Protocol and System Design for a Service-centric Network Architecture

PROTOCOL AND SYSTEM DESIGN FOR A SERVICE-CENTRIC NETWORK ARCHITECTURE

A Dissertation Presented

by

XIN HUANG

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

DOCTOR OF PHILOSOPHY

February 2010

Electrical and Computer Engineering

PROTOCOL AND SYSTEM DESIGN FOR A SERVICE-CENTRIC NETWORK ARCHITECTURE

A Dissertation Presented

by

XIN HUANG

Approved as to style and content by:

Tilman Wolf, Chair

Lixin Gao, Member

Weibo Gong, Member

Jim Kurose, Member

C. V. Hollot, Department Head
Electrical and Computer Engineering

To my dearest parents and those who are important in my life.

Thank you for all your love, support, and encouragement, especially during my tough times.
Without you, I would not have achieved what I have today.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Professor Tilman Wolf. With his mentorship, encouragement, and patience, I successfully tackled the problems discussed in this dissertation and significantly enhanced my research skills. The only thing I want to say is: I could not have asked for a more suitable person to guide me throughout the journey of my doctoral study and research.

I would like to thank the members of my committee, Professor Jim Kurose, Professor Weibo Gong, and Professor Lixin Gao, as well as Professor Ramakrishna Janaswamy, for their constructive advice and invaluable help on both of my research and career.

The members in the Network Systems Lab need also be acknowledged for the knowledge and experience they have shared with me. In particular, I want to thank Shiv Ganapathy and Shashank Shanbhag, who worked together with me on this great project. I would like to thank Dr. Ning Weng, Dr. Ramaswamy, and Qiang Wu for the enlightening discussions about both general and research problems. I also owe my thank to the KEB dudes (particularly Amrit Kumar, Gautam Das, Siva Gurumurthy, Nupur Murlidharan, Zhiguo Lai, Huaping Wang, and Lan Lan) for your friendship, help, and support, which makes my Ph.D. life so enjoyable and memorable.

Finally and most importantly, I am grateful to my family, especially my parents and my boyfriend, Jie Xu. Your constant support, love, and affection has always been the greatest blessings in my life.

ABSTRACT

PROTOCOL AND SYSTEM DESIGN FOR A SERVICE-CENTRIC NETWORK ARCHITECTURE

FEBRUARY 2010

XIN HUANG

B.Sc., TIANJIN UNIVERSITY, P.R. CHINA

M.Sc., TIANJIN UNIVERSITY, P.R. CHINA

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Tilman Wolf

Next-generation Internet will be governed by the need for flexibility. Heterogeneous end-systems, novel applications, and security and manageability challenges require networks to provide a broad range of services that go beyond store-and-forward. Following this trend, a service-centric network architecture is proposed for the next-generation Internet. It utilizes router-based programmability to provide packet processing services inside the network and decompose communications into these service blocks. By providing different compositions of services along the data path, such network can customize its connections to satisfy various communication requirements. This design extends the flexibility of the Internet to meet its next-generation challenges.

This work addresses three major challenges in implementing such service-centric networks. Finding the optimal path for a given composition of services is the first

challenge. This is called "service routing" since both service availability and routing cost need to be considered. Novel algorithms and a matching protocol are designed to solve the service routing problem in large scale networks. A prototype based on Emulab is implemented to demonstrate and evaluate our design. Finding the optimal composition of services to satisfy the communication requirements of a given connection is the second challenge. This is called "service composition." A novel decision making framework is proposed, which allows the deduction of the service composition problem into a planning problem and automates the composition of service according to specified communication requirements. A further investigation shows that extending this decision making framework to combine the service routing and service composition problems yields a better solution than solving them separately. Run-time resource management on the data plane is the third challenge. Several run-time task mapping approaches have been proposed for Network Processor systems. An evaluation methodology based on queuing network is designed to systematically evaluate and compare these solutions under various network traffic scenarios. The results of this work give qualitative and quantitative insights into next-generation Internet design that combines issues from computer networking, architecture, and system design.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS . . . . . **v**

ABSTRACT . . . . . **vi**

LIST OF TABLES . . . . . **xii**

LIST OF FIGURES . . . . . **xiii**

CHAPTER

1. INTRODUCTION **1**

1.1 Evolution of the Internet 1
1.2 Trends for the Next-Generation Internet 5
1.3 Service-Centric Network Architecture 8
1.4 Challenges and Opportunities 10

1.4.1 Control-plane Design Issues 10
1.4.2 Data-plane Design Issues 11

1.5 Organization and Contributions 12

2. SERVICE-CENTRIC NETWORK ARCHITECTURE **14**

2.1 Design Overview 14

2.1.1 Network Services 17
2.1.2 Dynamic Service Composition 19

2.2 Network Architecture 21
2.3 Network Service Abstraction 22

2.3.1 Network Service Stack 25
2.3.2 Interface 26
2.3.3 Deployment Model 28

2.3.4 Usage Scenarios . . . 29

2.4 Related Work . . . 32
2.5 Summary . . . 33

3. SERVICE ROUTING . . . 34

3.1 Introduction . . . 34
3.2 Service Routing Problem . . . 36
3.3 Global Algorithms . . . 38

3.3.1 Algorithm I: Randomized Placement . . . 39
3.3.2 Algorithm II: Layered Graph . . . 40
3.3.3 Algorithm III: Service Step Search . . . 42
3.3.4 Capacity Constraints . . . 43
3.3.5 Evaluation Results . . . 44

3.4 Decentralized Algorithms . . . 51

3.4.1 Service Routing as Dynamic Programming Problem . . . 52
3.4.2 DSMR Algorithm . . . 54
3.4.3 Approximate DSMR . . . 56

3.5 Analytical Comparison of the Algorithms . . . 57
3.6 Scalable Routing Protocol . . . 58

3.6.1 DSRP Setup . . . 59
3.6.2 DSRP Routing Information Exchange . . . 61
3.6.3 DSRP Connection Setup . . . 62

3.7 Results and Analysis . . . 64

3.7.1 Prototype Implementation . . . 65
3.7.2 DSMR Algorithm . . . 65
3.7.3 DSRP Protocol . . . 71

3.8 Related Work . . . 78
3.9 Summary . . . 79

4. SERVICE COMPOSITION . . . 81

4.1 Introduction . . . 81
4.2 Automated Service Composition . . . 83

4.2.1 Service Composition Problem . . . 84
4.2.2 Service Composition Framework . . . 84
4.2.3 Data and Communication Characteristics . . . 86

4.2.4 Automation of Service Composition 88
4.2.5 Examples 90

4.3 Automated Service Composition and Routing 93

4.3.1 Motivation 93
4.3.2 Service Composition and Routing Problem 95
4.3.3 Decision Making Framework 98
4.3.4 Combined Service Composition and Routing 101

4.4 Results and Analysis 104

4.4.1 Synthetic Service Hypercube Benchmark 104
4.4.2 Synthetic Benchmark Scenarios 106
4.4.3 Practical Scenarios 109

4.5 Related Work 112
4.6 Summary 113

5. DATA-PATH RESOURCE MANAGEMENT 115

5.1 Introduction 115
5.2 Evaluation Methodology 117
5.3 Workload Model 120

5.3.1 Applications 121
5.3.2 Traffic 121
5.3.3 Synthetic Traffic 121
5.3.4 Example Workload 124

5.4 Runtime System Model 127

5.4.1 Network Processor Model 127
5.4.2 Runtime Mapping 129
5.4.3 Example Runtime Systems 130

5.5 Analytical Evaluation 131

5.5.1 M/M/m–FCFS Model 132
5.5.2 Performance Metrics 133

5.6 Results and Analysis 134

5.6.1 Setup 134
5.6.2 Analytical Results 135
5.6.3 Simulation Results 141
5.6.4 Implications for Runtime System Design 147

5.7 Related Work 148
5.8 Summary 149

6. SUMMARY AND FUTURE WORK 151

6.1 Summary 151
6.2 Future Work 154

BIBLIOGRAPHY 156

LIST OF TABLES

Table **Page**

3.1 Time to Compute Path for Connection Request. 51

3.2 Qualitative Comparison of Algorithms. 51

3.3 Time and Space Complexity of Algorithms. 57

3.4 Service Matrix Convergence Time. 72

4.1 Preconditions and Actions for the Example Services. 92

4.2 Input and Output Semantics for Example Scenario 1. 92

4.3 Input and Output Semantics for Example Scenario 2. 92

4.4 Input and Output Semantics for Example Scenario 3. 93

4.5 Results for the 3 Example Scenarios. 93

5.1 Traffic Scenarios for Workload Characterization. 123

LIST OF FIGURES

Figure **Page**

2.1 Potential of Service-centric Network Architecture. 16

2.2 Protocol Stack: Current Internet vs. Service-Centric Networks. 17

2.3 Service Composition in Service-centric Networks. 20

2.4 Service-Centric Network Architecture. 21

2.5 Comparison of Virtualization Approaches. 23

2.6 Network Service Abstraction. 24

3.1 Routing Problem in Service-centric Networks. 35

3.2 An Example of the Service Routing Problem. 37

3.3 Randomized Solution for the Problem in Fig. 3.2. 39

3.4 Layered Graph Solution for the Problem in Fig. 3.2. 41

3.5 Service Step Search Solution for the Problem in Fig. 3.2. 43

3.6 CDF of Connection End-to-End Delay on Network 1. 46

3.7 Connection Drop Rate on Network 1. 47

3.8 CDF of Connection End-to-End Delay on Network 2. 48

3.9 Dropped Connections over Connection Attempts on Network 2. 49

3.10 Network Usage over Successful Connections on Network 2. 49

3.11 CDF of Processing Node Utilization on Network 2. 50

3.12 Decentralized Service Routing. 53

3.13 Service Matrix. 54

3.14 DSMR Approximation and Path Bounds. 56

3.15 DSRP Setup. 60

3.16 Space-Time Diagram for DSRP Setup. 60

3.17 DSRP Connection Setup. 62

3.18 Space-Time Diagram for DSRP Connection Setup. 63

3.19 Emulab Prototype. 65

3.20 Path Cost (Approximate DSMR vs. Layered Graph (Optimal Solution)). 66

3.21 CDF of Relative Path Cost (Approximate DSMR vs. Layered Graph (Optimal Solution)). 67

3.22 Example Path: Layered Graph vs. Approximate DSMR. 69

3.23 CDF of Approximate DSMR Path Cost to Optimal Path Cost for Different Path Lengths and four Services ($k = 4$). 70

3.24 Connection Setup Time (DSRP vs. TCP). 73

3.25 CDF of Connection Setup Time (DSRP vs. TCP). 74

3.26 CDF of Relative Connection Setup Time. 74

3.27 CDF of Control Information Size (DSRP vs. TCP). 75

3.28 CDF of Relative Control Information Size (DSRP vs. TCP). 76

3.29 Number of Entries Maintained on All 12 Controllers. 77

4.1 Service Composition Framework. 85

4.2 Semantics for Representing Characteristics. 87

4.3 An Example for the Service Composition and Routing. 94

4.4 Decision Making System. 99

4.5 Two Solutions: SSCR vs. CSCR. 102

4.6 Synthetic Benchmark Scenarios. (L=6, h=2, r=0.5, $\Omega = \{0, 1\}$) 104

4.7 Synthetic Service Hypercube Graphs. (L=3, h=1, r=0.56, $\Omega = \{0, 1\}$) 104

4.8 Path Cost under Synthetic Scenarios (CSCR vs. SSCR). 107

4.9 CDF of Relative Path Cost under Synthetic Scenarios (CSCR vs. SSCR). 108

4.10 Practical Network Scenario. 109

4.11 Path Cost under Practical Scenario (CSCR vs. SSCR). 111

4.12 CDF of Relative Path Cost under Practical Scenario (CSCR vs. SSCR). 111

5.1 Components of Evaluation Methodology and Their Interactions. 118

5.2 Example Workload Graph. 125

5.3 Packets per Second for each Path for Example in Fig. 5.2 126

5.4 System Model Instantiations for Different Runtime Systems. 129

5.5 Effectiveness of Static Allocation. 135

5.6 Processors Allocated over Time in Full Processor Allocation System. 136

5.7 Processor Utilization over Time. 137

5.8 Packets in System over Time. 138

5.9 Packets in System under Different Traffic Scenarios. 139

5.10 Packets in System over Different Data Rates. 140

5.11 Service Time CDF of All Applications, the Ideal System, and an Exponentially Distributed Random Variable. 142

5.12 Packets in System over Different Data Rates. 143

5.13 System Performance over Time. 146

CHAPTER 1

INTRODUCTION

1.1 Evolution of the Internet

The design of today's Internet origins from a internetworking architecture that was developed in 1970's, under the Defense Advanced Research Projects Agency (DARPA) [3, 22, 63, 64]. The top level goal for this DRAPA Internet architecture was to develop an effective technique for interconnecting existing networks, such as the original ARPANET [61], the ARPA packet radio network [17, 86], etc.

From this design goal comes the fundamental structure of the DRAPA Internet: a simple and effective packet-switched communication infrastructure, which connects a large number of networks with gateways (or "routers") that implement the store-and-forward packet forwarding algorithm [28]. Later, more and more protocols (i.e., TCP [32, 107] and UDP [62]) were developed and the current Internet protocol stack was built up.

One important principle of the current Internet design is to keep the network itself relatively simple (forwarding packets) while implement most of the complex functionalities on the end-systems (e.g., applications, retransmission of lost packets, congestion control) [59]. Since all the communication status information are kept on the end-systems, this design completely masks any transient failures happens inside the network.

This simple and transparent design of the network was an important factor in the success of the current Internet, since it enabled deployment of new applications without having to change the core of the network. However, with the development of

modern communication technology and the ubiquitous use of computer networks in our daily life, numerous new requirements and applications emerged. Many of them were not considered and even conflict with the current Internet design. Examples of such requirements are [16]:

- Security: The simple and transparent design of the network was an important factor in the success of the current Internet. However, this also facilitates security attacks, viruses, and other unwelcome data. The original users of the internet were members of the network research community, who shared the same environment with a high degree of trust. At that time, the power of this architecture design outweighed its risks. Nowadays, the pool of users has expanded to all kinds of people who have different interests and hence the security of the network has become one of the most critical issues in the current Internet design.

- Management and Control: With the expansion of the Internet toward commercial enterprise, government, and our daily life, an increasingly visible issue is the demands by the third parties (e.g., ISPs, governments, or the administrators of a corporate network). They usually require to investigate the headers of the data packets and control the way that network transfers (or drops) these data packets. These new requirements from the third parties to interpose themselves between communicating end-systems are against the simple and transparent design of the current Internet.

- Connectivity to Heterogeneous End-Systems: The current Internet architecture design pushes the functionality from the network to the end-systems because at the design time the end-systems were computers with enough processing capacity. However, with the development of the communication technology, modern networks are required to provide connectivity to mobile and ad-hoc sensor net-

works where the end-systems have very low processing power (e.g., RFID tags). These end-systems hence require specialized functionality or services from the network to compensate for their limited processing capability.

- Accommodating Innovative Applications: The current Internet provides a very simple and minimally specified packet transfer service, which is called "best effort delivery." However, with the development of the Internet, real-time multimedia applications have emerged as potential killer applications for the next-generation networks. Many of these applications require guarantees on end-to-end delay, jitter, loss, etc., which conflict with the "best effort" design philosophy of the current Internet architecture.

