Advances in Inter-domain Networking

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

Jie Li

December

2013

© 2013 Jie Li

APPROVAL SHEET

The dissertation

is submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

_____

AUTHOR

The dissertation has been read and approved by the examining committee:

Malathi Veeraraghavan

_____

Advisor

Ronald D. Williams

_____

Joanne Bechta Dugan

_____

Worthy Martin

_____

Martin Reisslein

_____

_____

Accepted for the School of Engineering and Applied Science:

Dean, School of Engineering and Applied Science

December

2013

Abstract

This dissertation describes advances made in support of two types of inter-domain network services: Scheduled Dynamic Circuit Service (SDCS) and IP datagram service. For SDCS, we developed a new reliable multicast transport protocol for distributing data to multiple receivers over virtual circuits. This protocol is called Virtual Circuit Multicast Transport Protocol (VCMTP). One of the key features of the VCMTP design is the ability to tradeoff file-transfer throughput for fast receivers with robustness (the percentage of successfully delivered files) for slow receivers using a configurable parameter called retransmission timeout factor. For a traffic load of 0.4, and a multicast group with 30 receivers, robustness increased significantly from 81.4 to 97.5% when the retransmission timeout factor was increased from 10 to 50. The corresponding drop in average throughput for fast receivers was small (86.9 to 85.8 Mbps). For IP-routed service, we designed and evaluated a clean-slate inter-domain routing and addressing architecture called Less-Is-More Architecture (LIMA), and showed how the solution can be adopted in today's IPv6 Internet. Noting that the current practices of using Provider-Independent (PI) addressing and propagating stub provider-aggregatable (PA) address sub-blocks to the default-free zone routers undermine the advantages of hierarchical addressing, LIMA proposes solutions that would allow multihomed stubs to use multiaddressing by obtaining PA address blocks, one from each of its providers. Solutions for address renumbering when stubs change providers, and for handling failures of access links without loss of reachability are proposed. An analysis of the current Internet BGP RIBs showed that the current global routing table size of 450K prefix entries would be reduced to about 24K entries if this solution was adopted by all stubs.

Acknowledgments

I want to give my sincerest gratitude to my advisor, Professor Malathi Veeraraghavan, who has given me unconditional support throughout the years. Without her guidance and patience it would be impossible for me to accomplish this achievement. I have learned many invaluable lessons from her passion for research and education, her curiosity for knowledge, and her great sense of responsibility for both her students and the projects in which she is involved. She was not only a knowledgeable advisor for my research, but also a great advisor for my life.

I also want to thank Professor Martin Reisslein, Steven Emmerson, and Professor Robert Russell for their contributions and collaboration through multiple phases of this work.

I thank Professor Ronald D. Williams, Professor Joanne Bechta Dugan, Professor Worthy Martin, Professor Martin Reisslein, and Professor Kamin Whitehouse for serving on my proposal and defense committee, and for providing insight on how to improve the depth and breadth of this work.

I would like to thank other graduate students in our research group, Zhenzhen Yan, Zhengyang Liu, Tian Jin, and Zhe Song, for always being kind and helpful. I enjoyed the life we spent together in our lab.

I want to thank my wife Zhenzhen Yan, my parents Wenmiao Li and Yuqing Zhu, and my wife's parents Qingxu Yan and Ming Gao for their love and support along the way. You are the greatest fortune I have in my life.

Finally, this work was carried out under the sponsorship of NSF OCI-1038058, OCI-1127340, CNS-1116081, ACI-1340910, and DOE DE-SC0002350 and DE-SC0007341 grants. I thank the National Science Foundation and Department of Energy for funding this research.

Contents

List of Tables

List of Figures

List of Abbreviations

AL	Application-Layer
ALC	Asynchronous Layered Coding
API	Application Programming Interface
AS	Autonomous System
BGP	Border Gateway Protocol
BOF	Begin-of-File
CDN	Content Delivery Networks
DFZ	Default-Free Zone
DHCP	Dynamic Host Configuration Protocol
DYNES	Dynamic Network System
EEM	Emulab Experiment Manager
EOF	End-of-File
FEC	Forward Error Correction
FIB	Forwarding Information Base
FSM	Finite State Machine
GRT	Global Routing Table
GUI	Graphical User Interface
IAB	Internet Architecture Board
IB	InfiniBand
IDC	Inter-Domain Controller
IDCP	Inter-Domain Controller Protocol
IDD	Internet Data Distribution (IDD)
IFID	Interface ID

IGMP	Internet Group Management Protocol
ION	Interoperable On-demand Network
IoT	Internet of Things
ISP	Internet Service Provider
ITMP	Inter-domain Tunnel Management Protocol
LDM	Local Data Manager
LIMA	Less-Is-More Architecture
LISP	Locator/Identifier Separation Protocol
MAST	Multi-Addressing with Stub Tunnels
MED	Multi-Exit Discriminator
MLD	Multicast Listener Discovery
MPLS	Multiprotocol Label Switching
mRCS	MAST Router Configuration System
MSDP	Multicast Source Discovery Protocol
NACK	Negative Acknowledgement
NBS	Name Based Sockets
NORM	NACK-Oriented Reliable Multicast
OFED	OpenFabrics Enterprise Development
OSCARS	On-demand Secure Circuits and Advance Reservation System
OSPF	Open Shortest Path First
OTN	Optical Transport Network
PA	Provider-Aggregatable
PI	Provider-Independent
PIM-SM	Protocol Independent Multicast Sparse Mode
POTS	Plain Old Telephone Service
RAMP	Reliable Adaptive Multicast Protocol
RBUDP	Reliable Blast UDP
RDMA	Remote Direct Memory Access
REN	Research-and-Education Network
RIB	Routing Information Base

RMTP	Reliable Multicast Transport Protocol
RoCE	RDMA over Converged Ethernet
RTT	Round Trip Time
SDCS	Scheduled Dynamic Circuit Service
SDL	Specification and Description Language
SLAAC	Stateless Address Auto Configuration
SONET	Synchronous Optical Network
SRM	Scalable Reliable Multicast
SRMP	Stub Reachability Management Protocol
SRMS	Stub Reachability Management System
TMS	Tunnel Management System
TTL	Time-To-Live
UCAR	University Corporation for Atmospheric Research
UML	Unified Modeling Language
VC	Virtual Circuit
VCMTPv1	Virtual Circuit Multicast Transport Protocol version 1
VCMTPv2	Virtual Circuit Multicast Transport Protocol version 2
VLAN	Virtual Local Area Network
VoIP	Voice over IP
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network
VSA	VCMTP Service Agent
VSM	VCMTP Service Manager

Chapter 1

Introduction

1.1 Background

Switched networks consists of endpoints and switches interconnected by links. Table 1.1 shows a classification of switches. Starting with the top left quadrant, consider **circuit switches**. The multiplexing scheme used in the data plane is based on "position," i.e., space (port), time and/or wavelength. The "position" of an incoming data frame is used to determine how to switch the frame to an appropriate outgoing link. When position-based multiplexing is used in the data plane, admission control is mandatory in the control plane. This is because specific positions have to be selected on each link of the end-to-end path for each multiplexed flow, and switches need to be configured with incoming-to-outgoing position mapping information before user data can be transmitted. This explains why the bottom left quadrant in Table 1.1 is not an option. In packet switches, incoming packets are demultiplexed and switched to appropriate outgoing links based on information carried in the packet headers. These switches can be operated with or without admission control. The former are referred to as **virtual-circuit (VC) switches** and the latter as **connectionless packet switches**.

Different types of switched networks can be interconnected to form an *internetwork*, which allows for communications between endpoints on different networks. For example, a smartphone in a cellular network can access a web server in an enterprise network. It is common for a single organization to operate an internetwork within its own domain

Table 1.1: Types of switches

Multiplexing / Admission control	**Circuit multiplexing - position based (port, time, lambda)**	**Packet multiplexing - header based)**
Connection-oriented (admission control)	Circuit switch (e.g., telephone, SONET, WDM, SDM)	Virtual-circuit (VC) switch (e.g., MPLS)
Connectionless (no admission control)	Not an option	Connectionless packet switch (e.g., Ethernet)

(intra-domain internetworking). A large organization, such as University of Virginia, usually operates an internetwork that consists of both Wi-Fi and Ethernet-switched networks. On another dimension, networks/internetworks that are *owned and operated* by different organizations are interconnected to form larger internetworks (inter-domain internetworking). The Internet, for example, is the global *internetwork* that connects almost all organizational networks/internetworks across the world.

Three types of communication services are offered across inter-domain internetworks today: IP service, leased line and Virtual Private Network (VPN) services, and Plain Old Telephone Service (POTS). IP service runs on connectionless packet-switched networks, and is the basic service offered on today's Internet. Leased lines are provided for exclusive use, typically on a monthly or yearly basis, for communications between two fixed endpoints. Leased-line services are offered over circuit-switched networks or virtual-circuit networks. Finally, POTS is offered over an inter-domain 64 kbps circuit-switched internetwork.

Fig. 1.1 shows a comparison of these three communication services along a horizontal line, as proposed by Veeraraghavan et al. [1]. The measure that increases as one moves along the line from right-to-left is the per-allocation rate-duration product. For example, each "allocation" in an IP-routed service across an Ethernet network is a maximum of 1500 bytes (Maximum Transmission Unit of Ethernet is 1500 bytes). At the other extreme, even a relatively low-rate T1 (1.544 Mbps) six-month lease is an allocation of almost 3 TB. In POTS, since average call duration is 3 minutes, each allocation is about 1.4 MB.

From these three examples, we see that there are significant gaps in the service offerings from a rate-duration product per allocation perspective. In addition to these gaps, the only service at the high-end, leased-line service, is restrictive in that the two endpoints of a leased-line circuit have to be identified at the time of purchase. Therefore, Research-and-Education Network (REN) providers, e.g., Internet2 and Esnet4, and commercial providers,

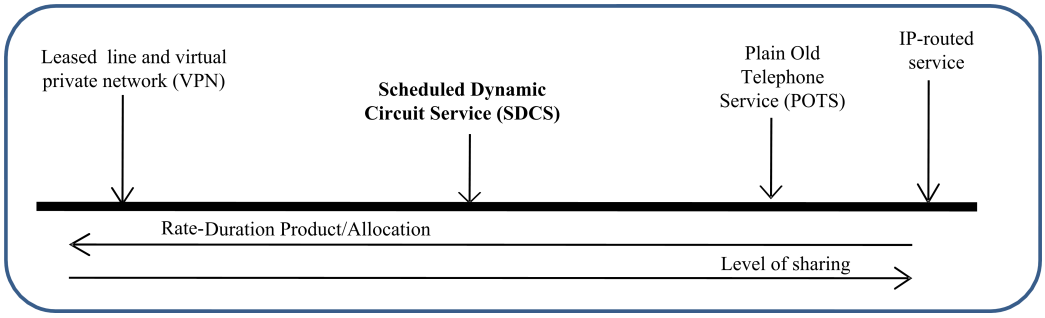

Figure 1.1: A spectrum of communication services [1]

e.g., AT&T and Verizon, have recently started to deploy a new type of communication service as a complement to their IP-routed and leased-line service offerings [1]. It is a Scheduled Dynamic Circuit Service (SDCS). Corporations, universities, research laboratories and other enterprises connect to their SDCS provider networks via SDCS-access links, and then on an as-needed basis place scheduling requests for a fixed-rate circuit for a fixed duration, for either immediate or future usage, to any other SDCS user. As Metcalfe noted, the value of a network service grows with the number of endpoints to which any single endpoint can connect [3]. Thus, SDCS addresses both the rate-duration product gap and connectivity issues.

The Dynamic Network System (DYNES) project led by Internet2 [4] and the ESnet On-demand Secure Circuits and Advance Reservation System (OSCARS) project [5] have led to deployments of SDCS on campus networks and ESnet respectively. Internet2 has also deployed an Interoperable On-demand Network (ION) in its backbone to let users "provision dedicated circuits across the Internet2 Network and other networks with dedicated circuit capabilities" [6]. More recently, several backbone RENs (Internet2 and ESnet in the US, GEANT2 in Europe, and Canarie in Canada) have been working on the specifications of a new protocol, called Inter-Domain Controller Protocol (IDCP), allowing centralized schedulers in different domains to communicate [7]. These deployments in SDCS have enabled an exciting new opportunity for networking research. New advances made to enhance/support SDCS could potentially have a significant impact.

A wide range of technologies are available to build virtual circuit networks. Multiprotocol Label Switching (MPLS) is a technology that is implemented within IP routers to offer a

virtual-circuit service in conjunction with IP-routed service on the same equipment. Ethernet Virtual Local Area Network (Ethernet VLAN) technology is based on the IEEE 802.1Q standard [8]. Ethernet switches can be configured to map incoming VLAN-tagged Ethernet frames to corresponding output ports. Examples of circuit networks are Synchronous Optical Network (SONET) and Optical Transport Network (OTN). These networks use time-division multiplexing, and support different sets of rates. Therefore, unlike the homogeneous 64 kbps network infrastructure used for POTS, SDCS can be based on different types of circuit/virtual circuit networks (i.e., networks that use different multiplexing schemes and protocols). Hence it is necessary to develop internetworking techniques to create a global SDCS that is offered across networks owned-and-operated by different organizations. The goal of IDCP and similar control-plane protocols [9] is to create such an inter-domain SDCS.

Of the four communication services shown in Fig. 1.1, the above description shows that SDCS is a growing inter-domain service. POTS, on the other hand, is being phased-out as telephony traffic is carried by Voice over IP (VoIP). Leased line and VPN services will continue to exist. The fourth service shown in Fig. 1.1 is IP-routed service. We briefly review this service next.

The Internet, on which IP-routed service is offered, consists of enterprise and residential networks interconnected by service provider networks. Enterprise networks (a.k.a. stub networks, as they only source or sink data) are owned and operated independently by corporations, universities, and government agencies. These networks are connected via "access links" into Internet Service Provider (ISP) networks. An enterprise or ISP network is regarded as an Autonomous System (AS), which is interconnected to other AS networks via border gateways, i.e., IP routers deployed at network boundaries. Each data packet carries a destination address that is used in a *routing table* lookup to determine the next-hop IP router. These routing tables are created automatically and updated by distributed routing protocols. For example, the Border Gateway Protocol (BGP) is an inter-domain routing protocol, while Open Shortest Path First (OSPF) is an intra-domain routing protocol for exchanging routing information within an AS.

SDCS and IP-routed services are the two topics of interest in this dissertation.

1.2 Problem Statement and Motivation

1.2.1 Increasing the adoption of SDCS

The current SDCS deployment is limited to a small number of domains. Besides Internet2 and ESnet, as of October 2013, 40 universities and regional RENs have deployed SDCS as part of the Internet2 DYNES project. There is little motivation to modify existing applications and/or develop new applications for SDCS given its limited deployment. On the other hand, without a significant growth in the number of applications that utilize SDCS, there is little justification for deploying new SDCS networks. This situation is comparable to the "chicken-and-egg" problem. It requires an initial investment to break this deadlock. The NSF Division of Advanced Cyberinfrastructure (ACI) has made such investments, e.g, the DYNES project for deploying SDCS, and our UVA project (NSF grant OCI-1038058) to create applications for SDCS.

The work presented in this dissertation was funded by this NSF ACI grant. We started with a case study of an existing scientific data distribution application called Internet Data Distribution (IDD) [10]. The IDD project was developed by the University Corporation for Atmospheric Research (UCAR) to distribute large amounts of meteorology data on a near real-time basis to a subscriber base of 170 institutions. Over 30 types of data products (referred to as feedtypes) are distributed through the IDD project. Currently the IDD project uses Application-Layer (AL) multicasting by creating a tree of Local Data Manager (LDM) servers (LDM is the application software used in the IDD project). For example, the LDM tree used to distribute CONDUIT high-resolution model data consists of 163 servers, with 22 root, 35 middle, and 106 leaf nodes. Unicast TCP connections are used between all upstream and downstream LDM servers.

While this method of using unicast TCP connections over IP-routed service has the ease-of-use advantage, it has certain disadvantages when compared to multicast delivery. For the same performance metric (e.g., latency), the unicast TCP approach will require more servers at the sender and a higher access-link bandwidth than the multicast approach. Alternately, for the same number of servers and access-link bandwidth, the latency of delivering data products could be smaller.

To avoid the costs of AL-multicasting, there has long been an interest in using IP-multicast, whereby network routers rather than application servers replicate packets for delivery to multiple receivers. However, native IP multicast has proven to be a challenge [11] [12] because of the complexity of distributed IP multicast routing protocols, such as Protocol Independent Multicast Sparse Mode (PIM-SM) [13] and Multicast Source Discovery Protocol (MSDP) [14], and because receivers without credentials can join a multicast group using Internet Group Management Protocol (IGMP) [15] and Multicast Listener Discovery (MLD) [16] accidentally or to maliciously undermine the throughput of the legitimate recipients. Also since the IP network is connectionless, congestion-related packet losses are possible. A congested path to any of one of the receivers can decrease throughput for all other receivers. These problems can be mitigated with a multicast virtual-circuit (VC) service [17] by leveraging the setup phase during which switches on the end-to-end path are configured for the VC. As a path is explicitly selected during VC setup, loop-free routes are ensured thus avoiding the problems of IP-multicast routing protocols. Since the network's VC scheduling/provisioning system needs to communicate with client software running on each receiver prior to data transfer, user credentials can be verified. Finally, since bandwidth and buffer resources are assigned to VCs at each switch in the setup phase, data-plane congestion is avoided.

Therefore, to bring the advantages of multicast and virtual circuits to scientific data distribution applications like IDD, we addressed the following problem: develop a reliable and scalable multicast transport protocol for virtual circuits.

1.2.2 Increasing inter-domain routing scalability

Global routing tables maintained by Tier-1 ISPs, which form the Default-Free Zone (DFZ) [1], have been "growing at an increasing and potentially alarming rate," as per a 2007 Internet Architecture Board (IAB) Workshop report [18]. Fig. 1.2 shows the growth of global routing table size since 1989. A contrarian view is espoused in [19], which states that the 17% exponential yearly growth rate is in step with improvements in memory technologies. The above argument notwithstanding, equipment vendors, such as Cisco, are leading IETF efforts,

[1]The DFZ routers are so called because they do not have a default route, but instead have reachability information for all addresses of the Internet. The routing table in these DFZ routers is referred to as the "global routing table."

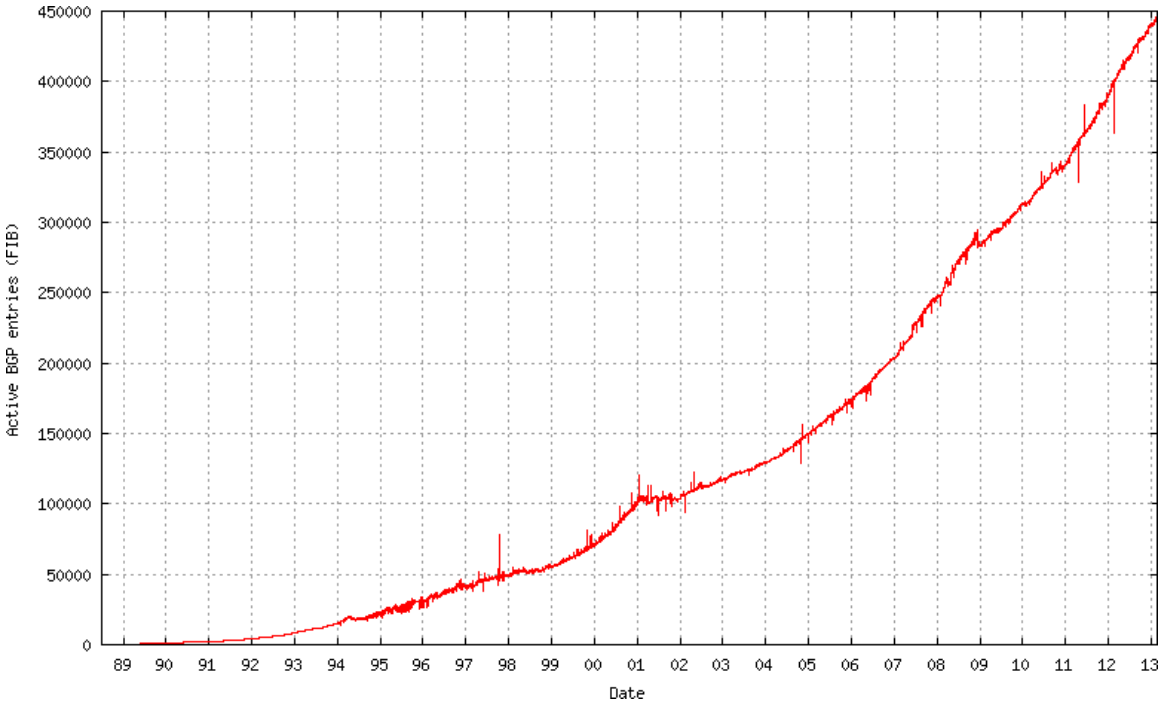

Figure 1.2: Global BGP Routing Table Growth (cited from BGP Reports [2])

such as Locator/Identifier Separation Protocol (LISP) [20], that address this very problem of global routing scalability. Whether or not this problem is solved in today's Internet, a future Internet design should avoid the conditions that lead to this rate of growth for two reasons. First, in the future Internet of Things (IoT), even home networks will be multihomed (i.e., connected to multiple providers) as the dependence on Internet reachability moves from desirable to obligatory, and multihoming is one of the causes of global routing table growth. Second, in the current environment of scaling back on resource consumption, whether for energy savings, or more broadly, sustainability, new designs for a future Internet of Things should keep the capital expenditures (router memory costs), and operating (administrative and power) expenditures low.

Multihoming has been identified as one of the reasons for the rapid growth in the size of the global routing table [21]. Provider-Independent (PI) addressing is another contributor to this growth. When a multihomed stub obtains a PI address block, it needs to be advertised to all the stub's provider networks for further propagation to the DFZ routers of the Tier-1 ISPs. In some cases, a multi-homed stub may obtain a Provider-Aggregatable (PA) address sub-block from the address block assigned to one of its providers. But the stub's longer PA

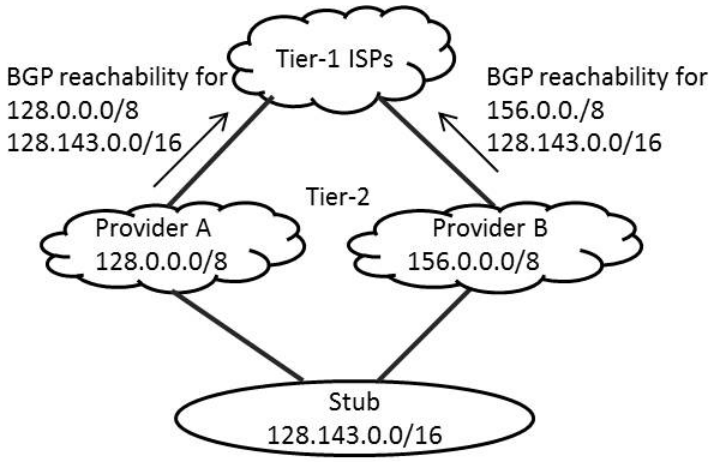

Figure 1.3: Example IPv4 multihoming

address sub-block needs to be advertised without aggregation to the DFZ routers by each of its providers for reasons explained with an example.

Fig. 1.3 shows a stub that obtains a PA address sub-block 128.143.0.0/16 from its provider A's address block 128.0.0.0/8. This 128.143.0.0/16 prefix needs to be advertised by Provider B in order for it to attract traffic destined for the stub via its network. However, Provider A also needs to advertise the 128.143.0.0/16 address block in addition to its aggregated address block 128.0.0.0/8 to the Tier-1 ISPs as shown in Fig. 1.3 because otherwise all traffic destined to the stub will flow via Provider B as a result of longest-prefix matching.

Both the use of PI addresses by stubs and the propagation of longer PA prefixes work against the scalability advantage of hierarchical addressing. PI addressing is effectively flat since it is (topological) location[2] independent. While there is a good reason for propagating longer PA prefixes of multi-homed stubs as illustrated by the above example, it nevertheless undermines the address aggregation advantage of hierarchical addressing.

Unlike flat Ethernet MAC addresses that are location independent, hierarchical addressing used in IPv4 networks is a key factor in the use of IP for large-scale networks such as the Internet. A disadvantage of hierarchical addressing is the operational cost incurred by administrators from having to configure address/subnet mask for each router interface and maintain Dynamic Host Configuration Protocol (DHCP) servers for the configuration of each host interface.

[2]The term "location" here is synonymous with topological location, not geographic.

While incurring these operational costs of hierarchical addressing, the current practices of using PI addressing and propagating stub PA address sub-blocks to the DFZ undermine the advantages of hierarchical addressing. Motivated by this need for increasing inter-domain routing scalability, the problem statement of this work is to define a new routing and addressing architecture for the future Internet.

1.3 Hypothesis formulation

As described in Section 1.2.1, towards creating applications for SDCS, we identified the meteorology data distribution project (called IDD) as potentially suitable for multicast virtual circuits. Correspondingly, the *first hypothesis* we formulated is as follows: the feedtypes distributed in the IDD project are received by tens to hundreds of receivers not millions (the number of receivers is important as it influences choices made in reliable multicast transport protocol design), and that feedtypes are almost continuous in their delivery of files (called data products). The latter phrase of the hypothesis is required to test whether the feedtypes are suitable for distribution over virtual circuits. Sending rates have to be almost constant if circuits are used, but if virtual circuits are used there can be some variation in sending rates, but not a significant variation. This is because switches can be configured to run scheduling algorithms for packet transmission on their egress interfaces that are work-conserving in nature, i.e., if a queue feeding a virtual circuit has a lower rate than the rate used for provisioning, packets from another queue will be served if the latter flow exceeds its expected rate.

For our next contribution, the design of a reliable multicast transport protocol, we formulated a *second hypothesis*: for the same performance metric (e.g., latency), using reliable multicast service will require fewer servers at the sender and a lower access-link bandwidth than using unicast TCP connections. Alternately, for the same number of servers and access-link bandwidth, the latency of delivering data products could be smaller.

A *third hypothesis* was formulated and tested for the Internet global routing scalability problem: it is feasible to adopt (i) an address assignment policy in which stubs (enterprises) are restricted to PA addressing, and (ii) a routing policy in which stub-level reachability

information is not propagated into the global routing tables, in conjunction with control-plane solutions to solve the address renumbering and multihoming problems created by this policy combination. Solutions for the address renumbering problem are offered for a future Internet clean-slate design, which we named a Less-Is-More Architecture (LIMA). A solution for the multihoming problem is presented in the context of an IPv6 based Internet. This solution is called Multi-Addressing with Stub Tunnels (MAST). The reason for switching our context from clean-slate designs to the IPv6 based Internet is to have a higher near-term impact.

1.4 Dissertation organization

The rest of the dissertation is organized into 6 chapters.

Chapter 2 presents a detailed analysis of two representative feed types distributed by IDD, i.e., CONDUIT and NEXRAD2. The purpose of the analysis is to test the first hypothesis presented in Section 1.3 about the distribution topology and file-arrival patterns of feed types. Our analysis showed that both feed types are received by about 150 receivers. Data products are sent almost continuously, especially in the case of NEXRAD2. Changes will be required to the VC structures as receivers change their subscriptions to feed types (necessitating dynamic control of the multicast VCs). Therefore, we concluded that a reliable multicast transport service over SDCS is suitable for the IDD application.

Chapter 3 describes our design, implementation, and evaluation of Virtual Circuit Multicast Transport Protocol version 1 (VCMTPv1). To our knowledge, this is the first reliable multicast transport protocol specifically designed for virtual circuits. The primary goal for VCMTPv1 was to minimize the negative performance impact of *slow receivers* (i.e., receivers that require retransmissions for blocks missed during the multicast) on the throughput experienced by *fast receivers* (i.e., receivers that do not require retransmissions). Consequently, a key design assumption in VCMTPv1 was to handle all data retransmissions after the file multicast so that a sender can utilize the complete network and CPU resources for high-speed multicasting. The performance of VCMTPv1 was evaluated with experiments to multicast a single large file (of sizes between 512 MB and 4 GB) to 7 receivers. This

design is useful if single files need to be multicast, i.e., interarrival times between files are long.

Chapter 4 presents our design, implementation, and evaluation of VCMTP version 2 (VCMTPv2). Motivated by the need for serving continuous file transfers as required in IDD, we changed some of the key design assumptions made in VCMTPv1. The concept of executing retransmissions after the file multicast, as assumed in VCMTPv1, has a limitation that it is only sustainable when file inter-arrival times are significantly longer than the service times required to transfer files. This assumption does not hold in IDD. In the VCMTPv2 design, retransmissions are executed in parallel with multicasts. Furthermore, multiple files may be served concurrently. Detailed Finite State Machines (FSMs) were defined for VCMTPv2. Large-scale multicast experiments for continuous file transfers (with up to 30 receivers) were conducted on the Emulab testbed [22] to evaluate the performance of VCMTPv2.

Chapter 5 presents our Less-Is-More Architecture (LIMA), a new clean-slate inter-domain routing and addressing architecture for a future Internet. In LIMA, we proposed the adoption of an addressing and routing policy combination that is different from the policies used in today's Internet. After presenting the new addressing and routing architecture, we describe the major software components in LIMA, including modified versions of the DNS and DHCP servers/clients. The policy combination proposed in LIMA creates an address renumbering problem. A solution for this problem is presented to ensure seamless (no loss of connectivity) transitions when a stub changes one of its providers. This address renumbering solution leverages multi-addressing and a novel concept called "dismembered addressing" proposed for LIMA. Other aspects impacted by the policy combination, such as multihoming and traffic engineering, are also discussed.

Chapter 6 proposes a detailed solution for the multihoming problem. However, instead of presenting this solution in the context of the clean-slate LIMA solution, we recognized that the policy combination proposed in Chapter 5 can be applied to an IPv6 based Internet. We named our solution Multi-Addressing with Stub Tunnels (MAST). By combining IPv6 multi-addressing with backup tunnels between a stub and each of its providers, stubs can enjoy the reliability advantages of multihoming without adding to the size of the global

routing tables. An analysis of the current Internet BGP Routing Information Bases (RIBs) was conducted. It showed that the current global routing table size of 450K prefix entries would be reduced to about 24K entries if the MAST solution was adopted by all stubs.

Chapter 7 summarizes our work, discusses potential future work, and concludes the dissertation.

1.5 Key contributions

The key contributions of this work are as follows.

1. We first characterized the traffic patterns of two meteorology data feedtypes distributed by IDD. The number of receivers are in the hundreds allowing for our VCMTP design to have unicast TCP connections for handling retransmissions of blocks missed during the multicast (in contrast, reliable transport protocols for multicasting to millions of receivers use forward error correction codes [23] and/or broadcast disk [24] approaches, both of which are less efficient than a scheme based on negative-ACKs and retransmissions). Also our characterization of the file-arrival patterns showed that virtual circuits are suitable for the IDD feedtypes. This work is published in a paper in the *International Conference on Communications, Mobility, and Computing (CMC2012)* [25].

