November 2015

Design and Implementation of an Economy Plane for the Internet

Xinming Chen
University of Massachusetts - Amherst

DESIGN AND IMPLEMENTATION OF AN ECONOMY PLANE FOR THE INTERNET

A Dissertation Presented

by

XINMING CHEN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2015

Electrical and Computer Engineering

DESIGN AND IMPLEMENTATION OF AN ECONOMY PLANE FOR THE INTERNET

A Dissertation Presented

by

XINMING CHEN

Approved as to style and content by:

Tilman Wolf, Chair

Michael Zink, Member

David Irwin, Member

Jim Griffioen, Member

Christopher V. Hollot, Department Head
Electrical and Computer Engineering

DEDICATION

To Shuo and my parents.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Professor Tilman Wolf, for his guidance and advice in the past four years. During these years, I learned much from him, including rigorous scholarship and hard working. He taught me how to tackle new research problems, pitch ideas, and give presentations. I believe these skills will be valuable not only for my research but also for my future life.

I would like to thank my dissertation committee members, Professor Michael Zink, Professor David Irwin, and Professor Jim Griffioen. They not only provided guidance on my research projects, but also provided valuable feedback on the dissertation writing.

I would like to thank Professor Michela Becchi and her student Brandon Jones. We have worked closely for a year on a regular expression matching project. Professor Becchi's erudite and expertise on regular expressions have filled me with admiration. It is a great honor to have published several papers with them.

I would also like to thank my labmates and colleagues. Hao Cai's math skill has really impressed me. I feel fortunate to have worked with him on the credential and path finding projects. I have also worked closely with Abhishek Dwaraki. His rich engineering experience assured the successful implementation of the ChoiceNet project. I also want to thank Thiago Teixeira, Luis Andrs Marentes Cubillos and Cong Wang for all those inspiring discussions we have had on our research problems. They brought fun and motivation to my research.

My parents have supported me unconditionally throughout my life. They have being very supportive of my study abroad even if they missed me very much. Special

thanks to my wife, Dr. Shuo Deng, who not only supports my everyday life, but also provides a lot assistance and excitation in my academic life. My Ph.D. experience would not have been so wonderful without her company.

ABSTRACT

DESIGN AND IMPLEMENTATION OF AN ECONOMY PLANE FOR THE INTERNET

SEPTEMBER 2015

XINMING CHEN

B.E., TSINGHUA UNIVERSITY

M.Sc., TSINGHUA UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Tilman Wolf

The Internet has been very successful in supporting many network applications. As the diversity of uses for the Internet has increased, many protocols and services have been developed by the industry and the research community. However, many of them failed to get deployed in the Internet. One challenge of deploying these novel ideas in operational network is that the network providers need to be involved in the process.

Many novel network protocols and services, like multicast and end-to-end QoS, need the support from network providers. However, since network providers are typically driven by business reasons, if they can not get economic profit from supporting new protocols and services, they will not deploy them. Therefore, we conclude that the lack of explicit economic relationship in the current Internet hinders the innovation of itself, and it is critical that a network architecture intrinsically considers economic relationships.

ChoiceNet is an NSF funded Future Internet Architecture (FIA) project that aims to address these challenges. ChoiceNet proposes an "economy plane" of the Internet to explicitly represent economic relationship within the architecture. This economy plane enables entities in the network to dynamically set up fine-grained, short-term economic contracts for network services. A marketplace can be established for advertising and selling services. The services can be simple path services (pathlets) between end-points, or more complex processing and storage services (e.g., transcoding and caching).

ChoiceNet is a comprehensive project, and its architecture is designed by researchers from several institutes. This work will not cover every aspect of it. Instead, this work will focus on five aspects of ChoiceNet: 1) service definition and protocol design, 2) marketplace design, 3) use plane design, 4) path finding algorithm design, and 5) access control for services. Service definition aims at a unified and extensible description of services, and the method to compose them. Marketplace design discusses the protocols used to advertise and request services. The use plane design describes how network providers and users will access the Marketplace while preserving the existing infrastructure and applications, it also discusses how to progressively deploy ChoiceNet in the current Internet. The path finding algorithm design proposes ParetoBFS, an algorithm finding all the Pareto-optimal paths in a multi-criteria network. The access control discusses how to prevent unauthorized usage of the services, we present OrthCredential, an algorithm for high-performance access control in ChoiceNet. To prove the feasibility of such an economy plane, this work presents a Software Defined Networking (SDN) based implementation of ChoiceNet. The implementation has been deployed and tested on GENI, a global test bed for network architectures.

By designing and implementing ChoiceNet, this work tries to offer a network architecture that users can select from several different network services rather than

being limited to a single choice. By enabling greater choice, ChoiceNet can promote competition among providers for price and quality. This competition will lead to lower prices and higher quality services, which are beneficial for consumers and eventually help bring sustained innovation into the Internet.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

Since its birth in the 1960s[1], the Internet has evolved in many aspects. As the diversity of uses for the Internet is increasing, demands for additional protocols and services are emerging. Recently, there has been much interest in the networking community to explore new network architectures of the future Internet [57]. Novel protocols and applications are being proposed, and the Internet provides a foundation for them by providing data communication functionality between end-systems.

However, while the technology for new network requirements is being developed, there are challenges in deploying innovation in the Internet. A key problem in deploying innovative features in the network core is that many protocols and services need support from providers throughout the network. In the early stage of the Internet, innovations were made in the core by a small group of researchers and operators. With todays dramatically larger community, consensus is more difficult to achieve and innovation is for the most part limited to the edge. If a novel service needs end-to-end support to deploy, either it fails to deploy, or it has to compromise and use overlay network to work above the TCP/IP layer. The example for the former situation is end-to-end QoS, which is still hard to achieve today. The example for the latter situation includes end-to-end secure channel (e.g. VPN) and IPv6's tunnel technology.

One reason to this obstacle, we believe, is the inadequacy of supporting an important relationship in the Internet architecture – the economy relationship. This work

[1]If we consider the APARNET to be the origin of Internet.

aims at introducing economics into the current Internet, and designing a platform where the customers and providers can build explicit economic relationship. So that the Internet can evolve better.

1.1 Introducing Economics to the Internet Architecture

There is undoubtedly some level of economy relationship in the current Internet. The economy relationship is mostly established by the paper-based service contract or peering agreements, which provide very limited choice to the customers, and are unable to change within short period. On the other hand, the money flow in the Internet is mostly flowing from the edge to the core, that is, money are paid to the local ISP, then the backbone carriers get their share from the edge. When new network features are designed, there are often misalignment between protocol design and economic motivation – since network operators are justifiably driven by business goals, they need to have clear incentives to support new network features. Two examples of such misalignment is multicast and end-to-end QoS:

Multicast Multicast can reduce the network traffic at the backbone carrier by sending only one copy of packet to multiple receivers. However, backbone carriers charge the peering ISPs by the amount of traffic, so they are unwilling to invest in equipments supporting multicast, which will in turn reduce their revenue [25].

End-to-end QoS End-to-end QoS requires the entire path to reserve the bandwidth and priority of a flow. In an Internet scale, it is nearly impossible for an ISP to set up service agreement to other arbitrary ISPs, not to mention such agreement need to be per-flow and short term. Therefore, end-to-end QoS are difficult to achieve at the Internet scale.

Therefore, there is a need for the *network architecture* to associate innovation with economic motivation. In the work by Clark et al. [23], they emphasizes the

importance of tying real-world tussles to the network architecture. To address these challenges and expose economic tussles within the architecture, researchers from the University of Massachusetts Amherst, the University of Kentucky, North Carolina State University, and the University of North Carolina propose an "economy plane" called ChoiceNet. It enables entities (e.g., users or their applications, providers, etc.) to dynamically set up fine-grained, short-term economic contracts for network services. These network services are offered and sold through marketplaces and can range from simple connectivity (à la pathlets [33]) to complex processing and storage services (e.g., caching for NDN [40]).

1.2 ChoiceNet Principles

The principles of the ChoiceNet has been described in the paper by Wolf et.al. [73]. This section briefly introduces the three key principles of ChoiceNet:

Encourage alternatives. The architecture of network must allow alternative services of the same type. The alternatives can be both technical (e.g. different transmission protocol) and economical (e.g. the same video streaming from different providers at different price). The service can range from infrastructure level such as alternative paths, to protocol level such as IP network vs. NDN. With alternatives, customers can make choices based on to their own requirements, thus encourage the competition between providers.

Know What Happened. After the customer has chosen a service, the architecture must provide the customer a mechanism to evaluate the service. Customers may find out the service performance metrics through a monitoring method, or find out the reputation of a service from a ranking system. Only with the evaluation to the service ("introspection"), the customer can understand whether the choice is good or not.

Vote With Your Wallet. After the customer knows how the selected service performs, the customer can choose to stick to the same service next time, or to switch to a new service. Such *vote with the wallet* will force the providers to offer services that match customers' needs. This iteration may happen at a very fine grain (down to flow level) and short period (as short as seconds).

Sticking to the 3 principles, the architecture of ChoiceNet is described in Chapter 2. With this "cycle of innovation," ChoiceNet is expected to prompt deployment of innovative technologies throughout the Internet.

1.3 Challenges and Requirements

ChoiceNet is a comprehensive project, and this dissertation only covers some of its components. This dissertation focuses on two aspect of the ChoiceNet: protocol design and prototype implementation. To be specific, the following topics will be covered: 1) service definition and protocol design, 2) marketplace design, 3) use plane design, 4) path finding algorithm design, and 5) access control for services.

These topics cover most part of the ChoiceNet. Some of the key theoretical and practical research challenges this work tries to address are:

1.3.1 Service Definition

- What is the proper abstraction for services, and how to propose a general enough service description to include all the possible services?
- How can services be composed to form complex/tailored services?

1.3.2 Marketplace Design

- How should the interface between the marketplace and the provider/customer be designed such that the service listing and requesting is easy, robust and extensible?

- How to use parallel design on marketplace to handle more request?

1.3.3 Use Plane Design

- What is the pricing strategy for services?
- How to support path and protocol alternative while maintaining compatibility with existing end systems?
- How to evolve the traditional, distributed network protocols in the new centralized network?
- How to change the end system to communicate with marketplace without affecting the upper layer applications?
- How to incrementally deploy ChoiceNet in the current Internet?

1.3.4 Path Finding Algorithm Design

- How to design a path finding algorithm that can find Pareto-optimal paths in multi-criteria networks?
- How to design it fast enough to find all the Pareto-optimal paths within 1 second on a typical sized network?
- How to find a subset of the Pareto-optimal paths in shorter time when the full Pareto-optimal set is not necessary?

1.3.5 Access Control for Services

- How to design an authentication scheme to forbid unauthorized use of services?
- How to make such scheme faster and use less memory?
- How to resist the DoS attack to the authentication scheme?

1.4 Organization and Contributions

This dissertation focuses on five of the research challenges discussed in Section 1.3: service definition, marketplace design, use plane design, path finding algorithm and access control. The rest of this dissertation is organized as follows, with the major contributions summarized in each.

Chapter 2 introduces the design of the ChoiceNet architecture which were published before. It introduces the key components of ChoiceNet, and the interactions between the entities.

Chapter 3 presents a design of a service access protocol based on the establishment of economic relationships between entities. We present a service description which allows straightforward composition. A protocol of service is also proposed, which defines the format of service listing, request and response.

Chapter 4 presents the detail of marketplace implementation. The APIs for customers and providers are defined, the path finding algorithm is introduced, and the parallel and distributed architecture of marketplace is proposed.

Chapter 5 discusses the use plane problems. Since the introduction of centralized control plane, some traditional network protocols need to be modified. These include neighborhood discovery, intra-domain routing and inter-domain routing. This chapter presents updated algorithms to solve these problems. The ChoiceNet App is also introduced in this Chapter, which is used on the end-system, interacting with the marketplace while keeping compatibility with existing applications. This chapter then proposes two hybrid deployment methods, to incrementally deploy ChoiceNet in the current Internet.

Chapter 6 presents *ParetoBFS*, a new multi-criteria path finding algorithm for ChoiceNet. This algorithm is a variant of the breadth-first search (BFS) algorithm and uses Pareto constraints to prune the traversal tree. Comparison with two existing algorithms shows ParetoBFS is tens to hundreds times faster and find more paths on

typical sized networks. This chapter also shows a sampling heuristic to decreases the running time by only finding a subset of Pareto-optimal solutions.

Chapter 7 focuses on the service access control problem. An algorithm named *OrthCredential* is proposed to prevent unauthorized access of service. OrthCredential uses Hadamard matrices to verify packets along the path. It provides a fast and memory efficient method for access control. It is also resistant to DoS attacks.

Chapter 8 describes the deployment on GENI – a test bed for network architectures. Results from prototype deployment are provided.

Chapter 9 concludes the previous chapters.

CHAPTER 2

CHOICENET ARCHITECTURE

As described in the introduction, the principle idea of ChoiceNet is to enable market-based competition among providers of network services, which improves quality of offerings and reduces cost to customers. To enable a competitive market, ChoiceNet introduces an explicit representation of economic relationships between entities in the network.

This chapter describes ChoiceNet architecture and operations within it. The architecture is designed by the entire ChoiceNet team, as described in [59, 72, 73]. It is summarized here to provide the background needed to understand the specific topics this work addresses in the following chapters.

2.1 Planes and Entities in ChoiceNet

Figure 2.1 shows the architecture of ChoiceNet. It is composed of the economy plane and the use plane. The economy plane is where the transactions happen between customers and providers. The use plane is where the services are realized based on the contracts settled in the economy plane.

In the economy plane, the main entities are *customers* and *providers*. Customers contact with providers for the access of services. One entity can act as both a customer and a provider at the same time. For example, a customer can resell the purchased service to others (either directly or as a bundle by adding other functions to it). The expected output of the economy plane is the establishment of the contract between the two entities.

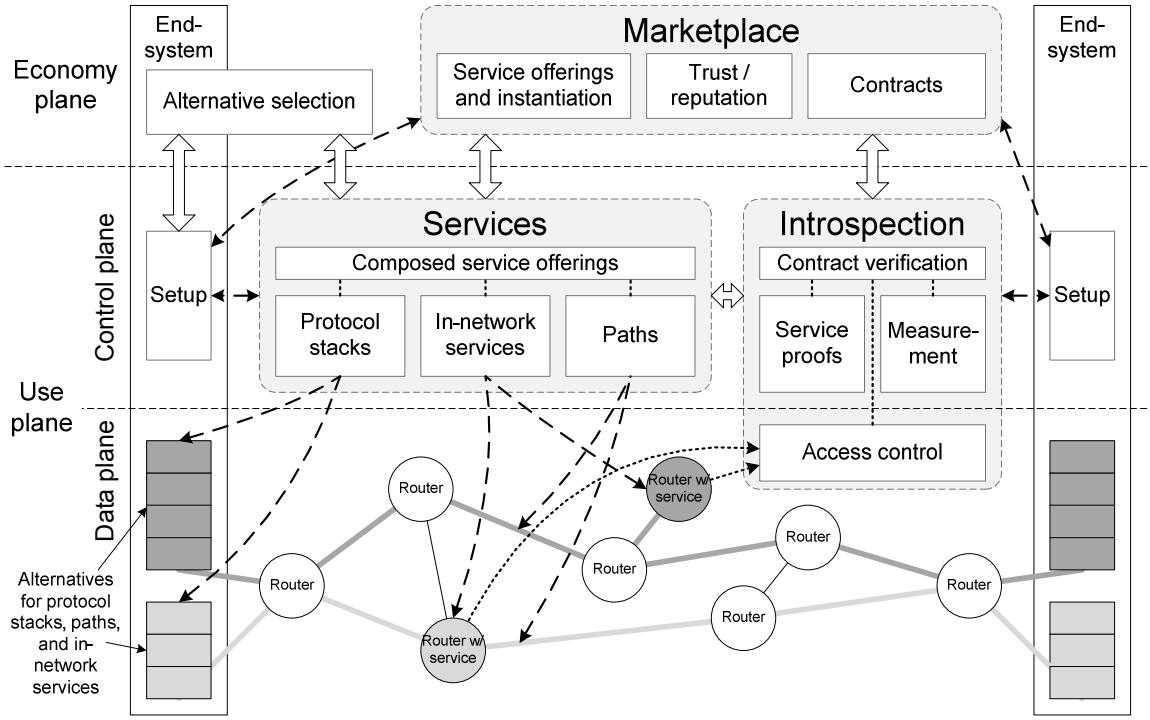


Figure 2.1. The architecture of ChoiceNet. [73]

In the use plane, the main entities are *clients* and *providers*. The provider will set up the services for authorized clients based on the contract established in the economy plane. The use plane contains the control and data plane like the current Internet does, and the services are provisioned in the way that control plane sending commands to data plane. Besides the service provisioning, the use plane also provides a mechanism for the entities to check what happened in the use of the services (the introspection). The providers can check whether the contract is valid (i.e. contract validation) and whether the client is authorized to use services (i.e. access control). On the other side, the clients can verify whether the delivered services meet the agreement (i.e. service proof and measurement).

2.2 ChoiceNet Components

This section describes the three key components in ChoiceNet. Some detail explanation can be found in latter chapters.

2.2.1 Services

The concept of service is adopted from the service centric networking [7,60,69,70]. In service centric networks, functionalities in the network are viewed as "network services" [26, 43, 69]. These services can be simple paths between nodes, and they can be more complex protocol processing and content storage. Service-centric network architectures then describe the semantics of various network services and allow composition of more complex services based on users' needs.

Services are generated by the local providers, then listed on the marketplace. End-system users and providers of service composition then purchase services from the marketplace. To create a competitive market for services, it is necessary to specify the semantics of services such that service offerings can be compared. At the same time, the semantics of services must be generic enough to allow adding new services.

Because all that a network service do is transferring data or modifying it, we define a service with 1) the *locations* of its input and output, and 2) the requirement of input format and the *transformations* it will apply to the input. Such definition allows easy composition of the services. The details of service definition and composition will be explained in Chapter 3.

2.2.2 Contracts

In the current Internet, the economic relationships are based on long-term, "paper-based" contracts. (e.g., monthly service agreements between users and network service providers, service-level agreements between providers, etc.). ChoiceNet enables contracts for network services at various time scales.

In ChoiceNet, contracts relate economic exchanges (e.g., payments) with operations within the network (e.g., access to a service). To be effective, contracts require enforcement. Thus, a customer needs to be able to verify that a service has been rendered to specification (e.g., as discussed in [6]) and a provider needs to be able

to perform access control to limit services to those customers who have established economic relationships. The latter is covered in Chapter 7.

2.2.3 Marketplaces

In order to choose, the customers need a place to find what services are available. The marketplace provides such a place where the providers can advertise services and customers can query them. To allow comparison across different services, ChoiceNet defines semantics for the service advertisement. There are mandatory attributes like service type, price and provider information. There are also supplementary attributes for each type of service, so that the customers can make choices based on them. The marketplace can also support service composition (e.g., as discussed in [27]).

The marketplaces may also act as trusted intermediary for economic transactions. It will reduce the risk of making payment to providers, and also help customers select services based on ratings.

The marketplace serves a central role in the ChoiceNet. To avoid single point failure and make the system scale up, multiple marketplaces may exist. They can help off-loading with each other, and compete with each other if they belong to different authorities.

Chapter 4 describes the design details of the marketplace.

2.3 ChoiceNet Interaction

This section introduces the interactions happen in ChoiceNet. Figure 2.2 shows the space-time diagram of interactions about the service advertisement and request.

The steps taken to set up connections (or more complex service offerings) in ChoiceNet are:

1. Providers advertise their services in one or more marketplaces.

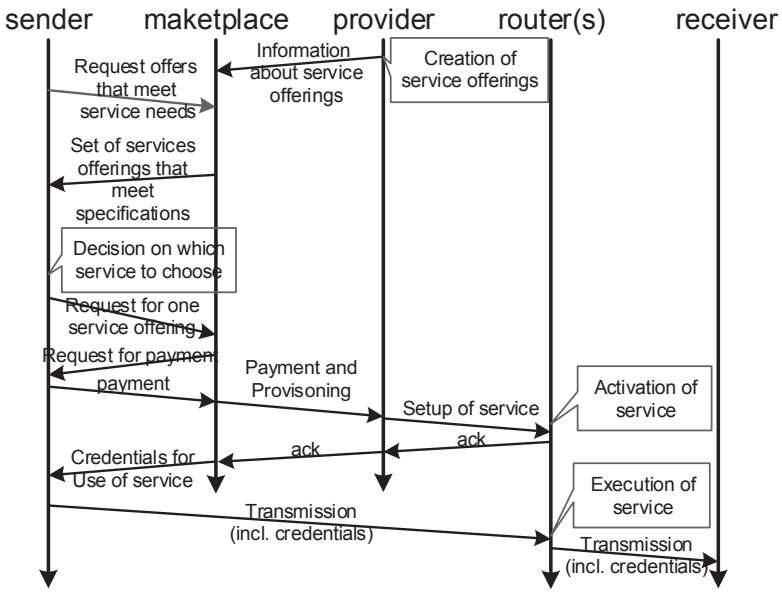


Figure 2.2. Space-time diagram of interactions in ChoiceNet. [21]

2. An end-system application (e.g., movie streaming app) queries the marketplace for available service offerings (e.g., QoS pipes, cached content).

3. The user (or a delegated entity, such as the operating system) makes a decision on which service to "purchase."

4. The providers involved in the service offerings set up their services in return for "consideration."

5. The end-system application uses the provided service.

A key challenge in this context is to connect the economic relationship among entities to the network services offered/purchased. In Chapter 4, we describe a protocol that establishes this connection.

CHAPTER 3

DEFINATION AND COMPOSITION OF SERVICES

At the heart of the ChoiceNet architecture is the concept of a *network layer service*. ChoiceNet's goal is to enable anyone to create a new network layer service, advertise the service to potential customers, provide the service, and be compensated for providing the service. Moreover, a user/customer (or a reseller/broker) should have the ability to combine network layer services together to form more complex services tailored to the specific needs of the user.

For example, a network layer service might be as simple as a "relay service" that forwards packets from one port on a switch to another port on a switch. A broker might then combine relay services together to create a (composite) "pathlet service" [34] that forwards packets along a particular path. Another broker might offer a service that combines pathlet services together to form an "end-to-end packet delivery service."

Although composed services have been explored in other contexts before [17, 39], past work has focused on the problem of integrating functionality, rather than that of compensating the operators of those services. A ChoiceNet network layer service not only needs to define "what the service does" so that it can be used/composed, but it must also specify "what a user of the service must do to compensate the provider of the service."

This chapter presents a network layer service abstraction for ChoiceNet, describe how it can be composed to form complex/tailored services, and how to use the ChoiceNet protocol for implementing the selection mechanism.

Some of the material in this chapter have been published in [22].

3.1 Consideration

In ChoiceNet, all network layer services require some form of *consideration* along with each service request. Consideration is the medium of exchange of value; that is, consideration is used by one party to convince another to provide a good or service. For practical reasons, the system must admit a variety of forms of consideration.[1] Some connection to a system for transferring money may be required (e.g., a credit card number or Bitcoin [54] transaction); in other cases a user may simply need to prove membership in some group (e.g., being a faculty member at a particular university). A receipt (proof of purchase) might also be accepted as consideration. In short, consideration in ChoiceNet can be any form that the customer and provider agree on for exchanging value.

3.2 Service Description and Composition

A network layer service description contains information about a service's characteristics. It is used to advertise the service in the marketplace, and is also used by *planning* services to compose services together. There are six parts to a network layer service description: (1) the data *transformation/operation*, (2) the *type of input* require, (3) the *type of output* generated, (4) the *input location*, (5) the *output location*, and (6) the *consideration* required. The first three components—the operation, input specification, and output specification—are similar to other interface description languages, web service definition languages, remote procedure calls, etc. The other three components are needed by the economy plane to sell/purchase services and compose them together.

[1] In some cases a network layer service might be offered for free and not require any particular consideration.

One can think of a network layer service as a channel with one or more input endpoints and one or more output endpoints. When the specified *consideration* is given along with request for service, the channel performs the specified operation, (possibly) transforming data arriving on the input endpoint(s) into data leaving on the output endpoint(s). The operation may also have side-effects (e.g., changing the state of the channel).

Composition is achieved by connecting the output from one channel to the input of another channel. However, it is not sufficient to know that a service's output type matches another service's input type. Channel endpoints need to be in the same *location* so that they can be connected. Locations are simply identifiers (names) selected from some namespace (i.e., *scope*) meaningful to the network layer service (e.g., ID of a switch, a port on switch, an AS number, an IP address, an ISP provider name, a geo-location, etc.). Endpoints sharing a location are composable, with ChoiceNet providing the functionality to connect output to input.

