TOWARD AN INTERNET-PROTOCOL (VERSION 6) DEPLOYABLE INFORMATION-CENTRIC NETWORKING FRAMEWORK

A Dissertation
Presented to
The Academic Faculty

by

Laura Heath

In Partial Fulfillment
Of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2014

TOWARD AN INTERNET-PROTOCOL (VERSION 6) DEPLOYABLE INFORMATION-CENTRIC NETWORKING FRAMEWORK

Approved by:

Dr. Henry Owen, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Raheem Beyah, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Russell Clark
School of Computer Science
Georgia Institute of Technology

Dr. John Copeland
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. George Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yusun Chang
School of Electrical and Computer Engngineering
Georgia Institute of Technology

Date Approved: November 12, 2014

ACKNOWLEDGEMENTS

I wish to thank my husband, Jeffrey Steel, and my parents, James and Nancy Heath, for their unwavering support over the many years that it took to reach this goal. I could not have done it without you.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

The purpose of this research is to create a new information-centric networking (ICN) framework and associated protocol that is implementable in the existing internet with feasible minimal changes to the existing internet infrastructure. We create a protocol that assigns globally unique data item names and embeds these names, plus associated metadata, into an IPv6 header. This technique allows the use of the existing IPsec suite of protocols to mitigate user privacy and security concerns which exist in other existing non-implementable ICN designs. Next, we analyze the layer four functionality which must be provided by a general-purpose transport protocol, and we give an initial implementation which is used by our prototype. Lastly, using the Mininet network virtualization suite, we show that using RFC-compliant IPv6 datagrams as the named content allows information-centric routing and switching to be done using unmodified hardware and software, and that it also ensures backwards compatibility with unmodified networks.

CHAPTER 1

INTRODUCTION

There is a growing realization that some early design decisions while developing the Internet have proven to be significant constraints to the performance and scaling of the network today. In particular, there was an unspoken assumption that all communication was host to host, and each host was one unique computer at one fixed location. Yet today, the majority of the data transfers over the Internet are either data distribution or provision of services—things that are not inherently (or even preferably) single-host to single-host, point-to-point connections. A great deal of network research and development over the last three decades has been devoted to finding ways to make a point-to-point design function in a non-point-to-point way—such as multicasting (one to many transmission), proxies (inserting an intermediary between two hosts in the communications stream), NAT translation (multiple hosts sharing one routable IP address), mirror sites (duplicate hosts in physically separate locations to serve identical content), and more. Each of these configurations have produced a crop of complicated work-arounds, which frequently break other services unexpectedly and lead to yet another *ad hoc* patch.

Many of these work-arounds would be unnecessary if the network could recognize and route by data content rather than host name and location. The fundamental problem is that the Internet socket (IP address and port) has become the (nearly) exclusive means of identifying the endpoints of interprocess communication over the network. This makes the IP address the so-called "thin waist" of the Internet—much innovation and diversity above it (in applications) and below it (in transmission and networking), but essentially no changes to the IP addressing scheme itself. Both applications above it and hardware and firmware below it have been designed with implicit or explicit dependence on the details of IP.

There have been multiple attempts to develop a system that replaces the host-centric interprocess communication with an information-centric one. In chapter 2, we analyze the three most well-known of these systems: the Data Oriented Network Architecture (DONA), the Named Data Network (NDN)/Content Centric Network (CCNx), and the Publish/Subscribe Internet Technology (PURSUIT) program. Yet despite the promise of improved network performance and simpler data handling, none of these systems is in wide use today. The slow and difficult process of shifting from IPv4 to IPv6 has shown that any change at the IP level can cause many unexpected and difficult to resolve problems. This, plus questions about cost, interoperability with legacy systems, and the absence of an economically viable deployment strategy, are the principal problems blocking the use of information-centric networking. Our conclusion is that, while an information-centric network has numerous potential benefits, any actually feasible system must be fully compatible with the current and future IP network hardware and software infrastructure.

We then discuss new developments in the field of software-defined networking and the continuing roll-out of the IPv6 network worldwide have made a fundamental re-working of the IP layer actually achievable for the first time. We specify a method of embedding information-centric networking labels into the standard IPv6 header, which allows information-centric networking packets to coexist on unmodified IPv6 networks.

Our method is shown to be fully RFC-compliant and feasible for use immediately on the commercial internet without modifying either the network infrastructure or the operating system of the end host. We also demonstrate how our system resolves the security and privacy issues inherent in other ICN designs, and we show how using the standard IP Security (IPSec) suite of protocols and a simple ephemeral publisher label system can be used to resolve these concerns.

We then look to the layer four requirements that a transport protocol for general use must meet to ensure reliable, safe and performant operation, In particular, we make comparison between our proposed method and the most common layer four system, transmission control protocol (TCP). We show that our system inherently provides most of the functionality needed, and we show that an initial solution to the question of network congestion control can be implemented by borrowing the methodology used by TCP. We conclude that our protocol is a good candidate for further development and research.

Next, we demonstrate an initial prototype of the protocol in a virtual network, showing that it can be used on the unmodified Linux operating system, off-the-shelf network virtualization software, standard software-defined network controllers and routing protocols, and commercial hardware. This allows the network to provide information-oriented transport services to applications, enabling access to named data items in place of host-oriented connections. Lastly, we compare the performance of this prototype to standard TCP for transferring short, ephemeral data items across a moderate-sized network.

We conclude with a discussion of what items remain to be resolved for this prototype protocol to become sufficiently robust for production-grade traffic. We expect a relatively modest investment in software development will enable us to meet the standard Python interfaces for transport objects, which should allow us to use the CLIP transport layer as a mix-in for higher-level applications. If successful, this would give access to a very robust and extensive ecosystem of features and programs; of particular note are the various cryptographic and network utilities, APIs for many production-grade systems such as databases, and user-level programs such as browsers and media players. Further research opportunities abound as well. We are particularly interested in the theoretical and practical question of improving congestion control. The current practice and methods depend explicitly on data flows being bi-directional, fixed-endpoint flows between two active processes. This is not true for our protocol, and so we believe that significant new findings are possible for a re-architected system.

CHAPTER 2

BACKGROUND/ORIGIN AND HISTORY OF THE PROBLEM

As described in the introduction, multiple organizations have started work on designing a new method of routing based on the identity of the information, rather than the host identity and location. There are three principal approaches to information-centric networking: the Data Oriented Network Architecture (DONA); the Named Data Networking (NDN) project, also called the Content Centric Networking (CCNx) framework; and the Publish/Subscribe Internet Technology (PURSUIT) project, which is an extension of the Publish/Subscribe Internet Routing Paradigm (PSIRP) program.

2.1 Data-Oriented Network Architecture (DONA)

One of the first attempts at information-centric networking was the Data-Oriented Network Architecture (DONA)[1], which built on the work of TRIAD[2]. These researchers' approach to creating routable data items is to completely redesign the Internet naming scheme, replacing the current DNS with an information-centric version. In their terms, domains are replaced by "principals" (publishers of data, identified by a public-private key pair). These principals assign a "label" (a unique numerical name assigned by the principal) to each content item. The content item is then uniquely identified by the number P:L, where P is the hash of the principal's public key. The routing information to the nearest copy of this data is then developed by a route handler (RH) in each internet domain. These RHs have two primary primitives: REGISTER (P:L), which populates the register of content item locations, and FIND (P:L), which functions much as the current DNS lookup, by either returning the physical location of the content item or forwarding the request to its next higher RH for resolution. As with current routing protocols, register entries will be shared with parent RHs and may be shared with peer RHs.

Thus, the information centricity of DONA results from two main innovations: a structured naming convention for data items, linking the principal (owner) of the data to the data itself; and a layer-four method for resolving the content name with a layer-three physical address (in practice, an IP address for the nearest copy of the desired data item). The first innovation, naming data based on its publisher rather than the socket unique to a pair of communicating host processes, has been adopted by all subsequent ICN designs in one form or another (including our own protocol). This reflects the fact that most networking use cases are actually a software process obtaining and operating on a desired block of data, rather than two processes on different end hosts interacting with each other.

The second innovation, the method of mapping the content name to the physical address of the desired data, is more mixed. From a positive standpoint, by making changes only in layer four, it allows the system to operate directly on an unmodified IP network. This has three fundamentally important benefits. First, it allows any end host with connectivity to the Internet to access the system. Any host which knows the IP address of the root RH can resolve the physical location of its desired data, and the system uses the standard IP routing and switching to deliver the data. This means that there is a clear path for incremental roll-out and also guarantees that legacy point to point traffic will not be affected by the new services offered. Second, by using unmodified layer-three delivery methods, DONA ensures that the stability and reachability guarantees of the current networking protocols are preserved in the new system. Third, by converting the content-centric search into a point to point delivery of data over IP, the system allows (but does not require) the use of industry standard encryption and signature techniques to preserve user privacy and security. DONA is the only ICN prototype available which provides these benefits, and in practice, they are absolutely necessary to meet for real-world traffic.

However, the content mapping scheme used in DONA has several significant problems, which sharply limit the system's ability to be implemented at scale. First, the computational burden of registrations on the RHs, particularly the Tier-1 RHs, would be very substantial. Each registration requires computationally expensive public key cryptography to verify the legitimacy of the registrant. (Key revocation lists will also be needed, and administering these lists has been a significant burden in other PKI deployments.) Second, the required routing tables will necessarily be large, as the namespace is very large and not logically linked to the physical addressing; this will only increase in size and complexity if the data are stored in multiple physical locations to improve speed and reliability. Both the original DONA paper and subsequent analysis at [3] have concluded that this type of mapping construct is physically impossible to implement at Tier-1 routers with current hardware. Third, the fact that the name is neither human readable nor predictable from known information will likely require an additional lookup to resolve the name of the desired content. This would require a total of four systems of data lookups to be established, maintained, secured, and queried (principal's name-to-public key binding, key revocation list, file name-to-number resolution, and physical location of the data) prior to the beginning of any data transfer. One additional issue is that, because the data is actually delivered via a point to point transmission between hosts, the system does not lend itself to the pervasive caching of data items in the network, as envisioned by the other ICN designs, or for alternate network delivery architectures (such as publish-subscribe or peer-to-peer). Accordingly, this system is more akin to a content delivery network than a fundamental redesign of the Internet routing paradigm.

2.2 Named Data Networking (NDN)

The Named Data Networking (NDN) project [4-6], sponsored by the National Science Foundation, is another approach to ICN. Initially, this approach abandoned IP numbers entirely as routing primitives. Instead, routers would use a name-based routing protocol, storing the results in a forwarding information base (FIB). Content “chunks” are to be named using a hierarchical, human-readable format (for example, a web page could be named `/edu/gatech/ece/cap/wiki`). Users send information requests, called interests, to the router. The router maintains a Content Store of named information, and if the router has a copy of the requested data already, it transmits the data out on the same interface that the request arrived on. If not, it stores the interface number in a pending interest table (PIT) and forwards the interest according to its FIB (for example, it may have a rule that all unfilled interests for data items from `/edu/gatech/ece/cap/` are forwarded to interface *n*, similar to the concept of a default gateway in IP routing). The NDN envisions that each node in the network will have a significant amount of storage, so that a relatively high percentage of interests are expected to be satisfied locally. Note that this is a much more ambitious redesign of network routing than DONA, and in particular changes the architecture to a store-and-forward concept. The routing is intended to be entirely content-centric and decentralized, with each router following its own local rules for propagating interests to, and servicing interests from, its neighbors. The NDN traffic could ride IP as its transport, but the forwarding decisions would be made based on the data's name and the record of interests at each node (not the IP address of the source or destination).

This initial approach had no routing in the network at all, with requests simply diffusing from node to node until they met with a stored instance of the data requested[7]. Beyond a small number of nodes, however, this approach proved unworkable for two main reasons. First, pending interests would expire (either as an explicit timeout or by being overwritten as the table became overfilled with new interests) before they reached a source of the data, meaning that data chunks were unreachable from some requestors. Second, if interests propagated to every node, regardless of whether it was on a path to the data, the pending interest tables became flooded with unnecessary entries. The problem becomes much worse when facing a malicious user flooding the network with spurious requests as a denial of service attack[8-11] or with attempted cache poisoning[12]. As a result, a number of different researchers have proposed ways to integrate IP routing into the NDN system[13-17], with the current reference system in CCNx being the NDN Link State Routing (NLSR) protocol[18]. While this system does improve the reachability of the network significantly, it comes at the price of abandoning the information-centricity of the routing and forwarding decisions—the network must run both a link-state routing protocol covering all network nodes and also maintain a PIT at each router and switch. It also means that the concept network is, in effect, required to ride an IP network (which must itself have all of the normal network services and infrastructure), which opens the question of what is to be gained by information-centric networking (as compared to, say, adding store-and-forward proxies or an application-layer change such as publish-subscribe). There are additionally significant questions about the behavior of the routing system when interest packets are lost. There is no clear indication when (or if) the router should retransmit interests, and how this would affect either performance or reachability[19]. We also do not know how the various attempts to improve NDN network performance via aggregating related interests[20], tag-based routing[21], collaborative caching of interests and data[22], bundling interests and content packets[23], rate limiting of interest propagation[24], and other methods will impact the stability and reachability of the network. (In contrast, we have both deep theoretical analysis and extensive practical experience with IP, which shows us how to guarantee routing stability and network reachability.)

Two related issues are the question of the size of the PIT and the time it would take to look up the name and determine its forwarding class. Even in the best case, the PIT requires a very large number of entries, with significant churn—some researchers estimate that to support a 20 Gbps link, the PIT would have to contain over 1.5 million entries, with 1.4 million lookups per second, and 900,000 insert and 900,000 delete operations per second[25]. The request then must be matched against the FIB, which maps data chunk names to forwarding classes. These names are typically longer than IP addresses and far less structured; they also are much more mutable (as data is moved around the network the best source will change), which sharply increases the number of read/write operations needed. FIB entries typically run to the tens of millions of lines[26]. Both PIT and FIB lookups and changes are memory operations, typically using a hash table approach[27, 28]. Note that architecture requires every switch and router in the NDN network to have a very significant amount of memory (several gigabytes at least) for the PIT and the FIB, over and above any storage for the content itself, and in addition to the hardware resources needed for the underlying network. Attempting to improve the performance of these memory resources is a principal thrust of ongoing NDN research[29-31], but the best available results have been an order of magnitude slower than line speed[32].

By contrast, the largest IP routing tables in system today (the global, unaggregated tier-1 BGP routing table) has approximately 522,000 entries as of this writing, and with CIDR aggregation this becomes less than 290,000[33]. Most routers and switches need far less than this, because they implement default routes for non-local traffic. Because the memory requirement is so much smaller, and the address space is carefully structured, IP routers and switches store their entire forwarding plane match set in ternary content-addressable memory (TCAMs). These identify the best-match rule in one clock-cycle (*i.e.*, the forwarding decision is made before the entire packet is read off the line, even on the fastest links). As we will show in chapter five, our proposed system uses this already-installed infrastructure without modification, giving us an expected 10:1 speed advantage at the forwarding plane.