2. We designed and implemented a new reliable multicast transport protocol for virtual circuits called VCMTPv1. A key design concept in VCMTPv1 is to execute the whole message multicast before handling retransmission requests. The advantage of this solution is that the throughput experienced by fast receivers is independent of the throughput experienced by slow receivers in the multicast group. However, separation of the multicast phase from the retransmission phase makes VCMTPv1 suitable only for single-file transfers, not for continuous file transfers. This work is published in a paper in the *International Conference on Communications, Mobility, and Computing (CMC2012)* [26].

3. We designed a new version called VCMTPv2 for applications with continuous file-transfer requirements. Unlike VCMTPv1, retransmission requests are handled in parallel with the execution of file multicasting in VCMTPv2. One of the key features of the VCMTPv2 design is the ability to tradeoff file-transfer throughput for fast receivers with robustness (the percentage of successfully delivered files) for slow receivers. An evaluation of VCMTPv2 in the context of a continuous file-arrival process was conducted. This work has been submitted in a paper to the *IEEE Transactions on Parallel and Distributed Systems (TPDS)* [27].

4. We designed a future internet architecture called LIMA in which per-packet data plane actions are kept minimal, while adding onus to handling relatively rare events such as address renumbering and access link failures. Analysis of BGP RIB data characterized the benefit of LIMA in reducing the global routing table size, and a cost of LIMA incurred when stubs change providers. This work is published in the *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS 2012)* [28].

5. We proposed MAST, a stub multihoming solution for IPv6 networks. MAST achieves the goal of global routing table growth rate reduction without requiring data-plane modifications to routers or end hosts. However, control-plane changes are necessary to adopt the MAST solution, and new stub network management systems are proposed to reduce administrative overhead. In addition to the overall architecture, other contributions include proposed additions to DNS and DHCPv6, and automated tunnel management. This work has been submitted in a paper to the *IEEE International Conference on Communications (ICC2014)* [29].

In addition, contributions were made to the following publication: A straw man proposal for future diverse internets with an Align-and-Decouple (AD) principle was published in a paper in the *Proceedings of 2011 IEEE Symposium on Computers and Communications (ISCC2011)* [30].

Chapter 2

Characterization of IDD feedtypes

2.1 Introduction

In this work, we analyzed real-time meteorology data distributed in the IDD project from the University Corporation for Atmospheric Research (UCAR) to over 160 institutions. The software used for this data distribution is called the Local Data Manager (LDM) [31]. Currently over 30 types of scientific data products (called *feedtypes* in IDD terminology) are distributed using LDM.

Our *findings* are as follows. Some of the feedtypes are such that new data products are created (e.g., from radar measurements) almost continuously. However, other feedtypes are such that data products appear in bursts, i.e., in some minutes the volume of data generated is orders of magnitude higher than in other minutes. The virtual circuit rate can be dynamically increased for the time durations when the volume of data generated is high. User requirements are for near real-time delivery. Furthermore, data products need to be disseminated from one upstream LDM server to 10s to 100s of receivers. Given the feedtype distribution topologies and the almost continuous nature of data product arrivals, multicast virtual circuits appear to be well suited for the IDD application.

Section 2.2 presents data analysis results for two representative feedtypes distributed by IDD. Section 2.3 discusses our conclusions based on the data analysis.

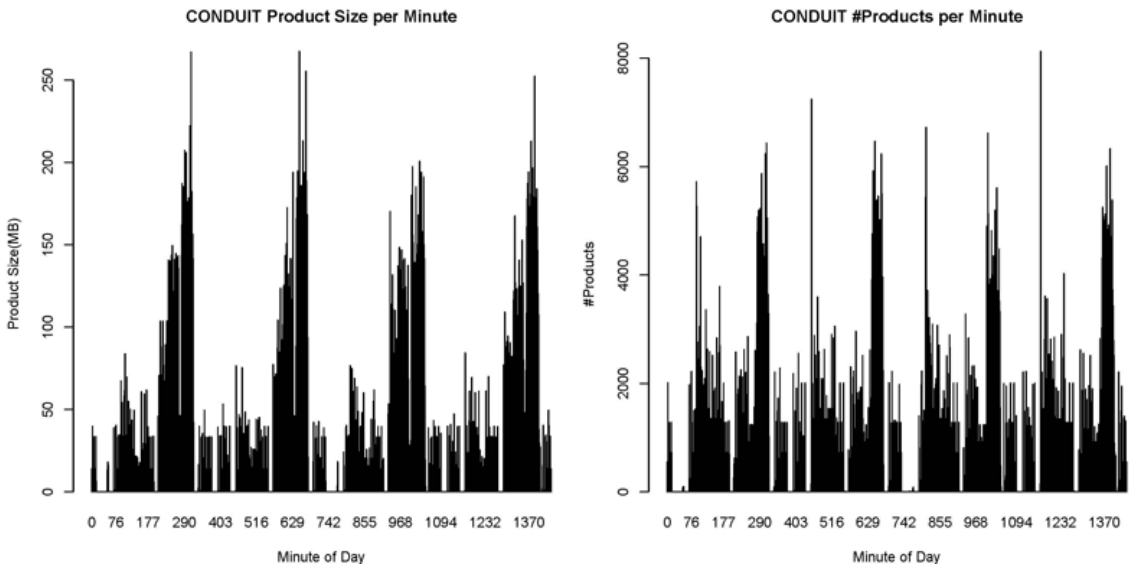

Figure 2.1: Daily characteristics of the CONDUIT data type

2.2 Data Analysis

A high-level review of the 30 feedtypes served by IDD from the real-time statistics site [32] shows that the CONDUIT and NEXRAD2 feedtypes are two representative "heavy-hitter" flows in terms of data volume and duration. Therefore, this section provides a detailed analysis of the data transfer patterns of the CONDUIT and NEXRAD2 feedtypes.

2.2.1 CONDUIT Feedtype Analysis

Single Flow Characteristics: To characterize the data transfer patterns of a single CONDUIT flow, we installed the LDM software (version 6.8.1) on one of our laboratory machines to receive the CONDUIT feedtype from one of the data servers located at the UCAR site (idd.unidata.ucar.edu). We ran LDM in verbose mode so that information about every single data product received was recorded in a local log file. The information collected in the log files was then parsed to analyze the characteristics of the CONDUIT data flow. Based on an analysis of 9 days of data (October 13, 2010-October 21, 2010), the results showed that the CONDUIT data flow has a relatively static daily transfer pattern. Fig. 2.1 shows a typical transfer pattern of a single CONDUIT data flow in one day. First, the total size of CONDUIT data generated and distributed to all receivers every day is about 60

GB, with a standard deviation (SD) of 0.3 GB. Second, the throughput varies significantly at different times of the day. The maximum throughput is about 250 MB per minute (an average of 33.3 Mbps) with an SD of 28.8 MB, while the minimum throughput is 0 MB per minute. Furthermore, there are four periods during a day, separated approximately by 6 hours, when throughput is at its peak level. Third, the number of data products generated per minute is large although most data products are small in size. The per-day average number of data products is 1.6 million (SD: 0.01 million), with up to 8000 data products (SD: 1054) transferred per minute during peak flow periods. The per-day average size of all data products is about 38 KB (SD:0.2 KB).

Data Distribution Topology: The next step is to analyze the data feed topology, or *Feed Trees* in IDD terminology, which shows the sender-receiver relationships. The hierarchical feed tree topology for each feedtype can be obtained from the IDD real-time statistics Web site [32]. We downloaded and parsed the real-time topology data for CONDUIT. The results are shown in Table 2.1. The real-time feed tree is a dynamic topology in that hosts join and leave the tree. The results in this table were parsed from the feed tree information obtained on April 15, 2011. The CONDUIT topology consisted of a total of 163 LDM nodes: 22 root nodes (7 at UCAR), 35 middle nodes (which run both downstream and upstream LDM servers), and 106 leaf nodes. The UCAR root servers together had a fanout of 104[1], which was the maximum value in the CONDUIT feed tree. Since LDM creates a separate TCP connection for each sender-receiver pair, the peak bandwidth requirement for UCAR's access link is 104×250 MB/minute, or about 3.5 Gbps just for the CONDUIT data distribution. Fig. 2.2 shows a visualized topology map which we downloaded from the IDD real-time statistics Web site. The central point shown in Colorado (with the maximum fanout) is the IDD site at UCAR.

Silence Periods: The results show that only a small percentage (less than 2%) of the time intervals between the transfer of two consecutive data products are larger than 1 second. Per day, there are only about 100 silence periods that last longer than 60 seconds out of a total of 1.6 million silence periods. Of these 100, about 15 are larger than 300 seconds. The maximum silence period observed across all 9 days of traffic was 35 minutes.

[1]Fan-out is the number of data receivers connected to the same site.

Table 2.1: CONDUIT Feed Tree Topology Information

Parameter	Number
Total Number of Distinct Nodes	163
Number of Root Nodes	22
Number of Middle Nodes	35
Number of Leaf Nodes	106
Maximum Fan-out Number	104

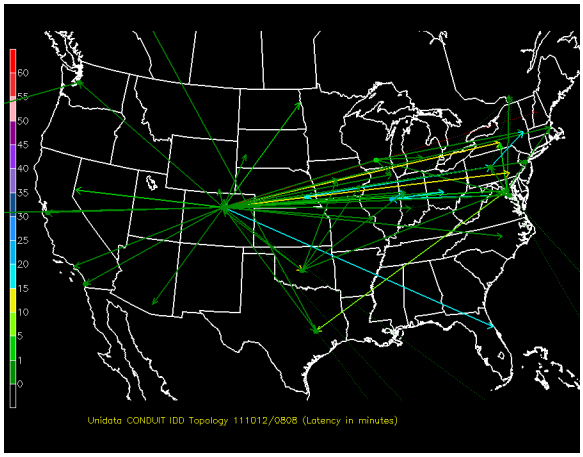

Figure 2.2: CONDUIT Distribution Topology Map

2.2.2 NEXRAD2 Feedtype Analysis

Single Flow Characteristics: We collected information about the NEXRAD2 feedtype using the same methodology described for the CONDUIT feedtype. Data was collected for the period Sept. 15, 2011 - Sept. 21, 2011. As with the CONDUIT feedtype, the data generation and transfer patterns of NEXRAD2 shows little variation from one day to the next. Fig. 2.3 shows the per-minute throughput and number of products for Day 7 (Sept. 21, 2011). The per-day average data size is 56.80 GB, with a standard deviation of 6.56 GB. Unlike CONDUIT, the variability of the NEXRAD2 throughput is relatively small. The per-minute average throughput varies from 4.0 Mbps to 7.8 Mbps.

Data Distribution Topology: The data distribution topology for the NEXRAD2 feedtype on a typical day is shown in Table 2.2. The NEXRAD2 feed tree had a total of 150 LDM servers: 36 root nodes, 40 middle nodes, and 74 leaf nodes with a maximum fanout of 55 from the UCAR LDM servers. Given the observed maximum throughput of 7.8 Mbps, this creates a peak bandwidth requirement of 55 $\times$ 7.8 Mbps, or about 429 Mbps, for the NEXRAD2 feedtype.

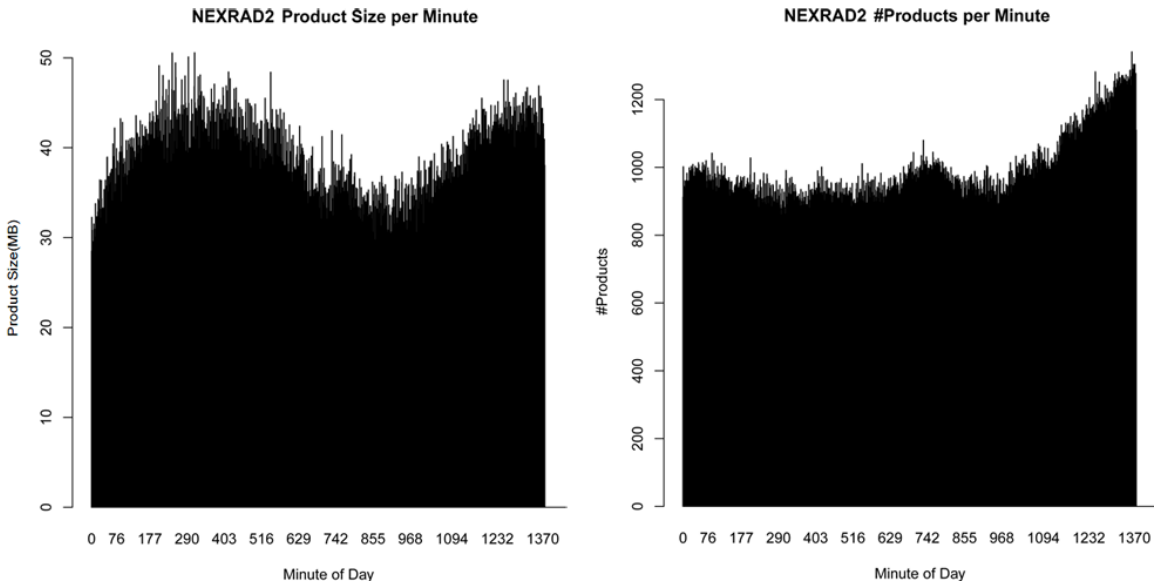

Figure 2.3: Daily characteristics of a single NEXRAD2 data flow

Silence Periods: There was only 1 silence period per day that had a duration longer than 300 seconds. This period occurred at the end of the day and lasted 5250 seconds. There were a total of about 81240 silence periods per day. Almost all of these silence periods were less than 1 second.

Table 2.2: NEXRAD2 Feed Tree Topology Information

Parameter	**Number**
Total Number of Distinct Nodes	150
Number of Root Nodes	36
Number of Middle Nodes	40
Number of Leaf Nodes	74
Maximum Fan-out Number	55

2.3 Conclusions

We conducted a detailed analysis of data-distribution topologies and file-arrival patterns for two representative feedtypes, i.e., CONDUIT and NEXRAD2. We determined that the numbers of receivers for both feedtypes are on the order of hundreds. As described in Section 1.5, with hundreds of receivers it is feasible to use a scheme based on receivers providing the sender negative ACKs when blocks are missed in the multicast (since virtual circuits provide in-sequence delivery, a missing block is easily identified using sequence

numbers). The sender can respond to negative ACKs with retransmissions. Such a scheme is not feasible if the number of receivers is in the millions.

Next, our analysis showed that both feed types had almost continuous file arrivals. In CONDUIT, the rate was not constant. To support CONDUIT on virtual circuits, dynamic rate adjustment is required. SDCS supports such rate adjustments.

In summary, the analysis showed that multicast virtual circuits are suitable for IDD feedtypes. Since the data has to be delivered reliably (i.e., without errors, losses or duplicates), we designed, implemented and evaluated a reliable multicast transport protocol for virtual circuits. Two versions of this protocol are presented in the next two chapters.

Chapter 3

VCMTPv1: A Reliable Virtual Circuit Multicast Transport Protocol

3.1 Introduction

This chapter presents the design, implementation, and evaluation of Virtual Circuit Multicast Transport Protocol version 1 (VCMTPv1). As mentioned in Section 1.4, the primary goal for this first version of VCMTP was to minimize the negative performance impact of slow receivers on the throughput experienced by fast receivers in a multicast group.

VCMTPv1 is *a multicast transport protocol for reliable message delivery over virtual circuits*. First consider the *multicast* aspect. If a sender can simultaneously send a message to multiple receivers, the server capacity and link bandwidth required can be lowered. Second consider the term "reliable message transfer." Reliability is required as this protocol is designed for data distribution, and not audio/video transmission. The reason for using the term "message" is that individual files, or data products in IDD terminology, can be regarded as messages. Contrast message transport service as offered by UDP with byte stream transport service offered by TCP. Our requirement is to transfer messages reliably to large numbers of receivers, not byte streams.

Finally, consider the proposed usage of virtual circuits (VC). A major reason for choosing VC for multicast service is that it shifts the burden of congestion control to the control plane. VC networks have a connection setup phase in which requests for bandwidth are accepted/rejected based on resource availability. As long as the sending rates are limited to the VC rates, there is no possibility of congestion in the data plane. This simplifies the reliable multicast transport protocol problem to just handling bit errors, and flow control problems (receiver buffer overflows). Other reliable multicast proposals, such as the Scalable Reliable Multicast (SRM) framework [33], note that multicast congestion control, which is required if the underlying network service is a connectionless service such as the IP-routed service, is a difficult proposition. For example, should the sending rate be tuned to that of the worst-case receiver? One approach proposed in SRM is to use reserved resources as with the Integrated Services (IntServ) architecture [34].

The main design concept of VCMTPv1 is to transmit the whole message first and then handle retransmissions. This is coupled with the concept of bandwidth adaptation as proposed in [35], whereby a multitasking receiver could change multicast groups to receive data on a lower-rate VC if it experiences receive buffer overflows. By controlling membership of receivers and using multiple (different-rate) virtual circuits with the sending rates tuned to the VC rates, only few retransmissions are likely to be required. These are handled at the end to avoid one or more slow receivers from slowing down data reception for the other receivers.

The remainder of this chapter is organized as follows. Section 3.2 discusses the overall design of VCMTPv1, and Section 3.3 presents a prototype implementation. Section 3.4 briefly describes the University of Utah Emulab testbed, which was used to evaluate VCMTPv1, and a system monitoring tool for managing the Emulab experiment remotely from a University of Virginia (UVa) host. Section 3.5 presents our experimental settings and the evaluation results with the prototype. Section 3.6 provides a brief overview of related work on reliable multicast transport protocols. Section 3.7 concludes this chapter.

For ease of presentation, in the rest of this chapter, we will use the term “VCMTP” instead of “VCMTPv1”.

3.2 Design of VCMTP

Two objectives in designing VCMTP are reliability and scalability. As with other reliable transport protocols such as TCP, VCMTP includes error control and flow control functions. Congestion control is not required in the data plane because congestion is handled in the control plane during VC setup (VC setup requests will be rejected if all bandwidth resources are used up for existing virtual circuits). For *error control*, to avoid senders having to maintain state information about every receiver, retransmission requests are receiver driven through Negative Acknowledgements (NACKs). This avoids the (positive) acknowledgement implosion problem in which the sender host is overloaded by acknowledgement messages from a large number of receivers. For packet retransmissions to individual receivers, VCMTP uses unicast TCP connections over IP-routed paths. To achieve scalability, unlike in TCP in which packet retransmissions are interleaved with the main data transfer process, the VCMTP sender executes retransmissions only after the whole message is multicast to all receivers. When receivers detect packet loss during the main data transfer period, the receivers immediately send retransmission requests to the sender, but the sender stores the requests and handles them at the end. The sender unicasts retransmissions to each receiver on the individual TCP connections. To leverage caching, the retransmission requests will be handled on a block-by-block basis, where a *block* is a VCMTP unit of data. This approach prevents the retransmission process from slowing down the main data multicasting process at the sender, which can be a serious problem when the number of *slow receivers* (i.e., receivers that experience data losses during the multicast) is large. Only those receivers that have packet loss will experience a reduction in the overall throughput (because of the retransmissions at the end), while other receivers that can keep up with the sending rate will experience higher throughput.

As the sender receives TCP connect requests, it sends a control message on the TCP connections informing each receiver of all available VCs with their corresponding rates. If a receiver experiences a high packet loss rate, a VCMTP multicast manager can have it switch to receiving data on a lower-rate VC if there is one. Such slow-downs can occur at the receiver because of other competing tasks running in the same host as the VCMTP process.

For *flow control*, there is choice of three mechanisms: window-based, rate-based, and on/off [36]. Window-based flow control is not suitable for multicast because the free-space available in the receive buffer will differ from one receiver to the next. Rate-based flow control allows a sender to adjust its sending rate dynamically. Such a scheme would again be difficult to implement in a multicast scenario as each receiver's receive rate can vary dynamically due to multitasking. The on/off mechanism allows a receiver to send an on/off message to the sender based on its buffer occupancy. The sender will start/stop sending data as per these control messages. Again, this scheme will slow down the reception rate for the fast receivers.

The solution to the flow control proposed for VCMTP is as follows. First, the sending rates of multiple VCs used to transmit the same data are sent to all receivers in the control plane, allowing each receiver to choose the multicast group corresponding to the VC whose rate is less than or equal to the rate it can sustain. Measurements are required at the receiver to have it continually evaluate whether its multicast receiving process is able to keep up with the selected rate. If it finds itself losing too many packets from its receive buffer, it will switch to a lower-rate VC if one is available. In spite of this arrangement, packets are still likely to be lost due to multitasking, and these losses are handled with NACKs and retransmissions.

3.3 VCMTP prototype

Fig. 3.1 illustrates how an Ethernet VLAN based virtual circuit is provisioned between a sender and multiple receivers. Tagged VLANs with rate policing and scheduling are used to create rate-guaranteed virtual circuits. Other VC technologies such as MPLS can also be used. VLANs are used here just to illustrate the concept. These virtual circuits are reserved and provisioned by schedulers such as the ESnet On-Demand Secure Circuits and Advance Reservation System (OSCARS) [5]. Class-D multicast IP addresses in the (224.0.0.0 to 239.0.0.0)/8 range will be assigned, one per multipoint VLAN. Each receiver binds a UDP socket to this address. As shown in Fig. 3.1, the multicast forwarding action is performed in

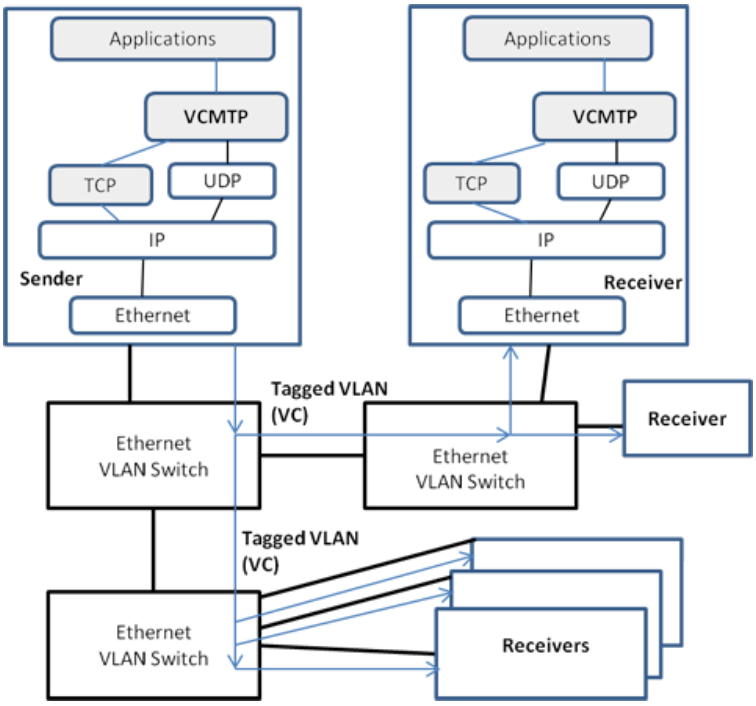

Figure 3.1: An example configuration

the Ethernet switches on the VLAN ID and multicast MAC address, which is derived from the Class D multicast IP address.

The inner blocks shown with in the sender and a receiver in Fig. 3.1 illustrate the concept that VCMTP uses UDP for the multicast transmission of the message (the arrows show the flow of this transmission), and TCP connections to send/receive NACKs and retransmissions. VCMTP adopts a block-oriented data transfer model in which a message (disk file or memory data) on the sender is first fragmented into multiple blocks (VCMTP packets), which are then encapsulated into UDP datagrams for multicasting. Upon receiving a multicast packet, the receiver process extracts the VCMTP packet from the UDP datagram, processes the VCMTP packet header, and then writes the payload to the corresponding data block given its position in the message. Unlike the byte streaming transfer model, the block-oriented data transfer model does not require data to be passed to the application in sequence. However, it requires that receivers be aware of the length of the message before the transfer, so that they can allocate the data storage (either on disk or in memory) correspondingly. In VCMTP, this information is communicated between the sender and all the receivers over the TCP connections before the message is multicast.

3.3.1 Sender Implementation

The VCMTP sender is implemented as a user-space library. The `VCMTPSender` class provides a set of APIs that are related to both control-plane functions, such as initializing a VCMTP multicast group, and data-plane functions, such as sending data to a multicast group. The application issues a VCMTP `JoinGroup` function call specifying a Class D IP address, allowing the VCMTP code to open a UDP socket using that IP address. The VCMTP code also starts a `VCMTP retransmission thread`, whose functions are to listen for TCP connection requests from multicast receivers, and to handle NACKs and retransmissions.

To multicast a message, the application can call one of two VCMTP functions in the `VCMTPSender` class:

1. *int VCMTPSender::Send(void* buffer, size_t length)*
2. *int VCMTPSender::Send(char* file_name)*

The first API is used for transferring in-memory data, and the second API is used for transferring disk files. For the second call, the VCMTP send function reads directly from the disk. In both functions, the sender first determines the size of the message and communicates this information to all the receivers that are connected to it via TCP connections. It then divides the data into blocks that can fit into the payload of a VCMTP packet. The VCMTP header includes a source port number, destination port number, block sequence number, and payload length. The VCMTP `send` function writes the VCMTP packets to the UDP socket.

The `select()` system call is used in the VCMTP retransmission thread for handling unicast TCP connections from all receivers. As it receives NACKs, it stores the block number along with the receiver TCP socket ID in a memory buffer called a *retransmission store*. Retransmissions are executed at the end of the message multicast on a block-by-block basis with a block being retransmitted to all receivers that reported it missing. This design allows the sender to leverage caching. For messages stored on disk that are larger than the system file cache size on the sender, disk reads are required with a specific offset for each block that needs retransmission. This design has a disadvantage when compared to the TCP approach in which data read from disks are held in a retransmission buffer in memory and retransmissions are completed immediately after loss detection.

3.3.2 Receiver Implementation

The receiving application calls a `VCMTP Receiver` method with a set of class D IP addresses, allowing the latter to bind one UDP socket for each Class D IP addresses corresponding to each of the virtual circuits (recall the concept of using different-rate circuits to handle the flow control problem). After joining a multicast group, the VCMTP Receiver also starts a `VCMTP retransmission thread`, which in turn sets up a unicast TCP connection to the VCMTP sender for sending NACKs and receiving retransmissions. The main VCMTP Receiver thread then starts receiving multicast data from the sender over the UDP socket. Upon receiving a UDP datagram carrying a VCMTP packet, it compares the current received block sequence number with the last received block sequence number to determine if there is a missing packet. If a packet loss is detected, the receiver stores the block number in a retransmission store buffer and sends it in a NACK to the VCMTP sender. For the correctly received VCMTP packet, it extracts the payload and copies it to the specified application data block, which involves a disk write for a file transfer. The VCMTP retransmission thread handles all retransmissions received on the TCP connections, and likewise writes the payloads into the application data block.

3.3.3 VCMTP Multicast Manager

Besides the VCMTP sender and receivers, a separate component called the *VCMTP Multicast Manager* is implemented to support a set of management-plane functions, such as performance monitoring, fault management, and configuration management. The performance monitoring module keeps track of the retransmission rates of receivers, and initiates a receiver's switch to a lower-rate VC when needed. The fault management module detects exception conditions during the message multicast, and initiates recovery. The configuration management handles the setup of VCs through circuit schedulers such as OSCARS.

3.4 The Emulab Experiment Manager

We evaluated our VCMTP prototype on the Emulab testbed [22]. Emulab is a network emulation testbed managed by the University of Utah and open to the research community.

It allows users to construct arbitrary network topologies for their experiments. Hosts and switches/routers in the testbed can be interconnected according to user specifications. Computing and networking resources are provided to each user in isolation of the resources used to other users through virtualization techniques. The testbed provides a variety of machine types, operating systems, and link rates to meet different requirements.

A challenge arises when running experiments that involve a large number of hosts. Each host in Emulab can be accessed and managed individually via SSH or remote desktop from across the Internet, but this approach does not scale for experiments involving a large number of hosts. To make it easier to manage a large number of hosts, we have developed a software package called Emulab Experiment Manager (EEM). The EEM provides a centralized management console for hosts in a distributed system. It is a Windows-based GUI that can be executed on any Windows machine with a public IP address. Fig. 3.2 shows the main management window of the EEM. When initiated, the EEM opens a TCP socket and listens for connection requests from client hosts (in this case, hosts in Emulab). Upon receiving a request, the EEM establishes a TCP connection to the host, and creates a sub-window in the management console corresponding to that host. This per-client sub-window allows a user to issue either system commands or user-defined commands (specific to a running application) on the corresponding client. The commands will be sent to the client host for execution, and the standard output will be sent back to the EEM for display on the sub-window. A user can thus control multiple distributed hosts from a single EEM management window.

3.5 Evaluation

For the evaluation of the VCMTP prototype, we chose hosts with 1 Gb/s Ethernet NICs to execute relatively high-speed experiments up to 800 Mbps. The nodes have 2 GB memory and commodity disk facilities. Section 3.5.1 describes a multicast experiment that illustrates a positive aspect of the VCMTP design, while Section 3.5.2 describes a unicast experiment that demonstrates a negative feature.

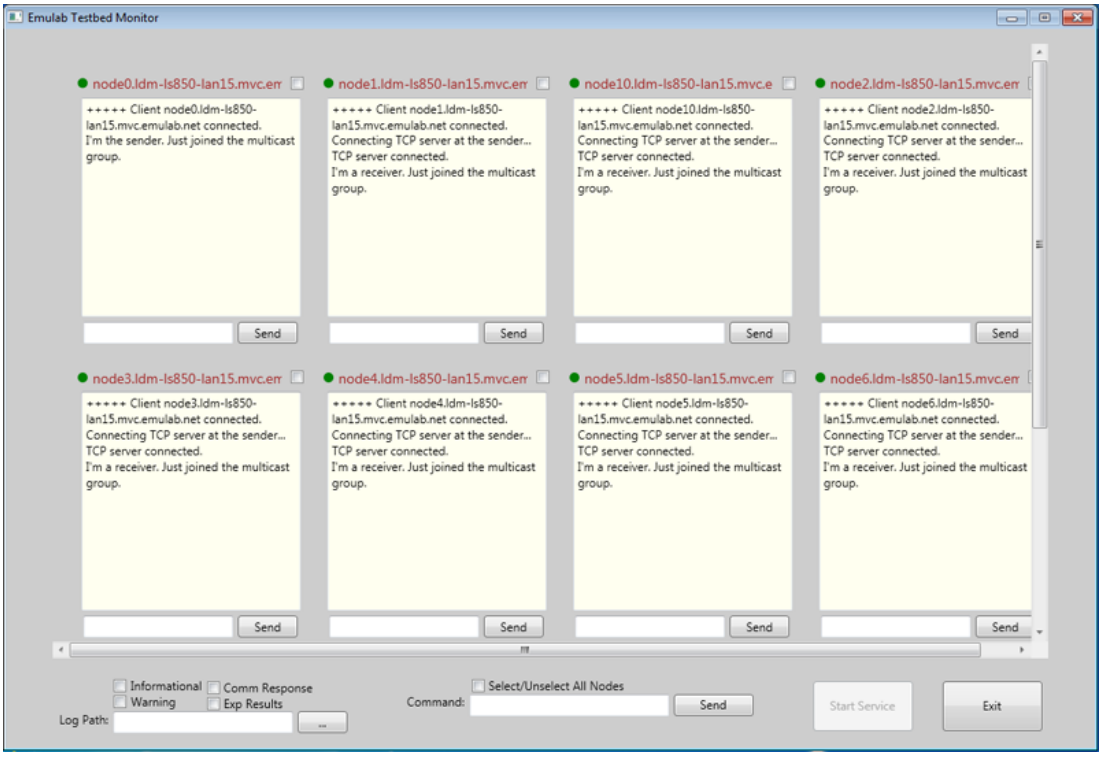

Figure 3.2: The Emulab Experiment Manager

3.5.1 Multicast Performance

In this experiment, one VCMTP sender multicasts disk files of different sizes to 7 receiver nodes. The sending rate is set to 600 Mbps, a rate at which there are only few receive buffer overflows, if any. For each file size, the experiment was repeated 10 times, and the average throughput was computed. The results are shown in Table 3.1. For the 512 MB file size, there were no retransmissions in all 70 receptions (7 receivers and 10 runs). For the remaining file sizes, there were some receptions in which retransmissions were required. The average throughput of the receptions without any packet losses is higher than the average throughput of receptions with losses. Thus in a single multicast, if 6 out of the 7 receivers experienced no packet losses, their (higher) throughput was unaffected by the losses incurred in the 7^{th} receiver. This illustrates the key design concept of VCMTP that allows for scalability.

