedings of the First International Conference in Networking Science and Practice (ITC) 28*. IEEE, 2016.

[59] "Global environment for networking innovations (GENI), NSF program solicitation," http://www.geni.net, 2006.

[60] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM TOCS*, 1984.

[61] B. Ahlgren and et Al, "Design considerations for a network of information," in *ACM CoNEXT*, 2008.

[62] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: Seeing the forest for the trees," in *ACM HotNets*. ACM, 2011.

[63] V. Jacobson *et al.*, "Networking named content," in *ACM CoNEXT*. ACM, 2009.

[64] S. Paul, R. Yates, D. Raychaudhuri, and J. Kurose, "The cache-and-forward network architecture for efficient mobile content delivery services in the future internet," in *Innovations in NGN. First ITU-T Kaleidoscope Academic Conference*. IEEE, 2008.

[65] K. Su, F. Bronzino, K. Ramakrishnan, and D. Raychaudhuri, "Mftp: A clean-slate transport protocol for the information centric mobilityfirst network," in *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 2015, pp. 127–136.

[66] A. Erramilli and R. P. Singh, "A reliable and efficient multicast for broadband broadcast networks," in *ACM Workshop on Frontiers in Computer Communications Technology*, 1988.

[67] S. Mukherjee, K. Su, N. B. Mandayam, K. Ramakrishnan, D. Raychaudhuri, and I. Seskar, "Evaluating opportunistic delivery of large content with tcp over wifi in i2v communication," *IEEE LANMAN*, 2014.

[68] M. Gerla and L. Kleinrock, "Flow control: A comparative survey," *IEEE Transactions on Communications*, 1980.

[69] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," in *ACM Transactions on Computer Systems*. Citeseer, 2000.

[70] L. Zhang, "Why tcp timers don't work well," in *ACM SIGCOMM*, 1986.

[71] I. Psaras and V. Tsaoussidis, "Why tcp timers (still) don't work well," *Computer Networks*, 2007.

[72] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols," in *Wireless Communications and Networking Conference, 2005 IEEE*, vol. 3. IEEE, 2005, pp. 1664–1669.

[73] hostapd, http://wireless.kernel.org/en/users/Documentation/hostapd.

[74] netem: network emulation tool, http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[75] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall, "How speedy is spdy," in *USENIX NSDI*, 2014.

[76] Alexa: the top 500 sites on the web, http://www.alexa.com/topsites.

[77] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, P. Ingram, E. Jackson *et al.*, "Network virtualization in multi-tenant datacenters," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 203–216.

[78] T. Wood, K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, "Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines," in *ACM Sigplan Notices*, vol. 46, no. 7. ACM, 2011, pp. 121–132.

[79] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[80] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, 2005.

[81] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 3–12, 2003.

[82] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In vini veritas: realistic and controlled network experimentation," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4, pp. 3–14, 2006.

[83] V. Valancius, N. Feamster, J. Rexford, and A. Nakao, "Wide-area route control for distributed services." in *USENIX Annual Technical Conference*, 2010.

[84] X. Wu and J. Griffioen, "Supporting application-based route selection," in *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*. IEEE, 2014, pp. 1–8.

[85] K. Psounis, "Active networks: Applications, security, safety, and architectures," *IEEE Communications Surveys*, vol. 2, no. 1, pp. 2–16, 1999.

[86] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 231–242.

[87] F. Bronzino, D. Raychaudhuri, and I. Seskar, "Experiences with testbed evaluation of the mobilityfirst future internet architecture," in *2015 European Conference on Networks and Communications (EuCNC)*.

[88] "iperf - the network bandwidth measurement tool," https://iperf.fr/.

[89] K. Nakauchi and Y. Shoji, "WiFi Network Virtualization to Control the Connectivity of a Target Service," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 308–319, June 2015.

[90] ——, "vbs on the move: Migrating a virtual network for nomadic mobility in wifi networks," in *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on.* IEEE, 2016, pp. 277–282.