A further problem surfaces when trying to internetwork NDN enclaves. The existing, unmodified IP network infrastructure cannot recognize or route NDN datagrams based on their names. If the network provider supplies routers that can serve both as NDN routers and IP routers, the NDN data can traverse an OSPF or IS-IS IP network as a type-length-value (TLV) formatted packet. However, these packets are unrecognizable to IP-only routers; hosts who are not connected to an NDN router cannot either send or

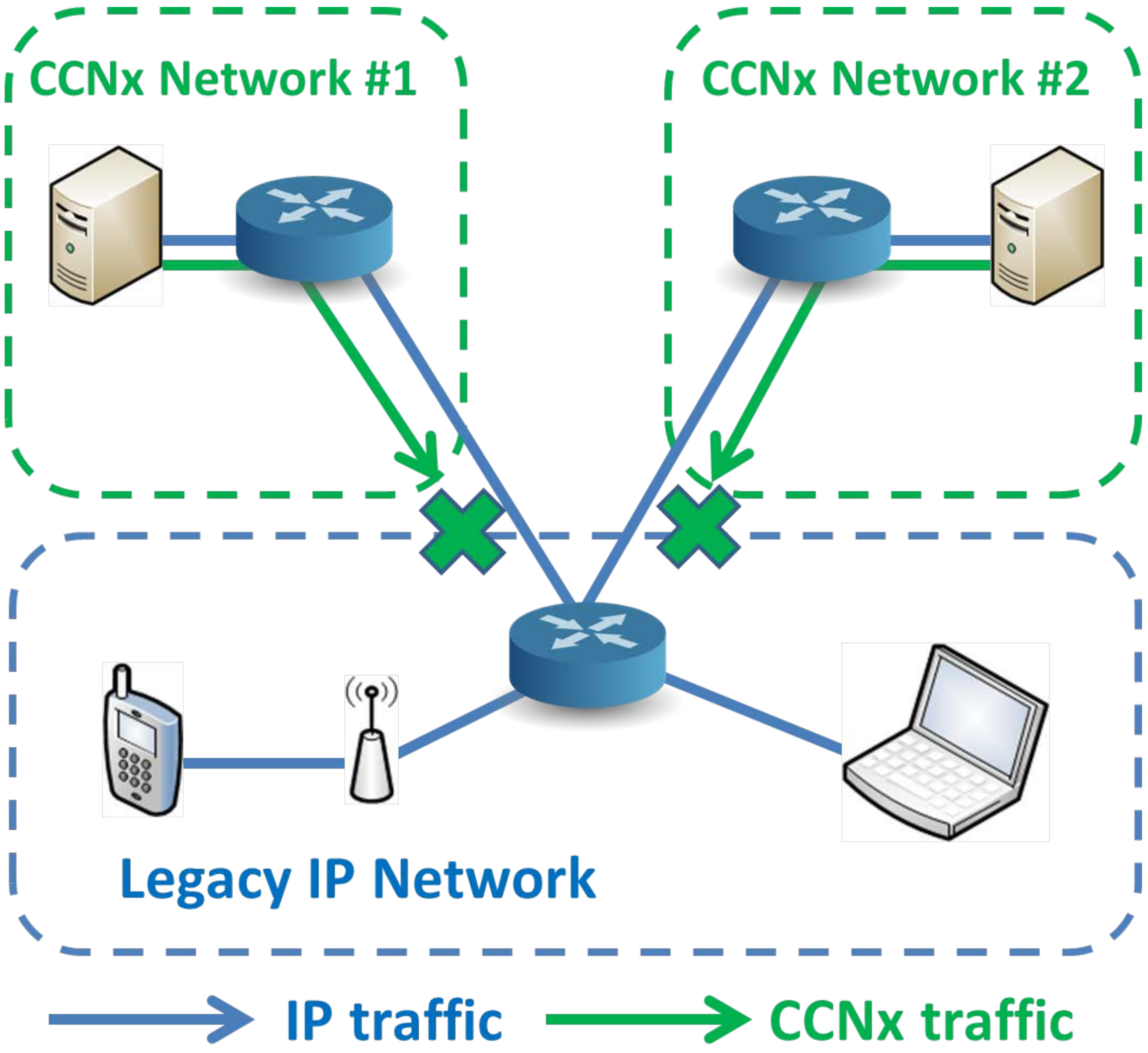


Figure 1. CCNx Routing. This is an example of the compatibility issues encountered with CCNx and the legacy IP network. While named content can circulate through "islands" of CCNx-enabled routers, the legacy network IP routers cannot understand the names or routing schemes. As a result, any users who are not physically connected to a CCNx router cannot access (or even see) the named data on the CCNx network. Additionally, CCNx cannot self-assemble into an internetwork across the legacy IP network—the BGP routers cannot recognize or advertise CCNx routing information. The only way to interconnect CCNx networks is to manually install a tunnel between the two CCNx networks.

receive NDN data. Likewise, NDN networks connected through non-NDN ISPs cannot self-assemble into an internetwork, because there is no mechanism for sending TLV datagrams via BGP. Thus, to develop a fully-integrated global network (anyone can reach anyone, as in the current design of the Internet), *all* routers connecting end users must be changed, and *all* autonomous systems (ASs) must change their routing methods. (A simple example of the connectivity problem is given at Figure 1.)

NDN also raises several issues regarding trust and security. As specified, it depends on “self-certifying” data, a cryptographic binding of the data name and data contents to a specific key. This binding can be verified (or not) by the content routers in the network and the end user, and the end user can use any method he wants to choose what keys to accept. The keys themselves also can be acquired by transmitting an interest. But there are significant concerns with this scheme. The community’s experience with the abuse of open mail relays [34, 35] suggests that network resources (such as cache space and interest tables) left unprotected will be exploited by the unscrupulous, and impersonation will probably occur on a large scale. This implies that verification of each data item will be done at many, if not all, routers in the data path. In turn, this means that the question of which keys to accept cannot be left to the end users—Alice may want to accept a key claiming to be from XYZ Corp. as legitimate, but if her ISP does not, then the data will never arrive. The same holds true for her ISP’s peers. Additionally, all of the problems with key management and revocation discussed in relation to DONA would apply here as well. (Trying to get a key by expressing an interest for it does not solve the security problem: one must decide if the data arriving in response is the correct key, or a false key; this requires the key to be certified with another key, which in turn must be certified, etc.) The only published approach which addresses this issue is [36], which requires the content chunk names to be uniform resource identifiers (URIs) that can be retrieved through the domain name system (DNS). While this does allow cryptographic certificates for each item to be distributed and verified, it begs the question of why one would not simply use HTTP for the data exchange instead of reinventing a new mechanism.

Lastly, even if data were to be successfully and effectively self-certifying, this does not resolve the question of verifying that the user requesting the data is legitimate. Current practice in CCNx is based on having an access control list (ACL) at each router[37-39], mapping the access permissions between every possible user and every possible data scope; this is likely to be impossible to implement at scale[39], even leaving aside the question of whether the routers can be trusted to correctly and securely implement the system. The amount and detail of data stored on the routers and switches in NDN also raises troubling questions about privacy compromises, either by snooping the router caches[40] or by traffic inspection[41].

2.3 Publish/Subscribe Internet Technology (PURSUIT)

The last major ICN prototype to be discussed is the Publish/Subscribe Internet Technology (PURSUIT) program [42], which is a further development of the Publish/Subscribe Internet Routing Paradigm (PSIRP) program [43]. This is an EU-funded effort which aims to create a new overarching architectural framework for networking. As may be obvious from the name, PURSUIT changes the network service model from request/response (like the current IP design) to publish/subscribe, and as a result develops an information centric design based on the name of the published item. The first prototype of this system is called Blackadder [44, 45].

As with other ICN designs, PURSUIT changes the networking primitive from a named end host to named data items. These are given statistically unique, flat (*i.e.*, not aggregatable) label identifiers by the publisher. The publisher then groups data items under one or more hierarchical "scopes" (collections of data items relevant to a particular mission, need or solution). Scopes themselves can be treated as data items that fall under other scopes, and are named with flat label identifiers as well. Scopes serve to publish the availability of logically-related data items, match requests with data, and then construct and execute a delivery graph from the source to the receiver. PURSUIT envisions that each scope will have a dissemination strategy, specifying rules appropriate for the data in question (*e.g.*, security policy, error control, fragmentation rules).

The PURSUIT architecture identifies three logically distinct functions that must be provided for each scope. First, the *rendezvous* function matches subscriber requests with published data, according to scope-specific policies. (Under the PURSUIT terminology, the data item name is called a "rendezvous identifier," specifically emphasizing that it is a functional tag rather than a file name, and that it is specific to a particular scope.) Next, the *topology management* function determines the optimal physical location of the data that can be used as a source and the best route for the data to follow during transmission. Lastly, the *forwarding* function executes the data flow.

In the Blackadder prototype, all three functions are implemented within the same device, called a publish-subscribe router. The rendezvous node maintains a list of published items and subscriber requests, matching the two as appropriate. The topology manager maintains a current network view and, when it receives a match from the rendezvous node, it calculates the publisher and optimal path for the request. Then it sends a notice to the publisher, containing the name of the requested item and a Bloom filter that specifies which interfaces each node in the path should use for forwarding the datagram. The forwarding function is simply comparing the given Bloom filter to the available outgoing interfaces at each node in the path; a match indicates that the node should forward the datagram via that interface. (This eliminates forwarding tables in the switches, which is seen by the Blackadder developers as a major issue in the feasibility of future networking designs.)

The PURSUIT project attempts to simultaneously solve multiple perceived difficulties, many of which are outside the scope of this research. Specifically for the purposes of this work, this system has one primary conceptual advantage: formally separating the functions of matching data requests and data availability, physically locating the data and calculating an optimal delivery graph, and the actual implementation of the delivery in the network forwarding fabric. These functions have very different requirements and operational demands, and logically separating them is the first step to optimizing them. (Our work primarily focuses on optimizing the delivery of the data, while simultaneously exposing the necessary metadata to higher level devices or processes to efficiently handle content discovery and policy application.)

Despite this major advance, we believe that the PURSUIT approach has significant drawbacks. First, the rendezvous function as described is likely to be very complicated to implement securely and effectively at scale. This is particularly true if scopes are allowed to fall under more than one parent scope—the question of which policies the sub-scope should inherit is likely to be very difficult to answer (and the network manager is probably not the best entity to decide in any event). Note that dissemination policies must be determined for every data item on the network, which implies that the number of decisions to be made will be exceptionally large in a production network. Second, the same concerns about reachability and routing stability mentioned in the case of NDN would also apply to PURSUIT as well, particularly in light of the fact that the Bloom filters used in the forwarding function have a non-zero error rate (*i.e.,* datagrams are sometimes forwarded through one or more incorrect interfaces in addition to the best path interface, which unpredictably creates loops in the layer 2 topology, resulting in uncontrolled packet storms). Third, it is unclear how this system could be extended to an *internetwork,* without tight coordination and exposure of a very large amount of customer data (such as scopes and dissemination policies).

Ultimately, these obstacles to a fully-functional network design were not able to be overcome by the research teams involved in the project. As of spring 2014, work on the PURSUIT project has come to an end.

2.4 New Developments in the Internet Protocol Network

The above discussion would seem to mark "case closed" on attempts to produce an information-centric network—despite multiple attempts, over many years by a number of researchers, no ICN design has emerged which was feasible to implement past a laboratory-sized network under a single, uniform network administration policy set. However, further developments in other areas of network in the last 12-24 months have opened intriguing new possibilities for fundamental network redesign. We believe that the development and implementation of advances in two other areas of networking (the design and roll-out of IPv6 and software defined networking) can facilitate a new and improved approach to ICN. Although it is beyond the scope of this document to fully describe the research behind these technologies, a brief description of the key points we will be highlighting in the development of the protocol and implementation is worth a short consideration.

2.4.1 Internet Protocol Version 6

The Internet Protocol version 4 has been in widespread use since 1981 [46], and it is fair to say that it is one of the most successful and transformative technology inventions ever made. Initially intended as simply one research design among many others, its combination of simplicity, flexibility and speed eventually triumphed over all other competitor systems. IPv4 is one of the most ubiquitous standards in the world.

However, by the late 1990s, it was clear that a new version would be required. The main issue was due to the exhaustion of the 32-bit address space used in IPv4, which had been made much worse by initial allocations of addresses that did not reflect the later globalization and spread of the Internet. Once it was clear that backwards compatibility would not be possible to achieve, the Internet Engineering Task Force (IETF) decided to take advantage of the opportunity for a fundamental redesign of the IP header. There were three major changes. First, the IP address length was increased to 128 bits, and the allocation of these addresses was simplified. Subnet masks in IPv6 could be any length, contrary to IPv4 practice, but the standards bodies also resolved that the minimum size to be issued to network administrators would be a 64-bit network address. (This would allow individual network administrators to subnet their networks as needed, without forcing them to have noncontiguous IP blocks in their network or to use network address translation to handle their addressing needs.) Second, they attempted to simplify the header by removing obsolete or unnecessary header fields. Under IPv4, there were many mandatory header fields which no longer reflected current network needs—for example, they were used for interoperability with non-IP networking standards which are no longer in use, or they were used for experimental protocols which were ultimately found to be unsuitable for general use. The IPv6 standard specified only a mandatory 40-byte header, containing only eight well-defined fields. This was designed to make the header very easy to parse in high-speed hardware. Third, to allow for future flexibility, the IETF added a mechanism for creating and appending new header fields if additional functionality were to be needed, as well as clarifying the overall policy for handling new or experimental headers in production networks.

Our research takes advantage of this redesign in two main ways: first, the larger address space, combined with a more systematic and flexible method of assigning network prefixes, means that we are able to add an additional level of syntactical meaning within the IPv6 address. We use this to make a switchable, routable identifier for information-centric datagrams. Second, making use of the ability to add a generic field for ancillary data in the header, we are able to make each datagram carry a significant amount of machine-readable, highly structured metadata that makes information centric networking possible within an RFC-compliant network. (Details of the header structure are given in the next chapter.)

Implementing IPv6 has been a very gradual process, with hardware and firmware generally being made IPv6 ready in the mid-2000s, but almost all production traffic remaining on IPv4 through the 2010s. New technologies implemented since the mid-2000s, including the 4th Generation LTE cellular network, were built to be native IPv6, and by some reports up to 25% of the cellular phone network traffic was native IPv6 by early 2014[47]. World IPv6 Day was celebrated in June 2012[48], when most major world Internet service providers permanently established IPv6 services for the public, and production traffic continues to grow (although from an initially low base). The result is that fully RFC-compliant IPv6 service is now widely available, and the remaining obstacles to its use are being addressed rapidly by commercial providers. Consequently, an ICN design based on IPv6 is now fully feasible to deploy.

2.4.2 Software-Defined Networking (SDN)

The additional features of the IPv6 header makes it technically possible to include appropriate labels for publisher and content, but without a reliable, practical means of making the detailed forwarding decisions needed to make use of this information, there would still be little point in proceeding. However, in recent years the development and implementation of SDN have radically extended our ability to provide fine-grained control of data flows in the network.

In SDN, the header fields of each packet are compared to a set of matching rules provided by a software controller (for example, source_address = “2001:1::1”, source_port = “80”, protocol = “tcp”). The highest priority match is selected, and the specified rule is executed (for example, forward the packet through interface 0). Thus, the forwarding plane (matching packets with a specific network action) is completely separate from the control plane (deciding how to categorize packets into forwarding classes). Additionally, SDN allows us to write almost arbitrarily complex programs as our network controller, including the ability to access databases or other sources of information as part of the decision process, and to modify and propagate the detailed rule sets required throughout the network automatically. Specifically for our purposes, SDN allows us to make packet handling decisions based, not simply on network prefix, port, and protocol, but also on the metadata we embed in the header.

The most common open source SDN framework is provided by the Open Flow specification[49], coordinated by the Open Networking Foundation[50] and supported by most industry participants. Almost all current networking hardware and virtualization software supports SDN, and open-source controllers are readily available. In particular for our project, OpenFlow version 1.3[51], finally approved in January 2014, called for full support for IPv6 header field matches, and in particular for the extension headers we will use for our metadata. The first virtual switches and SDN controllers to implement this new standard became available in spring 2014. Our research is not on SDN *per se*, however, the advances achieved by SDN solutions makes it possible to execute the detailed control requirements needed for our prototype to be successful. Prior to the introduction of this technology, it was simply impossible to design and implement a network which could be configured with such fine detail, and which could be modified on the fly during production to handle the large numbers of possible flows and policies to apply.

2.5 Summary

In summary, there have been many attempts at resolving a long-standing mismatch between what the Internet Protocol requires (purely host-to-host communication) and the way that many processes actually use the network (for interacting with named data items rather than specific end hosts). A number of research teams have approached the issue over the last 6-10 years, but no design has arisen which can actually be implemented in practice. Indeed, active research on these prototypes has largely stalled, due to unresolved questions about scaling, security, deployment, and other issues.

However, advances in other areas of networking have made an entirely new approach feasible. In the following two chapters, we will describe a way to extend the standard IPv6 header format to incorporate all of the necessary metadata for an information-centric network to be established on current hardware and software. We will begin with the layer three requirements needed to fully identify the datagram by producer and content. Then we will describe the layer three methodology for addressing the security and privacy of both sender and receiver, and several layer three specific technicalities. Next, we will move to layer four, and show how this proposed protocol meets the requirements for the layer four performance, with a particular comparison to the state-of-the-art transmission control protocol (TCP). Finally, we will show a simplified working prototype of the protocol, demonstrating that it can, in fact, be implemented on current (unmodified) hardware and software.

CHAPTER 3

LAYER THREE PROTOCOL DESIGN

3.1 Overall Design of an IPv6 Information-Centric Networking Protocol

The first step in designing a deployable, IPv6 based ICN is to define a datagram protocol to support the necessary data exchange and services required. We published the proposal for this protocol (described in this chapter) at the IEEE International Conference on Communications (ICC) as “CLIP: Content Labeling in IPv6, a Layer 3 Protocol for Information Centric Networking” [52]. As stated in the title, this work assumes that IPv6 remains the “narrow waist” of the network, and that ICN traffic must coexist with legacy point-to-point traffic for the indefinite future.

There are three main areas we have addressed in developing the protocols for converting the host-based IP framework into an ICN framework. First, there must be a globally unique, structured way of identifying individual data items. This protocol describes how to develop such a name and embed it into an RFC-compliant IPv6 datagram header, allowing unmodified IPv6 nodes to route datagrams by content name (including both publisher and item name) rather than host name. Second, the security and privacy of individual data items must be addressed. The data items must be combined with all necessary metadata to allow any node in the network to verify the integrity of the data and bind the entire item to a specific publisher. Additionally, there must be a provision for concealing both the content of the data item and, if necessary, concealing the publisher and consumer of the data from intermediate nodes. Lastly, the ICN routing protocol needs to demonstrate that it is safe (guaranteed to converge), autonomous (does not require coordination or exposure of intra-network data), and expressive (allows for diverse policies between autonomous systems). Additionally, to be realistically deployable, the system must be fully compatible with the current host-oriented network routing paradigm, allowing both information-centric and legacy traffic to coexist on the same network indefinitely without interfering with each other in any way.