Table 3.1: VCMTP Multicast Throughput (Unit: Mbps)

	512 MB	**1 GB**	**2 GB**	**4 GB**
Avg. (SD) throughput for receptions without retransmissions	579.49 (1.73)	574.75 (1.54)	588.26 (0.64)	582.19 (0.91)
Avg. (SD) throughput for receptions with retransmissions	N/A	561.40 (1.73)	580.32 (4.94)	576.10 (4.43)

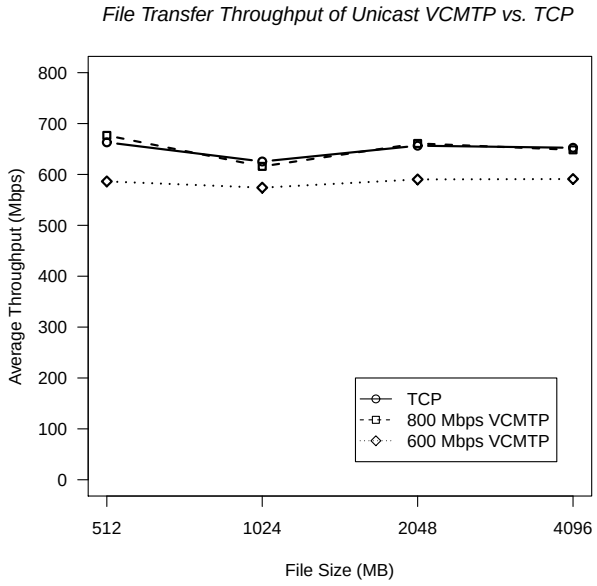

Figure 3.3: Unicast File Transfer Throughput for VCMTP vs. TCP

3.5.2 Choice of sending rate

This experiment compares unicast TCP with a one sender/one-receiver VCMTP configuration to illustrate the need for VCMTP receivers to choose a VC whose rate matches the rate at which the VCMTP application can deplete the receive buffer. Disk-to-disk file transfers with different sizes ranging from 512 MB to 4 GB were executed and measurements obtained. Two different sending rates, 600 Mbps and 800 Mbps, were used. For each file size and each experimental setting, 10 runs were executed, and the average throughput computed from the measurements. The results are shown in Fig. 3.3.

With TCP, the sending rate is automatically determined by the sender as part of the Slow Start/Congestion Avoidance algorithms. As shown in the figure, TCP achieves a

throughput of around 650 Mbps for all the file sizes considered. With VCMTP, there is a greater likelihood of receive buffer overflows at 800 Mbps than at 600 Mbps. The average retransmission rates for 512 MB, 1GB, 2 GB, and 4 GB file transfers are 4.46%, 11.04%, 8.63%, and 9.27% at 800 Mbps, and 0%, 0.26%, 0.07%, and 0.19% at 600 Mbps respectively. The 800 Mbps VCMTP transfer achieves the same throughput as TCP throughput as seen in Fig. 3.3, but the more cautious setting of 600 Mbps yields close to that throughput but with few retransmissions if any.

3.6 Related Work

Reliable multicast transport protocols have been proposed since the mid-1990s [33, 37–41]. Several concepts developed in these proposals are adopted in our work. For example, the session manager in RMTP [37] is similar to our VCMTP multicast manager, as is the use of receivers tracking the status of successful or failed packet transmissions instead of the sender. However, there are two key differences between RMTP and VCMTP. Positive ACKnowledgments (ACKs) are used in RMTP with a tree of designated receivers handling ACKs to avoid the ACK implosion problem [38], while VCMTP uses negative acknowledgements (NACKs). Second, RMTP uses window-based flow control, while VCMTP uses a flow-control problem avoidance technique through control-plane methods.

In SRM both repair requests and retransmissions are multicast to the entire group. Repair requests differ from negative ACKs (NACKs) in that the latter are sent to a specific sender, while repair requests are sent to all participants. While this approach will be considered in future work, VCMTP uses TCP unicast connections for the NACKs and retransmissions, as proposed in MTP-2 [39].

The NACK-Oriented Reliable Multicast (NORM) protocol [41] uses Forward Error Correction (FEC) to improve reliability and "uses probabilistic suppression of redundant feedback based on exponentially distributed random backoff timers." The latter helps with scalability. A TCP friendly congestion control algorithm is proposed as NORM is developed for multicast IP.

A difference between MTP-2 and VCMTP is that the former claims scalability only under no loss scenarios, while in VCMTP, using our design concept of completing the whole message transmission before executing retransmissions, we aim for scalability in the presence of losses. This concept is similar to that used in Reliable Blast UDP (RBUDP) [42], which is a unicast transport protocol for use across dedicated optical circuits.

A Reliable Adaptive Multicast Protocol (RAMP), described in IETF RFC 1458, was enhanced for use over an all-optical, circuit-switched, gigabit network in an ARPA-sponsored Testbed for Optical NEtworking (TBONE) [40]. The question of slowing down other receivers due to flow control problems in some slow receivers is not addressed; instead retransmissions are executed immediately after reports of losses.

3.7 Conclusions

A new reliable multicast transport protocol was proposed for virtual circuits. As virtual circuits are provisioned prior to data transfer with a guaranteed rate, there should be no congestion in the data plane, which is difficult to handle in a multicast setting. The proposed Virtual Circuits Multicast Transport Protocol (VCMTP) handles error control and flow control. A key design concept of executing the whole message multicast before handling retransmission requests provides scalability but does incur a cost due to disk reads when the file size is larger than host memory. A prototype of VCMTP was implemented and evaluated for high-speed file transfers. Our conclusions are as follows: (i) as long as the receivers can estimate the rate at which their receive buffers can be depleted and choose a VC with the corresponding rate, packet losses can be kept small; (ii) both the sender computing resources and bandwidth can be reduced through the use of multicasting; and (iii) the separation of the multicast phase from the retransmission phase makes VCMTP suitable for single-file transfers, but not for continuous file transfers. VCMTP has the limitation that it is only sustainable when file inter-arrival times are significantly longer than the service times required to transfer files. Since this assumption does not hold in IDD, in the next chapter we will present a modified design of VCMTP.

Chapter 4

VCMTPv2

4.1 Introduction

The VCMTPv1 design is not suitable for continuous file transfers since block retransmissions to individual receivers are handled at the end of a file multicast. For example, if a single receiver missed a single block, the sender retransmits the block over TCP. The sender would then await a TCP ACK for that block before starting the next file multicast. CPU resources at the sender could thus be wasted, especially for wide-area multicast since round-trip propagation delay can be on order of tens to hundreds of milliseconds. Given that the IDD project requires continuous file transfers, we undertook a new VCMTP design, and called it VCMTPv2. Unlike VCMTPv1, retransmission requests are handled in parallel with the execution of file multicasting in VCMTPv2.

One of the key features of VCMTPv2 design is its ability to tradeoff file-transfer throughput for "fast" receivers (receivers that can keep up with the data arrival rate) with an acceptable level of robustness (the percentage of successful file delivery) for slow receivers. This tradeoff is achieved with a per-file configurable parameter called retransmission timeout factor. This factor determines the amount of time during which receivers can request retransmissions of missed blocks after a file has been multicast. It is defined as a multiple of the file multicast time to make the retransmission period longer for larger files. For a given file arrival rate, the larger this factor, the higher the robustness for slow receivers but the lower the throughput for fast receivers.

The VCMTP design was prototyped, and evaluated on U. Utah's Emulab testbed [22]. Random number generators were used to create samples of file inter-arrival times and file sizes. The sizes were used to create files for actual transfers over VCMTP in the testbed, and measurements were obtained. Packet losses were injected using the Linux `tc` utility at a fraction of the receivers to emulate "slow" receivers. Throughput and robustness metrics were characterized as a function of traffic load (arrival rate divided by service rate).

VCMTP has the potential for a broader impact. Besides the IDD project, large data sets are created in other scientific research projects such as the Earth System Grid [43] and Large Hadron Collider (LHC) [44] experiments. For the many scientists involved in these projects, some newly created data files could be distributed in push mode instead of requiring each scientist to download files in pull mode. In addition, there are commercial applications for data distribution to multiple receivers, e.g., electronic distribution of newspapers, financial data, teaching modules, and video files. Finally, with Content Delivery Networks (CDN) [45], Web sites and other data are often replicated to many servers deployed across the Internet to ensure proximity, and hence lower propagation delays in reaching users.

The key contributions described in this chapter are (i) a new VCMTPv2 design, (ii) a validated analytical model for single-file multicasts, and (iii) the evaluation of VCMTPv2 in the context of a continuous file-arrival process.

Section 4.2 reviews prior work on reliable multicast protocols. Section 4.3 describes the VCMTPv2 design, and Section 4.4 provides a detailed description of the finite state machines. Different underlying network service options for VCMTPv2 are discussed in Section 4.5. Sections 4.6 and 4.7 describe the VCMTPv2 prototype and its evaluation. Section 4.8 describes the integration of VCMTPv2 with the LDM software, and demonstrates an experiment of distributing IDD feedtypes through the LDM-VCMTPv2 integration. An inter-domain IP multicast test that was executed to determine if IP-multicast capability was enabled in RENs between UVA and UCAR is described in Section 4.9. The work is concluded in Section 4.10.

For ease of presentation, in the rest of this chapter, the term "VCMTPv2" is replaced by "VCMTP".

4.2 Related Work

Application-layer multicasting solutions that leverage peer-to-peer methods such as Bit Torrent have been proposed for Grid applications [46]. In contrast, the VCMTP approach is designed for network multicast solutions.

While VCMTP is designed for efficient reliable data distribution across virtual circuits, multicast protocols have been designed for other reasons and other types of networks, e.g., for energy efficiency in wireless networks [47], for improved application throughput in data-center networks [48], and to improve MPI-collective operation performance in clouds [49].

Given the popularity of ATM virtual-circuit networks in the nineties, we searched the literature for prior work on reliable transport protocols for ATM networks. A reliable multicast solution for ATM networks was developed by Turner [50]. It required hardware-assistance at the switches, and did not handle the flow control problem (arising from receiver-buffer overflows). VCMTP addresses flow control, and does not require hardware assistance in the VC switches. There were other transport protocols for ATM multicast such as the one by Ma and El Zarki [51], but these solutions were for audio/video streaming, not data distribution.

Work on reliable multicast transport protocols for IP networks in the IETF Reliable Multicast Transport working group [52] include Asynchronous Layered Coding (ALC) [53] and NACK-Oriented Reliable Multicast (NORM) [41]. With ALC, the sender multicasts packets within a session on multiple channels at potentially different rates allowing for multiple rate congestion control. Each receiver can obtain packets from one or more channels within the session based on the available bandwidth on the path from the sender, and its own reception rate. This combination of layered coding transport and forward error correcting codes allows for massively scalable reliable multicast. More generally, solutions that combine error correcting codes such as RaptorQ codes [54] with techniques such as data carousel [55] or broadcast disk [24] are well suited to situations in which the number of multicast receivers is large (e.g., millions). While this approach works well when massive scalability is required, reception overhead can be high [23]. The sender computing resources and network bandwidth could be more than in a NACK-based scheme if the number of

receivers is moderate. As pointed out by Barcellos et al. [56], "the sender has no knowledge about the network and needs to be conservative in terms of redundancy to guarantee reliable delivery." Since the target applications of VCMTP are in the scientific community, the number of receivers is in the hundreds, not millions. For example, the CONDUIT feedtype in the IDD project has a maximum fan-out of 104 receivers [25].

When the number of receivers is small-to-moderate, a negative acknowledgement (NACK) based scheme, as in Scalable Reliable Multicast (SRM) [33] and NORM is a better approach (positive acknowledgment schemes, as in Reliable Multicast Transport Protocol (RMTP) [37], have the ACK implosion problem). In SRM, requests and retransmissions are also multicast, but in VCMTP, requests and retransmissions are sent over unicast TCP connections as in MTP-2 [39] and Reliable Adaptive Multicast Protocol (RAMP) [40].

In the VCMTP design, several concepts were taken from protocols proposed in the research literature on reliable multicast transport protocols [33, 35, 37–40, 56–60], as well as unicast reliable message-based protocols such as Stream Control Transmission Protocol (SCTP) [61]. For example, should VCMTP be message-based or byte-stream based? SCTP is a unicast reliable message-based transport protocol, while TCP is byte-stream based. Most reliable multicast transport protocols, such as SRM, Multicast Transport Protocol (MTP) [58], RAMP, and NORM, support message-based communications, with RAMP and NORM additionally supporting byte-stream mode. RMTP appears to be the exception in that it is byte-stream based. Our reasons for choosing the message-based option are explained in Section 4.3.

However, unlike NORM, VCMTP does not require data-plane congestion control schemes such as the TCP-friendly multicast congestion control [62] as it is designed for virtual circuits. NORM determines a group Round Trip Time (RTT) based on feedback from receivers and the RTT estimate of the current limiting receiver determines the sending rate. Some other reliable multicast proposals, such as SRM, note that multicast congestion control, which is required if the underlying network service is a connectionless service such as IP, is a difficult proposition. In VCTMP, data-plane congestion is avoided through the use of admit/reject decisions made by the VC scheduling/provisioning control-plane system in the setup phase.

4.3 VCMTP Overview

VCMTP is a negative-acknowledgment (NACK) based reliable transport protocol, designed for multicast virtual circuits, in which a file is segmented into blocks (limited by a maximum block size) and transmitted over a multicast network service to one or more receivers, and retransmissions of errored/lost blocks for individual receivers are carried over unicast reliable connections between the sender and each receiver. Even though the use of rate-guaranteed virtual circuits eliminates congestion related packet losses, retransmissions may be required due to bit errors and receive-buffer overflows in multitasking receivers ("flow-control" problem).

4.3.1 VCMTP Application Programming Interface (API)

The VCMTP API is message-based (unlike TCP's byte-stream API), and asynchronous (unlike TCP's synchronous API). A *message-based* API is more suitable for multicast communications than a byte-stream API [33]. In message-based communications, application data units (ADU) are preserved by the transport layer [63], unlike in TCP, where a TCP segment can include bytes from different application data units. This design choice allows for an easier identification of missed application data units if one of the multiple receivers suffers a network loss and needs a complete message retransmission. The API is *asynchronous*, which means that when an application invokes the VCMTP function to send a file, the application is not blocked but it cannot modify the user data until VCMTP completes the transfer. The VCMTP API requires separate calls to send files and receive completion notifications. This is unlike the synchronous API of TCP, which blocks the application when the TCP `send` function is invoked. However, the TCP `send` call returns quickly, i.e., as soon as the data is copied from user-space to kernel-space within the sender; in other words, TCP `send` does not wait until the data is delivered to the receiver before returning. The copy held in kernel space allows TCP to slowly send the data to the receiver and perform retransmissions if required. Meanwhile control of the user-space data is immediately released back to the application, allowing the latter to perform other actions, such as delete, move, copy, on this data. The disadvantage of TCP's approach is the higher latency incurred

in copying the data from user space to kernel space, an operation that is avoided in the zero-copy approach of an asynchronous API.

On the receive-side, the API there are two alternatives by which an application can determine where incoming files are stored. In the first method, called "per-file notification," the VCMTP receiver notifies the application upon reception of the Begin-of-File (BOF) control message for each new file, at which point the application must specify whether or not VCMTP should receive the file, and if so, where to store the file, and optionally what new name to give it. In the second method, called "batched notification," the application provides VCMTP a folder into which each new file is stored and a period for receiving notifications. VCMTP in turn will notify the application periodically via a completion event queue.

4.3.2 VCMTP functions

As VCMTP is a reliable transport protocol, it has *error control* functionality. Unlike in TCP where positive acknowledgments (ACKs) are used, VCMTP uses negative ACKs (NACKs), which are triggered by out-of-sequence blocks. Since virtual circuits guarantee sequenced delivery, sequence numbers can be used to detect missing blocks. VCMTP does not require data-plane *congestion control* since it is designed for virtual circuits that are established with guaranteed bandwidth and buffer allocations. Therefore, packet losses due to congestion events in VC switch buffers should be small, if any. For *flow control*, ON-OFF and window-based schemes are not feasible in the presence of multiple receivers, and hence there is no data-plane support for flow control in VCMTP. Instead, VCMTP relies on a control-plane solution in which each receiver callibrates its ideal receive rate, and sends this rate to the multicast sender to help the sender plan the multicast VCs. This approach may not eliminate, but can reduce, packet loss at the receivers.

4.3.3 VCMTP messages

The term "message" is reserved for control messages exchanged between the VCMTP sender and receivers. Examples include `Begin-of-File (BOF)` and `End-of-File (EOF)` control

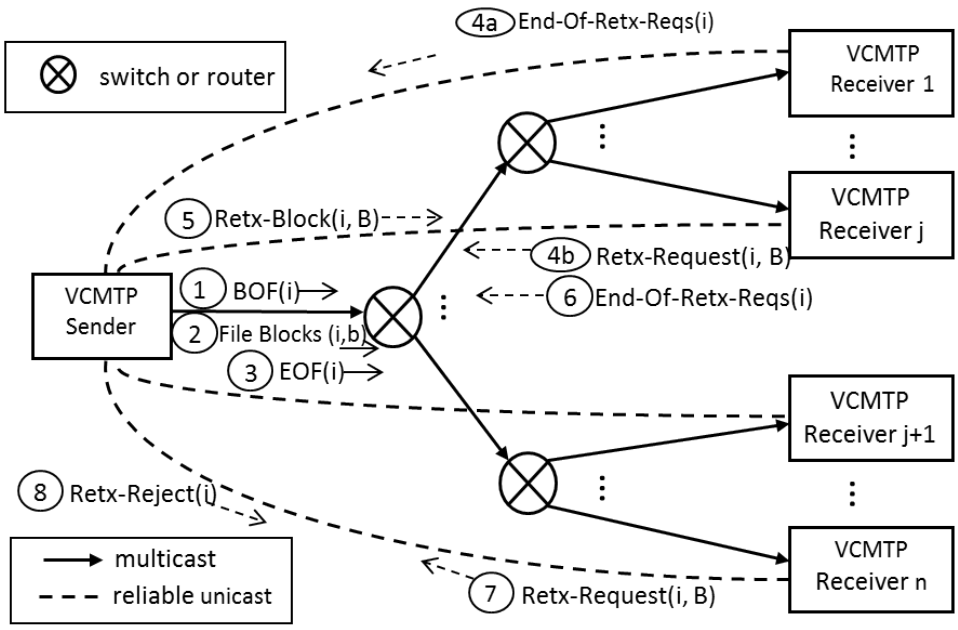

Figure 4.1: VCMTP Messaging for file i

messages. The term "file" is used to denote both disk and memory user data that is passed down by the application to VCMTP for multicasting.

Fig. 4.1 illustrates VCMTP messaging with the solid lines showing the multicast tree, and the dashed lines representing the reliable unicast connections. The VCMTP sender starts by multicasting a BOF message ① and then multicasts the file in the form of blocks ②. The BOF carries metadata such as name and size of the file. After the sender completes multicasting a file, it multicasts an EOF message to all receivers ③.

Each VCMTP receiver j identifies lost/errored blocks, if any, based on out-of-order block reception, at which point it requests retransmissions by sending `Retx-Request` control messages ④ᵇ and then receives retransmitted blocks ⑤, both on the reliable unicast connection. File multicasting and loss recovery are concurrent, which means the VCMTP sender can be concurrently transmitting blocks from a file, while handling loss recovery for previously multicast blocks from the same file, or blocks from a previous file.

After the EOF and retransmissions of all lost/errored blocks have been received, each receiver notifies the sender on the reliable unicast connection via an `End-Of-Retx-Reqs` control message, ④ᵃ or ⑥, that it has no more retransmission requests. A sender-side timer

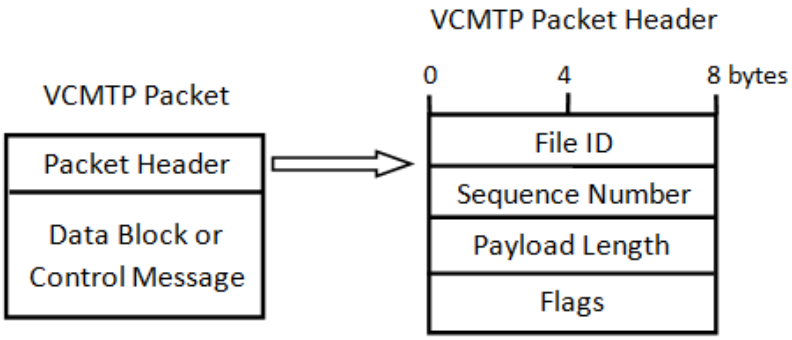

Figure 4.2: VCMTP Packet Format

is used for this retransmission phase because the file eventually needs to be released for the application to reclaim (recall the asynchronous API). If the sender receives a `Retx-Request` control message from a receiver for a file whose timer has expired ⑦, the sender rejects the request by sending a `Retx-Reject` control message ⑧, and the transmission of the file to that particular receiver is deemed a failure. The application has to use some alternative mechanism to request the file individually from the sender. Fig. 4.1 illustrates three cases, with receiver 1 receiving all blocks successfully and hence sending just the `End-Of-Retx-Reqs` control message, receiver j missing certain blocks but requesting them and receiving retransmissions successfully, and receiver n being somehow delayed in its request for retransmissions and hence receiving a `Retx-Reject`.

4.3.4 VCMTP packet format

In VCMTP, every data block or control message is encapsulated in a *VCMTP packet*. Each VCMTP packet includes a 32-byte header with the four 8-byte fields shown in Fig. 4.2. The `File ID` is a number assigned by the sender that uniquely identifies each file within a multicast group. The `Sequence Number` is used in a data packet to indicate the starting byte position of the VCMTP data block within the file identified by `File ID`. The `Payload Length` field indicates the size of the payload (either a data block or a control message) in bytes. Finally, the `Flags` field is a bit vector that indicates the type of VCMTP packet: (i) data block; (ii) retransmitted data block; (iii) `BOF` message; (iv) `EOF` message; (v) `BOF-Request` message; (vi) `Retx-Request` message; (vii) `End-Of-Retx-Reqs` message; and (viii) `Retx-Reject` message. The first two types of VCMTP packets carry data-plane file

Table 4.1: VCMTP packet format

Message Type	**Format**

`BOF`	`transfer_type (1)`
	`file_size (8)`
	`file_name (256)`

`EOF`	none
`BOF-Request`	none
`Retx-Request`	`start_pos (8)` `end_pos (8)`
`End-Of-Retx-Reqs`	none
`Retx-Reject`	none

blocks, while the remaining six types of VCMTP packets carry control-plane messages. The roles of all the control-plane messages except the `BOF-Request` were explained in the previous paragraph along with Fig. 4.1. The purpose of the `BOF-Request` message is to handle lost BOF messages. A receiver detects BOF loss if it starts receiving data blocks for a new file without a corresponding BOF. Upon receiving a `BOF-Request` from a receiver, the sender sends a corresponding `BOF` on the unicast reliable connection to that particular receiver.

Table 4.1 shows the format of each control message type. The number in the parenthesis after each field name indicates the size of the field in bytes. Four types of VCMTP control messages, `EOF`, `BOF-Request`, `End-Of-Retx-Reqs`, and `Retx-Reject` messages, do not have any payload fields. For these messages, the `File ID` and `Flags` fields in the VCMTP header are sufficient to serve their operational functions. On the other hand, a `BOF` message includes three fields: `transfer_type`, `file_size` (in bytes), and `file_name` (as an ASCII string). The `transfer_type` is used to indicate whether the forthcoming transfer is memory-to-memory, memory-to-disk, disk-to-memory, or disk-to-disk. Finally, a `Retx-Request` message includes two fields in the payload: `start_pos` and `end_pos`, which indicate the starting and ending byte positions of the missing/errored data blocks for which retransmissions are being requested. The file for which retransmissions are being requested is identified by `File ID` in the VCMTP packet header of the `Retx-Request` message.

4.4 VCMTP FSM specifications

Each multicast group consists of one sender and an arbitrary number of receivers. On the sender, three FSMs are defined: *multicast sender*, *coordinator*, and *receiver-specific retransmitter*. There is one instance of each of the first two FSMs, and as many instances of the last FSM as there are receivers in that group. The multicast sender transmits the BOF, file blocks, and EOF in multicast mode. Each receiver-specific retransmitter retransmits data blocks in response to retransmission requests from its corresponding receiver. The coordinator manages the receiver-specific retransmitters and oversees retransmissions. The coordinator is informed by the sender when all data blocks of a file have been multicast, at which time the coordinator starts the retransmission timer for the file. Upon receiving notifications of successful file completions from all receiver-specific retransmitters, or when the retransmission timer for the file expires (whichever occurs first), the coordinator notifies the application that file transfer is complete. Since the API is asynchronous, this notification is required for the application to reclaim the file. A set of variables are created for these VCMTP sender FSMs to control and track the status of the multicast activities. Table 4.2 lists these variables with brief descriptions.

On each receiver, there are two FSMs: *data receiver*, and *retransmission requester*. The data receiver receives the original multicast data blocks and the retransmitted data blocks, writes these blocks into memory/disk locations, and handles the reception of the BOF, EOF and `Retx-Reject` control messages. The data receiver also interacts with the application, notifying it of successful or failed file receptions. The retransmission requester is responsible for sending `Retx-Request`, `BOF-Request` and `End-Of-Retx-Reqs` control messages to the sender. Variables for the VCMTP receiver FSMs are listed in Table 4.3.

The reliable unicast connection is initiated by the data receiver FSM. On the sending side, the coordinator listens for connection requests from receivers. Connection closure is handled by the receiver-specific retransmitter on the sending side and the data receiver on the receiving side.

For each FSM, transitions between states are typically triggered by incoming events, and usually include a set of actions. Actions optionally include tasks that may (i) update

Table 4.2: Variables for VCMTP Sender FSMs

Variable	Description
metadata(i)	metadata about file i to be multicast, e.g., file name and size
retx_timeout(i)	status of the retransmission timer for file i
num_receivers	total number of receivers in the multicast group
receiver_vector	a vector of receiver IDs in the multicast group
file_done_sent(i)	a boolean variable that indicates whether the notification event for file i has been sent to user application
file_receive_status	a matrix that tracks the receive status of each file i for each receiver j

Table 4.3: Variables for VCMTP Receiver FSMs

Variable	Description
metadata(i)	metadata about file i to be received
received_bytes(i)	total number of data bytes that have been received for file i
write_condition(i)	a variable that indicates the status of the current write location for file i; can be either "temporary" or "final"

Table 4.4: State Transition Table for Multicast Sender FSM

Current State	Input Event	Output Event	Next State
NULL	API_Init_Sender	INL_Init_Coordinator	READY-TO-SEND
READY-TO-SEND	API_Send(i)	BOF(i)	SENDING-MSGS
	API_Close_Sender	INL_Close_Coordinator	NULL
SENDING-MSGS	API_Send for new files	DATA_Block EOF INL_Handle_Completion BOF for new files	SENDING-MSGS (files pending) READY-TO-SEND (no files pending)

Table 4.5: State Transition Table for Receiver-Specific Retransmitter FSM(j)

Current State	Input Event	Output Event	Next State
NULL	INL_Init_Retransmitter(j)		ACTIVE
ACTIVE	Retx-Request(i,B)	DATA_Block(i, b) (not timed out) or Retx-Reject(i) (timeout)	ACTIVE
	BOF-Request(i)	BOF(i)	ACTIVE
	End_Of_Retx_Reqs(i)	INL_File(i)_Done_Rcvr(j) (after all retx data blocks have been sent)	ACTIVE
	INL_Close_Retransmitter(j)	INL_Close_Unicast_Connection	ACTIVE
	INL_Unicast_Connection_Closed	INL_Retxmitter(j)_Exited	NULL

variables, (ii) make operational decisions, and (iii) generate output events. Various sources, such as Holzmann's textbook [64], Unified Modeling Language (UML) [65], Specification and Description Language (SDL) [66], as well as the TCP connection finite state machine [67], and BGP finite state machine [68], are used as the basis for the specification of VCMTP FSMs described below.

As a convention, the following prefixes are used in event names:

1. API for events generated or consumed by the application;

2. DATA for data-plane VCMTP blocks that are sent by the VCMTP sender;

3. INL for internal events created by one VCMTP component and consumed by another component in the same side (sender or receiver).