Network layer service descriptions are "advertised" by the network layer service to the marketplace, where the marketplace is itself a set of *marketplace services* that allow applications to browse or search the set of available services. Like all services, access to marketplace services requires consideration. Given the ability to discover available services (in the marketplace), one can implement *planning services*, which, given a particular request for service, identify (plan) a composed service that will meet the requirement. The planning service might then invoke *provisioning services* that "purchase" the planned set of composed services (i.e., providing the necessary consideration to each service), or it might return the plan to the user who would invoke a provisioning service to "purchase" the composed service. This ability to hierarchically compose services enables a variety of different business models including resellers, aggregators, brokers, etc.

3.3 ChoiceNet Protocol for Network Services

Conceptually, ChoiceNet services are "purchased" in the *economy plane* and "used" in the *use plane* (i.e., control and/or data plane). One of the challenges is to develop suitable protocols that enable both invocation of economy plane services and use plane services. For example, communication in the economy plane is likely to resemble conventional request/reply, client-server communication. Communication in the use plane, on the other hand, may take various forms, such as a client pushing data through a series of transformation services. Thus, it might seem that ChoiceNet should support two distinct communication protocols: one for customers purchasing services from providers, and another for applications using services.

While the economy plane/use plane distinction is conceptually useful, the services that are implemented in practice often cannot be easily classified as economy plane or use plane services. A path service, for example, may collect information from forwarding services to construct and sell paths and thus be considered a marketplace (economy plane) service, but at the same time be considered a use plane service because it computes and returns a set paths along with the "proof of purchase" needed to use those paths. In other words, it both *sells* forwarding service and *computes* paths, and this combination may be necessary to dynamically determine/set prices.

To embrace ChoiceNet's conceptual distinction between the economy plane and the use plane, but allow services to play both roles at the same time, we designed a *single* ChoiceNet communication protocol that is usable by services regardless of the plane to which they belong (or fall between).

3.4 ChoiceNet Protocol Messages

In the ChoiceNet protocol, services are invoked with a service *request* and may produce an *output*. Figure 3.1 shows the general structure of a *request* and *output*.

Figure 3.1. Basic components of a ChoiceNet (a) Request and (b) Output message.

The *request* message is similar to a remote procedure call, indicating which service should be invoked at the server and a list of arguments to be passed to the server. Unlike remote procedure calls, a ChoiceNet request also carries consideration. A generic service flags field carries flags understood by all services (e.g., a "price check" flag that allows a customer to learn the precise cost of performing tasks with a certain set of parameters). The flags field can also be used to indicate that certain fields will be carried in the payload, rather than the header; this allows larger values to be conveyed. Like the request message, the *output* message indicates for which service it is providing results. The message may also carry the output from the service (e.g., a list of "proof-of-purchase" tokens for use with a forwarding service).

3.5 ChoiceNet Protocol Interactions

There are several ways of how this simple ChoiceNet protocol can be used to create interactions that match realistic networking scenarios:

- Iterative use to enable choice: Choice is critical to enable service competition. To let users choose among multiple services, repeated ChoiceNet protocol operations can be used: First, a marketplace is queried to obtain a list of available services. The ChoiceNet protocol is used to send the query to the marketplace (and possibly provide consideration in case the search needs to be paid for). The marketplace returns the available services. In a second protocol exchange, the user then contacts the provider of choice to purchase the actual service.

- Recursive use for composed services: Network services may consist of several pathlets and potential processing and storage services. A provider can hide this complexity to a user by offering a single service. However, when a user purchases this service, multiple subservices need to be instantiated for use. In this case, the single ChoiceNet protocol interaction by the user may trigger multiple, recursive ChoiceNet protocol interactions.

- "One-shot" use for speed: In cases where the user has already made the choice of service, our ChoiceNet protocol can be used very efficiently since all necessary information (service selection and consideration) is included in a single protocol message. Thus, this information can be included in-band with data transmission and does not require additional messages between user and provider.

Note that all three scenarios use the same ChoiceNet protocol, but can achieve different goals.

3.6 Specifying Service Semantics

Having defined a common message structure for messages in both the economy and use planes, we ultimately need to define precisely what goes into each field of the messages shown in Figure 3.1. Depending on the target service, the information exchanged in these messages may range from simple flags and identifiers (similar to fields in an IP header) for forwarding services to complex XML structures for services that process packet payloads. Clearly, the customers and providers must agree on the meaning/semantics of the data carried in these fields. Much like there exist protocol standards for the network and transport layers of the current Internet, we expect similar standard will be defined for use/data plane services in ChoiceNet. However, services in the economy plane may rely instead on agreed upon *vocabularies* to define the semantics of messages.

To support a variety of different (extensible) vocabularies, we adopted a triple { *Attribute Name, Attribute Value, Vocabulary URL* } as the general structure for information being exchanged in the economy plane. *Attribute Name* identifies the import of the field, and is a literal that must be interpreted the same way by entities that exchange messages containing this attribute. That is, such entities must share a common *vocabulary*. A vocabulary, in this context, may be a simple *dictionary* of literals; the meaning or import of such literals is embedded in the logic of the entities exchanging the message. More generally, it is an *ontology*, where some of the rules for manipulation of such literals is embedded in the vocabulary itself. Examples of *Attribute Name* values are `ChoiceNet Version` or `Message Type`.

Attribute Value is a literal that provides the value of the attribute named by the *Attribute Name*. It may be a number, a string, a list, or it may nest a single, or multiple, other fields (whose values, in turn, may nest others). This allows ChoiceNet entities to ignore entire hierarchies of fields if they are not relevant to the entity's current role or interaction. In other words, an entity may understand the import of a message completely at the top level, without understanding all of the detail structure (but being able to pass them on, say, to another entity). For example, the concept of *consideration* can simply be represented by an attribute field with *Attribute Value* set to `Consideration`. Its value can be a nested structure, representing many different methods of transferring consideration such as mechanisms like PayPal, or previously established contexts like an account number to charge, or credit mechanisms like credit card numbers. Similarly, complex concepts like tokens can be encapsulated in single attribute fields with internal structure that can vary from use to use.

Finally, *Vocabulary URL* provides the basis for an extensible vocabulary, by allowing the sender of the message to indicate where the vocabulary being used for the value of the *Attribute Value* is available. It may well be that this vocabulary is the same as that needed to understand this field's *Attribute Name* itself, but the ability to

specify a different vocabulary for any field's *Attribute Value* allows providers of innovative services to immediately start using existing ChoiceNet marketplaces and other mechanisms, and incrementally build an ecosystem of other entities who understand the new custom vocabulary.

From the above, it is clear that services that rely on vocabularies must *a priori* understand all top-level attribute field *Attribute Name* values – this represents the bootstrapping vocabulary, and can be considered the common core vocabulary. This common core can be minimal. Further, we reasonably expect that the core vocabulary will grow over time, as practice makes it clear what vocabularies are most helpful to the ChoiceNet user community.

CHAPTER 4

MARKETPLACE

The marketplace is basically a server which accepts providers' advertisements and responds to customers' request. It is the intermediary of service, payment and trust. In this implementation, it is also responsible for the service composition. This chapter introduces the architecture of the marketplace, and how it interacts with the customers and providers.

4.1 The Architecture of Marketplace

In this implementation, the marketplace is a multi-threaded server written in Python, with MySQL as its database. The architecture of a single marketplace server is shown in Figure 4.1. To make the server scalable, it is designed to be multi-threaded. Every time a provider or a customer connects to the server, a new thread is created to handle the request.

The marketplace uses standard sockets to accept connections from the clients (i.e. the provider or the customer). To reduce the deployment complexity, there is only one persistent connection for each client, which is initiated by the client side. The requests and responses are encoded in JSON[1] text format. JSON is ideal for key-value pair representation, and it is easy to keep compatibility between different protocol versions. Two mandatory keys in the request and the response are *version* and *command*, representing the version of ChoiceNet protocol used and the name

[1] `http://json.org/`

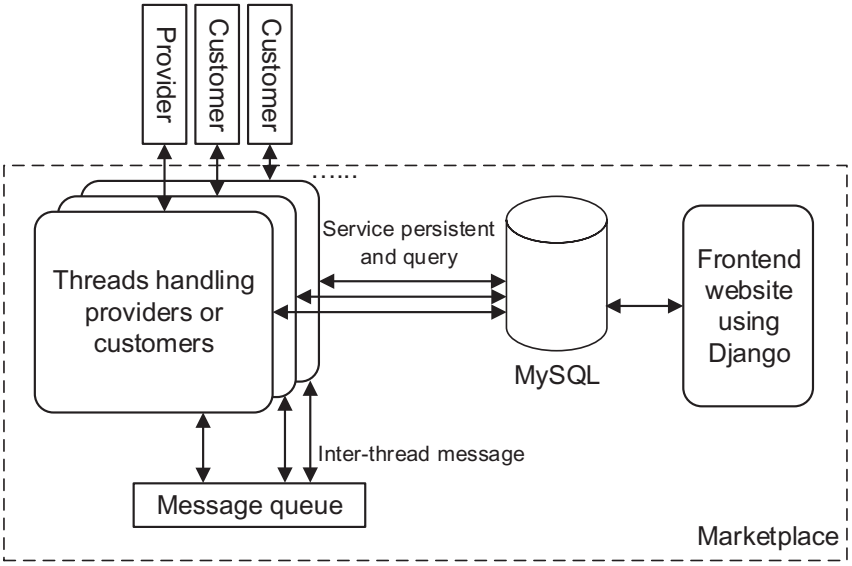


Figure 4.1. The architecture of marketplace server.

of the current command, respectively. Each command has its own extensive key definition, some examples of the key definitions are shown in Figure 4.2 and Table 4.1.

```
#request
{
    "ver": 0.1,
    "rpc_id": "0",
    "command": "client_connect",
    "client_type": "user",
    "client_id": "2fb5e13419fc89246865e7a324f476ec624e8740"
}
#response
{
    "ver": 0.1,
    "rpc_id": "0",
    "command": "client_connect",
    "response": True
}
```

Figure 4.2. JSON representation of a request and a response.

Table 4.1. Full list of marketplace command definition.

Command	Request		Response
	Key	Value	
client_connect	client_type	"user" or "controller"	True if accepted, otherwise False
	client_id	SHA1 hash of client's MAC and TCP source port	
get_client_info	–	–	{'client_id': the client's id, 'client_type':the client's type, 'client_ip': the client's IP observed by the marketplace, 'client_port': the client's TCP port observed by the marketplace}
check_marketplace_status	–	–	"ACTIVE" or "INACTIVE"
process_services	service_list	list of services	a list of boolean indicating whether each service is processed successfully
	op_code	"ADD" or "DELETE"	
delete_all_service	–	–	True if succeeded, otherwise False
submit_request	request_id	unique request id to identify transaction	fwd_path_options, rev_path_options
	src_ip	string of source ip	
	dst_ip	string of destination ip	
submit_path	request_id	request_id same as the previous submit_request	"FAILURE" or "SUCCESS"
	path_selection	seleted path No.	
stop_service	request_id	request_id same as the previous submit_request	True if succeeded, otherwise False
	service_id_list	list of services	

The communication between the marketplace and the clients is plain text for now. If there are security concerns, Transport Layer Security (TLS) can be used to encrypt the text, but this is not included in the current version of ChoiceNet protocol.

The requests are designed like remote procedure calls (RPC). Every request has a response. The request is synchronized, that is, the request halts until the response arrives. To prevent blocking by unpredictable failures, there is a 5 seconds timeout for each request.

Though the marketplace will never actively end a connection, a long-lived TCP connection may be unexpectedly torn apart by the client side or by network failures. To keep alive the TCP connection, the client sends a `check_marketplace_status` request to the marketplace every 20 seconds as a heartbeat. If the marketplace has not received any request from the client for 60 seconds, it assumes the client is dropped and end the connection.

The marketplace currently only offers one type of service—the pathlet service. It is a directional path defined by the *location* of source and destination, which are IP addresses. An example of pathlet service is shown in Figure 4.2. Note that the description varies for different service types, and the JSON representation ensures the extendibility in the future.

To handle payments, there is a web server co-located with the marketplace to perform authentication with PayPal. This web server interacts with the marketplace by sharing its database and exposes a HTTP-based JSON API to user applications for PayPal payments.

4.2 Pathlet Service Composition

The marketplace is responsible to compose the pathlet services. Each time the marketplace receives a new pathlet service reported by the provider, it stores it in the database. When the marketplace receives a service planning request, it constructs a

Table 4.2. Pathlet service example.

Attribute Name	Attribute Value
service_id	511d8dc5821e2b88495737ef6642e7461108955a
name	Link 10.1.0.2-10.3.0.2
description	1.00 Mbps, 15.00 ms latency, 0.01 USD/min
quantity	1
service_type	NetworkLink
controller_id	2fb5e13419fc89246865e7a324f476ec624e8740
controller_ip	192.168.0.15
end_point1_id	10.1.0.2
end_point1_ip	10.1.0.2
end_point2_id	10.3.0.2
end_point2_ip	10.3.0.2
service_bandwidth	1.0
service_latency	15.0
service_cost	0.01

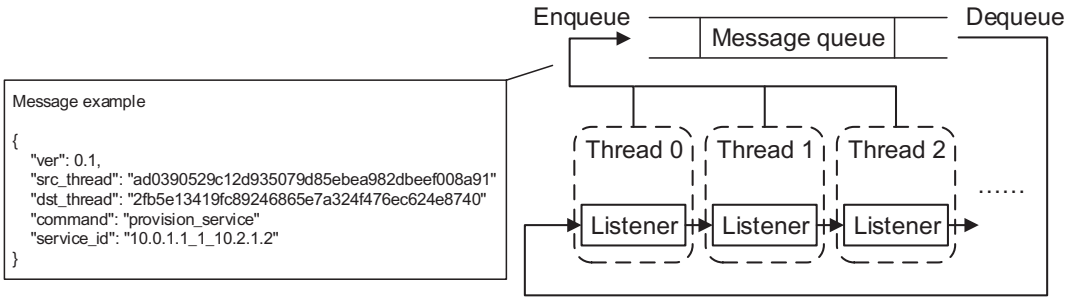


Figure 4.3. Inter-thread communication.

directional multi-graph from the pathlet services. It then uses the ParetoBFS algorithm to find all the Pareto-optimal paths, the detail of the ParetoBFS algorithm is presented in Chapter 6. After finding all the Pareto-optimal paths, they are presented to users for selection.

4.3 Inter-thread Communication

Sometimes, information needs to be shared between threads in marketplace. For example, when a thread receives a service provisioning request from a customer, it

needs to inform the provider's thread to start the service provisioning. Similarly, when the provisioning is done, a notification needs to be sent back to inform the customer. These all require an inter-thread communication scheme.

In this implementation, a message queue is used for inter-thread communication, as shown in Figure 4.3. Each thread has a thread ID, which is the same as the client ID that thread serves. Each message is a JSON string (similar to the marketplace communication protocol), and has a source thread ID and a destination thread ID. To handle messages, each thread registers a *listener* function in the message queue. When a message is sent to the message queue, all the threads will be able to see it, and the specified destination thread will be responsible to handle it. If the destination thread ID is a broadcast ID, all the threads will handle it.

CHAPTER 5

USE PLANE DESIGN

To leverage and maintain compatibility with existing applications, the use plane needs to be based on the IP protocol. A fundamental requirement of ChoiceNet is to support alternative paths. In particular, ChoiceNet needs to support per-flow dynamic routing based on the user/application's requirements, but legacy static routing and adaptive routing does not meet these requirements.

There are more than one way to implement such dynamic routing in the use plane. The University of Kentucky team has implemented a source routing based implementation in [22] using the Click modular router [44]. This work uses another approach, which is using *Software Defined Networking* (SDN).

SDN is an approach that decouples network systems into the control plane and the data plane [56]. Such decoupling enables the control plane to control each flow with flexibility. Using SDN, we can allocate the path for each flow by installing flow entries on switches along designated path. This approach allows providers in ChoiceNet to provision path services to users.

The use plane consists of the control plane and the data plane. This chapter introduces the control logic in the control plane, and how it manages the network in a centralized way while keeping the compatibility of end systems. This chapter also introduces the application installed on end systems – the ChoiceNet App, which manipulates packets and interacts with the marketplace.

5.1 Control Plane Design

The controller is the representative of the provider. Each provider domain (e.g. an Autonomous System) has a controller. In this implementation, the controller is an SDN controller with customized control logic. It detects the topology of switches with LLDP packets and detects hosts by their DHCP, ARP, and IP packets. When a new host or a new link has been detected, the controller updates the new pathlet service to the marketplace, thus the marketplace knows all the services in all ASes. Another task of the controller is provisioning – once a provisioning request is received, the controller installs flow entries on the switches along the designated path.

The main difference in paradigm of a controller in ChoiceNet and a standard SDN controller is: the installation of flow table entries is not triggered by the first packet of each flow. Instead, flow table entry installations are triggered by the provisioning command from the marketplace–after the users have requested and paid for the service.

5.1.1 Host Discovery in ChoiceNet

One difference between ChoiceNet and the traditional Internet is: ChoiceNet has to have loops inside the network to provide alternative path, and the loops should not be removed by spanning tree protocol. Therefore, broadcast protocols such as ARP must be handled differently to avoid broadcast storm. This section describes how the IP addresses are discovered and resolved in ChoiceNet:

- Each AS has a *virtual gateway*, which is used to keep compliant with the end system's gateway settings. It has a virtual IP and MAC, which doesn't have to be real.

- When an end system joins the network, it either 1) uses DHCP to get an IP address and the gateway setting, so the controller can know the existence of the

end system, or 2) uses static IP address, the controller can know the existence of the end system by detecting its ARP and IP packets.

- When an ARP request is received by an edge switch, the switch does not flood it to other ports. Instead, it sends the packet to the controller. Because the virtual gateway is configured as all the end systems' first hop, the destination IP of the request should be either a host within the same layer 2 network, or the virtual gateway's IP. In either condition, the controller has the record of the requested IP and its MAC address. It then dictates the edge switch to directly reply this ARP request.

- Because the MAC address may be virtual, the packets may not be accepted by the NIC when they reach a layer 3 router or reach the destination host. To solve this problem, when the controller installs flow entries into the switches, if the MAC address of the destination IP is known by the controller, it will add a "modify destination MAC" action in the flow entry of the last hop, to make sure the router or host's NIC can receive the packet.

5.1.2 Inter-domain routing

The inter-domain routing is done by the controller. Whenever a new host joins, the controller calculates the path between the new host and existing hosts as well as peering ports to other ASes. The controller then reports the paths as pathlet services to the marketplace. Since the controller knows the topology of the entire AS, given both ends, it is easy to use Breadth First Search algorithm to find all the paths between them. The paths are named by the IP address of the two ends, connected by a enumerate number to distinguish multiple path between the same end points, e.g. "10.1.0.2_0_10.3.0.2".

The pathlet services also include parameters about the link quality, such as throughput, latency and drop rate. These are manually configured in this implementation,

but they can be automatically detected by the method introduced in A. C. Babaoglu et al.'s work [5].

When a host disconnects, the controller will detect this through timeout scheme, then delete the corresponding pathlet services related to that host. The same applies to paths disconnection between switches.

5.1.3 Intra-domain routing

To achieve the peering between ASes, there are exchange points which connect two or more ASes. The exchange point works at layer 2 network, and each AS know other AS's peering port IP, either by manual configuration, or by the protocol introduced in SDX [35]. The controller consider the peering ports as "virtual hosts," and reports pathlet services between the host and peering ports' IP. The service is reported asymmetrically: the outgoing pathlet service is from the host IP to the other AS's peering port's IP, and the incoming pathlet service is from local AS's peering port's IP to the host IP. For example, in Figure 5.1, AS1 will report 10.0.0.2_0_10.1.0.1 and 10.0.0.2_0_10.2.0.1 as the outgoing pathlets, and 10.0.0.1_0_10.0.0.2 as the incoming pathlet. In this way, the marketplace can naturally compose the multiple segments of each path direction.

5.2 ChoiceNet App Design

On the user's system, to avoid rewriting every application to communicate to the marketplace, a program called ChoiceNet App is used to intercept out-going connections and to contact the marketplace on behalf of the application. A lot of solutions can be used to intercept packets, such as Windows Filtering Platform [47], NetFilter Queue[1], Intel DPDK[2], or writing a customized kernel module or driver.

[1] `http://www.netfilter.org/projects/libnetfilter_queue/`

[2] `http://dpdk.org/`

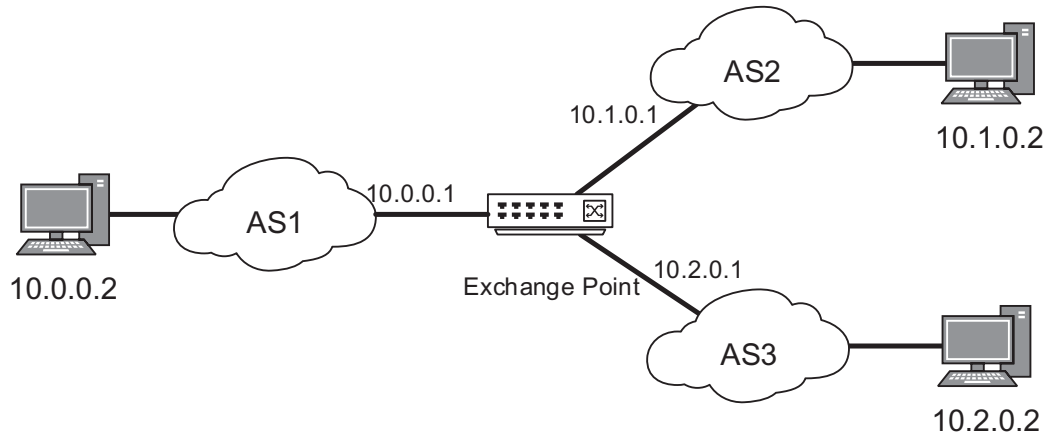


Figure 5.1. Peering between multiple AS.

This implementation uses NetFilter Queue for packet interception, because it can easily manipulate packet from the user space.

Figure 5.2 shows the function of the ChoiceNet App. It intercepts the initial packet of the desired type of connections (e.g. connections with destination port 80 if one want to make HTTP traffic to use ChoiceNet). The App then contacts the marketplace, asking for a path service to the destination IP. After the marketplace returns a list of available service combinations, the App prompts the user to select one service. After the selection, the user is redirected to a PayPal payment page. After receiving the payment notification from PayPal, the marketplace transfers the money to the account of the controller(s) and notifies the latter to provision the services. After the provisioning, the App releases the intercepted packet and traffic will traverse through the assigned path.

It may be impractical for the user to select and pay for each network connection. Instead, network services can be made more granular (e.g., encompassing all connections to a video service provider for 2 hours) and preferences can be specified in the

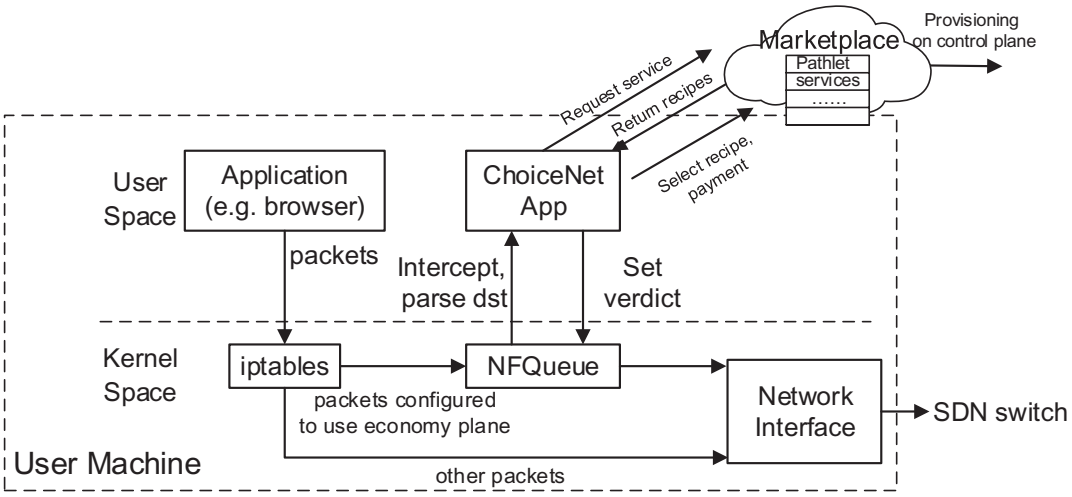


Figure 5.2. ChoiceNet App interactions on end system. [22]

ChoiceNet App to automate the service selection process. A "prepaid" account can be set up in the marketplace to avoid frequent PayPal authentication, too.

5.3 Incremental Deployment in Legacy Network

The previous sections discussed about the details in a pure ChoiceNet environment. However, it is impossible to deploy the ChoiceNet in one night. For one, SDN is not available end-to-end on the Internet; for another, the deployment pace of each provider are different. Therefore, an incremental deployment method is necessary, so that users can make use of the economy plane even if the core of Internet is still legacy network. We proposes two hybrid deployment method in two dimensions: hybrid architecture and hybrid application.

