Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2013-011 June 4, 2013

A Publish-Subscribe Implementation of Network Management

Jorge D. Simosa

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

A Publish-Subscribe Implementation of Network Management

by

Jorge D. Simosa

B.Sc. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2012

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 2013

Author .....
Jorge D. Simosa
Department of Electrical Engineering and Computer Science
May 28, 2013

Certified by .....
Dr. Karen R. Sollins
Principal Research Scientist
Computer Science and Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by .....
Prof. Dennis M. Freeman
Chairman, Master of Engineering Thesis Committee

A Publish-Subscribe Implementation of Network Management

by

Jorge Simosa

Submitted to the Department of Electrical Engineering and Computer Science on
May 28, 2013
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

As modern networks become highly integrated, heterogeneous, and experience exponential growth, the task of network management becomes increasingly unmanageable for network administrators and designers.

The Knowledge Plane (KP) is designed to support a self-managing network, given the organizational constraints of network management, as well as to create synergy and exploit commonality among network applications. In this thesis, to build an Information Plane that is suitable to the requirements of the KP, we propose a publish/subscribe system that provides a clear and systematic framework for resolving tussles in the network.

To evaluate the effectiveness of this design, we configured a network of PlanetLab nodes and conducted experiments involving a variety of file sizes and source-destination pairs. The results suggest that the system's performance is not only comparable to existing file transfer services, but that the system also introduces several performance gains that are unattainable with current network architectures.

Thesis Supervisor: Dr. Karen R. Sollins
Title: Research Scientist, Computer Science and Artificial Intelligence Laboratory

This work was funded in part by the National Science Foundation:
Award No. CNS-0915629.

Acknowledgements

It is such a great pleasure to reserve this space to show my appreciation for those who have influenced my life and have helped me reach this important point of my career. I would need an endless supply of paper and ink in order to hint at my enormous gratitude for what these individuals and institutions have done for me.

First, I would like to express my great pride in attending the best university in the world for the past five years. I can still remember the day when I received the acceptance letter. I knew instantly that it would open up so many doors of opportunity. MIT is an incredible place where brilliant minds work together to solve some of the world's toughest problems. It is an institution that not only tested the limits of my mind and body, but also gave me the chance to meet fantastic professors and friends that I will keep forever.

My thesis advisor, Dr. Karen R. Sollins, deserves my sincere thanks for agreeing to allow me to work on such an interesting project. Despite having our first meeting be a virtual one, we both had a high sense of trust and respect for each other's work and collaboration potential. Her sincere passion in helping me learn about network architectures and in providing me a clear view of the big picture has been instrumental to the development of my thesis. Yuxin (Cynthia) Jing, a student researcher in our Advanced Network Architecture group, who was extremely helpful in helping me get accustomed to our group's development environment. George Parisis, co-developer of the PURSUIT architecture, who provided invaluable answers to my never-ending list of technical questions.

To my friends, both inside and outside of MIT, who have made the past few years such a memorable experience for me. Their encouragement, support, and friendship go beyond the instances when we pulled all-nighters, either for work or pleasure. Kevin Zheng, Ken Lopez, Roberto Martinez, and Krishna Settaluri, whose charm and distinctive personalities are the foundation of the strong friendships we have built. Also to all the others from my internships, abroad experiences, research projects, and other life-changing moments, whom I have not yet mentioned here, you also deserve my gratefulness for helping me become the person I am today.

Most importantly, I am very grateful for the wonderful family I have, who have always fully supported me since the day that I was born. My father, Jorge, who instilled in me the importance of humility and love in a way that I cannot begin to describe. He has always been my idol and I hope to make him prouder every day. My mother, Myriam, who always focuses on my well-being and showed me that the hard work always pays off, even if we do not see it immediately. It is something that I live by on a daily basis and has certainly helped me overcome the darkest of nights. My sister, Stephanie, who always shows her enthusiasm and joy for what I have accomplished. I know that I will soon be congratulating her on accomplishments that are greater than mine. Last but not least, to my girlfriend, Daniella, who has inspired me to pursue my dreams and to hold a deep appreciation for life. She has been my partner-in-crime for over five years now and I know that this is only the beginning of what is to come.

Thanks be to God for all that He has given me!

-Jorge

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The architecture of today's networks, such as the Internet, is primarily focused on communication between machines, not necessarily considering them to be two users of information. Protocol suites such as TCP/IP aim to provide a reliable channel between two endpoints that wish to share streams of bytes, regardless of the requirements set by various types of traffic. The rapid and widespread growth of the Internet can be strongly attributed to this machine-to-machine communication principle as it allowed any machine to easily attach to the network and be able to reach any other machine. However, as the demands of applications, such as video streaming and content distribution, increase along with the ever-rising population of users, the capacity of point-to-point based networks becomes insufficient. Trends in today's network traffic suggest a possible paradigm shift from a focus on who is exchanging information to a focus on what is the information being exchanged. Thus, it is worthwhile to explore the potential of information-centric network (ICN) architectures in meeting the demands of tomorrow's applications and users.

One major challenge for the TCP/IP network model of today is the performance of the network under emergency conditions or in a disaster situation. Suppose, for example, that a major wildfire has occurred that encompasses an area that crosses multiple state lines. A vast array of national, state, and local emergency response teams have been assembled in order to contain and eliminate the wildfire before it causes further damages. Each level

of emergency response teams has developed and deployed its own infrastructure to collect critical information and determine the best possible response to the situation. Many of these resources may be invisible to other teams due to the lack of coordination before the events actually occur, resulting in duplicated efforts. Instead of focusing coordination efforts on figuring out who has a specific piece of information, these teams should instead focus on discovering what information is already available in order to make quick and informed decisions. From a network perspective, traditional IP architectures would require end-users to know the correct IP address for the host that has the desired information, whereas an ICN-based architecture would allow a user to request information that the network can quickly locate.

Another major challenge for today's TCP/IP stack is the network efficiency for mobile nodes, particularly nodes that can use multiple technologies to attach to the network. An interesting case is that of a smartphone user, who is moving across a campus while using data-intensive applications, such as video streaming from sites like YouTube, Hulu, and other video broadcast sites. Many of these sites have developed their own ad-hoc network protocols in the hopes of serving their users better; however, these protocols are usually limited to using a single stack of network technologies such as 3G, 4G, GSM, Wi-Fi for the duration of the session. This constraint is an artifact of the WHO-to-WHO communications paradigm that serves as the founding principle for the TCP/IP suite, which requires the user to specify the IP address and port number pair that identifies the destination address of all packets associated with that particular flow of information. Alternatively, the ICN architecture provides a higher-level abstraction, namely the focus on WHAT is being transmitted, which can allow the network to make use of multiple technologies simultaneously in order to provide an improved experience for the user.

Our primary goal for this project is to develop a network architecture for network management, one which provides an automated approach to inferring the state of the network. Currently, network administrators must manually install tools and run scripts to collect various characteristics of the network they are inspecting in order to perform proper tuning and configuration. These inspections require browsing through endless logs of network traffic in an attempt to find common patterns among various error-prone transactions. Instead,

the network architecture can be enhanced to automatically aggregate network statistics, use machine learning to diagnose network failures, and assist network administrators in resolving these inefficiencies. We hope that by placing information at the core of future network architectures, the system will be able to autonomously address these types of issues and enable future applications that can make more efficient use of the underlying network.

Contrary to IP-based networking, information-centric networking relies on the importance of what is being transmitted versus where it is being transmitted. As stated by Jacobson et al., "People value the Internet for what it contains, but communication is still in terms of where," [9] thus it is critical to shift the focus of network architectures towards content. In a typical scenario, nodes participating in an ICN must express interest in information that may be available over the network in order to receive the corresponding data from the publishers. Therefore, both the content providers and the consumers will need to agree on a content identifier system that allows the network to establish matches and create the appropriate communication channels based on the specific information they wish to transfer. We argue that this approach is much more efficient for most, if not all, of the traffic that traverses the Internet and provides enough flexibility to address many of the socio-economic challenges that exist today, such as IP rights, access control, privacy, and other tussles that occur in the networking world [26].

1.2 Overview

This thesis is centered on an implementation of an information-centric network architecture using a publish/subscribe model. The publish/subscribe model focuses on the importance of the distribution of information rather than on the point-to-point connections. In this model, publishers can announce the presence of information that they wish to distribute and make available to the network, while subscribers will request access to that information and receive it for future use. The network will use a hybrid approach to establish matches between publishers and subscribers through a distributed file directory service and a centralized routing service. We do not claim that this is a final solution to meeting tomorrow's network demands; however, we would like to broaden our perspective on the usefulness and

practicality of information-centric networks. In particular, our research will focus on the impact that ICN can have on the performance of network management.

Our system is designed to provide a reliable file transfer service, using the mechanisms provided by the publish/subscribe model, to establish a shared information database that can be used as input into various network management tools. The system includes a scope hierarchy, which mimicks the file hierarchy of the underlying file system in each of the participating nodes, that uses a common naming scheme for files that will be shared in the information-centric network. By agreeing upon the naming scheme, nodes can generate a unique identifier that references a particular piece of data, regardless of who created the file and how it was collected. For example, a node that is interested in receiving the network log of a particular node over the past 24 hours may use an identifier which contains some type of concatenation of the source node identifier, the time period, and the type of data that is stored in the file. As long as the source node uses the same algorithm to create the file name for the data it has collected, these two nodes will be able to share the file asynchronously. Thus, the system will be able to establish a match between the publisher node and the subscriber node based on the file's identifier.

In order to provide the reliable file transfer service, we must introduce additional mechanisms into the publish/subscribe model, due to its asynchronous nature. First, we need to provide a segmentation functionality that allows the publisher to break the file up into chunks that fit within a normal IP packet, while re-using the already established match between the publisher and set of subscribers. We can use a dissemination strategy that performs a longest-prefix match of the identifier, which allows us to simply append a fragment sequence number to the file's existing identifier. Next, we will need to establish a bi-directional channel between the publisher and subscriber for retransmissions requests and responses. We can build a bi-directional channel through the establishment of two uni-directional channels, using a pair of algorithmically generated scopes, that are created when the file is first published to the network. Lastly, we will need to keep track of the file's publication status, which determines the set of tasks that the publisher is expected to do during a given point in time. For example, late-arriving subscribers may or may not result in a start publish event at the publisher, which normally indicates that the publisher

should start publishing all of the fragments for that file. Thus, for files that have an existing match between a publisher and a set of subscribers, the publisher should publish a periodic heartbeat to make sure that all of the subscribers have an opportunity to establish a bi-directional channel as needed.

We can then evaluate the performance of our publish/subscribe system relative to existing file transfer services such as FTP and SCP, by varying the size of the input file. Since FTP and SCP are both designed to serve only a single source node and a single destination node, our experiments for the publish/subscribe system will also include only a single publisher and a single subscriber. We must note, however, that the design of our system supports any number of publishers and subscribers for a given information item, a functionality that is almost non-existant with traditional IP-based approaches. In order to reduce the possible bias in the results, we have selected two networks of 10 nodes each, one for smaller file sizes (100MB and below) and one for larger sizes (100MB and above), where one of the nodes is the designated RV/TM node while the other nodes switch between the roles of publisher and subscriber for different trials.

This thesis makes two major contributions:

- Demonstrates the feasibility of providing a publish/subscribe-based file exchange system that, even without congestion control, is close in performance to existing file transfer protocols.

- Highlights a list of significant hidden benefits to our approach that provide a more suitable file transfer service for an Information Plane that is complementary to the Knowledge Plane.

1.3 Organization of the Thesis

Chapter 2 includes an overview of the challenges in network management as well as a review of related works which have influenced our approach. Chapter 3 describes the primary intellectual contribution of this thesis: the design and implementation of the Information Plane, which provides a simple, yet useful file transfer service that can be used to share

and collect information for the Knowledge Plane. Examples of network management tools that can be incorporated into the Knowledge Plane are presented in Chapter 4 in addition to descriptions of previous work in the Knowledge Plane. In Chapter 5, a detailed performance analysis is provided, which compares the performance of the publish-subscribe file transfer design with common file IP-based file transfer services such as ftp or scp. Chapter 6 concludes this thesis with a summary of our findings, in addition to some key insights and questions that may lead to future work.

Chapter 2

Setting the Stage

In Chapter 1, we have introduced the idea of using an information-centric network archictecture in order to support services such as network management, as opposed to using today's IP-based approaches. We believe that an information-centric network architecture can address some of today's major challenges and support high-performance applications in the future. Section 2.1 will further illustrate challenges in network management that may be resolved through the publish-subscribe model. Section 2.2 provides a conceptual overview of the proposed network architecture, which is implemented and evaluated using an existing network testbed environment. Section 2.3 includes a summary of related works that have provided insight during the design of our approach to future network architectures.

2.1 Key Challenges

At its core, network management is concerned with the performance and behavior of the networks used for transporting pieces of information, which includes backbone, enterprise, wireless, ad-hoc, and other network environments. This type of management requires a wide variety of functions and activities such as disruption diagnosis and repair, prediction, and performance improvements, which usually fall under the responsibility of a single network administrator. The reason that we are proposing a framework for network management is two-fold. The first is that network management problems transcend many local network management domains that have little control or information about each other. The second

is that there are opportunities for complementary activities that are otherwise unavailable or unnecessarily duplicated across several domains.

Network management today rests on the shoulders of only a handful of specialized network administrators, which seems hardly scalable to the ever-increasing size of the Internet. These network administrators are forced to work on an ad-hoc basis and within a narrow localized domain with minimal collaboration across domains. Furthermore, the role of the Internet, as a network of networks, allows for the introduction and growth of unmanaged domains that can severely affect the performance of well-managed domains. Thus, network administrators face an increasing number of problems and constraints, with an overwhelming amount of complexity, that may soon render their actions and efforts useless. We hope to provide a network management framework that is largely autonomous and intelligent to support the expertise of the network administrators as they adapt today's networks to meet the needs of both today's and tomorrow's users.

There a few assumptions we must make when considering the challenges of network management, namely: (1) the extent of network managers' control is limited to their own local network; (2) those administrative domains also reflect proprietary and other policy boundaries; (3) need for network management crossing and extending beyond individual administrative domains; and (4) efficiency and performance, both because responses are often required quickly and because any activities should have minimal impact on the core transport service of the networks involved [21].

The two principal aspects of network management are the convergence of information and the reasoning over that information, as depicted in the following two lists.

Information:

- Storage - collection of information either locally or remotely
- Discovery - determine whether the information required is available
- Find information - enable access to desired information
- Share information - minimizing the amount of duplication and subdividing the total work

- Reason over information - rational reasoning over information outside its original context
- Extensible life and location - information maintains value over an extensive lifetime
- Policy formation/composition - formalizing the control of access, composite policies

Reasoning and computation:

- Nature of information - expect incomplete information, but assume it is statistically representative
- Efficiency/performance - minimize impact on performance while providing adequate functionality, accuracy, and detail
- Decomposition - distribute the computation required over a set of nodes
- Composition - compose tools and computations into more sophisticated ones
- Extensibility - incorporate newer and more effective supporting tools
- Organizing framework - organize functions under differing conditions of physical organization, behavior criteria, and policy constraints

We believe that current network principles and architectures are not flexible and resourceful enough to address many of these issues, thus we hope to evaluate the performance gains achievable through an information-centric architecture that supports network management.

2.2 Elements of the Architecture

We can separate the structure of the network management framework into two parts: the Information Plane and the Knowledge Plane. This abstraction mirrors the two primary goals of network management, namely the collection of information and the analysis of that information. For this thesis, we recognize that some of the major aspects of the Knowledge Plane have already been explored and developed as a result of the efforts of previous student

researchers, thus our focus is to implement a simple information access scheme that allows us to evaluate the performance and efficiency of network management under our proposed framework.

The Information Plane encompasses the collection and distribution of information items, such as measurements and other types of observations, which will be useful for analysis in the Knowledge Plane. An information item has an identifier that is unique within the scopes it is published under. Items can also contain, as part of their data, identifiers pointing to other information items in order to compose complex items. Information items will usually be accompanied by meta-data that describe important characteristics such as size, ownership, access control, and other performance factors. In orderto maintain scalability, we will also need to introduce the concept of regions, which provide a way to partition the information universe by functional or policy boundaries in order to decrease the resource requirements at each individual node of the network. These regions will hold the responsibility of sharing advertisements of information items they know about in order to expand the horizon of the content.