Table 4.6: State Transition Table for Coordinator FSM

Current State	Input Event	Output Event	Next State
NULL	INL_Init_Coordinator		ACTIVE
ACTIVE	INL_Unicast_Connection_Created(j)	INL_Init_Retransmitter(j)	ACTIVE
	INL_Handle_Completion(i)		ACTIVE
	INL_File(i)_Done_Rcvr(j)	API_File_Done(i) (no pending receivers)	ACTIVE
	INL_Retx(i)_Timeout	API_File_Done(i)	ACTIVE
	INL_Close_Coordinator	INL_Close_Retransmitter(j)	CLOSING-RETXMITTERS
CLOSING-RETXMITTERS	INL_Retxmitter(j)_Exited		CLOSING-RETXMITTERS (more pending receivers) or NULL (no pending receivers)

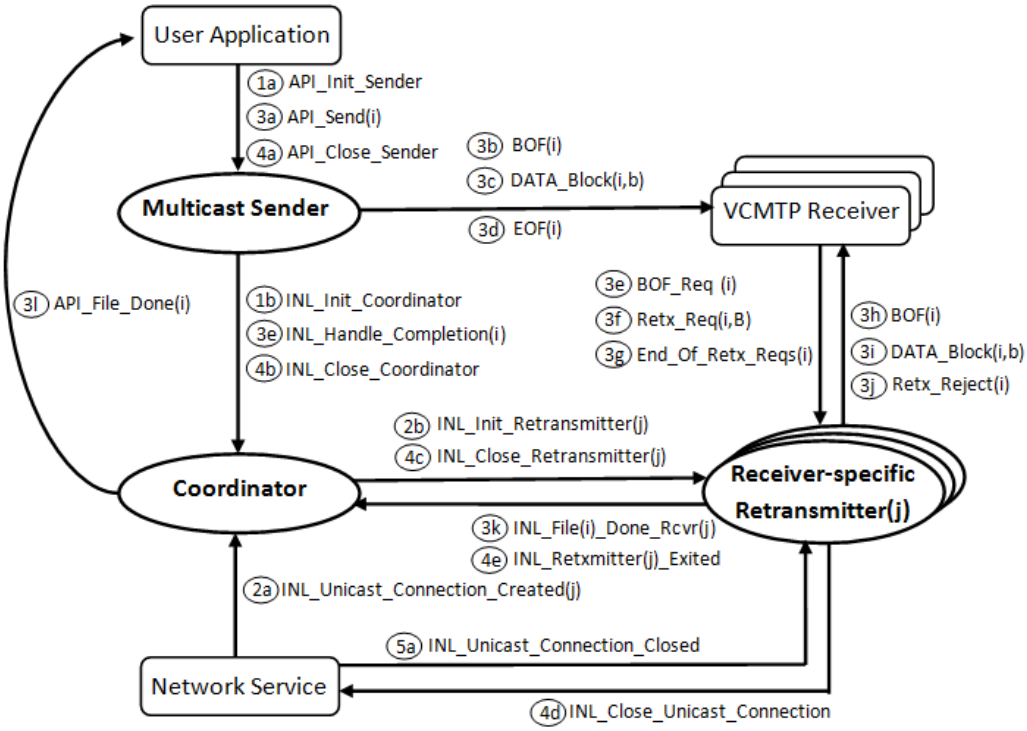

Figure 4.3: Object communication model for VCMTP sender

Tables 4.4, 4.5 and 4.6 show the state transition tables, with input and output events, for the three VCMTP sender FSMs. In addition, Fig. 4.3 shows the events communicated between the three FSMs in a VCMTP sender, along with interactions with external entities such as the user application, VCMTP receivers, and network service. Text-based specifications for the VCMTP sender FSMs are provided below.

VCMTP Sender - multicast sender

The multicast sender FSM has three states: NULL, READY-TO-SEND, SENDING-MSGS.

NULL state:

- The multicast sender stays at this state until an `API_Init_Sender` event is generated by an application. When this happens, the multicast sender:
  - sends an `INL_Init_Coordinator` event to create the coordinator, and
  - changes its state to READY-TO-SEND.

READY-TO-SEND state:

- If an `API_Send(i)` event is received, the multicast sender:
  - writes metadata about file i to a shared variable called `metadata(i)`,
  - multicasts a Begin-of-File control message (i.e., a `BOF(i)` event) to all receivers, which contains metadata information such as file size and file name,
  - multicasts the first data block of the file, and
  - changes its state to SENDING-MSGS.
- If an `API_Close_Sender` event is received, the multicast sender:
  - sends an `INL_Close_Coordinator` event to have the coordinator terminate itself,
  - changes its state to NULL.

SENDING-MSGS state:

- In this state, the multicast sender keeps multicasting file blocks to all receivers, i.e., by generating `DATA_Block(i, b)` events for all blocks b in file i for all files for which it previously received `API_Send(i)` events.
- If an `API_Send(i)` event is received in this state, the FSM performs the same actions as if this event is received in the READY-TO-SEND state except for the last action. Here, the FSM just remains in the SENDING-MSGS state. This procedure is required because an application should not be blocked from sending a smaller file while awaiting the completion of a large file multicast.
- When file i is fully multicast, the multicast sender:

- multicasts a corresponding `EOF(i)` message to all receivers to indicate the end of multicast phase for file i.
- notifies the coordinator through an `INL_Handle_Completion(i)` event that it should initiate the process for timing the retransmission phase and releasing access of the file to the application when the timer expires, or all receivers acknowledge completion of file reception, whichever comes first (additionally, the application can be informed of the success/failure of file delivery), and
- changes its state to READY-TO-SEND if file i was the only file being multicast, or remains in the SENDING-MSGS state if other files are being multicast.

VCMTP Sender - receiver-specific retransmitters

A receiver-specific retransmitter has two states: NULL and ACTIVE.
NULL state:

- A retransmitter stays at this state until the coordinator issues an `INL_Init_Retransmitter(j)` event, which causes this retransmitter to change its state to ACTIVE.

ACTIVE state:

- Upon receiving a `Retx_Request(i,B)` event, it consults the shared variable `metadata(i)` (which was written by the multicast sender), finds blocks in $\mathbf{B}$ and retransmits blocks $b \in \mathbf{B}$ over its reliable unicast connection to the receiver (denoted by the `DATA_Block(i, b)` event) unless the timer associated with file i has expired (determined by consulting a shared variable, `retx_timeout(i)`). In the latter case, it sends a `Retx_Reject(i)` message to its receiver. There is no state change.
- If a `BOF_Request(i)` event is received, it sends a `BOF(i)` to its receiver over the unicast connection. This can happen if the original multicasted BOF had bit errors or was dropped due to receive buffer overflow. There is no state change.
- If a `End_Of_Retx_Reqs(i)` event is received, it generates an `INL_File(i)_Done_Rcvr(j)` event and there is no state change.

- If an `INL_Close_Retransmitter(j)` event is received from the coordinator, the retransmitter generates an `INL_Close_Unicast_Connection` event to ask the unicast service to close the connection. There is no state change.

- If an `INL_Unicast_Connection_Closed` event is received, the retransmitter

  - terminates itself and sends an `INL_Retxmitter(j)_Exited` event to the coordinator before it exits;

  - changes its state to NULL.

VCMTP Sender - coordinator

The coordinator FSM has the following states: NULL, ACTIVE, and CLOSING-RETXMITTERS.
NULL state:

- The coordinator stays in this state until the *multicast sender* issues an `INL_Init_Coordinator` event. After the `INL_Init_Coordinator` event is received and the coordinator is initialized, it opens a unicast service that listens for connection requests from receivers. It then changes its state to ACTIVE.

ACTIVE state:

- As long as there is at least one receiver in the multicast group, the coordinator will stay in this state to coordinate the actions related to the three roles listed above.

- If the coordinator receives a `INL_Unicast_Connection_Created(j)` event from the local unicast service, it will:

  - send an `INL_Init_Retransmitter(j)` event to create a receiver-specific retransmitter corresponding to the receiver for which the unicast connection was created,

  - updates the `receiver_vector` with the new receiver ID,

  - increase a variable named `num_receivers`, which indicates the total number of current receivers, and

  - stays in the same state.

- When the coordinator receives an `INL_Handle_Completion(i)` event from the multicast sender, it:

  - creates a boolean variable `file_done_sent(i)` to indicate whether or not the `API_File_Done(i)` event has been sent to the application for file i, and initializes its value to be *false*,
  - initiates the retransmission timer for file i,
  - creates the `retx_timeout(i)` variable to maintain the status of the retransmission phase timer for file i,
  - updates the `file_receive_status` matrix to add a new row of receive status values for file i, and set the receive status for all receivers to be *false*,
  - stays in the same state.

- When the coordinator receives a `INL_File(i)_Done_Rcvr(j)` event (which is generated by the j^{th} receiver-specific retransmitter), it:

  - updates the `file_receive_status` matrix to set the receive status of file i for receiver j to be *completed*.
  - checks the `file_receive_status` matrix to see if all receivers have finished receiving file i: if all receivers have finished receiving file i, while `file_done_sent(i)` is false, it generates an `API_File_Done(i)` event to inform the user application about the completion of the file transfer, and then sets the `file_done_sent(i)` variable to be *true*; otherwise, it does nothing, and
  - stays in the ACTIVE state in both cases.

- When the coordinator receives an `INL_Retx(i)_Timeout` event, it will:

  - set the local variable `retx_timeout(i)` to "expired" for file i,
  - generate an `API_File_Done(i)` event if `file_done_sent(i)` is false,
  - set `file_done_sent(i)` to be true, and
  - stay in the same state.

Table 4.7: State Transition Table for Data Receiver FSM

Current State	Input Event	Output Event	Next State
NULL	API_Init_Receiver	INL_Create_Unicast_Connection	IDLE
IDLE	INL_Unicast_Connection_Created	INL_Init_Retx_Requester	RECVING-DATA
RECVING-DATA	BOF(i)	API_BOF(i)_Received	RECVING-DATA
	API_BOF_Response(i)		RECVING-DATA
	DATA_Block(i, b)	INL_BOF_Req(i) (BOF loss) or INL_Retx_Req(i,B) (data loss) or API_File(i)_Received	RECVING-DATA
	EOF(i)	INL_Retx_Req(i,B) (if data loss) INL_End_Of_Retx_Reqs(i)	RECVING-DATA
	Retx-Reject(i)	API_File(i)_Receive_Failed	RECVING-DATA
	API_Close_Receiver	INL_Close_Retx_Requester INL_Close_Unicast_Connection	RECVING-DATA
	INL_Unicast_Connection_Closed	INL_Close_Retx_Requester (if sender-initiated closure)	NULL

Table 4.8: State Transition Table for Retransmission Requester FSM(j)

Current State	Input Event	Output Event	Next State
NULL	INL_Init_Retx_Requester		READY-TO-SEND
READY-TO-SEND	INL_Retx_Req(i,B)	Retx-Request(i,B)	READY-TO-SEND
	INL_BOF_Req(i)	BOF-Request(i)	READY-TO-SEND
	INL_End_Of_Retx_Reqs(i)	End-Of-Retx-Reqs(i)	READY-TO-SEND
	INL_Close_Retx_Requester		NULL

- When the coordinator detects that a receiver-specific retransmitter has terminated, it will decrease the value of `num_receivers` by one, and update the `receiver_vector`. If the value of `num_receivers` becomes zero, it changes its state to NULL. Otherwise, it stays in the same state.

- When the coordinator receives an `INL_Close_Coordinator` event, it will:

  - generate an `INL_Close_Retransmitter(j)` event for each of the receiver-specific retransmitters, and

  - change its state to CLOSING-RETXMITTERS.

CLOSING-RETXMITTERS state:

- In this state, the coordinator waits until all receiver-specific retransmitters are terminated. When an `INL_Retxmitter(j)_Exited` event is received, the coordinator:

  - reduces the value of `num_receivers` by ones, and updates the `receiver_vector`,

  - changes its state to NULL if `num_receivers` equals zero.

Tables 4.7 and 4.8 show the state transition tables, with input and output events, for the two VCMTP receiver FSMs. In addition, Fig. 4.4 shows the interaction between the two FSMs in a VCMTP receiver, along with interactions with external entities such as the

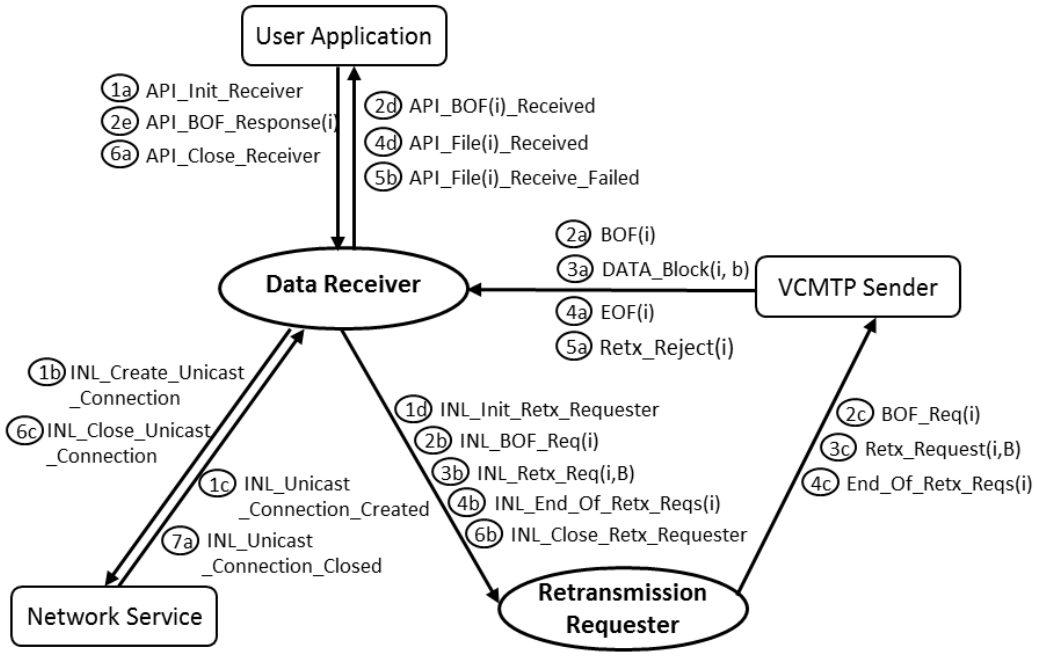

Figure 4.4: Object communication model for VCMTP receiver

user application, VCMTP sender, and network service. Text-based specifications for the VCMTP receiver FSMs are provided below.

VCMTP Receiver - data receiver

The FSM for a data receiver has three states: NULL, IDLE, and RECVING-DATA.

NULL state:

- A data receiver stays in this state until the user application issues an `API_Init_Receiver` event. After the data receiver is initialized, it:
  - sends an `INL_Create_Unicast_Connection` event to the local unicast service to open a connection with the sender;
  - changes its state to IDLE.

IDLE state:

- In this state, the data receiver keeps waiting for the unicast connection to the VCMTP sender to be created. When an `INL_Unicast_Connection_Created` event is received from the local unicast service, the data receiver:

- sends an `INL_Init_Retx_Requester` event to initiate the *retransmission requester*;
- changes its state to RECVING-DATA.

RECVING-DATA state:

- In this state, the data receiver handles both multicast data blocks, and retransmitted data blocks received on the unicast connection, and control messages (i.e., BOF and EOF messages).
- When a `BOF(i)` event is received, it determines if it is a BOF for a new file (received via multicast), or a retransmitted BOF (received on the unicast connection). If it is a BOF for a new file, the data receiver:
  - creates a shared variable called `metadata(i)`,
  - creates another shared variable called `received_bytes(i)`, which indicates the total number of bytes that have been received for file i, which is initialized to zero
  - creates a `write_condition(i)` variable, and sets it to "temporary" if the *per-file* notification mode was specified, or "final" if the *batched* notification mode was specified.
  - if the user application had specified the *per-file* notification mode, it generates an `API_BOF(i)_Received` event to notify the user application about the arrival of a BOF for a new file.
  - remains in the same state.

  Otherwise, if the BOF is a retransmission received on the unicast connection, then the data receiver:

  - generates the `metadata(i)` variable for file i,
  - if the *per-file* notification mode was specified, it sends an `API_BOF(i)_Received` event to the user application. Otherwise, if the *batched* notification mode was specified, the data receiver first copies the data already received for file i from the temporary buffer to the *destination location*, then deletes the buffer, and finally sets the value of `write_condition(i)` to "final."

- remains in the same state.

- When an `API_BOF_Response(i)` event (which is generated by the user application in the *per-file* notification mode) is received, the data receiver first checks if the application indicates that the file should be ignored. If so, it simply deletes the temporary buffer for file i, and configures `metadata(i)` to ignore all data blocks received for this file. Otherwise, the data receiver:

  - updates `metadata(i)` with the name and location to use for the received file i,
  - if there is no temporary buffer for file i in which already arrived data blocks are held, it moves to the next step; on the other hand, if there is a temporary buffer, it copies all the received data for file i from the temporary buffer into the right storage location as indicated by the user application, and it deletes the temporary buffer for file i,
  - determines if all blocks for file i have been successfully received by comparing `received_bytes(i)` against the total number of bytes for file i from `metadata(i)`. If they are equal, which means all bytes have been received for file i, then the data receiver generates an `API_File(i)_Received` event to inform the user application about the successful file reception; if not all blocks are as yet received, it sets the `write_condition(i)` to "final."
  - remains in the same state.

- When `DATA_Block(i, b)` is received, the data receiver first checks whether it is a duplicated data block (i.e., the same block has been previously received either through multicast or retransmission). If so, it ignores the received block. Otherwise, it then checks the transfer mode. If the reception is in *batched* notification mode, then three cases are possible: (i) BOF(i) has already been received, (ii) BOF(i) is not yet received and a BOF retransmission request has not yet been sent, and (iii) BOF(i) is not yet received but a BOF retransmission request has already been sent (in response to a previously received data block). Procedures for these three cases are listed under **(Proc A)**, **(Proc B)**, and **(Proc C)**, respectively. If the reception is in *per-file* notification

mode, it first checks to see if the application has sent the `API_BOF_Response(i)` event and `metadata(i)` indicates that file i should be ignored. If so, it drops the data block. Otherwise, four cases are possible: i) BOF(i) and the `API_BOF_Response(i)` event have already been received, and the application indicates to receive the file, (ii) BOF(i) has been received, but `API_BOF_Response(i)` event is pending, (iii) BOF(i) is not yet received and a BOF retransmission request has not yet been sent, and (iv) BOF(i) is not yet received but a BOF retransmission request has already been sent. Procedures for the four cases are **Proc A**, **Proc D**, **Proc B**, and **Proc C**, respectively.

In **Proc A**, the data receiver

- writes the data block into the *destination location.*
- *missing block handling procedure:* checks for missing blocks only if the data block is received via multicast. This is done by comparing the sequence number of the just received block with that of the last received block, and by using the convention of starting with sequence number 1 for each file to detect loss of the first block. If one or more blocks are missing, it sends `INL_Retx_Req_Msg(i,B)`, where **B** is a set of missing block sequence numbers, to the retransmission requester,
- increases the value of `received_bytes(i)` by the number of bytes in the received block,
- if the `write_condition(i)` is "final," and all blocks of file i have been successfully received, the data receiver generates an `API_File(i)_Received` event to inform the user application about the successful file reception, and
- remains in the same state.

In **Proc B**, the data receiver

- creates a temporary buffer to write the received block, and
- generates a shared variable `received_bytes(i)` for file i, and sets its value to the number of bytes in the received block,
- creates a `write_condition(i)` variable and sets it to "temporary,"

- sends an `INL_BOF_Req(i)` event to the retransmission requester,
- executes the *missing block handling procedure* described above, and
- remains in the same state.

In **Proc C**, the data receiver

- writes the received block into the corresponding temporary buffer,
- increases the value of `received_bytes(i)` by the number of bytes in the received block,
- executes the *missing block handling procedure* described above, and
- remains in the same state.

Proc D has two different cases. If `DATA_Block(i, b)` is the first received data block for file i, then the data receiver will take the same actions as **Proc B** except that step (4) (i.e., sending the `INL_BOF_Req(i)` event) is skipped. Otherwise, the data receiver will take the same actions as **Proc C**.

- When an `EOF(i)` event is received, the data receiver first checks whether `metadata(i)` exists for file i. If not, it means the EOF message is the first block received for file i, and the BOF message and all data blocks have been lost. In this case, it will send an `INL_EOF(i)_Only` event to the user application. On the other hand, if `metadata(i)` exists, it will first check whether file i should be ignored. Is the file should be ignored, it simply ignores the received EOF message. Otherwise, the data receiver:
  - determines if there are any missing blocks at the end of file i. If so, it will send a corresponding `INL_Retx_Req(i,B)` event to the retransmission requester;
  - sends an `INL_End_Of_Retx_Reqs(i)` event to the retransmission requester,
  - remains in the same state.
- When an `INL_EOF(i)_Only_Response` event is received, the data receiver first checks whether the user application indicates that the file should be ignored. If so, it will do nothing and stay in the same state. Otherwise, the data receiver will generate

an `INL_BOF_Req(i)` event and an `INL_Retx_Req_Msg(i,B)` event for all blocks of file i. There is no state change.

- When a `Retx_Reject(i)` message is received, it
  - sends an `API_File(i)_Receive_Failed` event to the user application to indicate an incomplete file reception, and
  - remains in the same state.
- When the `API_Close_Receiver` event is received, the data receiver:
  - sends an `INL_Close_Retx_Requester` event to the retransmission requester,
  – sends an `INL_Close_Unicast_Connection` event to the unicast connection.
- When the `INL_Unicast_Connection_Closed` event is received, the data receiver terminates itself and changes its state to NULL.

VCMTP Receiver - retransmission requester

The retransmission requestrer has only two states: NULL and READY-TO-SEND.
NULL state:

- The retransmission requester stays in this state until an `INL_Init_Retx_Requester` event is generated by the data receiver. After the retransmission requester is initialized, its state changes to READY-TO-SEND.

READY-TO-SEND state:

- When an `INL_Retx_Req(i,B)` event is received, the retransmission requester will generate a corresponding `Retx_Request(i,B)` event, which is sent to the VCMTP sender on the unicast connection. There is no state change.
- When an `INL_BOF_Req(i)` event is received, the retransmission requester will generate a corresponding `BOF_Request(i)` event, which is sent to the VCMTP sender. There is no state change.

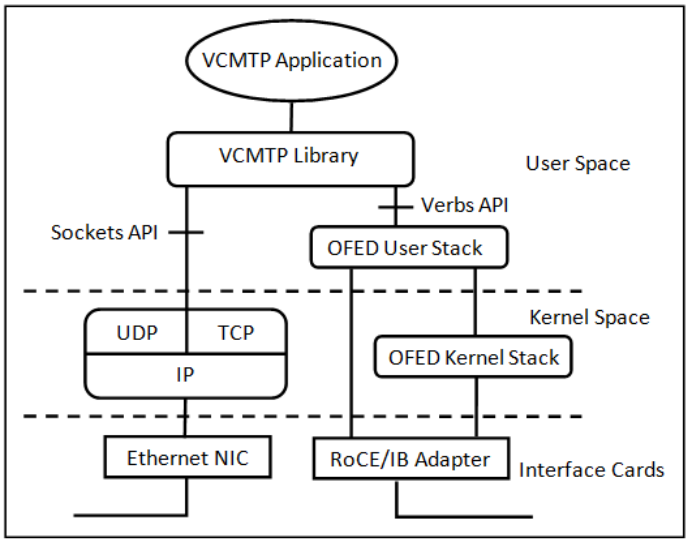

Figure 4.5: VCMTP implementation options

- When an `INL_End_Of_Retx_Req(i)` event is received, the retransmission requester will generate a corresponding `End_Of_Retx_Reqs(i)` event, which is sent to the VCMTP sender. There is no state change.
- When an `INL_Close_Recv_Retx_Requester` event is received, the retransmission requester terminates itself and changes state to NULL.

4.5 Multicast network service instantiations

As described in Section 4.3, VCMTP requires an underlying multicast network service (which can be unreliable) and a reliable unicast connection service.

Fig. 4.5 illustrates three potential options for the underlying network service, two of which use the *left-hand side protocol stack*, and the third option uses the *right-hand side protocol stack*. In the left-hand side, the host network interface card is standard Ethernet, while in the right-hand side, the host needs a Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE) [69] or InfiniBand (IB) adapter [70].

The left-hand side protocol stack can be executed across two types of multicast paths: (i) Layer-2 switched path: all switches on the path (e.g., in the network topology of Fig. 4.1) perform packet forwarding (including the multicast points) on Ethernet VLAN identifiers,

MAC addresses and/or MPLS labels, not on IP addresses; in this configuration, IP is used at the end points strictly to exploit the easy-to-use socket API, and (ii) IP-multicast path: packet multicasting is performed on IP packet headers, i.e., nodes marked ⊗ in Fig. 4.1 are IP routers. In both cases, a UDP socket is used by VCMTP to send file blocks over the multicast path, and TCP is used for the reliable unicast connections for loss recovery. In the IP-multicast configuration, in-sequence delivery is not guaranteed, which means a VCMTP receiver could receive duplicates (the original delayed block and a retransmitted block in response to a retransmission request that the VCMTP receiver would issue soon after receiving an out-of-sequence block), but VCMTP is designed to drop duplicate blocks. Thus, even though VCMTP was designed to run on multicast VCs, it can be run over an IP-multicast path. It is not an ideal solution but useful because of the ubiquity of inter-domain IP-routed service when compared to inter-domain virtual-circuit service.

The right-hand side protocol stack of Fig. 4.5 is designed to run VCMTP over RDMA networks, such as RoCE and IB [71]. This stack may be a better option for high-speed multicasting. Tierney et al. found through experiments that the CPU utilization is significantly lower with RoCE when compared to TCP for 10 Gbps speeds and higher [69]. This is because the InfiniBand (IB) transport- and network-layer protocols [70], which are part of RoCE, are implemented in hardware, while TCP/IP is implemented in software. InfiniBand is a packet-switched network, which is typically used within the local area, though some WAN extensions have been proposed [72]. The RoCE adapter card implements Ethernet and is therefore connected to an Ethernet switched network. End-to-end virtual circuits, realized as Ethernet VLANs, VLANs over MPLS label switched paths, or other Virtual Private LAN Service (VPLS) options [17], can be used across the wide-area to carry RoCE frames.

A VCMTP implementation designed for RoCE/IB would use the IB transport-layer unreliable multicast and reliable connection services (which are two types of IB transport-layer services among others). As shown in Fig. 4.5, the name of the interface equivalent to sockets for RoCE/IB is *Verbs* [73]. The verbs API includes operations such as RDMA Write, RDMA Read, RDMA Send/Recv, etc. RDMA relies on message based communications and is asynchronous. The OpenFabrics Enterprise Development (OFED) software is provided

by the OpenFabrics Alliance [74] and implements the verbs API for RoCE and InfiniBand adapters produced by several manufacturers. RDMA leverages zero-copy transfer, whereby data is moved from the sender user space directly to the channel adapter thus bypassing the kernel. At the receiver, the data is copied directly by the receiver channel adapter to user space bypassing the kernel. The OFED kernel stack is used for control operations, but can be bypassed in the data movement phase.

4.6 VCMTP prototype

We implemented a VCMTP prototype in C++ and tested it on Linux systems. The implementation consists of two sets of components: (i) VCMTP sender and VCMTP receiver applications; and (ii) VCMTP library functions.

In the FSMs described in Section 4.4, we referred to the layer beneath VCMTP as "network service," and assumed that the network offers a multicast service and a reliable unicast connection service. Given the ease of socket programming, we used UDP and TCP sockets in our VCMTP prototype. Effectively, our prototype implements the left-hand side protocol stack of Fig. 4.5. It can be run across an MPLS virtual-circuit or an IP-routed wide-area network.

4.6.1 VCMTP sender and receiver applications

The application software reads/writes files and calls VCMTP library functions (which are described next). The application software is multi-threaded as illustrated in Fig. 4.6. On the sending side, the (single) `multicasting thread`, and (single) `coordinator thread` implement the multicast sender FSM and coordinator FSM described in Section 4.4, respectively. For a multicast group consisting of N receivers, there are N `retransmission threads` in the VCMTP sender application; each thread implements the receiver-specific retransmitter FSM described in Section 4.4.

Each receiving host runs a VCMTP receiver application process, which consists of two threads as shown in Fig. 4.6. The `receiving thread` and `retransmission request`

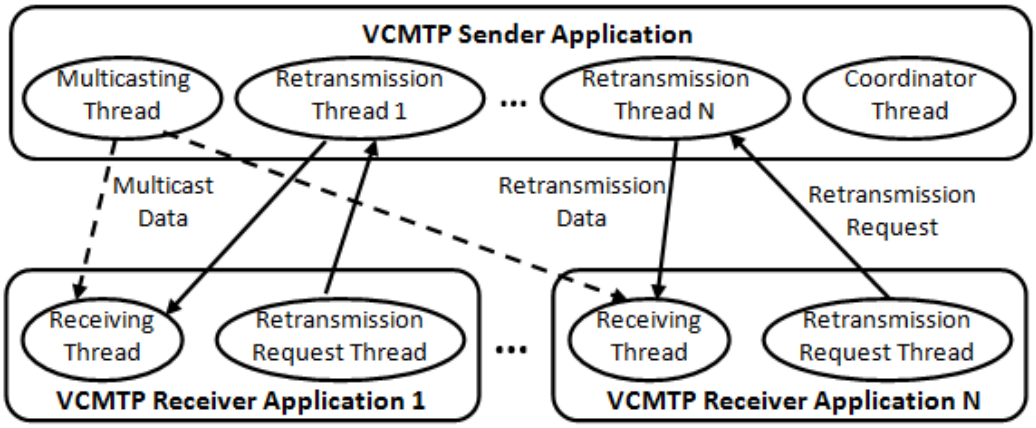

Figure 4.6: VCMTP Prototype Implementation

`thread` implement the data receiver and retransmission requester FSMs described in Section 4.4, respectively.

Fig. 4.6 also illustrates the interactions (see arrows) between the various threads of a VCMTP sender application and the threads of the multiple VCMTP receiver applications.

4.6.2 VCMTP library functions

The sender-side functions are described as follows.

1. `int StartGroup(VcmtpSenderParams config)`

The *StartGroup* function corresponds to the `API_Init_Sender` event, which causes the initialization phase actions described in Section 4.4 to be executed. The multicast sender FSM code implemented as part of this function opens a UDP socket using a multicast-group (Class-D) IP address, which configures the IP and Ethernet layers of the sending host. The parameter `VcmtpSenderParams` is a data structure that includes a set of values for multicast group configuration, such as the multicast group address, maximum send rate, and the port number on which the VCMTP sender will be listening for unicast TCP connection requests from VCMTP receivers.

2. `void SendMemoryData(void* data, size_t length, double retx_timeout_factor)`

The *SendMemoryData* function corresponds to the `API_Send(i)` event for a memory file. The parameter `data` is a pointer to the first byte of the data buffer. The `length` is the total size of data in number of bytes. The `retx_timeout_factor` is used to set a timer to

enforce a maximum duration allowed for retransmissions. The maximum retransmission duration is computed by multiplying the `retx_timeout_factor` with the file multicast time. If the timer expires, all retransmission requests for data blocks of this specific file will be rejected by the sender. The `retx_timeout_factor` offers administrators a knob for trading off robustness against throughput for a given configuration of sender, receivers and traffic intensity. More details about the performance impact of the retransmission timeout factor will be provided below.

Since VCMTP adopts an asynchronous programming model, all functions related to file sending/receiving will return immediately before a file is actually sent/received. In this implementation, the VCMTP library uses event queues for communications between a user application and the VCMTP transfer threads. For example, when a sender application calls the `SendMemoryData()` function, the latter adds a `VcmtpFileTransferEvent` to the *sending request queue*, and immediately returns the control of execution to the user application. The `VcmtpFileTransferEvent` includes information such as filename. The VCMTP sender multicast thread checks the sending request queue for `VcmtpFileTransferEvent`, and for each event present, the sender multicast thread starts multicasting the corresponding file. After a file has been multicast to all receivers, the sender multicast thread inserts a send completion event in the *sending status queue*. The sender application fetches this event later through the `GetNextEvent()` function.