To meet these requirements, new functions need to be added. However, the Internet is not flexible enough to be extended to accommodate these changes. The end-to-end design limits the functionality of the routers to only forwarding packets. Furthermore, routers usually rely on customized hard-ware based processing ability to forward packets at line speed [73,87]. All these make it difficult to change anything inside the network without the help of the router vendors.

As a result, functions are either added to later generations of routers (through the vendors) or implemented on add-on devices that are later connected to the edge of the Internet. The following lists a few existing examples:

- Random Early Detection (RED) [39] is a queue management scheme for routers to fairly drop packets from rogue TCP flows. This is can be implemented in a very simple fashion, but it constitutes a type of processing on a router.

- Firewalls [76] are a standard security component of most networks, which inspects network traffic passing through them, and deny or permit passage based on a set of rules. This enables the blocking of network traffic that could compromise the security of hosts on the network (e.g., port scanning). Firewall rules

can be numerous and complex, which requires significant computational power on the firewall to keep up with typical access link speeds.

- Network Address Translation (NAT) [37] enables multiple computers behind a network router to share a single globally unique IP address. The router needs to translate between the local and global IP addresses and modify the corresponding headers of the packets passing through it.

- Multicast is a method of forwarding packets to a group of interested receivers. It is required by a number of network applications such as IPTV, teleconferencing, online gaming and web cache updating. The class D multicast address [117] is used to identify groups of receivers. Routing algorithms and network layer protocols (e.g., DVMRP [30], PIM [96], ICMP [8]) were designed for implementing multicast over IP networks. Nowadays, most routers support IP multicast.

- Web switching [7] is a method of distributing traffic to a web server over several physical machines while presenting a single front-end to the outside. Web switches parse HTTP requests in packets and determine the appropriate server to which to forward the request. Since the HTTP request is sent only after the TCP connection is established, the web switch also has to splice the TCP connection between client and back-end server.

The following are some new functions that are proposed to be added in the network:

- IP traceback [92] allows the network to keep state on the traffic that was forwarded and provides the ability to identify the sources of possibly malicious data flows. For this purpose, routers need to compute a hash for the data packet and store it for a possible later audit.

- Content adaptation refers to a whole class of applications that modify content (e.g. web pages) into a form that is suitable for the end user. Examples of content adaptation include ad insertion in web pages and content transcoding for wireless clients [72].

However, since the Internet itself does not support the dynamic development of new functions, protocols, and applications. Most of these designs have been designed as point solutions for specific requirements. Thus, the coherence and the efficiency of the original Internet design is vanishing in a patchwork of technical embellishments.

The only way to avoid this degeneration would be to restore coherence to the architecture, which can happen only as the result of a deliberate effort to create a new Internet architecture for the future. Nowadays, the network community is in the process of designing the "Next-Generation Internet," an Internet that aligns better with its current and future requirements. Clean slate design is allowed in this area. That means, the designs for the next-generation Internet need not to be constrained by features of the existing network.

1.2 Trends for the Next-Generation Internet

General-purpose workstation processors that perform packet routing and forwarding in software were common router configurations in the 1980's. Typical link speeds of a few kilobits per second did not exceed the processing power of such a system. As performance demands for communication changed in the 1990's toward link speeds of several megabits per second, software-based routers were not able to keep up with this trend. As a result, ASIC-based routers were developed to provide basic protocol processing functionality at high speeds. The majority of current Internet routers are still ASIC-based. Their limitations in supporting new protocols led to efforts reintroduce the flexibility and extensibility of software-based programmable systems.

Integrating general-purpose processing as part of the data plane was proposed by Active Network [98, 99] at the end of 1990's. Through these data-path processing capability, network programmability is enabled. In active networks, end-system applications or users can deploy new functions through packets. Processing code is proposed to be encapsulated within a packet. As the packet traverse the Internet, the code is installed (and thus the corresponding function is deployed) on the routers it passes through.

Numerous research projects had spawned from this idea. Some were focus on exploring the new applications or services enabled by this architecture design [104, 105]. Others were aimed at investigating and implementing software-based active routers [33, 34, 109]. The key questions were how to dynamically install processing code in the data path, how to deploy code modules, and how to safely and securely execute arbitrary code on a router.

Active network suggests that end-systems (applications/users) have full access to the programmable router systems. Thus, new applications and services can be implemented in software and be created, deployed, and used dynamically. However, the arising issues of security, resource management and isolation, and programming complexity limit the practicality of such a general approach to network services.

A more moderate approach of providing general processing capability inside the network is found in the form of programmable networks [111]. A programmable network consists of programmable routers that have general-purpose processing units on their data path. These processing units can be programmed to perform various protocol operations as well as complex payload processing. However, unlike the active networks, programmable networks minimize the administrative issues that could rise from opening the network by allowing only a few authorities (e.g., router vendors, network administrators, or services providers) to deploy well-tested software on the routers. After these new functions are enabled, the end-systems may then choose

if their communication utilizes these functions. An example for utilizing processing capabilities to deploy new protocols is shown in [81].

The flexibility of a programmable network comes at a price. Software-based packet processing is inherently slower than customized logic that is optimized for a particular processing. However, commercial Network Processor (NP) systems provides a potential solution to this problem. The network processors are usually implemented as system-on-a-chip multiprocessors. They use several to dozens of simple, parallel RISC processors to achieve the computational power to handle Gigabit per second data rates.

Till now, a general architecture for network processors has not been developed. Numerous companies have announced and built such multiprocessor systems-on-a-chip based on their own architecture design (e.g., the IBMs Power NP [4], the Intels IXP1200 [58], the AMCC np7510 [5], and the EZchip NP-1 [38]). To make it worse, there even not exist a quantitative method for evaluating and comparing these designs. To solve this problem, different benchmarks [23, 42] and a evaluation framework [55] have been designed. On another hand, it is important to find a more systematic design approach to make fully use of the general processing capability provided by the multiprocessor NP systems. To achieve this same target, some projects work on the dynamic task mapping [68, 115]. Others focus on system architecture design [42, 43, 102, 110].

Network virtualization is another important trend for next-generation Internet design. It was first discussed in [6] to build a virtual testbed which can support multiple simultaneous architectures running on PlanetLab [1,82]. The intent is to dramatically reduce the barrier-to-entry for new architectural ideas to be evaluated in practice and thus helps to overcome the internet ossification impasse. Turner and Taylor further extended the discussion in [103] and suggested that virtualization should be an ar-

chitectural attribute of the next-generation Internet so that the problem of network ossification can be solved.

Since then, the networking research community has been working on implementing network virtualization. Some works focus on the virtualization technologies, e.g., separating user exclusion environments through containers [10], running separate guest operating systems [93], realizing the data plane virtualization through source code merging [66]. Virtualized testbed have been developed (e.g., GENI [78] and VINI [2]) to build a shared experimental platform for network researchers to broadly evaluate their protocol and application designs [13, 101, 102]. Other studies show how network virtualization helps to solve shortcomings in existing networks [21, 108] and how to contribute to the next-generation Internet architecture design [50, 80].

Nowadays, NetFPGA [71] has emerged as another potential hardware platform candidate for realizing programmability inside the network [79, 84]. A one-sentence definition about NetFPGA is: it is a programmable network hardware, which provides a line-rate, flexible, and open platform for research and classroom experimentation. It is widely used as a high performance programmable hardware platform in the next-generation Internet research community [24, 60, 79, 108]. It is also used to implement network virtualization [77].

In summary, it is widely believed that network programmability and network virtualization are two most important trends of the Internet evolution. They are the essential attributes of the next-generation Internet architecture.

1.3 Service-Centric Network Architecture

The trends in networking show that next-generation Internet will be governed by the need for flexibility. As mentioned earlier, new functions have been added as the Internet evolving. Existing examples are broadcasting, RED, Firewalls, NAT,

web caching, load balancing, etc. While terminology may vary, these functions are characterized by advanced packet processing performed on the data plane.

As networks encompass more diverse end-systems and novel networking paradigms, more and more advanced packet processing needs to be added into the data plane (e.g., IP traceback, Transcoding for wireless client, openflow [31, 79], etc.). Note that in this work, we call these advanced packet processing functions as "network services," or simply "services."

Network programmability and network virtualization are regarded as the essential architectural attributes for the next-generation Interment. They make it practical to realize flexibility inside the network. Network programmability enables innovative network services to be developed and deployed inside the network, and network virtualization allows these services run simultaneously on the same physical system without conflicting each other.

However, with more services existing inside the network, there comes an important question on the control-plane:

> *How to design an architectural framework, which is able to systematically manage these services, ease the use of these services, and maximize the utilization of these data-plane processing capabilities?*

A service-centric network architecture is proposed [112] as an answer to the question. A general overview about this service-centric architecture design is: services are distributed inside the network. When sending a connection request, end-system application specifies the desired communication functionality and performance properties of the connection, as well as the traditional source-destination pair. The network then determines the appropriate services and setup a data-path traversing these services to satisfy the communication requirements.

Instead of viewing the network as an entity that simply connects end-systems and transfers data, the service-centric network architecture treats the network as

an integral part of a distributed computing system. In this kind of network, new services can be deployed and installed on routers by service providers or network administrators. Networks can customize connections to satisfy various application requirements by providing different compositions of services along the data-path. This service-centric network architecture design integrates flexibility inside the network and thus expands the capabilities of the Internet to meet its next-generation challenges.

1.4 Challenges and Opportunities

Several theoretical and practical challenges are associated with designing such a service-centric network architecture. Some of the major design challenges are discussed below.

1.4.1 Control-plane Design Issues

- Network Architecture: High-level system design for implementing such a service-centric network. It answers the problems such as: What kind of network design (e.g., network components and their roles) is suitable for a service-centric network? How can nodes that provide services be managed in a scalable way? How can the control plane get to know, control, and manage the services? and what kind of state information need to be maintained inside the network?

- Communication Abstraction: developing a set of network application programming interface (API), service stack, and protocols to allow the end-system applications to fully utilize the capabilities of the service-centric network. It addresses the problems such as: What kind of abstraction is suitable for defining all the potential communication paradigms of the service-centric network? What should be the semantics of the abstraction? and How to interpret the abstraction semantics into a sequence of services that can satisfy the desired communication requirements?

- Service Routing: The communication abstraction translates a connection request into a sequence of services. The network then needs to decide how and where to map these required services to the physical resources (i.e., service nodes that provide these services inside the network). This part addresses the questions related to routing and connection setup: How to define an "optimal path"? What routing information is needed on the control plane? and How to find an optimal or near-optimal path?

- Security and Access Control: In service-centric networks, lots information need to be maintained on the control plane. For example, information about the services and their distribution, routing information, per-connection state information, etc. What type of information is private? and how could the private information be protected from being exposed to the public?

1.4.2 Data-plane Design Issues

- Router System Architecture Design: Service-centric network architecture requires part of the routers have programmable packet processing capability. General-purpose packet processing units need to be added to the data-path of the router systems, with a cost of lower packet handling speed. Thus, how to introduce the router programmability while at the same time keep packets processed at the line speed is the main target for the programmable routers system design.

- Resource Management: In a service-centric network, routers may need to provide different services simultaneously. An important design issue here is how to map the required services to the physical processing resources at runtime to maximize the performance of the router system.

- Security and Access Control: In service-centric networks, advanced packet processing capability is shared resource on the data plane. Thus, it is important to design appropriate security mechanisms and access control to provide a safe environment for communications in the service-centric network. Questions need to be answered are: What are the critical resources that need to be isolated? What's the right granularity for the isolation? How to implement the isolation on hardware-based router systems? What are the critical resources that need to be isolated?

1.5 Organization and Contributions

This dissertation focuses on three of the technical challenges discussed in Section 1.4: communication abstraction and service routing on the control plane, and resource management in the data plane.

The rest of this dissertation is structured as follows, and the major contributions in each part are also summarized accordingly.

Chapter 2 gives a general introduction to the service-centric network architecture. This chapter starts with a description about the big vision of the whole architecture design and several important concepts. Then, a detailed network architecture design is presented. Finally, a communication abstraction framework and two practical usage scenarios is described. This gives a general design overview about our solution for the communication abstraction, and at the same time illustrates where our service-centric network architecture stands in the clean- slate design of the next-generation Internet.

Chapter 3 introduces the service routing problem encountered in service-centric networks. This problem is proved to be NP-complete when considering resource constraints. Global algorithms are first proposed to solve the service-routing problem with resource constraints. However, global algorithms need a complete view about the whole network, which limits their scalability. A novel decentralized algorithm is then

introduced to solve this limitation. Finally, based on these algorithms, a matching protocol is designed to solve the service routing in large-scale networks. A prototype on Emulab is also implemented to demonstrate and evaluate our designs. The work in this chapter could be applied to any network with data-path services.

Chapter 4 discusses the problem faced in the design of the communication abstraction. That is, given a connection request, how to decide the optimal composition of services to satisfy the communication requirements of the request. This is called "service composition problem" in our work. A novel decision making framework is proposed to solve this problem. It allows the deduction of the service composition problem into a planning problem and automates the service composition process. This framework is further extended to combine the service routing and service composition problems into one problem.

Chapter 5 studies an important data-plane challenge, real-time resource management. Programmable routers provide an ideal platform for implementing the service-centric networks. Current programmable routers are based on the Network Processor (NP) technology, which utilizes multi-core system-on-a-chip architecture to achieve a good balance between programmability and performance. Several runtime task mapping approaches have been proposed for NP systems. I devised an evaluation methodology to compare these different designs. Based on queuing network abstraction, the methodology can systematically model the NP systems and analyze the performance of runtime task mapping algorithms under various network traffic scenarios.

Finally, Chapter 6 summarizes the contributions of this work, addresses future work, and concludes this dissertation.

Related work to each topic is discussed individually in each chapter.

CHAPTER 2

SERVICE-CENTRIC NETWORK ARCHITECTURE

Next-generation Internet will be governed by the need for flexibility. Heterogeneous end-systems, novel applications, and security and manageability challenges will require networks to provide a broad range of services that go beyond the simple store-and-forward capabilities of today's Internet. A service-centric network architecture is proposed in [112]. This next-generation Internet architecture design overcomes the flexibility constraints of the store-and-forward architecture design and expands the capabilities of the current Internet to meet its next-generation challenges.

This chapter gives a general introduction to the service-centric network architecture. First, the big vision of the whole architecture design is presented in Section 2.1, followed by a discussion about two important concepts in Section 2.1.1 and Section 2.1.2. A detailed introduction to the network architecture design is in Section 2.2. A communication abstraction framework design and two of its practical usage scenarios are then discussed in Section 2.3. This gives an example of how our service-centric network architecture could contribute to the design of the next-generation Internet. Finally, Section 2.4 gives a brief review about the related work, and Section 2.5 concludes this chapter.