The approach we took to build this Information Plane is based on a publish-subscribe model that allows for information to be made public through an advertising function and for subscriptions to the information to be made through an interest function. This approach allows us to separate the concepts of time, identity, and security in order to build a highly modular network architecture that provides greater functionality than current designs. For example, the sender and receiver no longer have to be in communication simultaneously since the network will automatically keep track of pending requests. In particular, we have built our system on top of the PURSUIT system [24], which already provides features such as identification, publication, subscription and transport of information items, but with no current reference to ontology or higher level functionality, such as reliable fragmentation. We will need to adapt their model to our environment in PlanetLab in order to enable future extensions that will lead to the establishment of a robust network management framework.

The publish-subscribe model featured in the PURSUIT project requires that nodes in the network fulfill the roles of Node, Rendezvous Node, and Topology Manager. A single node can be assigned any combination of these three roles if they wish to participate in

the network. A Node represents the common user that wishes to publish and subscribe to information in the network. The Rendezvous Node takes the responsibility of maintaining a directory of information items available under its scope or its neighbors' scopes as well as keeping track of pending requests from Nodes that fall under its authority. The Topology Manager is in charge of creating paths between publishers and subscribers based on the network's state in addition to directing other management functions. In our prototype implementation, the Rendezvous node and the Topology Manager will be running on a single node for reasons that are described in Section 3.4.

The Knowledge Plane lies above the Information Plane and provides the capability of reasoning over information in order to understand, hypothesize, infer, and act on knowledge. The overall goal is to build a network that is self-knowledgeable, self-analyzing, self-diagnosing, and self-managing. With a focus on network management, we hope to provide more intelligent and effective tools for network administrators as they adapt the network configuration to provide a better experience for its users. One of the major design considerations for this project is to enable the provision and rendezvous with pervasively available and reusable knowledge, which we hope to meet through the interactions between the Information and Knowledge plane [20].

Figure 2.1 illustrates the proposed layered architecture for network management, in which the Knowledge Plane lies above the Information Plane. The Knowledge Plane will provide higher-level tools such as error diagnostics and network monitoring, while the Information Plane will provide functionality such as policy formation and file discovery, in addition to supporting the publish-subscribe mechanism. The underlying network will be composed of Topology Managers (TM), Rendezvous Nodes (RV), and regular Nodes (N). In the case of network management, each node mightl publish the statistics from its perspective of the network while specially designated nodes aggregate the information gathered through subscriptions and present it to the Knowledge plane.

In Figure 2.2, we can observe a possible scenario where a node (Node N) is attempting to investigate the cause of congestion in nodes N1 and N2. The steps in this procedure, represented by the numbered boxes, are outlined as follows:

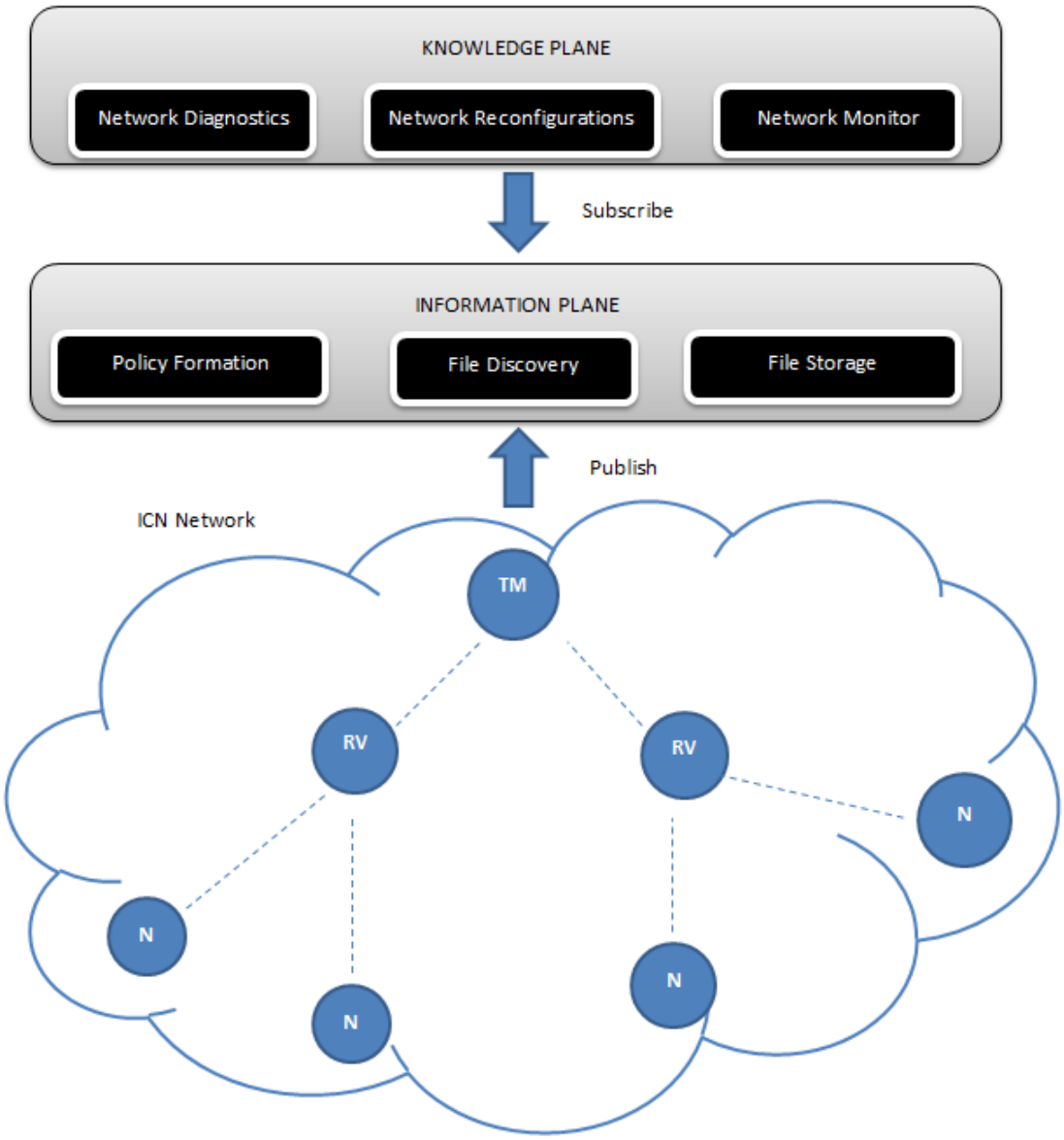


Figure 2.1: Layered Design of Network Manangement

1. Node N talks to its corresponding Rendezvous Node (RV1), who will find the location of network information about nodes N1 and N2.

2. RV1 communicates with its neighboring Rendezvous Nodes who may know about N1 and N2.

3. RV2 responds to the subscription request since it maintains the publications from nodes N1-N5.

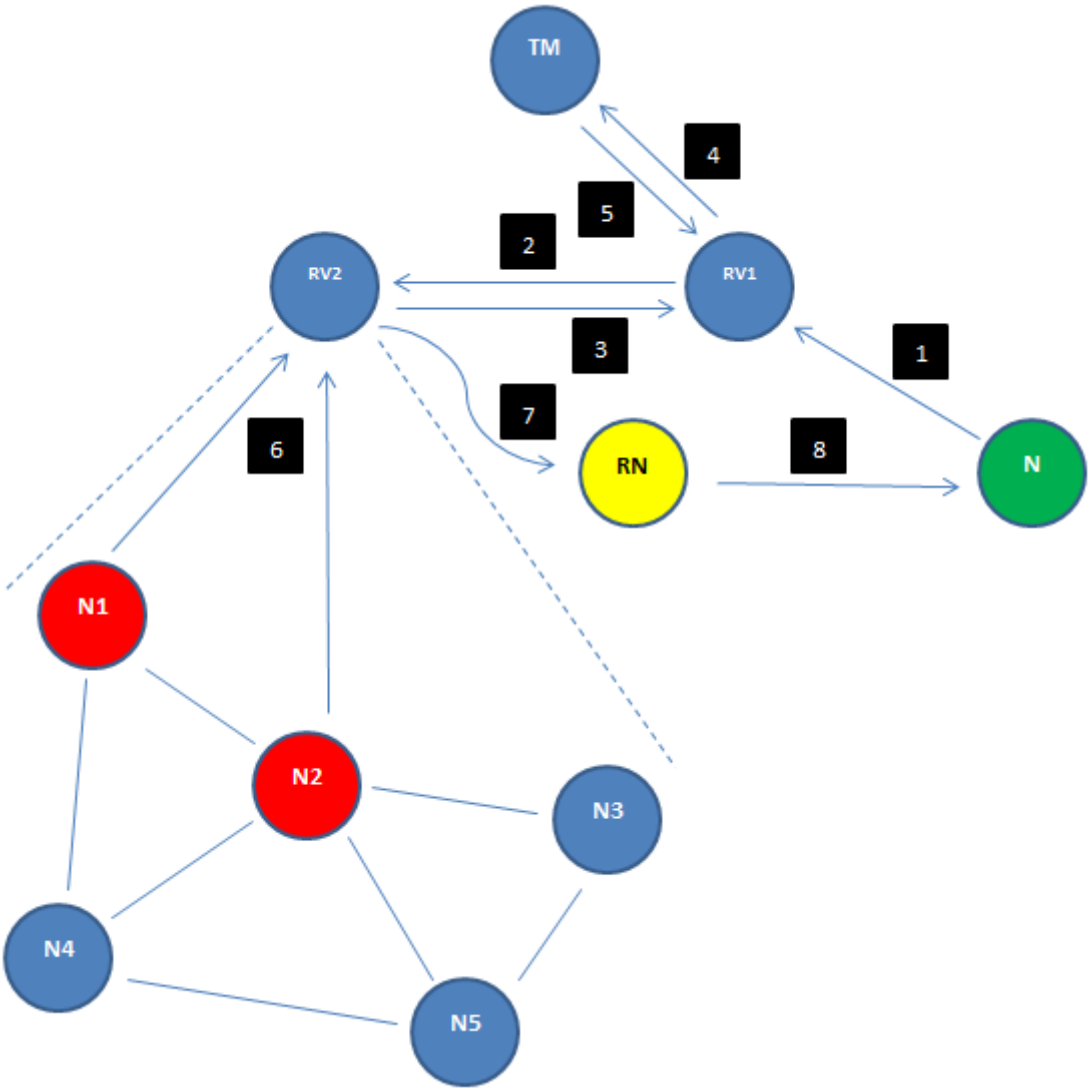


Figure 2.2: Network Congestion Example in Publish/Subscribe System

4. RV1 communicates with the Topology Manager (TM) to find the best path through the network

5. TM informs RV1 and RV2 that the best path goes through RV2 and relay node (RN).

6. RV2 requests and receives the relevant information items from N1 and N2.

7. RV2 forwards the information to RN from both N1 and N2.

8. RN transmits the congestion information to node N.

With our architectural approach to network management, we can raise a few key areas

of study: a) How to effectively manage and organize the distribution of information given the many organizational constraints such as physical topology, legal, social, and economic policy boundaries. b) How to model and evaluate the impact of our architectural proposal in terms of performance and efficiency, considering that network management is secondary to the transmission of user information. c) How to systematically manage information given the constraints on storage, caching, and potential lifetime of information items. d) How to anticipate tussles, i.e. differences of opinions and concerns, when using the network and being able to accommodate them in the network architecture. We believe that by building a prototype of our information-centric approach to network management, we can begin to understand how these challenges play out in the real-world systems.

2.3 Related Works

In this section, we will review a few of the related works that have inspired our approach to building a network architecture that can meet the complex challenges in network management.

2.3.1 Testbed Facility

To start off, we should first gain some understanding of our target testbed system, PlanetLab. PlanetLab is designed as an overlay network that facilitates the design of service-oriented network architectures by allowing both designers and users to interact and validate new networking technologies. The PlanetLab nodes are primarily hosted at research institutions all across the world in order to provide an avenue through which researchers can test their ideas on a distributed system that is representative of the Internet. The system is divided up into slices that are available to researchers, which allow experiments to run simultaneously on a global scale with minimal interference. We chose PlanetLab as our testbed not only due to its flexibility in terms of management and programmability, but also because of its role as a network substrate that experiences congestion, failures, and diverse link behaviors [17], which we can use to evaluate the performance of our architecture.

2.3.2 Information-Centric Networking

As suggested by Ghodsi et al. [7], we can find various proposals for data-oriented or content-centric network architectures, which can be all categorized into the research area of information-centric networking. Despite the varying terminologies among these proposals, there are a few key points that all of the major proposals share, indicating a convergence of ideas as to designing a network archictecture for the future.

First, the publish/subscribe paradigm that provides the foundation for many of these systems, has been around for over 25 years, thus it is not a new concept. The proposals may vary the names of their primitives, however, one of the primitives usually corresponds to publish, which enables information providers to advertise the availability of their content, while another primitive corresponds to subscribe, which enables consumers to request content. Second, in ICN designs, when a network element receives a request for content, it does one of two actions: (i) if it has the data cached, it can respond with the content directly, or (ii) if it does not have the content cached, it can request the content from its peers and then cache the content when this request is filled. Last, since the ICN approach results in content arriving from network elements other than the originating server, the security model cannot be based on where the packet came from; instead, ICN designs must secure the content rather than the path. All ICN designs thus adopt a content-oriented security model in which content is signed by the original content provider, so that network elements and consumers can verify the validity of the content merely by verifying the signature.

Content-Centric Networking [9] (CCN), which was developed at the Palo Alto Research Center, pursues the broader goal of promoting a communications architecture built on named data rather than remaining within the confines of the current network architecture that focuses on where information is being sent. They introduce the CCN network layer, along with the strategy and security layers, as an architecture that not only secures content itself, but also takes maximum advantage of multiple simultaneous connectivities, such as 3G, Bluetooth, and Ethernet. Their architecture is based on two CCN packet types: Interest and Data, which allows consumers to ask for content that the network can provide either on a cached or dynamic basis. In order to encourage incremental integration, many of their

design choices are based on common features of the TCP/IP architecture, which seems very constraining for our goal of a generalized framework. Instead of limiting ourselves to end-to-end routing based on a human-interpretable identifying name, our approach, which is based on the PURSUIT project, will use the rendezvous function as a network primitive to allow the network to have more flexibility in terms of routing, access control, and other network functions.

The PURSUIT architecture [25], which is based on work from the PSIRP project, serves as the foundation for our architectural approach to network management. Its design goals have a broader scope of expressing the notions of who, what, and why within the network and removing that burden from application developers. The authors believe that by embedding the needs and concerns of application developers and users in the network, network architects and managers will be able to adapt the system more dynamically and efficiently than the point solutions of today. By using the publish-subscribe function as the centerpiece of their architecture, they claim that the network will be able to better serve its users by closely matching functionality with expectations. We believe that their approach may be a viable option towards building the Knowledge plane, given the similarities in motives, e.g. being able to express the socio-economic tussles that degrade the performance of today's networks. We hope to adapt a smaller subset of their system to fit the characteristics of the PlanetLab environment in order to evaluate the performance gain of network management from an IP-based system to an ICN-based system.

2.3.3 Network Management

CoMon [16], a monitoring system for PlanetLab, provides us with an example of a system that gathers observations from a distributed set of nodes and provides some post-aggregation analysis of the system performance. Some of the primary usages of CoMon include: "Sufficient" monitoring, Community-aided problem identification, Login troubleshooting, and Node Selection, which are all specifically built for the PlanetLab community. One of the major concerns that established the need for a monitoring system was the fact that resource allocation was not very reliable at the time, where well-behaved experiments often suffered from poorly-implemented ones, which is very similar to the networking dilemma

that application performance is heavily dependent on the reliability of the underlying network resources. In fact, in order to support their Node Selection features, they opted to include a script-like interface that allowed users to find statistics on a per-node basis that satisfied their specific query, which mirrors our design principle of structuring knowledge through an XML-based ontology language. We hope to provide a more generalized approach to network management that can be applied to any network, not limited to PlanetLab.

One of the examples that closely resembles our vision of a self-managing network is the iPlane project. The designers of iPlane describe it as a scalable service providing accurate predictions of Internet path performance for emerging overlay services [13]. It adopts a systematic approach to predicting end-to-end performance based on the composition of statistics from each segment of the path. This method has shown to be very effective for overlay applications, such as BitTorrent and VoIP, which have required the use of application-specific approaches to estimating the state of the network. However, their approach relies heavily on the probing of an IP-based architecture, which keeps the information and knowledge layers tightly dependent. Moreover, their system concentrates all of the computation and analysis required to build the annotated Internet map at a central agent, which limits the performance of the system to the amount of available resources at the central node. We hope to pursue a more event-driven, distributed, and modular approach to building the Information and Knowledge planes for network management.