3. `void SendFile(const char* file_name, double retx_timeout_factor)`

The *SendFile* function corresponds to the `API_Send(i)` event for a disk file. Both this function and the *SendMemoryData* function implement the file multicasting and retransmission actions described for the sender FSMs in Section 4.4. The `file_name` parameter is the full path to the disk file. The `retx_timeout_factor` is the retransmission timeout factor described above. As with the `SendMemoryData()` function, when an application calls the *SendFile* function, it also returns immediately after adding a `VcmtpFileTransferEvent` to the sending request queue.

4. `int GetNextEvent(VcmtpMsgTransferEvent* event)`

The *GetNextEvent* supports the asynchronous API, and implements the actions described under "releasing the file back to the user application" in Section 4.4. It fetches the next

event (e.g., `API_File_Done`) from the event queue and returns control of the file back to the VCMTP sender application. The *event* is a pointer to a `VcmtpFileTransferEvent` object. If there are any events in the queue, the *GetNextEvent* function will dequeue the first event, copy its fields into the object pointed to by *event*, and then return SUCCESS. If the event queue is empty, the *GetNextEvent* function returns a `QUEUE_EMPTY` status.

5. `void CloseSender()`

The *CloseSender* function corresponds to the `API_Close_Sender` event and implements the closeout-phase tasks described in Section 4.4. The function closes all connections to the receivers, and releases all the resources of the multicast group.

The receive-side functions include the following.

1. `int JoinGroup(VcmtpReceiverParams config)`

The *JoinGroup* function opens a UDP socket with the multicast group IP address used by the corresponding sender. The `VcmtpReceiverParams` is a data structure that includes configuration parameters for the receiver and the target multicast group. In particular, it includes a `notification_mode` field to specify either *per-file* or *batched* notification mode. If a user application chooses the per-file notification mode, it needs to specify two callback functions in the `VcmtpReceiverParams` data structure. The *first* callback function, named `VcmtpBofFunction()`, is invoked by the VCMTP library when a new BOF message is received. The *second* callback function, `VcmtpRecvCompleteFunction()`, is invoked by the VCMTP library after a file has been successfully received.

On the other hand, if a user application chooses the batched notification mode, it needs to specify the storage locations for received files in `VcmtpReceiverParams`. The parameters include a directory path for received disk files, and a memory buffer for in-memory files. In general, the batched notification mode is more suitable for high-speed transfers, while the per-file notification mode is more suitable when a user application needs fine-grain control in file reception.

2. `void StartReceiver()`

The *StartReceiver* function starts the `receiving thread` and the `retransmission request thread`, as shown in Fig. 4.6. These threads implement the initialization, multicast

file reception and loss recovery actions described for the VCMTP receiver FSMs in Section 4.4. The threads will be configured using the parameters specified in the `JoinGroup()` function.

3. `int GetNextEvent(VcmtpMsgTransferEvent* event)`

The *GetNextEvent* function implements the "notification of the application" actions related to the asynchronous API of VCMTP on the receive side. When the VCMTP `receiving thread` completes receiving a file (either successfully or with failures), it will insert a notification event in the *receiving status queue*. A call to this function results in a fetch of the first event from the queue.

4. `void LeaveGroup()`

The *LeaveGroup* function executes the close-out phase actions described for the VCMTP receiver FSMs in Section 4.4. This function closes the unicast TCP connection to the sender, cleans up all resources, and terminates the data receiving and retransmission threads.

4.6.3 VCMTP Service Manager (VSM) and VCMTP Service Agent (VSA)

The VSM is designed to be a service-management system offering basic configuration management, fault management and performance monitoring for VCMTP. Each VCMTP host runs a VSA, which is connected to the VSM via a TCP connection. The VSA on each host communicates with the VCMTP sender application or VCMTP receiver application executing on that host via Linux pipes.

Configuration management consists of setting parameters such as the retransmission timer and sending rate at the VCMTP sender. The retransmission timer is set to be a factor of the total multicast time for a file. Thus the retransmission timeout is longer for larger files. The sending rate is matched to the virtual-circuit rate. We have not yet integrated the VCMTP software with control-plane software such as the Inter-Domain Controller (IDC) client [5] that is used for making advance reservations for virtual circuits. But this is part of our planned future work.

Fault management actions consist of the VSA (i) sending periodic keep-alive queries to the VCMTP sender/receiver application on its host, and (ii) reporting any failures to the VSM. The VSM then issues commands to the host operating systems to restart the failed VCMTP sender or receiver application.

Performance monitoring actions consist of the VSA (i) receiving information that is logged by the VCMTP receiver application on a per-received file basis, and (ii) either storing this information for future requests from the VSM, or reporting the information at fixed intervals automatically to the VSM. The VCMTP receiver applications were instrumented to log file size, BOF reception time and file completion time for each received file.

For administrators in operational settings, and researchers in experimental settings, the VSM offers a Graphical User Interface (GUI). The GUI consists of a separate sub-window for each VCMTP sender and receiver within its overall management window. A user can enter commands in any of the sub-windows (both Linux system commands and VCMTP-specific commands). For example, a user can restart failed VCMTP sender/receiver applications, initiate file multicasts, and obtain application status information and/or performance measurements from the VSAs.

For the experiments described in the next section, we wrote shell scripts to enable batch-mode execution. The VSM parses the shell scripts and sends appropriate commands in batch-mode to the various hosts. For example, a user can execute a script with commands to multicast all files in a directory and collect performance measurements for each file transfer. Such an automation of the experiment-execution process and measurement collection helped speed up our VCMTP evaluation process, which is described next.

4.7 VCMTP Evaluation

The VCMTP prototype was tested on the University of Utah's Emulab [22]. The underlying network service conformed to the left-hand side protocol stack of Fig. 4.5. Since all the hosts used in our experiments were in the same Ethernet-switched network, packet forwarding was based on MAC addresses. The destination MAC address is automatically derived from the Class-D (multicast) IP address, and multicast VCMTP frames were received by the hosts whose VCMTP receivers were configured to receive packets with these destination MAC and IP addresses. The IP layer is involved only at the hosts; it was used to exploit the easy-to-use socket API as noted in Section 4.5.

Two experiments were executed. The goals of Experiment 1 were to compare VCMTP with parallel unicast TCP connections (one per receiver) for a single large file transfer in a no-loss environment, and to validate an analytical model. The goal of Experiment 2 was to study the performance of VCMTP when serving continuously generated files as in the Unidata IDD project. In both experiments, the VCMTP block size was set to 1428B (bytes), so that with the 32B VCMTP header, 20B TCP header[1], 20B IP header, the packet would fit within the 1500B maximum transmission unit length of an Ethernet frame.

4.7.1 VCMTP vs. parallel unicast TCP

The first experiment validated a model for large-file (size F) transfer time across no-loss, low-RTT (round-trip time) paths to n identical receivers. The transfer times using parallel unicast TCP connections and VCMTP, respectively, are given by:

$$\begin{aligned} T_{tcp} &= max\{\frac{nF}{l_s}, \frac{nF}{c_s}, \frac{F}{l_r}, \frac{F}{c_r}\} \\ T_{vcmtp} &= max\{\frac{F}{l_s}, \frac{F}{c_s}, \frac{F}{l_r}, \frac{F}{c_r}\} \end{aligned} \tag{4.1}$$

where l_s, l_r, c_s, and c_r correspond to the access link rates at the sender and receiver(s), and server capacities (processing rates and memory/disk access rates) at the sender and receiver(s), respectively. In T_{tcp}, the first two terms model the case when the sender bottleneck rate (l_s or c_s) is less than n times the receiver bottleneck rate, while the next two terms model the case when the sender bottleneck rate is at least n times that of the receiver bottleneck rate in which case the sender can sustain n simultaneous transfers, each at the receiver bottleneck rate. The transfer time under VCMTP is independent of the number of receivers under the no-loss assumption.

As an example, consider the delivery of a 1 GB file to multiple receivers. Assume the following rates: $l_s = 1$ Gbps, $l_r = 100$ Mbps, and $c_s = c_r = 10$ Gbps. Fig. 4.7 shows the maximum transfer time to send the file to all receivers according to the above model. When the total number of receivers is less than or equal to 10 (l_s/l_r), the bottleneck is on the receiving side, and hence the same transfer times incurred using both VCMTP and multiple

[1]Since retransmissions are carried in TCP segments, this larger header size was considered instead of the 8B UDP header used in the multicast packets.

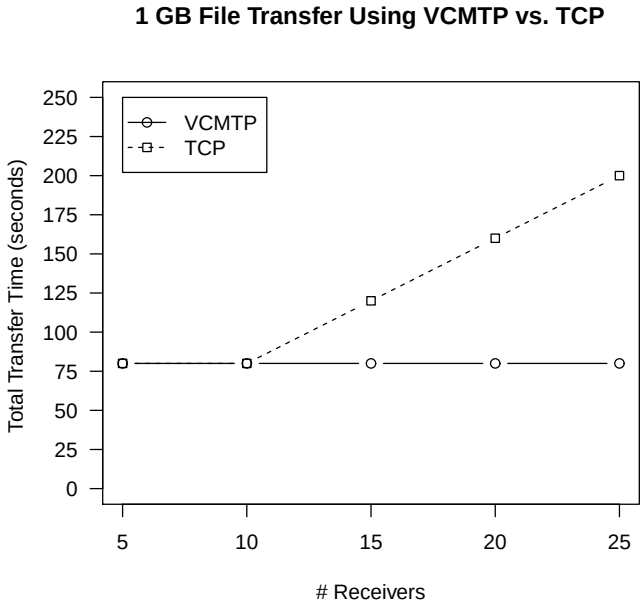

Figure 4.7: Example plot for equation (4.1): F: 1 GB, $l_s = 1$ Gbps, l_r= 100 Mbps, $c_s = c_r =$ 10 Gbps

unicast TCP connections are the same. However, when the number of receivers is larger than 10, the sender access link rate (l_s) becomes the bottleneck. While the VCMTP delay remains unchanged, the maximum transfer time increases with the number of receivers for unicast TCP. To achieve the VCMTP transfer time with unicast TCP, depending on the bottleneck, either the sender access link rate needs to be increased, or multiple sending hosts are required. Either way, for a given performance objective, the resource requirements with unicast TCP connections will typically be larger than that of VCMTP.

The results of the first experiment, in which the model assumptions held true, are described below. The Linux `tc` facility was used to limit rates at the sender and receivers. This experiment was executed on Emulab hosts with 1 Gbps NICs, Intel quad-core Xeon CPU, 12 GB memory and commodity disks. In the first run, the maximum rate was set to 600 Mbps at the sender, and 800 Mbps at the receivers. In the second run, the maximum rates were 600 Mbps and 150 Mbps at the sender and receivers, respectively. The results for the first run are shown in Fig. 4.8, and the results for the second run are shown in Fig. 4.9.

In the first run, the send rate was the bottleneck as it is lower than the receive rate. For VCMTP, it took about 15 seconds to deliver the 1 GB file. Furthermore, the total transfer

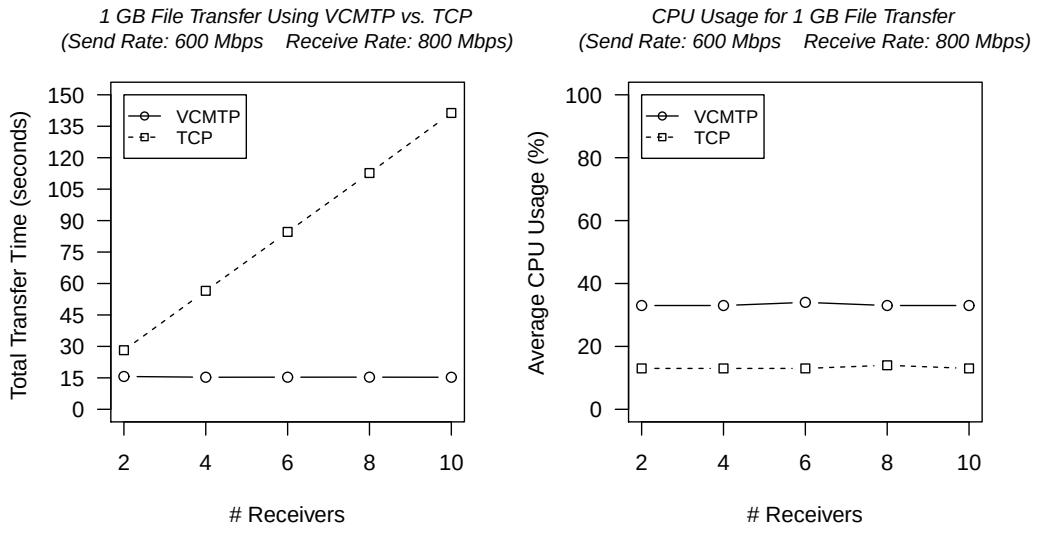

Figure 4.8: A comparison of performance and resource usage between VCMTP and unicast TCP (Run 1)

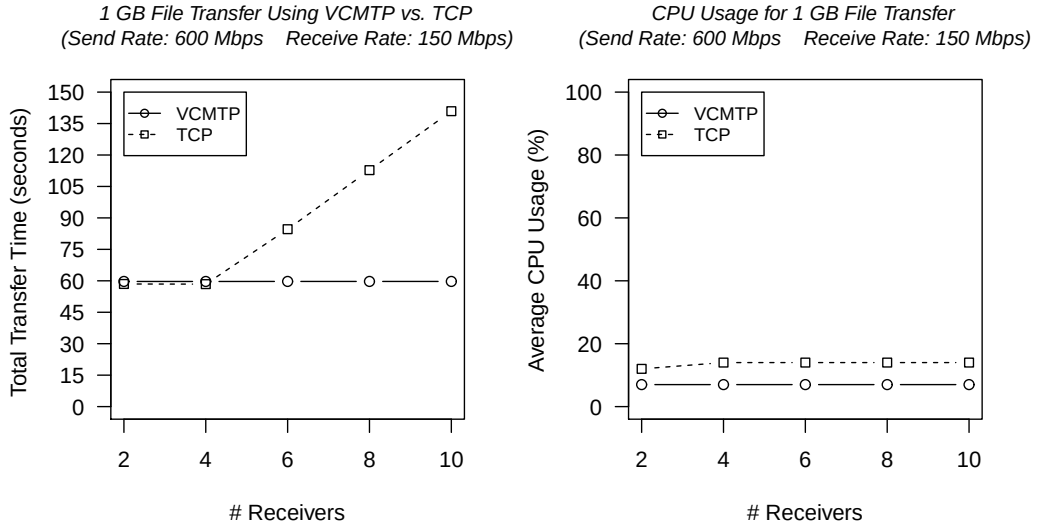

Figure 4.9: A comparison of performance and resource usage between VCMTP and unicast TCP (Run 2)

time did not increase with the number of receivers. For the unicast TCP scheme, the total transfer time grew proportionally with the number of receivers, and was always larger than the transfer time with VCMTP. On the other hand, the sender CPU usage with VCMTP was higher than with unicast TCP connections (33% vs. 13%). To sustain the high sending rate of 600 Mbps required significant CPU resources for the particular Emulab hosts used

in the experiment. Since the current VCMTP prototype is implemented as a user-space library, it involves an extra data copy from a user memory buffer into a VCMTP packet buffer for every data block before the VCMTP packet is sent to the UDP socket. Hence more CPU resources are required for sending the same amount of data using VCMTP when compared to using unicast TCP connections. With other potential implementations that are envisioned for VCMTP (see Section 4.5), such as RDMA-based implementations, CPU resource usage can be reduced for VCMTP.

In the second run, the sending rate is four times the receiving rate. When the number of receivers is equal to or less than 4, the transfer times under VCMTP and parallel unicast TCP connections are almost the same. But with larger numbers of receivers, the transfer times are lower under VCMTP. This is consistent with the model of (4.1), and demonstrates the basic advantage of multicast communications. For example, with 6 receivers, the time to transfer a 1 GB file using TCP is computed analytically to be 80 sec (using (4.1) and a send rate of 600 Mbps as indicated in Fig. 4.9), while the experimental measurement was 86.2 sec, and the analytical and experimental results for T_{vcmtp} are 53 sec and 59.9 sec, respectively. The spread between TCP delay and VCMTP delay increases with the number of receivers as seen in Fig. 4.9. In this run, CPU usage with VCMTP is lower than when with unicast TCP connections as seen in Fig. 4.9. At the lower sending rate of 150 Mbps, the effect of the sender CPU having to send multiple copies with unicast TCP connections is evident with the higher CPU time.

4.7.2 Evaluation of VCMTP with continuous file transfers

Since files are generated and distributed continuously in the IDD project, the second experiment was designed to test VCMTP performance in this setting. Furthermore, unlike the no-loss environment of the first experiment, in this second experiment, losses were injected deliberately. The VCMTP solution was evaluated on two metrics: throughput of "fast" receivers and robustness of "slow receivers," the distinction being that random packet losses were injected at slow receivers but not at the fast receivers. For this experiment, low-end Emulab hosts with 100 Mbps NICs, 850MHz Intel Pentium III processor and 512

Table 4.9: Experiment 2 results (continuous file transfers)

	ρ	Loss Rate	Timeout Factor	Robustness R as a percentage (SD)			Throughput Γ in Mbps (SD)		
				$n = 10$	$n = 20$	$n = 30$	$n = 10$	$n = 20$	$n = 30$
Config. 1	0.4	5%	10	86.3 (2.3)	83.4 (0.4)	81.4 (0.7)	92.8 (1.1)	90.5 (0.7)	86.9 (0.7)
Config. 2	0.4	5%	50	98.9 (0.8)	98.1 (0.7)	97.5 (0.3)	92.7 (0.8)	89.2 (0.7)	85.8 (0.8)
Config. 3	0.4	10%	10	79.9 (2.3)	74.0 (1.2)	65.2 (0.5)	90.5 (0.8)	85.1 (0.6)	82.3 (1.0)
Config. 4	0.4	10%	50	96.2 (1.9)	94.0 (1.6)	85.0 (1.3)	89.9 (0.5)	84.9 (0.7)	80.3 (1.6)
Config. 5	0.8	5%	10	35.0 (4.7)	25.9 (3.5)	15.7 (3.8)	93.9 (1.0)	92.0 (0.5)	91.7 (0.8)
Config. 6	0.8	5%	50	68.2 (6.0)	60.3 (5.5)	55.8 (6.0)	92.1 (0.8)	88.9 (2.7)	88.4 (1.5)
Config. 7	0.8	10%	10	22.9 (4.4)	11.6 (3.2)	10.6 (1.5)	93.6 (0.3)	91.1 (0.6)	89.2 (1.2)
Config. 8	0.8	10%	50	56.3 (9.5)	53.3 (1.7)	50.2 (3.6)	92.7 (1.5)	88.1 (1.8)	85.0 (1.6)

MB memory, were selected as it was easier to access large numbers of these hosts (when compared to the hosts with 1 Gbps NICs).

For each experimental run, a sample of 500 files was generated assuming the file arrival process to be Poisson with a rate of 25 files/sec, and that file sizes fit the Pareto distribution [75]. The shape parameter α was chosen to be 2, and two values of the scale (minimum-value) parameter k, 100 KB and 200 KB, were used. Traffic load ρ is the mean service time multiplied by call arrival rate. Since the mean for the Pareto distribution is $\alpha k/(\alpha - 1)$ for $\alpha > 1$, the traffic load corresponding to k values of 100 and 200 KB are 0.4 and 0.8, respectively, assuming the full link rate of 100 Mbps.

Experiments were run for eight configurations defined by two values of ρ, two values of the packet loss rate at slow receivers, and two values of the retransmission timeout factor (see Table 4.9). Packet losses were injected at random according to a set packet loss rate rate at a fixed fraction (40%) of the receivers (referred to as the "slow receivers"). The sender-side timer for the retransmission phase of a file was set to be a factor of the total multicast time for that file. Since file sizes are small, this factor was set to be larger than one; the specific values chosen were 10 and 50.

Table 4.9 shows the results for experiments with 10, 20 and 30 receivers. For each configuration, 5 runs were executed. Means and standard deviations, computed from measurements taken in the 5 runs, are shown for robustness and throughput in Table 4.9. Robustness, R, and throughput, Γ, are defined as

$$\begin{aligned} R =& \frac{\sum_{j=1}^{n_s} \sum_{i=1}^{m} S_{ij}}{m \times n_s} \\ \Gamma =& \frac{\sum_{j=1}^{n_f} \sum_{i=1}^{m'} (\frac{F_i}{T_{ij}})}{m' \times n_f} \\ n =& n_s + n_f \end{aligned} \tag{4.2}$$

where S_{ij} is an indicator variable that is set to 1 if file i was successfully received by receiver j and 0 otherwise (recall that when the sender-side retransmission timer for a particular file times out, all subsequent retransmissions requests for blocks of that file will be rejected), m is 500 (number of files in a run), m' is the number of files larger than 500 KB (to reduce the timer precision error, measurements for smaller files were dropped), n_s is the number of slow receivers, n_f is the number of fast receivers, F_i is the size of file i, T_{ij} is the time taken for the reception of file i at receiver j.

The robustness and throughput results of Table 4.9 are visualized in Fig. 4.10 and Fig. 4.11. The main observations from the results are as follows. *First*, the number of receivers affects both robustness and throughput in this continuously-arriving-files scenario. Since the ratio of slow receivers is fixed at 40%, the number of slow receivers increases when the total number of receivers n is increased. Consequently, a larger number of retransmission threads compete for sender-side computing and network resources, and hence robustness decreases. For the same reason, throughput is also impacted adversely. *Second*, as traffic intensity ρ increases or loss rate increases, both robustness and throughput decrease for the same resource contention reason. To meet certain robustness and throughput objectives, the server capacity at the sender should be selected based on these three parameters: number of receivers, traffic intensity, and measured loss rate. Individual receivers that suffer high loss rates should be moved to a lower-rate virtual circuit if available to reduce their influence on the throughput of other receivers. *Finally*, the sending-side retransmission timeout factor offers administrators a knob for trading off robustness against throughput for a given setting of a particular sender, set of receivers and traffic intensity. By increasing the retransmission timeout factor, robustness can be increased at a cost to throughput. For example, compare the results for Configurations 1 and 2 with $n = 30$ in Table 4.9. For a small drop in throughput (86.9 to 85.8 Mbps on average), robustness can be increased significantly from 81.4 to 97.5% (on average) just by increasing the retransmission timeout factor from 10 to 50. As *another example*, consider Configuration 7 of Table 4.9 with a 0.8 traffic intensity and 10% packet loss rate. In this case, robustness is very low at 10.6% when there are 30 receivers. Increasing the retransmission timeout factor to 50 (moving to Configuration 8) increases robustness, but only to 50.2%. Since such a low robustness is likely to be unacceptable,

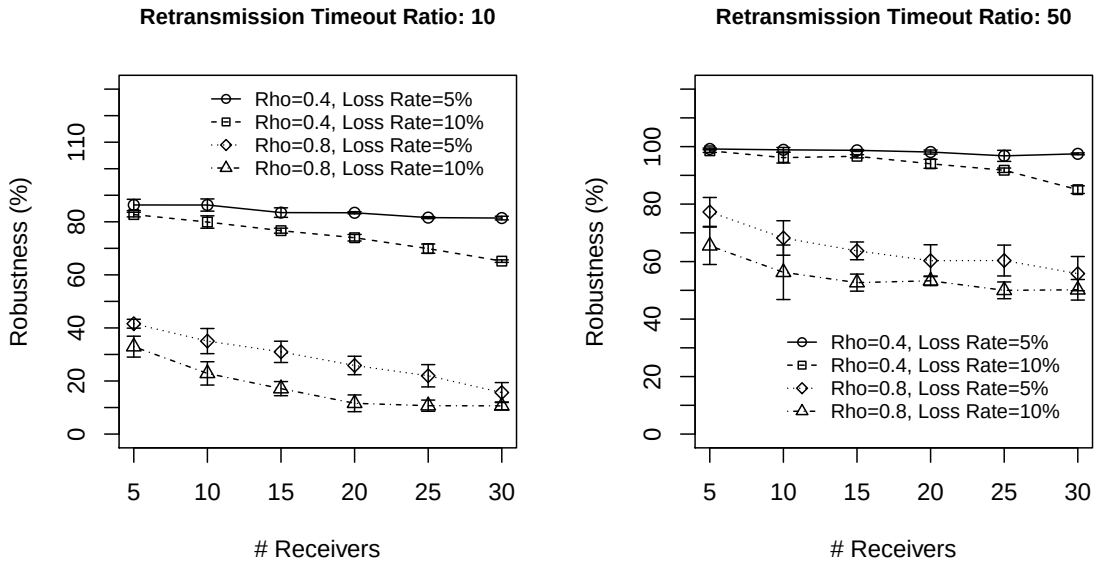

Figure 4.10: Average robustness of slow receivers in continuous file transfer

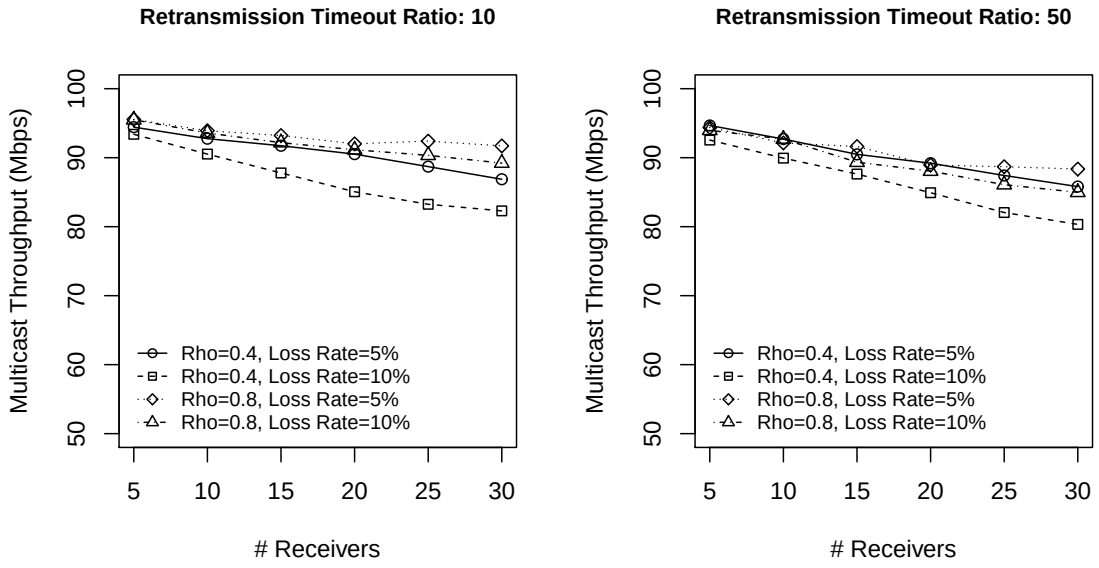

Figure 4.11: Average throughput of fast receivers in continuous file transfer

either some of the slow receivers should be moved to a lower-rate virtual circuit to reduce their packet loss rate, or more server/network capacity is required at the sender.

To reduce the negative impact of slow receivers on the throughput of fast receivers, in a final set of runs, we experimented with using priority scheduling; higher priority was

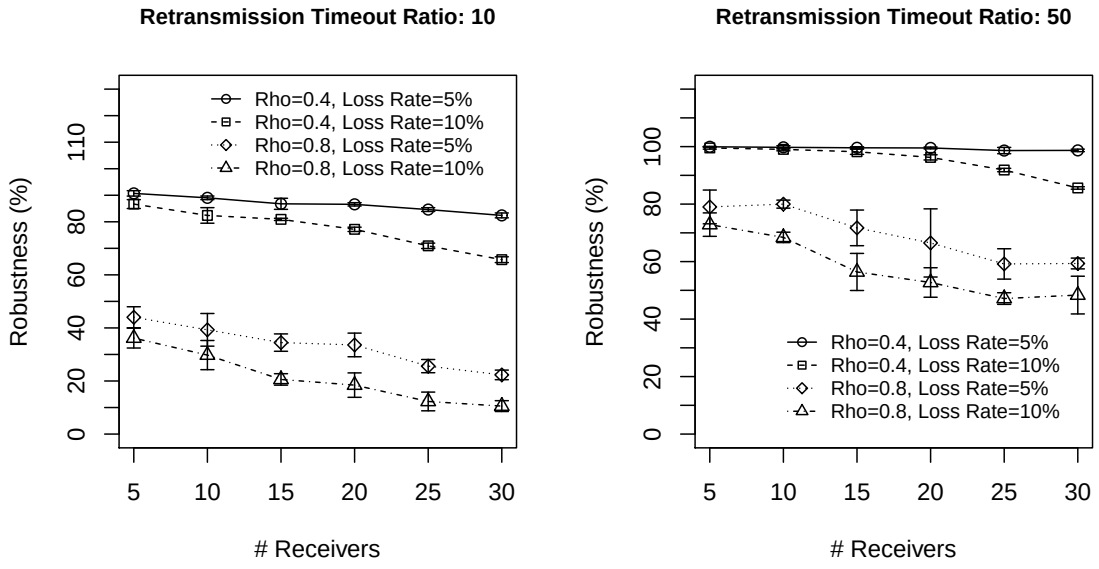

Figure 4.12: Average robustness of slow receivers in continuous file transfer (high priority mode)

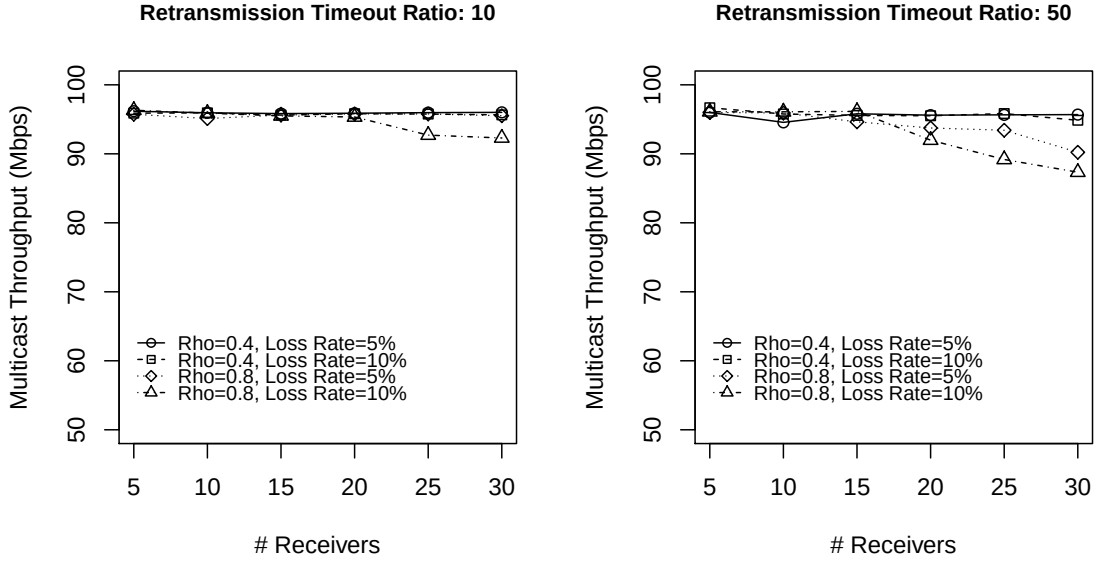

Figure 4.13: Average throughput of fast receivers in continuous file transfer (high priority mode)

assigned to the multicast threads. To study the effect of priority scheduling, we ran the same experiments as above except that the main multicast threads at the sender and receivers were executed in SCHED_RR mode, which is a soft real-time high-priority mode supported

in Linux. The robustness of slow receivers is shown in Fig. 4.12, and the throughput experienced by fast receivers is shown in Fig. 4.13. While the robustness of slow receivers drops slightly, the average throughput of fast receivers stayed high at around 95 Mbps even as the number of receivers was increased. The slopes of the throughput plots are smaller in Fig. 4.13 when the multicast threads are run under high priority when compared to the slopes of throughput plots without priority scheduling (Fig. 4.11).