5.3.1 Hybrid architecture

Since ChoiceNet keeps the compatibility of the IP layer, it can connect to the legacy with no problem. As shown in Figure 5.3, if one edge provider supports ChoiceNet architecture, its users can have choices both within the AS and outside

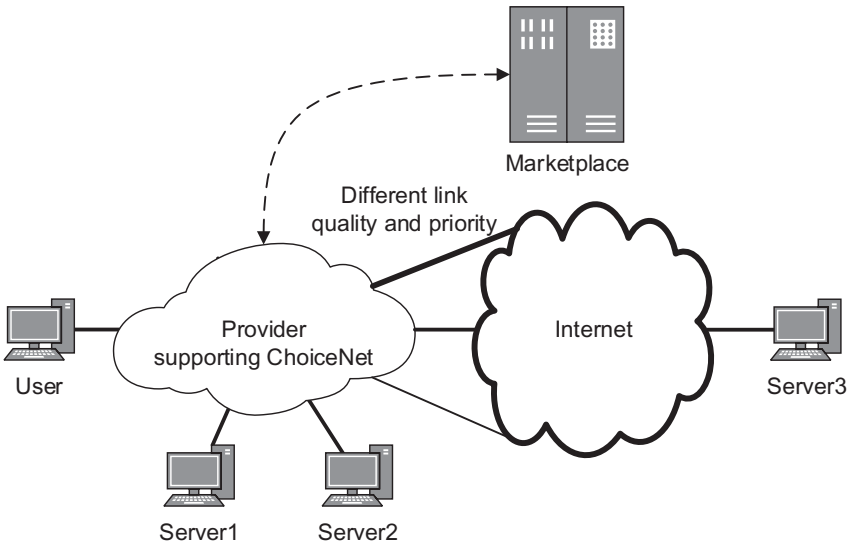


Figure 5.3. Hybrid architecture of ChoiceNet and the traditional Internet.

the AS. When the user is requesting a service, it can choose between local servers and remote servers (e.g. choose between Server1,Server2 and Server3), they can also choose from different egress link quality. The *user* here can be either a single host, or an NAT LAN using legacy network. As long as the user machine has ChoiceNet App deployed, it can make some level of choice.

5.3.2 Hybrid application

Making all the protocols negotiating with marketplace before transmission is expensive, and not all applications on the user side need an economy plane. The ChoiceNet App can set up rules in iptables to separate legacy traffic and the traffic that wants economy plane. For example, if a user wants only online video traffic to use economy plane, a rule can be set up in iptables, sending only the packets that has the video server as the destination IP and port to NFQueue. Legacy traffic can still be routed through SDN and other parts of Internet using traditional routing method.

CHAPTER 6

MULTI-CRITERIA PATH FINDING ALGORITHM

Path finding algorithm is a fundamental functionality in ChoiceNet. In the use plane, the providers need to find paths within its domain. In the marketplace, the pathlet services form a graph, and the service composition is also achieved by path finding algorithms. Onur et al. have proposed a scalable architecture for path computation [3], but it does not introduce any specific path finding algorithm. This chapter will focus on the algorithmic foundations for efficiently computing alternative paths in ChoiceNet.

Routing in the current Internet uses a single criterion, such as hop count or link weight. Although there are proposed solutions to the multi-criteria optimal path selection problem for quality-of-service routing, since the routers eventually need to pick only one path, they usually combine all criteria into a single path optimization metric a priori.

These traditional routing methods, however, do not apply to ChoiceNet, because a single metric cannot provide path alternatives to customers that weigh metrics a posteriori. ChoiceNet's routing algorithm must be able to consider more than one criterion (e.g. bandwidth, delay, price) and provide paths representing trade-offs between different criteria. On the other hand, the marketplace should remain neutral when offering services, thus, the routing algorithm must provide *all* the Pareto-optimal paths to the customers.

This chapter presents ParetoBFS, a variant of a breadth-first search that uses branch and bound techniques to find all the Pareto-optimal paths while effectively

limiting the potentially very large search space. We present several sampling techniques to further increase the speed of the search while degrading the quality of the results only marginally. The simulation results show that existing multi-criteria combinatorial optimization approaches can only search a small fraction of all the Pareto-optimal paths while our ParetoBFS can obtain the whole Pareto-optimal path set in shorter time.

Some of the material in this chapter have been published in [20].

6.1 Introduction

Routing, which is determining a path for traffic to flow between communicating end-systems, is one of the essential functionalities of any computer network. In typical networks, routing is based on a single criterion, such as path length, delay, or an artificially defined "weight." Widely used routing protocols, such as OSPF [53] and RIP [38], use single routing metrics and corresponding routing algorithms, such as Dijkstra's algorithm [24] and the Bellman-Ford algorithm [12], to efficiently determine the optimal path between two network nodes.

However, in ChoiceNet, single-criterion shortest paths no longer fit the whole spectrum of services. The cost of a path and the quality of a path need to be represented by independent metrics. When only a single metric is used, a single optimal solution (i.e., shortest path) is enough. But when multiple metrics are used, a set of paths needs to be found to represent the trade-offs among criteria. A key challenge for realizing multi-criterion path finding is the need to develop an efficient algorithm for determining suitable paths in the potentially very large space of all possible paths (exponential to the number of nodes). The multi-criteria path finding is an NP-hard [36] problem, but it is possible to develop solutions for typical-sized networks that work well in practice.

Previous work has addressed the multi-criteria optimal path problem in various contexts, for example Quality of Service (QoS) routing. A central problem in QoS routing is to find feasible paths between a source and a destination that satisfy multiple constraints (e.g., bandwidth, delay). Then, the best path among the feasible paths is selected based on a given optimization metric (e.g., delay-constrained least-cost path routing). When there are multiple optimization metrics, most approaches rely on an combinatorial optimization function [48], which combines all metrics into a single metric (e.g., weighted sum).

Using a single, combined metric simplifies the path finding problem, but also presents a fundamental limit on the ability to find solutions: a single optimization metric requires *a priori weighing* of each metric [28]. That is, before the path finding algorithm is run, the relative "value" between different metrics needs to be set. The result of the search is then optimal (only) for this fixed weighing of metrics. In ChoiceNet, however, this weighing cannot be done a priori, and the multi-criteria optimal path problem needs to find the set of *all Pareto-optimal* paths. A path is Pareto-optimal if there is no other path that is better in all metrics. Since multiple metrics allow for the existence of paths that are better than others in one or more metric, but not all, there can be a large number of mutually Pareto-optimal paths. Based on the set of Pareto-optimal paths, one path can be chosen for any possible weighing of metrics by the customer.

This chapter presents ParetoBFS, a variant of the breadth-first search (BFS) algorithm that uses Pareto constraints to prune the traversal tree. Experiments show that ParetoBFS can find *all* Pareto-optimal paths in a network in a reasonable time since typical-sized networks do not exhibit the characteristics that cause the problem space to become intractable. The specific contributions of this work are:

- The ParetoBFS algorithm that can find the entire set of Pareto-optimal paths in a network where the edges have arbitrary number of metrics, both sum-

and bottleneck-type, which cannot be achieved by most of existing approaches. Comparison with two existing algorithms shows ParetoBFS is tens to hundreds times faster and find more paths.

- A sampling heuristic for ParetoBFS that reduces the number of elements in the set of Pareto-optimal solutions and thus decreases the complexity of the path finding process. We show that despite not yielding all optimal solutions, this heuristic still yields solutions that are useful in practice.

- Results from simulation on both realistic and generated network topologies, as well as deployment in the ChoiceNet prototype.

We believe that this work provides a practical foundation for systematically using multi-criteria path finding in ChoiceNet.

The remainder of the chapter is structured as follows. Section 6.2 describes background in the area. Section 6.3 provides the formal description of the multi-criteria path finding problem. Section 6.4 describes the ParetoBFS algorithm. Section 6.5 presents the complexity analysis and experimental results. Section 6.6 introduces several sampling heuristics for ParetoBFS. Section 6.7 compares ParetoBFS with related work. Finally, Section 6.8 summarizes the key points of this chapter.

6.2 Background

Multi-criteria path finding has been studied extensively in the operations research community. This problem arises in many practical applications, including route planning in traffic networks [11] and QoS routing and traffic engineering in communication networks [65]. If the goal is to find the optimal path with some constraints on one or more metrics given a directed graph with edges that have a set of metrics, it is called multi-constrained path optimization (MCPO) [19, 29, 45, 48, 63, 66, 74]. Without the constraints on the metrics, this problem then becomes the multi-criteria optimization

(MCO) problem [28, 36, 48, 58]. Solutions to MCPO and MCO are usually similar in that they use a combinatorial function on the multiple metrics (a priori) to find the optimal path.

The goal of ParetoBFS is to find all the Pareto-optimal paths, which is different from the prior work. Therefore, ParetoBFS is a broader solution to address both MCPO and MCO problems since the resulting paths from previous approaches are usually a subset of the Pareto-optimal path set. These Pareto-optimal paths are important in many scenarios. For example, references [42] and [30] each describe a standalone routing service module that provides paths for other modules. Thus, the routing service module itself cannot make any choice for metric preferences. Also, in networks where paths are charged by their qualities, such as ChoiceNet [71], the cost and the quality of a path need to be represented by independent metrics. In these problems, there is no single objective function to select the best path, and it is impossible to give the paths an a priori ranking. Instead, the decision maker needs to see all the *Pareto-optimal* paths. Each Pareto-optimal path represents a trade-off between criteria, and may be equally important for the decision entity.

Section 6.7 compares the performance of ParetoBFS with some prior work in detail. The experiments show ParetoBFS is tens to hundreds times faster and can solve broader range of problems.

6.3 Problem Statement

Before describing the ParetoBFS algorithm in Section 6.4, we briefly introduce the network model and describe the formal definition of our path finding problem.

6.3.1 System Model

We model the network as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of edges interconnecting the nodes. n and m are the cardinalities of V and E, i.e., $n = |V|$, $m = |E|$, respectively.

To make the problem general enough, we consider that G is a directed multi-graph, which means there can be multiple edges between each node pair. (In practice, these multiple edges can correspond to different services that are offered on the same physical link, such as different QoS configurations.) In addition, we assume that each edge $\{e_{u,v}|u, v \in V\} \in E$ is associated with an edge criteria vector $w(u, v) = (w_1, w_2, ..., w_k)$, where k is the number of criteria. Each w_i corresponds to one of the independent criteria used in routing, such as bandwidth, latency, packet pass rate and cost. A path p from a source v_1^p to a destination v_r^p is defined as a finite sequence of edges that connects a sequence of vertices $(v_1^p, v_2^p, ..., v_r^p)$, $v_{i(i\leq r)}^p \in V$.

The path p can be assigned a path criteria vector $w^p = \{w_1^p, w_2^p, ..., w_k^p\}$. In this paper, the calculation of the path criteria vector must satisfy the following property: when a hop is added to the path's end, the optimality of the new path does not increase on any criterion. Criteria satisfying this property can usually be classified into two types: sum-type criterion (e.g., delay) where $w_i^p = \sum_{e_{u,v}\in p} w_i(u, v)$; and bottleneck-type criterion (e.g., bandwidth) where $w_i^p = min(w_i(u, v))$[1].

6.3.2 Pareto-optimal Path

To define Pareto-optimality, we first define a *dominant path* as follows. We use the notation $\succeq$ to denote the left operand is more optimal than or equals to the right operand.

Definition 1 (Dominant path) *path p dominates path q if and only if*

[1]There are also multiplicative criteria (e.g. link reliability, packet loss rate), but they can be transformed into sum-type criteria by using a logarithm.

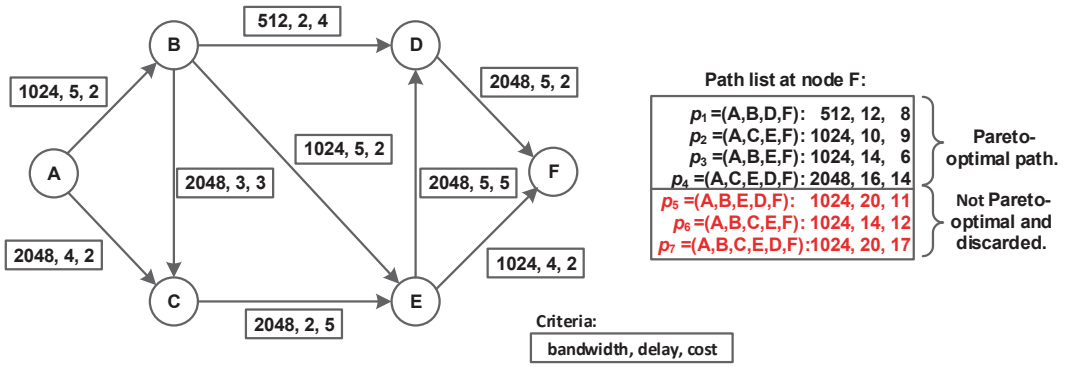


Figure 6.1. Example of Pareto-optimal path computation from node A to F.

$$w_i^p \succeq w_i^q, \forall i \in \{1, 2, ..., k\}.$$

and the strict inequality holds at least once.

Then we can define Pareto-optimality as:

Definition 2 (Pareto-optimal path) *Path set P is called a Pareto-optimal set if and only if*

$$p \text{ does not dominate } q, \forall p, q \in P.$$

A path in a Pareto-optimal set is called a Pareto-optimal path.

In this paper, the goal is to find all the Pareto-optimal paths from a source node to a target node in a given graph G. For instance, if each edge $e \in E$ has three metrics: bandwidth (w_1), delay (w_2) and cost (w_3), then the set of the Pareto-optimal paths P, which we finally find out, satisfies that, for $\forall p_i,\ p_j \in P, w_1^{p_i} > w_1^{p_j} \vee w_2^{p_i} < w_2^{p_i} \vee w_3^{p_i} < w_3^{p_j}$. This is different from the conventional multi-constrained optimal path problem [48], where a path optimization function f^p is used to combine all the metrics together and the optimal path is found by calculating the value of f^p on each path. As discussed above, the optimal path computed based on a single aggregated metric may not meet the multiple constraints being considered.

An example of the type of result we are aiming to obtain is shown in Figure 6.1. The edges of the graph are labeled with their respective metrics comprising of bandwidth (w_1), delay (w_2) and cost (w_3). There are seven paths $(p_1, p_2..., p_7)$ from source node A to destination node F. Among these paths, path $p_2 = (A, C, E, F)$ is strictly more optimal than path $p_5 = (A, B, E, D, F)$ because $w_2^{p_2} < w_2^{p_5}$ and $w_3^{p_2} < w_3^{p_5}$ while $w_1^{p_2} = w_1^{p_5}$. Therefore, path p_5 is not a Pareto-optimal path and would be discarded. Similarly, neither of the paths p_6 and p_7 are not Pareto-optimal paths because p_2 and p_3 is strictly more optimal than them. Finally, we get the Pareto-optimal paths $p_{1\sim4}$. (In the ParetoBFS algorithm, we maintain a list on each node to record all the Pareto-optimal paths to this node and their corresponding parameters. Such a list is shown in black on node F in Figure 6.1.)

6.4 ParetoBFS: Pruning with Pareto Constraints

In this section, we first describe the plain breadth first search (BFS) solution to the multi-criteria path finding problem. Then, we describe how we use pruning to reduce the running time of the algorithm to a practical level.

6.4.1 Plain BFS to Find All Paths

A brute force solution to the multi-criteria path finding problem is to enumerate all the possible paths, then extract the Pareto-optimal set from them.

Algorithm 1 shows a variant of BFS algorithm that finds all the simple paths from the source node to a target node. Unlike the normal BFS, it does not maintain "visited" tags on the nodes, because a node may be visited multiple times when the algorithm examines different paths. Algorithm 1 starts from a source node and enqueues it into a path queue, i.e., *path_queue*. Then, the source node is dequeued and all the directed edges of it are enqueued into *path_queue* as new paths from the source node to some node in the graph. Each time a path is dequeued from

Algorithm 1 BFS that finds Pareto-optimal paths by enumerating all the simple paths between two nodes

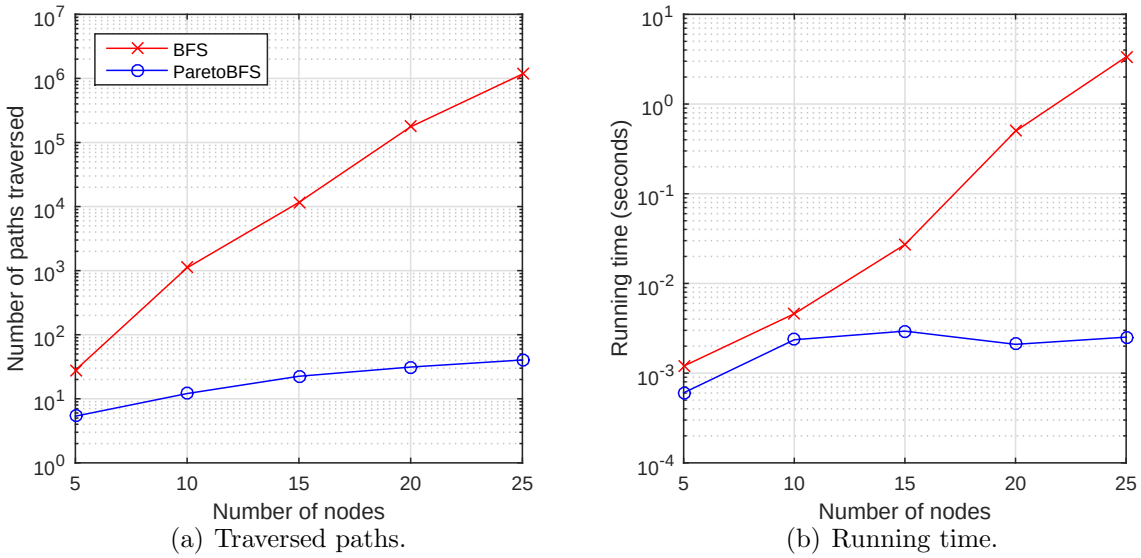
```
 1: procedure BFS(G, source, target)
 2:     for all v ∈ G(v) do
 3:         path_set[v] ← ∅
 4:     end for
 5:     path_queue.push([source])
 6:     while path_queue.length > 0 do
 7:         path ← path_queue.pop()
 8:         s1 ← path.end()
 9:         for all edge ∈ s1.out_edges() do
10:             s2 ← edge.dest_node()
11:             if s2 ∉ path then
12:                 new_path ← path.append(edge)
13:                 path_set[s2] ← path_set[s2]∪
                                {new_path}
14:                 if s2 ≠ target then
15:                     path_queue.push(new_path)
16:                 end if
17:             end if
18:         end for
19:     end while
20:     pareto_set ← ∅
21:     for all path ∈ path_set[target] do
22:         pareto_set ← pareto_add(pareto_set, path)
23:     end for
24:     return pareto_set
25: end procedure
```

path_queue, it is stored into the path set corresponding to its last node. Meanwhile, the out-edge neighbors of the dequeued path's last node are added to its end to form new paths. These new paths are further enqueued into *path_queue*. To prevent loops, Line 11 checks whether the neighbor is already in the path before appending it. After repeating this enqueue and dequeue process until *path_queue* is empty, *path_set* contains all the simple paths[2] from source node to all other nodes. Selecting

[2]A simple path is a path which does not have repeating nodes.

Algorithm 2 See if a path is Pareto-optimal for a Pareto-optimal set. If it is, add it to the set. It may evict existing paths from the set.

```
1:  procedure PARETO_ADD(pareto_set, new_path)
2:      result_set ← ∅
3:      for all path ∈ pareto_set do
4:          if path is strictly more optimal than new_path then
5:              return pareto_set
6:          else if new_path is not strictly more optimal than path then
7:              result_set.append(path)
8:          end if
9:      end for
10:     result_set.append(new_path)
11:     return result_set
12: end procedure
```



(a) Traversed paths. (b) Running time.

Figure 6.2. ParetoBFS and BFS comparison.

a Pareto-optimal set from it is straightforward, as shown in function *pareto_add* of Algorithm 2.

Algorithm 1 can be easily extended to find the Pareto-optimal paths from one source node to all the other nodes, by replacing Line 13 with a *pareto_add* function, removing Line 14, and doing Lines 20 - 23 on each node.

The algorithm is obviously not scalable. In a directed graph, the number of possible paths is usually exponential to the number of nodes. Moreover, for a multi-graph with p parallel edges between each pair of nodes, the total number of paths increases with a factor of p^h, where h is the number of hops in a path. Figure 6.2 shows the comparison with respect to the number of traversed paths and running time for ParetoBFS and BFS. It is run on BRITE-generated topologies, with 2 metrics and 1 parallel edge. The result is average over 60 runs with different graphs and source/target nodes. Figure 6.2(a) shows the number of paths traversed in Algorithm 1. It grows exponentially; enumerating all the possible paths is typically not feasible in both time and space. To make Algorithm 1 practical, it is necessary to prune the space of paths that are considered during the traversal.

6.4.2 ParetoBFS – Pruning While Searching

Since our goal is to find Pareto-optimal paths, we can stop considering a path if it is already strictly worse than other known paths. We call this process *pruning*. Formally, during the search process, a path ending with node v_i can be pruned if either of the following conditions satisfies:

1. The path is dominated by a path in the Pareto-optimal path set with destination node v_i.

2. The path is dominated by a path in the Pareto-optimal path set with destination node $target$.

An algorithm with such pruning maintains the same theoretical worst-case time and space complexity. In practice, however, pruning reduces the size of the search tree dramatically. Note that pruning does not affect the correctness of the final solution, because extension cannot make a suboptimal path optimal.

Applying the pruning method to Algorithm 1, we can get the ParetoBFS algorithm as shown in Algorithm 3. Instead of saving all the paths, a set *pareto_set* is used

Algorithm 3 ParetoBFS

```
1: procedure ParetoBFS(G, source, target)
2:     for all v ∈ G(v) do
3:         pareto_set[v] ← ∅
4:     end for
5:     path_queue.push([source])
6:     while path_queue.length > 0 do
7:         path ← path_queue.pop()
8:         s1 ← path.end()
9:         if path is Pareto-optimal for pareto_set[target] and path ∈ pareto_set[s1]
   then
10:                 ▷ Check whether the path satisfies the Pareto-optimal conditions
11:             for all edge ∈ s1.out_edges() do
12:                 s2 ← edge.dest_node()
13:                 if s2 ∉ path then
14:                     new_path ← path.append(edge)
15:                     if new_path is Pareto-optimal to pareto_set[target] and
   pareto_set[s2] then
16:                                 ▷ see if new_path can be added into the Pareto-optimal
   path set to node s2
17:                         pareto_add(pareto_set[s2],
                                       new_path)
18:                         if s2 ≠ target then
19:                             path_queue.push(new_path)
20:                         end if
21:                     end if
22:                 end if
23:             end for
24:         else
25:             continue
26:         end if
27:     end while
28:     return pareto_set[target]
29: end procedure
```

to save the Pareto-optimal paths from the source node to each node. It differs from Algorithm 1 in Lines 9, 15 and 17. Lines 15 and 17 check the Pareto-optimality before the enqueue step, to eliminate any suboptimal path. There is another check after the dequeue step in Line 9, because the Pareto-optimal sets may have changed during the time that path stays in the queue. Figure 6.2(a) shows that the pruning method

can effectively reduce the number of traversed paths by several orders of magnitude. The detailed performance and complexity analysis is shown in Section 6.5.

Algorithm 3 can be extended to find Pareto-optimal paths to all other nodes, by removing the condition checks involving the *target* Pareto-optimal set in Lines 9, 15 and 18. The running time increases because of the less strict pruning conditions.

6.5 Evaluation and Complexity Analysis

In this section, we discuss the effectiveness of our ParetoBFS algorithm in the context of network graphs to show that it is practically useful.

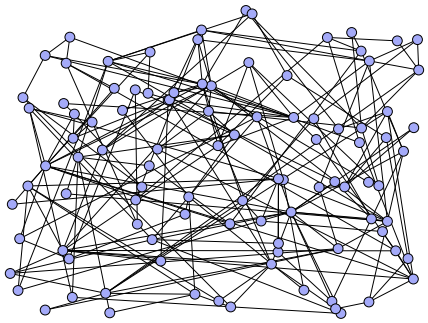
6.5.1 Methodology

To test the performance of the path finding algorithm, we use both generated topology and real-world topology. Although ParetoBFS can apply to both inter- and intra-AS topologies, most of the intelligent routing applications are used within private domains. So we focus on the intra-AS topology here. We use the BRITE topology generator [52] to generate router-level topologies. The sizes of the topologies range from 100 nodes to 10,000 nodes. BRITE provides three metrics for paths: length, bandwidth and latency. When testing with more than 3 metrics, we add extra random parameters besides these 3 metrics.