Our survey of related works would not be complete without a reference to the pioneering idea of embedding a Knowledge Plane in the network architecture, described by Clark et al [4]. They believe that rather than producing a wide variety of point solutions to overcome today's challenges, we should instead build a completely new layer in the network that makes it self-knowledgeable, self-managing, and resourceful for users, application developers, and network administrators alike. The paradigm of treating the underlying communications technologies as best-effort services has reached the limits of its potential as we see more and more users and network managers become frustrated with their inability to understand how the network is behaving. By introducing intelligence to the network, the system may eventually be able to make defensible decisions about how to resolve network failures and avoid inefficient network configurations. We hope to contribute towards this goal of a

self-managing network by building and evaluating a systematic approach towards network management that can certainly extract value from a shared network knowledge base.

Chapter 3

Designing an Information Plane

This chaper presents the main focus of this thesis: the design of the Information Plane that is centered on a publish/subscribe model of the network. Much of the underlying infrastructure is based on the publish/subscribe model presented in the PURSUIT project, thus several of the design concepts and function calls stem from their architecture [24] [25].

Section 3.1 lists some key definitions that will be useful to understand the dynamics of our information plane. Sections 3.2 and 3.3 describe the role and functions of the publisher and subscriber, which are located at the edge of the ICN and serve the primary interface with the real-world. These sections will outline the steps that each node takes in order to share information across the network and contain a discussion about current assumptions and their implications for future designs. Section 3.4 describes the role of the Rendezvous node (RV) and the Topology Manager (TM) which, in this current design, are both located on a single node in order to minimize failures and latency in the network. Section 3.5 explains how this publish/subscribe model of the network is useful for file distribution and how that can lead us to the design of a robust Information Plane.

3.1 Definitions

Our design of the Information Plane deals with two main objects: information items and scopes. An information item represents an object that contains some piece of data or an entire file, which in our case might be a log of network traffic, packet traces, and other

types of statistics about the network. Scopes, on the other hand, are similar in nature to the concept of directories, which allow you to build a hierarchy of items by grouping related items and scopes together.

An information item can be any form of data that is relevant for a particular user or application, such as dynamically generated content, a policy rule for other data, or simply pointers to other user-created data. Each information item is identified with a statistically unique identifier, which is referred to as a Rendezvous Identifier (RId), that is self-generated and follows the naming convention used by the publisher of the data. Scopes define a set of information that are related for a given problem space and, since they are treated as information items themselves, can form a hierarchy of scopes. Each information item in the network must be placed in at least one scope and must have an Rid that is unique to the scope they are assigned to. Scopes are identified through a Scope Identifier (SId) that is statistically unique within the scope in which it is placed. Figure 3.1 shows an example of an information graph that contains multiple root scopes, a hierarchy of scopes, and several information items.

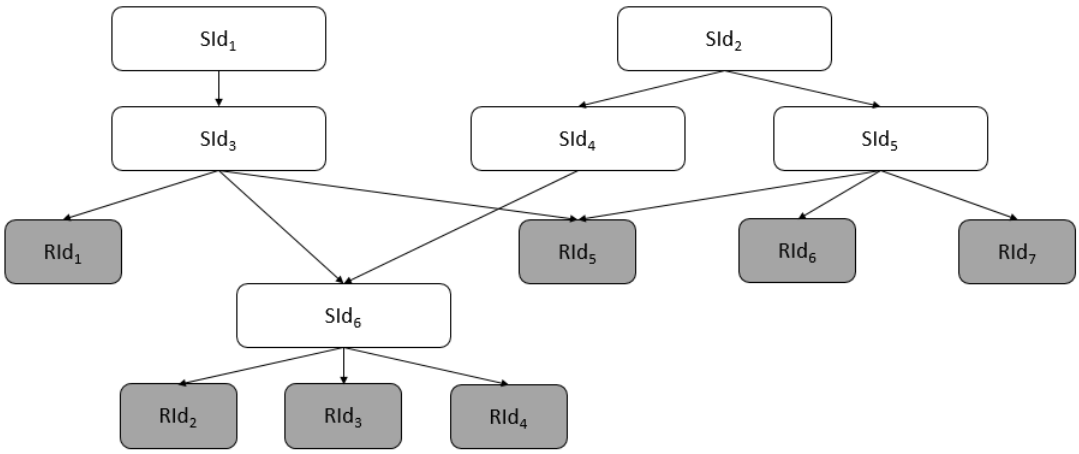


Figure 3.1: Information Scoping in PURSUIT Architecture

With the current version of the underlying ICN architecture, based on the PURSUIT project, the identifiers for either an item or scope must have a length equal to PURSUIT_ID_LEN, set to eight bytes in our current implementation. In order to support variable-sized naming, our design prepends empty spaces to the identifier as necessary in order to fit the 8-byte identifier chunk. Furthermore, we can support the idea of replicating

a file hierarchy by breaking up the full file path into eight byte chunks and recursively issuing commands based on each portion of the identifier from the start to the end of the path, which is useful for our file distribution example presented in Section 3.5.

Similar to the idea of links in a file system, our network model allows the same piece of information to be published under different scopes as long as the RId remains unique under both scopes. Alternatively, the user can generate two different RIds representing the same file in the local storage, in which case the system would treat them as two different information items even though they contain the same data.

In addition to the SId, scopes are also characterized by their associated dissemination strategy, which defines parameters such as the governance structure for identifier creation, data representation formats, and the required implementation instance of the network functions: rendezvous, topology formation, and forwarding. The Rendezvous function is concerned with the matching of publishers' availability of information and subscribers' interest in it, allowing for the asynchronous arrival of publish and subscribe events. The topology formation and management function involves using the rendezvous information in order to create a suitable delivery graph for the transfer of content. The forwarding function can then use the topology information to deliver the packets that are generated by both the publisher and subscriber. Together, these functions provide the underlying infrastructure that supports our publish-subscribe model of the network, while allowing for flexibility in determining the policies for a given scope or information item.

As will be shown in the later sections, a node can choose to either publish or subscribe items and scopes by issuing commands that include the object's identifier as well as the identifier of the parent scope, which is processed by the Rendezvous Node and Topology Manager (defined in Section 3.4). In our network model, the only information we store about an information item or scope is the identifier itself, the list of publishers and subcribers for that object, and the associated dissemination strategy. All of the meta-data and data must be provided by the nodes themselves whenever they receive an event that instructs them to do so.

All of the nodes in the network, including Publishers, Subscribers, Rendezvous nodes, and Topology Managers, make use of forwarding identifiers in order to communicate with

each other and participate in the network. A forwarding identifier depicts the path that should be used in order to reach some subset of the nodes in the network. During the initialization of the network, pre-defined forwarding identifiers are created to establish links between each node and its neighbors, each of which has an associated, semantics-free node identifier that is unique within the local domain. Whenever a publish or subscribe event is received at a Rendezvous node, it may need to ask the Topology Manager to create or update a forwarding identifier that matches the publisher with the current set of subscribers.

3.2 Publisher Node

The overall goal of our proposed network architecture is to be able to detect and diagnose network failures, in addition to taking steps to resolving the issues. In order to perform diagnosis, for example, the system would need to collect information about the network traffic in the area of the network failure from neighboring nodes. Information such as packet traces and logs of network statistics, which are usually collected on a per-node basis, are some examples of the information that can be useful for detecting network failures. We can assume that each node has enough processing capability to record both the incoming and outgoing traffic, in order to collect performance metrics of the network independently from other nodes. However, many network management tools require information from several nodes, not just a single node, to perform accurate analysis of the state of the network. Thus, each node should not assume that the information it collects is only useful for its own analysis, but rather it can contribute to a larger database of network information that can be used be higher-level nodes to monitor and adjust the network in an efficient manner.

The publisher node of our system serves as the entry point of real-world information into the ICN, where files are made available to other nodes in the networks through advertisements, i.e. a publish event. The publisher starts by publishing a scope using the publish_scope(string ID, string prefixID, Strategy st) function, which creates a new scope in the information graph. If the prefixID is empty, the scope is published as a root scope. Otherwise, the prefixID can hold a variable number of SIds that together form a scope path from the root to the parent scope. Publishing and re-publishing a scope triggers a notifica-

tion towards existing subscribers of the parent scope specified by prefixID. The published scope will serve as an identifier for a set of related information items and child scopes within a possibly large scope hierarchy, both in terms of spread and depth.

A publisher can then advertise the availability of an information item through the publish_info(string ID, string prefixID, Strategy st) function, which creates a new information item with the specified ID under the scope identified by prefixID. According to the dissemination strategy st, the piece of information will be published under a scope in the respective RV node, provided that the scope already exists and the item's dissemination strategy is compatible with the parent scope's strategy.

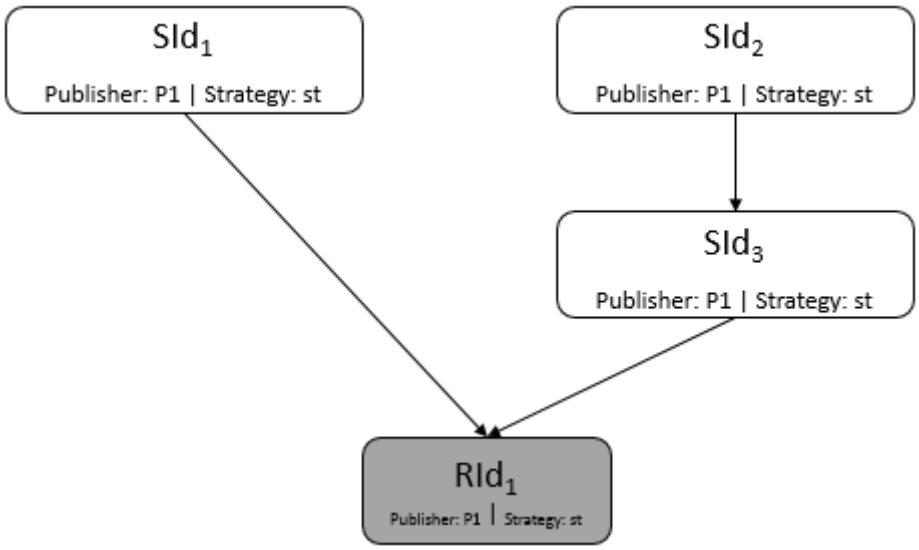


Figure 3.2: Simple Publish Graph

In Figure 3.2, we can see an example of an information graph that is created by a single publisher P1 that includes three scopes and a single information item. To create the root scopes SId_1 and SId_2, the publisher must issue two commands: publish_scope(string SId_1, string(), Strategy st) and publish_scope(string SId_2, string(), Strategy st). The scope SId_3 is also published in a similar manner: publish_scope(string SId_3, string SId_2, Strategy st). Since the information item will be published under two scopes, we will need to issue two commands: publish_scope(string RId_1, string SId_1, Strategy st) and publish_scope(string RId_1, string(SId_2+SId_3), Strategy st) where the + operator represents the concatenation of two strings.

The actual transfer of data happens after the RV node notifies the publisher of matching subscribers, thus the end-user applications are responsible for storing all of the published information items until one or more subscribers for these items appear, given the expected lifetime of the data. After learning about existing subscribers, the publisher can begin to publish data for a specific information item by using publish_data(string ID, string prefixID, Strategy st, char *data), which forwards data according to the delivery graph created by the Topology Manager. In some cases, however, a dissemination strategy may foresee the existence of subscribers and call for the publication of data without prior explicit notification, which is useful for applications such as video streaming. Section 3.5 will describe the steps required in order to perform reliable file transfer with these publish function calls.

The publisher may also decide that it is no longer wants to publish a particular scope or information item and issue a UNPUBLISH_SCOPE or UNPUBLISH_INFO event that informs the Rendezvous node to update the object's entry in the Rendezvous table accordingly.

3.3 Subscriber Node

The primary role of the subscriber node in our proposed design is to collect various information items that are adverstised through the network, which can be later passed on as input files to the Knowledge Plane. The Knowledge Plane, which would include various network management tools, can then analyze the files according to its strategies and goals. Therefore, the subscriber can be viewed as an intermediary between the Information Plane and the Knowledge Plane since it will need to subscribe and receive information items from the network, which are later converted into input files for the analysis tools.

While a publisher node is considered to be the entry point of real-world data into the network, a subscriber can be thought of as an exit point for information items, moving them from back into the real-world. The subscriber can start by subscribing to a scope using the subscribe_scope(string ID, string prefixID, Strategy st) function, which indicates an interest in receiving notifications about scopes and information items contained within the scope. The prefixID represents a scope path from the root to the parent scope and the

ID is the identifier of the requested scope. Subscribing to a scope does not result in any received data at the subscriber other than notifications about published items or scopes, thus the node would need to subscribe to each information item or scope individually if it is interested in receiving that data. In a later design, we may decide to provide a listing of the information items and scopes currently published underneath the specified scope when a node subscribes to it, perhaps through a special meta-data file that is maintained by the scope's publisher, but for simplicity we chose to omit this functionality in our current design.

In order to receive data, i.e. files, from the network, the subscribe will need to generate a SUBSCRIBE_INFO event that indicates its interest in receiving the data for that particular information item, despite the fact that the file may not yet be published. The subscriber can use the subscribe_info(string ID, string prefixID, Strategy st) function, after finding the proper identifier and parent scope for the item it is requesting. This implies that the subscriber should know, a priori, the naming convention and scope hierarchy that the publisher will be using. In future designs, we may also want to provide some type of search capability within the Information Plane, which takes a query and finds the matching files. However, since IP-based network management tools also require pre-defined identifiers and known locations of these items, we find the current version of our system to be sufficient in providing an equivalent functionality. A clear advantage of the publish/subscribe model is that the network will record all of the subscription requests, even if the item is not currently published. After the item is published, the network can automatically match up the publisher with the set of already existing subscribers.

In Figure 3.3, we can observe that Subscriber S1 has subscribed to two different information items by issuing the following commands: subscribe_info(string RId_1, string SId_1, Strategy st) and subscribe_info(string RId_2, string(SId_2+SId_3), Strategy st). Even though the information item RId_1 has no exisiting publishers, the system records the set of subscribers who anticipate the publication of that information item. On the other hand, since the information item RId_2 does have an existing publisher P1, the Rendezvous node will match the publisher and subscriber by sending a notification to Publisher P1 to begin publishing the data fragments for the information item.

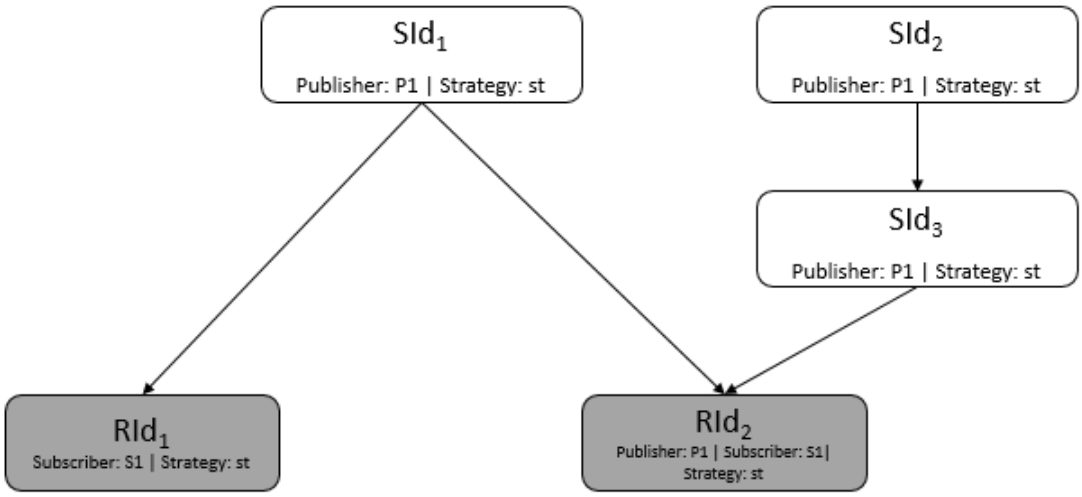


Figure 3.3: Simple Subscribe Graph

The subscriber may also decide that it is no longer interested in a particular scope or information item and issue a UNSUBSCRIBE_SCOPE or UNSUBSCRIBE_INFO event that informs the Rendezvous node to update the set of subscribers for the object accordingly. For our file distribution scheme, described in Section 3.5, the subscriber node will always unsubscribe from an information item after it correctly receives all of the file's fragments, which assumes that files do not change after being published to the network. This feature will prove to be essential in determining when to activate our heartbeat mechanism that accounts for late-arriving subscribers.