2.1 Design Overview

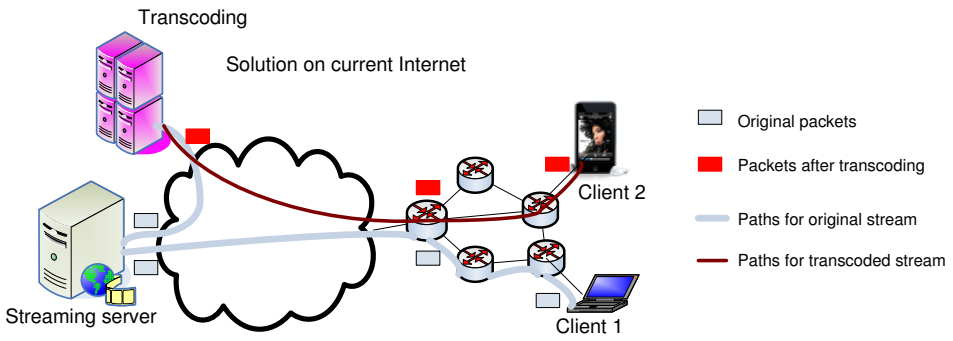
The primary design principle of the service-centric network architecture is that the network should focus on providing services instead of only forwarding data packets.

We believe that providing services should be the first-class networking functionality, while the transfer of data to the destination is secondary.

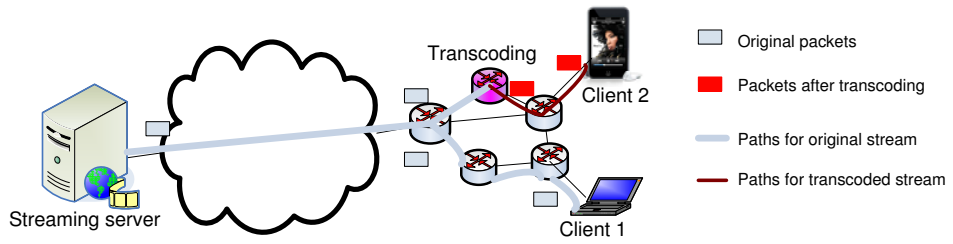
Instead of limiting processing to end-systems, service-centric networks utilize router-based processing functionality (i.e., programmable routers or visualized routers) to permit advance packet processing inside the network. These packet processing functionalities are distributed inside the network, in the form of network services. The purpose of these services is to implement the handling of data packets to achieve the desired functionality and performance properties of various communication requests. Using network services as the basic building blocks, the network then breaks a communication into a sequence of network services. By providing different composition of services along the data path, the network can customize connections to satisfy different communication requirements.

The potential of such a service-centric network architecture can be illustrated by a simple example in Figure 2.1. Assume we have an IPTV distribution service multicasting a HDTV stream to the subscribers. There are two different clients. Client 1 uses a receiver that is capable of displaying the source format (e.g., HD video stream in .wmv). Client 2 is a "thin" client who needs a transcoding (e.g., to convert the video into a lower definition in .mp4). The network systems that provide the required transcoding service are marked in pink.

In the current Internet (as shown in Figure 2.1(a)), the transcoding is mostly done on server farms that are connected to the network. In this case, two separate streams need to be sent. One is directly sent to client 1. Another is sent to the transcoding farm and then forwarded to client 2. However, in a service-centric network (as shown in Figure 2.1(b)), the transcoding is implemented as part of the communication infrastructure. For example, transcoding services are pre-deployed by the service providers on selected programmable routers. In this case, only one stream in the



(a) Solution on Current Internet



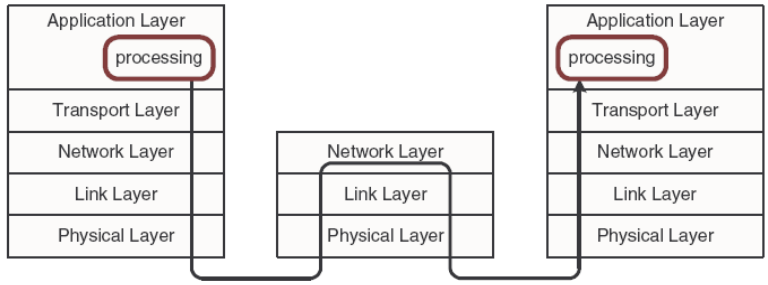
(b) Solution on Service-centric Networks

Figure 2.1. Potential of Service-centric Network Architecture.

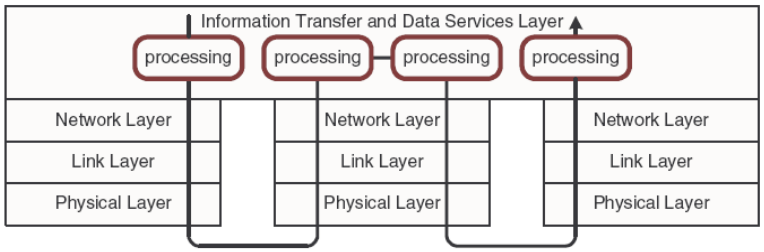
original format needs to be sent out, and the transcoding will happen transparently along the data path without the notice of the end-systems.

The above example shows that comparing to the current Internet design, the service-centric network architecture provides more flexibility (i.e., new services can be easily deployed inside the network), efficiency (less bandwidth and detour), and scalability (no centralized servers).

The difference in service stack between the current Internet architecture and the service-centric network architecture is contrasted in Figure 2.2. Instead of forcing the higher-layer processing to be limited to the end-systems, advance packet processing can take place throughout the network. In the service-centric network architecture design, traditional application layer and transport layer are merged into one layer, Information Transfer and Data Service Layer (ITDS layer). The functionality of the network layer remains relatively unchanged, except that advanced network-layer packet



(a) Current Internet Architecture



(b) Service-Centric Network Architecture

Figure 2.2. Protocol Stack: Current Internet vs. Service-Centric Networks.

processing functions could be deployed and enabled on router systems in the service-centric network architecture. The link and physical layers are shown unchanged in the figure since this work does not address these aspects of network architecture. A communication abstraction discussed in Section 1.4 provides a comprehensive and unified interface between the ITDS layer and the network layer.

Instead of viewing routers as simple electronic devices connecting end-systems, the service-centric network architecture regards the routers as an integral part of a distributed computing system. This design allows flexibility inside the network, and thus extends the capabilities of the network to meet its next-generation challenges.

2.1.1 Network Services

The most important concept of the service-centric network architecture is network services. We define the network services as:

any form of packet processing in the data plane and the related operations in the control plane.

Examples of some existing network services are:

- Reliability: buffering and acknowledgement-based retransmission of lost data as it is done in TCP.
- Privacy: encryption and decryption services between end systems (similar to SSL) or subnets (similar to VPN).
- Congestion Control: limiting of data transfer rate based on the state of the (sub-)network.
- Caching: storing of information for certain application layer protocols and making it accessible to other connections.
- Security: firewalling, intrusion detection, payload scanning and other mechanism to identify and mitigate attacks.
- Quality of Service: prioritized forwarding of data based on service requirements.
- Multicast: duplication and forwarding of data along multiple links and local retransmission for scalable, reliable multicast.
- Payload Transcoding: depending on the semantics of the transferred information, this service can adapt the content that is transferred. For example, large images from web pages can be down sampled for display on a cell phone. This service is very specialized and highly dependent on the transferred information and its coding in data.

Note that some of these services are implemented on end-systems in the current Internet (e.g., reliability service). However, in the service-centric network architecture, it is important that these services are also available inside the network since

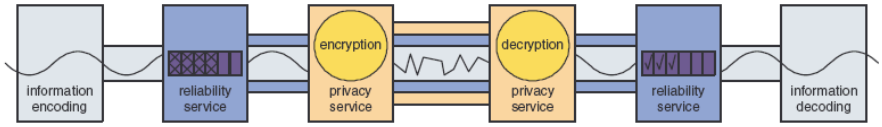
that provides flexibility in the network and expands the capabilities of the network to accommodate innovative applications (e.g., the reliability service can be used to reassemble a TCP stream on an intrusion detection system).

As networks need to handle more diverse end-systems (e.g., mobile devices, embedded systems) and novel networking paradigms (e.g., sensor networks, ad-hoc wireless networks), more and more data-path services are proposed. Some of the examples are:

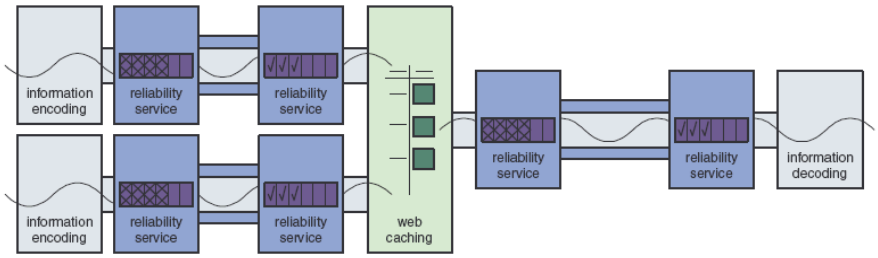
- IP traceback [92]: keeping track of the source of the traffic that was forwarded to help identify the sources of possibly malicious data flows.
- Content Adaptation: modifying the form of package content according to client profile. For example, advertisement insertion in web pages, and content transcoding for wireless clients [72].
- Flow-based routing: enabling partial network programmability by allowing the change of forwarding table in commercial routers. For example, OpenFlow [31, 79] enabled routers allow network administrators to define flows and their routing actions (e.g., drop the packets, forward the packets to a given port, or encapsulate and forward the packets to a controller).

2.1.2 Dynamic Service Composition

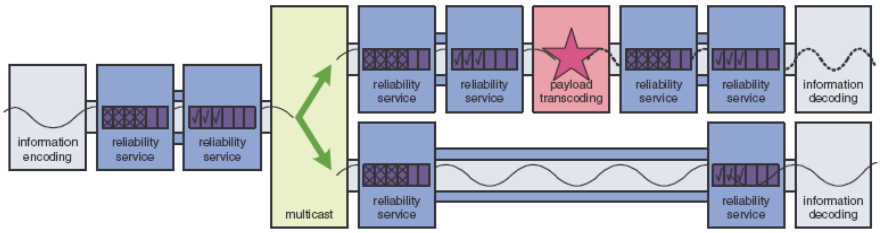
As mentioned in Section 2.1, the service-centric network architecture regards the routers as an integral part of a distributed computing system instead of viewing routers as simple electronic devices connecting end-systems. Advanced packet processing is provided inside the network in the form of basic building blocks, network services. These services could be developed, deployed, and enabled on selected routers by network service providers or network administrators. The flexibility of this network architecture design is further extended by allowing the dynamical composition of a variety of communication patterns from these building blocks.



(a) Example I: Reliable and Private Point-to-Point Communication



(b) Example II: Web Caching



(c) Example III: Content Distribution and Transcoding

Figure 2.3. Service Composition in Service-centric Networks.

In service-centric networks, an application (i.e., a program running at the ITDS layer) specifies the desired communication functionality and performance properties, and the network determines and provides the appropriate services to achieve these communication requirements.

Three examples shown in Figure 2.3 illustrate the service composition concept in the service-centric networks. The nesting of colored boxes indicates the layers of service combination.

Example I: Reliable and Private Communication. This type of communication can be composed from two network services that implement reliability and privacy. The combined service can then be applied to any packet that goes through the connection.

Example II: Web Caching. This type of communication can be achieved by combining a caching service with a reliability service. Multiple end-systems can use the same caching service and thus achieve the desired caching functionality.

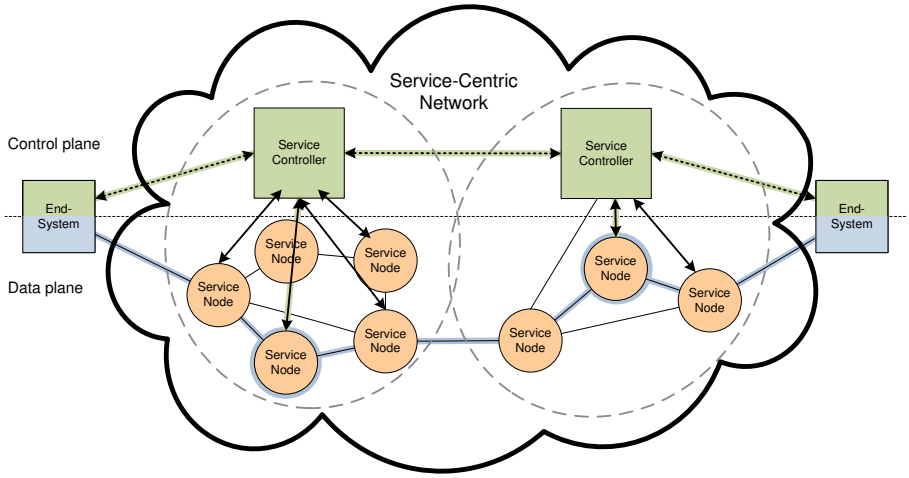


Figure 2.4. Service-Centric Network Architecture.

Example III: News Distribution. This type of communication uses multicast service (and reliability) service to reach a large number of end-systems. It may use a push-paradigm rather than having the end-systems request content individually. In addition, content transcoding service can be employed to provide adaptation to the physical channels characteristics.

Note that in most cases, there is an inherent sequential ordering for the services to be executed (e.g., intrusion detection needs to be performed after VPN decryption).

2.2 Network Architecture

The first question faced in building a service-centric network is:

> *How to structure the network internally to permit an effective and efficient implementation?*

A network architecture designed for the service-centric networks is discussed in [44]. The high-level design of the architecture is shown in Figure 2.4. The figure shows both control plane (on the top) and data plane (on the bottom).

A service-centric network consists of two types of network components: service nodes and service controllers. The service nodes are the programmable routers ca-

pable of providing network services. They are located on the data plane of the network. Service controllers are control-plane systems used to manage network resources (i.e., processing capability and link bandwidth), making routing decisions, and setup/teardown connections.

In this design, service controllers and service nodes are clustered into Autonomous Systems (ASs) because of two reasons: 1) network scalability and 2) to reflect the administrative structure of the current Internet. A service controller has the complete view about its own AS, including nodes, connectivity, link costs, service distribution, etc.

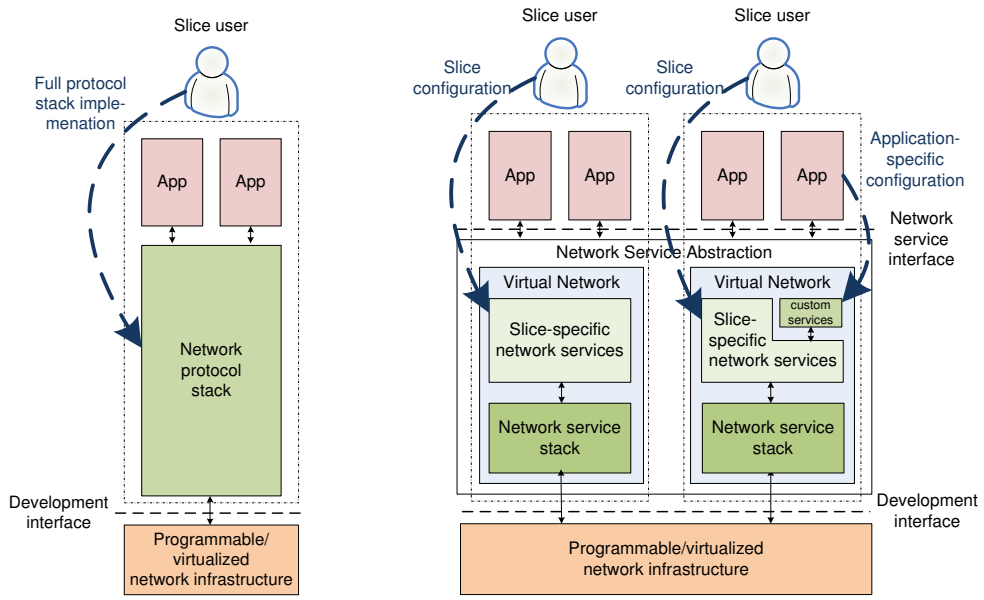
For simplicity, we used a two-level hierarchy in the example with one controller for each AS. Large-scale networks can be constructed by extending this basic architecture to a multi-level hierarchy or by including more than one service controller within the AS. More details about this network architecture design can be found in [44].