[91] S. Paul and S. Seshan, "Virtualization and Slicing of Wireless Networks," *GENI Design Document 06-17, GENI Wireless Working Group*, September 2006.

[92] X. Wang, P. Krishnamurthy, and D. Tipper, "Wireless Network Virtualization," *Proc. ICNC '13*, January 2013.

[93] F. Fu and U. C. Kozat, "Stochastic Game for Wireless Network Virtualization," *IEEE/ACM Transactions on Networking*, vol. 21, no. 1, February 2013.

[94] C. Liang and F. R. Yu, "Wireless Network Virtualization: A Survey, Some Research Issues and Challenges," *IEEE Comm. Surveys and Tutorials*, vol. 16, no. 3, July 2014.

[95] M. Yang, Y. Li, D. Jin, L. Zeng, X. Wu, and A. V. Vasilakos, "Software-Defined and Virtualized Future Mobile and Wireless Networks: A Survey," *Mobile Networks and Applications*, September 2014.

[96] Y. Shoji, M. Ito, K. Nakauchi, Z. Lei, Y. Kitatsuji, and H. Yokota, "Bring Your Own Network – A Network Management Technique to Mitigate the Impact of Signaling Traffic on Network Resource Utilization –," *Proc. MobiWorld 2014*, January 2014.

[97] K. Nakauchi, Y. Shoji, M. Ito, Z. Lei, Y. Kitatsuji, and H. Yokota, "Bring Your Own Network – Design and Implementation of a Virtualized WiFi Network –," *Proc. IEEE CCNC'14*, January 2014.

[98] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communications Review*, vol. 38, no. 2, April 2008.

[99] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, "Omf: a control and management framework for networking testbeds," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 4, pp. 54–59, 2010.

[100] M. Singh, M. Ott, I. Seskar, and P. Kamat, "Orbit measurements framework and library (oml): motivations, implementation and features," in *Tridentcom 2005.* IEEE, 2005, pp. 146–152.

[101] S. Banerjee *et al.*, "Public safety focus: Connected vehicles assisting first responders," https://www.us-ignite.org/apps/bRzcwAY6mtPENpFqvFFQTd/, 2015.

[102] A. Babu, F. Bronzino, D. Raychaudhuri, and I. Seskar, "Cloud services enhancements through application specific routing in mobilityfirst fia,"

http://groups.geni.net/geni/wiki/GEC22Agenda/EveningDemoSession#CloudServicesEnhancementsThroughApplicationSpecificRoutinginMobilityFirstFIA, 2015.

[103] F. Bronzino, P. Karimi, and I. Seskar, "Introduction to the mobilityfirst fia protocol suite," http://groups.geni.net/geni/wiki/GEC21Agenda/MobilityFirst, 2014.

[104] F. Bronzino, C. Han, Y. Chen, K. Nagaraja, X. Yang, I. Seskar, and D. Raychaudhuri, "In-network compute layer in mobilityfirst future internet architecture fia," http://groups.geni.net/geni/wiki/GEC20Agenda/EveningDemoSession#MobilityFirst, 2014.

[105] F. Bronzino, K. Alimole, K. Nagaraja, D. Raychaudhuri, and I. Seskar, "Context services in mobilityfirst fia," https://www.geni.net/?p=2924, 2013.

[106] F. Bronzino, K. Nagaraja, D. Raychaudhuri, and I. Seskar, "Mobilityfirst network api use in mobile applications," http://groups.geni.net/geni/wiki/GEC16Agenda/EveningDemoSession#MobilityFirstNetworkAPIuseinMobileApplications, 2013.

[107] A. Lara, B. Ramamurthy, K. Nagaraja, A. Krishnamoorthy, and D. Raychaudhuri, "Cut-through switching options in a mobilityfirst network with openflow," in *2013 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE, 2013, pp. 1–6.

[108] "Openflow switch specification, version 1.5.0," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf, 2014.

[109] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 127–132.

[110] Y. Chen, B. Liu, Y. Chen, A. Li, X. Yang, and J. Bi, "Packetcloud: an open platform for elastic in-network services," in *ACM MobiArch*. ACM, 2013.