3.4 Rendezvous Node and Topology Manager

In our system, the Rendezvous Node in conjuction with the Topology Manager provide the underlying functionalities of an information item directory service as well as a centralized routing service. As mentioned previously, our current design uses a single Rendezvous Node and a single Topology Manager, which are both located within a single node. The advantage of this approach is that we can decrease the latency of interactions between the Rendezvous Node and Topology Manager, thus allowing us to provide faster service for publisher and subscriber nodes. Unfortunately, it also brings a few disadvantages, namely it introduces a single point of failure for the network and would not be scalable for a larger network

of nodes. Our group is currently working on enabling multiple Rendezvous nodes within a single network, which can coordinate to distribute the responsibility of tracking publications and subscriptions, for a later revision of our system. In the future, we would also want to enable multiple Topology Managers, in order to distribute the workload across multiple nodes, but unfortunately we have not gained enough experience with the architecture to provide this in the current system.

The Rendezvous node is primarily in charge of maintaining the table of publications and their respective subscribers. Whenever it receives a publish event, it can refer to the table to see if the publication exists and identify the set of subscribers, if they already exist, who should receive a notification about the publish event. If it determines that the publication is not part of its current table, it can create a new entry by recording the item's identifier, the parent scope's identifier, and the identifier of the node who generated the publish event. Additionally, the Rendezvous node can process subscription requests by looking for the desired information item or scope within its current table. If it does not find the appropriate entry, it can choose to either store the subscription request by creating a new entry that does not have a current publisher or, if it finds that another Rendezvous node holds responsbility for the item, asks the Topology Manager to update the delivery graph between the publisher and the set of subscribers by updating the forwarding identifier.

Type	Prefix ID	Object ID	Strategy	Publishers	Subscribers
Scope	*empty*	SId_1	st	P1	*empty*
Scope	*empty*	SId_2	st	P1	*empty*
Item	SId_1	RId_1	st	*empty*	S1
Scope	SId_2	SId_3	st	P1	*empty*
Item	$SId_2 + SId_3$	RId_2	st	P1	*empty*
Item	SId_1	RId_2	st	P1	S1

Table 3.1: Rendezvous Table Example

Table 3.1 illustrates an example of the Rendezvous Node's table of publications, based on the information graph presented in Figue 3.3. One thing to point out is the existence of the last two entries of the table, representing the same information item RId_2, which suggests

two different information items in the network even though Publisher P1 knows that they are referencing the same piece of data. This allows the Publisher P1 to unpublish one of the items without affecting the other, which may result from policy and security concerns. A subscriber would need to provide the prefix ID when subscribing to an information item in order to be added to the correct entry in the Rendezvous Node's table.

The Topology Manager holds the responsibility of providing a routing service both during the initialization of the network and during its use, particularly by establishing forwarding identifiers that represent a channel between a publisher and a set of subscribers. It is designed to find the optimal forwarding paths from publishers towards subscribers, taking into account dissemination policies and current network conditions. Information about the state of the network is provided by helper functions, which continously collect the relevant network statistics. The initialization of network connectivity will entail the instantaneous dissemination and gathering of network knowledge, in which case each node could announce its existence by broadcasting a scope whose scope ID is equivalent to the node's ID. In similar fashion, it could continue to learn about the network connectivity if nodes correctly infer their neighbors through the broadcasts and recursively broadcast new scopes using the scope ID of the received broadcast as the parent ID. The details of this mechanism are described in the PURSUIT deliverable [25], however, for our purposes, we can assume that the Topology Manager will have a broad enough view of the network topology, thus it will be able to create a delivery graph that may include the use of relay nodes, which are not specifically subscribed to the item but can help reach all of the subscribers in an efficient manner.

3.5 File Distribution

This section describes the steps required in order to support reliable file transfers between a publisher and a set of subscribers in our publish/subscribe model of the network. In short, the publisher will advertise files that it currently holds in storage and wishes to make available to other nodes in the network through a publish event, while the subscribers will indicate their interest in a particular file by generating a subscription event that the Ren-

dezvous Node will process. The Rendezvous Node and Topology Manager will work together to establish a delivery graph between a publisher and the matched set of subscribers.

In order to provide a reliable file transfer service, we must complete several steps:

- Recursively publish/subscribe both information items and scopes based on their complete identifiers
- Provide segmentation functionality for large files that allows both publishers and subscribers to identify the location of a fragment within the file
- Establish a bi-directional channel between the publisher and subscriber for retransmission requests and replies
- Determine the state of the file transfer based on publish events
- Account for late-arriving subscribers by publishing periodic heartbeats

One must note that with our current design, the publisher and subscriber must predetermine and agree upon the process by which the scope hierarchy is generated, in addition to knowing the naming convention used to generate the full identifier of the information item. They may choose to overload the information item's identifier in order to display certain characteristics, or tags, of the file it represents. Similarly, we can potentially provide a search feature through an appropriate hierarchy of scopes, where the scopes themselves also provide clues about the characteristics of information items and scopes beneath it.

We can first simplify the expected structure of the scope hierarchy by imitating the underlying file system through a hierarchy of scopes that represent the full path of any given file. For example, a file whose full path on the local disk may look like /rootdir/dir1/dir2/exampleitem.bin should have an equivalent path in our network's Rendezvous tables that looks like /rootscope/scope1/scope2/exampleitem.bin.

Since the identifiers in our architecture must have a fixed length, we support variable length file paths by prepending additional spaces as necessary. In particular, we begin by splitting the file path (excluding the filename itself) by using the "/" character as a token. For each intermediate directory in the path, we prepend the spaces in order to make the component's name have a length that is a multiple of PURSUIT_ID_LEN. We can then

recursively publish scopes starting from the first directory in the path until to the parent directory of the file itself, while publishing intermediate scopes as necessary. In a similar manner, we can parse the filename itself by prepending spaces to the beginning of the filename and recursively publishing scopes until we reach the last PURSUIT_ID_LEN-sized chunk, which is used as the information item's identifier in our information graph.

In Figure 3.4, we can trace through the steps involved in converting a file path, defined by the local storage, into an appropriate hierarchy of scopes that is compatible with the underlying PURSUIT architecture. The instructions listed on the left side of the figure outline the main steps of our algorithm, which supports variable-sized components of the file path by appending the necessary amount of empty spaces to generate 8-byte scope identifiers. The algorithm can then specify the parent scope of a given file path component by using the modified file path up to that point. Note that for simplicity, our algorithm ignores components that are either "." or "..", which is used in Linux to represent a link to the current directory and the parent directory respectively, since in some cases we may not have access to the referenced directory's name. Thus, one could say that our scope hierarchy scheme flattens the hierarchy of the file path to only contain the properly named directories. Nonetheless, the scope hiearchy that is generated through our algorithm for the example file path is depicted on the right side of the figure.

The subscriber can then take a similar approach, assuming that it already knows the expected path of the file it is interested in. However, it will only need to subscribe to the direct parent scope of the information item and the information item itself. As mentioned previously, our publish/subscribe model also naturally supports the concept of links, where an item can "reside" within multiple directories, by allowing a publisher to publish the same information item under two different scopes. Our system, however, currently does not support file system-defined links, such as soft links in Linux, we expect the publisher to have a full copy of the file residing within the parent directory. In a later revision of our design, we may decide to allow a publisher to specify whether it is publishing a file or a shortcut to an already existing file.

Our design also provides a segmentation feature that allows publishers and subscribers to share large files across the network by breaking up files into fixed-size chunks that can fit into

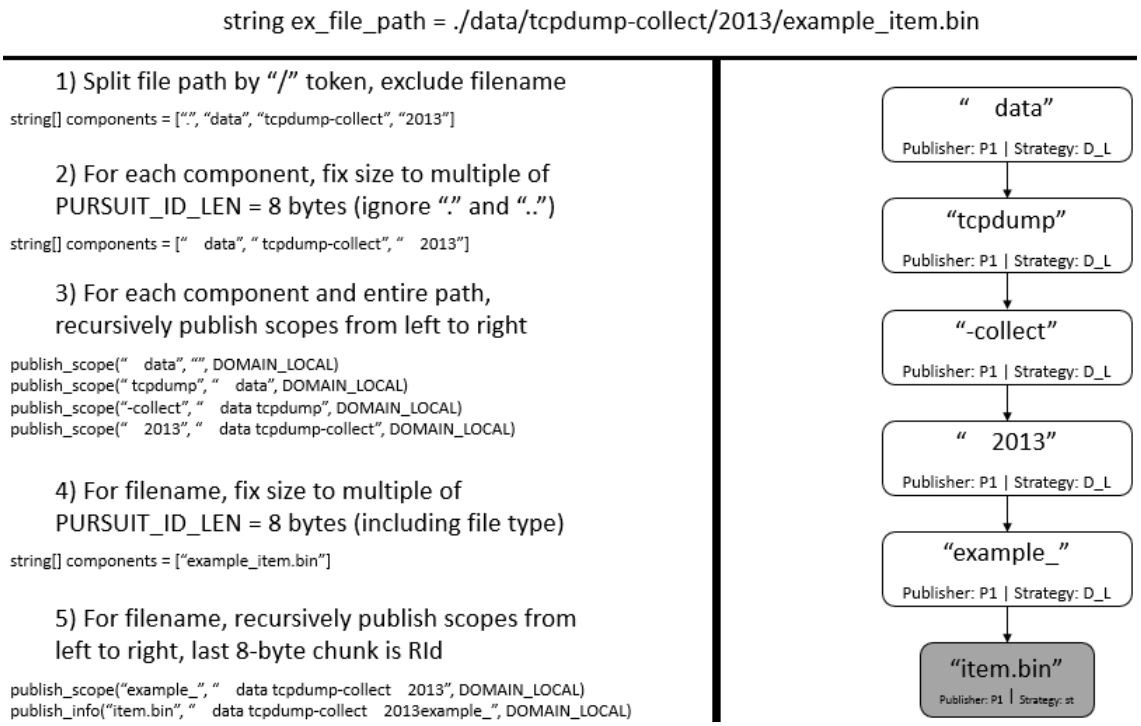


Figure 3.4: Conversion from File System Hierarchy to PURSUIT Scope Hierarchy

a transmissible packet. Since our Information plane currently resides above the standard IP layer, we must design a packet structure that fits within an encapsulated IP packet, whose size is usually on the order of 1,500 bytes. Therefore, the Publisher node will generate fragments that contain 1,024 bytes of data and generate an appropriate fragment identifier which includes both the identifier of the information item and the packet's sequence number. Moreover, our design also includes the total number of fragments for a given information item within the identifier of the published fragments, allowing the subscriber to deduce the expected number of fragments and build a bitmap that tracks the set of already received fragments.

The segmentation feature of our design is portrayed in Figure 3.5, where the file item.bin is split into chunks of 1,024 bytes as shown in the top right corner of the figure. If we assume that the information graph is the one shown in the left side of the figure, we can then build a fragment identifier that includes the original item's identifier as well as two other values: the total number of fragments n and the fragment number F_j of the current chunk. The fragments are then published using the publish_data command as illustrated in the bottom

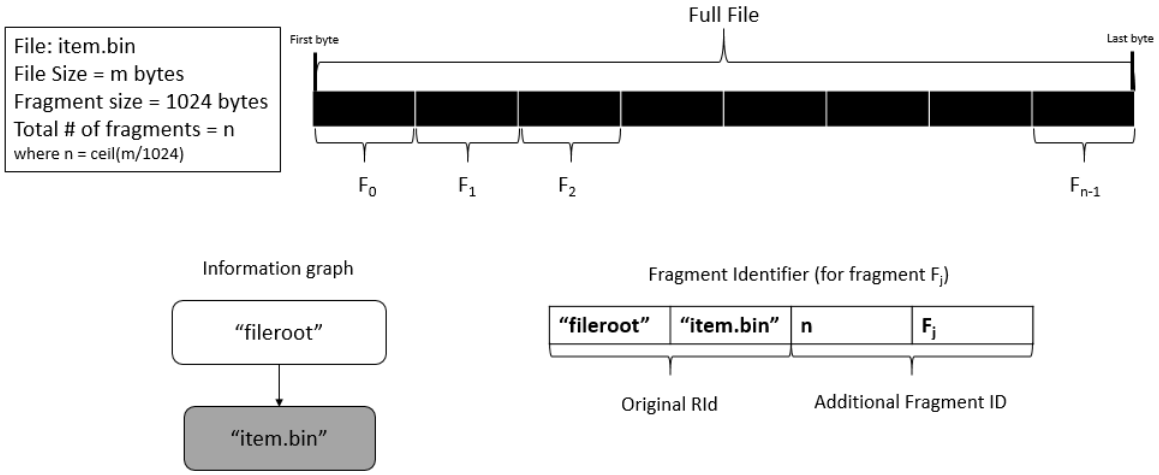
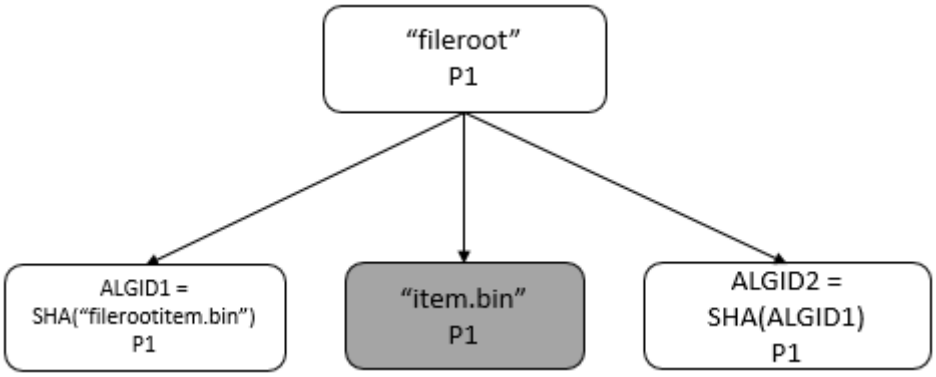


Figure 3.5: Fragmentation Scheme

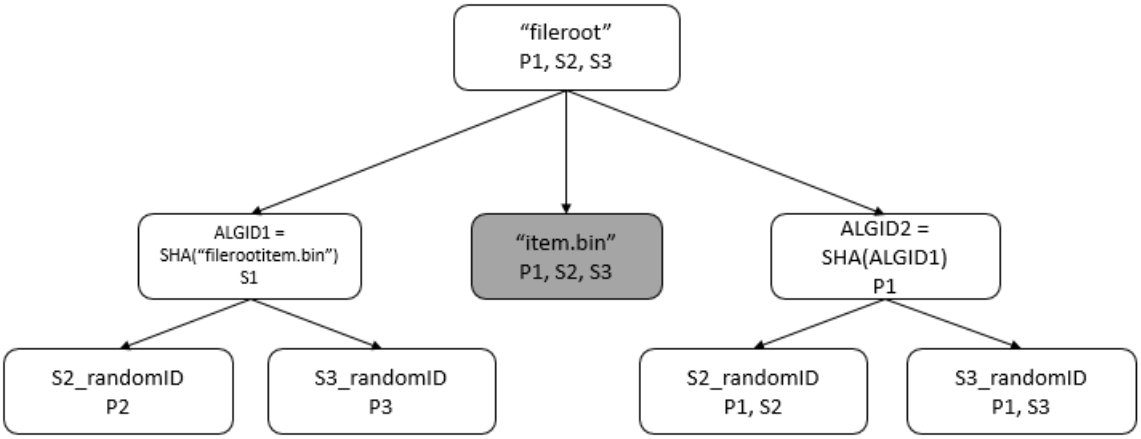
of the figure. The DOMAIN_LOCAL value is used for the strategy field, since it instructs the Rendezvous node to use longest-prefix matching to identify an already existing forwarding identifier created for that particular information item "filerootitem.bin", which is provided as the prefix ID, in order to correctly delivery the data fragments of the file to the set of subscribers.

Due to the nature of our publish/subscribe model of the network, we must take additional steps in order to create a bi-directional channel between the publisher and each subscriber, which will be used in order to transmit retransmission requests and replies. We can seperate the bi-directional channel into two single uni-directional channels, one in which the original publisher will publish retransmitted fragments to the subscriber and the other in which the subscriber will publish retransmission requests to the publisher. Our current design establishes the scope for both channels on a per-item basis whenever the information item is initially advertised to the network by calculating the SHA-1 hash of the item's full identifier, which is used as the identifier for the scope where retransmission requests will be published, and the SHA-1 hash of the output from the previous hash function, which serves as the identifier of the scope where retransmitted fragments will be sent to the subscriber.