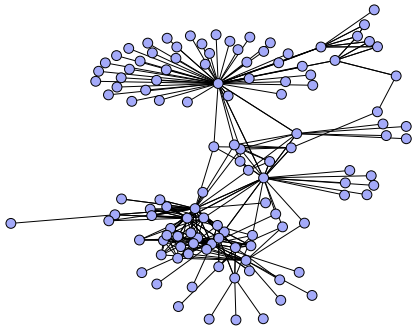
BRITE provides four generation models: Waxman [67], BA [9], BA-2 [1] and GLP [15](the GLP model is mainly for AS-level topologies). The node placement has two options: random and heavy-tailed. The bandwidth distribution has four options: constant, uniform, exponential and heavy-tailed. We test all the combinations on graphs with 1,000 nodes, 3 metrics and 1 parallel edge. We list the running time (seconds) and the Pareto-optimal path count in Table 6.1. Each result is an average of 100 runs. It can be observed that, except for the constant options, other combinations of parameters do not show significant difference in the path finding result. Therefore,

Table 6.1. Different BRITE parameters' impact.

Node Placement	Bandwidth Distribution	Model							
		Waxman		BA		BA-2		GLP	
		time	paths	time	paths	time	paths	time	paths
Random	Constant	0.03	1.00	0.03	1.00	0.06	1.00	0.03	1.00
	Uniform	0.36	7.46	0.26	5.22	0.64	7.42	0.12	2.24
	Exponential	0.36	6.60	0.23	4.16	0.62	7.22	0.09	1.94
	HeavyTailed	0.42	6.64	0.28	4.84	0.68	7.40	0.12	2.36
Heavy Tailed	Constant	0.04	1.00	0.05	1.00	0.08	1.00	0.03	1.00
	Uniform	0.59	8.46	0.37	5.24	0.89	7.78	0.15	1.98
	Exponential	0.52	7.22	0.30	5.78	0.81	7.96	0.15	2.46
	HeavyTailed	0.48	7.62	0.25	4.68	0.84	7.76	0.13	2.70



(a) BRITE generated topology, 100 nodes.



(b) Rocketfuel topology, AS 4755, 121 nodes.

Figure 6.3. Examples of test topologies.

we can arbitrarily pick these parameters. In the following experiments, the generation model is set to Waxman, a most commonly used intra-AS model, the node placement is set to random, and the bandwidth distribution is uniform distribution.

As for the real-world topology, we use Rocketfuel [61], an ISP topology data set measured by the University of Washington. Each Rocketfuel data file represents a topology of one AS, ranging from 100 nodes to 10,000 nodes. The data we use does not include any metric such as bandwidth or latency, so we randomly generate values for the metrics using a normal distribution.

Both the generated and the real-world topologies are uni-graphs, i.e., topologies with only one edge between the same pair of nodes. However, sometimes we need more than one edge between two nodes, these parallel edges can be either physical links with different metrics, or service offerings on the same link but with different QoS limits. To extend the uni-graphs to multi-graphs, each edge of the uni-graph is duplicated and assigned with Pareto-optimal metrics.

We use Python to implement our algorithms because of its convenient graph libraries, and the ability to integrate into the pox[3] SDN controller, which also uses Python. We use the pypy[4] interpreter to run the experiments, which can achieve performance close to the native code. One exception is the convex sampling in Section 6.6, we use CPython for that experiment because the convex hull calculation uses pyhull, which is not pypy compatible.

The processor we use is an Intel Core2 Quad CPU Q9400 running at 2.66 GHz. The software configuration is Ubuntu 14.04 64-bit with kernel version 3.13.0-24 and pypy 2.6.0.

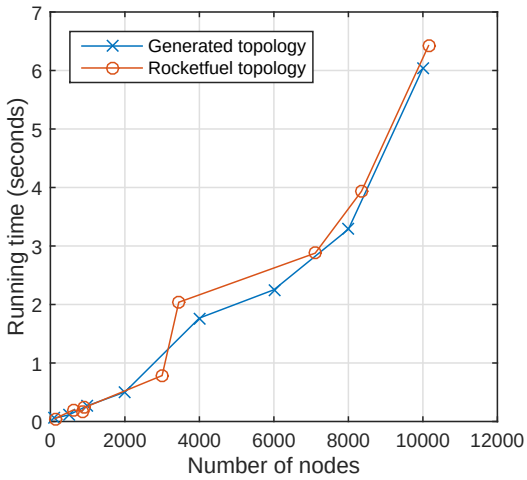
6.5.2 Complexity Analysis

In this section, we provide a theoretical analysis on the plain BFS and ParetoBFS algorithms (i.e., Algorithms 1 and 3). Let $G = (V, E)$ be the graph, where $V = (v_1, v_2, ..., v_n)$ is a set of all nodes of the graph and $E = (e_1, e_2, ..., e_m)$ is a set of all edges of the graph. The number of criteria is k. We assume the source node is v_1 and the target node is v_n.

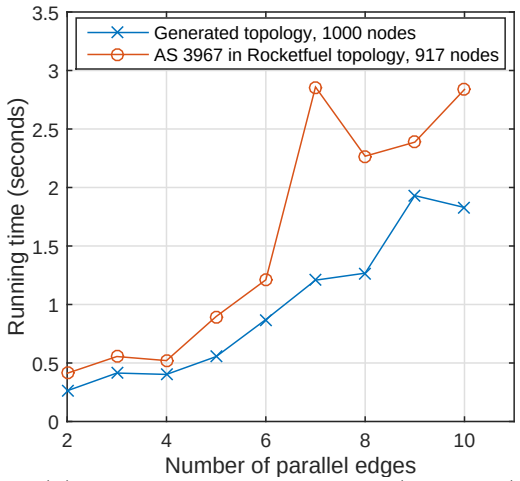
Recall that Algorithm 1 first finds all possible paths and then the Pareto-optimal paths among all these paths. On the other hand, Algorithm 3 finds the Pareto-optimal

[3]http://www.noxrepo.org/pox/about-pox/

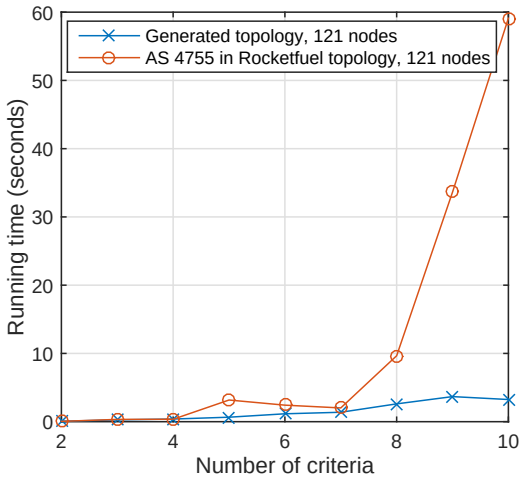
[4]A Python interpreter with JIT compiler. http://pypy.org/



(a) Number of nodes. (2 criteria, 3 parallel edges)



(b) Number of parallel edges. (2 criteria)

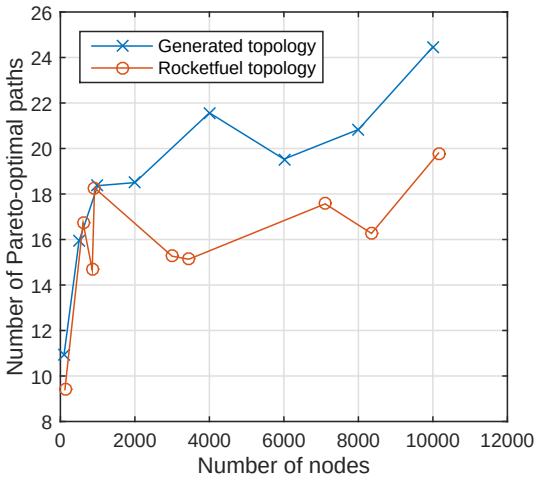


(c) Number of criteria. (3 parallel edges)

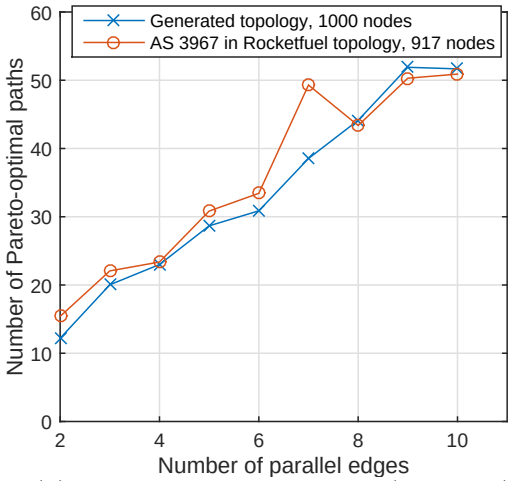
Figure 6.4. The time complexity of ParetoBFS to different variables.

path each time when it visits a node. We first analyze the time to find all the paths in Algorithm 1.

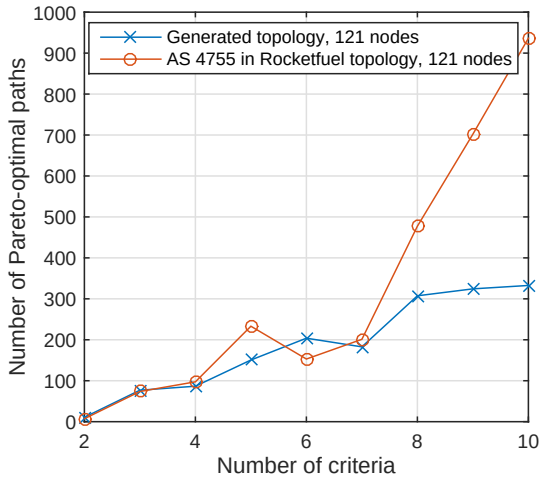
As discussed in Section 6.3, a suboptimal path cannot become optimal when a hop is added to its end. Therefore, all Pareto-optimal paths considered in this paper are simple paths, which do not have repeating vertices. In a directed graph, for a simple path, we can order the vertices so that edges only point forward. E.g., if node u is a descendent of node v, then node u comes after node v in the sorted list of nodes.



(a) Number of nodes. (2 criteria, 3 parallel edges)



(b) Number of parallel edges. (2 criteria)



(c) Number of criteria. (3 parallel edges)

Figure 6.5. The number of Pareto-optimal paths found.

In Algorithm 1, the times that each node v_i $(i = 1, 2, ..., n)$ is visited are the number of the paths from source node v_1 to node v_i. Let v_2 be the next node. The number of paths from v_1 to v_2 is the number of parallel edges between them. Let v_3 be one of v_2's neighbours, the number of paths from v_1 to v_3 is the number of (direct) edges from v_1 to v_3, plus the paths that use v_2 as an intermediate vertex. More generally, let $e(i, j)$ be the number of directed edges between node v_i and node v_j $(e(i, j) = 0$ if

v_i and node v_j are not adjacent nodes), and $d(j)$ be the number of paths from v_1 to v_j, then we have:

$$d(j) = e(1, j) + \sum_{i=2}^{j} d(k)e(k, j).$$

For each node v_j, computing $d(j)$ takes time proportional to the in-degree of node v_j, and overall it will take $O(m)$ time. Therefore, Algorithm 1 visits each node $O(m)$ times, and the total time to find all the possible paths in Algorithm 1 is $O(nm)$ time. To calculate the complexity of the Pareto selection phase, we denote p as the number of all the paths from source node v_1 to target node v_n. p could be 1 if there is only 1 simple path from node v_1 to node v_n, however, p could also be $n!$ when graph G is full mesh (each node connects to every other node). The operation of Algorithm 2 takes $O(k)$ times computation for each path in the input *pareto_set*. The process of screening out the Pareto-optimal paths adds 1 Pareto-optimal path each time from the temporary *pareto_set*, and the number of paths in *pareto_set* goes from 0 to $p-1$. Therefore, the process will compute $O(k(1 + 2 + \cdots + p)) = O(kp^2)$ times. Then, the running time for Algorithm 1 is $O(nm + kp^2)$.

In contrast to Algorithm 1, Algorithm 3 deletes the non-Pareto-optimal paths from source node v_1 to node v_j each time when it visits node v_j. Therefore, the number of paths saved in *path_queue* in Algorithm 3 will be less than that of Algorithm 1. The number could be the same when all paths are Pareto-optimal. Thus, in the worst case, Algorithm 3 also visits each node $O(m)$ times. We denote p^* as the Pareto-optimal paths between the source node v_1 and the target node v_n. The total running time for Algorithm 3 is $O(nmkp^*)$.

The time complexity of Algorithm 1 is dominated by the number of the paths p. In fact, in a typical network topology, p usually grows exponentially with the number of nodes n. We can take the graph in Figure 6.1 as an example. If we have 2 parallel edges between each connecting node pairs, then number of the paths from

node A to node F becomes $3 \times 2^3 + 3 \times 2^4 + 1 \times 2^5 = 104$, which is much larger than n ($n = 6$). Besides, the number of the possible paths doubles when a new node is added into the graph. On the contrary, the time complexity of Algorithm 3 may not be dominated by the number of the Pareto-optimal paths p^* when p^* is just a small fraction of p. However, the optimal path fraction would grow rapidly when the number of considered metrics increases. In this case, the time complexity is dominated by p^*, and also grows approximate exponentially with n. The experimental results in the next section indicate the correctness of our analysis here.

6.5.3 Experimental Results

In this section, we present the experimental results of the ParetoBFS algorithm. We first present the running time of the plain BFS and ParetoBFS algorithms in Figure 6.2(b). It shows that the running time of plain BFS increases exponentially with the increase of the number of nodes. The complexity of ParetoBFS is sub-exponential, i.e., the running time may grow faster than any polynomial solution but is still significantly smaller than an exponential solution. This makes sense because ParetoBFS's running time grows exponentially with the number of nodes in the worst case, which happens when the number of the Pareto-optimal paths makes up a large part of the paths between the source and target node. However, in a realistic network topology, the Pareto-optimal paths are usually a small fraction of the total paths. So the pruning method can prevent the curve from going too steep, because it keeps removing non-Pareto-optimal paths at each node, therefore it avoids unnecessary comparisons afterwards.

We then present the running time of ParetoBFS to find all the Pareto-optimal paths in graphs with different parameters. Here, we only focus on the running time. The memory consumption is proportional to the running time, because it depends on the length of the *path_queue*. Figure 6.4 shows how the average running time of

ParetoBFS grows with the increasing number of nodes, parallel edges and criteria, respectively. Each data point is an average of 30 runs. Figure 6.4(a) shows that ParetoBFS can find all the Pareto-optimal paths on a 10,000-node topology in 30 seconds. Figure 6.4(b) shows a similar complexity with the number of parallel edges as in Figure 6.4(a). This is also reasonable because increasing the number of parallel edges and increasing the number of nodes have the same effect on the traversal queue length, and the pruning methods also have similar effects on these two metrics. Figure 6.4(c), however, shows a steeper growth than the previous figures. For instance, if there are a number of k metrics $w_1, w_2..., w_k$ on each edge (the value of w_k is generated randomly), considering two neighboring nodes with two parallel edges connecting them, the probability that these two edges are Pareto-optimal is $1 - \frac{1}{2^{k-1}}$. When k grows, the number of the Pareto-optimal paths between two nodes approaches the number of all the paths between them. This is the worst case for ParetoBFS which makes the running time grows exponentially. The large number of metrics also makes the Pareto pruning not working efficiently, which makes the running time grow faster than in Figure 6.4(a) and 6.4(b). In order to reduce the running time when the number of metrics is high, Section 6.6 proposes several sampling methods to reduce the size of the Pareto-optimal set.

Figure 6.5 shows how the number of the Pareto-optimal paths, p^*, grows with the increasing number of nodes, parallel edges and criteria, respectively. In Figure 6.5(a), the Rocketfuel-topology curve fluctuates, because each real topology has unique interior structure which is not as uniform as the generated topology. In Figure 6.5(b), it can be observed that the number of the Pareto-optimal paths varies linearly with the number of parallel edges when there are 2 criteria on each edge. The curves show the correctness of the analytic $O(nmkp^*)$ running time for ParetoBFS in the last section when compared with Figure 6.4(b). When the number of parallel edges doubles, p^* and m also double. Therefore, if the curves of p^* in Figure 6.5(b) can be considered as

Algorithm 4 Sampling the Pareto-optimal set after adding a path. The algorithm is used in place of the *pareto_add* function. The sampling function can be one of the considered sampling methods.

```
1: procedure SAMPLING_ADD(pareto_set, new_path)
2:     result_set = pareto_add(pareto_set, new_path)
3:     if result_set.length > th then
4:         sampled_set = sampling(result_set)
5:         return sampled_set
6:     else
7:         return result_set
8:     end if
9: end procedure
```

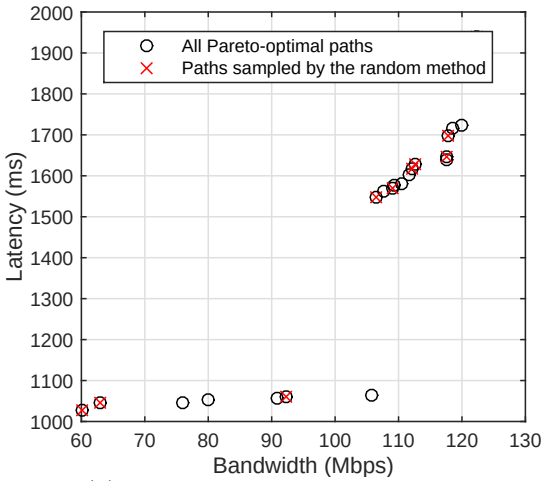
linear, the curves are polynomial in Figure 6.4(b). Figure 6.5(c) shows how p^* varies with the number of criteria. It can be observed that p^* increases in Figure 6.5(c) more than in Figure 6.5(a) and in Figure 6.5(b). As discussed in Section 6.5.2, a large number of the criteria results in the number of the Pareto-optimal paths approaching the number of all the paths.

6.6 Sampling Pareto-optimal Paths

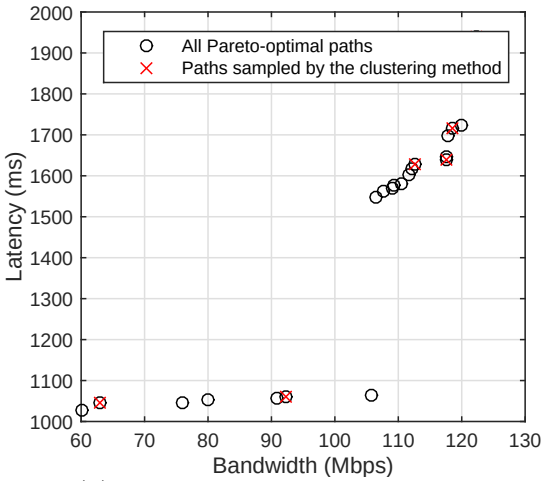
ParetoBFS finds all the Pareto-optimal paths. But as the number of criteria increases, the size of Pareto-optimal set may grow exponentially. Even for a small 1,000-node network with just 3 criteria, there may be hundreds of Pareto-optimal paths between two nodes.

Sometimes it is not necessary or too slow to find all the Pareto-optimal paths, so we introduce a heuristic based on sampling. Sampling the Pareto-optimal set can be useful in two ways: (1) sampling reduces the difficulty of choice for the entity selecting among Pareto-optimal paths; (2) if sampling happens during the search, the number of traversed paths can be further reduced, and the algorithm can find a set of paths that are close to the Pareto-optimal set in a shorter time.

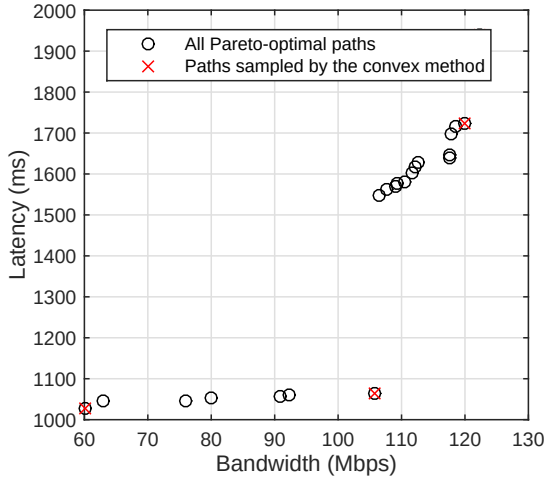
Algorithm 4 describes how to sample paths. Every time after a path is added to the Pareto-optimal set, the algorithm checks the Pareto-optimal set size. If the size



(a) Random sample, $th = 10$, $l = 6$.



(b) clustering sample, $th = 10$, $l = 6$.



(c) Convex sample, $th = 10$.

Figure 6.6. The effect of different sampling methods. (2 criteria, 3 parallel edges, 10,000 nodes)

is larger than a threshold th, a sampling method is used to reduce the Pareto-optimal set to l paths. It should be noted that sampling can discard some useful path halfway. It is possible that the final result is not a subset of the original ParetoBFS result.

Assuming $P = \{p_1, ..., p_m\}$ is the Pareto-optimal set found by ParetoBFS, and $Q = \{q_1, ..., q_n\}$ is the Pareto-optimal set found by ParetoBFS with sampling. To compare the effectiveness of the sampling methods, we propose the following metrics:

- Running Time Ratio (RT) is defined as the ratio of the running time to find Q to the running time to find P. This metric indicates how the sampling method affects the running time.

- Path Count Ratio (PC) is defined as the ratio of Q's size to P's size, that is, $PC = n/m$. This metric indicates how many Pareto-optimal paths can be found using this sampling method. It does not indicate the optimality of the paths.

- Path Quality (PQ) is defined as the average k-dimensional Euclidean distance between P's and Q's criteria vector sets $w^Q = \{w^{q_1}, ..., w^{q_n}\}$ and $w^P = \{w^{p_1}, ..., w^{p_m}\}$. Each w^{q_i} or w^{p_i} is a Pareto-optimal path's criteria vector. To calculate PQ, first normalize w^P and w^Q into w^P's range:

$$w_j^{p_i\prime} = \frac{w_j^{p_i} - \min(w_j^{p_1} \dots w_j^{p_m})}{\max(w_j^{p_1} \dots w_j^{p_m}) - \min(w_j^{p_1} \dots w_j^{p_m})}, \substack{i\in\{1\dots m\} \\ j\in\{1\dots k\}}$$

$$w_j^{q_i\prime} = \frac{w_j^{q_i} - \min(w_j^{p_1} \dots w_j^{p_m})}{\max(w_j^{p_1} \dots w_j^{p_m}) - \min(w_j^{p_1} \dots w_j^{p_m})}, \substack{i\in\{1\dots n\} \\ j\in\{1\dots k\}}$$

  then for each $w^{q_i\prime}$, calculate the distance from its closest $w^{p_t\prime}$:

$$d^{q_i} = \min_{t\in\{1\dots m\}} \sqrt{\sum_{j\in\{1\dots k\}} (w_j^{q_i\prime} - w_j^{p_t\prime})^2}$$

  Then PQ can be defined as: $PQ = \frac{1}{n}\sum_{i=1}^{n}(d^{q_i})$. It can be viewed as the average distance between w^P and w^Q, $PQ = 0$ means Q is a subset of P.

The sampling method must be fast and be able to process an arbitrary number of criteria. Assuming there is no preference over any criterion, the sampling methods should treat each criterion equally. In this section, three sampling techniques are investigated: random, clustering, and convex sampling.

6.6.1 Random Sampling

This method randomly samples l paths from the Pareto-optimal set. It is fast, but does not make use of any information of the data points. The result of a 2-criteria example is shown in Figure 6.6(a). Q mostly overlaps with P, which means that, after sampling, we can still find an approximate subset of the Pareto-optimal paths.

6.6.2 Clustering Sampling

It is an intuitive idea to cluster Pareto-optimal points that are close to each other in the k-dimensional space, especially when looking for redundant paths is not the goal. Here, we use Lloyd's clustering algorithm [49] to divide the points into l groups, and select the points closest to the center of each group.

Lloyd's algorithm's time complexity is $O(nkli)$ (n being the number of points; k being the dimension; l being the number of groups; i being the number of iterations). The example of a clustering result is shown in Figure 6.6(b). The points are more dispersed than Figure 6.6(a), thus they are more representative.

6.6.3 Convex Sampling

The assumption of convex sampling is that the points on the convex hull are better than the ones inside. This can be illustrated by Figure 6.7. Points 1-5 are Pareto-optimal points. Points 1, 2, 4, 5 and the *nadir point* (not a real data point) forms the convex hull. Point 3 is inside the hull. Compared to Point 2, Point 3 only improves a little in bandwidth, but sacrifices a lot in latency. The similar situation applies to Point 3 and 4. Therefore, Points 2 and 4 seems more preferable than Point 3. This method works better if the criterion is sum-type, because the points on the convex hull are more likely to stay optimal when the path is extended.