4.8 Integration of VCMTP with LDM

The VCMTP library was integrated with the LDM software and tested. This integration allows the IDD feedtypes distributed by LDM to be multicast to receivers using VCMTP. Fig. 4.14 shows a high level overview of the integration. The *integrated LDM-VCMTP server* executes a downstream LDM process and a VCMTP sender process. The downstream LDM process is configured to subscribe to specific feedtypes from a specific upstream LDM server (e.g., one of the UCAR LDM servers). Upon receiving a new data product (i.e., a new data file) from the upstream LDM server, the downstream LDM process passes the data product to the VCMTP sender application. The VCMTP sender then multicasts the new data product to all VCMTP receivers in the multicast group.

A more detailed illustration of the integration process is shown in Fig. 4.15. The LDM configuration file *ldmd.conf* lists the feedtypes subscribed to by the downstream LDM downstream process. When the downstream LDM process starts receiving new data products from the upstream LDM server, it uses a utility called *pqact*, which is part of the LDM software distribution. The purpose of this utility is to enable a user to request a specific set of actions to be executed on each received data product. Execution of a decoder program was indicated as the action to be performed on received products. This program name was specified in the *pqact.conf* file, which is the configuration file for the `pqact` tool. The decoder program obtains the metadata for each received data product, stores the data product to disk, and then passes the name of the data product to the VCMTP sender process. The VCMTP sender process then multicasts the received file to all VCMTP receivers.

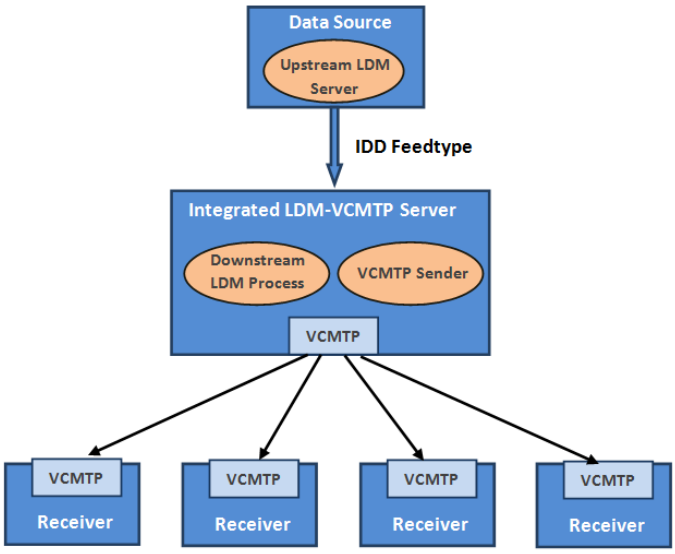

Figure 4.14: Integration of VCMTP with LDM: Overview

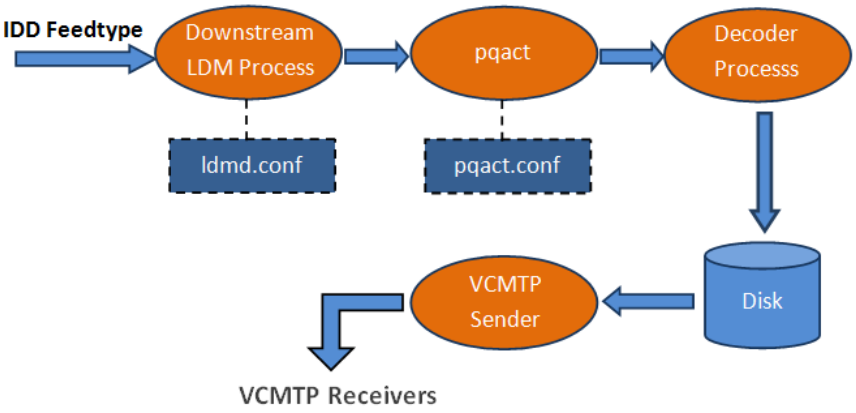

Figure 4.15: Integration of VCMTP with LDM: Detailed Steps

Next, the integrated LDM-VCMTP software was tested. Our plan was to use the U. Utah Emulab testbed to run the integrated LDM-VCMTP server while having it subscribe to IDD feedtypes distributed by a UCAR upstream LDM server. However, the Emulab firewall blocks incoming traffic to most well-known ports for security reasons, and LDM uses the well-known port number 388 for its TCP connections.

To work around this issue, we introduced a "relay" node on the path of the transfer from a UCAR upstream LDM server to an Emulab host that ran the VCMTP sender application. The experimental setup is shown in Fig. 4.16. The downstream LDM process was executed on a relay node named `Zelda2`, which is located at UVA. It connected to one

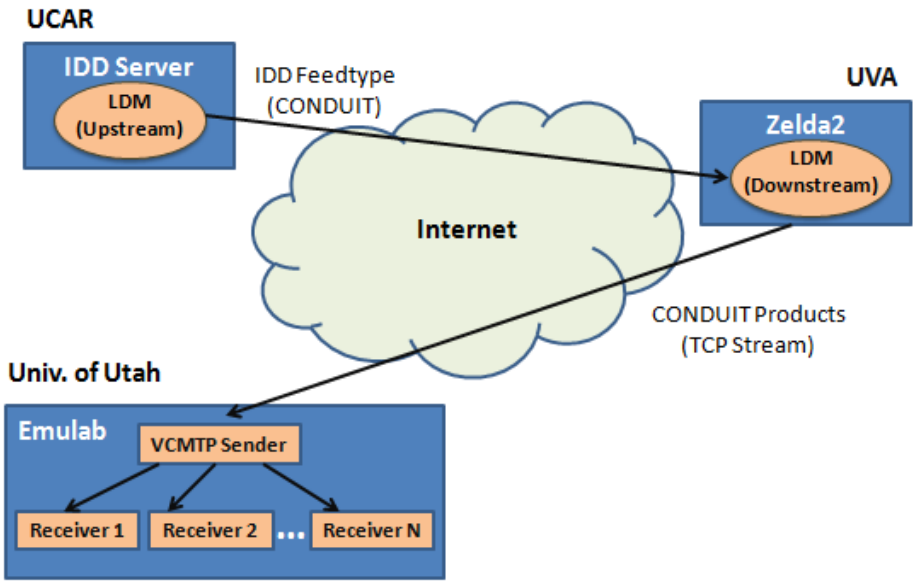

Figure 4.16: Integration of VCMTP with LDM: Experimental setup

of the UCAR upstream LDM servers to receive the CONDUIT feedtype. For each received data product, the downstream LDM process invoked the decoder program specified in the `pqact` configuration file. The decoder program then sent each received data product to the VCMTP sender application through a TCP connection, using a temporary port number that is higher than the range allocated to well-known ports. On the host in Emulab, the VCMTP sender application received files over the TCP connection from `Zelda2`, and multicast the files to a set of VCMTP receivers (which were executed on other Emulab hosts).

We conducted experiments with up to 25 VCMTP receivers. With a moderate proportion of slow receivers and artificially introduced loss rates, all the receivers could successfully receive all data products from the CONDUIT feedtype. Under extreme retransmission settings (e.g., 50% packet loss rates at 20% of all receivers), the slow receivers were not able to keep up with the sending data rate. In such cases, the VCMTP sender cut off the slow receivers after the expiration of the retransmission timer, causing the robustness measure to drop.

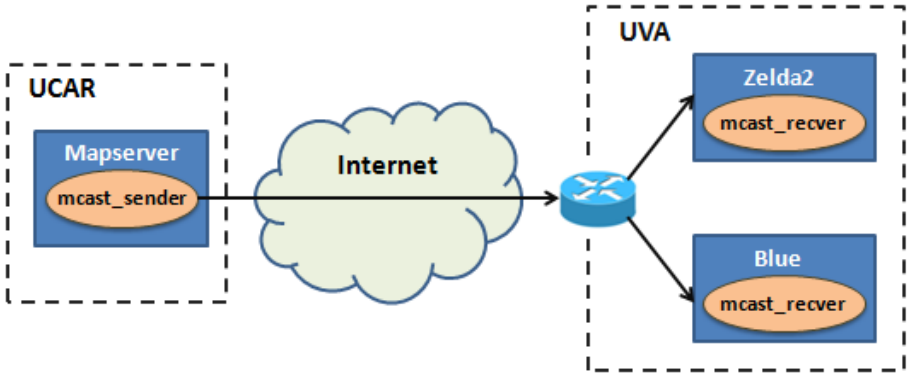

Figure 4.17: Wide Area IP Multicast Experiment between UCAR and UVA

4.9 Inter-domain IP multicast test

An experiment was conducted to test the feasibility of using inter-domain IP multicast between UVA and UCAR. There are five domains between UVA and UCAR. These include UVA, UVA's regional REN called Natcap, Internet2 core network, UCAR's regional REN called FRGP, and UCAR. All these networks support IP-multicast.

A multicast test program, consisting of an `mcast_sender` and an `mcast_recver`, was developed for this experiment. As shown in Fig. 4.17, one UCAR host, *Mapserver*, and two UVA hosts on different subnets, *Zelda2* and *Blue*, were used for the test. During execution, the *mcast_sender* process on Mapserver would periodically multicast a packet including an ASCII string message to a specific multicast IP group address. The *mcast_recver* process on both Zelda2 and Blue listened on a UDP socket to receive multicast packets from Mapserver, and parsed the received message string.

In this UCAR-to-UVA IP multicast test, both Zelda2 and Blue could successfully receive multicast packets from Mapserver and correctly parse the message string. Four lessons were learned from this experiment. *First*, all the routers on the path between the sender and the receivers need to be configured to run IP multicast. IP multicast requires routing protocols such as MSDP and PIM-SM, as mentioned in Section 1.2.1, to spread reachability information for the multicast IP address, and Internet Group Management Protocol (IGMP) [15] for hosts to join and leave multicast groups. *Second*, an appropriate multicast IP address needs to be selected for the test. Within the 224.0.0.0/4 class D address space allocated

for multicast, many addresses have been assigned by IANA to a registered organizations or specific multicast applications [76]. When sending multicast packets over the Internet, it is important not to use a multicast IP address that is in conflict with these assigned multicast addresses. Therefore, the IETF RFC 2770 [77] specified that the 233.0.0.0/8 address block, also called the GLOP block, should be used by individual Autonomous Systems (AS) with special rules. To choose a multicast address for an application, an AS should embed its 16-bit AS Number (ASN) in the intermediate two bytes of the 233.0.0.0/8 address, so that this address does not conflict with multicast applications originated from other ASes. Furthermore, the last byte of the address is used to differentiate applications within the same AS. A multicast IP address constructed by following these rules is safe to use in sending multicast packets across the Internet. *Third*, a program written for wide-area IP multicast needs to explicitly set the Time-To-Live (TTL) value in packet headers. This is because in the kernel implementation for IP multicast on Unix-like operating systems, the default TTL value is set to 1 to prevent multicast packets from being forwarded beyond the local subnet. The TTL value can be explicitly configured to a larger value for a specific UDP socket through the `setsockopt()` system call in Unix-like operating systems. *Finally*, default firewall settings in some Linux versions block multicast packets. The `iptables` file was edited to accept packets destined to the particular GLOP multicast IP address, and the `iptables` service was restarted to allow the multicast traffic.

4.10 Conclusions

We designed, prototyped, and evaluated a new reliable virtual circuit multicast transport protocol called VCMTPv2, which is capable of supporting continuous file transfers. An analytical model was developed for TCP and VCMTPv2 based single-file distribution, and validated experimentally. For continuously generated files, a key design aspect of VCMTPv2 is the tradeoff between file-delivery throughput for fast receivers and robustness for slow receivers. A VCMTPv2 configurable parameter called retransmission timeout factor can be adjusted to tradeoff these two metrics. For a traffic load of 0.4, and a multicast group with 30 receivers, robustness was increased significantly from 81.4 to 97.5% by increasing the

retransmission timeout factor from 10 to 50. The corresponding drop in average throughput for fast receivers was small (86.9 to 85.8 Mbps).

Chapter 5

A Less-Is-More Architecture (LIMA) for A Future Internet

5.1 Introduction

In a 2007 Internet Architecture Board (IAB) Workshop report [18] it was observed that the global routing table is "growing at an increasing and potentially alarming rate." A number of activities in the IETF, such as the locator-identifier split protocol (LISP) [20], arose as a result of this report. Early work by Bu, Gao and Towsley [21] showed the various causes of this high growth rate. The growth rate of the global routing table is accompanied with an increased BGP update rate. Therefore, both memory capacity/speeds and CPU power in IP routers have to grow at significant rates. A contrarian view is espoused in [19], which states that the 17% exponential yearly growth rate is in step with improvements in memory technologies. The above argument notwithstanding, the current emphasis on sustainable design and the anticipated growth from the Internet of Things (IoT) offer sufficient motivation for addressing this problem.

In this work, a new addressing and routing design called the Less-Is-More Architecture (LIMA) is proposed as a clean-slate inter-domain solution for a future Internet. Unlike recently proposed identifier-locator split solutions, LIMA uses just (topological) location-independent names and location-dependent addresses. The feasibility of using a policy

Table 5.1: Classification of addressing and routing mechanisms

Routing Policy / Address Assignment	**Stub reachability permitted in global routing tables**	**Stub reachability not permitted in global routing tables**
Provider Independent (PI) addresses permitted for stubs	Today's Internet (IPv4 and IPv6)	eFIT [78], ILNP [79], LISP [20], HIP [80], MILSA [81], FARA [82], NPTv6 [83], Shim6 [84], TurfNet [85]
Only Provider Aggregatable (PA) addresses for stubs	None	LIMA (our solution)

combination of restricting stubs to provider-aggregatable addressing only, and disallowing stub-level reachability from being propagated into the global routing tables, is studied. This policy combination results in significantly smaller global routing tables but creates the address renumbering problem (when stubs change providers). A solution for this problem is presented to ensure seamless (no loss of connectivity) transitions when a stub changes one of its providers. This address renumbering solution leverages multi-addressing and a novel concept called "dismembered addressing" proposed for LIMA. Other aspects impacted by the policy combination, such as multihoming and traffic engineering, are also discussed.

The *hypothesis* of LIMA is that it is feasible to adopt (i) an address assignment policy in which stubs (enterprises) are restricted to Provider- Aggregatable (PA) addressing, and (ii) a routing policy in which stub-level reachability information is not propagated into the global routing tables, in conjunction with control-plane solutions to solve the address renumbering and multihoming problems created by this policy combination.

After reviewing other work in Section 5.2, and presenting the LIMA addressing/routing design in Section 5.3, our solutions to four challenges are presented in Section 5.4. Section 5.5 describes LIMA components. Section 5.6 presents preliminary analysis of the benefits and costs of LIMA. Finally, section 5.7 concludes our work.

5.2 Related work

We categorize recent solutions to the global routing table problem into four classes as shown in Table 5.1. What allows the solutions in the top-right cell of the table to have the advantage of limiting the size of the global routing table without having to change applications and transport protocols is the introduction of a third parameter called an `identifier`. While ideally, location-independent `names` and topological location-dependent `addresses` (also

called `locators`) should be sufficient, because applications embed IP addresses, and TCP does not have built-in support for address migration, this third parameter, identifier, is used to serve these two roles while locators are used for global routing. Furthermore, without the identifier, all host and router interfaces of stubs would need to be renumbered when stubs change their providers if all that exist are locators. While this concept of locator-identifier split has the above advantages, its disadvantages are that additional actions are required for every data-plane packet, plus control-plane mapping is required between identifiers and locators. Examples of data-plane overhead include address translations in NPTv6 [86] and DRUID [87], and IP-in-IP tunneling in LISP [20] and shim6 [84]. While the control-plane overhead of mapping from identifiers to locators may seem trivial from a bandwidth perspective, it could add operational costs in troubleshooting misconfigurations. In contrast, the LIMA solution does not require identifiers; it uses just names and addresses.

LIMA's hierarchical addressing solution is the opposite of schemes that propose flat addressing, such as Routing on Flat Labels (ROFL) [88, 89]. The latter notes that besides the path stretch problem, hierarchical schemes complicate management, mobility and multihoming. It is precisely these challenges that are addressed in this work.

Why our solution is "less-is-more" We found that the LIMA policy combination as stated in Section 5.1, and the dismembered addressing concept, can be tested with IPv6 as the network layer (NL), as it supports a key requirement of our design, multiaddressing, whereby an interface can be addressed with multiple addresses. LIMA is "less-is-more" because relative to the current-day IPv6 solution, it eliminates ARP and longest-prefix matching, relative to LISP and NPTv6, no tunneling or translations are required, and relative to Named Data Networking (NDN) [90], which requires name based per-packet lookups, and Accountable Internet Protocol (AIP) [19], which uses 160-bit addresses, LIMA requires lookups of much smaller (e.g., 32 bit) fixed-length dismembered address components. To avoid these more complex per-packet processing actions, LIMA pays a penalty in requiring additional management, but only for the relatively rare events of address renumbering and access link failures for multihomed stubs. Also, while the network layer is left untouched, almost all other aspects such as applications, socket interface, transport layer protocols,

DHCPv6, DNS, and BGP, require changes to support LIMA to handle the four challenges it creates. But these changes can be evolved into the current-day Internet.

5.3 LIMA Routing and Addressing

LIMA is a design for inter-domain communications. Hence LIMA routers, which are envisioned to be IPv6 routers with some additional LIMA control-plane functionality, are designed for use as *border routers*, not internal routers. After presenting the LIMA addressing and routing schemes, we describe two examples of intra-stub networks.

5.3.1 Addressing

Simply put, LIMA addressing is a hierarchical scheme. It is similar to IPv4/IPv6 addressing in that there is a global routing prefix and an interface identifier. LIMA differs in the following ways:

1. it uses autonomous system (AS) numbers as prefixes
2. globally unique AS numbers are assigned only to providers, and stubs[1] are assigned provider-local AS numbers by their providers; this approach is unlike in IP where PI addressing is allowed for stubs
3. it reuses the address used in intra-domain networking as the interface identifier; this reuse is comparable to the IPv6 option in which MAC addresses are expanded to the EUI-64 format and used in the interface-ID field.

The first concept has been proposed by others such as [19]. The number of prefixes in today's IPv4 Internet (˜335K) is much higher than the number of provider AS numbers (˜6185, as reported in a later section). The second concept appears in other work, such as eFIT [78], which distinguishes user networks from provider networks, though as mentioned in Section 3.1, eFIT requires an "identifier" while LIMA does not. The third concept is in keeping with the less-is-more theme in that it eliminates ARP and associated security threats. Even though ARP is eliminated, we propose the use of DHCPv6 in LIMA and

[1] A "stub" is an enterprise such as a business, university or governmental agency.

not IPv6 Stateless Address Auto Configuration (SLAAC) [91] for security reasons. MAC addresses are sometimes viewed as confidential (e.g., they reveal the NIC manufacturer's name), but this can be circumvented with dynamically assigned MAC addresses. To prevent spoofing, source address filters can be added.

While the above description of LIMA addressing offers readers a differential view relative to IP addressing, the basic principle is to *dismember the address* into three distinct components: globally unique `provider AS number`, provider-local `stub AS number`, and stub-local `intra-domain address (IDA)`. These components can be mapped on to the IPv6 address structure, by assigning, for example, the top 4 bytes to the globally unique provider AS number, the next 4 bytes to the provider-local stub AS number, and the bottom 8 bytes to carry the IDA. DHCPv6 and DNS will be modified to support this concept; the terms *L-DHCPv6* and *L-DNS* are used to represent the LIMA versions of these protocols. Providers can be assigned multiple AS numbers, and a provider can allocate multiple AS numbers to its stubs.

On the question of how to determine whether an organization is a stub or a provider, consider Content Delivery Network (CDN) providers, such as Google, Yahoo, Microsoft and Akamai. These providers do not typically offer IP transit service, and yet their domains are connected to many other stub domains. Similarly, some consumer ISPs do not provide transit service in that all packets either originate or terminate from their customers. One *answer* to this question is to use the number of inter-domain links from a given AS as the determinant for classifying an AS as a provider or a stub. This is an issue for further study.

5.3.2 LIMA routing

LIMA routing differs from IP routing as follows. In IP, Tier-1 routing tables include information about PI and multihomed PA stubs, and routers implement longest prefix matching. In LIMA, Tier-1 routers will not have any information about stubs, and longest-prefix matching is eliminated. Instead, in LIMA, separate routing tables are maintained for the provider AS number and stub AS number in *provider network border routers*. Parallel lookups of both routing tables for fast operation can be implemented in hardware. For datagrams reaching the destination provider network, the stub AS number table is consulted

to determine the border router within that provider network to which the datagram should be forwarded. *Stub border routers* consult the provider AS number routing table for outgoing datagrams, and the stub AS number routing table for incoming datagrams in stubs with multiple stub AS numbers.

5.3.3 Intra-stub network examples

We illustrate the concept of dismembered addressing with two intra-stub examples: a flat Ethernet switched network, and a hierarchical private IPv4 routed network. In the Ethernet case, IDAs are MAC addresses. As mentioned earlier, we propose that dynamic MAC addresses be assigned by the L-DHCPv6 server for strong asset management. In addition, the L-DHCPv6 server sends the {provider AS number, stub AS number} pairs to endpoints during initialization. L-DHCPv6 clients concatenate these dismembered components to create their IPv6 addresses for interface configuration. A host in a multihomed stub with a single Ethernet interface will have a single IDA (which is the MAC address), but multiple IPv6 addresses created by concatenating multiple {provider AS number, stub AS number} pairs with the MAC address. Similarly, a stub with a hierarchical private IPv4 routed network will use private IPv4 addresses as IDAs, which will be carried in the interface ID field of IPv6 addresses.

Each host interface will be assigned one (canonical) name corresponding to the IDA. Additionally a name can be assigned to a host, and mapped to the multiple canonical names of its interfaces. The L-DNS server will store a mapping between the name and the IDA for all endpoints, and a single entry mapping the organization name to its {provider AS number, stub AS number} pairs. A fully-qualified domain name for a host is created by concatenating the organization name with the host's name. L-DNS queries and secure dynamic DNS updates will support the dismembered address structure, and can be for just the stub name or for a particular endpoint name.

5.4 Solutions for the four challenges

5.4.1 Address renumbering

To achieve fully automated renumbering, LIMA adopts mechanisms from [92, 93], and classifies them as: (i) host-related, (ii) DNS-related, and (iii) router-related.

Host-related The key features are: (a) multiaddressing, (b) Name Based Sockets (NBS) [94], (c) the LIMA concept of address dismemberment, and (d) the use of SCTP [95] or MPTCP [96]. *Multiaddressing* is key to address renumbering with `zero downtime` as the stub can maintain its link to the old provider for a day or two while executing all the steps needed for address renumbering. Next, restricting applications to only use domain names to prevent any address caching within applications is feasible with *NBS*. Applications only store and deal with names, while the NBS layer translates names to addresses. Third, address dismemberment allows for a *broadcast push* of the new {provider AS number, stub AS number} to all endpoints within the stub where L-DHCPv6 clients receive the pushed parameters, create IPv6 addresses by concatenating with the unchanging IDAs, and configure their interfaces. Finally, as most TCP connections are short-lived, and the access link to the old provider needs to be maintained for some time to allow DNS cached entries to expire (as explained next), these connections will terminate before the old provider based addresses are no longer usable. But given that there are some long-lived TCP connections, a transport protocol that supports dynamic address reconfiguration is preferred to one that requires reconnections. We propose the use of *SCTP* or *MPTCP* both of which support this feature.

DNS-related Next, consider `DNS updates` and `DNS caching`. Today's DNS servers maintain a full IP address record for each domain name. In LIMA, we propose organizing this database to hold one entry that maps an organization name (e.g., virginia.edu) to one or more {provider AS number, stub AS number} pairs, and then have individual records that map host names to IDAs. Such a structure would make it easier to handle a provider change. To allow *cached* DNS records in other stubs and providers to work, stubs will maintain the link to the old provider for the maximum time-to-live value.

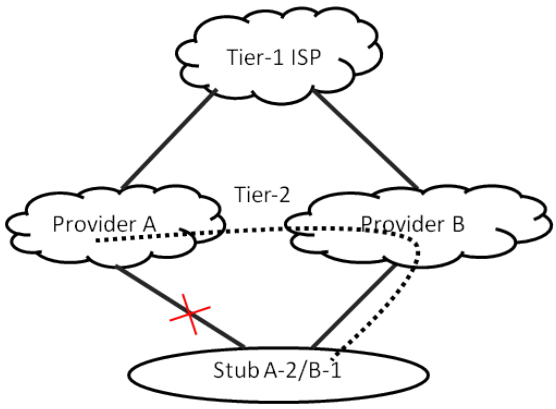

Figure 5.1: LIMA multihoming

Router-related We propose a LIMA router controller that runs (i) an L-DHCPv6 client, (ii) an L-DNS client, and (iii) a programmatic interface to the router. Techniques developed for automatic router configuration, such as Netconf [97], will be adopted for the dismembered addressing style of LIMA. Tunnel configuration applications should be designed to use names and not IP addresses. Automated techniques for updating firewall filters are required.

5.4.2 Multihoming

Fig. 5.1 shows what would happen under the LIMA policy in which the stub will have received a provider-local stub AS number from each provider forming its {provider AS number, stub AS number} pairs, e.g., A-2 and B-1. If the stub's link to provider A fails, given LIMA's routing policy restriction, the Tier-1 ISP will not have an entry for A-2, and consequently will not route datagrams addressed with A-2 via provider B. Our proposed solution to this problem is to provision a tunnel from the stub border router to provider A's border router passing through provider B's network as shown with the dotted line in Fig. 5.1, and use this tunnel as a backup path for the direct access link from the stub to provider A. A similar tunnel should be provisioned a priori to protect the access link to B. The Multi-Exit Discriminator (MED) field in BGP can be used to set the direct link as the primary option, and the backup tunnel as the secondary option, leading to seamless packet forwarding in case of access link failures.

In addition to this hitless forwarding of datagrams, *three* actions are required. *First*, to prevent new connections from using the A-2 addresses, the stub's fault management

system, upon receiving SNMP traps from the router indicating link failures, informs the L-DHCPv6 server causing it to broadcast messages to alert all endpoints to stop using A-2 based global addresses. *Second*, the fault management system should also inform the L-DNS server (a LIMA version of DNS server) causing it to stop providing A-2 addresses in its replies to queries. *Third*, the L-DHCPv6 clients at endpoints should initiate SCTP or MPTCP dynamic address reconfiguration for any ongoing connections.

5.4.3 Mobility

While a significant fraction of the devices in a future Internet of Things (IoT) will likely be wireless, and of this fraction, a significant portion may be mobile, the fraction of this set that is roaming (i.e., outside the home location) is likely to be small. We thus conclude that a hierarchical structure such as LIMA can be used, with mobility handled in much the same way as mobile IP.

However, to reduce the *path stretch* problem, we propose augmenting a mobile IP type solution with a dynamic DNS solution. There have been a number of proposals to use the secure dynamic DNS update feature to handle mobile location management [98, 99]. LIMA incorporates these solutions. Involving the DNS server is useful in LIMA because this allows the stub DNS server to inform its roaming mobiles of a change in its home {provider AS number, stub AS number} when a provider is changed. A mobile IP like solution is still needed in LIMA to handle connections initiated with DNS cached addresses wherein the home stub border router supports home agent functionality for its own endpoints, and foreign agent functionality for visitors.

5.4.4 Traffic engineering

Provider traffic engineering The elimination of current-day prefixes in favor of ASNs and longest-prefix matching in LIMA, and the routing policy of disallowing stub reachability in global routing tables, can cause path stretches. For example consider two backbone providers, Internet2 and ESnet. These two networks interconnect at Los Angeles, Seattle, Chicago, New York, and Washington. Consider two stub customers of ESnet, one in California (CA)

and the other in New York (NY). If stub reachability propagation is allowed, ESnet can report longer-prefix reachability to Internet2 for these two stubs. Packets destined to the ESnet CA stub originating at a stub connected to say a Kansas City router in Internet2 would be forwarded westward within Internet2 towards its Seattle router, while packets destined to the NY stub would be forwarded in the opposite direction toward Chicago if different MED values were configured by ESnet in the BGP updates sent by different routers for each stub. However, in LIMA, if ESnet is allowed to advertise only one provider AS number, such efficient routing cannot be achieved.

Our *proposed solution* is to allow the assignment of multiple AS numbers to providers allowing them to address different parts of their networks with different provider AS numbers. Analysis will be undertaken in future work to achieve a good trade-off between keeping the number of assigned AS numbers small so as to not increase the global routing table size significantly while achieving low path stretch values.

Stub Traffic Engineering To load balance incoming traffic across its providers in today's Internet, a multihomed stub can selectively send longer prefixes into the global routing table through each of its providers. But with the LIMA policy, this is not possible. We propose a *DNS and DHCP based solution* to this stub traffic engineering problem. The authoritative DNS server for the stub could order the multiple addresses returned in DNS responses if applications are programmed to choose addresses accordingly. For outgoing traffic engineering, the L-DHCPv6 client could use different orders when communicating the assigned {provider AS number, stub AS number} pairs to its endpoints, and have the NBS layer choose addresses accordingly.