2.3 Network Service Abstraction

In this section, we propose a communication abstraction framework called network service abstraction. It provides a comprehensive and unified interface for the applications to fully utilize the capability provided by the service-centric networks. Note that the "application" here refers to the ITDS layer programs based on our architecture design.

Figure 2.5 shows an example on how this communication abstraction framework could be used to ease the development of new network services, protocols, and applications on programmable or virtualized network platforms.

In conventional design (shown in Figure 2.5(a)), the user of a programmable/virtualized platform needs to implement the full protocol stack in order to use the assigned slice. Note that we use the term "user" to mean the entity that sets up the slice, not necessarily the end-user who uses the applications that run on that slice. Providing a



(a) Conventional Virtualization.

(b) Service-Based Network Virtualization.

Figure 2.5. Comparison of Virtualization Approaches.

complete protocol stack implementation can be difficult to implement if novel networking functions are included. Thus, the existing approach requires a lot of technical understanding and time-consuming development.

With our network service abstraction (shown in Figure 2.5(b)), each slice contains a virtual network that permits the instantiation of network services to implement novel data path functions. The user provides only a slice configuration in form of a service specification (rather than a full network protocol stack). This configuration is used to instantiate slice-specific services, which are used for all communication between the applications within that slice. As illustrated on the right, it is also possible to instantiate additional custom services for some applications.

The main point of using network service abstraction is that a user interfaces at the level of service configurations rather than having to provide an entire protocol stack. The abstraction uses network service to specify the functionality of a virtual network slice. The services specified by a user are automatically applied to all connections in

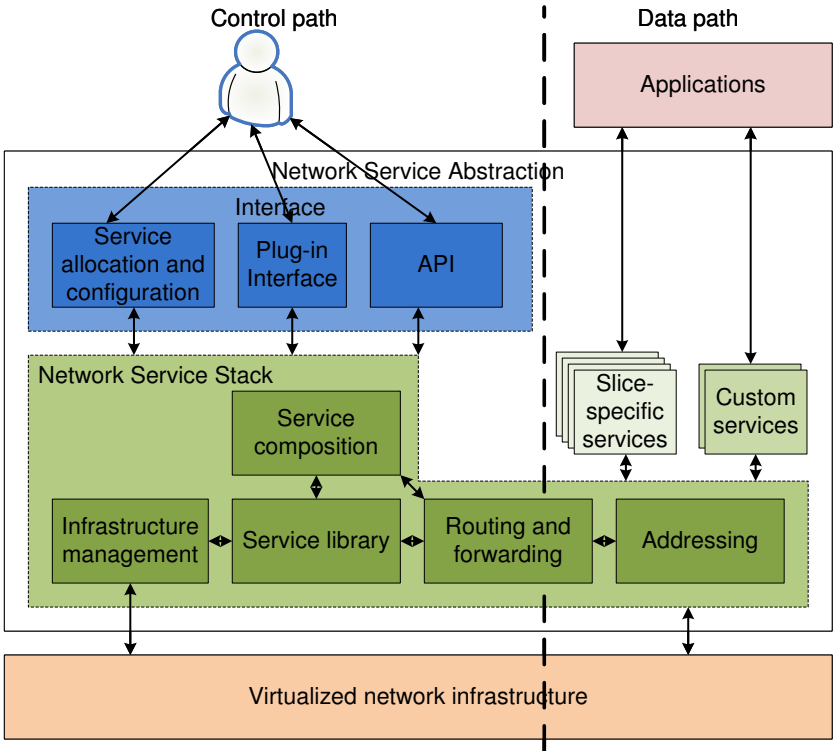


Figure 2.6. Network Service Abstraction.

the network. Thus, it is possible to easily deploy virtual network slices with novel functionality without having to develop complete protocol stacks. More details on the design of network service abstraction are provided in the following paragraphs.

This example also gives an idea on where our service-centric network architecture stands in the clean-slate design of the next-generation Internet.

Figure 2.6 shows the detail design of the network service abstraction. The abstraction contains two major components: service stack and interface. The network service stack is a simplified implementation of the service-enabled network. It has the properties of the service network architecture (supporting network services and service composition) and provides the basic network functionalities (e.g., addressing, routing, packet forwarding). The interface provides the user with a tool for interacting with the underlying network service stack and the virtualized infrastructure. Through the interface, users may configure the network as an instantiation of the net-

work service stack, deploy services on the network nodes, compose network services into communication networks and novel applications and even develop new services.

2.3.1 Network Service Stack

The network service stack consists of the following functional components:

- **Infrastructure management** is responsible for the virtualized network slice management, including link and processing resource management and loading network services onto selected nodes. We assume that all these functionalities are supported by the underlying virtualized infrastructure. The infrastructure management block only encapsulates these functionalities and provides them to the application developers as a part of the network service abstraction.

- **Service library** is a collection of pre-defined network services supported by our system. The default service library includes the network service blocks corresponding to IP protocol stack, as well as many widely used application protocols such as HTTP, FTP, SMTP, etc. The instantiations of the network service stack are by default configured as the IP network. That is, network nodes run IP protocols up to the network layer. This default configuration can be changed by user defining another service composition sequence for the network service configuration. Furthermore, users can also develop new network services and add them to the service library.

- **Service composition** is responsible for checking the validity of a given service composition sequence or automatically generating a valid service composition sequence according to the connection request and the services supported by the library. As will be described in Chapter 4, we reduce the service composition problem into a planning problem and use an existing planning tool, LPG, to validate or automatically generate the service composition sequence.

- **Addressing** is one of the basic functionalities of a network. It is used to identify different network systems and their interfaces. IPv4 is the default addressing scheme for our network service stack. However, any other applicable addressing scheme may also be developed and configured as the addressing scheme for an instantiation of the service network stack (i.e., the virtual network running on a virtual slice). This involves the use of the plug-in block and the service allocation and configuration block of the interface, which are discussed later in section 2.3.2.

- **Routing and forwarding** is another basic functionality of the network layer. As will be described in Chapter 3, routing in service network is much more complex than simply calculating the shortest path between two end systems. It has to guarantee both the required service availability along the route and the least cost. The default routing protocol for the network service stack is the service routing protocol that will be discussed in Section 3.6. Similar to addressing, users are allowed to add new routing protocols and set them as the routing protocols for an instantiation of the service network stack.

By using this set of functions, the network service stack is able to instantiate slice-specific or custom services in the data path of the network.

2.3.2 Interface

The interface is used by the user that creates the virtual network slice to specify the network functionality that is required for this slice. Interface components include:

- **Service allocation and configuration**: Through this interface, users can configure their network service instantiations, such as the addressing mechanism, routing protocol, the service composition sequence that is by default executed in the network, services supported by the network, and the nodes that have

these services installed on them. All these configurations are handled by the infrastructure management block of the network service stack.

- **Application Programming Interface (API)**: This interface defines the API supported by the virtual network slice. There are two types of communication requests: 1) a sequence of required services: this is in case the application needs to add new services besides the services provided by default by the instantiation of the network service stack, and 2) a description of the input/output stream: this is needed in some particular situations where the application cannot decide on the required services. Section 2.3.4.2 explains such a scenario with a detailed example. As described in [91], the first type of communication request is expressed as a service pipeline, which is capable of defining the sequence of services as well as where the service should be executed. In this case, the request is forwarded to the service composition block of the network service stack and be checked for validity. If it is valid, the composition block forwards it to the service controllers. The service controllers then run the configured routing protocol to setup a connection for the request. In [90], an expression for the second type of request is suggested in OWL format. Instead of describing the sequence of required services, the characteristics of the input/output stream is defined. In this case, the service composition block tries to generate a valid sequence of required services according to the request. If one exists, it forwards this sequence of services to service controllers to setup the connection. Otherwise, it returns an error as feedback to the users.

- **Plug-in Interface**: This interface enables users to develop new network services and add them into the service library. These network services can then be deployed onto selected service nodes and combined with other services on the data path to implement different communication networks and applications.

Furthermore, the users can deploy their own addressing mechanism or routing protocol through this interface and later configure them to be used by the instantiations of the network service stack. Note that when adding a new service to the library, the corresponding description and composition rules of the service should also be added into the service composition block so that the service can be correctly combined.

Through these interfaces, the user can control the functionality of the virtual network slice. As we show in Section 2.3.4, this interface is much simpler to use than having to develop a complete network stack from scratch as necessary in conventional virtualization approaches.

2.3.3 Deployment Model

A possible deployment model of our proposed network service abstraction is: The abstraction is distributed as open source to the network community. Users define a network topology, required network resources, and create a virtual slice on the virtualized infrastructure. After setting up the virtual slices, users install the abstraction on the virtual slice. Through the abstraction interface, users can either choose the default set of instantiated services or request custom services to be added to the default combination for that slice. If custom services are enabled, applications can request these services at connection setup time.

The design of the network service abstraction also follows the next-generation Internet model described by Turner and Taylor [103]. They claim that network virtualization will lead to a diversified Internet, where the role of conventional ISPs will be separated into three entities: Substrate Network Providers (SNPs), who build and maintain the virtualized network infrastructures, Meta-Network Providers (MNPs), who deploy and operate meta-networks, and Application Developers (ADs), who develop network applications for the end-users. In this model, SNPs can build meta-

networks spanning multiple substrate networks from different SNPs, and ADs can choose the most suitable meta-network from among those available, or they may design a new overlay network, tailored to their application's specific needs. In this model, our abstraction provides a good tool for the MNPs and ADs to interface with the SNP's virtualized infrastructure and ease their developing tasks.

2.3.4 Usage Scenarios

In this section, we use two examples to illustrate developing networks and applications on programmable/virtualized platforms through network service abstraction.

2.3.4.1 Secure Banking

The first example shows how to set up a secure network slice that could be used for secure banking. The network configuration requires a secure medium for all communication between end-systems (e.g., banking servers, customers, etc.). To provide security, all connections are encrypted and are monitored to detect intrusion attempts or denial of service attacks. Thus, an encryption/decryption service, an Intrusion Detection Systems (IDS) service, and an IP traceback service (to trace back attack traffic) could be used.

The deployment of this network would be achieved through the following steps: The user creates a virtual slice for secure banking on the virtualized infrastructure and installs the abstraction. The network is configured using the default setting (IP protocol stack and service routing protocol) through the service allocation and configuration interface, which in turn calls the infrastructure management block to download the services needed to form the IP stack and install them on routers and end-systems. This configuration means that these services are the default services for the packets going through this virtual network. The service library does not provide the required services for secure banking (IDS, IP traceback and encryption/decryption)

by default. Therefore, these services need to be added to the library through the plug-in interface.

Whenever any connection is set up in the network, the required services are instantiated. Secure banking requires a secure communication channel between the client and server. Thus, the encryption/decryption services need to be instantiated on the end-systems. IDS, however needs to be instantiated on a service node closest to the server so as to enable it to monitor all the traffic directed toward the server. It is important to note that the IDS service can also be instantiated on the server itself. IP traceback requires the service to be instantiated on all the routers in the datapath. Thus, a connection setup initiated by the client needs to be extended to contain the required services. According to [90], a request to the service controller to set up a connection to the server may be in the following form:

```
*:*>>ServerIP:ServerPort
```

The syntax of this service request is described in [91]: "`*:*`" denotes the end-system source sending traffic to a server with the provided address and port; "`>>`" indicates the sequence of services. When receiving this request, the mandatory services are added:

```
*:*>>encryption>>IPtraceback>>decryption>>
IDS>>ServerIP:ServerPort
```

Thus, all required services are forced onto each connection and thus the overall functionality of the slice is enforced. The service placement for any particular connection is then done by the service allocation and configuration block. After the connection setup, the end-systems can communicate securely.

2.3.4.2 IPTV Video Distribution

The second example deals with an IPTV video distribution service that provides customers with either live TV or video-on-demand (VoD) services. High quality

IPTV playback requires a receiver that is capable of processing large amounts of video data (e.g., personal computer, HDTV). Delivering content to customers with mobile devices (e.g. cell-phones, PDAs etc.) requires transcoding of the video (e.g., from HDTV-1080p (1920×1080) to H.264 (208×176)). Also, a multicast service and a quality-of-service (QoS) service need to be instantiated for scalable and high-quality distribution.

To deploy such a network, the user creates a virtual slice on the virtualized infrastructure, installs the abstraction, and configures the network using the default setting (IP protocol stack and service routing protocol) through the service allocation and configuration interface as above. Also, multicast, transcoding, and QoS scheduling services are added to the service library.

When setting up connections, multicast and QoS services need to be installed on all the routers in the data path. The transcoding service is installed on a few selected nodes that are along branches of the multicast tree leading to mobile clients. Multicast and QoS are set as mandatory services for every connection, while transcoding is set as an optional service. The application connection request

```
*:*>>multicast(Client1IP:Client1Port,
Client2IP:Client2Port)
```

is augmented to include required QoS and optional transcoding:

```
*:*>>QoS>>multicast(QoS>>Client1IP:
Client1Port,QoS>>transcode(1080p,H.264)>>
QoS>>Client2IP:Client2Port)
```

Using our service-based network virtualization system, a user can quickly and easily deploy a network that supports security, QoS, multicast, or any other services that can provide novel networking functionality.

2.4 Related Work

Network virtualization was first discussed in [6] to build a virtual testbed that can support multiple simultaneous architectures running on PlanetLab [1, 82]. The intent is to dramatically reduce the barrier to entry for new architectural ideas to be evaluated in practice and thus helps to overcome the internet ossification impasse. Turner and Taylor further extended the discussion in [103] and suggested that virtualization should be an architectural attribute of the next-generation Internet so that the problem of network ossification can be solved.

Since then, the networking research community has been working on implementing network virtualization. Some works focus on the virtualization technologies, e.g., separating user exclusion environments through containers [10], running separate guest operating systems [93], realizing the data plane virtualization through source code merging [66]. Virtualized testbed have been developed (e.g., GENI [78] and VINI [2]) to build a shared experimental platform for network researchers to broadly evaluate their protocol and application designs [13, 101, 102]. Other studies show how network virtualization helps to solve shortcomings in existing networks [21, 108] and how to contribute to the next-generation Internet architecture design [50, 80].