3.2 Datagram header design

The first portion of the protocol design is the IP header content. The first step in the process was to create a globally-unique name for each content item. Similar to the existing ICN designs, we adopted a naming convention of associating a globally-unique publisher label (PL) with a unique content item label (CL). We then integrated this naming convention into the IPv6 address space. Our approach was to create a large subnet for all ICN traffic, then create individual subnets for each content publisher, and lastly to append the content label to the interface ID and destination options header.

3.2.1 Publisher Labels

We defined three types of PLs: global PLs, which are unique to a specific organization or user; local PLs, which are non-routable and managed by the local network provider; and anonymous PLs, which are designed for ephemeral use.

3.2.1.1 Global Publisher Labels

The global publisher label (PL) will have to be assigned by a universally-recognized organization, similar to the ways that IP addresses, MAC addresses, and AS numbers are issued today. While this is a less-than-ideal requirement, in practice there is no other way to ensure that the publisher label is actually unique. Indeed, as the PL will become part of the globally-routed IP address, it may make most sense to have these numbers issued in exactly the same way that IP addresses are allocated today. Additionally, this partially resolves the problem of associating a network-level item (the PL) with a real-world identity of a person or organization. In this work, we assume that the length of PL is 64 bits long, and begins with 0xC. The length is chosen because the smallest allocation of IPv6 address space to be assigned is the /64 subnet, as described in RFC 6177 [53] and RFC 4291[54]. Thus, it is certain that there will be at least 64 bits of address space available for assignment on any RFC-compliant network. The prefix 0xC is used for two reasons. First, a unique subnet prefix for all ICN content (from all publishers) makes subsequent special handling by network nodes much easier to implement in hardware. A unique subnet prefix allows aggregation of the ICN address space and separates it logically from the legacy point-to-point IP space, easing the administrative overhead of policy enforcement. Second, this prefix is not currently specified for use in any other network addressing schemes, including stateless autoconfiguration [55] and IPv4 embedded/compatible/mapped addresses [56]. This gives a global space of 2^{60} unique names for publishers, while still leaving a very large number of subnets available in each network address for legacy host names.

3.2.1.2 Local Publisher Labels

On the link local address FE80::/10, the publisher label may be locally assigned by the network provider. This will allow for locally produced and consumed data to be locally administered as well. It is expected that most users, and possibly even individual processes on local hosts, will receive ID numbers of this type for intra-network use. The network provider may make arrangements to republish local data under a commercial or organizational publisher label for any data needing global availability.

Readers may question why we are not using the unique local addresses (FC00::/7) defined in RFC 4193 [57] for this purpose. There are two main reasons. First, the RFC specifically requires the remaining 40 bits of the network address to be generated randomly and further specifies that these addresses must not be aggregated by routers. In contrast, under this protocol, network providers would be expected to structure and aggregate the network addressing scheme to simplify administration and logical data flow. Second, the early experience with "site local" addressing showed that addresses can "leak" via the application layer, particularly with mobile or multi-homed hosts [58]. This can lead to erroneously addressed packets flooding through the network. (For example, a user would download his e-mail at work using one addressing scheme, but attempting to use the same address from home would cause an IP number conflict on his ISP's network.) By using the link-local addresses, this problem is largely avoided: the network does not route FE80::/10 addresses, so an IP address conflict could only occur if another host on the same layer-2 switched link had both the same global ID, local ID and local publisher ID (118 address bits total).

3.2.1.3 Anonymous or Ephemeral Publisher Labels

Lastly, to address the need for no-cost and anonymous publisher labels, The PL numbers beginning with 0xCFF are reserved as a type of "unlicensed spectrum" for anonymous or ephemeral use. We foresee two main uses for addresses in this class. First, we expect that there will be a need for ephemeral "rendezvous" points, particularly for one-time data transfers between individuals. Second, these addresses can be used to conceal the true publisher providing data to a host, as discussed under security and privacy concerns (see below under “Security and Privacy”).

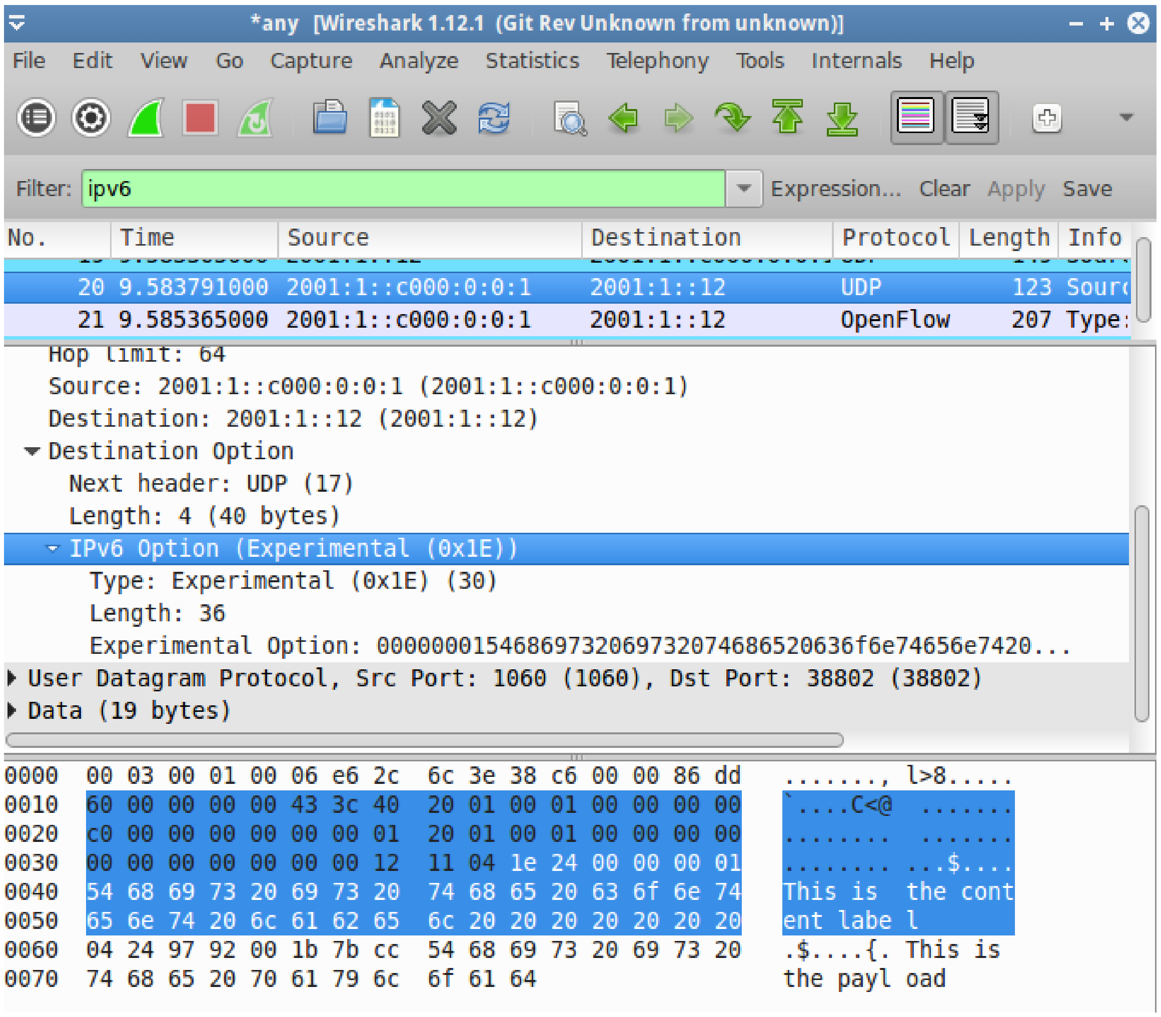
3.2.2 Content Labels (CLs)

The last component needed for this system is a unique content item label. The label begins with a two-byte field for timestamp or version number, assigned by the publisher. Next, the publisher must assign a unique content item label. The question of content naming is an area of dispute among the designers of different ICN systems, with some systems choosing a human-readable, hierarchical naming convention (similar to the URLs used in HTTP), while others use a cryptographically-derived self-certifying name.

Octet	**0**	**1**	**2**	**3**
0	*Version*	*Traffic class*	*Flow label*	
4	*Payload length*		*Next header*	*Hop limit*
8	Source network			
12				
16	Source interface or Publisher Label[1]			
20				
24	Destination network			
28				
32	Destination interface or Publisher Label[1]			
36				
varies	*Hop by hop, Fragment, AH, and/or ESP header[2] (as needed)*			
varies	Next header	CL length	Timestamp or version number	
	Content label (CL)			
varies	*Other headers (ESP[2], fragmentation, mobile IP, etc.) if needed*			
	PAYLOAD			

1. When a host is requesting a content item, the destination interface will be the Publisher Label (PL); when a content server is replying with a content item, the source interface will be the PL
2. The ESP header may appear before or after the Content Label (CL).

Figure 2 Header format


*any [Wireshark 1.12.1 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: ipv6
Expression... Clear Apply Save

```
No.  Time         Source              Destination   Protocol  Length Info
  20 9.583791000  2001:1::c000:0:0:1  2001:1::12    UDP          123 Sourc
  21 9.585365000  2001:1::c000:0:0:1  2001:1::12    OpenFlow     207 Type:
```

```
  Hop limit: 64
  Source: 2001:1::c000:0:0:1 (2001:1::c000:0:0:1)
  Destination: 2001:1::12 (2001:1::12)
▾ Destination Option
    Next header: UDP (17)
    Length: 4 (40 bytes)
  ▾ IPv6 Option (Experimental (0x1E))
      Type: Experimental (0x1E) (30)
      Length: 36
      Experimental Option: 00000001546869732069732074686520636f6e74656e7420...
▸ User Datagram Protocol, Src Port: 1060 (1060), Dst Port: 38802 (38802)
▸ Data (19 bytes)
```

```
0000  00 03 00 01 00 06 e6 2c  6c 3e 38 c6 00 00 86 dd   ......., l>8.....
0010  60 00 00 00 00 43 3c 40  20 01 00 01 00 00 00 00   `....C<@ ........
0020  c0 00 00 00 00 00 00 01  20 01 00 01 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 12  11 04 1e 24 00 00 00 01   ........ ...$....
0040  54 68 69 73 20 69 73 20  74 68 65 20 63 6f 6e 74   This is  the cont
0050  65 6e 74 20 6c 61 62 65  6c 20 20 20 20 20 20 20   ent labe l
0060  04 24 97 92 00 1b 7b cc  54 68 69 73 20 69 73 20   .$....{. This is
0070  74 68 65 20 70 61 79 6c  6f 61 64                  the payl oad
```

Figure 3. Screen capture of a CLIP packet in Wireshark

This protocol functions equally well with either convention. (But see the discussion of privacy and security below, where we describe why we believe that cryptographic-based names are not desirable in practice, and how to provide the necessary binding using IPsec protocols.)

The CL is placed into the destinations options header field in the IPv6 datagram as shown in Figures 2 and 3. This has three main advantages. First, the content label can be very large—up to the size of the datagram (a minimum of 1280 bytes on any network and a maximum of 2^{32}-1 bytes for a jumbogram). Second, this allows the label length to be variable—the only system requirement is for the header to be a multiple of eight bytes long. We expect that different types of content will have different optimal sizes for individual data items, and this flexibility allows content owners to make an appropriate decision for their content and operational needs. Third, the IPv6 specification states that the destinations options header is not read or acted on during transit. This means that the legacy network will not need any additional equipment or policy to handle ICN traffic. A deployment of the ICN system does not require any arrangements with service providers beyond receiving IPv6 service. However, if a network provider chose to implement an information-centric proxy or cache, the content information would be easily available in a standard structure. That is, they could implement this proxy or cache without the need for deep packet inspection, statistical tests on variable length windows, etc.—and in fact, without standing up a full ICN network.

3.3 Security and Privacy

The next issues to be addressed are security and privacy. There are three main security considerations that any deployable ICN system must address: key management; the secure binding of real-world identity of the publisher, publisher label, content label, and the content itself; and preserving the privacy of end users. As noted above, we assume that the binding between real-world identity and PL is done when the PL is issued. An addressing and header protocol such as this one also cannot address the question of key distribution. We assume that this is handled by another method, such as DNS IPSECKEY records [59] or Kerberos [60], and that a security association (SA) has been created.

Once a key has been distributed, the binding of the PL, CL, and data can proceed. IPv6 provides a straightforward solution with the Authentication Header (AH) feature of the IPsec suite of protocols [61]. It binds all non-mutable elements of the datagram (*i.e.*, everything except the traffic class, flow label, and hop limit) with a cryptographic signature. The signature may be based on either symmetric or public keys, and the protocol allows for new cryptographic algorithms to be implemented in the future. This flexibility is critical for a long-term production system, to allow for security and performance upgrades where and when needed. Note, however, that there are some important conceptual differences between the legacy point-to-point security association and an ICN security association. Most importantly, the use of challenge-and-response methods (like encrypting nonces) and mutually-constructed secrets (as in the Diffie-Helman algorithm) is not generally possible, as there is no explicit connection between the publisher and receiver. Additionally, in the context of this protocol, a "session" is not an end-to-end connection between two hosts, but rather a period of authorized access to certain CLs.

It is also important to discuss the question of using cryptographically-generated, or *self-certifying*, names as a security feature to prevent denial of service attacks. As described in [62], under this concept publishers are identified by a fixed-length hash (P) of their public key (assumed to be bound to their real-world identity by a DNS-type lookup system). The data item is named by concatenating P with a label L, which is in turn generated by cryptographically hashing the tuple <data, public key, L, metadata, signature>. This tuple is the data item provided to a requestor. This makes it effectively impossible to impersonate a publisher, as any node can hash the provided public key and verify both P and L. This is extremely risky in a production network, however. Key compromises are an unfortunate reality, and when (not if) they occur, every data item that the victim has ever created must be renamed—and the new names will have no relation to the compromised names. Such an event would be incredibly traumatic for the victim, particularly if the compromised network cannot be shut down for overhaul. Even in the best case, cryptographic algorithms have a finite lifespan, and production environments typically require a transition period measured in years; during that time, both algorithms must be supported. If names are generated by specific hash algorithms, an overlap would seem to require duplicate copies of every data element (one using the old hash, and one using the new) or some method of name translation for every data item. In light of these difficulties, we do not recommend using the naming scheme to implement security features.

The next issue to address is the issue of privacy for the end user. ICN systems route and cache datagrams based on the names of the publisher and content item; as a result, anyone can determine what data the end user is accessing. This could have serious negative consequences for the end user. This new protocol, combined with the IPsec suite of protocols, can effectively mitigate some of these issues. Using the AH header alone, as described above, does not provide privacy services and will leave both the identity of the publisher and the content of the data item easily available to any observer. This has the same privacy issues currently experienced with unencrypted internet traffic. The remedy for this is to apply the Encapsulating Security Payload (ESP) protocol [63] in one of three configurations:

- ESP in transport mode to encrypt the payload only. In transport mode, only data that comes after the ESP header is encrypted, so in this case, the ESP header is placed after the CL and before any transport-layer headers (including port and protocol numbers). The AH must also be used here to bind the PL and CL to the payload. Observers will know the PL, CL, and the approximate size of the payload, but not the actual content. (As an aside, publishers are free to generate CLs for their data items which might appear to be gibberish to an outsider.)
- ESP in transport mode, covering the CL and payload. In this configuration, the ESP header is placed in front of the CL, and ESP will conceal all information about the CL (even its length). An observer will know that the end host received an (approximate) amount data from a particular publisher, but no other details are visible.
- Lastly, in cases where the user wishes to conceal the PL, CL and payload contents, the publisher would use ESP in tunnel mode. In this mode, the entire original packet is encrypted, and a new header generated. This new header would have either a republisher's PL or an ephemeral PL (as described above), and an ephemeral CL. This will conceal the publisher, content name, and payload. This provides the full spectrum of layer-3 security and privacy features available to legacy network users.

3.4 Layer Two and Three: Routing and Switching Considerations

Lastly, we must address the question of how this proposed protocol will perform at layers two and three in the current network architecture.

3.4.1 Switching Performance

As we noted in chapter one, adding metadata into the header is pointless unless we are able to use that metadata in making datagram forwarding decisions at line speed. In the case of our design, the requirement is to be able to read the source/destination IP address and destinations options field, and then based on that information assign a datagram to a forwarding equivalence class. This decision must be made and implemented fast enough to prevent a chokepoint at the node. This is where software defined networking comes to our aid: switches which implement the OpenFlow 1.3 standard (or later) are able to perform these matches, if given a ruleset by the SDN controller. We will show in chapter five that we were able to use the Ryu SDN controller and the Open vSwitch virtual switch to implement any forwarding rule or traffic classification defined by the business logic of our controller.