Figure 3.6 depicts two main stages of our reliability scheme, which involves creating scopes with algorithmically generated identifiers for retransmission requests/replies. Recall



(a) Information Graph with Algorithmically Generated Scopes after Item Publication



(b) Information Graph after new Subscribers

Figure 3.6: Reliability Scheme for the Publish-Subscribe Model

that we can solve the problem of establishing a bi-directional channel between the publisher and a subscriber by initializing two uni-directional channels. Whenever a publisher,such as N_1 in this example, decides to publish an information item, in addition to publishing the item under the parent scope, it also publishes two additional scopes. The first scope, denoted by ALGID1 in Figure 3.6(a), represents the scope for retransmission requests, which the publisher automatically subscribes to. The second scope displayed in Figure 3.6(a), denoted by ALGID2, is used by the publisher to reply to retransmission requests, thus it automatically publishes this scope during the first stage. Figure 3.6(b) illustrates the second stage of our reliability scheme, which is triggered after the first matching subscription request. In this example, there are two nodes who issue subscription requests for the information item, namely nodes N_2 and N_3. Each of the subscribers will generate a random ID that will be used as the identifier for both uni-directional channels within their respective

scopes, represented in Figure 3.6(b) as S2_randomID and S3_randomID. Since the publisher is already subscribed to ALGID1, it will receive notification of any published retransmission requests and reply with the missing fragments accordingly.

To determine the set of packets that should be included as part of the next retransmission request, the subscriber does two things: a) maintain a bitmap, whose length is equal to the total number of fragments, tracking the received packets by setting its corresponding bit value in the bitmap to 1, b) initiate a timeout loop that decrements a timeout counter for each fragment such that when the counter reaches zero, the corresponding packet is deemed to be lost and its sequence number is included in the next retransmission request. In order to reduce the number of retransmission requests that the publisher might receive, the subscriber attempts to request only consecutive sets of fragments by providing the sequence number of the first lost fragment and the total number of consecutive lost packets behind it. Since the subscriber linearly scans the bitmap before generating the next retransmission request, this design choice seems to be a natural choice.

An information item may be in one of three primary states: published with no subscribers, published with subscribers, or subscribed with no publishers. Whenever a publisher advertises an information item through a PUBLISH event, the item enters the published with no subscribers state, in which the item's identifier and the identifier of the publisher node are stored within the directory table of a Rendezvous node. The publisher should not publish any pieces of data to the network since none of the other nodes have expressed an interest in that information. However, when another node does subscribe to that particular information item, the item enters the published with subscribers state, in which the subscriber expects to receive all notifications and data that pertain to that particular item. The publisher will receive a START_PUBLISH event, which indicates the presence of a first subscriber, as a signal to begin publishing all of the data that is referenced by the published information item. The current system, however, does not generate START_PUBLISH events after each new subscriber, therefore, as will be described below, our design must include a mechanism that accounts for late-arriving subscribers.

After all of the subscribers decide to remove their interest in the information item by generating UNSUBSCRIBE_INFO events, the information will return back to the published

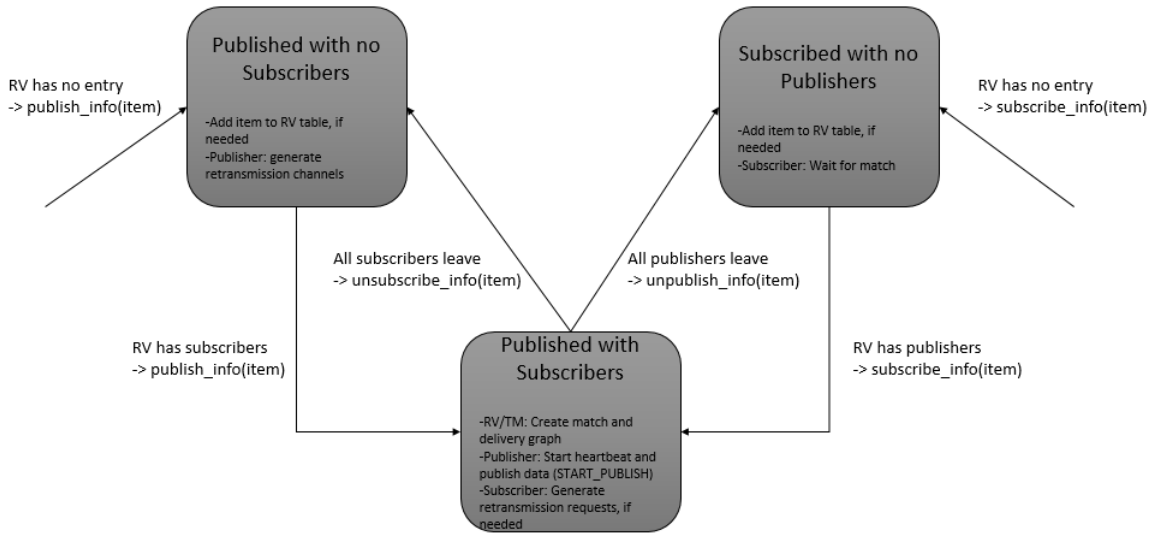


Figure 3.7: Publication State Diagram

with no subscribers state. The subscribed with no publishers state is particularly interesting since it allows subscribers to express their interest in an information item that is currently not published, suggesting that even though the item does not currently exist, the subscribers expect that it will be published in the future. Therefore, when the information is finally published to the network, the Rendezvous node can automatically create a match between the publisher and the set of subscribers, which will lead the publisher to immediately begin publishing the available data. Figure 3.7 illustrates the various states of an information item and includes a description of the transitions between them and the main tasks for the nodes within each state.

In order to account for late-arriving subscribers, who may or may not result in a START_PUBLISH event, our design of the publisher includes a heartbeat mechanism, which periodically publishes a packet for each information item that is currently in the published with subscribers state. In particular, our design chooses to publish the first fragment of the file as the periodic heartbeat, although alternatively one could publish meta-data about the information item. Upon reception of the heartbeat, late-arriving subscribers will initialize their timeout loops and begin to generate retransmission requests for the rest of the fragments, which results in the eventual reception of all fragments at each subscriber without overloading previous subscribers.

Chapter 4

Integrating the Knowledge Plane

This chapter presents a brief overview of the Knowledge Plane as well as some examples of tools that may be useful for network management systems. Section 4.1 provides further insight into the concept of the Knowledge Plane, including examples of previous work within our group. Section 4.2 contains a description of tcpdump, a popular command-line packet analyzer, that could be used by nodes to generate data that will be shared in the Information Plane. Similarly, Wireshark, a more comprehensive network protocol analyzer, is commonly used as both a data collection and data analysis tool, as outlined in Section 4.3. A simpler data analysis tool, namely TCP Trace, can be used to generate rates of traffic and track sessions based on packet capture files from other tools, as explained in Section 4.4. Section 4.5 describes CAIDA, an organization that not only provides access to anonymized network logs but also provides links to tools that perform various kinds of analysis on these files, e.g. CoralReef (Section 4.5.1) and iatmon (Section 4.5.2).

4.1 Overview and Previous Work

The Knowledge Plane, as described by Clark et al. [4], involves a control plane that manages the underlying Information Plane in an effort to enable a self-managing network. At a high level, the Knowledge Plane gathers observations and constraints, to which it applies reasoning in order to generate responses to any queries about the state of the system. It requires the cooperation of many hosts and servers in sharing information, regardless of

whether or not it was the original source of the data. By embedding concepts of knowledge representation and dissemination, incorporating machine learning techniques, supporting policies on trust and security, in addition to numerous other mechanisms, the Knowledge Plane represents a network archictecture that is fully capable of addressing new challenges in network management and the increasing requirements of network applications. The rest of this section will review a few key contributions made by previous researchers in our group in building a Knowledge Plane, thus motivating the need to evaluate the applicability of a publish/subscribe model for the Information Plane.

Li [11] presents the concept of agents, which is a participant that works together with others to perform a task such as collecting packet traces or running sophisticated intrusion detection techniques. More interestingly, these agents may also issue requests to other agents in the form of a message that looks for an answer to a particular problem or delegates tasks to achieve a common goal. For example, intrusion detection becomes very difficult when the attacker spreads the malicious traffic workload over multiple paths in the network, however, an agent can send requests to numerous local detectors in order to perform a more powerful aggregate analysis. From a higher-level perspective, the Knowledge Plane can be separated into two parts: 1) network knowledge plane (NetKP): which is an application-independent mechanism that collects general characteristics about the network, 2) specialized KPs (Spec-KPs): which are application-specific, specializing in different areas in order to achieve a certain level of functionality under a given set of constraints. In this manner, the agents can collaborate together within the network knowledge plane or, perhaps for security reasons, focus on a particular region or problem that is traditionally handled within a local domain.

In a similar train of thought, Lee [10] introduces an approach to fault diagnosis based on a Common Architecture for Probabilistic Reasoning in the Internet (CAPRI) in which distributed, heterogeneous agents with different capabilities and goals conduct diagnostic tests based on shared observations, beliefs, and knowledge to probabilistically infer the causes of failures in the Internet. Since diagnosis usually requires the communication of diagnostic information among multiple diagnostic agents, CAPRI provides a common language for the representation and communication of network diagnostic information through an extensible, distributed component ontology. Additionally, CAPRI provides a common

service description language that enables agents to describe diagnostic capabilities simply in terms of their inputs and outputs. In this manner, agents can advertise their own diagnostic capabilities in order to enable the aggregation of multiple specialized services. This idea of advertising diagnostic capabilities mirrors the concept of advertisement in the Information Plane, where a publisher indicates the presence of a file that is available through a publish event.

Beverly's dissertation [2] addresses network management from a machine-learning perspective, suggesting that network architects should find ways to embed intelligence into the network, both at the core and at the end-nodes. Adding intelligence to end nodes is not an entirely new concept, given that they already contain a variety of intelligent functionalities, however, there is still room for nodes to gather and use data in non-traditional ways to their advantage. On the other hand, adding intelligence to the network core is a very contentious idea among network architects but, given the ever increasing demands of network applications, is an idea that should be explored, as exemplified by the concept of the Knowledge Plane. Beverly identifies three major areas that can benefit from learning including: optimizing network performance, mitigating security threats, and learning as a fundamental element in network architecture. We believe that embedding intelligence into the network architecture, through a substrate such as the Knowledge Plane, is the key to meeting the demands of future network applications and thus hope to contribute to the cause through the development of the Information Plane.

The remaining sections of this chapter include several examples of network analysis tools, whose functionalities mirror those we would expect to see in the Knowledge Plane. An interesting thing to note is that these tools tend to produce similar outputs given the same dataset, thus we believe it would sensible to allow applications to share data, through our publish/subscribe-based system, rather than having them independently produce the same results. This would both increase the efficiency of computations and network traffic as well as enable the aggregation of results as input into tools that perform higher-level analysis.

4.2 tcpdump

tcpdump [8] is commonly referred to as a packet sniffer, since it can intercept all of the incoming and outgoing packets through a particular network interface and record various kinds of information, such as the packets themselves, packet headers, or some other aggregated form of network traffic information. The program allows the user to specify whether its output should show up directly in the terminal, since it is designed as a command-line library, or send the captured information to an output file that can be saved for later use. In order to read already existing capture files, the user can include the '-r' option that instructs tcpdump to read from a file rather than perform live recording of a network interface. For most of the network management scenarios that we consider, it is very likely that the output of a tool like tcpdump would be recorded on the local disk storage and published to the Information Plane.

You are allowed not only to specify the network interface you wish to capture packets from, but also to define a particular expression that is used by tcpdump as a filter to retrieve information only from matching packets. For example, one could specify a particular output port or destination to only capture packets from traffic streams that correspond to a specific applications. Since tcpdump is familiar with many of the popular network protocols, one can specifically capture network protocol packets such as SYN and FIN packets from TCP traffic streams. tcpdump can also be configured to capture only a specified number of packets, in addition to the default mode of collecting all packets until it is issued an interrupt signal by the user.

```
tcpdump -c 25 -r equinix-chicago.dirB.20080319-200100.UTC.anon.pcap 'ip[2:2] > 64'
reading from file equinix-chicago.dirB.20080319-200100.UTC.anon.pcap, link-type RAW (Raw IP)
16:01:00.000014 IP 19.146.240.144.12085 > 230.170.121.29.2278: P 3498443313:3498443382(69) ack 3756245607 win 64919
16:01:00.000017 IP 165.45.138.226.49152 > 128.207.210.165.1728: . 3243287490:3243288950(1460) ack 441904190 win 65333
16:01:00.000019 IP 61.83.97.60.30163 > ec2-54-244-111-1.us-west-2.compute.amazonaws.com.53395: . 695335398:695336858(1460) ack 3106266752 win 65535
16:01:00.000020 IP 245.216.116.215.21288 > 61.82.68.246.https: P 1398672966:1398673166(200) ack 2449590191 win 65389
16:01:00.000020 IP 224.211.152.117.nntps > 226.200.172.60.1177: . 734640205:734641665(1460) ack 4035894740 win 6432
16:01:00.000022 IP 224.211.152.117.nntps > 226.200.172.60.1177: P 1460:2920(1460) ack 1 win 6432
16:01:00.000023 IP 224.211.152.117.nntps > 226.200.172.60.1177: . 2920:4380(1460) ack 1 win 6432
16:01:00.000031 IP 243.89.132.245.57992 > 241.67.149.9.www: . 3215971573:3215972861(1288) ack 1083770272 win 65535 <nop,nop,timestamp 687358541 2284678592>
16:01:00.000036 IP 63.204.236.201.www > 7.44.29.238.2310: . 745050270:745051730(1460) ack 3255932966 win 6432
16:01:00.000042 IP ec2-54-214-220-213.us-west-2.compute.amazonaws.com.3074 > 237.92.209.252.33533: UDP, length 72
16:01:00.000042 IP 52.192.10.63.51107 > 237.60.11.36.1511: P 1636260734:1636260780(46) ack 4119912515 win 16602
16:01:00.000051 IP 50.181.84.83.3646 > 227.45.12.232.6699: P 1996704751:1996704969(218) ack 1142427186 win 64240
16:01:00.000067 IP static-122-0-24-218.mykris.net > 237.211.117.217: [|ESP]
16:01:00.000069 IP 252.73.111.121 > 240.136.127.217: [|ESP]
16:01:00.000092 IP 122.73.96.78 > 61-30-106-117.static.tfn.net.tw: [|ESP]
16:01:00.000095 IP 242.2.205.226.www > host-92-19-160-120.as13285.net.4838: . 4036099291:4036100705(1414) ack 3953474217 win 65535
16:01:00.000099 IP 239.170.161.229.13864 > 247.188.250.218.24490: UDP, length 172
16:01:00.000128 IP 103.48.28.141.60009 > 7.225.110.79.8633: UDP, length 281
16:01:00.000142 IP sapnfsnt1.atlanta.hp.com.14837 > 227.118.129.199.1097: . 766030264:766031724(1460) ack 1986233363 win 65387
16:01:00.000148 IP 243.26.251.61.www > n003-000-000-000.static.ge.com.54067: . 3693794198:3693795646(1448) ack 3687298057 win 63243 <nop,nop,timestamp 1611782695 537273793>
16:01:00.000152 IP 238.78.158.122.www > 246.192.226.6.50316: . 3447769066:3447770506(1440) ack 1221134317 win 54 <nop,nop,timestamp 3369575295 182134860>
16:01:00.000153 IP 243.26.251.61.www > n003-000-000-000.static.ge.com.54067: . 1448:2896(1448) ack 1 win 63243 <nop,nop,timestamp 1611782695 537273793>
16:01:00.000154 IP 166.24.166.25.3034 > 224.125.59.10.1935: P 3512660158:3512661179(1021) ack 1001213436 win 65472
16:01:00.000156 IP 243.26.251.61.www > n003-000-000-000.static.ge.com.54067: . 2896:4344(1448) ack 1 win 63243 <nop,nop,timestamp 1611782695 537273793>
16:01:00.000158 IP 63-255-126-25.ip.mcleodusa.net.www > 11.208.236.49.42701: . 1620746965:1620748379(1414) ack 1576534668 win 11312
```

Figure 4.1: tcpdump Example Output

Figure 4.1 illustrates a simple use case for tcpdump, we configure the program to read from an already existing file and apply a particular filter that selects packets of interest. The '-c' option indicates the number of matching packets to find before exiting, while the '-r' option allows us to specify an input file to read from, which in this case is a pcap file generated from another packet capture tool. Alternatively, we could have also captured live data from our network interface by omitting the '-r' option and instead providing the name of the network interface that we wish to observe. More importantly, tcpdump accepts an expression field that identifies the characteristics that the packets of interest must have. In this case, we wanted to focus solely on the first 25 IP packets that have a length greater than 64 bytes.