5.5 LIMA components

Fig. 5.2 shows the internal architecture of a stub network with a border IPv6 based LIMA router. Applications will need to be modified to use NBS sockets rather than TCP or UDP sockets. NBS defines a new family type AF_NAME for the socket call. The `listen` and `accept` calls for the receiving side, and the `open` call for the sending side use domain names instead of IP addresses. `Read` and `write` system calls interface with the NBS socket

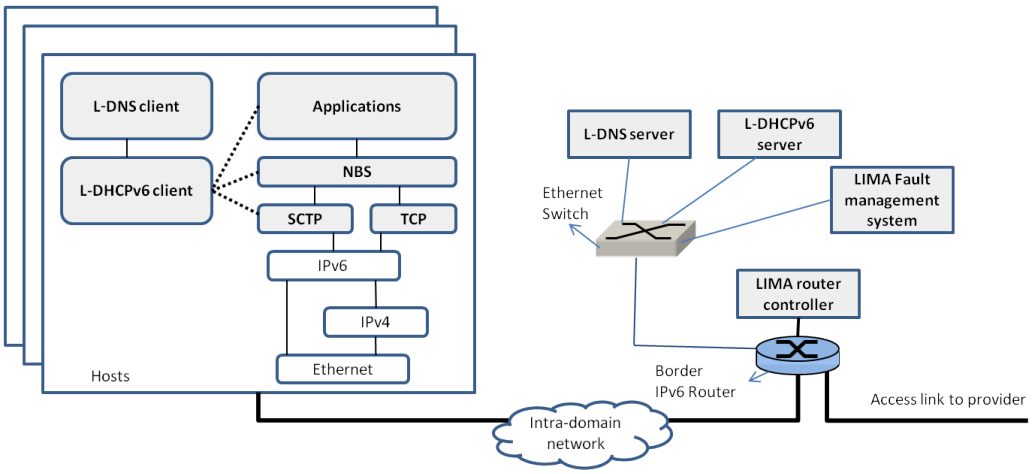

Figure 5.2: LIMA stub architecture

descriptor. The source name is carried in an IPv6 extension header to the destination in the first packet for the receive side application to use cached information about sources but with names rather than IP addresses (e.g., licensing servers).

Besides modifying DHCPv6 to support the *dismembered structure* of LIMA, a *broadcast push* operation is required for address renumbering or when an access link fails. The DHCPv6 `Reconfigure` message cannot be used as this message was designed for reconfiguring a single client, while we need a message to add or delete just the {provider AS number, stub AS number} pairs for all globally addressable endpoints within the stub. It is also sufficient for the L-DHCPv6 server to send just the IDAs of the stub border router interfaces (equivalent to Gateway address in today's Internet) and the IDA of the DNS server (today the whole IP address of the DNS server is sent to DHCP clients). As LIMA will rely more on domain names than the current Internet, more frequent secure dynamic DNS updates are anticipated from DNS clients running on endpoints as shown in Fig. 5.2. Initial registration of names to IDA mappings will be executed through the L-DHCPv6 server given that new hosts will not have the required certificates for authenticated DNS updates. This requires adding names to DHCPv6 messages.

Similarly modifications are needed to DNS to support the dismembered structure. For example, the resource record (RR) database structure will be modified as described in Section 5.4.1. An Autonomous System (AS) resource record, and support for mobility will also be added.

Table 5.2: Across all stubs (approx. 33K)

Month	#Provider additions	#Provider deletions
05/2011	1396	1049
06/2011	1408	1024
07/2011	1454	1112
08/2011	1435	1102
09/2011	1317	943
10/2011	1359	1092

The *LIMA fault management system* shown in Fig. 5.2 is required to interface with the L-DHCPv6 and L-DNS servers via a new protocol for access link failure handling. The *LIMA router controller* will support address renumbering of the router interfaces during initialization and provider changes.

Modifications to BGP are anticipated to support LIMA's dismembered addressing. For example, as the stub AS numbers are provider-local, BGP updates regarding stub AS numbers are required between stub border routers and their provider border routers, while provider AS number reachability will be propagated between providers.

5.6 Analysis

While a number of different analysis and prototyping efforts are required to fully evaluate LIMA, in this preliminary study, we report two sets of analyses. Section 5.6.1 examines the benefit of LIMA in global routing table size reduction, while Section 5.6.2 characterizes a measure of address renumbering.

5.6.1 Routing data analysis

By analyzing the Routeviews RIB data [100] for the last ten years, we characterize the growth in the total number of AS numbers, stub AS numbers, and provider AS numbers, as shown in Fig. 5.3. In LIMA, global routing tables will follow the low-rate growth pattern seen in the provider ASs plot, with a current-day number of 6185 providers. Contrast this with 335K prefixes today, and the exponential yearly growth rate of 17% [19].

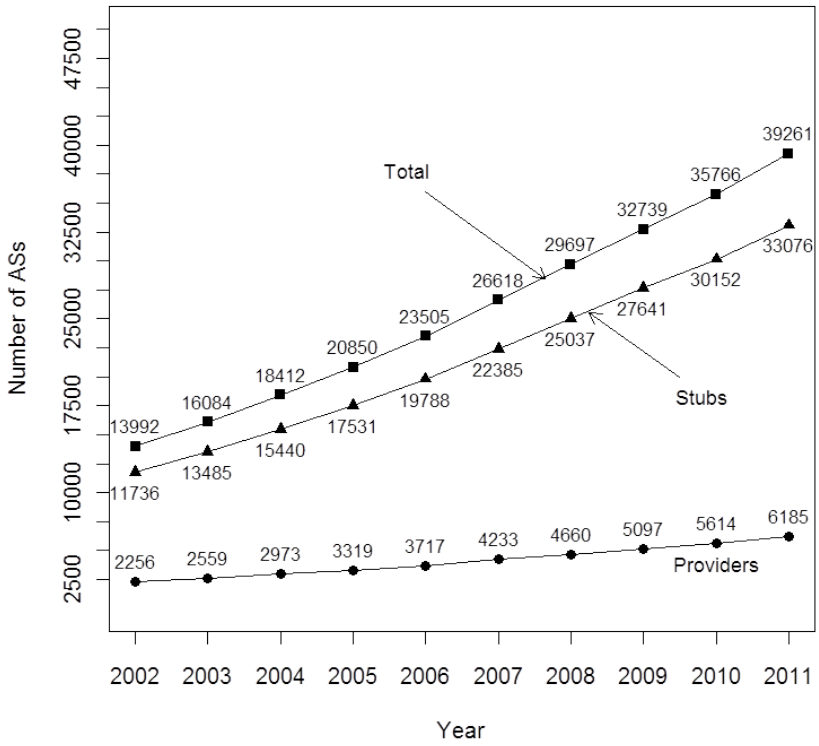

Figure 5.3: Provider, stub, and total number of ASs

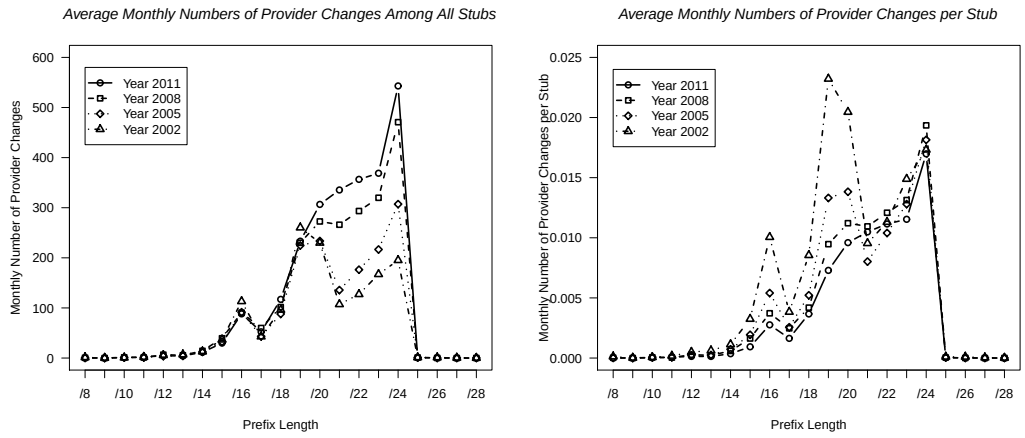

Figure 5.4: Average per-month numbers

5.6.2 A measure of the address renumbering overhead

To characterize the costs of address renumbering, we analyzed Routeviews RIB data to determine the frequency with which stubs add and/or delete providers as shown in the

Table 5.2. Some stubs just added or deleted a provider. But as both provider additions and provider deletions incur a renumbering operation, both are listed in the table.

Fig. 5.4 plots the mean number of provider changes per month made by stubs as a function of their prefix block sizes. The larger the prefix block size, the more potential for trouble tickets even if the address renumbering procedure is fully automated. While the absolute numbers have grown with time, the ratio has remained almost constant. For example, in years 2011 and 2002, the average monthly number of provider changes made by /24 stubs are 543 and 195, and the total number of stubs in 2011 and 2002 are 29466 and 11250, respectively. Therefore, the average monthly number of provider changes per stub is 0.018 in 2011 and 0.017 in 2002.

5.7 Conclusions

A policy combination that eliminates Provider Independent (PI) addressing for stubs, and disallows the propagation of stub-level reachability into global routing tables was proposed. While address renumbering of a large stub is challenging today, with some changes, such as disallowing the use of IP addresses in applications in favor of names, name based sockets, and dismembered addresses allowing for broadcast pushes of new provider numbers, this solution seems promising. LIMA scales back the per-packet processing actions relative to today's IP (by eliminating longest-prefix matches) as well as the newer locator-identifier split solutions. Instead it adds control- and management plane actions, which are however required only for handling relatively rare events such as provider changes and access link failures. This less-is-more approach will decrease capital expenditures of routers by lowering memory and processing costs, as well as operational (administrative and power consumption) costs. While LIMA is a clean-slate solution, some of its proposed solutions can be tested and applied in today's Internet. In the next chapter, a detailed solution for the multihoming problem is presented in the context of an IPv6 based Internet.

Chapter 6

MAST: A Stub Multi-homing Solution for IPv6 Networks

6.1 Introduction

In Chapter 5, we identified multihoming as one of the challenges that need to be addressed to implement the policy combination proposed for LIMA. A solution for the multihoming challenge was outlined in Chapter 5 for LIMA.

In the work presented in this chapter, the multihoming solution is developed further. But since the LIMA solution can be realized in an IPv6 Internet, we changed the context from the clean-slate LIMA design to IPv6. Therefore, in this chapter, the underlying Internet is assumed to be based on IPv6. We call this solution Multi-Addressing with Stub Tunnels (MAST).

IPv6 supports multiaddressing, i.e., the assignment of multiple addresses to an interface. Leveraging this feature, a stub can acquire multiple address sub-blocks from the PA address blocks of each of its multihomed providers, and configure most, if not all, its host and router interfaces with addresses derived from the multiple PA-address blocks. Stub PA address sub-blocks are not propagated into the DFZ. Instead tunnels are used between a MAST stub and each of its providers to serve as backup paths in case of access link failures. The MAST solution requires a new Inter-domain Tunnel Management Protocol (ITMP), a new

Stub Reachability Management Protocol (SRMP), and enhancements to DNS and DHCPv6.

The novelty of the MAST solution lies in its achieving the goal of global routing table growth rate reduction without requiring modifications to routers as is needed in identifier-locator split solutions [78, 82, 85, 101–106]. Furthermore, MAST requires no changes in end hosts unlike in other solutions [61, 96, 107]. Thus no changes are required in data-plane entities (hosts and routers). However changes are proposed to the stub DNS server (not to top-level domain and root DNS servers) and DHCPv6 server for stubs that adopt the MAST solution, and new stub network management systems are required to reduce administrative overhead. Further, changes are limited to only those stubs that adopt MAST. Hosts in a MAST stub can communicate through standard TCP/IPv6 protocols with hosts in a non-MAST stub. Thus, MAST can be incrementally deployed in the IPv6 Internet unlike clean-slate designs. The MAST solution should be coupled with automated address renumbering mechanisms [93].

Section 6.2 extends the BGP data analysis presented in Section 5.6. Section 6.3 describes the MAST solution, which is evaluated in Section 6.4. After a brief review of related work in Section 6.5, the paper is concluded in Section 6.6.

6.2 An extended BGP RIB nalysis

Since the MAST solution enables both single-homed and multi-homed stubs to obtain PA addresses, we extended the analysis presented in Section 5.6 to determine the numbers of different types of stubs in today's Internet.

6.2.1 Analysis Method

Public data archives containing BGP Routing Information Base (RIB) data are available on the Route Views project Web site [100], which is hosted by the University of Oregon. The BGP RIB data are received by multiple BGP "beacons," and published every two hours. For our analysis, we used the archived data for the period Jan. 2002 to Jan. 2013, collected by the beacon at route-views2.oregon-ix.net. Each RIB entry shows reachability information for a prefix, specifically the next-hop router (`NEXT_HOP`) and the Autonomous System (AS)

path `AS_PATH`, which lists the AS Numbers (ASNs) of all the ASes on the path to reach the destination, not including the AS number of the BGP beacon itself.

Our analysis software processed the RIB data to determine the following information about each AS: (i) whether the AS is a stub or provider, (ii) the prefixes advertised by the AS, (iii) the AS Numbers (ASNs) of upstream and downstream ASNs to which the AS is connected, and for each stub AS, whether it has PI or PA addressing.

Stub or provider Our analysis program used the `AS_PATH` information to categorize an AS as a stub if its ASN had not appeared in the middle of any `AS_PATH`, and a provider otherwise. This step was executed after deleting repeated ASNs, which occur often in `AS_PATHs` because of a common practice called *AS Prepending*. An AS typically prepends its ASN multiple time for a specific prefix to artificially increase the length of the `AS_PATH` in order to make it less preferable in the BGP decision process.

Prefixes corresponding to an AS The originating AS for a prefix appears at the end of the `AS_PATH`. Thus, for each AS, the corresponding prefixes can be determined. Not all prefixes originated by an AS are necessarily owned by that AS because of the use of private ASNs [108]. For example, AT&T's ASN was listed as the originating AS for 1505 prefixes in October 2010, but of these AT&T owned only 109 prefixes [109]. Private ASNs are not included in `AS_PATHs`. The use of private ASNs implies that there are more stubs than are visible from the BGP data.

Upstream and downstream ASes For each AS, the number and ASNs of its upstream and downstream ASes can be readily obtained from the `AS_PATHs`.

PI or PA For each prefix of each stub ASN, our analysis software compares the stub's prefixes with the prefixes of the stub's upstream ASes to determine whether the stub has PI or PA addressing.

The output of the analysis software was used to populate a MySQL database for easy querying. We executed our analysis program on the first available RIB file on the 5th day of each month to create a per-month Autonomous Systems (AS) Table for the 2002-2013 period.

Table 6.1: Numbers of different types of stubs (RIB Data source: 12AM December 1, 2012)

	PI	PA	Both	Total
Single-homed	13613	1938	881	16432
Multi-homed	14695	2853	2397	19945
Total	28308	4791	3278	36377

6.2.2 Analysis Results

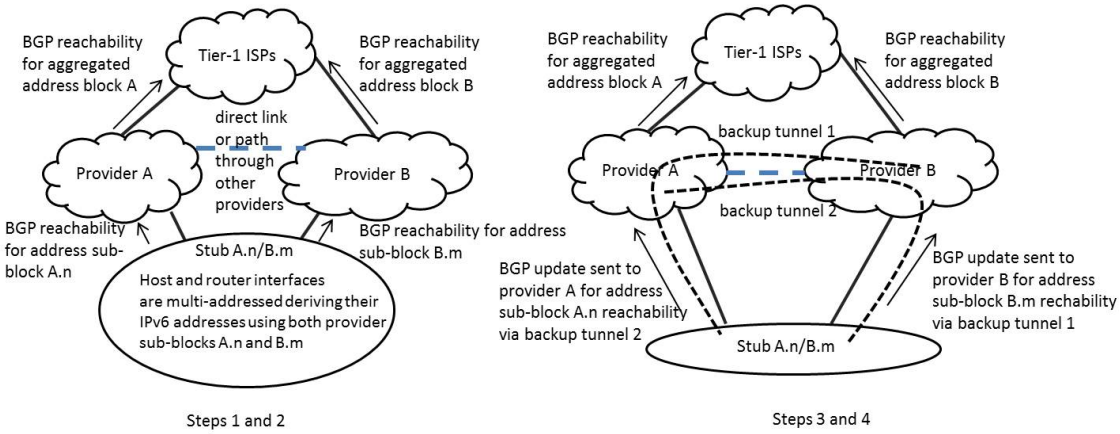

Figure 6.1: Stub multihoming in MAST: Basic concept

First, of the 43100 unique ASNs found in the RIB data collected on Dec. 1st, 2012 at 12AM, we classified 36377 as stubs and 6723 as providers using the approach described in the previous section. If all stubs adopted the MAST solution, the global routing table would have only addresses for the 6723 providers.

Next, we determined the numbers of stubs of different types as shown in Table 6.1. It is of interest that some single-homed stubs have both PI address blocks as well as PA address sub-blocks derived from their single access providers. Such possibilities could occur when companies merge.

Overall, Table 6.1 shows more multihomed stubs than single-homed and more PI stubs than PA stubs. But this is misleading for *two reasons*. *First* stubs use private ASNs as mentioned in the previous section. These are stubs that run BGP routers, but have not procured a public ASN. ARIN's AS policy manual [110] notes that it "in order to be assigned an AS Number, each requesting organization must provide ARIN with verification that it has one of the following: (i) a unique routing policy (its policy differs from its border gateway peers), or (ii) a multihomed site." Stubs that do not meet these requirements use

private ASNs from the range 64512 through 65535. Therefore the number of single-homed stubs with PI or PA addressing with routers that run BGP could be significantly greater than the numbers indicated for these categories in Table 6.1.

The *second reason* is that many stubs do not run BGP routers at all. These include home and small business networks. As of June 2013, 72.4% of American households (which amounts to 88 million households) have high-speed Internet access [111]. Most of these home networks are single-homed and have PA addressing. Therefore, the number of single-homed PA stubs listed in Table 6.1 is vastly understated. It includes only those stubs that run BGP routers and have public ASNs.

If we include stubs with private ASNs and the home/small business network stubs, clearly the percentage of multihomed stubs and the percentage of PI stubs is currently not that large. However, as home users come to rely increasingly on the Internet, especially as the Internet-of-Things (IoT) and its corresponding services get deployed, then more home networks are likely to become multi-homed. The impact of propagating longer prefixes into the DFZ will then result in a more explosive growth of the global routing size.

6.3 Proposed Solution: MAST

As per the name "Multi-Addressing with Stub Tunnels (MAST)," there are two key components to regaining the scalability advantage of hierarchical addressing: multi-addressing, and stub-based tunnels. After explaining these basic aspects of the MAST design in Section 6.3.1, Section 6.3.2 proposes two new stub-based network management systems, and enhancements to DHCPv6 and DNS. Section 6.3.3 briefly describes the need for an automated inter-domain tunnel management system. Section 6.3.4 estimates the number of entries that would be needed in the global routing table, and in access provider's routers under the MAST solution.

6.3.1 Basic Concept

The basic concept has *four* steps. *First*, a stub obtains one or more address sub-blocks from each of its providers, and configures its host and router interfaces with these multiple

address sub-blocks. This concept is illustrated in left-hand side diagram of Fig. 6.1. The example stub shown in the figure obtains PA address sub-block allocations from each of its providers, e.g., A.n from the provider whose address block is A, and B.m from the provider whose address block is B. The prefixes A.n and B.m are configured as the subnet addresses for, preferably all, the host and router interfaces within the stub.

In the *second* step, stubs enable BGP on their direct interfaces to each provider, and propagate, to each provider, only the particular address sub-block derived from that provider's address block. The providers only propagate reachability for their aggregated address blocks, not their stubs' longer prefixes, to the Tier-1 ISPs. The left-hand side diagram of Fig. 6.1 illustrates this step 2. The stub propagates A.n to provider A and B.m to provider B, provider A propagates only its aggregated address block A to the Tier-1 ISPs, and similarly provider B propagates only its aggregated address block B to the Tier-1 ISPs.

A problem arises in case of access-link failures. For example, if the stub's access link to provider A fails, since the stub's longer-prefix address sub-block A.n was not propagated to the DFZ routers in the Tier-1 ISPs via its other provider B, packets with destination addresses from the A.n sub-block cannot be routed to the stub, which undermines the very purpose of multihoming.

To address this reachability-during-failures problem, a *third* step is included in MAST. Tunnels are setup between the stub and each of its providers by leveraging one of the stub's other provider's access links. There are various methods to implement tunnels, such as GRE [112], IP-in-IP [113], and L2TPv3 [114]. For example, the right-hand side diagram of Fig. 6.1 shows two tunnels: tunnel 1 from the stub to provider B via the stub's access link to provider A, and tunnel 2 from the stub to provider A via the stub's access link to provider B. These tunnels are referred to as "backup tunnels" (see Fig. 6.1) because they offer secondary paths, and are not used unless failures occur. This ensures that traffic destined to or sourced from addresses derived from a particular provider's address block are primarily routed through that provider, and routed through the stub's other providers (and possibly other intermediate providers) only when failures occur. The left-hand side diagram of Fig. 6.1 shows that the two providers of a stub may not be directly connected to each other, and therefore the backup tunnels could be passing through other providers. Tunnel

configuration requires actions only at the source and destination routers of the tunnel, and therefore the intermediate providers, including the stub's other provider through which the tunnel passes, are not involved in tunnel provisioning.

The *fourth* step enables reachability propagation on these backup tunnels. BGP is enabled on these tunnels, which are viewed as logical interfaces on each router, but the local-preference metric is set to a lower value than the route via the direct access links in order to favor the latter. As illustrated in the the right-hand side diagram of Fig. 6.1, BGP updates are sent from the stub to provider B for reachability of its address sub-block B.m through the backup tunnel 1 setup via provider A, and conversely to provider A for reachability of its address sub-block A.n through the backup tunnel 2 setup via provider B. The RIBs will store reachability via these backup tunnels as alternate routes. In case of an access-link failure, the router's BGP software automatically sends messages for unreachable routes, and the Forwarding Information Base (FIBs) are updated with the alternate routes from the RIBs. As this solution uses mechanisms and software that are already part of deployed routers, it needs no new protocols (and hence standardization), nor does it require router modifications. In summary, this fourth step ensures stub reachability on its address sub-block derived from a provider's address allocation even if the direct access link from the stub to that provider fails, which is important for the preservation of active TCP connections.

6.3.2 Intra-Stub Design

In the previous section, we described the MAST solution for preventing loss of active TCP connections in the presence of access-link failures. In this section, we consider the question of how future connections can be prevented from using addresses derived from the address block of the provider to whose network the stub's access link has failed. More broadly, this section describes the MAST intra-stub architecture.

The data-plane path of the stub network requires no modifications allowing for the use of standard applications, transport protocols, IPv6 with multi-addressing, and multiple access links from possibly multiple gateway IPv6 routers. On the other hand, for the MAST solution, we propose two new network management systems and modifications to DHCPv6 and DNS.

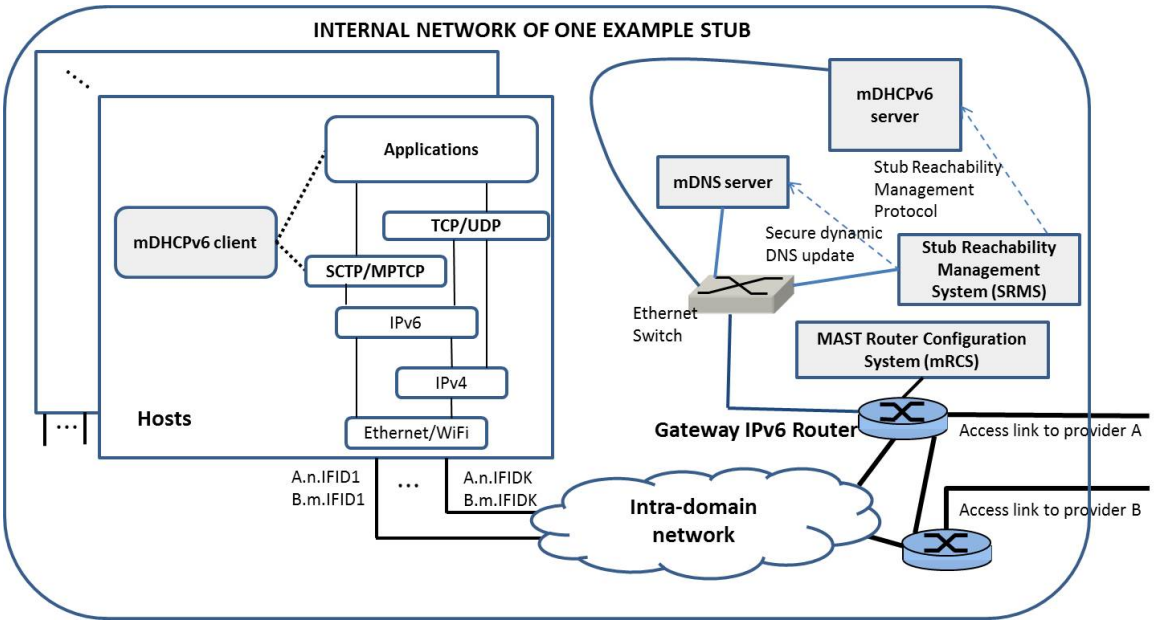

Figure 6.2: An example MAST stub architecture

Data plane Fig. 6.2 shows an example architecture of a stub in the MAST solution. The stub is multihomed in that it has access links to two providers A and B. The access links are shown to terminate on different routers for increased reliability. Hosts are shown to have multiple interfaces, each of which is configured with multiple addresses. For example, as shown in Fig. 6.2, if a host has K interfaces, each interface k, $1 \leq k \leq K$ is assigned two IPv6 addresses, A.n.IFIDk and B.m.IFIDk. The subnet identifiers (prefixes) A.n and B.m are the address sub-blocks assigned to the stub by its providers A and B, respectively, as described in the previous section. IFID stands for Interface Identifier, which is part of the IPv6 address [115]. IPv6 supports multi-addressing, and hence this feature is already available.

Stub Reachability Management System (SRMS) The role of the SRMS is to detect access link failures and initiate recovery actions inside the stub, and to manage the backup tunnels described in the previous section. When an access link fails, the first step is to detect the failure. The gateway IPv6 router to which the failed access link is connected will detect the failure and notify, using SNMP traps, the Stub Reachability Management System (SRMS) (see Fig. 6.2). We refer to the addresses derived from the corresponding provider's address block as *Discouraged Addresses* rather than "unreachable" because these addresses are

still reachable via the backup tunnels, but are less preferred as they use more network resources. All hosts and routers should be notified to stop using these discouraged addresses as their source IPv6 addresses in future outgoing connections, until recovery. After recovery, these addresses should be re-enabled. For these notifications, a new protocol called *Stub reachability management protocol* is proposed for communication between the SRMS and the DHCPv6 [116] server. Also, these addresses should be discouraged in future incoming connections and re-enabled after recovery by notifying the DNS server so that the latter can stop issuance of these addresses in response to queries. For these notifications, the SRMS can use secure dynamic DNS updates [117]. In Fig. 6.2, we refer to the DNS and DHCPv6 servers servers as `mDNS` and `mDHCPv6` servers, respectively, because modifications are required to support the MAST solution as described next.

mDHCPv6 When the mDHCPv6 server receives a message from the SRMS to discourage or re-enable an address prefix, the mDHCPv6 server notifies all its clients of this change. We propose to add a new DHCPv6 message called `Broadcast Prefix Push`, in which just the subnet address (prefix) is pushed to all clients. Client and server code need to be modified to handle subnet addresses (prefixes), e.g., A.n or B.m, separately from interface IDs. mDHCPv6 clients will need the ability to aggregate the subnet addresses received in `Broadcast Prefix Push` messages with the interfaces' IFIDs to create complete IPv6 addresses before issuing `ifconfig` or equivalent commands to configure the interfaces. Currently, there is a `Reconfigure` message in DHCPv6, but this message was designed for reconfiguring a single client, not multiple clients. Hence, we propose the new `Broadcast Prefix Push` message.

mDNS Along the same lines as for mDHCPv6, we propose to splinter IPv6 addresses and store address prefixes separately from Interface IDs (IFIDs). A DNS resource record (RR) type already exists for address prefixes [118], which can be used to support MAST. However, a new RR is required for IFID. Furthermore, the DNS server code should be modified to assemble IPv6 addresses from the values stored for the separated address prefix and IFID RRs, before responding to queries. When the mDNS server is notified by the SRMS about a discouraged address prefix through a secure dynamic DNS update, it changes the status of the

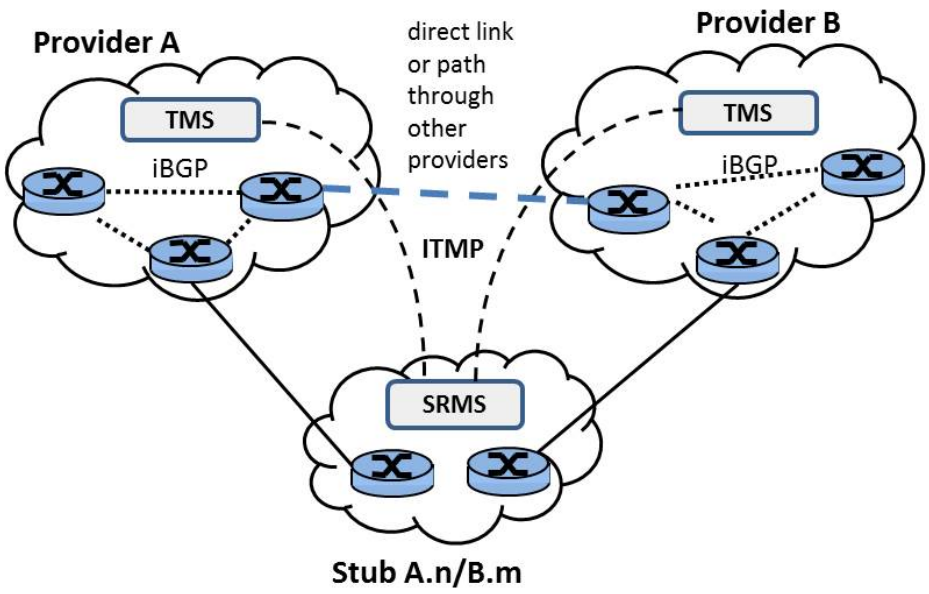

Figure 6.3: Inter-domain tunnel management

corresponding prefix RR, and avoids using the prefix in responses to queries. This action will prevent future incoming connections to the stub (those that are preceded by an authoritative DNS server lookup) from using the discouraged addresses. However, since DNS servers cache addresses, until the time-to-live expires, future incoming connections/datagrams may still appear destined to these discouraged addresses, but since reachability will be available through the backup tunnels, continued communication is possible.

MAST Router Configuration System (mRCS) Fig. 6.2 shows a network management called mRCS, which implements a mDHCPv6 client to receive the `broadcast prefix push` messages with the status of address prefixes, and a command-line client to configure the IP addresses of the router interfaces. Since routers run software with applications such as HTTP servers and SSH servers for administrative use, the routers need to be informed when an access link fails so that future connections avoid the discouraged addresses.

6.3.3 Inter-domain Tunnel Management Protocol (ITMP)

As tunnel setup typically requires administrator actions, for providers with many stubs (e.g., the largest number of per-provider stubs observed in the current RIBs was 2589), the

administrative overhead could be significant. Therefore, we propose an automated approach for inter-domain tunnel management.

As shown in Fig. 6.3, we propose an Inter-domain Tunnel Management Protocol (ITMP), which is executed between the SRMS in each multihomed stub and a Tunnel Management System (TMS) in each provider. Tunnel setup action will be initiated at the multihomed stub by an administrator using the SRMS. After authentication, the TMS at each provider can configure the corresponding gateway routers to terminate the tunnel, provide it an IP address and subnet mask, and enable BGP on it with a lower local preference relative to the direct access link. Details of the ITMP are not provided here due to space limitations, but the goal is to require no administrative support in the provider networks for tunnel configuration operations. Since the tunnels are offering stubs the added value of reachability in the presence of failures, stub administrator involvement is required.