Note that, in an SDN network, there is no difference between the forwarding plane of a layer two device and a layer three device. In traditional networks, there was a major difference between the forwarding ability of switches and routers—switches could only make forwarding decisions based on L2-headers (such as Ethernet MACs), but were extremely fast; whereas routers could parse and rewrite most or all of the header fields, but they implemented much of their functionality in software and were as a result much slower than the hardware-only forwarding in the switches. In SDN, this distinction disappears. The SDN controller reduces the business logic provided by the programmer to a set of matching rules to be implemented by the forwarding plane—there is no difference between a “switch” and a “router” in terms of forwarding speed or functionality. Any difference between a “layer two” device and a “layer three” device (*e.g.*, whether the hop limit is decremented at the node) is solely due to the business logic provided by the controller. Thus, we conclude that, for network nodes that implement OpenFlow 1.3 or higher, CLIP datagrams can be forwarded at line speed. (We will demonstrate this in chapter five.)

3.4.2 Routing Performance and Stability

Nonetheless, there still remains the matter of network routing performance and stability, even if the function is now done on an SDN controller rather than semi-autonomous, independent routers in the network. An absolute prerequisite for any networking protocol is certainty regarding routing performance and stability, especially under high load and under realistic conditions (including the presence of malicious users and misconfigurations). As noted above, successful internetworking requires that the routing algorithms are safe, autonomous, and expressive; yet such a guarantee is elusive for current ICN prototypes. As noted in chapter 2, each ICN prototype handles routing in a different way. From a theoretical standpoint, conditions for global reachability and stability in name-based routing have not been determined. This means that there are no guarantees that any solution is possible, let alone feasible, at scale and with realistic considerations of human and machine failure. From a practical standpoint, little actual data is available on the performance and safety of these designs, particularly concerning the effects of divergent policies on internetwork routing. In contrast, by using IP natively for naming and routing, we are able to apply the extensive theoretical and practical results of working with routing protocols under a very wide variety of conditions.

In the case of interior gateway protocols, used for routing traffic within an autonomous system, we have many choices with proven track-records and extensive configuration guidelines, such as OSPF and IS-IS. All modern routing protocols have been modified to handle RFC-compliant datagrams and hosts, so we expect no divergence of behavior based on the data tags. Additionally, we can be certain that, if the network's routing protocols are successfully routing host-centric traffic, then the information-centric traffic can also be handled by the same mechanism (*i.e.* the network administrator will not be forced to deploy two separate routing protocols, and there is no possibility of routing paradox based on the mismatch between two different routing protocols.) As we will demonstrate in chapter five, our initial prototype behaved identically to host-centric datagrams on our test network.

Exterior routing protocols are more complicated, as no one has a complete picture of the entire network and internal policies will vary quite extensively. As noted above, the competitor ICN designs had essentially no way to address the issue of information-based routing at this level (at best, they permitted tunneling between isolated ICN autonomous systems). In contrast, our system can use exterior BGP to route between and across autonomous systems. BGP has a long history of real-world deployment and the details of configuring and troubleshooting it are well known. ICN-specific routing policy issues (if they arise) could be addressed using the BGP Extended Communities Attribute [64, 65]; note also that there is extensive ongoing research in methods of passing routing and link state information from both interior and exterior gateway protocols up to the application layer that may be useful for ICN-enabled devices (*e.g.* for content discovery). An example of this system of operation is given at Figure 3.

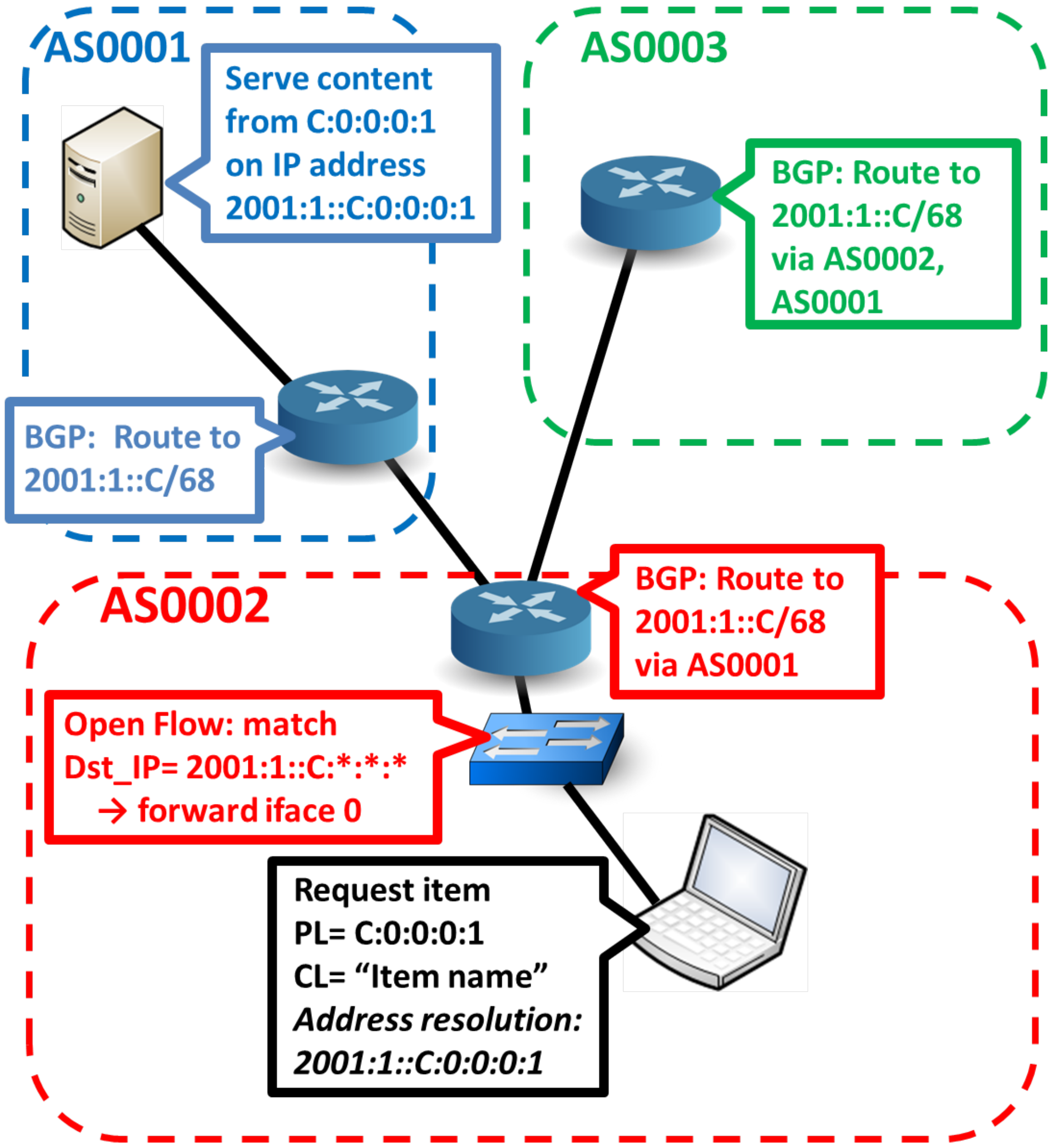

AS0001
Serve content from C:0:0:0:1 on IP address 2001:1::C:0:0:0:1
AS0003
BGP: Route to 2001:1::C/68 via AS0002, AS0001
BGP: Route to 2001:1::C/68
AS0002
BGP: Route to 2001:1::C/68 via AS0001
Open Flow: match Dst_IP= 2001:1::C:*:*:* → forward iface 0
Request item PL= C:0:0:0:1 CL= "Item name" Address resolution: 2001:1::C:0:0:0:1

Figure 4. Interior and exterior routing of CLIP datagrams. In this example, AS0001 is serving content from publisher C000:0:0:0:1. The content server listens on IP address 2001:1::C000:0:0:0:1 for CLIP datagrams. AS0001 advertises a route to subnet 2001:1::C/68 by BGP. AS0002 hears the BGP announcement and advertises the route to its BGP neighbors. It also runs its own internal routing protocol (such as OSPF) and determines the routes within its own network for this subnet, and it installs the appropriate forwarding rule in its switches. Then, any host on its network can reach the content provided by AS0001—even if AS0002 is not content-aware, the networks automatically form an internetwork for CLIP data.

Since total assurance of routing stability is an absolute prerequisite for a production network, we believe that this alone makes our proposed ICN protocol superior to other ICN designs. Note also that, by using RFC-compliant IP addressing, we also gain assurance that the datagrams will not only be routable, but also switchable with unmodified commercial hardware, including optical systems and radio links, and also virtualized networks. The only requirement is that the network infrastructure meets IETF standards.

3.4.3 Backwards compatibility

The question of backwards compatibility of the ICN prototypes to date has only been addressed in the sense of ensuring that a given ICN network can traverse an IP-based network. That is, the ICN is either an overlay on top of the IP transport network, or it tunnels packets from one ICN "island" to another across an IP network. (In contrast, we use IPv6 compliant datagrams, and hence our data is fully routable and switchable by RFC-compliant nodes. That is, if a link fails or a better route opens, an unmodified network will automatically make the appropriate decision for forwarding our data.) Backwards compatibility in the full sense, that is, whether hosts on an unmodified legacy network can access ICN content and hosts on an ICN network can access data on the legacy network, is not possible when the ICN network does not use the IP waist; non-ICN routers will not recognize or route ICN data. By using the native IPv6 packet header design, this protocol resolves this issue of creating a delivery path between any IPv6-enabled host and an ICN-enabled device containing a specific named data item. Likewise, it ensures that the legacy traffic can coexist with ICN traffic on the network. (We demonstrate this in chapter five, where we show that CLIP datagrams and TCP datagrams do not interfere with each other.) However, content discovery, whereby a host becomes aware of the presence of a desired data item, is left to a higher-level protocol.

3.4.4 Object Sizing

Another area of strong disagreement between ICN designs is the size of the individual objects. The principal concern is that, by naming data objects instead of end hosts, the number of named items to be located by the network expands by many orders of magnitude. The smaller the size of the named data item, the worse the name-space scaling problem becomes. Recent results indicate that it is physically impossible to route ICN-named packets at line speed and Internet scale using current memory technology [3, 66].

In one sense, our new protocol bypasses the problem: it is pure IPv6, and so by definition is routed at line speed (*i.e.*, as fast as legacy traffic is now). The content-specific portion of the name is placed in the destinations options header and can be ignored in transit. Our definition of a data item is a datagram, but in IPv6 the size may notionally be anywhere from one byte to 4 GB. While the extremes are unlikely to have any real-world application, this does demonstrate that content owners will have wide latitude to size their objects appropriately for their operational needs under this naming convention. For example, one would expect that bulk transfer objects could be quite large, while streaming media content would likely be broken into very small, sequentially accessed objects. Likewise, we expect publishers to choose CLs for their data which group similar content together, to improve the storage and retrieval performance of their data.

In another sense, the expansion of the potential namespace is a significant question when considering content discovery—that is, when determining the name of the item and candidate physical locations. This is an ongoing research question in the current (host-based) IP network as well; but also, it is clear that this cannot be solved by a layer-three protocol. Additionally, we should make a distinction between a networking protocol (such as this one) and a storage protocol. We would expect that content-labeled datagrams could persist in caches or proxies for a period of time, but probably only for a period of minutes to hours (depending on the type of information they contained) before expiring. We expect that this, combined with aggregation of data by publisher name, will limit the growth in name lookups.

3.5 Summary

This chapter described the layer three design of an information-centric datagram. We used the features included in the IPv6 standards to embed the necessary metadata for creating a globally-unique identification of each data item created by a publisher. We described how to use currently-available security protocols to reliably bind the publisher, data name, and payload to a known cryptographic key, thereby ensuring that data may not be modified en route, spoofed, or falsified. We also demonstrated a methodology for concealing either the content of the data, or even the true identities of the publisher and the data item name, if needed for security. We showed that these security mechanisms are in use today in host-oriented traffic, and so can be implemented with unmodified hardware and software. We also discussed how an unmodified, RFC-compliant IPv6 network is able to route and switch data based on the identity of the data rather than the identity of the end host. As a result, the ICN network can co-exist with the legacy point-to-point network seamlessly, allowing an incremental roll-out without further coordination. Additionally, our protocol inherits the benefits of the proven stability and security of the IPv6 routing framework.

With the layer two and three issues clarified, we can now turn our attention to the layer four issues which must be addressed for a successful implementation of production traffic. No previous prototype of an ICN has been deployed on a medium to large scale, and so we must change our frame of reference from a comparison of our protocol to one of the ICN designs, to a comparison to a fully functional design. Specifically, we will show how our protocol addresses the functionality provided by transmission control protocol (TCP), which is the most commonly-used layer four communications protocol in the Internet today.

CHAPTER 4

LAYER FOUR PROTOCOL DESIGN

In the last chapter, we discussed the layer three design of our protocol. We showed how an information-centric datagram is produced, transmitted, received and parsed. We also showed existing layer three protocols implemented in every standard networking stack can be used to mitigate or eliminate the fundamental security and privacy concerns that are not addressed by competitor ICN designs. The CLIP design is inherently backwards compatible, since it conforms to the current networking standards in use and requires no network configuration changes to function.

Although this is already much farther forward than other ICN designs, there are still further issues to consider for a fully deployable network protocol. If we are to propose a protocol for general purpose use, we must assume that the current software infrastructure must be compatible with our new methodology—much like we focused last chapter on demonstrating how to function within the currently installed network and physical infrastructure, so we must address how our proposed protocol will meet the existing interfaces with applications. Today, most higher-level applications (such as data base applications or web servers) use the transmission control protocol (TCP) as an abstraction from the actual details of data transfer. This provides four major functions, over and above simple data transfer and multiplexing given by UDP: flow control, retransmission of lost packets, in-order delivery of packets, and congestion control. In this chapter, we show that our protocol can address three of the four straightforwardly, and in fact in a much simpler and cleaner way than TCP. In the case of the last (congestion control), we can currently imitate the TCP methodology, but there is good reason to think that further research is warranted and may be able to significantly improve system performance in a fully information-centric network environment.

4.1 Flow control

Flow control is the methodology used to prevent inbound data flows from exceeding the requesting host's ability to buffer and process data. TCP is a stream-oriented protocol, which means that the length of the data stream is formally infinite—that is, the receiver does not know (and cannot specify) the length of the inbound data stream, nor does the receiver know how many packets are in the data stream or what payload size each packet has. Nonetheless, in reality buffer space is always finite, and so the transmission protocol must provide some method for the receiver to communicate with the sender how much receive buffer space is available at any given moment. Additionally, any layer four protocol must cope with the situation that network speed and processing speed may be different between sender and receiver, and that both may change without notice during the course of transmission.

At the TCP layer, flow control is done with sending a receive window size as additional header data during transmission. The receiver sends a sliding window in its outbound TCP header, indicating the number of bytes it is able to accept; the sender then restricts itself to allowing only the specified amount of data to be in flight (that is, transmitted by the sender but still unacknowledged by the receiver). This means that the payload size of each datagram may vary, depending on the last acknowledgement and window size received by the sender. In practice, this system can break down in two main ways. First, if a datagram containing the window size update and acknowledgement is lost in transit, the system can end up in *deadlock*, where the receiver has requested additional data but the sender is unaware of this fact. As a result, TCP must also implement a *persist timer*, which triggers when no window update or acknowledgement has been received within a specified time. The receiver then resends an acknowledgement packet and receive window, in an attempt to resynchronize the window size with the sender. A second problem is known as *silly window syndrome*, where the receiver advertises a very small receive window by mistake, leading the sender to transmit datagrams with only a few bytes of data in the payload. This consumes enormous amounts of bandwidth and processor time, due to the transmission and processing overhead, and can lead to severe network degradation.

By contrast, the CLIP protocol simply side-steps most of these problems. Since the transmission is data-oriented, not connection-oriented, the receiver has total control of the inbound data size: it only requests those data items which it is prepared to receive (*i.e.*, has reserved a sufficient buffer for the data item prior to requesting it). Data item size is limited by the MTU of the link in all cases, and it may be known in more detail by the application (*e.g.*, the data items are fixed-length segments, representing a fixed-time segment of an audio or video stream, or the application has received a manifest of data items to request which include their maximum lengths). The application should simply not request data items which it is not currently prepared to buffer—there is no operating-system or network control needed, as the application has all the information it needs to make this decision.

Note also that, even though TCP may formally consider the data stream to be infinite, higher-level protocols typically structure their data transmissions to include the size of the data item, usually in the first few bytes of the transmission. (For example, HTTP requests typically include a `Content-Length` field in the header of the item, to allow the receiving process to allocate sufficient memory for the remainder of the transmission.) This is another example of the mismatch between what is specified by the layer three/four architecture and what is actually needed by the application layer. In the case of CLIP, the datagram length is contained in the first 40 bytes of the IP header, and is always less than or equal to the MTU size of the transmission link (which can be queried by the application prior to requesting the data item). Thus, we expect that applications will in no case be worse off by using CLIP as a transport, and if they are content-aware when requesting a data item, they may be better off.