4.3 Wireshark

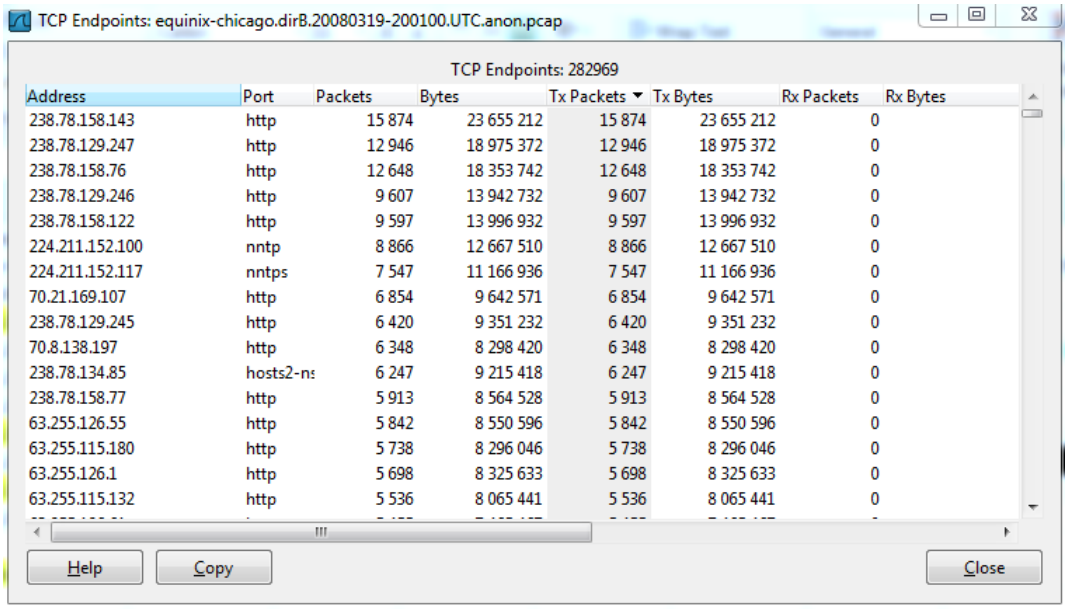
Similar to tcpdump, Wireshark [5] can also perform live captures of packets traversing a local network interface, in addition to offering many other features such as a GUI and promiscuous mode capturing. The GUI provides a very user-friendly view of the captured packets and allows you to display/hide certain characteristics of the packet, such as packet length, payload, checksums, etc. The program can also be configured to install TShark, a text-based command-line library, which is nearly identical to tcpdump. Wireshark accepts a variety of input file types including pcap, Cisco Secure IDS iplog, and Microsoft Network monitor, and supports various types of offline analysis. It even allows the user to modify existing input files and create new ones, a feature which was used for our experiments that involved varying file sizes.

Wireshark is also able to perform deep packet inspection of packets from hundreds of protocols, a number that continues growing as developers contribute their work. This enables the application to display human-interpretable information, within the GUI, about each packet it has captured and, in some cases, color codes packets from different streams for increased readability. Additionally, it provides the opportunity to generate various types of visualizations representing different characteristics of the set of captured packets. All together, Wireshark supports a wide variety of application scenarios, such as: network

administrators troubleshooting network problems, network security engineers examining security problems, developers debugging protocol implementations, and users learning the internals of network protocols.

Traffic	Captured	Displayed	Marked
Packets	3938771	3938771	0
Between first and last packet	8.128 sec		
Avg. packets/sec	484606.548		
Avg. packet size	644.232 bytes		
Bytes	2537482964		
Avg. bytes/sec	312199125.073		
Avg. MBit/sec	2497.593		

(a) Wireshark's Summary Feature



(b) Wireshark Filtering and List Display

Figure 4.2: Wireshark Example Outputs

Figure 4.2 displays a couple of partial screenshots of the Wireshark GUI, which allows you to perform a variety of analysis techniques and present the results in a user-friendly manner. As mentioned previously, Wireshark is designed to accept a variety of input files that contain traces of network traffic, including the pcap files that we are using for the experiments. Figure 4.2(a) contains a bird's eye-view of some of the common network statistics we can easily measure for a stream of packets, such as total number of bytes

and the amount of time between the first and last recorded packet. Similar to tcpdump, Wireshark supports the use of expressions to find packets of interest, as shown in Figure 4.2(b). In this example, we are focusing on captured packets that are part of TCP streams and ordered them according to the total number of packets transmitted from a particular endpoint.

4.4 TCP Trace

Developed by Shawn Ostermann, TCP Trace [15] is a network analysis tool that specifically targets TCP dump files, i.e. files with captured packets from TCP sessions. The TCP dump files, captured by programs such as tcpdump, snoop, and etherpeek, are processed in order to calculate various statistics such as elapsed time, bytes and segments sent and received, etc. In particular, the three types of text outputs are: a) detailed statistics, which includes list of common network metrics such as average segment size and throughput, b) RTT statistics, which estimates the RTT between a TCP sender and receiver using the timestamps, and c) CWND statistics, which estimates the size of the congestion window during the TCP session.

TCP Trace also includes graphing tools that provide a visualization of the behaviors of the TCP sessions that are being analyzed. For example, a time sequence graph that shows the general activity and events that happened over the duration of the TCP session, such as congestion window updates and retransmission requests. The throughput graph displays the estimated throughput value over each time interval of the TCP session, while the RTT graph is an analogous illustration of the RTT estimate during each time interval. There are a number of other supported graphs which together help form a better picture of the characteristics of the TCP session that is being analyzed. One can imagine using a tool like TCP Trace to diagnose particular instances of network failures as well as to evaluate the efficiency of TCP variants.

The TCP Trace program can interpret most types of network dump files and produce an output that aggregates packets according to the TCP flow they are a part of, as depicted in the example output in Figure 4.3. By grouping packets together according to their traffic

```
tcptrace -b -n -f'packets>2000' equinix-chicago.dirB.20080319-200100.UTC.anon.pcap
Output filter: ((c_packets>2000)OR(s_packets>2000))
1 arg remaining, starting with 'equinix-chicago.dirB.20080319-200100.UTC.anon.pcap'
Ostermann's tcptrace -- version 6.6.1 -- Wed Nov 19, 2003

TCP packet 50252: reserved bits are not all zero.
        Further warnings disabled, use '-w' for more info
3938771 packets seen, 3391656 TCP packets traced
elapsed wallclock time: 0:00:17.754068, 221851 pkts/sec analyzed
trace file elapsed time: 0:00:08.127771
TCP connection info:
*** 40520 packets were too short to process at some point
        (use -w option to show details)
 50: 239.173.84.100:54929 - 240.107.64.7:80 (cu2cv)        2255>    0<  (unidirectional)
191: 130.160.140.236:60731 - 240.136.111.19:4630 (nq2nr)   4500>    0<  (unidirectional)
619: 238.78.158.121:80 - 3.150.46.20:24109 (auo2aup)       2193>    0<  (unidirectional)
665: 238.78.158.122:80 - 228.64.139.205:61835 (ayc2ayd)    2142>    0<  (unidirectional)
750: 54.52.22.187:80 - 242.12.150.240:50581 (beq2ber)      3429>    0<  (unidirectional)
751: 238.78.158.77:80 - 3.183.14.242:51763 (bes2bet)       2338>    0<  (unidirectional)
942: 63.164.39.153:1798 - 134.121.187.153:80 (btk2btl)     2250>    0<  (reset)  (unidirectional)
1067: 54.245.55.45:64549 - 54.244.157.236:80 (cda2cdb)      2098>    0<  (unidirectional)
1101: 60.43.159.226:36421 - 60.235.209.85:20542 (cfq2cfr)   2175>    0<  (unidirectional)
1258: 238.78.158.77:80 - 7.31.16.171:3359 (crs2crt)         2172>    0<  (unidirectional)
1681: 238.78.151.65:80 - 9.242.200.21:4574 (dyg2dyh)        2320>    0<  (unidirectional)
2270: 238.78.129.245:80 - 230.170.145.164:3475 (fro2frp)    2166>    0<  (unidirectional)
2674: 57.144.148.96:54459 - 227.249.48.117:34648 (gwq2gwr)  2078>    0<  (unidirectional)
3177: 57.201.174.37:3256 - 60.235.213.138:22 (iji2ijj)      2198>    0<  (unidirectional)
3740: 240.5.95.168:80 - 3.216.26.163:57004 (kaq2kar)        2202>    0<  (unidirectional)
5308: 54.245.55.15:55801 - 54.244.157.209:80 (org2orh)      2005>    0<  (unidirectional)
7103: 54.245.55.54:61657 - 54.244.157.236:80 (tzi2tzj)      2175>    0<  (unidirectional)
8394: 238.78.129.246:80 - 90.62.42.128:62740 (xuq2xur)      2117>    0<  (unidirectional)
15275: 238.78.158.76:80 - 230.172.225.22:51934 (asdy2asdz)   7422>    0<  (unidirectional)
23520: 54.245.54.163:80 - 3.216.26.138:55466 (bqoe2bqof)     2026>    0<  (unidirectional)
75845: 54.2.216.129:80 - 242.86.141.216:2645 (hpje2hpjf)     2511>    0<  (unidirectional)
79704: 242.12.242.27:20 - 134.121.185.221:3057 (iaua2iaub)   2611>    0<  (unidirectional)
123374: 57.201.211.74:80 - 224.132.73.217:11095 (mzzg2mzzh)   2788>    0<  (unidirectional)
142989: 239.210.196.130:80 - 242.12.150.248:53544 (pgac2pgad) 2955>    0<  (unidirectional)
165690: 54.245.54.153:80 - 239.127.121.66:53007 (rvei2rvej)   4440>    0<  (unidirectional)
```

Figure 4.3: TCP Trace Example Output

stream, i.e. between a single source-destination pair, TCP Trace simplifies the task of identifying the packets that correspond to a particular TCP session, a task that is usually performed manually by the network administrator but is unscalable for large dump files. TCP trace also provides an option to display the output on a graph, e.g. a time sequence or an RTT estimate graph, however, since our example file is anonymized and primarily contains unidirectional traffic, we could not produce an interesting example to include here.

4.5 CAIDA

The Cooperative Association for Internet Data Analysis (CAIDA) [22] is a collaborative undertaking among organizations in the commercial, government, and research sectors aimed at promoting greater cooperation in the engineering and maintenance of a robust, scalable global Internet infrastructure. CAIDA's mission statement is to investigate practical and

theoretical aspects of the Internet in order to: a) provide macroscopic insights into Internet infrastructure, behavior, usage, and evolution, b) foster a collaborative environment in which data can be acquired, analyzed, and (as appropriate) shared, c) improve the integrity of the field of Internet science, d) inform science, technology, and communications public policies. It aims to achieve these goals through three main program areas: a) Research and Analysis, 2) Measurements, Data Procurement, and Curation, c) Data and Tools.

For the purposes of this thesis, we primarily focus on the Data and Tools component, since it provides us with anonymized packet trace files that can serve as input into our Information Plane and references examples of network analysis tools. The files used in the experiments presented in the Results section of this thesis (Chapter 5), were derived from existing trace files provided by CAIDA. Additionally, as will be presented in Sections 4.5.1 and 4.5.2, we found various tools, supported by CAIDA, that are examples of the tools and features that we expect the Knowledge Plane to incorporate in the future.

4.5.1 CoralReef

CoralReef [23] is a comprehensive software suite developed by CAIDA to collect and analyze data from passive Internet traffic monitors, either from live captures or already existing trace files. The package includes support for standard network interfaces and specialized high performance monitoring devices, as well as applications for capture, analysis, and web report generation. The CoralReef 3.0 release introduced the libcoral library, which provides an API that developers can use to create new Internet traffic analysis tools. Additionally, it includes a suite of software solutions for monitoring and analysis of network traffic data. The software suite is designed in layers, such that the drivers for network interfaces are located at the lower layer and HTML reports that can present output from various analysis programs are located at the higher layer.

Figure 4.4 includes sample output from CoralReef's crl_stats program, which analyzes a network traffic capture file and calculates various statistics about the underlying traffic streams. For example, we can easily determine the total number of unique IPv4 addresses as well as the balance between unique sources and unique destinations, which might be particularly interesting for a network user who is designing a client-server application. The

```
./crl_stats pcap:~/Desktop/caida_gz_files/equinix-chicago.dirB.20080319-200100.UTC.anon.pcap
Maximum capture length for interface 0:          unknown
First timestamp:                      1205956860.000000000
Last timestamp:                       1205956868.127771000
Unknown encapsulation:                                    0
IPv4 bytes:                                      2535396713
IPv4 pkts:                                          3938619
IPv4 flows:                                          354560
Unique IPv4 addresses:                               282641
Unique IPv4 source addresses:                        167767
Unique IPv4 destination addresses:                   114878
Unique IPv4 TCP source ports:                         47094
Unique IPv4 TCP destination ports:                    39702
Unique IPv4 UDP source ports:                         40421
Unique IPv4 UDP destination ports:                    31397
Unique IPv4 ICMP type/codes:                             14
IPv6 pkts:                                              152
IPv6 bytes:                                           33505
non-IP protocols:                                         0
non-IP pkts:                                              0
```

Figure 4.4: CoralReef Example Output

online documentation for CoralReef includes the description of an example toolchain that results in a live HTML graph displaying certain features of the network traffic that the program is capturing. As mentioned earlier in this section, CoralReef also includes the libcoral library which allows designers to utilize already existing CoralReef tools to create new network analysis tools, an idea that resonates well with the extensibility of the Knowledge Plane to target specific network management requirements.

4.5.2 iatmon

A common practice to study malicious activity on the Internet is to employ network telescopes which record unsolicited one-way Internet traffic. However, since traffic levels continue to increase at an unimaginable rate, it has become more difficult to aggregate sets of one-way traffic, thus leading Nevil Brownlee to develop iatmon [3] (Inter-Arrival Time Monitor), a freely available measurement and analysis tool that allows you to separate one-way traffic into clearly-defined subsets. It can be described as a monitor that reads network trace data from a file or live interface, builds a hash table of source addresses for one-way traffic, and writes summary files describing the one-way sources.

Figure 4.5, replicated from Brownlee's paper, provides an interesting story about the nature of traffic workloads from January 1st to July 1st of 2011 as observed from his machine. We can observe that some of the common traffic types include uTorrent, TCP and UDP, UDP probes, among others. While some traffic types vary greatly over time, such as

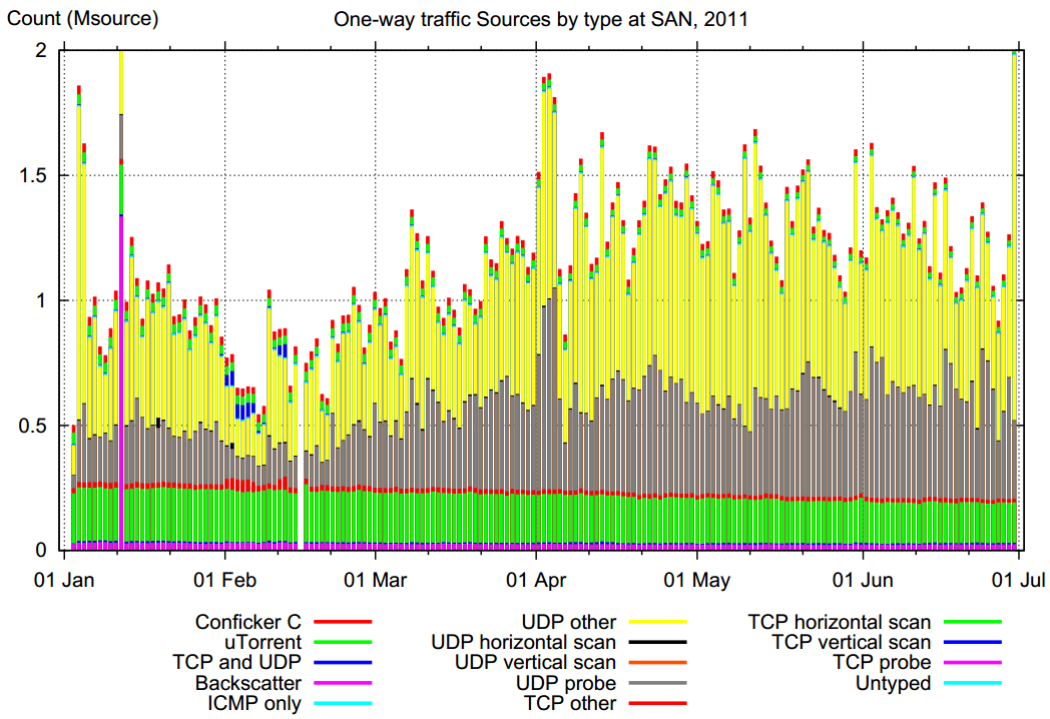


Figure 4.5: iatmon Example Output [3]

the UDP other traffic, other traffic types, such as uTorrent, seem to stay relatively stable over time. Even tools that provide a simple feature, like separating one-way traffic into clearly-defined subsets, can be of incredible help to network administrators who wish to better comprehend the network behaviors they are diagnosing. One could also imagine that distributed network management tools would benefit from sharing this type of aggregated data rather than having to execute the same operation at each node, which is easily supported by our publish/subscribe model of the network.