Much less effort has been dedicated to designing a suitable interface for virtualized infrastructures to ease network innovation and to facilitate the use of virtualized infrastructure by customers with limited networking expertise. The Emulab testbed [75] uses ns-2 scripts for topology and requirements specification, but does not support an easy way of adding custom protocol stacks and services as pertinent to the next-generation Internet. In our work, we present an abstraction framework that allows users to describe the functionality of their slice at a high level using network services. The network service abstraction uses the service description to set up a network with the desired functionality on top of a thin set of required functionality that is com-

mon to all slices. This abstraction is based on our work on networks with data path services [44, 53, 91, 112].

2.5 Summary

This chapter gives a brief description about the service-centric network architecture design. First, the "big vision" of the service-centric networks is presented and two most important concepts, network service and service composition, are described. Then, an network architecture for implementing such a service-centric network is introduced. Finally, a network service abstraction based on service-centric network architecture is introduced. It provides a comprehensive and extensible interface to ease the development of novel networks and applications on various programmable and virtualization platforms.

The work described in this chapter forms a general background for the future chapters. It shows the design and the potential of our service-centric network architecture and also gives an idea of where our service-centric network architecture design could fit in the picture of the next-generation Internet.

CHAPTER 3

SERVICE ROUTING

3.1 Introduction

Routing has been widely studied in conventional networks and several centralized and distributed algorithms are widely deployed. In the context of service-centric networks where services are provided in the data path, routing becomes a much more complex problem that goes beyond simply determining the shortest path between two nodes. Services that are required for an end-to-end connection may be available only on some service nodes that are not located along the shortest end-to-end path. In such a case, it is necessary to determined through which service nodes traffic should flow. When more services are necessary, the problem becomes increasing difficult.

In this chapter, we address this routing problem faced in the service-centric networks. We answer one of the most fundamental routing problems in the context of service-centric networks:

> *How can we achieve optimal or near-optimal routing of connection requests that involve data-path services?*

This problem is illustrated in Figure 3.1, where a connection that requires two services (Service 1 and Service 2 should be executed along the path in order) is shown. Several nodes are potential candidates for handling each service. Here, different colors are used to represent nodes providing different services. As illustrated in the figure, the traditional shortest path (i.e., the shortest path between the two end systems) may not be able to provide the required services (i.e., the purple path does not pass

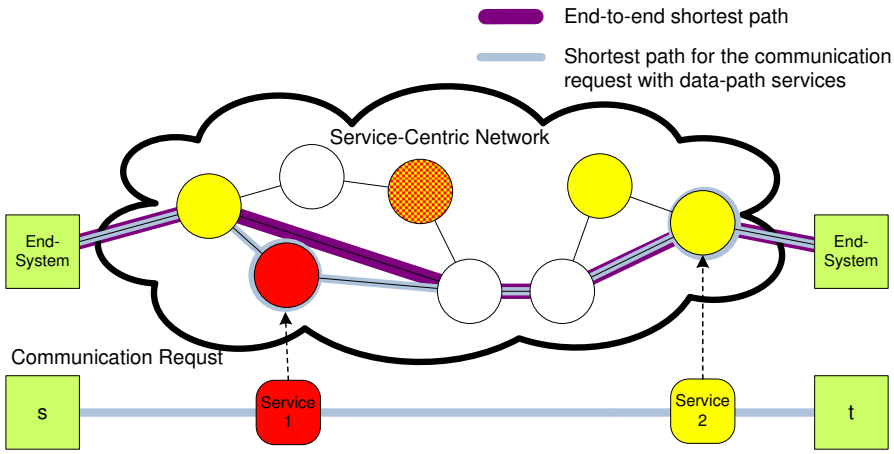


Figure 3.1. Routing Problem in Service-centric Networks.

through a node that is capable of providing service 1). Thus, some detour is needed to satisfy the service requirement, and we want this detour to be optimized in terms of cost. That is, the goal is to find optimal (e.g., least-cost) path through the network such that the required services are performed *in order* along the path.

In this context, we need to determine how this problem can be solved algorithmically, and how a protocol can efficiently implement this solution.

The specific contributions in this chapter are:

1. A formal definition of the routing problem encountered in networks with data-path services, which is called "service routing problem" in this work.

2. A global algorithm for solving the service routing problem with resource constraints.

3. A decentralized algorithm for solving the service routing problem in a scalable way.

4. A scalable routing protocol for implementing service routing in large-scale networks.

5. An extensive evaluation of the performance of the decentralized algorithm and the routing protocol on a prototype implementation on Emulab.

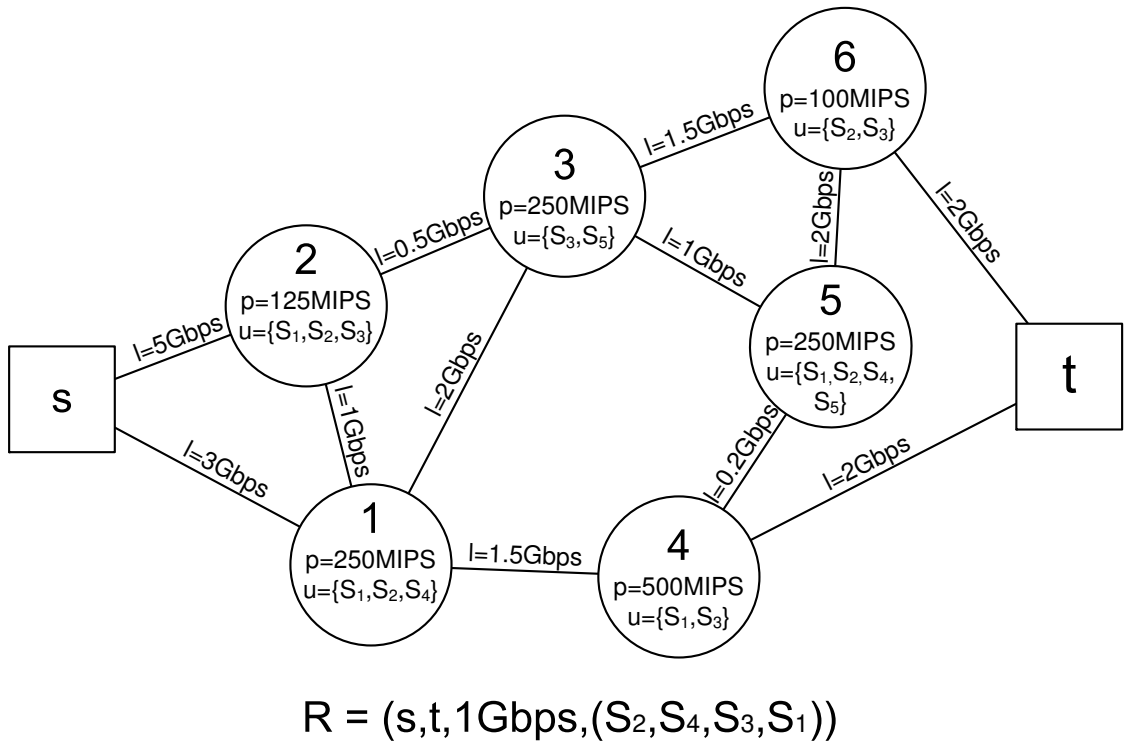
In this chapter, Section 3.2 discusses and formalizes the service routing problem in networks with data-path services. Section 3.3 and Section 3.4 introduce both global and decentralized routing algorithms to solve this problem. Based on these algorithms, Section 3.6 describes the design of a service routing protocol. Section 3.5 and Section 3.7 analyze and compare the different algorithms and evaluate the performance of proposed routing protocol using the results from our Emulab-based prototype implementation. A brief review about the related work is in Section 3.8, and Section 3.9 concludes this chapter. Parts of this chapter are published in [52], [54], and [53]

3.2 Service Routing Problem

Routing traffic that requires various services through a network needs to consider and balance the tradeoff of two dimensions: (1) Service: to find a valid path. That is, a path from source to destination with capability of providing required services in order along the path, and (2) Cost: to find the optimal path among all the valid paths. The definition of optimal is described in more detail later in this section.

This problem, which we call the *service routing problem*, is defined formally as follows:

The *network* is represented by a weighted graph, $G = (V, E)$, where nodes V correspond to routers and end systems and edges E correspond to links. Each edge $e_{i,j}$ that connects nodes v_i and v_j is labeled with a weight $w_{i,j}$ that represents the communication cost (e.g., delay). Each node v_m is labeled with the set of services that it can perform $u_m = \{S_j | \text{service } S_j \text{ is available on } v_m\}$ and the processing cost $c_{m,j}$ (e.g., processing delay) of each service. A *connection request* is represented as $R = (s, t, (S_{j_1}, \ldots, S_{j_k}))$, where s is the source node, t is the destination node, and



R = (s,t,1Gbps,(S_2,S_4,S_3,S_1))

Figure 3.2. An Example of the Service Routing Problem.

$S_{j_1}, \ldots, S_{j_k}$ is an ordered list of services that are required for this connection. An example of the service routing problem is shown in Figure 3.2.

Given a network G and a request R, we need to find a path for the connection such that the source and destination are connected and all the required services are processed along the path. The path is defined as $P = (E^P, M^P)$ with a sequence of edges, E^P, and services mapped to processing nodes, M^P: $P = ((e_{s,v_{i_1}}, \ldots, e_{v_{i_h},t}), (S_{j_1} \rightarrow v_{m_1}, \ldots, S_{j_k} \rightarrow v_{m_k}))$. To determine the quality of a path, we define the total cost $C(P)$ of accommodating connection request R as the sum of link cost and processing cost: $C(P) = \left(\sum_{\{(x,y)|e_{x,y} \in E^P\}} w_{x,y}\right) + \left(\sum_{\{(j_i,m_i)|S_{j_i} \rightarrow v_{m_i} \in M^P\}} c_{m_i,j_i}\right)$.

In many cases, it is desirable to find the *optimal* connection setup. In our work, we evaluate the optimality in terms of the least cost allocation of a single connection request. We assume that cost for communication and cost for service processing can be represented using a single metric (e.g., delay).

In the context of this problem statement, we make several assumptions: 1) We assume once link and processing resources are reserved, they are dedicated to the connection for which they were allocated, 2) We assume that the type and order of services is known at connection setup time and does not change over the lifetime of the connection, and 3) We assume that cost for communication and cost for service processing can be represented using a single metric (e.g., delay).

It has been shown in [26] that even the problem of finding the optimal solution for a single connection in a capacity-constrained network can be reduced to the traveling salesman problem, and thus is NP-complete. This means that no deterministic, polynomial-time algorithm can be found for the service routing problem with resource-constrains. In this case, the only feasible solution is to use a heuristic algorithm that can find a near-optimal solution.

Broadly, routing algorithms can be classified into two categories: global algorithms and decentralized algorithms. A global algorithm computers the least-cost path between a source and destination using complete, global knowledge about the network (e.g., nodes, connectivity, and link costs). While in a decentralized routing algorithm, the calculation of the least-cost path is carried out in an iterative, distributed manner. No node has complete information about the costs of all network links.

In the following sections, global algorithms are first introduced in Section 3.3 as the solutions for service routing problem, since they are more intuitive. Then, Section 3.4 moves on to the more complicated decentralized algorithms.

3.3 Global Algorithms

In this section, we introduce three global algorithms that find approximate solutions to the service placement problem in capacity-constrained networks: (1) *randomized placement*, (2) *layered graph*, and (3) *service step search*. We have chosen these three algorithms for our discussion and evaluation because randomized place-

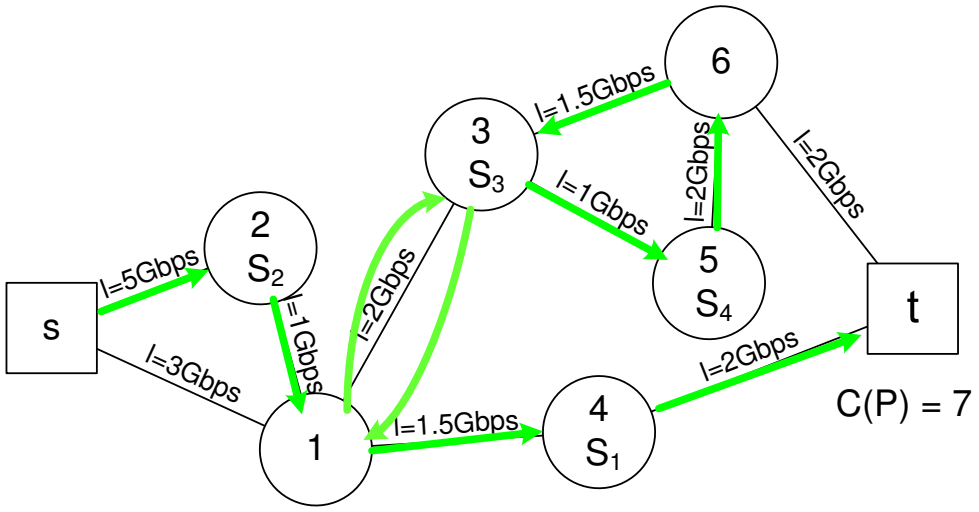


Figure 3.3. Randomized Solution for the Problem in Fig. 3.2.

ment represents the naïve approach to solving the problem, layered graph represents the current state of the art in network service placement, and service step search is our novel contribution to solve practical problems that occur in the layered graph algorithm.

3.3.1 Algorithm I: Randomized Placement

The randomized placement algorithm is a very simple approach to service placement. As indicated by the name, nodes where processing is performed are chosen randomly among the nodes that can provide the requested service. Then, the shortest path between these nodes is determined to connect the entire path. If a node does not have sufficient processing capacity or if no path with sufficient link capacity can be found to that node, the random node selection process for that service is repeated. The algorithm terminates when a valid path is found or if the random selection fails a predetermined number of times. Figure 3.3 shows the solution that the randomized placement algorithm finds for the example problem. In the figure, links with insufficient capacity are omitted. Service placements are noted on the nodes. Note that the solution is likely to be different every time the placement is repeated.

The benefit of randomized placement lies in its simplicity. Random choices for processing nodes can be computed easily and quickly. To determine the shortest path between these nodes, a simple shortest path algorithm (e.g., Dijkstra's algorithm) can be used. Another benefit of randomized placement is that randomization avoids the problem of systematically getting stuck in a non-optimal solution. The main drawback is that the randomized placement is completely oblivious to the quality of its choices. However, it has been shown that repeated randomized placement can lead to solutions that converge on the optimum [65].

The computational complexity of the randomized placement algorithm for a single request with k services in a network with $|V|$ nodes and $|E|$ edges is $\mathcal{O}(k\,(|E| + |V| \log |V|))$. This is derived from k iterations of the shortest path algorithm, which has a computational complexity of $\mathcal{O}(|E| + |V| \log |V|)$.