4.2 Retransmission of lost packets

The Internet is designed to be a best-effort delivery system; it explicitly refuses to provide any retransmission services in the event of a packet loss. This means that the end hosts must have some mechanism for detecting and responding to lost packets. While some use-cases simply abandon the lost data (*e.g.,* streaming media), in many cases applications require some mechanism for requesting retransmission of lost packets. In TCP, this is done through the use of an acknowledgement of the cumulative number of (payload) bytes received. The receiver includes this value in the header of its next transmission to the sender. The sender is required to assume that all data was lost and must be retransmitted, except for data which has been explicitly acknowledged by the receiver. But the sender also has to maintain a running estimate of the round trip time between the sender and receiver, to allow at least twice the estimated round trip time to elapse before attempting to retransmit. Although this does work, in the sense that all lost data will be retransmitted automatically, it often leads to large amounts of data being retransmitted unnecessarily (since all data transmitted after a single lost packet will be assumed lost as well). TCP was further modified with the ability to selectively acknowledge some data that had been received after a missing packet[67]. More fixes were then needed to prevent instability in the presence of duplicate acknowledgements[68], and several modifications to the method for estimating the correct timeout value[69, 70], especially in the presence of time-varying networks (such as mobile networks). The end result is a system that is functional, but quite complicated to successfully implement in a real network and which requires estimation and assumptions about behavior for both the sender and the receiver.

By contrast, our protocol is much simpler. The sender has no responsibilities for detecting or correcting packet losses whatsoever. Its responsibilities end when the datagram is transmitted; any “retransmission request” is considered to be a new request. All responsibility is placed on the requester, to determine whether the data is still needed after the expiration of a timer, and if so, to formulate a new request. (We will discuss the development of this timer under the heading of congestion control, below.) By moving the responsibility for retransmission from the sender to the receiver, our system is much simpler and requires fewer assumptions about remote hosts; it also removes some failure modes and attack surfaces.

4.3 Reordering of packets

Production networks can also sometimes deliver packets out of order, or duplicate packets and deliver more than one copy of the same packet to the receiver. In stream-oriented protocols, such as TCP, these out of order packets must be detected and resolved for the original data stream to be reconstructed. TCP uses a sequence number to indicate the relative position of the current payload in the stream of data to be reconstructed for the receiver. There are two issues with this methodology. First, the operating system does not release any data to the application which requested it until the reordering is complete. If this process requires a retransmission, delays can be quite large (especially relative to the processor speed), and while this is happening, the application cannot proceed with processing the partial data which is on hand. Worse, the operating system's receive buffer can (and often does) saturate with data, meaning that flow control mechanism outlined above shuts off the requests for more data until the missing data is located and the queued data is passed out of the operating system buffer up to the application. This causes further delays once the request process restarts (again, network delays are typically very large compared to processor speed, and TCP will take several back-and-forth window resizing messages before stabilizing at the best available bandwidth utilization rate). Second, and perhaps more pernicious, is the issue of security. It is a well-known problem that if attackers can spoof the sequence number mechanism, they can inject false data into the data stream, force the closure of the socket, or other attacks. By randomizing the initial sequence number chosen, the TCP client can make it more difficult for off-path attackers to implement these attacks[71, 72], but this remains an exploitable issue[73, 74]. Fundamentally, the problem is that there is no secure binding between the source, the sequence number and the payload. This means that there is no method, either for the end host or any network node in between the hosts, to positively recognize that a spoofed packet has been sent.

Our protocol bypasses these problems as well. Each data item is treated as its own entity—there is no sequencing needed. Once a datagram is processed off the transport medium, its contents can be passed to the application which requested it. (Any reassembly of data items into a stream or larger item must be done by the application, which will have full information on the context and meaning of the data. The decision to delay waiting for more data is left to the application.) The operating system does not need to store data waiting on another packet to complete, and the application has immediate access to all data that has been received. Likewise, since there is no concept of a stream of data extending over multiple datagrams, there is no session which can be hijacked, terminated, or otherwise disrupted. And as described in the previous chapter, the protocol is designed to securely bind the metadata in the header with the payload, in a way that can be verified both by the receiver and optionally by any other node in the network.

4.4 Congestion control

Network congestion is defined as a network link or node facing a traffic load so high that its quality of service declines, usually shown by rising delay, increasing data losses, and possibly the inability to establish new connections. Transient congestion is a routine issue in real networks. Far more serious is the phenomenon of *congestion collapse,* a situation in packet-switched networks where the network's effective throughput drops by several orders of magnitude and persistently remains in the low-throughput, high-loss state. The usual culprit for congestion collapse is an overly aggressive retransmission scheme for dropped packets. Once a loss threshold is met, repetitive attempts at resending unacknowledged packets can overwhelm the available bandwidth of a link or router, resulting in a near-total loss of effective capacity. It is a particularly difficult problem, because in most networks the control signals travel in band (*i.e.*, over the same congested links or nodes), and so it can become almost impossible to apply a remedy to the problem once it begins. Congestion collapse has been observed multiple times in production networks, and in particular, the collapse events on the NSF-Net between 1984 and 1987 were a primary reason for the development and publication of the first effective TCP congestion control mechanisms[75] which were included as part of the 4.3 BSD-Unix ("Tahoe") distribution.

Congestion events must be addressed through multiple means throughout the network. Specifically for our purposes, *congestion control* refers to methods to prevent excessive loads from entering the network. *Congestion avoidance* refers to methods used within the network to avoid or cope with congestion, including mechanisms such as network scheduling, random early detection (RED), active queue management, load balancing, and various congestion notification schemes. Any feasible layer four protocol must implement some form of highly-reliable congestion control, *i.e.* must have a mechanism for automatically reducing its offered load to a level at or below the rate capable of being processed by the network. Although many schemes for sending explicit congestion warnings or signals to data producers have been proposed over the years, all have suffered from the fundamental problem that congested signaling is conducted in band, and so a congested network will be forced to drop the congestion signaling packets along with the user data. In other words, the network is only able to reliably transmit congestion signals when there is no congestion! In fact, the only completely certain means of detecting congestion events is the loss of quality of service at the end points, *i.e.*

loss of packets in transit, out of order delivery events, or an increase in latency. This means that the end points must monitor for lost, delayed, or out of order packets; if this behavior exceeds a threshold, the end host infers network congestion and must sharply reduce its offered load to the network. Network traffic management and control becomes very difficult unless a very large majority (if not all) end nodes on the network use an equivalent decision methodology for congestion control, and the principal benchmark in existing networks is the behavior of TCP under load. So we must take a closer look at TCP's methodology, with an eye toward closely imitating its behavior in our methods.

As with flow control, TCP uses a window-based approach to congestion control. Here, the congestion window is based on the amount of data that is in flight (sent but not acknowledged by the receiver), rather than the amount of data expected inbound. The protocol also requires a retransmission timeout, based on a moving estimate of the round trip time between sender and receiver[76], to detect missing packets. TCP has three distinct stages of behavior. First, *slow start*, used at the beginning of a new session and after a major congestion event is detected. In this phase, the offered load is kept extremely low and increased very cautiously. If no congestion is detected in this phase, the protocol transitions to a period of rapid increase in window size (resulting in a rapid increase in transmission rates). Then, when some packets begin to be lost, the algorithm infers it is close to the available bandwidth and slows its window-size increases sharply, eventually plateauing at a rate close to, but just below, the highest rate that can be processed by the network. This phase is called *congestion avoidance*, and it will remain in this phase until a major congestion event is detected or the connection is closed.

Interestingly, while it is common to speak of TCP as a single protocol, there are actually a number of different TCP versions in use in production systems today, each of which implements a somewhat different congestion control algorithm. This is also a continuing area of research (both academic and corporate)—as a small sampling of many recent examples, see [77-80]. The reason is that, while a protocol following these steps will be stable under all conditions (*i.e.*, it will not cause congestion collapse in the network, and it will converge to a transmission rate near the available bandwidth), its performance characteristics are extremely sensitive to small variations in when and how to make changes to the congestion window, the physical characteristics of the network in question (particularly latency and error rate), and the traffic patterns of the higher-level applications using the TCP stack. In addition to multiple variations in the window sizing algorithms and thresholds for declaring a congestion event, there are also numerous published methods for tuning TCP to perform better over particular types of physical links, adjust time stamps, shape TCP windowing behavior, and apply various changes to system parameters[81]. Network devices (such as routers) and often configured specifically to tune their performance for TCP[82, 83], and network administrators often design quality of service (QoS) policies specifically to shape TCP flow behavior. The result is that "TCP-like" behavior is an inexact description (at best).

Interestingly, congestion control is also a major area of research for NDN as well[24, 84, 85]. In this case, congestion is not simply a question of the traffic on the link, but also consumption of space in the forwarding information base, pending interest table, and content store. Broadly, NDN, like most store-and-forward systems, can usually recover lost packets (whether due to congestion or to brief link outages), and so if congestion is a short-term event NDN will perform well. However, under sustained congestion, NDN's PIT and content store become flooded and unable to store requests long enough for the replies to return. In these cases, the network performance drops drastically, and it is possible to lose reachability to segments of the network. Also, as noted in chapter two, the current implementation of NDN (CCNx) has shifted to a hybrid between routed and flood-search architectures, and very little is known about how congestion will affect network stability. Additionally, the store-and-forward architecture gives at least the possibility that intelligent caching will enable the network to avoid congestion, and various initial attempts at developing such systems have been presented [86-88].

We draw three practical conclusions for the development of our protocol. First, we must implement some form of congestion control, even in the simplest prototype possible, in order to ensure that our test network will remain functional. Per the guidance given in RFC 2914[89], our method of congestion control should be no more aggressive in offering load to the network than TCP, and it should be at least as responsive as TCP in backing off in response to congestion (as measured by dropped or out of sequence packets). As we will explain further in the next chapter, we kept to the simplest possible solution for our prototype system: we merely hard-coded the congestion control mechanism from TCP into our traffic generator. Specifically, we used Cubic TCP (a less aggressive variant of BIC TCP) [90-92], because it is the default version of TCP used in our version of Linux. (However, we also ensured that our test network was not highly loaded, so no major congestion events occurred.) This ensured that our prototype would behave identically, in terms of congestion control at least, with the TCP system we were comparing it to.

Second, we note that we think it is very likely that our temporary solution to providing a congestion control mechanism will perform very poorly in a production environment, especially in comparison with TCP driven by higher-level applications such as web servers or browsers. There are two reasons for this belief. First, the TCP algorithms have more data available to them (in the form of a bi-directional, persistent data flow between the end points) than our protocol does, and so we would expect that simply importing the TCP methods directly would not give as good results. Second, TCP is designed to begin its flows in an extremely conservative way (slow start). Very short flows do not have time to scale up in speed, and so they remain quite inefficient and slow. (This is one of the main reasons why higher level applications, such as web servers, usually have a mechanism for maintaining a connection for a very long time, even when not actively trying to exchange data with the client—it sharply reduces the amount of time that the session spends in slow start.) Under our protocol, all data exchanges are single datagrams, so we would expect our system to be much less performant than a classic TCP with persistent sessions.

Third, and more fundamentally, our system architecture explicitly separates the data item from the connection to the physical location it is stored. We cannot assume that all CLIP datagrams are sourced from the same physical location, even if they are produced by the same publisher. We allow (and are expecting) the network to source CLIP datagrams from multiple sources and to take into account network congestion levels when forming the delivery graphs for the data. This re-thinking of the problem gives us both great promise—we will be able to let the network source data from a different physical location in response to congestion, and we will enable new options for caching or proxying data closer to the requestors and so reduce the amount of nodes and links transited—but also great uncertainty. As we will discuss in more detail in our concluding chapter, we believe that there is significant scope for further research and optimization over our current (admittedly naïve) approach to congestion control, particularly in the possibilities of using routing and switching rules that are both information- and congestion-aware in choosing the source and forming the delivery graphs for CLIP datagrams.

4.5 Summary

In this chapter, we have considered some of the necessary functionality that our proposed protocol would have to meet in order to meet the data transmission needs of higher-level applications, with a specific and detailed comparison to the current de facto standard, TCP. We conclude that most of the additional functionality provided by TCP (over and above the basic networking functionality that UDP provides) can also be easily met by our protocol, almost by default. The main exception is the question of congestion control. Here, we found that the existing theoretical development and the practical experience gained in the various versions of TCP were not 100% applicable to our methods, because the TCP results were predicated on measurements and parameters which are only fully pertinent to connection-oriented transport methods. We showed that, for our initial prototype, we instantiated the TCP congestion control methods, and that this was mathematically guaranteed to be stable; however, we expect that the performance characteristics under congestion would probably not be as good as could be expected from legacy TCP. We concluded that further research is well warranted into the issue of congestion control in an ICN design, and that potentially excellent new opportunities exist for performance enhancements and information- and congestion-aware routing and switching decisions could make a major difference in system performance.

CHAPTER FIVE

PERFORMANCE EVALUATION

5.1 Application Notes

Before we begin discussing the design of our prototype and test bed, it is worth discussing the system prerequisites for implementing our system. Although IPv6 has been specified fully since 2003[93], there are significant gaps in the completeness and correctness of the implementations in specific areas. IPv6 only entered significant mainstream production traffic after the exhaustion of IPv4 addresses in 2012-2013 made it impossible to avoid the issue. As of this time (fall 2014), IPv6 traffic amounts to roughly 4-5% of world IP traffic[94]. As a result, actually obtaining fully RFC-compliant IPv6 service and software (as required by this protocol) is, at least temporarily, more difficult to achieve than might be expected. In general, low-level systems are typically compliant, whereas higher-level systems often show significant lapses or errors in implementation. We summarize the key issues below.

5.1.1 Hardware and operating system issues

In general, there are few remaining hardware or device driver incompatibilities with the complete IPv6 specification. All major operating systems were updated to fully implement IPv6 by 2006, due to both industry and government requirements. However, due to bugs and security flaws in the initial implementations of many IPv6 network devices and software programs, it remains common for the IPv6 functionality in both hardware and operating systems to be disabled by default. Disabling IPv6 is also commonly recommended as part of the troubleshooting process for network failures and application crashes. As a result, it is necessary to check the system configuration to ensure that the system to be used for this prototype has IPv6 enabled by default. The precise details of the configurations vary significantly by operating system; we will limit our discussion to the Linux operating system, which is the system used for our prototype.

First, users should check that their system kernel has loaded the IPv6 modules, for example by entering the following code at the command prompt:

```
test -f /proc/net/if_inet6 && echo "Kernel is IPv6 ready"
```

If there is no output, the IPv6 module has been disabled. There are three places to look for problems. First, inspect the file `/etc/modprobe.d/blacklist.local` to see if IPv6 has been blacklisted. If so, remove the command and reboot your system. Next, inspect the file `/etc/modprobe.d/aliases`. The following entries will disable IPv6:

```
alias net-pf-10 off
alias ipv6 off
```

These entries should be replaced with:

```
alias net-pf-10 ipv6
```

Then reboot your system. The last place to look for problems is in the GRUB bootloader file `/etc/default/grub,` where the command `ipv6.disable=1` will disable IPv6 during system boot. If it is there, remove the command, reboot your system, and re-run the test.