Chapter 5

Results

This chapter provides a detailed performance analysis that compares our system to current IP-based protocols that are used to share files among various nodes across the network. The experiments involve varying file sizes across different publisher-subscriber pairs. Since we intend to analyze the performance of general file transfer using our system, we decided to use two current protocols that are commonly used for file transfers, namely File Transfer Protocol (FTP) and Secure Copy (SCP).

Section 5.1 presents the results of using ftp, scp, and publish-subscribe as a file transfer service, including a comparsion analysis that provides insight into the efficiency of our system. Section 5.2 provides a detailed explanation of key improvements that the publish/-subscribe model provides, which may not be apparent in the results recorded in Section 5.1.5. Ideas for future improvements, based on the insights gained from the performance analysis, are presented in Section 5.3.

5.1 File Distribution Example

In this section, we describe the results of our performance analysis of various tools that provide a file transfer service, including our design based on the publish-subscribe model. We begin by describing the use of PlanetLab as a test environment in Section 5.1.1. Section 5.1.2 provides an overview of the File Transfer Protocol (FTP), which utilizes a client-server architecture to allow file sharing across network nodes. On the other hand, Section 5.1.3

describes the Secure Copy (SCP) program, which allows a node to transfer a file securely to another node in the network over an Secure Shell (SSH) connection. More importantly, Section 5.1.4 presents our publish/subscribe-based system that supports reliable file transfers between a publisher and a set of subscribers. A comparative analysis of these various file transfer services is provided in Section 5.1.5.

Our test plan for obtaining a performance analysis of these three programs involves varying the file size of the information item that will transferred from a source node to destination node, i.e. publisher to subscriber. The set of file sizes we chose to experiment with are: 1MB, 10MB, 50MB, 100MB, and 500MB. Varying the file size is straightforward to perform with each of the programs, since that simply requires varying the size of the input file and measuring the corresponding network statistics, e.g. throughput and time for the file transfer. We could have also experimented with varying the number of publishers and subscribers for a single information item, as well as changing the topology of the network, however these parameters are slightly more complex to configure with FTP and SCP since they assume a single publisher and a single subscriber in addition to implicitly using the already existing underlying network topology, i.e. the Internet. For future iterations of our design, we should consider evaluating the performance of our system compared to FTP and SCP using a wider variety of parameters and configurations.

5.1.1 PlanetLab test environment

PlanetLab represents a world-wide network of nodes, hosted both at public and private institutions, that is available for researchers who wish to perform experiments and development related to the field of computer networking. As suggested in Section 2.3.1, we chose to use PlanetLab for our experiments due to its role as a network substrate that is very representative of a public network, such as the Internet. After completing registration with PlanetLab, the system will automatically distribute your public key to a subset of the available nodes, i.e. nodes that are assigned to your slice of the network, thus allowing you execute a remote login session, such as SSH, in order to configure and deploy the network application.

In order to perform our experiments on PlanetLab, we needed to complete the following

steps. First, we chose ten nodes at random that seemed to have fairly reliable connections to the Internet and installed the required libraries for our application, e.g. the Blackadder library (the foundation of the PURSUIT architecture) and openssl. We also needed to define a configuration file for Click (the modular router that is used by Blackadder) that included the set of nodes we chose and specified the connections between them from an overlay network perspective. Once all of the necessary libraries and configurations are installed at each node, we can then begin running experiments where publishers advertise information items that are later transferred to subscribers who are interested in a particular file. However, in order to protect against users who attempt to use an unfair share of the PlanetLab nodes, most of the nodes have an established daily bandwidth limit, usually around 10GB. Therefore, we decided to configure a second network of nodes, which are Internet2 nodes that do not have a daily bandwidth limit, in order to perform the experiments that involve large files sizes (100MB and above).

5.1.2 FTP

File Transfer Protocol (FTP), as presented in RFC 959 [18], is a network protocol that enables file transfers from one host to another over a TCP-based network, such as the Internet. The protocol has four main objectives: 1) promote sharing of files, 2) encourage implicit use of remote computers, 3) shield a user from varying file systems, and 4) transfer data reliably and efficiently. It is designed as a client-server architecture, where the user initiates a connection with the server host and, provided that they have valid credentials, can interact with the file system at the server. The credentials primarily consist of a valid login name and password pair, which then authorize the user to execute common file system commands, such as ls (list contents of a directory) and cd (change the working directory), on the remote file storage device.

FTP is composed of two primary channels between the client and server, namely an FTP request-reply channel and a Data Connection channel as depicted in Figure 5.1. The FTP commands, which specify the parameters for the data conncetion (e.g. data port and transfer model) and the nature of file system operation (e.g. store, retrieve, append, and delete). After receiving an appropriate sequence of FTP commands, the server will initiate

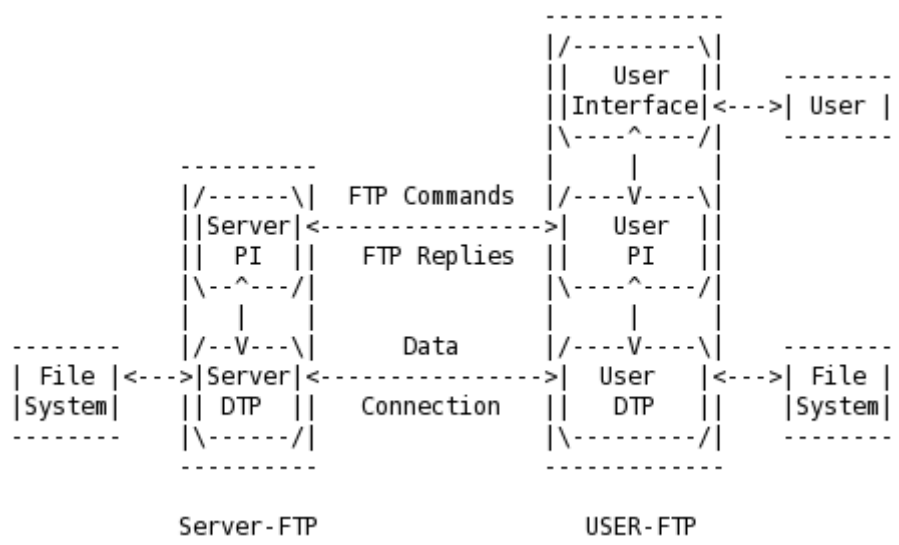


Figure 5.1: FTP Client-Server Model [18]

the data connection and data transfer in accordance to the specified parameters. The client can then receive the data by listening on the specified data port and process it in order to store the data on the local file system. The manual page for the ftp command, which includes the various parameters and options, is defined in [12].

For our test environment, we must take a few steps in order to create a configuration that will be comparable to the configuration of our publish/subscribe system. We must first establish a FTP server at each publisher node in order to allow subscriber nodes to initiate FTP connections with them and request files from the server's storage. In particular, we chose to use the vsftpd (Very Secure FTP daemon) program, described in [6], to instantiate an ftp server at the publisher node since it is compatible with the Fedora distribution in the PlanetLab nodes and supports both secure and unsecure FTP. Each subscriber node can then establish their own FTP session, using a program such as wget [14], with the publisher to locate and download files that they are interested in. We configured the vsftpd program such that a client can simply provide the login name and password pair used for the PlanetLab account, in order to receive authorization to access the file system on the remote server.

5.1.3 SCP

The Secure Copy (SCP) program provides a method of securely transferring files between two hosts over an unreliable network. It uses Secure Shell (SSH), defined in a number of RFCs including [27], for data transfer and uses the same mechanisms for authentication, which ensures the authenticity and confidentiality of the data. SSH is a protocol for secure remote login, in addition to other secure remote services, over an insecure network. It consists of three major components: Transport Layer protocol, User Authentication Protocol, and Connection protocol. Similar to FTP, SSH establishes multiple channels between the local host and the remote host in order to agree upon connection parameters and share data. SCP defines a specific application environment, where SSH is used to simply share files between the two nodes instead of more general remote services. The manual page for SCP, provided in [19], lists the options and parameters for performing a secure file transfer.

SCP provides a functionality that is very similar to FTP, however, since these programs were developed independently, we have decided to include SCP as an alternate benchmark to evaluate our publish/subscribe system. It involves generating public-private key pairs to encrypt the network connection and then using password authentication to log in. In this case, SCP also supports the use of the already existing PlanetLab account, i.e. login name and password pair, in order to authorize the transfer of files from one node to another. The program accepts two filenames, the first representing the source file and the second representing the destination file, each of which can either be local or remote files. Therefore, for our test environment, the subscriber will need to know a priori the files that are available at the publisher. However, we believe that this is stil an acceptable program to compare to, since our current version of the publish/subscribe system involves the assumption that the subscriber knows a priori the naming convention of the files it is interested in. Thus, an equivalent assumption can be used to validate the workflow of SCP.

5.1.4 Publish/Subscribe System

Section 3.5 explained, in great detail, our scheme for providing a reliable file transfer service through our publish/subscribe model of the network. Here, we will review some of the main

steps that allow us to support the sharing of information between a publisher and a set of subscribers.

Recall that our file distribution scheme is composed of two main stages: one where the publisher establishes the algorithmically generated scopes after publishing a particular information item, the other where the subscriber calculates a random ID that is used as an identifier of the unique retransmission channel between the publisher and the subscriber. After publishing the information item, the publisher will also publish two scopes, the first which is used by the subscribers to publish their retransmission requests, while the second is used by the publisher to reply to individual retransmission requests that may be arriving from different subscribers. Each subscriber, upon receiving a fragment corresponding to an information item they have subscribed to, will generate a random ID and publish an information item under the first algorithmic scope for retransmission requests, while subscribing to an information item under the second algorithmic scope for replies from the publisher.

As long as both the publisher and subscriber nodes can agree upon a hierarchy and naming convention for files they that intend to share, our system will automatically produce the appropriate scope hierarchy that ensures a unique identifier for each file. Since our design mimicks the hierarchy of the underlying file system, the publisher and subscriber only need to provide the full path of the file they wish to publish or subscribe to. Contrary to the assumptions of FTP and SCP, where the file is assumed to already exist, our publish/subscribe system allows subscribers to indicate their interest in a file regardless of whether the information item has been published or not. The asynchronous nature of our publish/subscribe system is difficult to quantify and include in our performance analysis, thus we will further explain its benefits in Section 5.3.

5.1.5 File Distribution Comparison

In this section, we present some preliminary results that showcase the performance of our publish/subscribe system versus traditional file transfer services, in particular FTP and SCP. We will begin by describing the commands we ran in order to perform the file transfers themselves. We will then review the results of our experiments and provide some insight into the implications on the direction of our future work.

File Size	Number of src/dst pairs	Number of trials/pair	Total Number of Trials
1MB	15	25	375
10MB	15	16	240
50MB	9	8	72
100MB	9	4	36
500MB	6	2	12

Table 5.1: Number of Experiments Executed

Table 5.1 provides a summary of the number of experiments we performed for each file size, in terms of the number of different publisher-subscriber pairs we tested and the number of trials for each pair. Recall that our test network was composed of one RV/TM node and 9 other regular nodes, which played the roles of publisher and subscriber during different trials. We ran the same number of experiments for the three designs we were testing, namely FTP, SCP, and our publish/subscribe system, in order to reduce the possible bias of our results.

The FTP command we used for the trials is similar to the following example:

```
wget -x -nH -P ~ ftp://host/full\_file\_path
```

The options we included are: '-x' (creates necessary directories at the destination), '-nH' (disables generation of host-prefixed directories), '-P ' (sets the prefix at the destination to the home directory). The host field holds the IP address of the source node while the full_file_path specifies the file path of the information item of interest.

In a similar manner, the SCP command we specified for the experiments is analogous to the following example:

```
scp -c arcfour mit_psirp@host:full_file_path file_parent_directory
```

Here we only needed to include the '-c' option, which allows us to specify the type of cypher algorithm we wish to use to secure the data transfer. In particular, we chose to use the arcfour variant, which is shown to be the fastest one, albeit the least secure one as well. We also need to specify the file's parent directory in order to replicate the behavior of our publish/subscribe system which uses the user-specified file path to determine where to save the transferred file.

Since our system currently does not support optional command-line arguments, the approach to executing a trial starts with a very simple command:

```
manual_file_publisher
```

or

```
manual_file_subscriber
```

By running the executable binary in the shell, a command prompt, which is generated by the application, will ask the user to specify the name of the file they wish to either publish or subscribe to, i.e. provide the full file path. At the publisher node, the application will simply acknowledge the input if the file is found on the local disk and then initiate a new prompt for another possible publication. On the other hand, after the subscriber receives the first fragment of a file, the application will display information in three parts: 1) shows a summary of details about the Rendezvous match, including the identifiers for the bi-directional channel, 2) displays a progress meter indicating the amount of the file transfer that has been completed so far, 3) presents the time it took for the file transfer to complete after all of the fragments are received and written to disk.

Figure 5.2 presents a comparison of the three different file transfer services in the form of a line graph. The x-axis contains the various file sizes we used for our experiments on a logarithmic scale, while the y-axis records the total time it takes for the transfer of a given file. Since we ran the experiments multiple times across different pairs of nodes, we only show the average latency value for each file size category. The plots for each of the file transfer services are distinguished through a different line style, as specified in the graph's legend.

At first glance, one may notice an exponential behavior between the file size and the download time, however this is simply an artifact of using a logarithmic scale on the x axis. If the graph used a linear scale for the x-axis, the plots would have a more linear shape, but we believe this graph provides more clarity for identifying the data points of interest. For small file sizes, the download times between the different systems are nearly equivalent, with the SCP command taking slightly more time due to its encryption feature. With a

file size of 50 MB, we begin to note a larger difference between the FTP command and the other two systems (SCP and Publish/Subscribe), which is expected since FTP avoids the overhead of encapsulation and encryption as opposed to the Publish/Subscribe system and the SCP service respectively.

We begin observing the decreased performance of our publish/subscribe system, relative to FTP, at the larger file sizes (100MB and above). There are a few reasons that can explain this trend, which includes the overhead of encapsulation, inefficient payload sizes, and a lack of congestion control. First, since our system works as an overlay network on top of IP, we must incur the performance cost of encapsulation, meaning that our packets not only have to include our own headers and payload but also headers from other protocols such as the IP packet header. Similarly, the constraint of encapsulation emphasizes the need to maximize the usefulness of the space we have within an IP packet. In our current version, we simply assume a conservatively low payload size (1,024 bytes) in order to include the forwarding identifiers within the encapsulated packet. Essentially this means that the throughput of our system is lower since we need to transmit more packets, on average, to transfer the same amount of data.

Most of all, the declining performance of the publish/subscribe service at the higher file sizes suggests that the system is experiencing dropped packets more frequently, most likely as a result of buffer overflows at intermediate nodes or even at the end nodes themselves. Our current approach towards mitigating this problem is to artificially slow down the publisher by inserting a small delay after the publication of each fragment in addition to inreasing the timeout interval at the subscriber. However, future iterations of our design should include an efficient congestion control mechanism that allows us to match the publisher's transmission rate to the capacity of the path similar to FTP and SCP. Section 5.3 will provide further details about improvements we can make to the design of the system in order to avoid these inefficiences as well as others that might be uncovered in the future.