3.3.2 Algorithm II: Layered Graph

The layered graph algorithm attempts to find a lowest cost path by combining communication cost and processing cost into a single graph structure, called layered graph. Then, a simple shortest path algorithm is run on the structure to determine the best sequence of communication and processing steps. The construction of the layered graph is done as follows: a total of $k + 1$ copies of the original network are used to represent the graph layers. The top layer (layer 0) is used for communication before service 1 is performed. The next layer (layer 1) represents communication that is performed after service 1 (and before service 2) is completed. Layer 0 and 1 are connected with vertical edges on all nodes where service 1 processing is possible. The costs of these edges correspond to the processing costs of service 1 on the nodes that they connect. The layering and connecting with vertical edges is continued until all $k + 1$ layers are in place. Then, using Dijkstra's algorithm, the shortest path from the source node's instance in layer 0 to the destination node's instance in layer k is

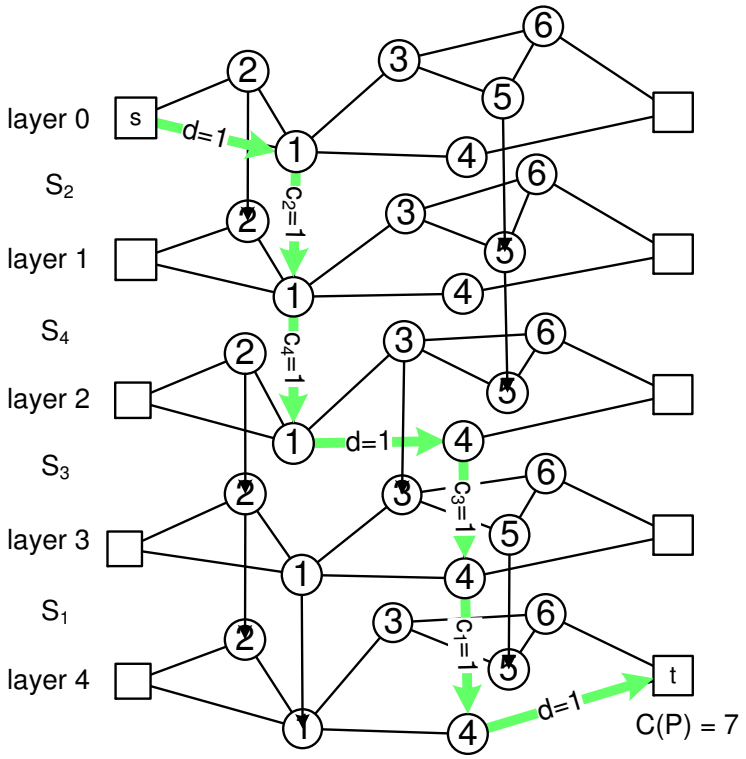


Figure 3.4. Layered Graph Solution for the Problem in Fig. 3.2.

determined. This resulting path provides the least cost connection from the source to the destination while ensuring that all services are performed in sequence (due to vertical edges). To obtain the final path in the original network, all nodes and edges are projected onto a single instance of the network. Vertical edges in the path correspond to service placements and horizontal edges are used to connect the path. The layered graph for the example problem is shown in Figure 3.4. This algorithm was described by Choi et al. in [26].

The benefit of the layered graph algorithm lies in its ability to find the least-cost path while considering all service requirements. While this appears to be an ideal solution, we explain in Section 3.3.4 that it shows limitations when considering link and processing capacity constraints. Thus, it may not be able to find a suitable path for a connection request at all.

The computational complexity of the layered graph algorithm is simply that of Dijkstra's algorithm on the layered graph. Since it contains $k+1$ times as many nodes and edges as the original graph, the complexity is $\mathcal{O}(k|E| + k|V| + k|V| \log(k|V|))$.

3.3.3 Algorithm III: Service Step Search

We introduce a new approach called service step search to address some of the shortcomings of the layered graph algorithm. The idea behind service step search is to build a service step search graph that is structured according to the sequence of services that are requested. In the service step search graph, all nodes that can perform a particular service are placed into one level. Nodes in one level are fully connected to all nodes in the next level (i.e., the next service). The weights of these edges correspond to the cost of the shortest path between the corresponding nodes. The service step search graph for the example problem is shown in Figure 3.5. To determine the least-cost path in the network, Dijkstra's shortest path algorithm is run across the service step search graph. This shortest path algorithm is slightly modified to: (1) consider the processing cost of each service node along the path in addition to link cost, and (2) consider the resource constraints (i.e., the processing and link capacities that may have already been committed to earlier steps in the path).

The benefit of the service step search lies in its ability to consider that processing and link capacities may have already been committed to earlier steps in the path. Thus, it solves one of the key problems of the layered graph algorithm. The drawback of the algorithm lies in the higher computational complexity due to the multiple instances of shortest path computations for constructing the service step search graph.

The computational complexity of service step search is dominated by the cost for constructing the search graph, which requires $k \cdot |V|$ shortest path tree computations, and thus is $\mathcal{O}(k|V||E| + k|V|^2 \log |V|)$. The actual search for the best path can be done in $\mathcal{O}(k|V|^2 + k|V| \log(k|V|))$.

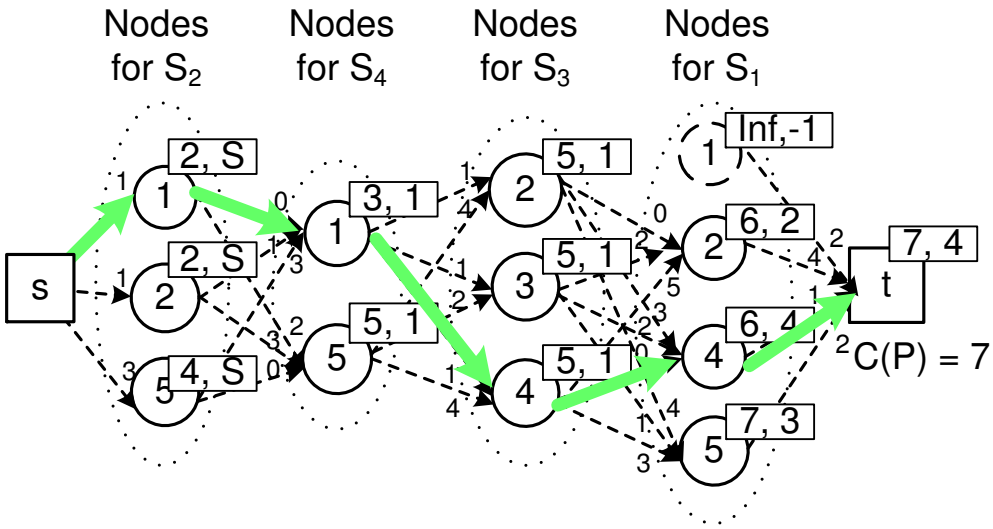


Figure 3.5. Service Step Search Solution for the Problem in Fig. 3.2.

3.3.4 Capacity Constraints

Link and processing capacity constraints are the key consideration for solving the service placement problem. The capacity limitation can occur at two instances: (1) when links or nodes do not have sufficient capacity for a particular connection *before* the placement algorithm is run and (2) when capacity becomes insufficient *while* the algorithm is run.

The first case can be dealt with easily. Before attempting to map connection request R, all links $e_{i,j}$ that do not have sufficient bandwidth (i.e., $l_{i,j} < B$) are (virtually) removed from the network. Similarly, the set of available services u_i for node v_i is adjusted such that service S_k are (virtually) removed if the node does not have sufficient processing capacity for processing the service with required bandwidth (i.e., $p_i < B \cdot z_{i,k}$). The remaining graph thus contains only links and processing nodes that can accommodate request R.

However, it is also possible that the available resources run out while an algorithm is computing the best path (e.g., when reusing a link or a processing node). In such cases, the connection allocation may fail. This issue of capacity constraints has been

explored in [25] and is partially addressed by the service step search algorithm. This algorithm can consider which nodes and links have been used along the (so far) shortest path and thus exclude the constrained links and nodes when calculating for the following steps. As shown in Figure 3.5, when searching the node for S_1, node v_1 is ignored. This is because the shortest paths from source s to all the available nodes for service S_3, map S_2 and S_4 both on v_1. The processing capability of v_1 is thereby used up. Thus, the service step search algorithm can avoid oversubscribing a node or link. However, it may still not be able to find the optimal solution under all circumstances. Nevertheless, our results in Section 3.3.5 show that service step search performs considerably better than layered graph due to this improvement.

3.3.5 Evaluation Results

With an understanding of the placement problem and the three algorithms that we consider, we turn to the question of how well these algorithms perform, and what are the tradeoffs of using different algorithms to solve the service placement problem.

3.3.5.1 Evaluation Metrics

To evaluate algorithms from the perspective of an individual connection, we consider the following metrics: (1) Successful Connection Establishment: This binary metric considers if the algorithm was able to determine a valid path. As explained earlier, the connection establishment may fail due to resource constraints or due to the inability of the algorithm to find a path. (2) End-to-End Delay: This metric reflects the quality of the path found. Shorter paths and more powerful service processing nodes lead to lower end-to-end delay. This metric is equivalent to the cost of the path, $C(P)$, as described above. In our experiments, we use millisecond (ms) as the unit of the delays.

From the perspective of the entire system, we consider the following metrics: (3) Link Resource Usage: This metric reflects how much network link bandwidth is used

by an algorithm to accommodate a set of connection requests. This metric is measured in bandwidth (Mbps) times link delay (ms). (4) Processing Resource Usage: This metric reflects the usage of processing power measured in MIPS. It considers the processing load balancing, i.e., how the required processing power is distributed on each individual service node. The more evenly load is spread across all available service nodes, the more balanced the algorithm performs. Instead of condensing load balance into a single metric, we compare the cumulative distribution function of each algorithm.

3.3.5.2 Evaluation Setup

We use simulation to evaluate performance of the three mapping algorithms. We use two different network setups to highlight different scenarios:

- Network 1 (not resource constrained): In this configuration, we use 96 nodes (divided into 12 ASes of 8 nodes each) with a choice of 4 different services. The Inter-AS link delays are set as 10ms, and the Intra-AS link delays are set as 2 ms. A total of 10,000 connection requests is tested on this network. For each request, the bandwidth is chosen randomly to be between 10 Mbps and 100 Mbps. The source and destination nodes as well as service requirements are also chosen randomly. The bandwidth for each link in the network is configured such that there is just enough link resource for accommodating the corresponding shortest-path connections. The processing capacity for each node is configured such that there is just enough processing resource for all 10,000 connections evenly distributed across all nodes capable of providing a particular service.

- Network 2 (resource constrained): In this configuration, we use again 96 nodes, but the network is divided into 2 subnets (6 ASs in each subnet) such that service s_1 is provided only in subnet 1 and service s_2 is only provided in subnet 2 (no other services are provided). The two subnets are connected by 192

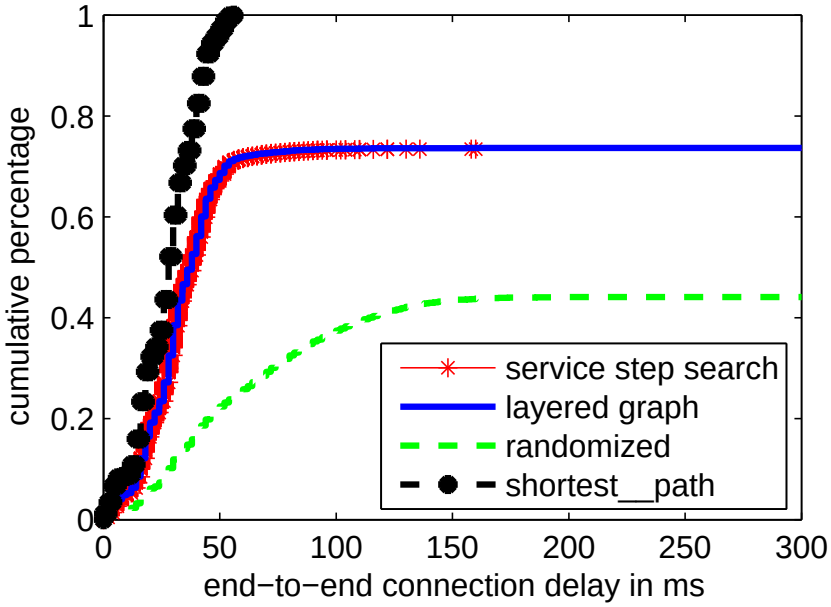
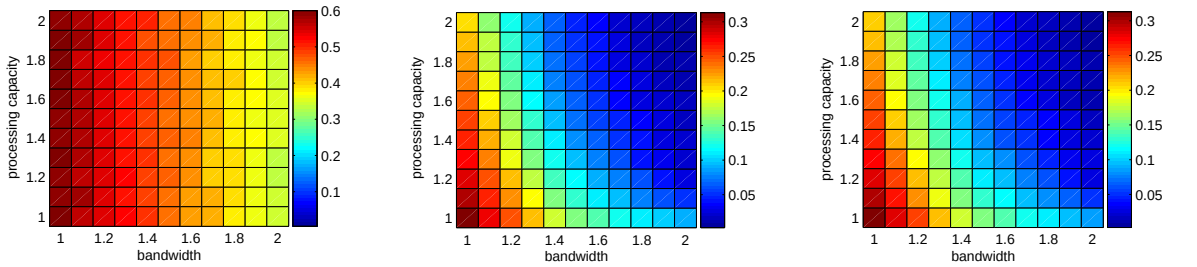


Figure 3.6. CDF of Connection End-to-End Delay on Network 1.

parallel "bottleneck" Inter-AS links, whose bandwidth capacity is set at 100 Mbps. While the Intra-AS delay and Inter-AS delay within the same subnet are set as 2 ms and 10 ms, we set the delay for each of these bottleneck links to be slightly more than 100 ms. This setup is carefully chosen to illustrate the capacity constrained scenario where layered graph is forced to traverse the same bottleneck link twice during a connection setup while the link only has the bandwidth of supporting once. A total of 200 connection requests are tested on this network.

3.3.5.3 Performance Results

We first compare the performance of the three algorithms in terms of connection drop rate and end-to-end connection delay on Network 1. In Figure 3.6, we show the cumulative distribution function of connection delay for all three algorithms in comparison to the shortest path. Clearly, allocating services along a path increases the delay, no matter what algorithm is used. Since the bandwidth resource are lim-



(a) Randomized Algorithm (b) Layered Graph Algorithm (c) Service Step Search Algorithm

Figure 3.7. Connection Drop Rate on Network 1.

ited to the maximum requirement for shortest path, we can also observe that some connections are dropped (y-axis cutoff). The randomized algorithm can only successfully allocate around 40% while layered graph and service step search algorithms can successfully allocate around 70%. This shows the (expected) inefficiencies in choosing service nodes randomly.

To consider how over-provisioning of link and processing resources changes the drop rates, consider Figure 3.7. This figure shows the connection drop rate of 10,000 connection requests for different levels of over-provisioning of link bandwidth and processing capacity. The over-provisioning factor (shown on the x-axis for bandwidth and on the y-axis for processing capacity) ranges from 1× to 2×. The color indicates the fraction of connections dropped (e.g., 0.3=30%). As expected, increasing either the bandwidth or the processing capacity helps to reduce the connection drop rate. However, it is apparent that for all three algorithms, increasing bandwidth decreases connection drop rate much faster than increasing processing capacity. That is, bandwidth has more impact on successfully allocate connections than processing capacity. Furthermore, the figures again show that randomized algorithm gives higher connection drop rate than the other two algorithms.