Next, look for IPv6 to be enabled on each interface. This is found in the configuration files `at /etc/sysctl.conf`, looking for the entries:

```
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Change the values to zero, then run `sudo sysctl -p` or reboot your system to make the changes take effect. (Note that the Mininet network emulator may append these entries to your configuration files during installation by default, so one should re-check your system configurations after installing it.)

Under the appropriate RFCs, it is a requirement that, for dual-stack systems, IPv6 should be considered the primary interface (*e.g.* when querying the operating system for the host address). This can cause unexpected disconnection or exceptions in applications which were not written with IPv6 in mind. We would like to remind our readers of this fact, so that they may be able to troubleshoot any issues which may come up after activating IPv6 on their hosts. This is less common now, and will become progressively less likely as software updates are completed; however, one should be aware of the possibility.

5.1.2 Switching and routing issues

There are similar issues with switches and routers and the not-necessarily full and complete application and correct default configurations for IPv6. Again, the hardware, firmware and custom operating systems on commercial routers and switches are almost universally IPv6-compliant. However, it remains a common practice for system administrators to disable the IPv6 functionality of their networks, both for security and performance reasons. Additionally, even if they have enabled IPv6 traffic, they may not have installed the necessary network functionality (such as DNS serving AAAA records, DHCPv6, router advertisements, etc.) to enable plug and play operation. The issue is even more complicated if the network's IPv6 traffic is being tunneled through the IPv4 network, and there are many reports of ISPs using unusual configurations in their services. Users should contact their system administrators for specific configuration questions and to coordinate for network functions. Additionally, many networks rate limit UDP traffic, since poorly designed UDP applications may have not installed functioning network congestion response behavior; if so, these policies will limit the overall effectiveness of this protocol as well.

In this work, we are using a virtual network, provided by the network emulator Mininet.[95, 96]. This allows us to have full control over the network parameters. Note that this system is only IPv6-capable as of version 2.1.0. As mentioned in the previous section, installing Mininet may modify the host's configuration files to de-activate IPv6 system-wide, so configurations should be re-checked after installation or upgrade. The configuration file should also be changed to increase the size of various system parameters for IPv6 traffic. In addition, if using Mininet in a virtual machine (as recommended in the Mininet documentation), IPv6 must be enabled on the interface(s) being used within the VM, and a DHCPv6 server and a routing advertisement daemon (such as radvd) must be provided, or the interfaces must be manually configured on each host before the simulation begins (*e.g.,* by using a configuration script). Alternately, using a bridged network will allow DHCPv6 and router advertisements from the computer's external network to propagate into the virtual network. (Stateless autoconfiguration should work without a DHCPv6 server being present; however, since the VM typically generates a random MAC address for each interface, each session, there will be no way of knowing in advance what the addresses will be. This makes troubleshooting and configuration difficult. Additionally, these autoconfigured addresses are link-local addresses, and so will not be visible to hosts that do not share the link.) For our emulations, we decreased the throughput and increased the network latency to ensure that the network effects would dominate over effects due to either virtualization or CPU sharing.

For the switch used in the Mininet system, we used Open vSwitch[97]. This system fully supports IPv6 and the OpenFlow 1.3 standard as of version 2.3.0. Included with the distribution is a basic, OpenFlow compliant controller, which will implement a simple L2 switch. Note that the switch must be manually re-configured to use OpenFlow 1.3 after starting Mininet—the default is OpenFlow 1.0, which does not have IPv6 capability. It is worth mentioning that the version of Open vSwitch we used is also used as the native controller on many commodity L2 switch devices, and it can be used in place of the manufacturer's control software on many versions of switching hardware. It is also commonly used as a virtual switch in large data centers. This gives us a high degree of confidence that our testbed network accurately reflects the behavior of our protocol on production networks.

We also used the Ryu software defined network (SDN) controller[98, 99] to provide control of the switch. One must use version 3.14 or later, and include settings to use OpenFlow 1.3 or later, to be IPv6 compliant. As a reminder, like all SDN controllers, the Ryu controller does not provides network services or IPv6 specific protocols (such as neighbor discover protocol or duplicate address detection) without further programming, and the web server gateway interface (WSGI) features of the Ryu controller are not IPv6 capable as of this writing. (The reference controller included with Open vSwitch will implement the layer two protocols needed for IPv6 out of the box, but switching to a remote controller will disable all onboard functionality by default.) Also remember that, when using SDN controllers, the settings on the switches (and switches serving as routers) are done with the SDN controller, but the host configurations must be done either via DHCPv6 and routing advertisements, or by manually configuring each host (or using an install script).

We looked for an IPv6 compliant software router for use in our prototype. Unfortunately, the open source routers available for use as of the time of writing were not fully RFC-compliant. In particular, we found no open source router which correctly implemented parsing for the extension headers that we used for our system. (We also found many open bug reports pertaining to implementation of routing protocols on IPv6 in these routers.) As a result, for this work we instead installed static routes via the Linux routing module on the hosts, and programmed static routes in the switches (converting them from layer 2 to layer 3 devices via software, as is typical of software-defined networks). However, we should emphasize that this is a temporary problem in the open source community. Vendor-specific, closed source hardware routers are used reliably today on IPv6 networks, and we expect that the increasing use of IPv6 in important networks will eventually drive improvements to the open source software routers. While the calculation of IPv6 routes was not included in this work, the ultimate performance of the datagram forwarding should be identical regardless of how the route was determined.

5.1.3 Programming language and application issues

Once again, at the level of programming languages and specific applications, we found that full compliance with the IPv6 specification was far from universal at this time. Although the RFCs covering the IPv6 extension headers were published in 2003[93], concurrent with the necessary extensions of the C programming language, implementation in higher-level languages remains problematic. Specifically for this project, the fact that there was only one SDN controller available which could implement the OpenFlow 1.3 specifications needed for our protocol meant that a significant portion of our code would be written in the Python language[99]. As a practical necessity, we needed functionality in two major areas: a fully IPv6-compliant socket module (for transmitting and receiving the packets), and an asynchronous I/O module (to handle control flow within the program). Unfortunately, the Python language split into two incompatible branches (2.X and 3.X) prior to the inclusion of the commands needed for the IPv6 extension headers, but after the main asynchronous I/O modules had been written in version 2.X. Most production networking systems and utilities that are written in Python are still based on the older 2.X family. This meant that the Ryu SDN controller was written in Python 2.7, to take advantage of the excellent performance characteristics of the asynchronous `greenlet` module and a wide array of modules and programs for features such as SSL, web sockets, DNS services, and more. The developers handled the new requirement for IPv6 header fields by building in a packet sniffer, similar to the Scapy program[100]. While this is functional for the controller, it is impractical for general purpose use, since it requires the use of raw sockets and construction/parsing packets as data streams in user space. (In general, the controller only needs to parse the first packet in the flow, and it is already running with root privileges.) This left us in the unfortunate position of having most of the necessary functionality "stranded" in Python 2.7, while the necessary interfaces to the networking stack in the operating system giving access to the UDP extension headers—namely, the commands `sendmsg()` and `recvmsg()` and the associated data structure `msghdr`—only implemented in Python 3.4. Backporting this functionality into the Python 2.X family has been ruled out by the Python development team.

In the long term, this issue should fade away as more functionality within Python is ported to the Python 3.X family. In March 2014, some of the key building blocks for a full network solution were ported to Python 3.4 release, including a new (provisional) asynchronous I/O module (`asyncio`); a complete and stable `ipaddress` module; many security features involving TLS, certificate handling, and hashing; and many relevant bug fixes[101]. This means that, going forward, it should be possible to write a CLIP-specific transport object in Python, compatible with the existing UDP and TCP transport objects used in higher-level networking objects, such as `socketserver`, which in turn access higher level abstractions such as database APIs, cryptographic utilities, web services, DNS, and more. Conceptually, this gives us an indication of how this protocol might eventually be used in a production environment, and it indicates that the larger open source community is headed in the same direction as we are. For the present, however, we were forced to implement our prototype in two parts—traffic generation and parsing in Python 3.4, and the controller logic in Python 2.7.

5.2 Traffic generation for testing

Before beginning the analysis of the network performance of the protocol, we first had to develop a basic content server and client programs, to generate traffic flows that could be handled within the network. Since the focus of this work is a layer three/four network protocol, not a content distribution server optimization, we instantiated the simplest possible data source/sinks that would exercise the network behavior.

First, it is worth emphasizing that we are comparing our proposed protocol to TCP (at layer four), not higher-level protocols such as HTTP. Our TCP server is a simple design, listening for all content requests on one address and port, then creating a new worker thread to handle each request. Each request is considered an atomic transfer (i.e. not part of a larger stream) with no higher-level protocol actions taken and the socket closed after each request. Our CLIP server is even simpler: UDP servers do not need to maintain session state or wait for responses before sending, so there is no need for threading at all. Each publisher's data is handled by a separate process, with a distinct IPv6 address for each publisher (although they share the same physical or virtual interface with processes serving other publishers' data). This could, of course, be implemented equally as coroutines or threads within a larger server framework in a production system, but we used separate processes which could be attached and detached from interfaces at will. We also assume that there is no interaction between data items (*i.e.*, there is no need for locks in the TCP server, or interprocess communication in the CLIP server) and all data is pre-computed data strings stored in memory (rather than, for example, file transfers or objects on a storage array). In both systems, we assume that the clients already know the data item and publisher that they wish to access (*i.e.,* there is no look-up mechanism implemented on this network).

As noted in the previous chapter, congestion control and packet retransmission is a difficult question to answer. As a first attempt at the problem, we simply hard-coded a basic version of the same algorithm used by TCP into the client traffic generator in our network. While this is functional in the short term, it is probably not optimal for production traffic, as it makes implicit assumptions that all data with a given publisher label is sourced from the same place. This is true in our test network, but probably will not be true in production networks. To avoid biasing our results based on this open issue, we also kept the overall network utilization low enough that congestion events were rare. We also did not simulate packet loss in our networks. (Dropped packets were kept below 1% on all of our emulations by rate-limiting our traffic generators.)

5.3 Network configuration

The Mininet emulator does not have the concept of simulator time—packets are forwarded based on the overall system clock, not an internal simulator clock. As a result, it is possible to code a nework emulation which cannot be physically instantiated (*e.g.,* we may be able to code a 100 Gbps link into the emulator, but our system can only process 100 Mbps of packets in total, across all links). We also wanted to ensure that network effects dominated over effects due to our traffic generators, including the congestion control question mentioned above. As a result, we reduced the per-link bandwidth to 10 Mbps, with a delay of 10 milliseconds per link. We believe this demonstrates accurately the relative performance of TCP and CLIP UDP in low-congestion networks.

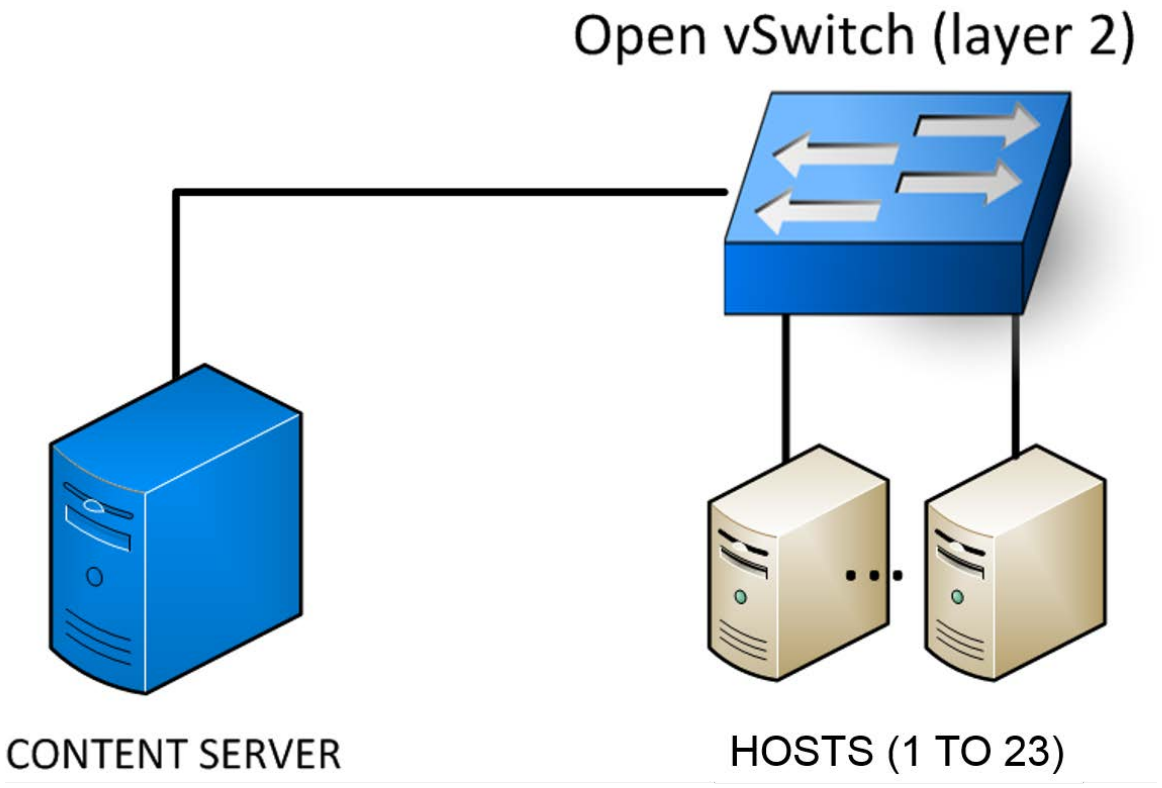


Figure 5. Switched network

In each case, we used one host on the network as a server, with the remaining hosts running client programs which accessed a randomly-chosen publisher-data item pair. As noted above, the TCP server ran one process on one IP address, and spawned a new thread to handle each client connection. The CLIP UDP server ran a separate (single thread) process for each publisher, with one IPv6 address per publisher. This meant that there were multiple public IP addresses on a given Mininet interface in the CLIP UDP case. Although the IPv6 RFCs make no limit to the number of IPv6 addresses which might be assigned to a specific interface, on our emulator there appeared to be a practical limit of about 24 IPv6 addresses assigned to each interface. In terms of the number of data items per publisher, we found no difference in performance for either TCP or CLIP

UDP, subject to the total size of the data structures in memory (*i.e.,* the performance was due entirely to memory and operating system issues, if the data items were accessed from a data structure held in memory.) A diagram of our switched network is given at Figure 5.

5.4 Network performance

We began our analysis of the protocol with a switched-only network. In this configuration, we have one virtual host serving content, and a variable number of other hosts requesting data items (chosen at random) from the server. We initialized the network with a pingall command, to ensure that all MAC addresses were registered with the switch before beginning performance evaluation. Here, we explored three main questions: first, can CLIP datagrams be forwarded normally through an unmodified switching fabric and the unmodified Linux operating system networking stack? (Remember, this is not true of the NDN/CCNx and PURSUIT reference designs discussed in chapter two.) Second, does the CLIP protocol give at least as good latency and throughput performance as standard TCP? And last, if both TCP and CLIP datagrams are comingled in the network, do they interfere with each other? In other words, would allowing information-centric networking datagrams onto a legacy host-oriented network cause service degradation to either protocol? We believe that favorable answers to these questions form an absolute requirement for a deployable ICN system.

5.4.1 TCP baseline performance

We begin by establishing a baseline performance benchmark in a TCP-only scenario. In this case, the behavior of the network flows is dominated by the TCP connection's requirements for a three-way handshake to initiate communication and four-way handshake to tear down the connection. Both the server and the client traffic generators were coded using the `sendall()` method, which requires the data to be transmitted immediately (without waiting for the ACK from the distant end) and thus would frequently (but not always) result in a PSH packet also being emitted. (In this network, all packets arrived in order, and packet loss was kept below 1% by rate limiting the traffic generators.) As expected, adding a 10 msec delay for each link resulted in an overall transmission time of at least 85 msec; in cases where one packet was lost, delays could reach 110 msec. As we increased the number of hosts simultaneously attempting to request data from the server, we saw modest increases in the average delay, from an average of 90 msec with one client only, rising to 95 msec average for 23 clients submitting requests simultaneously. This was principally due to packet queueing in the server's send and receive buffers. We regard this baseline as a good example of a "best case" scenario for the current, host-oriented networking protocols. TCP has been under continuous development and heavy use for over 40 years, and both the operating system and the network infrastructure are highly optimized for TCP performance. The network was lightly loaded, which led to very few retransmissions (as mentioned above, we would expect TCP to show poor performance with retransmissions, since it would be in its slow-start stage basically continuously with short flows such as these).

5.4.2 CLIP baseline performance