We use the *qhull* library, which implements the Quickhull algorithm [10]. Its time complexity is $O(nlogv)$ in 2-d and 3-d, and $O(n^{\frac{v(\lfloor d/2 \rfloor - 1)}{\lfloor d/2 \rfloor}})$ for higher dimensions (n being the number of points; v being the number of points on the convex hull). The

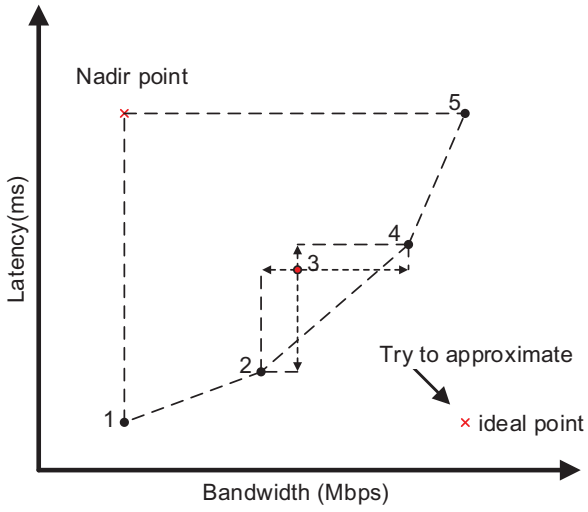


Figure 6.7. Example of convex sampling.

result in Figure 6.6(c) successfully eliminates the points inside the convex hull. For dimensions higher than 4, the performance of qhull degrades rapidly, it may no longer help speeding up the algorithm.

The advantage of the convex sampling is that it always reserves the corner points (e.g. Points 1 and 5 in Figure 6.6(c)), which represent the extreme values in one dimension, and they are more important if the decision maker wants to choose the highest value in one dimension. Another advantage is that the calculation of the convex hull does not require normalizing each dimension, thus improving the speed.

The disadvantage is that the convex sampling cannot control how many points are sampled. It is possible that too few or too many points are left, which brings uncertainty to the quality of the sampling result as well as the running time.

6.6.4 Comparison of Sampling Techniques

We test the three sampling techniques on 9 Rocketfuel topologies, whose sizes range from 121 to 10,152, and get the average RT, PC and PQ. The results are listed in Table 6.2. The sampling threshold th and sample size l also affect RT, PC

Table 6.2. The effectiveness of sampling methods.

k	th	l	random			clustering			convex		
			RT	PC	PQ	RT	PC	PQ	RT	PC	PQ
2	10	5	1.175	0.850	0.141	1.632	0.869	0.004	1.058	0.828	0.001
3	20	10	0.530	0.461	0.022	1.405	0.455	0.026	0.431	0.546	0.007
4	100	10	0.473	0.384	0.030	1.087	0.413	0.032	0.393	0.502	0.030

and PQ. They are chosen from trial runs, to get a compromise between the running time and the result accuracy.

As for RT, the sampling techniques do not reduce the running speed when $k = 2$, but they tend to reduce the running time at higher dimensions. The random and convex sampling speeds are about the same. The clustering is much slower than the other two, thus is not recommended. As for PC, all the three techniques can find a similar amount of Pareto-optimal paths, even for 4 criteria problems, they can still find 40% to 50% of the Pareto-optimal paths. The convex sampling performs slightly better at higher dimensions. As for PQ, the convex sampling has the best path quality, but its PQ increases much faster than the other two, this may be because the convex sampling cannot control the sample size, so the result accuracy is less adjustable.

Overall, the convex sampling works the best among the three sampling methods, at least for $k = 2, 3, 4$. It is faster, finds more Pareto-optimal paths with higher path quality. Therefore, the convex sampling is recommended when dealing with 2, 3 and 4 criteria topologies.

6.7 Comparison with Related Work

As discussed in Section 6.2, much previous work has addressed the multi-criteria path finding problem. There are several survey papers and bibliographies [28, 32, 50, 64], which summarize more than 40 papers about the multi-criteria shortest path

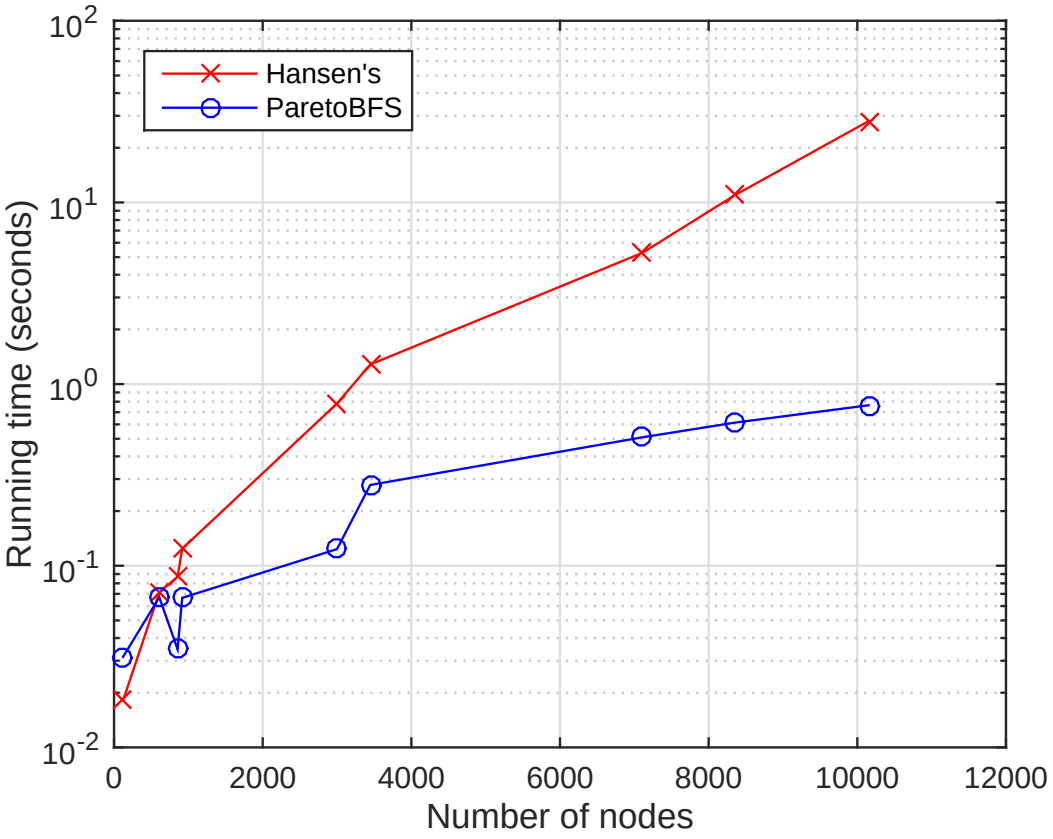


Figure 6.8. Running speed of Hansen's algorithm and ParetoBFS. (1 sum-type metric and 1 bottleneck-type metric, 1 parallel edge, Rocketfuel topology)

problem. Unfortunately, most of the papers only deal with sum-type metrics. Only two papers – Hansen [36] and Pelegrin et al. [58] – consider one sum-type and one bottleneck-type metric. Gandibleux et al. have a paper considering one bottleneck-type and an arbitrary number of sum-type metrics [31]. We have implemented Hansen's algorithm, and the comparison with ParetoBFS is shown in Figure 6.8.

The Hansen's algorithm examined here is Algorithm 2 in reference [36]. It uses a multiple labeling scheme. Since Hansen's algorithm finds the exact Pareto-optimal set, we only compare the running time here.

Figure 6.8 shows the running speed between Hansen's algorithm and ParetoBFS, we can see that ParetoBFS's running time grows slower with increasing nodes. Even

for small topologies with a few hundred nodes, ParetoBFS is as fast as Hansen's algorithm. For the large topology with 10,000 nodes, ParetoBFS is about 40 times faster than Hansen's algorithm. Not to mention that Hansen's algorithm is only designed for the bi-criteria problem, while ParetoBFS is capable of dealing with more criteria.

Other than the *exact methods* (i.e. to find all the Pareto-optimal paths) like ParetoBFS and Hansen's algorithm, many papers propose *approximation methods* to find a subset of Pareto-optimal paths in an efficient manner. These are known as *fully polynomial approximation schemes* (FPAS). All the FPAS we investigated are only for sum-type metrics[5]. Here, we compare ParetoBFS with a popular FPAS – Martins' algorithm [51].

Martins' algorithm only gives an approximation of the Pareto-optimal set, which may differ from the exact Pareto-optimal set. Similarly, we compare the quality of results as in Section 6.6. The results on 4 Rocketfuel topologies are shown in Table 6.3. Even for graphs with hundreds of nodes, the running speed of Martins' algorithm is tens to hundreds times slower than ParetoBFS. On larger Rocketfuel topologies, Martins' algorithm becomes too slow to be feasible. Though Martins' algorithm finds a reasonable portion of the Pareto-optimal set (about 40% to 60%) and the quality of paths is very close to the exact Pareto-optimal set, Martins' algorithm is too slow compared to ParetoBFS. Besides, ParetoBFS can find all the Pareto-optimal paths while Martins' algorithm only finds a part of them.

Table 6.4 compares the complexity of ParetoBFS with Hansen's algorithm and Martins' algorithm. From the comparison, we can see that ParetoBFS is superior than prior work in various aspects: It is able to take an arbitrary number of sum-

[5]In some work (e.g., [48]), it is suggested that bottleneck types can be converted to sum types by reciprocal. That is, define the optimal goal as: $f^p = \sum_e \frac{1}{bandwidth(e)}, e \in p$, where p is a path and e is an edge on p.

Table 6.3. Comparing Martins' with ParetoBFS on 4 Rocketfuel topologies.

# of	k=2			k=3			k=4		
nodes	RT	PC	PQ	RT	PC	PQ	RT	PC	PQ
121	1.1	0.56	0.0000	1.3	0.41	0.0000	1.5	0.44	0.0000
609	23.7	0.42	0.0050	178.7	0.38	0.0018	121.2	0.38	0.0003
855	126.5	0.68	0.0000	233.4	0.61	0.0004	258.9	0.53	0.0007
917	34.4	0.41	0.0074	169.6	0.24	0.0008	279.1	0.37	0.0006

Table 6.4. Comparison of path finding algorithms. (p^* and p are the numbers of all the Pareto-optimal and possible paths between two nodes, respectively.)

Type	Number of criteria	Number of Pareto-Optimal paths	Complexity
Plain BFS	k	p^*	$O(mn + kp^2)$ $(p > p^*)$
ParetoBFS	k	p^*	$O(mnkp^*)$
Hasen's [36]	2	p^*	$O(p^{*2} \log n)$
Martins' [51]	k sum-type metrics	ω $(\omega < p^*)$	$O(k^2 \frac{m}{n} \omega^2 \log \omega)$

type and bottleneck-type metrics. Besides, it finds the full Pareto-optimal set faster than other exact methods. Our experiments also show that it is even faster than certain FPAS in practice.

6.8 Summary

This chapter addresses the problem of finding multiple, mutually Pareto-optimal paths in a network, where multiple criteria are used for path finding. Such information is necessary in ChoiceNet, where the marketplace needs to provide path choices to customers for a posteriori selection.

This chapter describes ParetoBFS, an algorithm to find all the Pareto-optimal paths in a network. Experiments show it works well and the algorithm can get a solution on a typical network in reasonable time. This work also proposes several sampling techniques to further reduce the running time when finding all the Pareto-

optimal paths is not necessary or not feasible. Results from both generated and real topologies has been presented to show that ParetoBFS is practically useful.

CHAPTER 7

ACCESS CONTROL IN DATA PLANE

In ChoiceNet, an important problem is to make sure only authorized users (i.e., those who have paid for a particular network service) can access the service. Most existing authentication approaches are based on cryptographic techniques. However, cryptography has high computational cost, making it unsuitable for the data plane of the network, where potentially every packet needs to be checked at Gigabit per second link rates. This chapter describes a novel design for data plane access control, called OrthCredential. The main idea is to use a set of *orthogonal* sequences as credentials that can be verified easily to protect the data plane against various attacks. These orthogonal sequences can be constructed by Hadamard matrices. The evaluation shows that OrthCredential only requires less than 300 processor cycles for verification with 64-bit credentials, much less than existing access control schemes such as HMAC. And it provides reasonable security strength (e.g., less than 10^{-8} probability of successful attack).

Some of the material in this chapter have been published in [16].

7.1 Introduction

A key technical challenge in ChoiceNet is to provide *access control* to the network services. For example, in a source routing based ChoiceNet implementation, the paths are specified in the packet header. It is possible that a malicious user (i.e. attacker) can modify the packet header to use paths it should not use. Therefore, it is necessary to check the packets before they are granted access to certain resources.

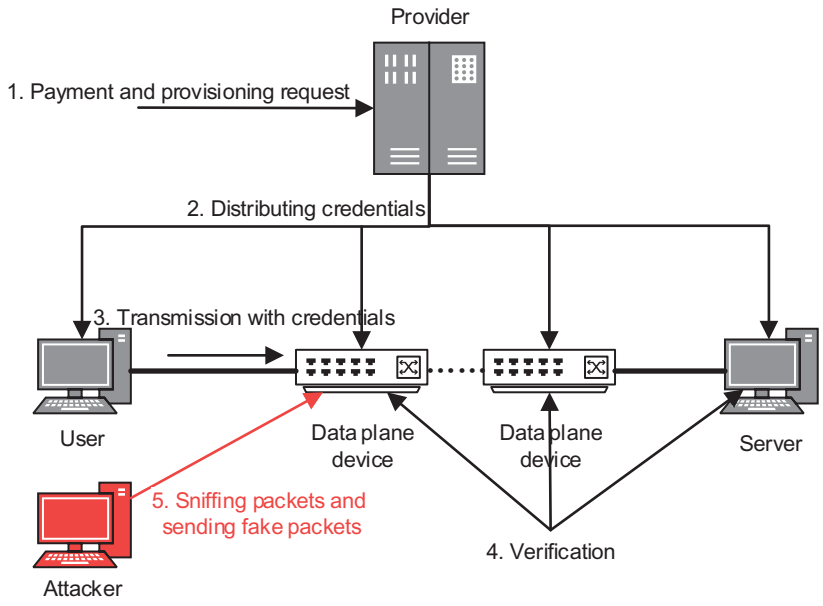


Figure 7.1. Interactions between a user, an attacker and a provider.

Fig. 7.1 illustrates the interactions between a provider, a user and an attacker. After the provisioning request is sent to the provider, it distributes credentials to the user and the data plane devices. The user attaches a credential to each packet for service, which is created via some method (e.g., cryptographic hash) with the secret information. The data plane devices can validate whether a user (or network traffic sent by the user) is authorized for access by validating the credentials. While potential attackers always try to extract the credential information from the legitimate packets and pretend to be the authorized users, the access control scheme should make sure only the user who sends packets with the valid credentials can access the services.

It is important to note that, the ChoiceNet is potentially a part of the future Internet with up to billions of users with billions of services. Traditionally, an effective checking mechanism may only require that authorized packets have some property that is hard for attackers to duplicate, while easy for legitimate users to create. However, when considering the common case that millions of packets on a link need

to be checked by a router simultaneously, it is critical to develop authentication methods that can be checked with low performance impact, while providing sufficient protection from access by unauthorized attackers.

This chapter proposes a novel design for data plane credential called OrthCredential (Orthogonal Credential), which enables access control and can be generated and verified at high data rates with low processing overhead and low storage requirements. The main idea of OrthCredential is that the user uses a sequence (credential) which is orthogonal to the verifier's sequences. And the verifier checks the *inner product* of the user's credential and the sequence on the verifier. The result of the inner product equals 0 means the credential is valid. These orthogonal sequences can be easily constructed from Hadamard matrices.

While designing and enriching access control protocols has received much attention, our focus is on decreasing the cost to satisfy data plane devices' computational capability while guaranteeing an acceptable level of security. The advantage of OrthCredential is in two aspects: 1) the computation of inner product of two binary sequences can be done by fast integer operations on CPU; 2) the verifier only needs to save a few basic orthogonal credentials and a sum of received valid credentials to check the validity of multiple received packets. The OrthCredential scheme has low verification time since inner product computations are much simpler than cryptographic operations. The main advantages of OrthCredential can be summarized as follows:

Low Verification Time We use an inner product computation to replace complicated cryptographic operations so that the verification time is significantly decreased (less than 350 clock cycles per packet if requiring less than 10^{-10} probability for an attacker to guess a valid credential). The speed comparison with existing access control scheme is shown in Section 7.6.2;

Fast Verification of Invalid Credential OrthCredential can detect invalid credentials even faster than valid ones. It can typically be done in a single inner product computation (less than 50 clock cycles). This eliminates the possibility of overwhelming the data path devices' computational resource by sending large volume of invalid packets.

Low Storage Requirement on Router The verifier (e.g., a router) only needs to save a small part of the orthogonal sequences and the sum of received valid credentials, which leads to very small storage consumption per flow. In Section 7.6, we will show that a space consumption of no more than 0.1 KB on the router can promise a random attack probability of less than 10^{-10} while preventing replay attacks simultaneously.

Small Packet Header OrthCredential header in a packet is small (no more than 28 clock bytes) and thus does not incur significant overhead in packets in the data plane of the network.

Section 7.3 provides a statement of the access control problem. Section 7.2 briefly discusses the conventional solutions and their shortcomings. Section 7.4.3 to 7.5 propose the new credential design, OrthCredential. Section 7.6 presents the evaluation results and a prototype implementation. Finally, Section 7.7 makes the conclusion.

7.2 Related Work

Most of the existing approaches of such packet checks are a poor match for the problem of access control to reserved resources while considering the "low cost" requirement, for two reasons. First, most of the authentication mechanisms are based on cryptographic schemes (e.g., digital signatures, HMAC [46] or UMAC [14] that uses a hash of the packet contents with a shared secret in it), or traceback schemes (trace packets to their source by reconstructing the path followed by packets). How-

ever, these schemes are difficult to use in practice due to expensive encryption computations or a complex process to identify packets, especially when considering the increasing number of users and services in the future networks. Computation is a notoriously scarce resource in routers, and thus an attacker could saturate a router's authentication checking capacity with bogus packets at some point along a flow's path, effectively denying service without forging valid credentials. An attacker that repeatedly floods the same packets can be denied by a modestly sized replay cache in the router. However, a difficult case is if the attacker can amass packets from many flows within a single validity window (e.g., ICING [55]). Besides, though some mechanisms (e.g., FPAC [18]) help mitigate the computation cost by encrypting a short random nonce rather than the payload, the router needs to maintain a receiving window for acceptable credentials considering that packets sent by users are hard to always arrive in strictly monotonic order. Therefore, to guarantee most of the valid packets to be checked by the router, additional memory consumption is required for each flow, which is possibly prohibitive for a router when supporting millions of flows simultaneously.

Traditional mechanisms provide a level of security that is very high and ensure that the probability of an attacker generating a legitimate packet is astronomically low. Though such high level of security is critical in some scenarios that the end-to-end acceptance of data that has been forged can be disastrous, it is overkill in the context of many network services (e.g., bandwidth or buffer space), where the goal is to guarantee low delay and loss probabilities to particular classes of packets. It is sufficient to limit the frequency of accepting a bogus packet to an acceptable level, which, in the contrary, opens the possibility of low performance impact in verifying time and space consumption of each router.

7.3 Preliminaries

Before the introduction of OrthCredential system, we begin the discussion with a description of security requirements, attacker capabilities and incapabilities. Then, we list the performance metrics considered in our system.

7.3.1 Security Requirements

This work only considers the problem of access control after a contract has been established between a user and a provider. In Fig. 7.1, we assume that the communications in steps 1-2 of the iterations are private and the service credentials (or credential seeds) can not be observed by a third party. This end-to-end security can be achieved by using existing protocols (e.g., TLS).

The access control system must have the following security properties to be considered as a solution to our problem:

Security Strength To provide a secure network infrastructure, it is crucial that credentials are only available to authorized users in the network. Therefore, credentials should be difficult to be guessed or faked. Brute force methods must yield a sufficiently low probability of success that the packets sent by authorized users is unaffected.

Verification without Trusting Hosts The verifier should prevent malicious hosts spoofing packets. Authenticity should be determined solely from the packet content and static per-flow information (e.g., public key or shared secret), and not from any other host information that changes per-packet.

Replay Prevention The authentication mechanism should include a method to detect reused credentials. This implies that the used credential information must be stored on a per-packet basis in a certain form. Given a fixed field size in the packet, this requirement implies that the number of packets that can be verified

is bounded; however, it should be large, so that resynchronization is required infrequently, even for high-data-rate links.

7.3.2 Attacker Capabilities

A malicious user, or an *attacker*, is trying to grant the access to some services (by sniffing packets from authorized users and extracting the credentials from the packets, or by some other methods). Fig. 7.1 illustrates such attacker's behavior. Such attack may interfere with authorized packets, making the user failing to get the guaranteed service. In some scenarios, this may bring lost revenue to the provider. Our assumptions about the attacker's capabilities can be summarized as the following:

- Ability to eavesdrop at some point along the path from the user to the verifier. Sniffing legitimate packets traveling along the path can be accomplished by breaking into an end system (non-router) connected to a shared-medium network somewhere along the path.

- Ability to extract the credential information within the packets and pretend to be the valid users and transmit the packets under correct formats. The valid credentials can be derived through long-term observation and analysis of the credentials in the authorized packets.

- Ability to send arbitrary packets or flood a particular link, router, or host to which it connects, e.g., Denial of Attacks (DoS). By breaking into and taking control of many end systems within the network, and making them to transmit bursts of packets addressed to some target simultaneously, the attacker can cause packets arriving a verifier at a rate close to the capacity of the channel.

Additionally, we constrain the capabilities of the attacker as follows:

- An attacker does not have access to the secret capabilities materials associated with the credential information between users and providers. As discussed ear-

lier, we assume that the delivery of the secret capabilities materials is through strict encryption.

- An attacker cannot stop the legitimate packets along the path (i.e., cannot drop the network traffic on a router);
- If an attacker transmits a modified copy of an authorized packet, the packet cannot arrive before the original one.

It is conceivable that the above assumptions might be violated, which OrthCredential system does not defend against. However, the limitations on the attacker's capabilities are necessary to keep the discussion of attack scenarios and security requirements within scope.

7.3.3 Performance Metrics

Our model of verifier processing implies some performance constraints on the authentication check. We assume that authentication can be pipelined with other operations, and consider only requirements that follow from the basic architecture of the verifier (e.g., router). The performance constraints on the verification process are summarized as the following:

Verification time the time spent on verification of an arriving packet is vital for the verifier. Since credentials need to be validated for every packet that arrives in a verifier, they must be verified with low computational requirements. Furthermore, different credential system may perform differently on the verifying time for an invalid credential and valid credential. The second one is also important for the verifier, since there may be floods of DoS attacks within the network.

Successful attack probability the probability of the attacker to successfully send a packet with random or duplicated credentials. This probability should be low enough to ensure the attacker can only successfully send one packet after a long

time trying. For example, if the probability is 10^{-9}, the attacker will need about an hour to send one fake packet on a 1M pkt/s link. We can consider this safe enough because in reality, the credentials may have changed many times during an hour.

Storage consumption the storage consumption for the credentials is also crucial for the verifiers, since there are maybe millions of users who have the access to the same service. Moreover, packets may arrive the verifier out-of-order. Verification mechanisms need to allow arbitrary packet order and save related information within a reserved window.

7.4 Overview of OrthCredential

In this section, we describe OrthCredential system in a high level, deferring the design details to Section 7.5.

7.4.1 Goals and Non Goals

OrthCredential is an authentication system intended for use in authorizing access control to reserved network resources to address the requirements in Section 7.3.1.

OrthCredential is different from conventional MACs that use cryptographic algorithms. It uses orthogonal sequences as credentials and determine validity by observing whether their inner product equals 0. The probability of counterfeiting it is higher than other crypt-schemes, but it is still low enough to be safe. The goal of OrthCredential is to achieve high performance and low cost for verification by eliminating some security guarantees that we have discussed above. It needs to be emphasized that OrthCredential is not designed as a replacement of conventional MAC algorithms. We believe many MAC schemes perform very well in end-to-end data transmission for high security demands, but it cannot meet the requirements in a general service-oriented network with billions of services and users.

Table 7.1. A full description of the relevant notations.