6.3.4 Routing Tables

The DFZ routers in the Tier-1 ISPs, as shown in Fig. 6.1, will have the global routing table. Since in the MAST solution stubs use PA addressing and access providers do not propagate the stubs' longer prefixes to the Tier-1 ISPs, the global routing table will be smaller than in the current solution where PI addressing and stub prefix address propagation are widely used. In a best-case scenario, if all stubs adopt the MAST solution, the global routing table will only have provider address blocks.

The routing tables in a provider network that offers direct access service to stubs will have three sets of entries: (i) Either the global routing table (GRT) or specific entries advertised by their transit providers with a default entry; (ii) entries for each of its stubs; and (iii) entries for reaching each of its stubs via backup tunnels. Thus for a provider i with m_i multihomed stubs and s_i single-homed stubs, the number of routing table entries R_i is given by

$$|GRT| + s_i + 2 \times m_i \leq R_i \leq |GRT| + s_i + \sum_{j=1}^{m_i} n_{ij} \tag{6.1}$$

where n_{ij} is the degree of multihoming of the j^{th} multihomed stub of provider i. The first term in both bounds assumes that provider i stores the entire GRT, which is true for Tier-1

ISPs that also have stubs. This assumption is made to find the worst-case number of routing table entries. The second term in both bounds shows that a provider's routers must have single routing table entry for each for its single-homed stubs. The last term of the lower bound of (6.1) assumes that each multihomed stub has only one backup tunnel for each of its access links regardless of its degree of multihoming. This is sufficient for protection against single-link failures. A provider i requires a primary entry for each of its m_i multihomed stubs via the stub's direct access link, and a secondary entry to reach the stub using the address sub-block derived from the provider's address block via a single backup tunnel from the provider to the stub. This explains the factor of two for the last term in the lower bound. The upper bound assumes that a multihomed stub will create as many backup tunnels to each of its providers as its degree of multihoming minus one $(n_{ij} - 1)$ for protection against multiple simultaneous access link failures. Adding an entry for the path through the direct access link accounts for the last term in the upper bound.

In the next section, our analysis will quantify the numbers for these three sets of entries.

6.4 Analysis

Fig. 6.4 shows four plots projecting the sizes of routing tables to illustrate the potential of the MAST solution under a best-case assumption that all stubs use the MAST solution. Plot 1, a flat line at 24436 entries, corresponds to the projected size of the GRT (this is a 94.6% reduction from the 450K number in today's GRT). In Oct. 2010, there were only 5623 providers. However, a provider could advertise multiple prefixes as illustrated by the AT&T example in Section 6.2.1. For each prefix advertised by a provider, our software determined whether the prefix itself or its superset was owned by the corresponding ASN by consulting the Prefix-to-AS mapping dataset downloaded from ARIN [109]. Since providers advertise prefixes owned by stubs with private ASNs, this check was required. The Oct. 2010 dataset was the one available from ARIN when we executed this analysis, and therefore the corresponding RIB data was used.

Plot 2 shows the number of entries required in each provider for its own single-homed (SH) stubs. The providers were sorted by the upper-bound numbers of entries. The provider

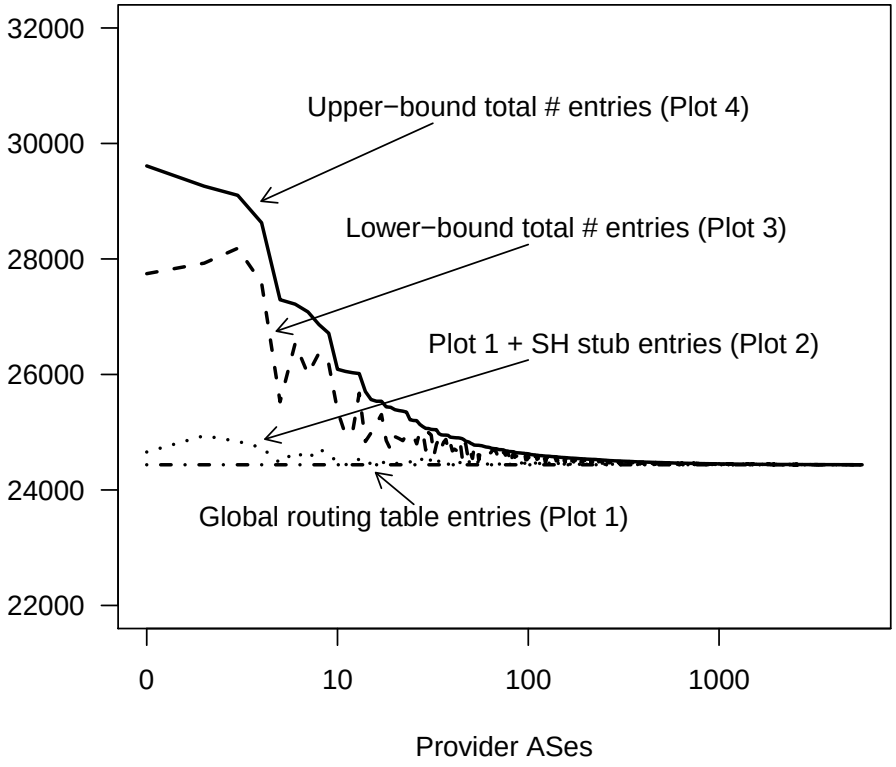

Figure 6.4: Routing table size (in prefixes) in MAST (Input Data: AS table for 6AM October 5, 2010)

that has the most number of stubs is 2077 of which 405 were single-homed and the remaining multi-homed.

Plots 3 and 4 correspond to the lower and upper bounds of (6.1), respectively. In today's Internet, more than 76% of multi-homed stubs have just two providers. For stubs with more than two providers, multiple backup tunnels can be created as explained in Section 6.3.4.

Finally, we considered the question of how the maximum number of per-provider stubs changed over time. In 2003, this number was around 2100. It was more-or-less constant until 2009 when the number went down to about 1900, but in the last three years this number has increased to over 2500 as of Dec. 2012.

6.5 Related work

The multiaddressing solution is proposed most notably in the Shim6 architecture [107]. Host interfaces are assigned locators that are derived from PA address sub-blocks. A TCP connection is established using one of the locators as the IP address. If a failure occurs, which is detected by an end-to-end reachability protocol, a new locator is used, but the TCP connection is held open as it uses the old locator as the upper layer identifier (ULID) and a shim6 extension header is inserted (a form of tunneling). This solution requires end host protocol stacks to be modified, a requirement not incurred in MAST.

Another proposal that is similar to MAST in its use of multiaddressing is HANA [119,120]. The stub network deployment of this work is similar to the MAST structure of the providers' addressing being flat with a hierarchical approach for stub addressing. However, their solution for dealing with access-link failures is different from ours. Their solution uses backup paths between providers, a Routing-with-Detour concept that requires new information in packet headers, and a light-weight routing process that runs concurrently with BGP. This implies that routers need to modified, unlike in our scheme.

A review of IPv6 multihoming solutions [121] surveyed several approaches being standardized in the IETF, such as Locator-Identifier Split Protocol (LISP) [20], Identifier-Locator Network Protocol (ILNP) [79], Host Identity Protocol (HIP) [80], Network Prefix Translation (NPT) [83] and others. There are many research proposals that are based on identifier-locator split, such as eFIT [78], HRA [103], TurfNet [85], HAIR [104], and MILSA [105]. Our MAST solution does not require an identifier and uses only names and addresses as in the current Internet. By avoiding identifiers, the additional map-n-encap [121] overhead of mapping and encapsulation are avoided.

6.6 Conclusions

Stubs purchase Internet access links to multiple providers for reliability reasons. Such multi-homing has been identified as one of the primary causes for the exponential growth rate of the global routing table (GRT). This work proposed a stub multihoming solution called Multi-Addressing with Stub Tunnels (MAST). By combining IPv6 multi-addressing

with backup tunnels between a stub and each of its providers, stubs can enjoy the reliability advantages of multihoming without adding to the size of the GRT. MAST requires no changes to end hosts or routers, and limits changes to the stub's DNS and DHCPv6 servers. Stubs that do not adopt the MAST solution can still communicate with MAST stub hosts. Thus, MAST can be incrementally deployed. An analysis of the current Internet BGP RIBs was conducted. It showed that the current global routing table size of 450K prefix entries would be reduced to about 24K entries if the MAST solution was adopted by all stubs.

Chapter 7

Conclusions and Future Work

We first summarize the work presented in this dissertation and draw four key conclusions, and then discuss potential future work to advance our current research.

7.1 Summary and conclusions

In this dissertation, we presented our work towards advancing inter-domain networking. Our main contributions are as follows: (i) We characterized the traffic patterns of two meteorology data feedtypes distributed by IDD; (ii) we developed, implemented, and evaluated the Virtual Circuit Multicast Transport Protocol (VCMTP), which is the first reliable multicast transport protocol specifically designed for virtual circuits; (iii) we designed and evaluated the Less-Is-More Architecture (LIMA), which is a clean-slate inter-domain routing and addressing architecture for a future Internet; and (iv) we proposed a detailed solution for the stub multihoming problem in the context of an IPv6 based Internet.

Chapter 2 presented an analysis of two representative feed types distributed by IDD, i.e., CONDUIT and NEXRAD2. Our findings were that both feed types are received by about 150 receivers, and data products are sent almost continuously, especially in the case of NEXRAD2. Changes will be required to the VC structures as receivers change their subscriptions to feed types (necessitating dynamic control of the multicast VCs). Therefore, we *concluded* that a reliable multicast transport service over Scheduled Dynamic Circuit Service (SDCS) is suitable for the IDD application.

Chapter 3 described the design, implementation, and evaluation of Virtual Circuit Multicast Transport Protocol version 1 (VCMTPv1). The primary goal for VCMTPv1 was to minimize the negative performance impact of slow receivers on the throughput experienced by fast receivers. Consequently, a key design assumption in VCMTPv1 was to handle all data retransmissions after the file multicast so that a sender can utilize the complete network and CPU resources for high-speed multicasting. However, due to the separation of the multicast phase from the retransmission phase, VCMTPv1 has the limitation that it is only sustainable when file inter-arrival times are significantly longer than the service times required to transfer files.

Chapter 4 presented our design, implementation, and evaluation of VCMTP version 2 (VCMTPv2). Motivated by the need for serving continuous file transfers as required in IDD, we changed some of the key design assumptions made in VCMTPv1. In the VCMTPv2 design, retransmissions are executed in parallel with multicasts. Furthermore, multiple files may be served concurrently. For continuously generated files, a key design aspect of VCMTPv2 is the tradeoff between file-delivery throughput for fast receivers and robustness for slow receivers. A VCMTPv2 configurable parameter called retransmission timeout factor can be adjusted to tradeoff these two metrics. Our *conclusion* from the VCMTPv2 evaluation is that, for the same performance metric (e.g., latency), using unicast TCP connections will require more servers at the sender and a higher access-link bandwidth than using reliable multicast service. Alternately, for the same number of servers and access-link bandwidth, the latency of delivering data products could be smaller.

Chapter 5 presented our Less-Is-More Architecture (LIMA), a new clean-slate inter-domain routing and addressing architecture for a future Internet. A policy combination that eliminates Provider Independent (PI) addressing for stubs, and disallows the propagation of stub-level reachability into global routing tables was proposed. This policy combination creates an address renumbering problem, which we addressed by leveraging multi-addressing and a novel concept called "dismembered addressing" proposed for LIMA. Other aspects impacted by the policy combination, such as multihoming and traffic engineering, were also discussed with proposed solutions. Our *conclusion* is that it is feasible to adopt the above policy combination in conjunction with control-plane solutions for the address renumbering

and multihoming problems.

Chapter 6 proposed a detailed solution for the stub multihoming problem. Instead of presenting this solution in the context of the clean-slate LIMA solution, we applied the policy combination proposed in LIMA to an IPv6 based Internet. By combining IPv6 multi-addressing with backup tunnels between a stub and each of its providers, stubs can enjoy the reliability advantages of multihoming without adding to the size of the global routing tables. An analysis of the current Internet Border Gateway Protocol (BGP) Routing Information Bases (RIBs) showed that the current global routing table size of 450K prefix entries would be reduced to about 24K entries if the MAST solution was adopted by all stubs. Our *conclusion* is that the MAST solution can be incrementally deployed in the IPv6 based Internet to increase global routing scalability.

7.2 Future Work

This work can be extended in the following directions:

1. Our current VCMTP implementation can be ported to Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCE), as illustrated in Section 4.5. The integrated LDM-VCMTP software can be tested over wide-area dynamic virtual circuits, e.g., the Internet2 DYNES deployment.

2. For LIMA, our proposed solutions for address renumbering and traffic engineering can be prototyped and evaluated on an IPv6 based Internet. Further data analysis of the RIB information (e.g., BGP updates) can be conducted to evaluate the benefits and costs of our proposed policy combination and solutions for the above challenges.

Bibliography

[1] Malathi Veeraraghavan, Mark Karol, and George Clapp. Optical dynamic circuit services. In *IEEE Communications Magazine*, volume 48, pages 109–117, November 2010.

[2] BGP routing table analysis reports. http://bgp.potaroo.net/.

[3] B. Metcalfe. Metcalfe's law: A network becomes more valuable as it reaches more users. *Infoworld*, 1995.

[4] Internet2. Dynamic network system.

[5] Oscars. http://www.es.net/OSCARS/docs/index.html.

[6] Interoperable OnDemand Network, Internet2. http://www.internet2.edu/ion/.

[7] InterDomain Controller Protocol. http://hpn.east.isi.edu/dice-idcp/.

[8] Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks. http://standards.ieee.org/getieee802/download/802.1Q-2011.pdf.

[9] Network Service Interface Working Group. http://www.ogf.org/gf/group_info/view.php?group=nsi-wg/.

[10] Internet Data Distribution. http://www.unidata.ucar.edu/software/idd/.

[11] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. Revisiting IP multicast. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 15–26, New York, NY, USA, 2006. ACM.

[12] B. Quinn and K. Almeroth. IP Multicast Applications: Challenges and Solutions. RFC 3170 (Informational), September 2001.

[13] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas. Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised). RFC 4601 (Proposed Standard), August 2006. Updated by RFCs 5059, 5796, 6226.

[14] B. Fenner and D. Meyer. Multicast Source Discovery Protocol (MSDP). RFC 3618 (Experimental), October 2003.

[15] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376 (Proposed Standard), October 2002. Updated by RFC 4604.

[16] R. Vida and L. Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810 (Proposed Standard), June 2004. Updated by RFC 4604.

[17] K. Kompella and Y. Rekhter. Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling. RFC 4761 (Proposed Standard), January 2007. Updated by RFC 5462.

[18] D. Meyer, L. Zhang, and K. Fall. Report from the IAB workshop on routing and addressing. RFC 4984, Internet Engineering Task Force, September 2007.

[19] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable internet protocol (AIP). In *Proc. of the ACM SIGCOMM*, pages 339–350, 2008.

[20] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID Separation Protocol (LISP). Technical report, IETF Draft, March 2009.

[21] T. Bu, L. Gao, and D. Towsley. On characterizing BGP routing table growth. *Computer Networks*, 45(1):45–54, 2004.

[22] Emulab. http://www.emulab.net/.

[23] John W. Byers, Michael Luby, and Michael Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communications*, 20:1528–1540, 2002.

[24] S. Acharya, M. Franklin, and S. Zdonik. Dissemination-based data delivery using broadcast disks. *Personal Communications, IEEE*, 2(6):50 –60, dec 1995.

[25] Jie Li, Malathi Veeraraghavan, Matthew Manley, and Steve Emmerson. Analysis and selection of a network service for a scientific data distribution project. In *International Conference on Communications, Mobility, and Computing (CMC)*, May 21-23 2012.

[26] Jie Li and Malathi Veeraraghavan. A reliable message multicast transport protocol for virtual circuits. In *International Conference on Communications, Mobility, and Computing (CMC)*, May 21-23 2012.

[27] Jie Li, Malathi Veeraraghavan, Steve Emmerson, and Robert. D. Russell. VCMTP: A Reliable Message Multicast Transport Protocol for Virtual Circuits. In *Submission to IEEE International Conference on Communications (ICC)*, 2013.

[28] Jie Li, Malathi Veeraraghavan, Martin Reisslein, Matthew Manley, Ronald D Williams, P Amer, and J Leighton. A Less-Is-More Architecture (LIMA) for a Future Internet. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2012*, pages 55–60, 2012.

[29] Jie Li and Malathi Veeraraghavan. A Stub Multi-homing Solution for IPv6 Networks. In *Submission to IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2013.

[30] Malathi Veeraraghavan, Jie Li, and Martin Reisslein. A strawman proposal for future diverse internets. In *IEEE Symposium on Computers and Communications (ISCC)*, pages 792–795. IEEE, 2011.

[31] Local Data Manager. http://www.unidata.ucar.edu/software/ldm/.

[32] IDD real-time statistics. http://www.unidata.ucar.edu/software/idd/rtstats/.

[33] Sally Floyd, Van Jacobson, Ching-Gung Liu, Steven McCanne, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *ACM SIGCOMM*, page 342, August 1995.

[34] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994.

[35] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '96, pages 117–130, New York, NY, USA, 1996. ACM.

[36] Xuan Zheng, Anant Padmanath Mudambi, and Malathi Veeraraghavan. FRTP: Fixed Rate Transport Protocol - A modified version of SABUL for end-to-end circuits. In *Pathnets Workshop, held in conjunction with Broadnets 2004*, October 2004.

[37] Sanjoy Paul, Krishan K. Sabnani, John C.-H. Lin, and Supratik Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15:407, April 1997.

[38] B. Whetten and G. Taskale. An overview of reliable multicast transport protocol II. *Network, IEEE*, 14(1):37 –47, jan/feb 2000.

[39] C. Bormann, J. Ott, H.-C. Gehrcke, T. Kerschat, and N. Seifert. MTP-2: Towards achieving the S.E.R.O. properties for multicast transport. In *Internet Conference on Computer Communications and Networks (ICCCN)*, September 1994.

[40] A. Koifman and S. Zabele. RAMP: A reliable adaptive multicast protocol. In *IEEE Infocom '96*, page 1142, March 1996.

[41] B. Adamson, C. Bormann, M. Handley, and J. Macker. NACK-Oriented Reliable Multicast (NORM) Transport Protocol. RFC 5740, Internet Engineering Task Force, 2009.

[42] E. He, J. Leigh, O. Yu, and T.A. Defanti. Reliable blast udp : predictable high performance bulk data transfer. In *Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on*, pages 317 – 324, 2002.

[43] Earth System Grid. http://www.earthsystemgrid.org.

[44] The Large Hadron Collider. http://lhc.web.cern.ch/lhc/.

[45] Dan Li, Arun Desai, Zheng Yang, Kenneth Mueller, Stephen Morris, and Dmitry Stavisky. Web content caching and distribution. chapter Multicast cloud with integrated multicast and unicast content distribution routing, pages 109–118. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[46] M. Den Burger and T. Kielmann. Collective receiver-initiated multicast for grid applications. *Parallel and Distributed Systems, IEEE Transactions on*, 22(2):231–244, 2011.

[47] Prasanth Karunakaran, Hamidreza Bagheri, and Marcos Katz. Energy efficient multicast data delivery using cooperative mobile clouds. In *European Wireless, 2012. EW. 18th European Wireless Conference*, pages 1–5, 2012.

[48] Dan Li, Yuanjie Li, Jianping Wu, Sen Su, and Jiangwei Yu. ESM: Efficient and scalable data center multicast routing. *Networking, IEEE/ACM Transactions on*, 20(3):944–955, 2012.

[49] Tatsuhiro Chiba, Mathijs den Burger, T. Kielmann, and S. Matsuoka. Dynamic load-balanced multicast for data-intensive applications on clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 5–14, 2010.

[50] Jonathan S. Turner. Extending ATM networks for efficient reliable multicast. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.6034, 1996.

[51] Hairuo Ma and Magda El Zarki. A new transport protocol for broadcasting/multicasting mpeg-2 video over wireless ATM access networks. *Wirel. Netw.*, 8(4):371–380, July 2002.

[52] IETF Reliable Multicast Transport Working Group. http://datatracker.ietf.org/wg/rmt/charter/.

[53] M. Luby, M. Watson, and L. Vicisano. Asynchronous Layered Coding (ALC) Protocol Instantiation. RFC 5775 (Proposed Standard), April 2010.

[54] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder. RaptorQ Forward Error Correction Scheme for Object Delivery. RFC 6330, Internet Engineering Task Force, August 2011.

[55] L. Rizzo and L. Vicisano. A reliable multicast data distribution protocol based on software fec techniques. In *High-Performance Communication Systems, 1997. (HPCS '97) The Fourth IEEE Workshop on*, pages 116 –125, 1997.

[56] Marinho P. Barcellos, Maziar Nekovee, M. Koyabe, Michael Daw, and J. Brooke. Evaluating high-throughput reliable multicast for grid applications in production networks. In *Cluster Computing and the Grid*, pages 442–449, 2005.

[57] M. Beck, Ying Ding, E. Fuentes, and S. Kancherla. An exposed approach to reliable multicast in heterogeneous logistical networks. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 526–533, 2003.

[58] S. Armstrong, A. Freier, and K. Marzullo. Multicast Transport Protocol. RFC 1301 (Informational), February 1992.

[59] Su Wen, Jim Griffioen, and Kenneth L. Calvert. Building multicast services from unicast forwarding and ephemeral state. *Computer Networks*, 38(3):327–345, 2002.

[60] Jgroups - a toolkit for reliable multicast communication. http://www.jgroups.org.

[61] R. Stewart. Stream Control Transmission Protocol. RFC 4960, Internet Engineering Task Force, September 2007.

[62] J. Widmer and M. Handley. TCP-Friendly Multicast Congestion Control (TFMCC): Protocol Specification. RFC 4654, Internet Engineering Task Force, August 2006.

[63] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM symposium on Communications architectures & protocols*, SIGCOMM '90, pages 200–208, New York, NY, USA, 1990. ACM.

[64] Gerard J. Holzmann. Design and validation of computer protocols, October 1990.

[65] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd ed.)*. Addison-Wesley.

[66] SDL-88 Tutorial. http://www.sdl-forum.org/sdl88tutorial/index.html.

[67] W. R. Stevens. *TCP/IP Illustrated Vol. 1 The Protocols*. Addison Wesley, 1994.

[68] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006. Updated by RFC 6286.

[69] B. Tierney, E. Kissel, M. Swany, and E. Pouyoul. Efficient data transfer protocols for big data. In *2012 IEEE 8th International Conference on E-Science (e-Science)*, pages 1–9, Oct. 2012.

[70] InfiniBand Trade Association. InfiniBand Architecture Specification Volume 1, Release 1.2.1. http://infinibandta.org, November 2007.

[71] Overview of UNH EXS 1.3.0 for Programmers. https://www.iol.unh.edu/services/research/unh-exs/exs-overview.pdf.

[72] H. Subramoni, Ping Lai, R. Kettimuthu, and D.K. Panda. High Performance Data Transfer in Grid Environment Using GridFTP over InfiniBand. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 557–564, 2010.

[73] RDMA Aware Networks Programming User Manual Rev 1.4. http://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf.

[74] The OpenFabrics Enterprise Development (OFED). https://www.openfabrics.org/index.php.

[75] V. Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *Networking, IEEE/ACM Transactions on*, 3(3):226–244, 1995.

[76] IANA. IPv4 multicast address space registry. http://www.iana.org/assignments/multicast-addresses/multicast-addresses.xml.

[77] D. Meyer and P. Lothberg. GLOP Addressing in 233/8. RFC 2770 (Experimental), February 2000. Obsoleted by RFC 3180.

[78] Daniel Massey, Lan Wang, Beichuan Zhang, and Lixia Zhang. Enabling future Internet innovations through transitwire (eFIT). http://www.nets-find.net/Funded/eFIT.php.

[79] R. Atkinson, S. Bhatti, and S. Hailes. Evolving the internet architecture through naming. *IEEE Journal on Selected Areas in Communications*, 28(8):1319–1325, October 2010.

[80] T. Heer and S. Varjonen. Host Identity Protocol Certificates. RFC 6253 (Experimental), May 2011.

[81] J. Pan, R. Jain, S. Paul, and S.-I. Chakchai. MILSA: A new evolutionary architecture for scalability, mobility, and multihoming in the future internet. *IEEE Journal on Selected Areas in Communications*, 28(8):1344–1362, October 2010.

[82] David Clark, Robert Braden, Aaron Falk, and Venkata Pingali. FARA: Reorganizing the addressing architecture. In *Proc. of ACM SIGCOMM Workshop on Future Directions in Network Architecture*, pages 313–321, 2003.

[83] M. Wasserman and F. Baker. IPv6-to-IPv6 Network Prefix Translation. RFC 6296 (Experimental), June 2011.

[84] C. de Launois and M. Bagnulo. The paths toward IPv6 multihoming. *IEEE Communications Surveys and Tutorials*, 8(2):38–50, 2006.

[85] J. Pujol, S. Schmid, L. Eggert, and M. Brunner. Scalability analysis of the TurfNet internetworking architecture. In *Proc. of IEEE GlobeCom*, pages 1878–1883, November 2007.

[86] M. Wasserman and F. Baker. IPv6-to-IPv6 network prefix translation. http://tools.ietf.org/html/draft-mrw-nat66-12, March 2011.

[87] J. Touch, I. Baldine, R. Dutta., G. Finn, B. Ford, S. Jordan, D. Massey, A. Matta, C. Papadopoulos, P. Reiher, and G. Rouskas. A dynamic recursive unified internet design (DRUID). *Computer Networks*, 2011.

[88] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on flat labels. In *Proc. ACM SigComm*, 2006.

[89] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy. Scalable routing on flat names. In *Proceedings of ACM Co-NEXT*, pages 20:1–20:12, 2010.

[90] Lixia Zhang, Deborah Estrin, and Jeffrey Burke et. al. Named Data Networking (NDN) Project. Technical report, 2010.

[91] S. Thomson, T. Narten, and T. Jinmei. IPv6 stateless address autoconfiguration. RFC 4862, Internet Engineering Task Force, September 2007.

[92] B. Carpenter, R. Atkinson, and H. Flinck. Renumbering still needs work. RFC 5887, Internet Engineering Task Force, May 2010.

[93] Tim Chown, Mark Thompson, Alan Ford, Stig Venaas, Christian Schild, and Christian Strauf. Cookbook for IPv6 renumbering in SOHO and backbone networks. Technical report, University of Southampton, 2005.

[94] J. Ubillos, M. Xu, Z. Ming, and C. Vogt. Name-based sockets architecture. http://tools.ietf.org/html/draft-ubillos-name-based-sockets-03, September 2010.

[95] P. Natarajan, F. Baker, P.D. Amer, and J.T. Leighton. SCTP: What, Why, and How. *IEEE Internet Computing*, 13(5):81–85, sept.-oct. 2009.

[96] A. Ford, C. Raiciu, and M. Handley. TCP extensions for multipath operation with multiple addresses. http://tools.ietf.org/html/draft-ietf-mptcp-multiaddressed-02, October 2010.

[97] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241 (Proposed Standard), June 2011.

[98] A. Ahmed, S. Reaz, M. Atiquzzaman, and S. Fu. Performance of DNS as location manager. In *IEEE Int. Conference on Electro Information Technology*, pages 1–6, May 2005.

[99] B. Yahya and J. Ben-Othman. Achieving host mobility using DNS dynamic updating protocol. In *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, pages 634 –638, oct. 2008.

[100] Route Views Project. http://www.routeviews.org/.

[101] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Experimental), January 2013.

[102] RJ Atkinson and SN Bhatti. Identifier-Locator Network Protocol (ILNP) Architectural Description. RFC 6740 (Experimental), November 2012.

[103] Xiaohu Xu and Dayong Guo. Hierarchical routing architecture (hra). In *Next Generation Internet Networks, 2008. NGI 2008*, pages 92–99, 2008.

[104] Anja Feldmann, Luca Cittadini, Wolfgang Mhlbauer, Randy Bush, and Olaf Maennel. HAIR: Hierarchical Architecture for Internet Routing. In *Proc. of ACM ReArch*, December 2009.

[105] Jianli Pan, Subharthi Paul, Raj Jain, and Mic Bowman. MILSA: A mobility and multihoming supporting identifier locator split architecture for naming in the next generation internet. In *Proc. of IEEE GlobeCom*, November 2008.

[106] Xiaowei Yang, David Clark, and Arthur W. Berger. Nira: a new inter-domain routing architecture. *IEEE/ACM Trans. Netw.*, 15(4):775–788, August 2007.

[107] E. Nordmark and M. Bagnulo. Shim6: Level 3 multihoming shim protocol for IPv6. RFC 5533, Internet Engineering Task Force, June 2009.

[108] J. Mitchell. Autonomous System (AS) Reservation for Private Use. RFC 6996 (Best Current Practice), July 2013.

[109] T. S. Eugene Ng and Alan L. Cox. Maestro: An architecture for network control management. http://www.nets-find.net/Funded/Maestro.php.

[110] ARIN number resource policy manual. https://www.arin.net/policy/nrpm.html#five.

[111] National Telecommunications and Information Administration, US Dept. of Commerce. Household broadband adoption climbs to 72.4 percent. http://www.ntia.doc.gov/blog/2013/household-broadband-adoption-climbs-724-percent.

[112] S. Hanks, T. Li, D. Farinacci, and P. Traina. Generic Routing Encapsulation (GRE). RFC 1701 (Informational), October 1994.

[113] C. Perkins. IP Encapsulation within IP. RFC 2003 (Proposed Standard), October 1996. Updated by RFCs 3168, 6864.

[114] J. Lau, M. Townsley, and I. Goyret. Layer Two Tunneling Protocol - Version 3 (L2TPv3). RFC 3931 (Proposed Standard), March 2005. Updated by RFC 5641.

[115] R. Hinden and S. Deering. IP version 6 addressing architecture. RFC 4291, Internet Engineering Task Force, February 2002.

[116] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic Host Configuration Protocol for IPv6. RFC 3315, Internet Engineering Task Force, July 2003.

[117] B. Wellington. Secure domain name system (DNS) dynamic update. RFC 3007, Internet Engineering Task Force, November 2000.

[118] P. Koch. A DNS RR Type for Lists of Address Prefixes (APL RR). RFC 3123 (Experimental), June 2001.

[119] Yang Song, Lixin Gao, and K. Fujikawa. Resilient routing under hierarchical automatic addressing. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5, 2011.

[120] Fujikawa Kenji, Harai Hiroaki, and Ohta Masataka. The basic procedures of hierarchical automatic locator number allocation protocol HANA. In *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, pages 124–131, New York, NY, USA, 2011. ACM.

[121] Habib Naderi and Brian E. Carpenter. A review of IPv6 multihoming solutions. In *The 10th International Conference on Networks*, 2011.