Next, we implemented the CLIP protocol alone on the network, to verify that an unmodified Linux operating system, unmodified switching system, and an open source SDN controller could successfully form, transmit, receive, and parse CLIP datagrams, and that the network could correctly classify and respond to information-centric datagrams. We were fully successful in doing so. Additionally, since this protocol has no concept of connections, there was no need for session setup or teardown. Instead, data transmissions were normally only a single pair of datagrams. Packet loss was detected by a simple timeout on the client—no reply matching the request by the timeout was counted as a packet loss, and the individual request was retransmitted. There was no code needed at all on the server for lost packets, windowing, etc.

Performance was in line with expectations for non-connection oriented traffic. For a single client network, average delay times were 45 msec, which was identical with the average delay observed for ICMP datagrams generated by the ping commands used to start the network emulation. This means that the information labelling on CLIP packets adds no measurable delay to forwarding decisions, queues, or network card behavior—a key performance objective for this system. Although these headers are experimental, both the operating system and the switching infrastructure have no difficulty in generating, matching or parsing the data fields needed. Standard open-source SDN controllers can also be successfully programmed to produce the forwarding plane behavior desired.

Average delay rose only to 47 msec when the number of hosts was increased to 27. As with TCP, the increase was primarily due to datagrams queueing in the receive or transmit buffers at the server; however, note that the rate of increase was slower with CLIP than with TCP, as there were fewer packets in the series which would be subject to these intermittent delays. Packet loss detection on the client was hard-coded to match the stock TCP settings, so no difference was expected due to this factor.

5.4.3 Commingled TCP and CLIP performance

The next performance question to be addressed is whether host-oriented and information-centric datagrams can coexist on the same system and network architecture without interfering with each other. This is necessary, because we expect that host-oriented traffic will continue to be used for tasks that are inherently host-to-host data flows (*e.g.* VoIP calls), and we further expect that no production network provider would be willing to implement an experimental architecture without concrete assurances that the experimental traffic would not disrupt normal network behavior.

To test this situation, we comingled both CLIP and normal TCP on the same test network, by running both servers simultaneously on the same host and running TCP and CLIP client programs on the same client hosts. Specifically for this network, the server on host 1 opened one listening socket on 2001:1::11 for TCP traffic (forking new threads for each accepted connection request) and served TCP content requests for all 23 publishers from that IP address. At the same time, host 1 also opened CLIP listening sockets on 2001:1::C000:0:0:1 through 2001:1::C000:0:0:23, serving content for one publisher for each IP address. All 24 IP addresses were assigned to the same (virtual) ethernet interface and shared the same 10 Mbps link to the switch. On the clients, each host had one IP address, with both CLIP and TCP traffic generators sharing the IP address, virtual interface and 10 Mbps link.

The results for comingled traffic were virtually identical with the single-use cases (both TCP and CLIP). Some small increases resulted occasionally due to queueing on the interfaces' transmit and receive buffers (which were being shared more aggressively in this case than in the single use cases). However, this is primarily due to the overall network load, rather than conflict between the protocols (either in the operating system's networking stack or the network itself). Overall results are given at Figure 6. We conclude that a network made of standard networking equipment and the (unmodified) Linux operating system can easily handle the experimental protocol we have designed, and that the experimental traffic can be mixed with legacy traffic safely and effectively.

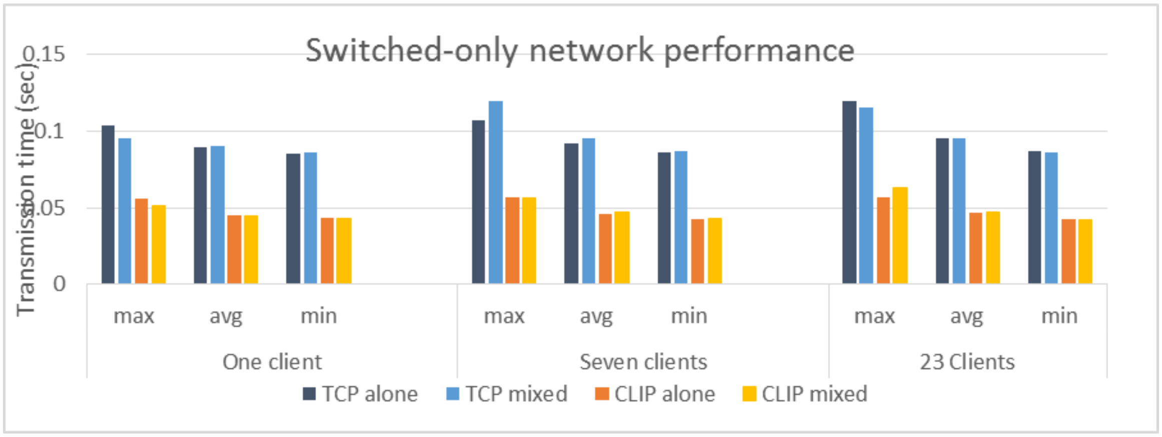


Figure 6. Transmission delays for TCP and CLIP traffic on a switched-only network

5.4.4 Performance in a routed network

For completeness, we also instantiated a routed network, to verify that the network behavior remained consistent across network boundaries. As noted above, we used the Linux routing module as the routing method, interconnecting two separate network domains (2001:1::/64 and 2001:2::/64). As in the switched network, one host on the 2001:1::/64 network served content both on TCP and CLIP UDP; however, in this configuration, the client requesting content was on the 2001:2::/64 network. A diagram is given in Figure 6.

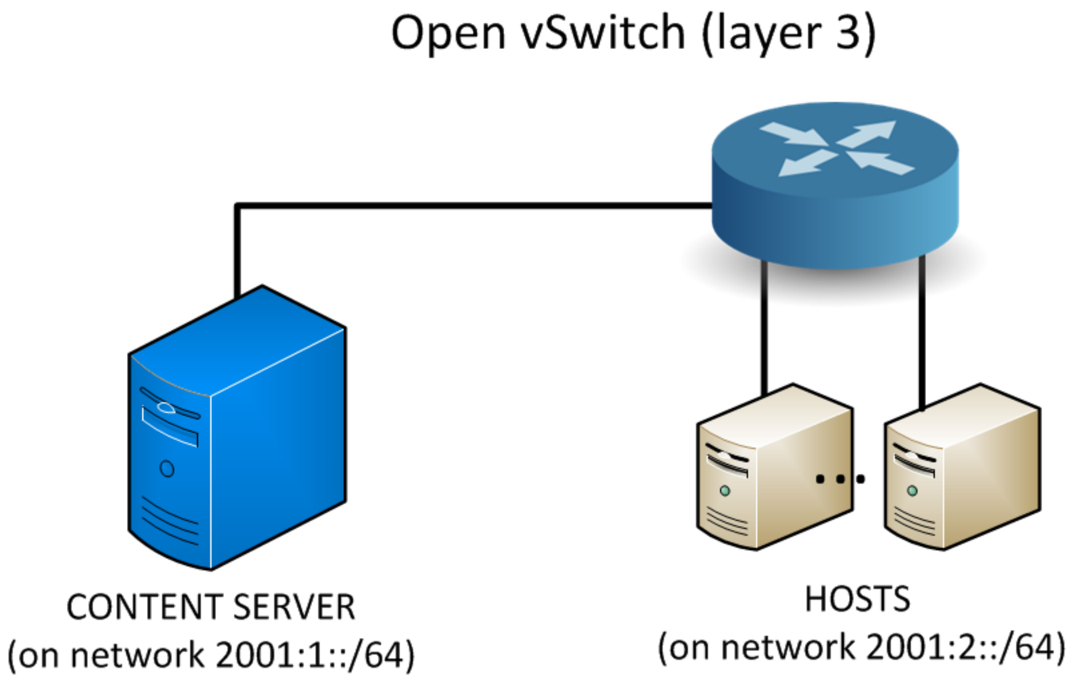


Figure 6. Routed network.

In software defined networks, the forwarding plane decisions are reduced to a match on packet field entries—there is no distinction in practice between decisions based on layer four fields (*e.g.* TCP or CLIP headers), layer three information (IP address and network prefix) and layer two information (ethernet frame headers) when forwarding datagrams. Thus, we expected no difference in performance between a routed and a switched network, either for TCP or CLIP. This was, in fact, the case—latency remained at an average of 89-90 msec for TCP and 44-45 msec for CLIP packets, and performance characteristics remained identical to the previous case.

We would like to note that, in this case, we made no distinction between handling information-centric datagrams and host-oriented datagrams. However, if making such a distinction was desirable—for example, directing all information-centric traffic to a specific server, network appliance or cluster of computing resources—this could be done very straightforwardly with routing. Specifically, the network administrator would simply route based on a 68-bit network prefix rather than the 64-bit prefix shown above. In our example, the information-centric subnetwork would be 2001:1:0:0:C/68, whereas the traditional host-oriented subnetwork would be the remainder of the address space.

5.5 Comparison to Named Data Networking

We began our discussion of information centric networking with an extended analysis of the NDN approach, with a particular focus on the reference implementation, CCNx. We concluded chapter two with the comment that we believe the flaws and missing functionality of NDN/CCNx are so great that it is unlikely to ever be implementable in a production environment. Now that we have completed our development of our competitor protocol, it is worth returning briefly to compare our design with the best-available alternate ICN design to demonstrate how much functionality and performance capability one would gain by using our design rather than CCNx.

As is shown in Figure 7, we believe that we have shown a protocol that is fundamentally much superior to the CCNx design. By meeting the same interfaces as standard IPv6, we are able to make use of many theoretical and practical results that CCNx is incapable of addressing: from full compatibility with current network architecture fundamentals, such as QoS queue policies and VLANs, to an easy implementation on standard hardware and software and full backwards compatibility with the current IPv6 networks. We believe this will give both researchers and network

		NDN	**CLIP**
Layer 2	Switching	Store-and-forward at each node, interests propagate by flooding Requires custom hardware (large memory caches for PIT, FIB, and content store) Slower than line speed (approx. 10x slower)	Standard datagram forwarding No hardware or software modifications needed, no additional memory requirements Line speed forwarding *Optionally*, can specify different forwarding rules for CLIP and point-to-point datagrams
	VLAN	No VLAN capability demonstrated	Fully compatible with VLANs
	QoS	No QoS capability demonstrated	Normal QoS policies applied (via SDN controller or programming the switch/router) *Optionally*, can specify different QoS policy for CLIP and point-to-point traffic, different producers, or different data items
Layer 3	Routing	Hybrid routing system No stability or reachability results	Standard routing system, guaranteed stability No requirement for separate routing algorithm or table *Optionally*, can maintain separate routing table for ICN traffic
	Subnetting	Cannot subnet NDN traffic	Normal subnetting process
	Internetworking	Not compatible with BGP Does not self-assemble into internetworks	Compatible with unmodified BGP, self-assembles via normal BGP advertisements Stability guaranteed via Gao-Rexford *Optionally*, can use BGP extended communities attributes to pass ICN-specific information
Layer 4	Fragmentation	Fragmentation method unspecified	Application must perform MTU discovery and fragmentation (per IPv6 standard)
	Encryption	Unspecified	IPSec standard

Figure 7 Comparison of NDN and CLIP features

administrators a clear path to pursue the potential gains of an information-centric architecture.

In this chapter, we compared our protocol to the best and most widely-used transport protocol in the Internet today: TCP, which has been under intensive study and development for four full decades. We would like to remind our readers that this is an exceptionally ambitious comparison for a new protocol, and it is only because we believe that our design provides features and benefits that simply cannot be compared to NDN that we do so.

5.6 Summary

In this chapter, we have presented a functional prototype of our information-centric networking design. We covered the system and software prerequisites needed to implement this design, and we highlighted several possible issues or misconfigurations which might be problematic. We also described the programming environment needed to develop the SDN controller and traffic generators for our testbed system. Finally, we instantiated a simple network on the Mininet network virtualization and compared our prototype to the industry standard TCP protocol. We concluded that our ICN system is deployable on unmodified hardware, operating systems and software now, and that its performance characteristics are good enough to warrant further development and refinement.

CHAPTER SIX

CONCLUSION

In this work, we began by considering the area of information centric networking designs, developed over the last 6-10 years by various research groups. Although these systems have many positive features, there were insurmountable problems with each of them in the areas of routing reachability and stability, scalability, and security. In addition, there were very large practical considerations about interoperability, the cost and feasibility of major hardware and software changes, and backwards compatibility with the legacy network, which prevented these designs from being feasible for deployment.

However, advances in the roll-out of IPv6 and the rise of software defined networking have opened the door to a new approach to ICN, based on incorporating the metadata needed for ICN into the standard IPv6 header and using SDN to provide the detailed forwarding rules needed for true information-oriented networking. We were able to implement our design on standard hardware and software, and to make use of the deep and robust theoretical and practical results developed over four decades of work with IP. We were able to establish a fully-functional switching and routing design, inheriting the stability and reachability results gained from routing protocol research and practice in IP. We presented how our design was backwards compatible with the legacy IP network and does not require special permissions from the network provider or changes to the host's operating system or hardware to function. We showed that we can intermix host-oriented and information-oriented traffic, without interference. And finally, we showed that an information-centric transport protocol is feasible to implement, is able to meet the application-layer requirements for reliable transport, and may be both faster and simpler to implement for certain use-cases. We conclude that this system is a feasible approach for future internet design.

6.1 Work remaining for full protocol implementation

The title of this work is “Toward an Internet Protocol Version 6 Deployable Information Centric Networking Framework”, and so it is worth outlining briefly why it is “toward,” that is, what still remains to be accomplished in order for CLIP to be considered a complete, deployable protocol. We summarize the work we have completed, and the key items remaining to be done, in Figure 8. We believe that we have demonstrated most of the capabilities needed for deployment of our protocol, and we have a clear approach to achieving the remaining items. We also believe that our design, even though in its early stages, is clearly superior to other ICN prototypes which have been presented to the community for research and application.

6.2 Future research

Changing the networking approach from fixed-endpoint flows to a fully information-centric flows opens many avenues for future research. Here, we highlight a few of the many possibilities.

6.2.1 Develop an abstraction of the transport layer for use in Python

As noted in chapter five, the release of Python 3.4 in the summer of 2014 made a number of key programming modules available for the first time. In particular, the inclusion of the `asyncio` library (on a provisional basis), along with the full implementation of IPv6 and the continuing improvements in the code base for many network services and programs, makes an attractive opportunity for developing this prototype into a developer-friendly toolset. In particular, we think that it is a high-value proposition to develop a transport and protocol abstraction for CLIP, which would fit well within the standard networking framework in Python. Under this abstraction, we

		Requirements	**Results or Approach**
Layer 2	Performance	Forward datagrams at line speed Commingle CLIP and endpoint-oriented datagrams without interference	Demonstrated in this work
	Compatibility	Compatible with unmodified operating systems, switches, and SDN controller	Demonstrated in this work
Layer 3	Routing	Compatible with standard routing protocols, stable, reachable No performance degradation in comparison with endpoint-oriented traffic	Demonstrated using Linux routing module *Future work:* Implement a stand-alone router using OSPF and IS-IS
	Transport layer functionality	Provide: flow control, error checking, datagram reordering, congestion control	Demonstrated flow control, error checking in this work Initial solution for congestion control (provides network safety) Datagram reordering is not relevant to this protocol, as each datagram is separately addressed Showed that this protocol does not delay in providing data to the application waiting for missing datagrams *Future research:* develop better congestion control algorithms; investigate the benefits of selective network caching
	Internetworking	Compatible with BGP Self-assemble into internetworks	Showed operation of BGP *Future research:* Implement internetwork with BGP *Future research:* Evaluate possibilities of extended community attributes in BGP for performance

Figure 8 Work completed and items remaining to be accomplished

would code a transport object, which handles the actual I/O, and a protocol object, which interfaces with the transport object and handles flow control, data parsing, and similar tasks. Higher level objects and applications would then use the protocol to read and write data, abstracting away the specifics of the layer four and below mechanics.

If successful, this would have a number of immediate benefits for the research community. First, it would allow native use of a wide variety of Python libraries, including cryptographic suites, database APIs, higher-level networking libraries (such as web servers) and data handlers. We are particularly interested in the libraries implementing SSL and certificate handlers, because (as described in chapter three) we believe that any production implementation of our protocol would require the use of cryptographic signatures as a minimum, and likely also encryption.

Additionally, there are a wide variety of Python implementations of many networking services, such as DNS, which are a practical necessity for more complicated research and testing (not to mention actual production networks). We believe that there is significant scope for applying these network services in an information-aware or even information-focused manner, to improve performance of the network and to enable innovative and effective network design.

6.2.2 Investigate congestion control mechanisms in ICN networks

We wrote at some length in chapter four about how we applied the issue of congestion control to our prototype. To reiterate, we used the TCP congestion control algorithm in our work to date, simply because we needed a guaranteed stable algorithm in order to produce an initial implementation. However, we realize that our system makes some very fundamental changes to the structure of the data flows, and as a result, we expect that there are much better approaches possible. In TCP, the decision to retransmit (when, and how many times) is made by the sender, based both on any acknowledgement data it may have received from the receiver, plus two different timers (to handle the situation where data from the recipient has been dropped), an estimate of the round trip time to the distant node, a record of the networking stack's last settings before the previous congestion events, a measurement of the line speed of its immediate network connection, and a handful of other system parameters that may have been tuned by the network administrator. Our method is different in a number of ways: first, the decision to retransmit is made by the receiver (by means of sending another request) rather than the sender. This will obviously change the dynamics of the decision, as well as what information might be available when the decision is made. Second, we cannot assume that the endpoint of the transmission is necessarily the same from datagram to datagram, as we explicitly allow the network to source the data from different locations (*e.g.*, from a cache or proxy), so an attempt to measure the round trip time may be highly error prone. Third, the traditional methods of tuning TCP performance, particularly window shaping and the behavior in response to RED or other queue management methods, may produce significantly different results on an information-centric network than on an endpoint-oriented network. Furthermore, an information-aware network might be able to actively avoid congestion events by intelligently staging and serving content-labeled datagrams or alternate routing ICN content, making congestion control less difficult to achieve. Lastly, we may be able to uncover possibilities for the network to provide certain "tuning parameters" to the end hosts, for publishers or data which the network recognizes (either because it has a content distribution scheme pertaining to that information, has a cached copy of the data, or has a routing table entry covering the data and so has more information about the network characteristics which might apply).