Variables	Description
$seed$	The secret information sent to the user and the verifier by the provider.
n	Length of each credential.
k	The index of generated Hadamard matrices ($k = 1,\ 2,\ 3...$).
$H(k)$	The k^{th} orthogonal matrix generated by user ($k = 1,\ 2,\ 3...$).
$h_i(k)$	The i^{th} row of $H(k)$ ($1 \leqslant i \leqslant n$).
$r_i(k)$	The random vector corresponding to $h_i(k)$.
$key(k)$	A secret key corresponding to $H(k)$ which is used to generate $r_i(k)$.
$c_i(k)$	The i^{th} credential and it satisfies $c_i(k) = h_i(k) + r_i(k)$
m	The number of the saved rows of $H(k)$ in the verifier ($1 \leqslant m \leqslant n$).
$H_m(k)$	A number of m rows in $H(k)$ that saved in the verifier (router).
$h_{m,i}(k)$	The i^{th} row of $H_m(k)$ in the verifier ($1 \leqslant i \leqslant m$).
c	The credential extracted from received packets by verifier.
h	The vector computed from c and it is expected to satisfy $h = c - r_i$ for some i.
$counter(k)$	The newest number of the credentials verified as valid, the value of *couter* resets when k is updated.
$sum(k)$	The sum vector of $v_i(k)$ which is saved in the verifier and $sum(k) = \sum_{i=1}^{n} v_i(k)$, the initial value of $sum(k) = 0$.
$sum\_bit[i]$	$sum\_bit[i]$ saves the i^{th} bit of each entry of $sum(k)$ ($1 \leqslant i \leqslant \log_2 n + 1$).

There are several functions that OrthCredential is not designed for: (i) OrthCredential does not guarantee the security and integrity of the payload in the packet during the packet's forwarding; (ii) OrthCredential does not guarantee the security of the path that the packet goes on. Actually, a secure path between two nodes can be also seemed as a service and OrthCredential can only provide the access to these secure paths which are set up by provides.

7.4.2 Deployment Scenario

The ideal deployment scenario for OrthCredential is at the network layer. Consider a path service with given bandwidth guarantees as an example: The providers would deploy OrthCredential routers at the ingress of their networks (i.e., edge routers). As shown in Fig. 7.1, after the user established a contract with a provider, the provider sets up the service and sends a secret generating seed to both the user and the edge

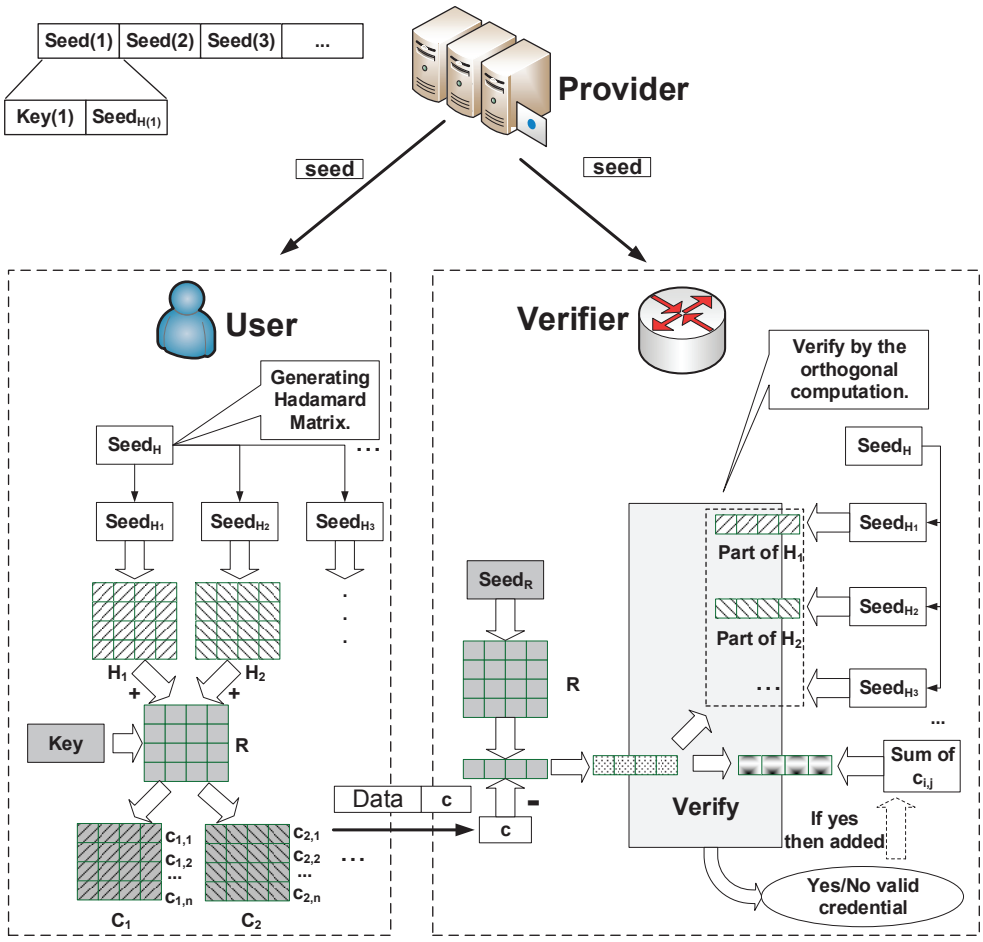


Figure 7.2. Credentials generation and verification in OrthCredential system.

routers along the transmission path. Each time the user uses the path service of the provider, the user sends packets with a credential generated by the secret seed. The edge router of the provider only forwards packets that contain a valid credential. Therefore, only an authorized user can access the bandwidth provided in the path service. The generation and verification mechanism of a credential is discussed further in Section 7.5.

7.4.3 Architecture and Components

OrthCredential is based on the technique of generating a series of sequences with mutual-orthogonal properties known to the user and the verifier. After a user has pursued some service from a provider, the provider will set up the service and send a secret generating seed to both the user and the devices along the transmission path. The seed contains the secret information of generating different $n \times n$ orthogonal matrices $H(k)$ (k is a numerical order). Besides, the seed also contains the corresponding keys (denoted as $key(k)$) for each $H(k)$. Each time when a packet is sent, the user chooses the first unused row vector $h_i(k)$ from $H(k)$ (i is the index of rows in $H(k)$) and generates a random vector $r_i(k)$ by using $key(k)$, then the user can get a credential by:

$$c_i(k) = h_i(k) + r_i(k), \qquad \text{where } k = 1,\ 2,\ 3, ...$$

A similar process is in the verifier: the verifier will generate $H_m(k)$ which includes a number of m rows of $H(k)$, and save it in memory. Once all rows from $H(k)$ are run out of, $H(k+1)$ will be generated from the seed. For every packet arriving at routers, the router extracts credential c and subtracts the corresponding $r_i(k)$ and results h. Then, h is used to compute the inner product with each $h_{m,i}(k)$, row vector of $H_m(k)$ ($\forall 1 \leqslant i \leqslant m$), to check its validity. Finally, to prevent replay attacks, h will be used to compute the inner product with $sum(k)$, the saved sum of the received valid orthogonal credentials, to check if c has been used or not, since a used credential will result in a non-zero inner product. If c is verified as valid, then the router will add h into $sum(k)$ to prevent replay attacks with credential c. Fig. 7.2 shows the entire process that OrthCredential works, where the details will be discussed in Section 7.5. The concise definitions of the notations we used in this paper can be found in Table 7.1.

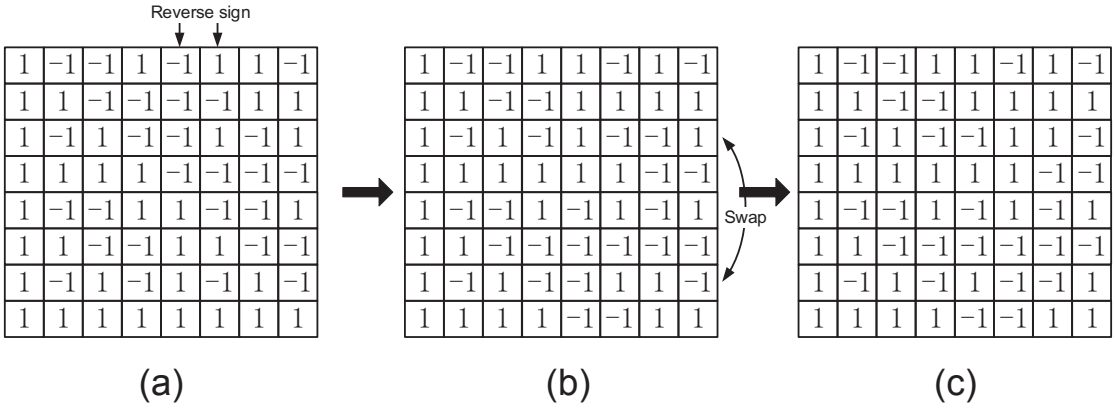


Figure 7.3. An illustration of Hadamard matrix and its property. (a) is an 8×8 Hadamard matrix, in (b) it reverses the sign of its fifth and sixth column and in (c) it swaps the third and seventh row. After all these transform operations, the matrices in (b) and (c) are still Hadamard matrices.

The security of OrthCredential is based on a very low probability of an attacker to obtain a credential satisfying that the results of its inner product with each $h_{m,i}(k)$ and $sum(k)$ equal 0 simultaneously. Even for a replay packet, the result of the inner product of h ($h = c - r_i(k)$) and $sum(k)$ equals $||h^2||$ but not 0 either.

In our OrthCredential system, these orthogonal matrices $H(k)$ are Hadamard matrices. In the next section, we will introduce the Hadamard matrix and its important properties which are used in OrthCredential.

7.4.4 Hadamard Matrix

7.4.4.1 Definition

A Hadamard matrix is an $n \times n$ matrix H containing entries from the set $\mathbf{Z_2} = \{-1, 1\}$, with the property of $HH^T = nI_n$.

This equation implies that all distinct rows or columns of a Hadamard matrix are linearly independent. Therefore, all rows or columns of a Hadamard matrix H are mutually orthogonal, i.e., have an inner product of 0. It needs to note that, in this chapter, all the computations are in the vector space, and each credential is viewed as

a vector. Let (v_1, v_2) denote the inner multiple computation, then vectors v_1 and v_2 are *orthogonal* if and only if $(v_1, v_2) = 0$. For instance, $v_1 = (-1, -1, 1)$ is orthogonal to $v_2 = (-1, 1, 0)$ because $(v_1, v_2) = (-1) \times (-1) + (-1) \times 1 + 1 \times 0 = 0$. In this chapter, we use an "entry" to denote an element in a vector, for instance, each entry in a row of a Hadamard matrix is 1 or -1.

An illustration of a Hadamard matrix is Fig. 7.3(a). In our OrthCredential system, we use 0 represent -1 in the Hadamard matrix. Then, the verification of checking whether the result of (c_1, c_2) equals 0 can be simply achieved by checking whether the number of '0's equals the number of '1's in the result of bitwise AND operation $c_1 \& c_2$.

7.4.4.2 Properties

It is proved that, an $n \times n$ Hadamard matrix exists for $n = 1$, $n = 2$, and $n = 4k$ for any $k \in \mathbb{N}$ [41]. We further introduce a basic, but very important, property of Hadamard matrix:

Theorem 1 *There are several operations on Hadamard matrices that preserve its property: (i) Swapping rows, and changing the sign of rows; (ii) Swapping columns, and changing the sign of columns; (iii) Transposition.*

An illustration of the property is shown in Fig. 7.3: Fig. 7.3(a) illustrates an 8×8 Hadamard matrix; in Fig. 7.3(b), we change the sign of its fifth and sixth column; in Fig. 7.3(c), we further swap the third and seventh row. After these operations, the transformed matrices are still Hadamard matrices. Strictly speaking, two Hadamard matrices H, H' are said to be different if H' cannot be produced from H by these transform operations (i)-(iii). Therefore, there is only one 2×2 Hadamard matrix, though it has eight different expressions. Reference [41] has enumerated the number of inequivalent classes of Hadamard matrices from $n = 2$ to $n = 32$, for $n = 32$, there are 13,707,126 matrices. When n is bigger than 32, the numbers of different Hadamard

matrices are even much larger. This ensures generation of a random Hadamard matrix that the attacker cannot guess easily, which also limit the possibility of forging it. Even if the attacker can guess one credential brute-forcely, it will expire when the next Hadamard matrix is generated.

7.5 Design details of OrthCredential

This section details OrthCredential's design, which aims to meet the requirements stated in Section 7.3.

Table 7.1 describes the notation what we use throughout our design discussion and our pseudo code, and Fig. 7.4 shows the OrthCredential header format. The header includes two types of information for each user. The first is the user's information, the provider will assign a unique ID for each user who has established a contract with. The verifier will turn to the corresponding verifying materials by checking this ID information. The second is the information used for verification: $\{n, i, k, c_i(k)\}$. The length of $c_i(k)$ (i.e., n) in our current system is 32, 64 or 128 bits. The verifier will tell the difference by checking n or the "header length" part.

In total, the overhead of the OrthCredential header in a packet is 16 to 28 bytes. The credentials part could be 32, 64 or 128 bits. *header length* and *credential length* are in bytes. The original IP protocol is encapsulated in the OrthCredential payload.

7.5.1 Creating Credentials

As described in Section 7.4.3, a credential $c_i(k)$ is given by $c_i(k) = h_i(k) + r_i(k)$. We generate $r_i(k)$ by using a random() function with the seed $(key(k), k, i)$. Each $r_i(k)$ is different due to different sets of $(key(k), k, i)$. The reason why we add $r_i(k)$ with $h_i(k)$ in OrthCredential system is, if $h_i(k)$ can be easily observed by an attacker, the attacker can get a valid credential by generating lots of sequences that are orthogonal to $h_i(k)$. Therefore, $r_i(k)$ can help decrease the probability for attackers to forge a

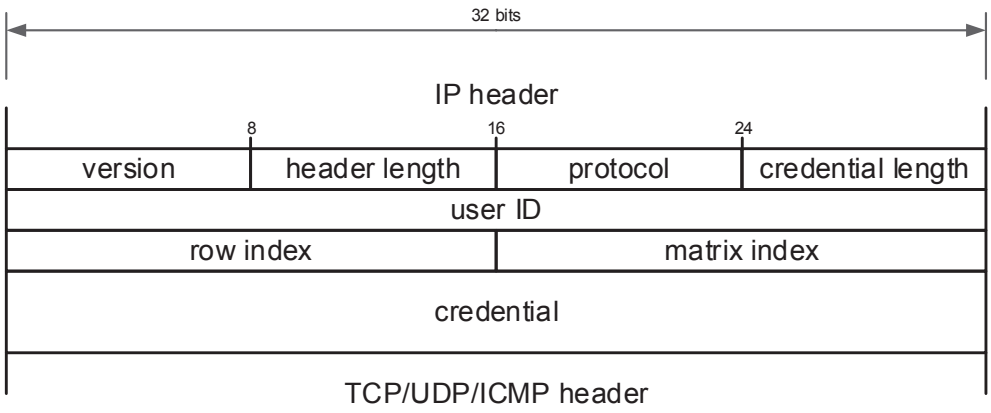


Figure 7.4. OrthCredential header.

valid credential. We next describe how to construct a Hadamard matrix and thus get $h_i(k)$.

People have derived many construction methods to generate a Hadamard matrix for a given n [4]. The Hadamard construction method in our system is a simple, but efficient one - *Kronecker Product Construction*: if S, T are matrices, their Kronecker Product $S \otimes T$ is the matrix U constructed by replacing each $S_{i,j}$ in S by $S_{i,j}T$. It can be proved that the Kronecker Product $H_n \otimes H_m$ is a Hadamard matrix of order nm if H_n, H_m are Hadamard matrices of orders n and m. This implies that we can get a $2n \otimes 2n$ Hadamard matrix by the product of an $n \times n$ Hadamard matrix with the basic 2×2 Hadamard matrix of

$$\begin{pmatrix} +1 & +1 \\ +1 & -1 \end{pmatrix}.$$

Before doing Kronecker Product operation, the original $n \times n$ and the basic 2×2 Hadamard matrices will take several transform operations in Theorem 1 first, respectively. Then, the generated Hadamard matrix will also take several transform operations. All these transform operations are described in *seed* sent by the provider. For the original $n \times n$ Hadamard matrix, we use a Walsh-Hadamard transform to get it.

Algorithm 5 Generating a credential

```
if i%(n+1) = 0 then
    k← k+1, i ← 1
    key(k) ← seed(k).key
    H_1 ← Hadamard_Transform(n/2)
    H_2 ← Hadamard_Transform(2)
    H_1, H_2 ← Transform_Operation(H_1, H_2, seed(k))
    H(k) ← Kronecker_Product(H_1, H_2)
    H(k) ← Transform_Operation(H(k), seed(k))
h_i(k) ← the i^th row of H(k)
r_i(k) ← Random(key(k), k, i)
c_i(k) ← h_i(k) + r_i(k)
return (c_i(k), k, i)
```

For instance, in Fig. 7.3(a), if we let the bottom row as 0^{th} row and the leftmost column as the 0^{th} column, then its $(i,j)^{\text{th}}$ entry $H_{i,j}$ can be written as $H_{i,j} = (-1)^{\sum_{p=1}^{n}(i_p \cdot j_p)}$, where $i = \sum_{p=1}^{n} i_p 2^p$ and $j = \sum_{p=1}^{n} j_p 2^p$. The Walsh-Hadamard transform is easy to implement, since the computation of $\sum_{p=1}^{n}(i_p \cdot j_p)$ can be simply achieved by checking the number of '1's in the result of the bitwise operation i AND j. In real implementations, the verifier will save a number of basic Hadamard matrices beforehand, since they may be used for each flow's verification. In Section 7.6.1, we will show that this generation method can guarantee a very low repeating probability.

Each time when a packet is sent, the user chooses an unused row vector $c_i(k)$ as the credential and places it together with its index k, i in the packet. Once all rows from $H(k)$ are run out of, $H(k+1)$ will be generated from the secret seed. We design the maximum k is 100, and the user will ask for a new seed from the provider when k is out. Each n-bit credential $c_i(k)$ ($c_i(k) = h_i(k) + r_i(k)$) satisfies the following equations:

$$\begin{cases} \big(h_i(k),\ h_j(k)\big) = 0, & \text{when } i \neq j, \\ \big(h_i(k),\ h_j(k)\big) \neq 0, & \text{when } i = j. \end{cases} \tag{7.1}$$

Algorithm 6 Verifying a credential

$r_i(k) \leftarrow \text{Random}(key(k), k, i)$
$h \leftarrow c - r_i(k)$

▷ Step 1: Verify $h = h_{m,i}(k)$ or $(h, h_{m,i}(k)) = 0$
for $1 \leqslant i \leqslant m$ **do**
    if $h = h_{m,i}(k)$ **then**
        break
    else
        $result \leftarrow$ number of 1s in $(h \oplus h_{m,i}(k))$
        if $result \neq n/2$ **then**
            return InValid

▷ Step 2: Verify $(h, sum(k)) = 0$
$result \leftarrow 0$
$number\_h_0 \leftarrow$ number of 0s in h
$number\_h_1 \leftarrow$ number of 1s in h
for $1 \leqslant i \leqslant \log_2 n + 1$ **do**
    $number_0 \leftarrow$ number of '0's in $(\bar{h}$ & $sum\_bit[i])$
    $number_1 \leftarrow$ number of '1's in $(h$ & $sum\_bit[i])$
    $result \leftarrow result + 2^{i-1}(number_1 - number_0)$
if $result \neq couter(k) \cdot (number\_h_1 - number\_h_0)$ **then**
    return InValid

▷ Step 3: Update $sum(k)$
for $2 \leqslant i \leqslant \log_2 n + 1$ **do**
    if $i = 2$ **then**
        $carry \leftarrow h$
    else
        $temp \leftarrow carry \oplus sum\_bit[i]$
        $carry \leftarrow carry$ & $sum\_bit[i]$
        $sum\_bit[i] \leftarrow temp$
$couter(k) \leftarrow couter(k) + 1$;
return Valid

7.5.2 Verifying Credentials

The verifier only generates and saves a number of m rows of $H(k)$ (i.e., $H_m(k)$), not the whole matrix. In real implementations, m is usually $10\% - 30\%$ of n. The value of m depends on the security requirements of the service, and if the verifier saves the whole $H(k)$ (i.e., $m = n$) then it can achieve the best protection against

forgery attacks. In Section 7.6, we will show that even a small m can also provide a very good protection for the access to the service.

For each packet arriving at provider devices, the verifier extracts credential c, then subtracts the corresponding row r_i and finally gets h. Then, h is used to compare with or compute the inner product with $h_{m,i}(k)$ to verify the validity of c. To avoid replay attacks, h will also be used to compute the inner product with $sum(k)$ to check if c has been used or not, since a used credential will result in a non-zero inner product. A received credential c is verified as valid when the following conditions are satisfied:

$$\begin{aligned} &1.\quad \exists\ i\ (1 \leqslant i \leqslant m),\ h = h_{m,i}(k)\ \ \text{or} \\ &\qquad \forall\ i\ (1 \leqslant i \leqslant m),\ \big(h,\ h_{m,i}(k)\big) = 0; \end{aligned} \tag{7.2}$$

$$2.\quad \big(h,\ sum(k)\big) = 0. \tag{7.3}$$

If h satisfies the above equations (i.e., c is a valid credential), the local variable $couter(k)$ will plus 1. Here, $couter(k)$ implies that h is the $(couter(k))^{\mathbf{th}}$ valid credential received by the verifier. In order to protect against a replay attack with credential c, the verifier will add h with $sum(k)$ and save the result as a new $sum(k)$. It must be emphasized that: (i) the addition here is between two vectors not two numbers; (ii) though we use 0 represent -1 in h during verification (as described in Section 7.4.4), but during this addition operation, h should be the original vector with entries 1 or -1. Thus, the value of each entry of $sum(k)$ is an integer in the range of $[-n, n]$. We next explain the verification details during real implementations:

Step 1: Verify Equation (7.2). The operations of verifying whether the result of $(h,\ h_{m,i}(k))$ equals 0 can be simply achieved, which is through checking whether the number of '0's equals the number of '1's in the result of bitwise operation h AND $h_{m,i}(k)$.

Step 2: Verify Equation (7.3). The cost of the verification process mainly depends on the operations between h and $sum(k)$ when m is small. An intuitive solution would be represent $sum(k)$ as an array with n integers which are in the range of $[-n, n]$, and use n iterations to calculate the inner product. However, we can use the following schemes to decrease the cost to $O(log_2 n)$:

- Remove negatives in $sum(k)$: during the addition operation between h and $sum(k)$, we update $sum(k)$ with the result of $sum(k) + h + e_n$, where $e_n = (111...11)_n$. Therefore, the value range of each entry in $sum(k)$ becomes $[0, 2n]$.

- Recompose $sum(k)$: we use a set of $\{sum\_bit[i]\}$ to represent $sum(k)$, where $1 \leqslant i \leqslant \log_2 n + 1$. Each $sum\_bit[i]$ is an n-bit local variable in the verifier and the j^{th} bit of $sum\_bit[i]$ denotes the i^{th} bit of the j^{th} entry of $sum(k)$. Fig. 7.5 is an example which shows the relations between $sum(k)$ and $sum\_bit[i]$ when $n = 32$. From the above discussion, we know that, each time $sum(k)$ updates by adding the result of $h + e_n$ whose each entry is 2 or 0, therefore each entry of $sum\_bit[1]$ is always 0. In real implementations, the verifier does not save $sum\_bit[1]$ to decrease the storage consumption.

By the above schemes, Equation (7.3) could be written as:

$$couter(k) \cdot (h,\ e_n) = \sum_{i=1}^{log_2 n+1} 2^{i-1} \cdot (h,\ sum\_bit[i]).$$

The computations of inner product in the above equation can be realized by simple bitwise operations as the same as computing $(h,\ h_{m,i}(k))$. Therefore, we reduce the processing time of verifying Equation (7.3) from $O(n)$ to $O(\log_2 n)$.