To illustrate the benefits of service step search over layered graph (which have both performed equally well in the non-resource-constrained case), we consider Network 2. Figure 3.8 shows the delay and flow drop rate for this scenario. Clearly, layered

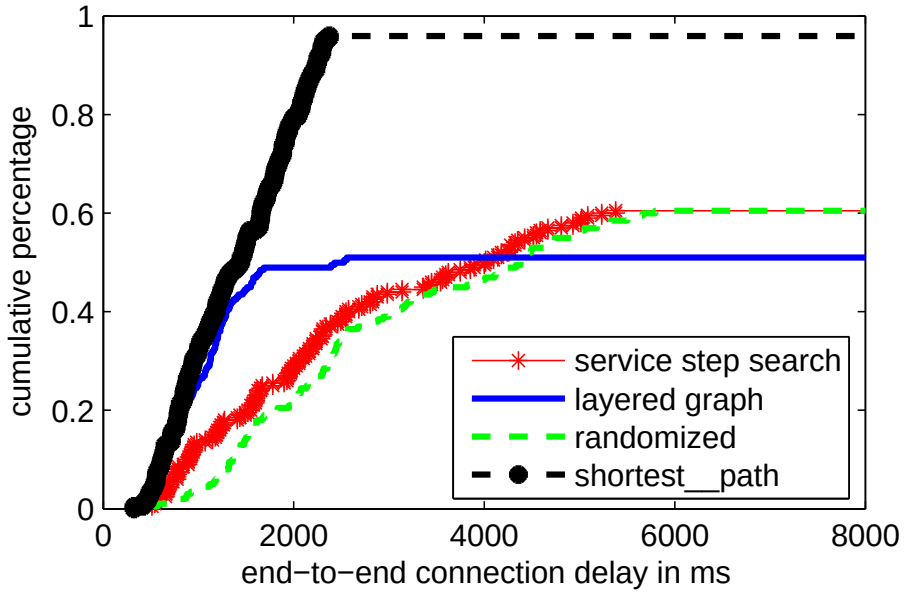


Figure 3.8. CDF of Connection End-to-End Delay on Network 2.

graph performs more poorly than the other two algorithms. This is due to resource bottlenecks that are discovered during connection setup and cannot be resolved by layered graph. In contrast, the service step search and randomized algorithms perform better. Looking more carefully at the number of dropped connections over the number of connection attempts, we see in Figure 3.9 that layered graph drops connections from the very beginning. The other algorithms start dropping when network resources are exhausted.

In terms of systems metrics, Figure 3.10 shows the link resource usage over an increasing number of successful connections for all the three algorithms as well as a shortest path baseline. As expected, the randomized placement algorithm requires most link resources to connect service nodes. This is due to the random choice of service nodes that does not consider the overall path cost. The service step search algorithm performs better and uses less link resources. The layered graph requires the least link resources among all algorithms, but as discussed before is limited in its ability to find valid paths for connections.

Figure 3.9. Dropped Connections over Connection Attempts on Network 2.

Figure 3.10. Network Usage over Successful Connections on Network 2.

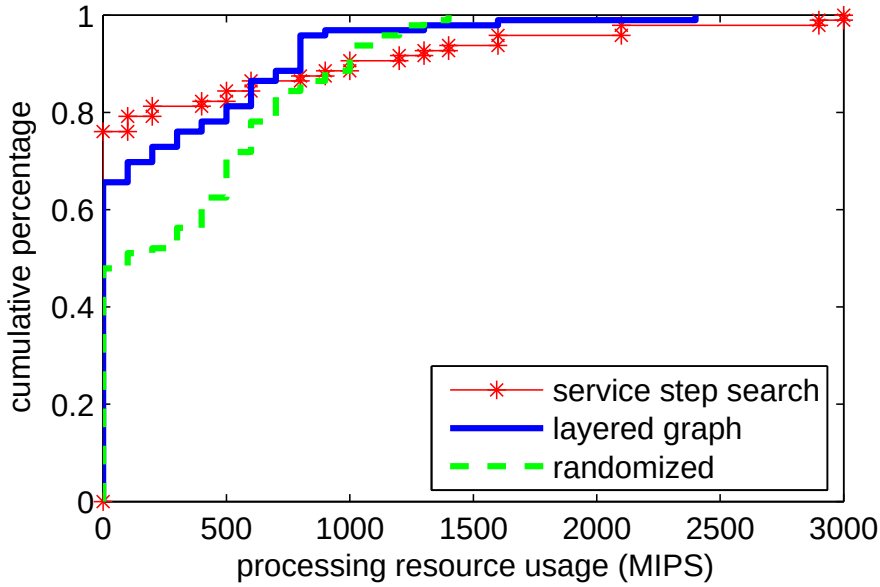


Figure 3.11. CDF of Processing Node Utilization on Network 2.

When considering processing load on service nodes, Figure 3.11 shows the cumulative distribution function of the load. We observe that randomized algorithm achieves the most balanced load, followed by layered graph and service step search. The imbalance in the latter two algorithms is due to nodes on the shortest path being chosen first before other nodes are used. From the figure, we can see that nearly 80% of the nodes are idle for the service step search, 63% for the layered graph and 45% for randomized algorithm.

The above results show that service step search provides the best mapping result when considering both unconstrained and constrained networks. However, the improvements come at the price of higher computational complexity. The results in Table 3.1 show the time it takes to place 10,000 connection requests with 0-4 services in Network 1 and 200 connections with 2 services in Network 2. Both layered graph and service step search are considerably more computationally demanding than randomized mapping. Service step search is the most complex algorithm among these three.

Table 3.1. Time to Compute Path for Connection Request.

Algorithm	Computation Time	
	Network 1	Network 2
Randomized placement	2.83 s	40 ms
Layered graph	13.14 s	480 ms
Service step search	29.30 s	700 ms

Table 3.2. Qualitative Comparison of Algorithms.

Algorithm	link usage	drop rate	e2e delay	running time	balance
Randomized placement	-	-,+	-	+	+
Layered graph	+	+,∘	+	∘	∘
Service step search	+	+	+	-	-

3.3.5.4 Evaluation Summary

In this section, we have presented and evaluated three global algorithms for solving the service routing problem in networks with data-path services: The randomized algorithm represents the naïve approach to solving the problem, layered graph represents the current state of the art in network service placement, and service step search is our novel contribution to solve practical problems that occur in the layered graph algorithm.

The results show that the service step search performs best in terms of finding low delay path and minimizing connection drop rate. The results also show that there is a clear trade-off between the quality of mapping and the runtime of the algorithm, as shown in Table 3.2. While service step search performs well in most of the scenarios, it has the highest running time. Randomized placement, while performing lower in terms of link resource usage and end-to-end delay, may still be a choice to consider due to its fast run time and ease of implementation.

3.4 Decentralized Algorithms

A fundamental shortcoming of the global algorithms is that it needs a global and complete view about the whole network (e.g., the nodes, connectivity, and link costs)

to calculate the optimal path of a request with data-path services. This limits their scalability and makes it unrealistic to implement them in large-scale networks. To address these issues, we develop a *decentralized* routing algorithm and its approximation as novel approaches to solve the service routing problem in large-scale networks.

In this section, we first present the general concepts of the two decentralized algorithms - the Distributed Service Matrix Routing (DSMR) algorithm and the approximate DSMR algorithm. Later, we explain the working of the algorithms in more detail.

3.4.1 Service Routing as Dynamic Programming Problem

In decentralized routing algorithms, nodes do not have a complete view about the whole network. Instead, each node begins with only the knowledge of the costs of its own directly attached links. Then, through an interactive process of calculation and exchange of information with its neighboring nodes, a node gradually calculates the least-cost path to a destination or set of destinations.

As illustrated in Figure 3.12, to solve the service routing problem in a decentralize way, a node v needs to decide two things: (1) Select path. That is, to decide which neighbor node should be its next-hop node, and (2) Select service. That is, to decide whether to provide service on itself, and what services to perform on itself before it forwarding the packets to the next-hope node.

The decentralized routing algorithm needs to answer two things: (1) How does a node make the best routing decision? and (2) What routing information need to be exchanged between nodes so that the nodes can choose the optimal path? The tricky part here is that the node should not know the complete information (e.g., nodes, connectivity, link costs) about the whole network. Each node should be able to make the right decision by only having some abstract information about the world beyond itself.

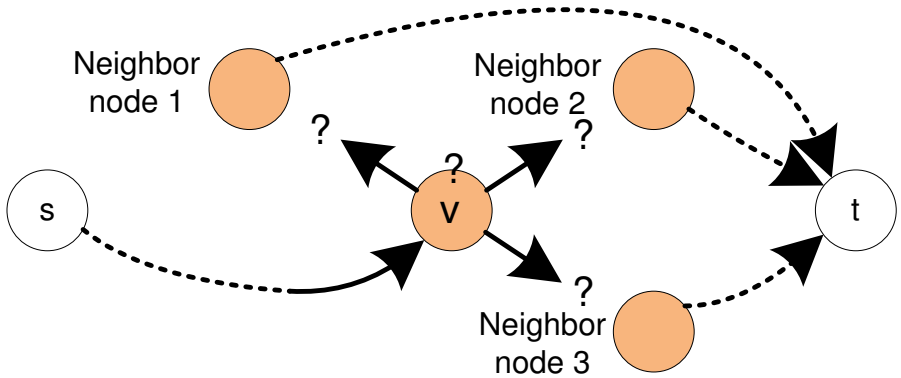


Figure 3.12. Decentralized Service Routing.

Distance Vector is a well-known decentralized routing algorithm, which is widely used in the current Internet to find the shortest path between two end systems. The difference of service routing problem from the shortest path problem is that instead of only considering the path cost we also need to consider a second dimension, the services. Following the idea of the Distance-Vector (DV) Routing Algorithm, we have discovered that a solution to this problem. The basic idea is to use a dynamic programming approach similar to what Bellman proposed for shortest path routing [14]. Let $c_v^{j_1,\ldots,j_k}(t)$ denote the cost of the shortest path from node v to node t where services $S_{j_1},\ldots,S_{j_k}$ are performed along the way. For shortest path computation (i.e., no services), we use the notation $c_v^-(t)$. Thus, a node v can determine the least-cost path by considering to process i $(0 \leq i \leq k)$ services and forwarding the request to any neighboring node n_v $(n_v \in \{x \in V | e_{v,x} \in E\})$:

$$c_v^{j_1,\ldots,j_k}(t) = \min_{0\leq i\leq k}\left(\sum_{l=1}^{i} c_{v,j_l} + \min_{n_v}\left(w_{v,n_v} + c_{n_v}^{j_{i+1},\ldots,j_k}(t)\right)\right). \tag{3.1}$$

The argument i on the right side determines how many of the k services that need to be performed should be processed on node v. Note that if $i = 0$, no service is processed, i.e., $\sum_{l=1}^{i} c_{v,j_l} = 0$. If $i = k$, all the services will be processed on node v, i.e., $c_{n_v}^{j_{i+1},\ldots,j_k}(t) = c_{n_v}^-(t)$. The argument n_v determines to which neighbor of v the remaining request should be sent.

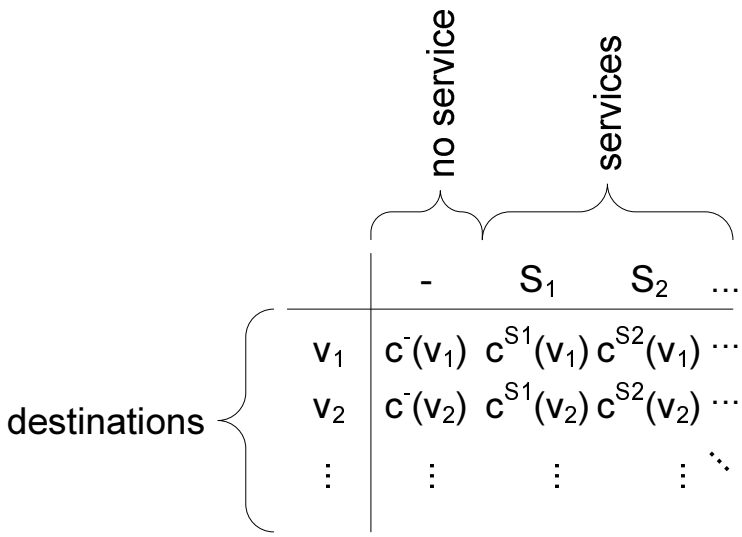


Figure 3.13. Service Matrix.

3.4.2 DSMR Algorithm

Following the above idea, we can design a distributed algorithm to solve the service placement problem. We structure our discussion by distinguishing two important steps of the algorithm: (1) routing information exchange, where information pertinent to routing and service placement is exchanged between nodes, and (2) request routing, where the routing decisions for connection requests are made by nodes based on the information exchanged by step (1).

3.4.2.1 Routing Information Exchange

In this procedure, nodes exchange their $c_v^*(\star)$ values ('$*$' represents any service sequence and '$\star$' represents any destination) with neighboring nodes (similar to how it is done in RIP for distance vector routing [51]).

To capture this information, we define a control plane data structure called "service matrix" (see Figure 3.13). This matrix is a two-dimensional extension of a distance vector, where the second dimension is an enumeration of all possible services. As in distance vector routing, the first dimension is a list of destinations (or destination prefixes).

At the initialization stage, each node builds up the matrix based on the knowledge it possesses about itself (e.g., services available, processing capacity, link delays to neighbors etc.). Next, each node advertises its service matrix to its neighbors. In this way, all nodes can collect all $c_n^*(*)$ values from their neighbors and thus solve Equation 3.1 to populate their own service matrices and service tables. The structure of the service table of a node is the same as the service matrix. However, except $c_n^*(*)$, the least-cost, an entry of the routing tables also records the value of i and n_v that correspond to the least cost. Respectively, these two values indicate the services that need to be processed on node v and the next-hop node. This collect-update-broadcast procedure continues till every node finally get all the advertisements from other nodes and the matrices of all the nodes converge to a consistent state.

3.4.2.2 Request Routing

When handling connection routing and setup, a request $R = (s, t, (S_{j_1}, \ldots, S_{j_k}))$ is propagated from s to t. A node v along the path needs to determine from its service matrix which services $S_{j_1}, \ldots, S_{j_i}$ should be processed locally. This can be done simply by looking up $c_v^{S_{j_1},\ldots,S_{j_k}}(t)$. Rather than having to compute Equation 3.1 for each connection request, we assume that the control plane precomputes $c_v^*(*)$ and the corresponding number of local services i and next hop n_v.

Using a simple lookup into the service matrix, a node can determine that it should allocate i services to itself and pass a request for the remaining services along to its neighbor n_v (i.e., $R' = (s, t, (S_{j_{i+1}}, \ldots, S_{j_k}))$). By the time the request reaches t, all services (except for those that may be optimally placed onto t) are allocated to nodes along the path.