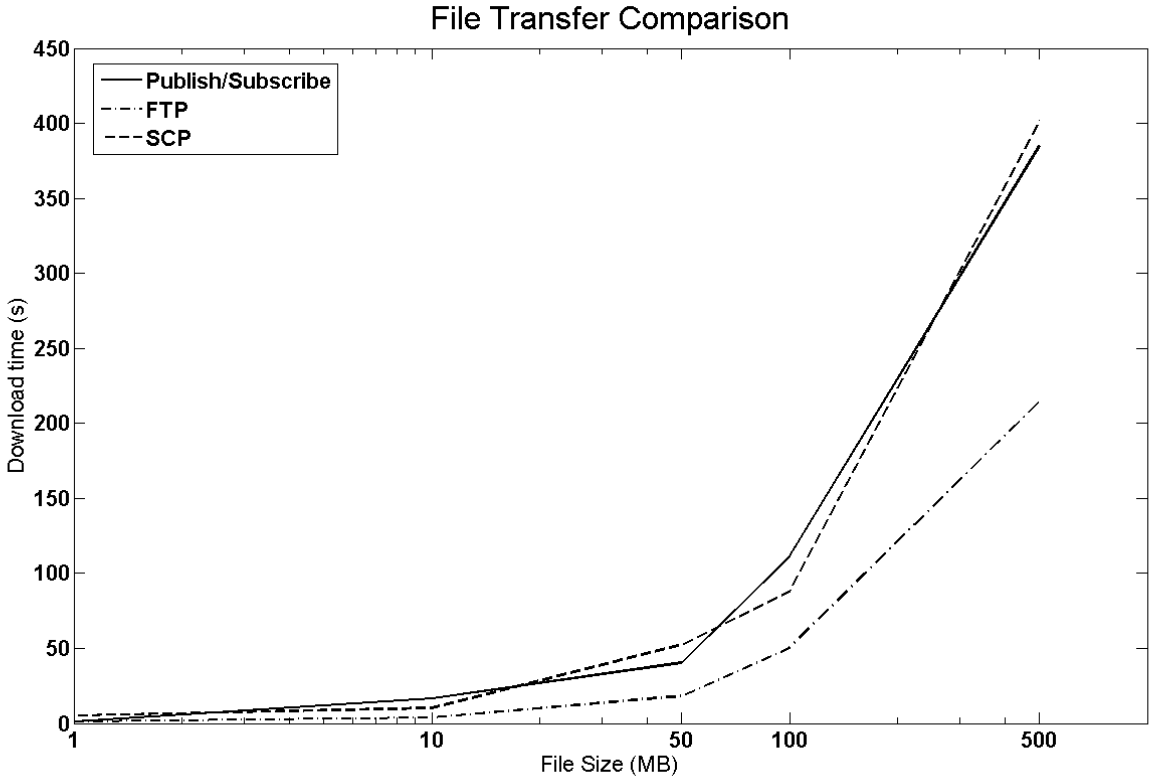


Figure 5.2: File Transfer Comparison Graph

5.2 Hidden Benefits

We believe that the publish/subscribe paradigm provides additional benefits that may not appear at first glance in our performance analysis. The three main benefits that we will discuss in this section are: the separation of data and control plane, node cooperation to achieve higher goals, and an asynchronous, on-demand service that avoids issues such as network implosion.

As suggested by Clark et al. [4], most discussions of network architecture recognize two architectural divisions, or planes: a data plane, over which content is forwarded, and a control/management plane, which is used to directly measure and repair the data plane. Existing control systems are designed to cut across the layering of the data plane in order to give visibility and access to all the aspects of the network, however it is hardly scalable in a large network. Instead, the Knowledge Plane is designed to involve the edge nodes themselves, given that they already hold significant intelligence, in addition to providing a compositional, cognitive, and unified framework that accounts for the challenges in managing an inter-domain network. An effective way to disseminate information that might be incomplete but gives an edge a broad perspective of the network is through a publish/subscribe model that focuses on the content itself rather than the data path.

Similar to the idea of agent cooperation in Li and Lee's theses, nodes who join the Information Plane are also expected to collaborate in order to deliver information across the network. Our system allows multiple nodes to become publishers of the same information item, which enables the Topology Manager to create the most effective delivery graph given all of the possible paths between the set of publishers and the set of subscribers. Additionally, nodes are designed to automatically cache content that they receive from the network, which can be used to fulfill later subscription requests for the same information item. Thus, we can see that the publish/subscribe model provides various avenues through which nodes collaborate, mechanisms that are not readily accessible in traditional IP-based approaches to data delivery.

The asynchronous nature of the publish/subscribe model also brings a few advantages that are particularly useful for network management, and content distribution in general.

First, we no longer have to hold the assumption that the source and destination nodes must agree upon a time to begin the file transfers. On the one hand, our system supports the publication of information items that do not yet have any subscribers, while on the other hand, our system also supports the subscription of information items that are not currently published. We can think of this approach as an on-demand service that initiates file transfers only when an explicit match between at least a single publisher and a single subscriber is made. It not only avoids generating unnecessary probing of the network, which increases congestion, but also avoids issues such as the implosion problem, when a group of nodes trying to get the same information overload the source of that information. Instead, the Rendezvous node and Topology Manager can work together to create a suitable delivery graph that balances the workload over various reliable paths. It is the focus on the content itself, rather than the focus on the path, which allows us maximize the network efficiency in distributing knowledge.

Furthermore, the publish/subscribe model naturally supports a "multicast-like" delivery service that works in two ways: a) subscription requests can be merged when they are a part of the same information item, b) responses can be merged when paths overlap. If we were to embed an SQL-based functionality into the system, as proposed in Section 5.3, the system could accept a set of tags provided by the subscription request and process them in order to identify the matching information items. Similarly, responses to subscription requests can be merged by finding a suitable delivery graph that reaches multiple subscribers simultaneously, a task that the current version of the Topology Manager already performs.

Lastly, PURSUIT supports not only policies on the individual information items themselves but also on a scope, which is simply a construct that groups similar items together. Whenever the Topology Manager computes the path for a given delivery graph, it also includes the policy constraints on the path based on the policy for the item itself or its parent scope. This is contrary to the approach in the Internet, where policies are applied in an ad-hoc manner based on location.

5.3 Future Improvements

After running many trials and observing the behaviors of the publish/subscribe system, we have captured a few key ideas that can serve as improvements for later versions of the publish/subscribe system. The list of possible enhancements to our system includes: an improved timeout mechanism, batching up multiple writes to a file, and efficient balancing of RIds and payload when generating new packets to be sent over the network. We will review some of the major ones, here in this section, that would not only improve the performance of our system but also provide additional functionalities that may be useful for an information-centric network, particularly for the goal of automated network management.

First, we must note that the current design of our system does not perform any congestion control to account for either slow end-nodes or a weak path between publisher and subscriber. This problem is not apparent when transferring small files, but is exacerbated when sending large files. In our design, the publisher does not take into consideration the capacity or round-trip time of the path to the subscriber due to the system's asynchronous nature. After receiving a notification to start publishing, the publisher simply queues up all of the fragments of the file and sends them to the network interface in quick succession, which may result in a large number of dropped packets due to buffer overflows in the network. Packet drops may also result from a slow subscriber, who cannot process the received packets fast enough, since file writes are generally slower than file reads. We have, however, mitigated the potential bottleneck at the subscriber by utilizing a memory-mapped file that is saved to the local disk only at the completion of the file transfer. But, in order to avoid and respond to network congestion, the publisher and subscriber must work together to estimate the capacity of the path and relay congestion information in a manner similar to TCP. One may consider using the already existing retransmission channels to share beliefs about the current state of the network.

We can also consider improving the throughput of our system through a careful balance between the forwarding identifiers and the data payload that must fit within a single IP packet. In our experiments, we already incur a performance cost in needing to include the IP header, since PlanetLab runs on top of the IP layer. Thus, it is important to maximize the

utility of leftover space in the IP packet, which contains information from Blackadder such as the dissemination strategy, forwarding identifier size, and possibly multiple forwarding identifiers. Our current design assumes a fixed data payload size of 1,024 bytes to avoid possible IP fragmentation, given that we need to reserve enough space for the other meta-data. However, if the nodes could precisely determine how much space is needed to contain the meta-data, then we could consider using variable-sized payloads. Since the subscriber also needs to know the payload size in order to allocate the appropriate amount of memory, we can include a designated fragment, perhaps the first fragment, that contains the payload size of the file fragments as well as other characteristics of the file.

The publisher can also include some functionality that allows it to publish an entire directory of the local file system, rather than having to publish the files one-by-one. For simplicity, our current design allows the user to specify the full file path of the information item they are interested in either publishing or subscribing to. In a future version of our system, we should consider supporting file paths that point to directories, which implies publishing all of the files directly underneath the directory as well as recursively publishing any child directories. This should not require significant changes to the application since it automatically generates the scope hierarchy based on the full file path. On a similar note, the current version of the subscriber makes the assumption that the file subscriptions occur sequentially and only one-at-a-time, i.e. we must wait for the current file transfer to complete in order to begin a new one. Unfortunately, supporting simultaneous subscriptions would require additional functionalities that would increase the complexity of the subscriber, thus we leave this goal for a future design.

Another researcher in our group is also investigating various caching schemes that may be useful to incorporate into the PURSUIT architecture, such as a time-based or content-based cache [1]. The current version of the PURSUIT architecture, which can be split into three main functions: handling communications (Rendezvous), the graph structure of the network (Topology Management), and data storage (Caching), uses a caching function that remains underdeveloped. Using a more sophisticated caching system would increase the availability of data, improve network performance, and provide reliability and resistance to failures. From our data delivery perspective, we would be greatly interested in seeing the

benefits on network performance, in terms of being able to handle subscriptions faster and in a more reliable manner in case of network failures.

Finally, we should consider embedding more intelligence into the network in order to support an SQL-like interface where the system can automatically identify files that match a set of characteristics that the subscriber node is interested in. Our current design makes the assumption that both publishers and subscribers will know a priori the naming convention for files that will be shared across the network. Ideally, we would want to allow subscribers to simply provide a set of characteristics, or tags, that represent information it wishes to receive from the network. The RV nodes can then collaborate to identify the files that correspond to that particular set of characteristics and create matches between the publishers and subscribers.

Chapter 6

Conclusion

The immediate contributions of the work presented in this thesis report are two-fold.

First, we designed and demonstrated the feasibility of providing a publish/subscribe-based file exchange system that, even without congestion management, is close in performance to traditional file transfer protocols. For smaller file sizes, our publish/subscribe system had nearly identical performance compared to both FTP and SCP. For larger file sizes, the system incurred a larger performance loss for a few reasons such as lack of congestion control and inefficient payload sizes. However, we believe that the gap can be closed through futher improvements to the publish/subscribe system, such that the performance cost is no longer an issue.

Second, we highlighted a number of ways in which there are significant hidden benefits to our approach that provide a more suitable file transfer service for an Information Plane that is complementary to the Knowledge Plane. The benefits include: the separation of data and control plane, node cooperation to achieve higher goals, and an asynchronous, on-demand service that avoids issues such as network implosion. Additionally, the publish/subscribe model provides a clear and systematic way of organizing and disseminating information across socially-constructed boundaries in the network. It naturally leads to concepts, such as knowledge agent cooperation and a scalable network management system, that increase the value of adding intelligence to the network.

6.1 Future Work

As mentioned in Chapter 2, our architectural approach to network management leads to a few key areas of study that should be further explored.

First, we should find an effective way to manage and organize the distribution of information given the many organizational constraints, particular social and economic policy boundaries that are not considered in today's network architectures. This is analogous to the routing scheme for inter-domain communications, where the policies of an intra-domain network are exposed, only up to a certain level of detail, at the edge nodes that communicate with other domains or autonomous systems. We believe that the use of scopes in the publish/subscribe model simplifies the problem of embedding the concept of boundaries into the system. By grouping related data together in a scope and allowing access to the scope only through the Rendezvous function, the system can create a suitable delivery graph whenever a match between publishers and subscribers is established, provided that the match follows the policy constraints for that information space.

We should also model and evaluate the impact of our architectural proposal in terms of performance and efficiency, considering that network management is secondary to the transmission of user information. The performance analysis of the publish/subscribe system in this thesis report is only a small part of the evaluation process that we must undertake in order to more fully evaluate the usefulness of our approach. For example, performance analyses of other aspects of the architecture, such as features in the Knowledge Plane, would further elucidate the achievable performance gains of our system.

An important topic to explore is how to systematically manage information given the constraints on storage, caching, and potential lifetime of information items. This is very similar to the first topic area of information distribution under organizational constraints, however it focuses on the characteristics of the content itself. Some data might be designed to be short-lived, e.g. a live webcast, while other information items are expected to be available over a longer period of time, e.g. a monthly/yearly activity log, both of which should be supported by the system. However, there are many other complex scenarios, such as replicated state machines in data centers, that require further research to find a suitable

support structure that we can embed into the network.

Finally, we will need to discover how to more effecitvely anticipate tussles, i.e. differences of opinions and concerns, when using the network and be able to accommodate them in the network architecture. As opposed to traditional network architectures where tussles are too complex to identify and resolve systematically, the publish/subscribe model clearly exposes tussles whenever a transfer of content is requested from one node in the network to another. In other words, a subscription request can only be fulfilled if the Rendezvous node determines that the subscriber has authorized access according to the policies defined in the item's meta-data or the meta-data of the parent scope.

In summary, a scope publish/subscribe information model can provide a rich set of enhanced capabilities that, in turn, may provide more generalized functionality, increased flexibility, and more reasoned approaches to the support of an Information Plane underpinning the Knowledge Plane.

Bibliography

[1] K. Beckler, "Improved caching strategies for publish/subscribe internet networking," 2011.

[2] R. E. Beverly, "Statistical Learning in Network Architecture," Cambridge, MA, 2008. [Online]. Available: http://hdl.handle.net/1721.1/44210

[3] N. Brownlee, "One-way Traffic Monitoring with iatmon," in *Passive and Active Network Measurement Workshop (PAM)*. Vienna, Austria: PAM 2012, Mar 2012.

[4] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A Knowledge Plane for the Internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '03. New York, NY, USA: ACM, 2003, pp. 3–10.

[5] G. Combs, "Wireshark." [Online]. Available: http://www.wireshark.org

[6] C. Evans, "vsftpd - Secure, fast FTP server for UNIX-like systems." [Online]. Available: https://security.appspot.com/vsftpd.html

[7] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: seeing the forest for the trees," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 1:1–1:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070563

[8] V. Jacobson, C. Leres, and S. McCanne, "TCPDUMP/LIBPCAP." [Online]. Available: http://www.tcpdump.org

[9] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 1–12.

[10] G. J. Lee, "Probabilistic Internet Fault Diagnosis," MIT, Advanced Network Archictecture, Cambridge, MA, Tech. Rep., 06 2007.

[11] J. Li, "Agent Organization in the Knowledge Plane," MIT, Advanced Network Archictecture, Cambridge, MA, Tech. Rep., 06 2008.

[12] Linux NetKit, "Internet File Transfer Program - Linux man page," 1999. [Online]. Available: http://linux.die.net/man/1/ftp

[13] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An Information Plane for Distributed Services," in *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 26–26.

[14] H. Niksic.

[15] S. Ostermann, "TCP Trace." [Online]. Available: http://www.tcptrace.org

[16] K. Park and V. S. Pai, "CoMon: A Mostly-Scalable Monitoring System for PlanetLab," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 65–74, Jan. 2006.

[17] PlanetLab, "About PlanetLab," 2012, website. [Online]. Available: http://www.planetlab.org

[18] J. Postel and J. Reynolds, "File Transfer Protocol (FTP)," 1985. [Online]. Available: http://tools.ietf.org/html/rfc959

[19] T. Rinne and T. Ylonen, "Secure Copy - Linux man page," 2013. [Online]. Available: http://linux.die.net/man/1/ftp

[20] K. R. Sollins, "FIND: KPBase: a Common Infrastructure for Network Management," December 2008, unpublished.

[21] K. R. Sollins, "An Architecture for Network Management," in *Proceedings of the 2009 workshop on Re-architecting the internet*, ser. ReArch '09. New York, NY, USA: ACM, 2009, pp. 67–72.

[22] The Cooperative Association for Internet Data Analysis, "CAIDA." [Online]. Available: http://www.caida.org

[23] The Cooperative Association for Internet Data Analysis, "CoralReef Software Suite." [Online]. Available: http://www.caida.org/tools/measurement/coralreef/

[24] D. Trossen, "Architecture Definition, Components, Descriptions, and Requirements," 2009. [Online]. Available: http://www.psirp.org

[25] D. Trossen, "Conceptual Architecture: Principles, Patterns and sub-Components Descriptions," 2011. [Online]. Available: http://www.psirp.org

[26] D. Trossen, M. Sarela, and K. Sollins, "Arguments for an Information-Centric Internetworking Architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 26–33, Apr. 2010.

[27] T. Ylonen, "The Secure Shell (SSH) Protocol Architecture." [Online]. Available: http://tools.ietf.org/html/rfc4251