Step 3: Update sum(k). An intuitive solution of updating $sum(k)$ with the result of $sum(k) + h$ would also need n iterations to accumulate the sum of each entry in $sum(k)$ and the corresponding entry (1 or -1) in h. However, with the

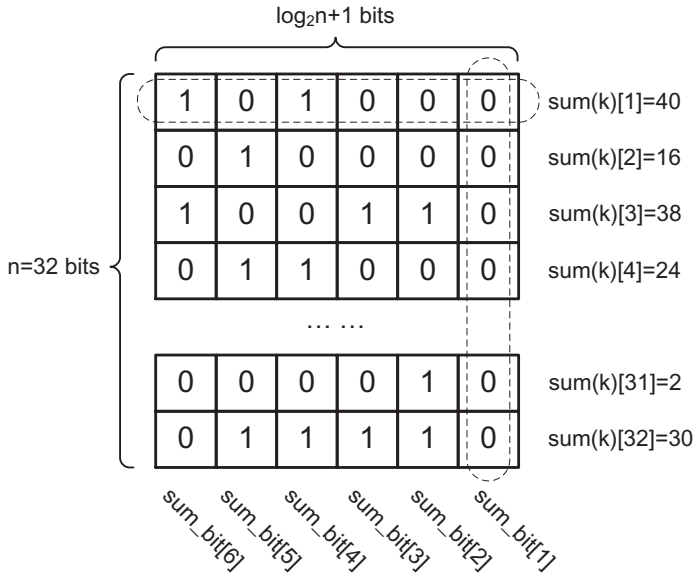


Figure 7.5. Bitwise calculation of $sum(k)$.

above representation of sum_bits, we can also decease the processing time of updating $sum(k)$ from $O(n)$ to $O(\log_2 n)$. As described above, we actually update $sum(k)$ with $sum(k) + h + e_n$, where each entry of the result of $h + e_n$ is 2 or 0. Therefore, in real implementations, we will let each entry of $sum\_bit[2]$ plus 1 if the corresponding entry of this result is 2. If the addition on an entry of $sum\_bit[2]$ produces a carry, then reset this entry and transmit a carry to the corresponding entry of $sum\_bit[3]$, and so forth.

In addition to the verifying operations in Step 1, the whole verifying time (count in clock cycles) is in a scale of $O(m + \log_2 n)$. In Section 7.6, we will present the running time under real implementations. The pseudo codes of these algorithms for generation and verification of a credential are shown in Algorithm 5-6.

7.5.3 Attacks

The security of OrthCredential is based on a very low probability of an attacker to obtain credentials that satisfies Equations (7.2) and (7.3) simultaneously. We briefly analyze how OrthCredential counters various threats.

An attacker may try to obtain credentials by brute force. We use a part of n mutually orthogonal vectors (i.e., $H_m(k)$) to make these "grope around" attacks impossible. Though the probability that a random vector is (with entries "-1" or "1") orthogonal to $h_{m,i}(k)$ cannot be concluded theoretically, we verify the very low probability for random successful attack by experiments. In our implementation of OrthCredential system, this random attack successful probability would be even lower because a valid credential also has to be orthogonal with $sum(k)$. The results of the experiments are shown in Section 7.6.

An attacker may try to obtain credentials by replaying the valid credentials sent by the user. We use a $sum(k)$ to perfectly prevent this kind of attack. It works since once h is verified to be valid, then it will be added into $sum(k)$. Thus, we have $(h, sum(k)) = (h, h) = |h|^2 \neq 0$. If h is expired for k, h will be also verified to be invalid due to a very low probability that h is orthogonal to each new $h_{m,i}(k)$.

An attacker may try to obtain credentials by generating Hadamard matrices. OrthCredential provides double protection against this attack. First, as discussed in Section 7.5.1, the properties of Hadamard matrices guarantee a very low probability that an attacker generates the same matrix as the user's. Besides, we use a dynamic random vector $r_i(k)$ to "encrypt" each row of the generated Hadamard matrix. If the attacker obtains the information of some $r_i(k)$ of a user, the attacker can derive $h_i(k)$ by observing the packets sent by the user, it is still very far away from breaking through the verifier. This is because $h_i(k)$ cannot be used directly, the attacker needs to generate vectors that are orthogonal to $h_i(k)$. However, it is still a very low probability that these generated vectors are orthogonal with all the $h_{m,i}(k)$ in the verifier.

An attacker may eavesdrop the communications between the user and the provider. Each time when the credentials generated from *seed* are run out of, the user will ask its service provider for a new credential seed. In OrthCredential

system, any end-to-end traffic between the user and provider will be encrypted using existing protocols (e.g., TLS/SSL). There is no way for an attacker to obtain secret information by this behavior.

There are also other attacks described in Section 7.3.2 that are out of our assumption of the attackers' incapabilities and OrthCredential cannot defend against. Actually, it's a future work we will consider in OrthCredential.

7.6 Evaluation

In this section, we evaluate the performance of security, verification time and storage consumptions in OrthCredential. We have implemented the algorithm in C++. We use a PC with an Intel Core2 Quad CPU Q9400 running at 2.66GHz to test the algorithm's performance. The operating system is Ubuntu 14.04 64-bit with kernel version 3.13.0-24 and gcc version 4.8.2. Both the credential generation and verification codes are compiled in one program with gcc -O3 optimize level. We do not use platform specific instructions or assembly codes. The time consumed by each step is measured by CPU clock cycles.

7.6.1 Security

We first run simulations to ensure a very low probability of generating the same Hadamard matrix by the construction method which is described in Section 7.5.1. During the generating process, we take 3 basic transform operations each time (permutation, changing signs or transposition). We generate $100,000$ Hadamard matrices for $n = 32$, 64 and 128 and find the number of the same generated Hadamard matrices among them. The repeatability result is that, when $n = 32$, the repeatability is 0.18% and it is nearly 0 when $n = 64$ or 128. Considering that a dynamic random vector $r_i(k)$ will also "encrypt" each row of the Hadamard matrix, we believe that it is impossible for an attacker to guess a valid credential.

We then simulate a scenario that an attacker sends random credentials to brute force the verification process. As discussed in Section 7.5.2, the first step of verifying a valid credential is to determine whether this credential is orthogonal with each $h_{m,i}(k)$ while a random credential is hard to achieve this. Fig. 7.6 shows the success probability of such random attack. It can be observed that, when the credential length (i.e., n) is 128, $m > 10$ can guarantee the breakthrough probability less than 10^{-9}, which can be considered safe enough. For some services with low-security requirements, we consider 32-bit or 64-bit credentials can also be used with a proper chosen m.

We also simulate a scenario that an attacker sends replay packets, the result is that all these replay packets are discarded by the verifier no matter what n and m are. There are also other attacks in reality that we cannot simulate, the discussions can be found in Section 7.5.3.

7.6.2 Verification Time

In OrthCredential system, we use computations of inner product to replace complicated cryptographic operations so that the verification time is significantly decreased, most of the possible computations are simplified to use the basic bitwise operations.

We have tested the different credential length (i.e., n) and different number of rows of a Hadamard matrix stored on the verifier (i.e., m). Fig. 7.8(a) shows the time needed to verify a valid credential. When m is relatively smaller than n, the time increases almost linearly with m. That is because, in this case, the probability that h (calculated from the received credential c) equals a saved $h_{m,i}(k)$ is relatively small, thus a valid user's credential has to be calculated against all the m rows of $H_m(k)$ to get verified. When m approaches n, the verifier has a larger probability of saving an $h_{m,i}(k)$ that equals h, which makes the verification process jump to the sum verification, therefore the curve becomes gradual.

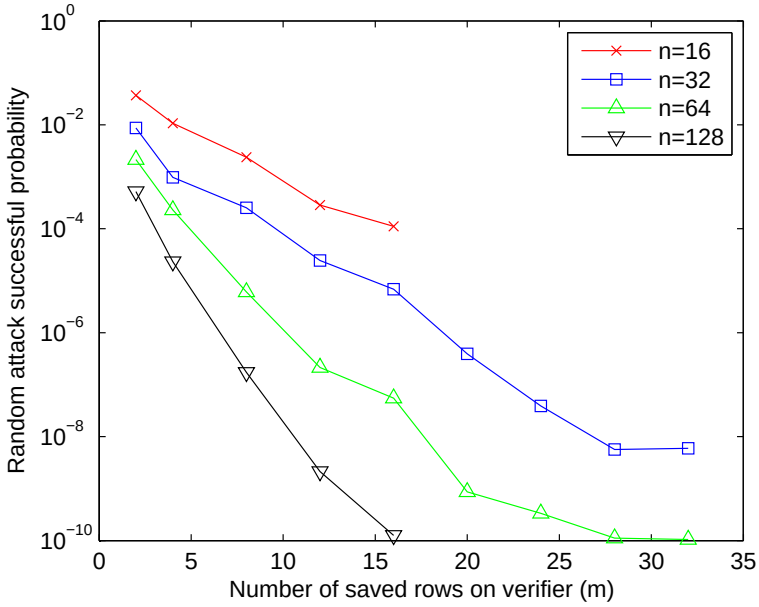


Figure 7.6. Successful attack probability of random generated credentials.

As discussed in Section 7.5.2, it requires three steps to verify a valid credential, which runs in a scale of $O(m + \log_2 n)$ time. The main cost depends on Step 1 and Step 2 since they both need an operation to count the number of '1's or '0's in a vector (i.e., POPCOUNT operation), while Step 3 only does the basic AND or XOR bitwise operations. 16-bit POPCOUNT is done by looking up twice in a 8-bit lookup table, while 32 and 64-bit POPCOUNT uses a variable-precision SWAR algorithm introduced in [2]. Because the lookup table method is faster, we can see the time for $n = 16$ is almost half of the time when $n = 32$. 32-bit and 64-bit credential cost almost the same time, because for a 64-bit CPU, operating a 64-bit integer is as fast as a 32-bit integer. 128-bit credential needs twice the time of 64-bit credential, because the CPU cannot do 128-bit integer calculation natively, all the calculations must be performed as two 64-bit operations. It is worth noting that although some new Intel x86 CPUs have SSSE3 and SSE4.2 instructions that can do fast POPCOUNT [37], we do not use it because OrthCredential system is platform dependent. If we use it, the speed can be further improved.

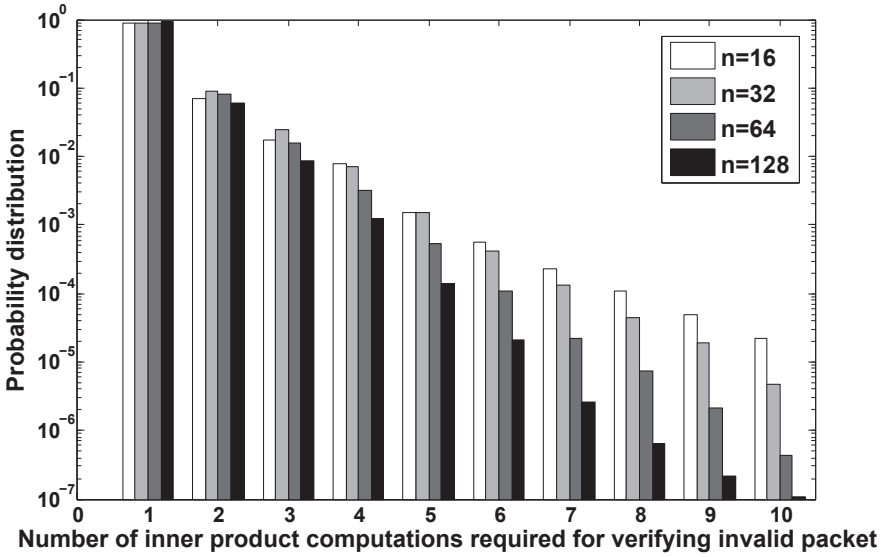


Figure 7.7. Number of inner product computations to identify invalid packets.

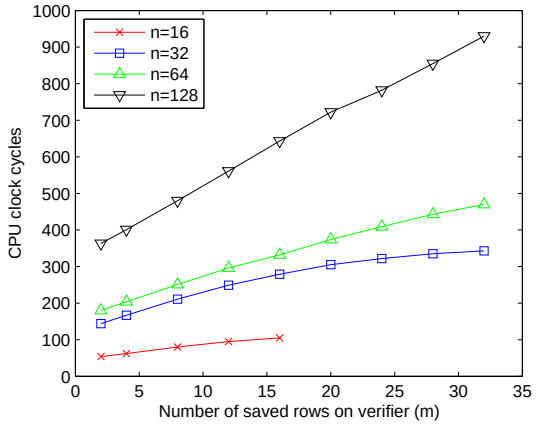
Fig. 7.8(b) shows the average time needed to verify a random attack credential. Theoretically, the verifier can discard the invalid credential within the first few inner product computations in Step 1 due to the low random attack successful probability. Therefore, the time is much less compared to verify a valid credential, and it is also nearly irrelevant to m. When $n = 16$, it takes a longer time to verify a random credential, this is due to the higher random attack successful probability, which will lead to a longer verification process. Fig. 7.7 shows the probability distribution of the number of the inner product computations required for a random credential before it is discarded by the verifier. When the number is larger than 10, the probability is nearly zero.

Fig. 7.8(c) shows that the replay attack credentials take almost the same verification time as a valid credential, albeit none of them will be verified to be a valid one. This is because a replay credential will not be identified until completing Step 2 – inner product computation with $sum(k)$. While Step 3 only does serval basic AND or XOR bitwise operations that require less than 10 clock cycles to proceed, the error of experiments can cover up this slight difference between Fig. 7.8(a) and Fig. 7.8(c).

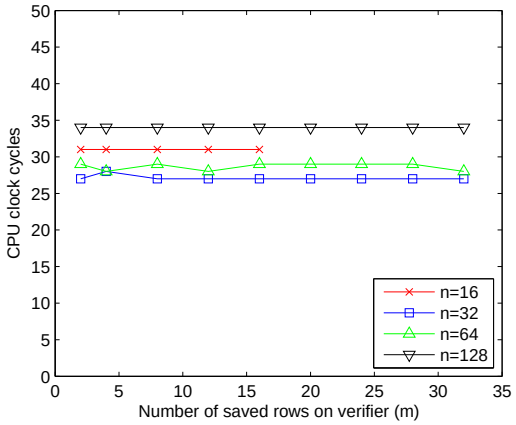
Conventional cryptographic schemes will take much more verification time. Table 7.2 shows the experiment results of the average per-packet verification cost for some classical conventional cryptographic algorithms for 500-byte packets (an average packet length in Internet). These algorithms are also implemented in C++ without platform specific instructions and assembly codes. To make more comparisons, we also list the cost of some path verification mechanisms (ICING [55], TVA [75] and DPCP [68]) in the table. By comparing Table 7.2 with Fig. 7.8, we can see the enormous advantages on the verification speed, especially considering the case that a DOS attacker floods a path with lots of random attack packets (the verification time of the schemes in Table 7.2 does not change for random attack packets).

7.6.3 Storage Consumption

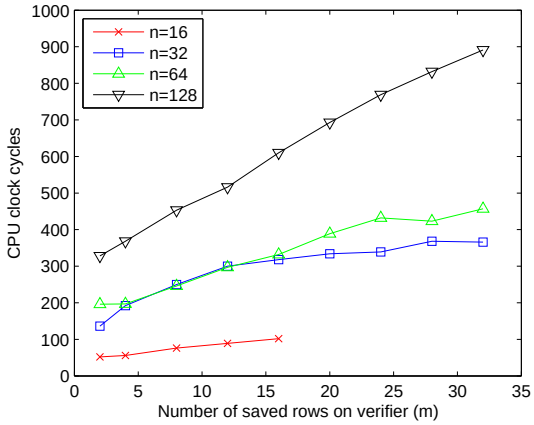
As illustrated in Fig. 7.4, the total overhead of OrthCredential header in a packet is 16 to 28 bytes. In OrthCredential, the storage consumption of the verifier for each user depends on two parts: a number of m rows of an $n \times n$ Hadamard matrix and a set of $\{sum\_bit[i]\}$ where $2 \leqslant i \leqslant \log_2 n + 1$. Thus, the whole storage consumption in the verifier is $mn + n \log_2 n$ bits. It must be noted that this storage consumption is under a consideration of preventing replay packets. Many authentication schemes (e.g., HMAC/UMAC, ICING) cannot provide anti-replay protection naturally and have to keep a window to save received authorized credentials. In this point of view, our OrthCredential does not need to save all the newest authenticated credentials, but uses a sum of the received authenticated credentials instead, which decreases the storage consumption efficiently. For instance, if OrthCredential uses 64-bit credentials with 5 rows of a 64×64 Hadamard matrix to implement verification, then the router only requires a space of 80 bytes to prevent all possible replay packets while other conventional cryptographic schemes may need a space of 512 bytes to achieve it.



(a) A valid credential.



(b) A random attack credential.



(c) A replay attack credential.

Figure 7.8. Verification time of different types of packets. (measured in CPU clock cycles)

Table 7.2. Average verification costs of different access control schemes. (assuming 500-byte packets).

Algorithm	**Cycles/Packet**
HMAC (MD5)	5,335
HMAC (SHA-1)	8,931
AES/CTR (128 bit key)	7,277
DMAC (AES)	12,223
ICING (x-hop)	$2{,}080x + 19{,}520$
TVA (x-hop)	$3{,}264x$
DPCP (512-bit credential)	34,780

7.7 Conclusions

This chapter introduces a novel credential design to provide efficient access control in the data plane of a network. A new credential design, OrthCredential, is presented to solve the problem of protecting reserved services with very low overhead in terms of verification time and memory consumption, while guaranteeing good security performance. The prototype implementation has shown a small credential header (e.g., 20 bytes) can be checked in less than 300 processor cycles and require less than 800 bits of memory per flow on a router. We believe that OrthCredential will be an important part of the ChoiceNet.

CHAPTER 8

DEPLOYMENT ON GENI

GENI (Global Environment for Network Innovations) [13] is a testbed for networking and distributed systems research. It allows experimenters to allocate resources such as virtual machines and links all over the world. It is well suited for networking experiments at scale.

To demonstrate the ChoiceNet implementation described in this work, it is deployed on ExoGENI [8]. In this implementation, we use pox[1] as the SDN controller, and install OpenvSwitch[2] on GENI nodes to use them as SDN switches. The SDN protocol is OpenFlow 1.0. The hosts are running 64-bit Ubuntu 14.04.

The test topology is shown as Figure 8.1, which is similar to Figure 6.1. There are two ASes, one is located at GENI Project Office (GPO) in Massachusetts, the other is at University of Florida (UFL). The clients are in the UFL domain, and the server is in the GPO domain. There are three inter-AS links, each has a different bandwidth, latency and price. The intra-AS links also have different metrics. The bandwidth is hard-coded in the topology when it is created. The latency is measured from ping tests after the topology is created. The price is set according to the bandwidth and latency value.

The inter-AS and intra-AS links form 13 paths between the clients and the server. But only 3 paths are Pareto-optimal. Therefore, the marketplace should return the following three paths to users:

---

[1] `http://www.noxrepo.org/pox/about-pox/`

[2] `http://openvswitch.org/`

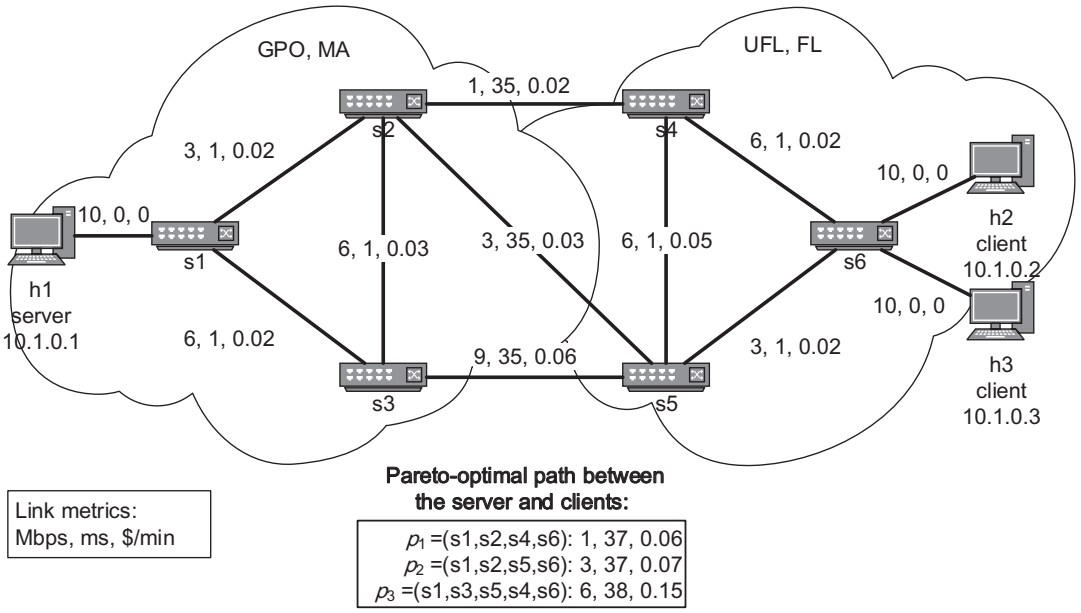


Figure 8.1. GENI test topology.

1. p_1=(s1, s2, s4, s6), bandwidth: 1 Mbps; latency: 37 ms; price: \$0.06/min;

2. p_2=(s1, s2, s5, s6), bandwidth: 3 Mbps; latency: 37 ms; price: \$0.07/min;

3. p_3=(s1, s2, s5, s4, s6), bandwidth: 6 Mbps; latency: 38 ms; price: \$0.15/min.

In this experiment setup, when the ChoiceNet App needs to make choices from multiple paths, it reads the preference from a configuration file, then automatically submit the choice to the marketplace. This eliminate the delay of human selection. There are two pre-defined preferences: `lowest_price` and `highest_bandwidth`. The ChoiceNet App is capable of making more flexible decisions such as "highest bandwidth-price ratio when price <\$0.1/min" or "lowest price when bandwidth >3 Mbps", but we do not show these complicated preferences in this experiment.

In the following sections, we will show how the user preference can affect the final user experience. We will also show the quantitative result collected from the experiments.

8.1 Video Streaming Test

To demonstrate that the hybrid architecture described in Section 5.3 is feasible, we choose Youtube[3] as the example application. To be specific, $h1$ is configured as an HTTP proxy, $h2$ uses VLC player[4] to stream Youtube video with http proxy configured as $h1$. So the video data will be transmitted to $h1$ first, then transmitted to $h2$ through one of the three Pareto-optimal paths described above.

In this experiment, the video will not be displayed on the screen, because we are controlling the hosts over terminals remotely. But the quality of video can be reffered from its bitrate. The higher the bitrate is, the better video quality the user will experience. Youtube has an adaptive approach for serving videos: it will measure the link speed first, then serves videos with similar bitrate.

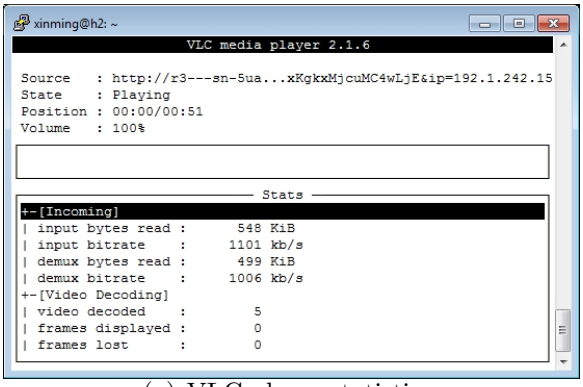
Figure 8.2 shows the streaming result. The user preference is `lowest_price`. The output of ChoiceNet App shows the 1 Mbps path is selected (p_1 in Figure 8.1). The demux bitrate shown in Figure 8.2(a) is around 1 Mbps, meaning the use plane has correctly provisioned the path. Figure 8.3 shows the streaming result when the user preference is set to `highest_bandwidth`. The 6 Mbps path is selected (p_3 in Figure 8.1). The demux bitrate is around 4.5 Mbps (it is not reaching 6 Mbps because of the limitation of the connection between Youtube and $h1$).

This test scenario demonstrates that this implementation is capable of performing a full service listing and service transaction, thus provides more choices to the user.

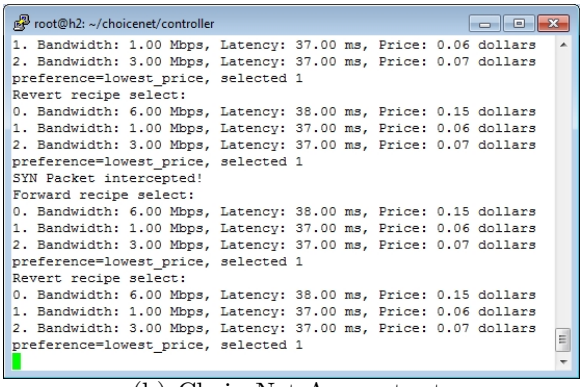
---

[3] `http://www.youtube.com/`

[4] `http://www.videolan.org/`

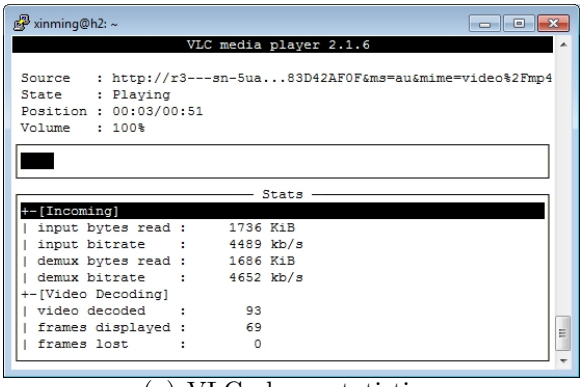


(a) VLC player statistics.