For all these reasons, we expect that a back-to-basics analysis of the congestion control mechanisms in an information-centric environment is worthwhile. Additionally, we would expect a significant amount of learning to occur if and when this system is used for production traffic and on heterogeneous networks.

6.2.3 Investigate network caching

We believe that one of the main anticipated benefits of this information-centric approach is to enable the effective implementation of automatic caching of data when and where most appropriate for the data in question. Since CLIP datagrams stand on their own (even regarding authentication and encryption), and there is no session-related overhead or communication requirements, network engineers are free to cache data at will. However, this system does not *require* large-scale caching, as NDN does; it has the ability to "fail soft" (default to point-sourcing) if the cache is not present, full, or unable to process the data. This allows a substantial freedom of design, as well as the possibility of avoiding many cases of network congestion and faster recovery from link outages. As a result, we believe that there could be significant benefit in designing a network caching system, probably implemented either as a network appliance or as a peer-to-peer system on end hosts. We are particularly interested in adapting the Python `memcached` API (a daemonized memory cache dictionary typically used for short-term storage of database queries for reuse by other applications), which would seem to have a relatively easy application to the (publisher label, content label) addressing scheme we have used for CLIP.

6.3 The future of the network

A fully-deployed information-centric network could bring significant performance improvements to the legacy network. We emphasize that, unlike other ICN designs, our system facilitates information-centricity, but it does not require it—in contrast to NDN and other systems, there is no need to change the architecture of applications which are inherently point-to-point (such as telephony, chat, or identity management). Nonetheless, we would envision that, if our design became common, that there would be a wide variety of information-aware network functions, services, and appliances that could be developed. For example, we imagine providing information-aware security services, that could impose policies based on the identity of the information, not a hypothesis based on the endpoints and protocols. We imagine the simplification of the network, eliminating the many layers of proxies, TCP accelerators, deep packet inspections, attempts at data deduplication, and more, which make today's networks so brittle and difficult to configure, manage, and secure. We imagine automatic data caching when and where it is most beneficial, based on the physical and logical characteristics of the network and the specific needs of the applications using the network. We imagine networks which can help end hosts and applications make intelligent choices about sourcing data and verifying its provenance. In short, we believe that this protocol opens many doors to future researchers and developers, to fundamentally improve the performance and security of the network, by better matching the actual needs of the application to the layer three implementation done by the networked devices today.

REFERENCES

[1] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, *et al.*, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 181-192, August 2007.

[2] *www-dsg.stanford.edu/triad*.

[3] D. Perino and M. Varvello, "A reality check for content centric networking," presented at the Proceedings of the ACM SIGCOMM workshop on Information-centric networking, Toronto, Ontario, Canada, 2011.

[4] *www.named-data.net*.

[5] *www.ccnx.org*.

[6] L. Zhang, A. Afansyev, J. Burke, V. Jacobson, K. C. Claffy, P. Crowley, *et al.*, "Named Data Networking," *ACM SIGCOMM Computer Communication Review,* vol. 44, pp. 66-73, July 2014.

[7] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "Adaptive forwarding in named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 42, pp. 62-67, 2012.

[8] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and Z. Lixia, "Interest flooding attack and countermeasures in Named Data Networking," in *IFIP Networking Conference, 2013*, 2013, pp. 1-9.

[9] T. Jianqiang, Z. Zhongyue, L. Ying, and Z. Hongke, "Identifying Interest Flooding in Named Data Networking," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, 2013, pp. 306-310.

[10] Z. Li and J. Bi, "Interest cash: an application-based countermeasure against interest flooding for dynamic content in named data networking," presented at the Proceedings of The Ninth International Conference on Future Internet Technologies, Tokyo, Japan, 2014.

[11] P. Gasti, G. Tsudik, E. Uzun, and Z. Lixia, "DoS and DDoS in Named Data Networking," in *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, 2013, pp. 1-7.

[12] C. Ghali, G. Tsudik, and E. Uzun, "Network-Layer Trust in Named-Data Networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 12-19, 2014.

[13] S. DiBenedetto, C. Papadopoulos, and D. Massey, "Routing policies in named data networking," presented at the Proceedings of the ACM SIGCOMM workshop on Information-centric networking, Toronto, Ontario, Canada, 2011.

[14] W. Kai, Z. Huachun, C. Jia, and Q. Yajuan, "RDAI: Router-Based Data Aggregates Identification Mechanism for Named Data Networking," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on,* 2013, pp. 116-121.

[15] H.-g. Choi, J. Yoo, T. Chung, N. Choi, T. Kwon, and Y. Choi, "CoRC: coordinated routing and caching for named data networking," presented at the Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems, Los Angeles, California, USA, 2014.

[16] J. J. Garcia-Luna-Aceves, "Name-based content routing in information centric networks using distance information," presented at the Proceedings of the 1st international conference on Information-centric networking, Paris, France, 2014.

[17] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "On the role of routing in named data networking," *Proceedings of the 1st international conference on information-centric networking (INC '14),* pp. 27-36, 2014.

[18] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhand, and L. Wang, "NLSR: named-data link state routing protocol," *Proceedings of the 3rd ACM SIGCOMM Workshop on Information Centric Networking (ICN '13),* pp. 15-20, 2013.

[19] A. J. Abu, B. Bensaou, and J. M. Wang, "Interest packets retransmission in lossy CCN networks and its impact on network performance," *Proceedings of the 1st international conference on information-centric networking (INC '14),* pp. 167-176, 2014.

[20] X. Jiang and J. Bi, "Interest set mechanism to improve the transport of named data networking," *SIGCOMM Comput. Commun. Rev.,* vol. 43, pp. 515-516, 2013.

[21] M. Papalini, A. Carzaniga, K. Khazaei, and A. L. Wolf, "Scalable routing for tag-based information-centric networking," *Proceedings of the 1st international conference on information-centric networking (INC '14),* pp. 17-26, 2014.

[22] X. Yuemei, L. Yang, L. Tao, Z. Guoqiang, W. Zihou, and C. Song, "A dominating-set-based collaborative caching with request routing in content centric networking," in *Communications (ICC), 2013 IEEE International Conference on,* 2013, pp. 3624-3628.

[23] M. Almishari, P. Gasti, N. Nathan, and G. Tsudik, "Optimizing bi-directional low-latency communication in named data networking," *SIGCOMM Comput. Commun. Rev.,* vol. 44, pp. 13-19, 2013.

[24] M. Amadeo, A. Molinaro, C. Campolo, M. Sifalakis, and C. Tschudin, "Transport layer design for named data wireless networking," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, 2014, pp. 464-469.

[25] H. Dai, B. Liu, Y. Chen, and Y. Wang, "On pending interest table in named data networking," presented at the Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems, Austin, Texas, USA, 2012.

[26] W. Yi, X. Boyang, T. Dongzhe, L. Jianyuan, Z. Ting, D. Huichen, *et al.*, "Fast name lookup for Named Data Networking," in *Quality of Service (IWQoS), 2014 IEEE 22nd International Symposium of*, 2014, pp. 198-207.

[27] C. Jaeyoung, H. Jinyoung, C. Eunsang, K. Hyunchul, K. Taekyoung, and C. Yanghee, "Performance comparison of content-oriented networking alternatives: A tree versus a distributed hash table," in *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on*, 2009, pp. 253-256.

[28] W. So, A. Narayanan, and D. Oran, "Named data networking on a router: fast and dos-resistant forwarding with hash tables," presented at the Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems, San Jose, California, USA, 2013.

[29] F. Li, F. Chen, J. Wu, and H. Xie, "Longest Prefix Lookup in Named Data Networking: How Fast Can It Be?," in *Networking, Architecture, and Storage (NAS), 2014 9th IEEE International Conference on*, 2014, pp. 186-190.

[30] W. So, T. Chung, H. Yuan, D. Oran, and M. Stapp, "Toward terabyte-scale caching with SSD in a named data networking router," presented at the Proceedings of the tenth ACM/IEEE symposium on Architectures for networking and communications systems, Los Angeles, California, USA, 2014.

[31] Y. Wang, D. Tai, T. Zhang, J. Lu, B. Xu, H. Dai, *et al.*, "Greedy name lookup for named data networking," presented at the Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems, Pittsburgh, PA, USA, 2013.

[32] Y. Haowei, S. Tian, and P. Crowley, "Scalable NDN Forwarding: Concepts, Issues and Principles," in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, 2012, pp. 1-9.

[33] http://www.cidr-report.org/as2.0/. (31 October 2014).

[34] J. Klensin. RFC 5324 Simple Mail Transfer Protocol [Online].

[35] G. Lindberg. RFC 2505 Anti-Spam Recommendations for SMTP MTAs [Online].

[36] W. Wong and P. Nikander, "Secure naming in information-centric networks," presented at the Proceedings of the Re-Architecting the Internet Workshop, Philadelphia, Pennsylvania, 2010.

[37] B. Hamdane, M. Msahli, A. Serhrouchni, and S. G. El Fatmi, "Data-based access control in named data networking," in *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*, 2013, pp. 531-536.

[38] B. Hamdane, A. Serhrouchni, A. Fadlallah, and S. G. E. Fatmi, "Named-Data security scheme for Named Data Networking," in *Network of the Future (NOF), 2012 Third International Conference on the*, 2012, pp. 1-6.

[39] B. Hamdane, A. Serhrouchni, and S. G. E. Fatmi, "Access control enforcement in Named Data Networking," *8th International Conference on Internet Technology and Secured Transactions (ICITST),* pp. 576-581, 2013.

[40] N. Ntuli and H. Sunyoung, "Detecting router cache snooping in Named Data Networking," in *ICT Convergence (ICTC), 2012 International Conference on*, 2012, pp. 714-718.

[41] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda, "Privacy risks in named data networking: what is the cost of performance?," *SIGCOMM Comput. Commun. Rev.,* vol. 42, pp. 54-57, 2012.

[42] *www.fp7-pursuit.eu/PursuitWeb/*.

[43] *www.psirp.org*.

[44] *github.com/fp7-pursuit/blackadder*.

[45] G. Parisis, D. Trossen, and D. Syrivelis, "Implementation and Evaluation of an Information-Centric Network," *IFIP Networking Conference, 2013,* pp. 1-9, 2013.

[46] J. Postel. RFC 791, The Internet Protocol [Online].

[47] *http://www.internetsociety.org/deploy360/blog/2013/04/over-25-of-verizon-wireless-traffic-is-now-over-ipv6/*.

[48] *http://www.worldipv6launch.org/*.

[49] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, *et al.*, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Computer Communication Review,* vol. 38, March 2008 2008.

[50] *https://www.opennetworking.org/*.

[51] O. N. Foundation, "OpenFlow Switch Specification (version 1.3) ", ed, 2012.

[52] L. Heath, H. Owen, R. Beyah, and R. State, "CLIP: Content labeling in IPv6, a layer 3 protocol for information centric networking," in *Communications (ICC), 2013 IEEE International Conference on*, 2013, pp. 3732-3737.

[53] T. Narten, G. Huston, and L. Roberts. RFC 6177 IPv6 Address Assignment to End Sites [Online].

[54] R. Hinden and S. Deering. RFC 4291 IP Version 6 Addressing Architecture [Online].

[55] S. Thomson, T. Narten, and T. Jinmei. RFC 4862 IPv6 Stateless Address Autoconfiguration [Online].

[56] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, and X. Li. RFC 6052 IPv6 Addressing of IPv4/IPv6 Translators [Online].

[57] R. Hinden and B. Haberman. RFC 4193 Unique Local IPv6 Unicast Addresses [Online].

[58] C. Huitema and B. Carpenter. RFC 3879 Deprecating Site Local Addresses [Online].

[59] M. Richardson. RFC 4025 A Method for Storing IPsec Keying Material in DNS [Online].

[60] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. RFC 4120 The Kerberos Network Authentication Service (V5) [Online].

[61] S. Kent. RFC 4302 IP Authentication Header [Online].

[62] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in content-oriented architectures," presented at the Proceedings of the ACM SIGCOMM workshop on Information-centric networking, Toronto, Ontario, Canada, 2011.

[63] S. Kent. RFC 4303 IP Encapsulating Security Payload (ESP) [Online].

[64] Y. Rekhter. RFC 5701 IPv6 Address Specific BGP Extended Community Attribute [Online].

[65] S. Sangli, D. Tappan, and Y. Rekhter. RFC 4360 BGP Extended Communities Attribute [Online].

[66] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: seeing the forest for the trees," presented at the Proceedings of the 10th ACM Workshop on Hot Topics in Networks, Cambridge, Massachusetts, 2011.

[67] M. Mathis, J. Mahdavi, S. Floyd, and M. Romanow. RFC 2018, TCP Selective Acknowledgment Options [Online].

[68] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolski. RFC 2883, An Extension to the Selective Acknowledgement (SACK) Option for TCP [Online].

[69] D. Borman, B. Braden, V. Jacobson, and R. Scheffnegger. RFC 7323, TCP Extensions for High Performance [Online].

[70] V. Jacobson, R. Braden, and D. Borman. RFC 1323, TCP Extensions for High Performance [Online].

[71] S. Bellovin. RFC 1948, Defending Against Sequence Number Attacks [Online].

[72] F. Gont. RFC 6528, Defending Against Sequence Number Attacks [Online].

[73] Z. Qian and Z. M. Mao, "Off-path TCP Sequence Number Interference Attack-How Firewall Middleboxes Reduce Security," *2012 IEEE Symposium on Security and Privacy*, pp. 347-361, 2012.

[74] Z. Qian, Z. M. Mao, and Y. Xie, "Collaborative TCP sequence number inference attack: how to crack a sequence number in under a second," *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12),* 2012.

[75] J. Nagle. RFC 896, Congestion Control in IP/TCP Internetworks [Online].

[76] V. Paxson, M. Allman, J. Chu, and M. Sargent. RFC 6298, Computing TCP's Retransmission Timer [Online].

[77] X. Sun, "TCP Congestion Control Algorithm Research," *8th International Conference on Information Science and Digital Content Technology (ICIDT),* vol. 3, 2012.

[78] P. Yang, W. Luo, and L. Xi, "Towards Measuring the Deployment Information of Different TCP Congestion Control Algorithms: The Multiplicative Decrease Parameter," *Global Telecommunications Conference (GLOBECOM 2010),* pp. 1-5, 2010.

[79] Q. Wang and D. Yuan, "An improved TCP congestion control mechanism with adaptive congestion window," *2010 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS),* pp. 231-235, 2010.

[80] Y. Nemoto, K. Ogura, and J. Katto, "An adaptive TCP congestion control having RTT-fairness and inter-protocol friendliness," *2013 IEEE Consumer Communications and Networking Conference (CCNC),* pp. 178-183, 2013.

[81] E. Weigle and W.-c. Feng, "A Comparison of TCP Automatic Tuning Techniques for Distributed Computing," 2002.

[82] K. Avrachenkov, U. Ayesta, J. Doncel, and P. Jacko, "Optimal congestion control of TCP flows for internet routers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, pp. 62-64, December 2012 2012.

[83] H. Wang, Z. Tian, and Q. Zhang, "Self-Tuning Price-Based Congestion Control Supporting TCP Networks," *Proceedings of the 19th International Conference on Computer Communications and Networks (ICCCN), 2010,* pp. 1-6, 2010.

[84] S. Salsano, A. Detti, M. Cancellieri, M. Pomposini, and N. Blefari-Melazzi, "Transport-layer issues in information centric networks," presented at the Proceedings of the second edition of the ICN workshop on Information-centric networking, Helsinki, Finland, 2012.

[85] M. Amadeo, C. Campolo, and A. Molinaro, "Multisource data retrieval in IoT via Named Data Networking," *Proceedings of the 1st international conference on information-centric networking (INC '14),* pp. 67-76, 2014.

[86] P. Heungsoon, J. Hoseok, and K. Taewook, "Popularity-based congestion control in named data networking," in *Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on*, 2014, pp. 166-171.

[87] J. Li, H. Wu, B. Liu, J. Lu, Y. Wang, X. Wang*, et al.*, "Popularity-driven coordinated caching in named data networking," presented at the Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems, Austin, Texas, USA, 2012.

[88] J. M. Wang and B. Bensaou, "Improving Content-Centric networks performance with progressive, diversity-load driven caching," in *Communications in China (ICCC), 2012 1st IEEE International Conference on*, 2012, pp. 85-90.

[89] S. Floyd. RFC 2914, Congestion Control Principles [Online].

[90] *BIC and CUBIC TCP*. Available: http://research.csc.ncsu.edu/netsrv/?q=content/bic-and-cubic

[91] W. Hua and G. Jian, "Analysis of TCP BIC Congestion Control Implementation," *2012 International Computer Science & Service System (CSSS),* pp. 781-784, 2012.

[92] J. Wang, J. Wen, Y. Han, J. Zhang, C. Li, and X. Zhang, "CUBIC-FIT: A High Performance and TCP CUBIC Friendly Congestion Control Algorithm," *Communications Letters of the IEEE,* vol. 17, pp. 1664-1667, 2013.

[93] W. Stevens, M. Thomas, E. Nordmark, and T. Jinmei. RFC 3542, Advanced Sockets Application Programming Interface for IPv6 [Online].

[94] Google. (2014). *IPv6 Adoption*. Available: http://www.google.com/intl/en/ipv6/statistics.html

[95] *www.mininet.org*.

[96] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hot Nets-IX),* October 2010.

[97] *http://openvswitch.org/*.

[98] *http://osrg.github.io/ryu/*.

[99] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-base comparison and selection of Software Defined Networking (SDN) controllers," *World Congress on Computer Applications and Information Systtems (WCCAIS) 2014,* 2014.

[100] *http://www.secdev.org/projects/scapy/*.

[101] *https://docs.python.org/3/whatsnew/3.4.html*.