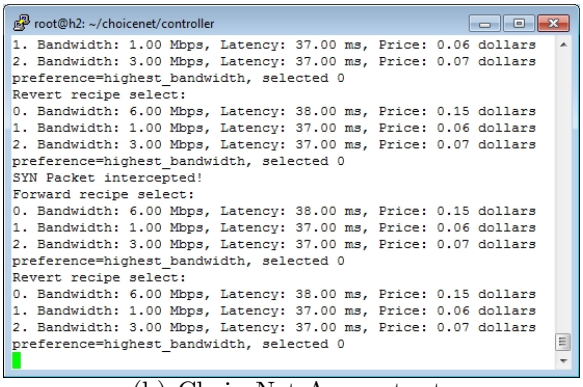


(b) ChoiceNet App output.

Figure 8.2. Streaming result for lowest price.



(a) VLC player statistics.



(b) ChoiceNet App output.

Figure 8.3. Streaming result for highest bandwidth.

It also shows that ChoiceNet can be used to get services from the public Internet, even if ChoiceNet is only deployed on a local area network.

We have done another experiment to illustrate the "Vote With Your Wallet" principle described in Section 1.2. By changing the user preference while the video is playing, the user can observe the video quality changing with user's selection. To achieve this, we use a new video streaming technique named DASH (Dynamic Adaptive Streaming over HTTP) [62] instead of Youtube. It is an adaptive bitrate streaming technique, which will change the video bitrate according to the download speed of the previous video segments. The video server is hosted on $h1$. The

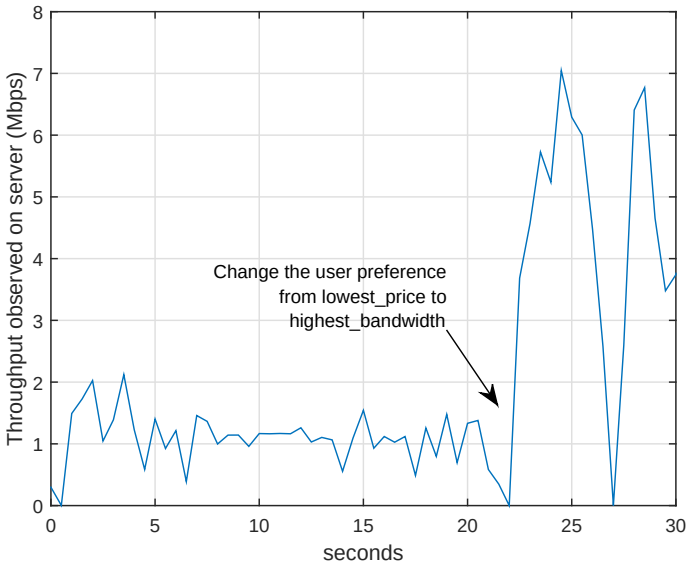


Figure 8.4. Demonstration of "Vote With Your Wallet."

video player is on $h2$. The initial user preference is `lowest_price`. We change it to `highest_bandwidth` after a few seconds.

Figure 8.4 shows the throughput observed on the server. In the beginning, the throughput is about 1 Mbps [5]. After the user changes the preference, the new video segments are routed through the 6 Mbps path. The server detects the increased throughput, thus increases the bitrate of the upcoming video segments. This can be observed at the 22nd second, when the throughput bursts to about 6 Mpbs. This experiment illustrates the scenario where the user is not satisfied with the current service, thus shift to a more expensive service for better quality. Such paradigm is critical for promoting the providers' competition.

[5]Sometimes the burst throughput exceeds the bandwidth limit, because the limit is implemented by GENI's QoS function instead of a physical limit, so it can be broken through during burst transfer

8.2 iperf Test

The way ChoiceNet can help network evolution is through long-term economical interactions. Though it is difficult to show this in a short-term demonstration, we can set up multiple economical entities and observe their economical interactions.

In this experiment, we set up two customers, both trying to use the highest bandwidth path. In a network without economy plane, they may end up with using the same path and competing with each other. In ChoiceNet, after the provider reserves the highest bandwidth path for the first customer, it becomes unavailable to the second customer. The second customer will get the second-highest bandwidth path. This both increases the utilization of the entire network, and avoids possible congestions.

In order to get more accurate throughput data, we use iperf[6] to generate the test TCP traffic. $h1$ is the server, $h2$ and $h3$ are clients, and both clients use `highest_bandwidth` preference. Figure 8.5 shows the egress throughput observed on $h1$. In the beginning, $h2$ requests for the highest bandwidth path, the 6 Mbps path is chosen and provisioned (p_3 in Figure 8.1). Since p_3 reserves all the bandwidth on link s4-s5 and s4-s6, the second client only have one option: $p2$. Therefore, when $h3$ requests for the highest bandwidth path, it can only get the 3 Mbps path. From the figure it can be observed that both client can get the full bandwidth they purchased. Since the traffic is travelling on two different paths, they didn't experience any competition from each other.

This test scenario demonstrates that the economical interactions incurred by ChoiceNet helps to fully utilize the available network capability. It encourages the users to use alternative instead of competing on a single path. Although this is a short-term example, we believe comprehensive economical interactions will prompt the evolution of the network services.

---

[6] `https://iperf.fr/`

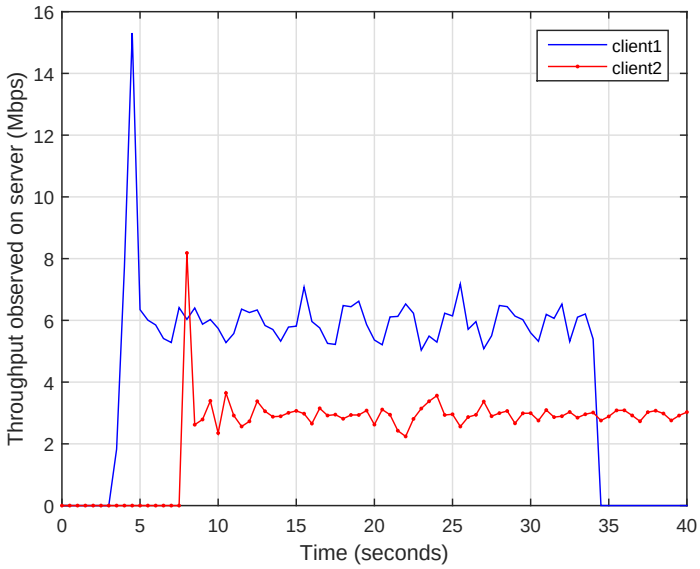


Figure 8.5. Interactions of two customers.

8.3 Performance Evaluation

Although the ChoiceNet enables choice, it also introduces additional overhead when originating a flow. To quantify the additional overhead, we measure the average time it takes to perform each step of setting up a flow (see Table 8.1).

The results here assume the choice is made instantly. The path query (which contains the ParetoBFS running time) and provisioning time highly depends on the round-trip time (RTT) between the marketplace and the providers/users. In this experiment, the RTT is about 70 ms between the marketplace and the users. Since the topology is small, ParetoBFS running time is trivial compared to other overhead. For larger topology with several thousands of nodes, the ParetoBFS running time may increase to the order of seconds. Once the flow is set up, packets flow through the data plane without any added overhead.

Compared to the standard SDN, where each switch needs to send the first packet to the controller for decision, ChoiceNet installs flow table entry on all the switches along the path at the same time. This reduces the setup time if there are more than one switch on the path.

Table 8.1. Breakdown of connection setup times.

Task	**Time**
path query	72.19 ms
ParetoBFS running time (included above)	0.38 ms
Provision the path	150.11 ms
Total connection setup time	222.30 ms

8.4 Conclusions

This chapter presents the deployment of ChoiceNet implementation on GENI. The test results show this implementation is capable of performing a complete pathlet service transaction. The users can choose the paths according to their preference. This brings various advantages such as flexible path migrating and load balancing driven by bidding.

CHAPTER 9

SUMMARY

Economic relationships between entities in the network are a critical driver for operation of the Internet. This work presents the design of ChoiceNet, an economic plane that can associate economic contracts with network layer services. After presenting the architecture of ChoiceNet, this work first defines a syntax for network layer service. Such syntax allows an unified representation of services, making a standardized service selling and purchasing protocol on the marketplace. Then we describes the architecture of the marketplace. APIs for customers and providers are introduced, and parallel computing are used to boost the marketplace performance. We have also solved various design challenges in the use plane, including the control logic in a centralized environment, the ChoiceNet App which integrate the economy interaction into the end system, and the routing algorithm for both intra- and inter-domain path. Two types of hybrid deployment with legacy network are proposed, which helps the incremental deployment in the current Internet. We have deployed the implementation on GENI. Evaluation results show this implementation is capable of enabling the economical interactions between entities.

This work also addresses some fundamental algorithmic problems in ChoiceNet. Two new algorithms, ParetoBFS and OrthCredential, are presented. ParetoBFS solves the problem of finding all the Pareto-optimal paths in a multi-criteria network. Experiments show it works well and can get a solution on a typical network in reasonable time. OrthCredential is a new credential design to solve the problem of

protecting reserved services with very low overhead in terms of verification time and memory consumption, while guaranteeing good security performance.

We believe that integrating the realities of economic relationships into the core of new network architectures is critical to ensure that innovative technology can be deployed in the future Internet. The design and implementation described in this dissertation is an important step into this direction.

BIBLIOGRAPHY

[1] Albert, Réka, and Barabási, Albert-László. Topology of Evolving Networks: Local Events and Universality. *Phys. Rev. Lett. 85* (Dec 2000), 5234–5237.

[2] AMD. *AMD Athlon Processor x86 Code Optimization Guide*, 2002.

[3] Ascigil, Onur, Calvert, Kenneth L., and Griffioen, James. On the Scalability of Interdomain Path Computations. In *Proceedings of the 13th IEEE/IFIP Networking* (2014), pp. 34–42.

[4] Assmus Jr., E. F., and Key, J. D. *Designs and their codes*. Cambridge University Press, Cambridge, Great Britain, 1992.

[5] Babaoglu, A.C., and Dutta, R. A verification service architecture for the future internet. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on* (July 2013), pp. 1–9.

[6] Babaoglu, Ahmet Can, and Dutta, Rudra. A verification service architecture for the future internet. In *Proc. of the 22nd IEEE International Conference on Computer Communications and Networks (ICCCN)* (Nassau, Bahamas, Aug. 2013).

[7] Baldine, Ilia, Vellala, Manoj, Wang, Anjing, Rouskas, George, Dutta, Rudra, and Stevenson, Daniel. A unified software architecture to enable cross-layer design in the future Internet. In *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Honolulu, HI, Aug. 2007).

[8] Baldine, Ilia, Xin, Yufeng, Mandal, Anirban, Ruth, Paul, Heerman, Chris, and Chase, Jeff. Exogeni: A multi-domain infrastructure-as-a-service testbed. In *Testbeds and Research Infrastructure. Development of Networks and Communities*. Springer, 2012, pp. 97–113.

[9] Barabasi, Albert-Laszlo, and Albert, Reka. Emergence of Scaling in Random Networks. *Science 286*, 5439 (1999), 509–512.

[10] Barber, C. Bradford, Dobkin, David P., and Huhdanpaa, Hannu. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software 22*, 4 (1996), 469–483.

[11] Barrett, Chris, Bisset, Keith, Holzer, Martin, Konjevod, Goran, Marathe, Madhav, and Wagner, Dorothea. Engineering label-constrained shortest-path algorithms. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management* (2008), AAIM '08, pp. 27–37.

[12] Bellman, Richard. On a routing problem. *Quarterly of Applied Mathematics 16*, 1 (Jan. 1958), 87–90.

[13] Berman, Mark, Chase, Jeffrey S, Landweber, Lawrence, Nakao, Akihiro, Ott, Max, Raychaudhuri, Dipankar, Ricci, Robert, and Seskar, Ivan. GENI: A federated testbed for innovative network experiments. *Computer Networks* (2014).

[14] Black, John, Halevi, Shai, Krawczyk, Hugo, Krovetz, Ted, and Rogaway, Phillip. UMAC: Fast and secure message authentication. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)* (Springer-Verlag, 1999), pp. 216–233.

[15] Bu, T., and Towsley, D. On Distinguishing between Internet Power Law Topology Generators. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2002), vol. 2, pp. 638–647 vol.2.

[16] Cai, Hao, Chen, Xinming, and Wolf, T. Orthcredential: A new network capability design for high-performance access control. In *IEEE 22nd International Conference on Network Protocols (ICNP)* (Oct 2014), pp. 233–244.

[17] Calvert, K. L., and Zegura, E. W. Composable active network elements. `http://www.cc.gatech.edu/projects/canes/`.

[18] Calvert, K.L., Venkatraman, S., and Griffioen, J.N. FPAC: fast, fixed-cost authentication for access to reserved resources. In *Proc. of IEEE INFOCOM 2002* (March 2002), pp. 1049–1058.

[19] Chen, Shigang, and Nahrstedt, K. On Finding Multi-constrained Paths. In *Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference on* (Jun 1998), vol. 2, pp. 874–879 vol.2.

[20] Chen, Xinming, Cai, Hao, and Wolf, Tilman. Multi-criteria routing in networks with path choices. In *Proc. of 23rd IEEE International Conference on Network Protocols (ICNP'15)* (2015).

[21] Chen, Xinming, Dwaraki, Abhishek, Cai, Hao, and Wolf, Tilman. Specification and composition of network services in future internet architectures. In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop* (New York, NY, USA, 2012), CoNEXT Student '12, ACM, pp. 45–46.

[22] Chen, Xinming, Wolf, Tilman, Griffioen, Jim, Ascigil, Onur, Dutta, Rudra, Rouskas, George, Bhat, Shireesh, Baldin, Ilya, and Calvert, Ken. Design of a Protocol to Enable Economic Transactions for Network Services. In *Proceedings of 2015 IEEE International Conference on Communications (ICC)* (2015).

[23] Clark, David D., Wroclawski, John, Sollins, Karen R., and Braden, Robert. Tussle in cyberspace: defining tomorrow's Internet. *SIGCOMM Computer Communication Review 32*, 4 (Oct. 2002), 347–356.

[24] Dijkstra, Edsger W. A note on two problems in connexion with graphs. *Numerische Mathematik 1* (Dec. 1959), 269–271.

[25] Diot, C., Levine, B. N., Lyles, B., Kassem, H., and Balensiefen, D. Deployment issues for the IP multicast service and architecture. *IEEE Network 14*, 1 (Jan. 2000), 78–88.

[26] Dutta, Rudra, Rouskas, George N., Baldine, Ilia, Bragg, Arnold, and Stevenson, Dan. The SILO architecture for services integration, control, and optimization for the future Internet. In *Proc. of IEEE International Conference on Communications (ICC)* (Glasgow, Scotland, June 2007), pp. 1899–1904.

[27] Dwaraki, Abhishek, and Wolf, Tilman. Service instantiation in an Internet with choices. In *Proc. of the 22nd IEEE International Conference on Computer Communications and Networks (ICCCN)* (Nassau, Bahamas, Aug. 2013).

[28] Ehrgott, Matthias, and Gandibleux, Xavier. A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization. *OR-Spektrum 22*, 4 (2000), 425–460.

[29] Ergun, Funda, Sinha, Rakesh, and Zhang, Lisa. An Improved FPTAS for Restricted Shortest Path. *Inf. Process. Lett. 83*, 5 (Sept. 2002), 287–291.

[30] Farrel, Adrian, Vasseur, Jean-Philippe, and Ash, Jerry. A Path Computation Element (PCE)-Based Architecture. RFC 4655, Network Working Group, Aug. 2006.

[31] Gandibleux, Xavier, Beugnies, Frdric, and Randriamasy, Sabine. Martins' Algorithm Revisited for Multi-objective Shortest Path Problems with a MaxMin Cost Function. *4OR 4*, 1 (2006), 47–59.

[32] Garroppo, Rosario G., Giordano, Stefano, and Tavanti, Luca. A Survey on Multi-constrained Optimal Path Computation: Exact and Approximate Algorithms. *Comput. Netw. 54*, 17 (Dec. 2010), 3081–3107.

[33] Godfrey, P. Brighten, Ganichev, Igor, Shenker, Scott, and Stoica, Ion. Pathlet routing. In *Proc. of the ACM SIGCOMM Conference on Data Communication* (Barcelona, Spain, Aug. 2009), pp. 111–122.

[34] Godfrey, P. Brighten, Ganichev, Igor, Shenker, Scott, and Stoica, Ion. Pathlet routing. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication* (New York, NY, USA, 2009), SIGCOMM '09, ACM, pp. 111–122.

[35] Gupta, Arpit, Vanbever, Laurent, Shahbaz, Muhammad, Donovan, Sean Patrick, Schlinker, Brandon, Feamster, Nick, Rexford, Jennifer, Shenker, Scott, Clark, Russ, and Katz-Bassett, Ethan. Sdx: A software defined internet exchange. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 579–580.

[36] Hansen, Pierre. Bicriterion Path Problems. In *Multiple Criteria Decision Making Theory and Application*, Gnter Fandel and Tomas Gal, Eds., vol. 177 of *Lecture Notes in Economics and Mathematical Systems*. Springer Berlin Heidelberg, 1980, pp. 109–127.

[37] Haque, Imran S., Pande, Vijay S., and Walters, W. Patrick. Anatomy of high-performance 2d similarity calculations. *Journal of Chemical Information and Modeling 51*, 9 (2011), 2345–2351.

[38] Hedrick, C. Routing information protocol. RFC 1058, Network Working Group, June 1988.

[39] Huang, X., Shanbhag, S., and Wolf, T. Automated service composition and routing in networks with data-path services. In *Proceedings of the IEEE ICCCN 2010 Conference* (2010).

[40] Jacobson, Van, Smetters, Diana K., Thornton, James D., Plass, Michael F., Briggs, Nicholas H., and Braynard, Rebecca L. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies (CoNEXT)* (Rome, Italy, Dec. 2009), pp. 1–12.

[41] Kharaghani, H., and Tayfeh-Rezaie, B. Hadamard matrices of order 32. *Journal of Combinatorial Designs 21*, 5 (May 2012), 212–221.

[42] Khetrapal, Gautam, and Sharma, Saurabh Kumar. Demystifying Routing Services in Software-Defined Networking. Tech. rep., Aricent Inc., 2013.

[43] Khondoker, R., Reuther, B., Schwerdel, D., Siddiqui, A., and Muller, P. Describing and selecting communication services in a service oriented network architecture. In *Kaleidoscope: Beyond the Internet? - Innovations for Future Networks and Services, 2010 ITU-T* (2010), pp. 1–8.

[44] Kohler, Eddie, Morris, Robert, Chen, Benjie, Jannotti, John, and Kaashoek, M. Frans. The click modular router. *j-TOCS 18*, 3 (2000), 263–297.

[45] Korkmaz, Turgay., and Krunz, Marwan. Multi-constrained Optimal Path Selection. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2001), vol. 2, pp. 834–843 vol.2.

[46] Krawczyk, H., Bellare, M., and Canetti, R. Hmac: Keyed-hashing for message authentication. RFC 2014.

[47] Kresten, Proteus Valre. *Windows Filtering Platform*. VolutPress, 2012.

[48] Li, Zhenjiang, and Garcia-Luna-Aceves, J. J. A Distributed Approach for Multi-constrained Path Selection and Routing Optimization. In *Proceedings of the 3rd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks* (New York, NY, USA, 2006), QShine '06, ACM.

[49] Lloyd, Stuart P. Least Squares Quantization in PCM. *Information Theory, IEEE Transactions on 28*, 2 (Mar 1982), 129–137.

[50] Martins, E. Q. V. *Bibliography of papers on Multiobjective Optimal Path Problems*. http://www.mat.uc.pt/∼eqvm/cientificos/biblio/mo.ps.Z, 1996.

[51] Martins, Ernesto Queirs Vieira. On a Multicriteria Shortest Path Problem. *European Journal of Operational Research 16*, 2 (1984), 236 – 245.

[52] Medina, Alberto, Lakhina, Anukool, Matta, Ibrahim, and Byers, John. BRITE: An Approach to Universal Topology Generation. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (Washington, DC, USA, 2001), MASCOTS '01, IEEE Computer Society, pp. 346–.

[53] Moy, John. OSPF version 2. RFC 2328, Network Working Group, Apr. 1998.

[54] Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`.

[55] Naous, Jad, Walfish, Michael, Nicolosi, Antonio, Mazières, David, Miller, Michael, and Seehra, Arun. Verifying and enforcing network paths with icing. In *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2011), CoNEXT '11, ACM, pp. 30:1–30:12.

[56] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, Palo Alto, CA, USA, Apr. 2012.

[57] Pan, Jianli, Paul, S, and Jain, R Communications Magazine IEEE. A Survey of the Research on Future Internet Architectures. *Communications Magazine, IEEE 49*, 7 (2011).

[58] Pelegrin, Blas, and Fernndez, Pascual. On the sum-max bicriterion path problem. *Computers & Operations Research 25*, 12 (1998), 1043 – 1054.

[59] Rouskas, G.N., Baldine, I., Calvert, K., Dutta, R., Griffioen, J., Nagurney, A., and Wolf, T. Choicenet: Network innovation through choice. In *Optical Network Design and Modeling (ONDM), 2013 17th International Conference on* (April 2013), pp. 1–6.

[60] Ruf, Lukas, Farkas, Karoly, Hug, Hanspeter, and Plattner, Bernhard. Network services on service extensible routers. In *Proc. of Seventh Annual International Working Conference on Active Networking (IWAN 2005)* (Sophia Antipolis, France, Nov. 2005).

[61] Spring, N, Mahajan, R, Wetherall, D, and Anderson, T. Measuring ISP topologies with Rocketfuel. *Networking, IEEE/ACM Transactions on 12*, 1 (2004), 2–16.

[62] Stockhammer, Thomas. Dynamic adaptive streaming over http –: Standards and design principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems* (New York, NY, USA, 2011), MMSys '11, ACM, pp. 133–144.

[63] Tsaggouris, George, and Zaroliagis, Christos. Multiobjective Optimization: Improved FPTAS for Shortest Paths and Non-linear Objectives with Applications. In *Algorithms and Computation*, Tetsuo Asano, Ed., vol. 4288 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 389–398.

[64] Ulungu, E. L., and Teghem, J. Multi-objective Combinatorial Optimization Problems: A Survey. *Journal of Multi-Criteria Decision Analysis 3*, 2 (1994), 83–104.

[65] Wang, Z., and Crowcroft, J. Quality-of-service routing for supporting multimedia applications. *Selected Areas in Communications, IEEE Journal on 14*, 7 (Sep 1996), 1228–1234.

[66] Warburton, A. Approximation of Pareto Optima in Multiple-objective, Shortest-path Problems. *Oper. Res. 35*, 1 (Feb. 1987), 70–79.

[67] Waxman, B.M. Routing of multipoint connections. *Selected Areas in Communications, IEEE Journal on 6*, 9 (Dec 1988), 1617–1622.

[68] Wolf, T., Natarajan, S., and Vasudevan, K.T. High-performance capabilities for 1-hop containment of network attacks. *Networking, IEEE/ACM Transactions on 21*, 6 (Dec 2013), 1931–1946.

[69] Wolf, Tilman. Service-centric end-to-end abstractions in next-generation networks. In *Proc. of Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Arlington, VA, Oct. 2006), pp. 79–86.

[70] Wolf, Tilman. In-network services for customization in next-generation networks. *IEEE Network 24*, 4 (July 2010), 6–12.

[71] Wolf, Tilman, Griffioen, James, Calvert, Kenneth L., Dutta, Rudra, Rouskas, George N., Baldin, Ilya, and Nagurney, Anna. ChoiceNet: Toward an Economy Plane for the Internet. *SIGCOMM Comput. Commun. Rev. 44*, 3 (July 2014), 58–65.

[72] Wolf, Tilman, Griffioen, James, Calvert, Kenneth L., Dutta, Rudra, Rouskas, George N., Baldine, Ilia, and Nagurney, Anna. Choice as a principle in network architecture. vol. 42, ACM, pp. 105–106.

[73] Wolf, Tilman, Griffioen, James, Calvert, Kenneth L., Dutta, Rudra, Rouskas, George N., Baldine, Ilia, and Nagurney, Anna. ChoiceNet: toward an economy plane for the Internet. *ACM SIGCOMM Computer Communication Review 44*, 3 (July 2014), 58–65.

[74] Xue, Guoliang, Zhang, Weiyi, Tang, Jian, and Thulasiraman, K. Polynomial Time Approximation Algorithms for Multi-Constrained QoS Routing. *Networking, IEEE/ACM Transactions on 16*, 3 (June 2008), 656–669.

[75] Yang, Xiaowei, Wetherall, David, and Anderson, Thomas. Tva: a dos-limiting network architecture. *IEEE/ACM Transactions on Networking 16*, 6 (December 2008), 1267–1280.