3.4.2.3 Practical Constraints

The DSMR algorithm provides the globally optimal path and service allocation. However, it requires that all possible service sequences $S_{j_1}, \ldots, S_{j_k}$ are listed in the

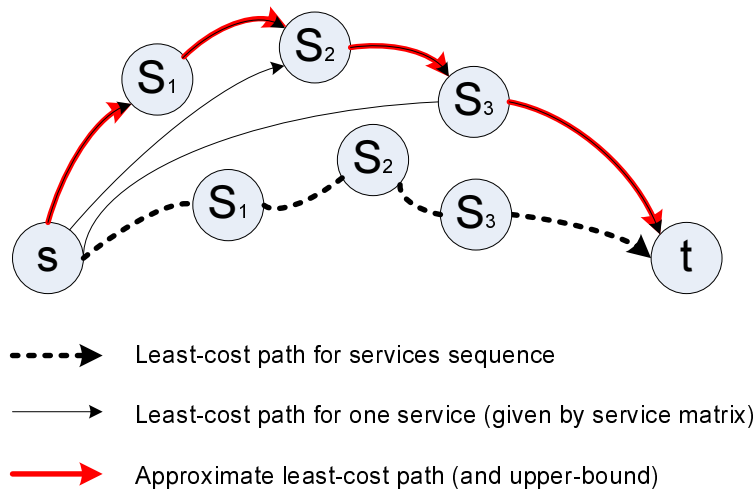


Figure 3.14. DSMR Approximation and Path Bounds.

service matrix, which is impractical for large network implementation. Suppose there are $|S|$ different services available in a network and all of them could be combined arbitrarily to form a connection request. If the maximum number of services in any request is k_{max}, then a total of $\mathcal{O}\big(|S|^{k_{max}}\big)$ columns would be required in the service matrix. Clearly, that does not provide a level of scalability necessary for large networks.

To address this problem, we discuss an approximate solution to the service placement problem, where only the routing information for no services or single service (i.e., $|S| + 1$ columns in the service matrix) are necessary.

3.4.3 Approximate DSMR

To reduce the size of the service matrix, we can use an approximation that requires only information about the optimal placement of single services, no matter what sequence of service are requested. The idea is illustrated in Figure 3.14 for a sequence of three services.

The approximate DSMR algorithm processes the sequence of services from request $R = (s, t, (S_{j_1}, \ldots, S_{j_k}))$, from back to front. First, the optimal path between s and t for S_{j_k} is determined. Since S_{j_k} is a single service, its optimal path and allocation

Table 3.3. Time and Space Complexity of Algorithms.

Algorithm	Time (route lookup)	Space
Layered graph	$\mathcal{O}(k(\lvert E\rvert + \lvert V\rvert)\log(k\lvert V\rvert))$	$\mathcal{O}(k \cdot (\lvert E\rvert + \lvert V\rvert))$
DSMR	$\mathcal{O}(1)$	$\mathcal{O}\big(\lvert S\rvert^{k_{max}} \cdot \lvert V\rvert\big)$
Approx. DSMR	$\mathcal{O}(k)$	$\mathcal{O}(\lvert S\rvert \cdot \lvert V\rvert)$

to node v_{m_k} can be determined from the reduced service matrix, which contains only $|S|+1$ columns. This step corresponds to placing S_3 in Figure 3.14. With S_{j_k} placed, the same service matrix can now be used to determine where to place $S_{j_{k-1}}$ along a path from s to v_{m_k}. This step corresponds to placing S_2 in Figure 3.14. The process is repeated until all services have been placed.

Using this approximate placement algorithm, a node can compute an upper bound for the optimal path cost $C(P^{opt})$: The upper bound path, P^+, is the path that is determined by the approximate DSMR algorithm. Its cost is the sum of all paths for individually placed services minus the redundant portions of the path. Thus, $C(P^+) = \sum_{i=1}^{k-1}\left(c_s^{S_{j_i}}(v_{m_{i+1}}) - c_s^-(v_{m_{i+1}})\right) + c_s^{S_{j_k}}(t)$. Since $c_s^{S_{j_i}}(v_{m_{i+1}}) > c_s^-(v_{m_{i+1}})$, the sum within the expression yields a positive cost.

An important question is how tight the upper bound is, since it determines the quality of the approximate DSMR algorithm compared to the optimal solution (provided by either DSMR or the layered graph algorithm). This question will be evaluated and answered in Section 3.7.2.

3.5 Analytical Comparison of the Algorithms

The time and space complexity of the three algorithms, centralized layered graph, DSMR, and approximate DSMR, are compared in Table 3.3. The layered graph algorithm requires a significant amount of time to perform a route lookup since it needs to create the layered graph on demand and compute the least-cost path. In contrast, DSMR requires only constant time to perform a lookup since the service matrix contains the optimal path for all possible destinations and service combina-

tions. However, the data structure size for DSMR is prohibitively large. Approximate DSMR strikes a good balance between lookup time and space requirement with lookup times proportional to the number of services. The service matrix data structure is also small and proportional in size to the number of services and destination nodes. This comparison, again illustrates the benefits of using a distributed algorithm over a centralized algorithm, but also highlights the need for using approximate DSMR over the original DSMR to limit space requirements.

3.6 Scalable Routing Protocol

In this section, we introduce the routing protocol designed to facilitate route selection for service-centric networks. Our work is based on the service-centric network architecture design shown in Figure 2.4 (Section 2.2). For scalability consideration, the architecture adopts the concept of the hierarchical network, where service nodes are grouped into Autonomous Systems (AS). Each AS has a service controller and a set of service nodes. The controller has a complete view of its topology.

The protocol can be divided into two different levels: Intra-AS routing, which determines the path within the AS, and Inter-AS, routing which routes between ASs. For scalability, we use decentralized algorithms (described in Section 3.4) to decide the Intra-AS routing. We use global algorithms (described in Section 3.3) to calculate the Inter-AS routing, since the global algorithms consider resource constrains when calculating the "optimized" routing and achieve better solutions than the decentralized algorithms in most of the scenarios (As shown later in Section 3.7).

We make the following assumption for the connections in our work: (1) The network supports the establishment of connections with fixed paths. This is required since traffic needs to traverse the a particular set of service nodes as determined by our algorithm, and (2) Services are performed in a fixed sequence that is specified at connection setup time and will not change over the life-time of the connection.

Despite these necessary assumptions, the problem remains general enough to be representative of scenarios that occur both in the current Internet as well in potential future networks.

Next, we move on to the design details of the routing protocol. We divide the discussion into three different stages:

1. DSRP Setup: describes how nodes register to their local service controller and how controllers collect information about topology and services offered within their AS.

2. DSRP Routing Information Exchange: describes how service matrices are initialized, exchanged and updated.

3. DSRP Connection Setup: describes how a request for connection by an end-system is translated into the actual setup of a path.

To keep the initial design of the protocol simple, we make a few reasonable assumptions: (1) every node knows the IP address of its local service controller, (2) every AS has a globally unique ID (AS_ID), (3) all nodes register at the DSRP Setup stage, and (4) all information on the control plane is transmitted using a reliable communication channel (e.g. TCP).

3.6.1 DSRP Setup

As illustrated in figures 3.15 and 3.16, the DSRP setup stage consists of the following steps.

1. The node (e.g., node A in AS1) sends a registration request to its local controller.

2. On receipt of the registration request, the local controller assigns a locally unique ID (node_ID) to the node, and replies to the request with the (AS_ID, node_ID) pair.

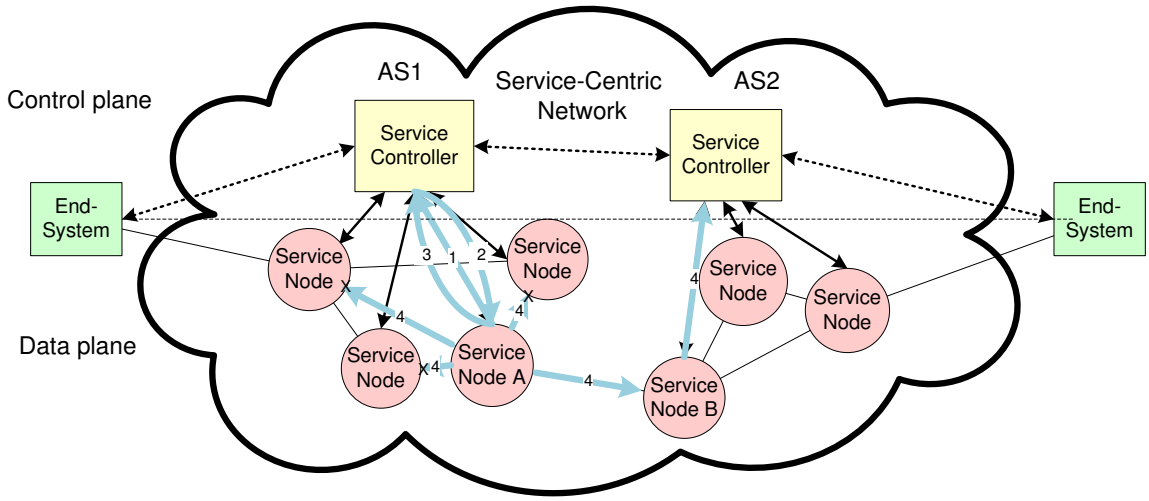


Figure 3.15. DSRP Setup.

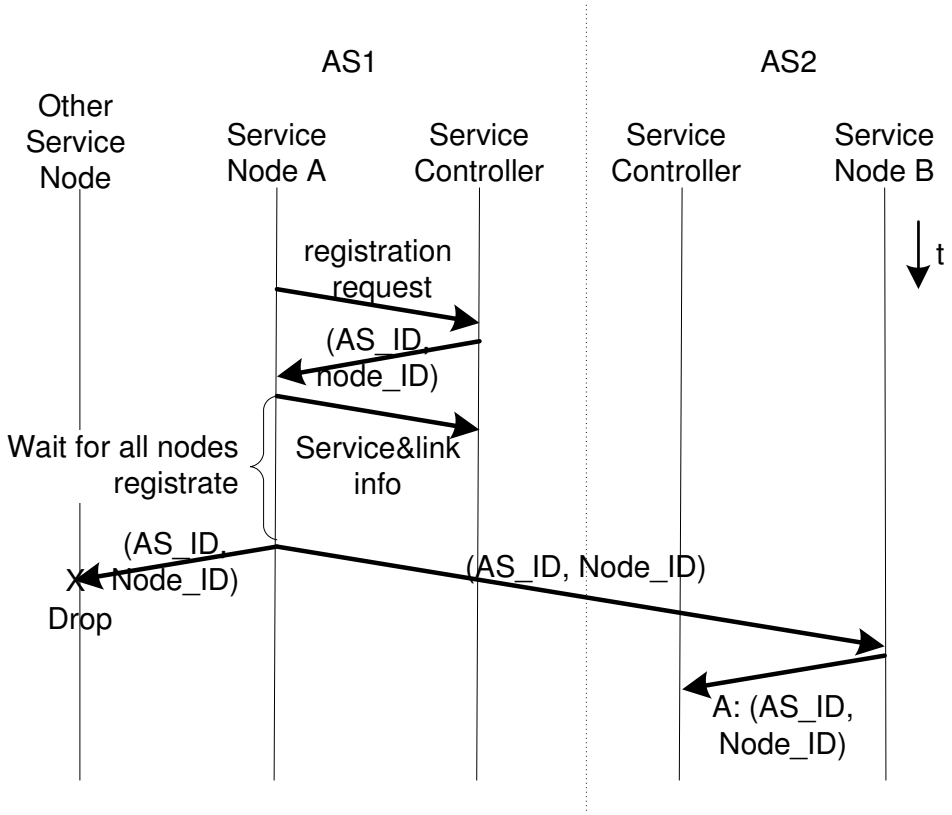


Figure 3.16. Space-Time Diagram for DSRP Setup.

3. The node then sends information about all its interface connections (the IP address pairs of all its links), the services it offers, its processing capacities etc. to its local controller.

4. Finally, the node exchanges its (AS_ID, node_ID) pair with all its neighbors. Information about neighboring nodes from different ASs are passed on to the local controller. This enables the controller to know about the connections to its neighboring ASs.

At the end of the DSRP setup stage, the service controllers have a complete view of their AS's topology and know exactly what services are provides on each node within their AS. In addition, they have also learnt about the nodes connecting to neighboring ASs.

3.6.2 DSRP Routing Information Exchange

After the setup stage, each node probes its neighbors to obtain the corresponding link delays. In addition, the nodes constantly check their CPU utilization to get an estimate of the time to process each of their available services. All this information is periodically reported to their controllers, which now have enough information to run the centralized layered graph algorithm.

The distributed Inter-AS routing is the main challenge and contribution of DSRP, which makes use of the DSMR or the approximate DSMR. The following text describes the initialization, exchange and update of the service matrices.

The service matrix is initialized using the centralized layered graph algorithm after the service controller has enough information about its AS.

After initialization, the service controllers send an OPEN message to establish a TCP connection between each of their neighboring controllers (i.e., service controllers belonging to neighboring ASs). This TCP connection is used to exchange service matrices at the end of pre-defined exchange intervals (e.g., 30 seconds). On receipt

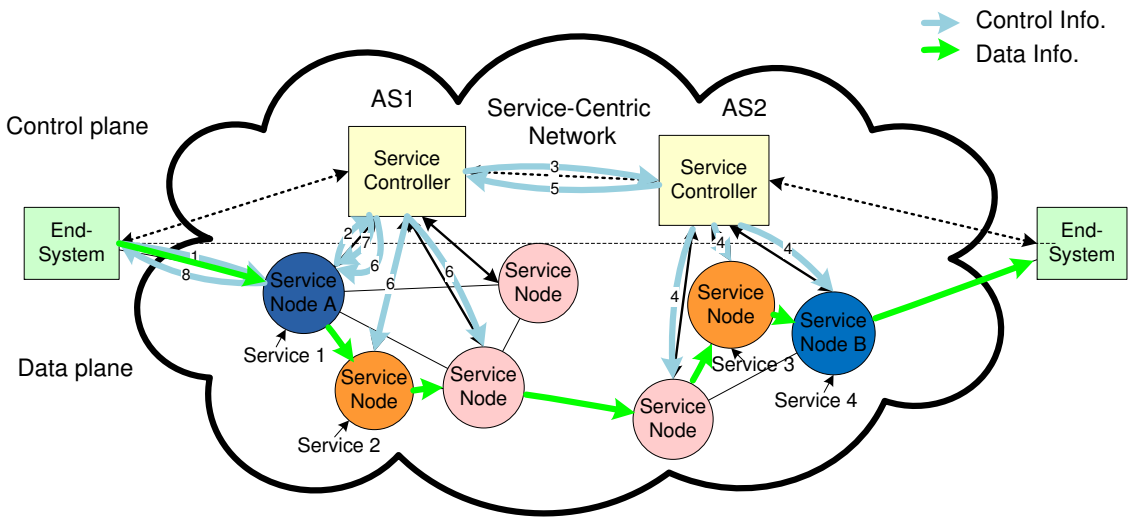


Figure 3.17. DSRP Connection Setup.

of a service matrix from a neighbor, the service controller recalculates a new service matrix. The current matrix is updated only when it is different from the new one. At the end of the exchange interval, the service matrix will be exchanged using an UPDATE message only if it was updated during this current interval. All the service matrices eventually converge to a stable state and the routing information across the network becomes consistent.

When a service controller detects any change in its AS (e.g., a new node comes up or an existing link goes down or its connection with a neighboring AS changes, etc.), its service matrix is updated accordingly. This change is sent to all the neighboring service controllers who once again recalculate their matrices and exchange them if they get updated. This change too will eventually get propagated through the network.

3.6.3 DSRP Connection Setup

The procedure for connection setup in a service-centric network is illustrated in figures 3.17 and 3.18. When an end-system requests the use of network services, the service node (Node A) to which it is connected sends a connection request to its local service controller (also called source AS controller). On receipt of a connection

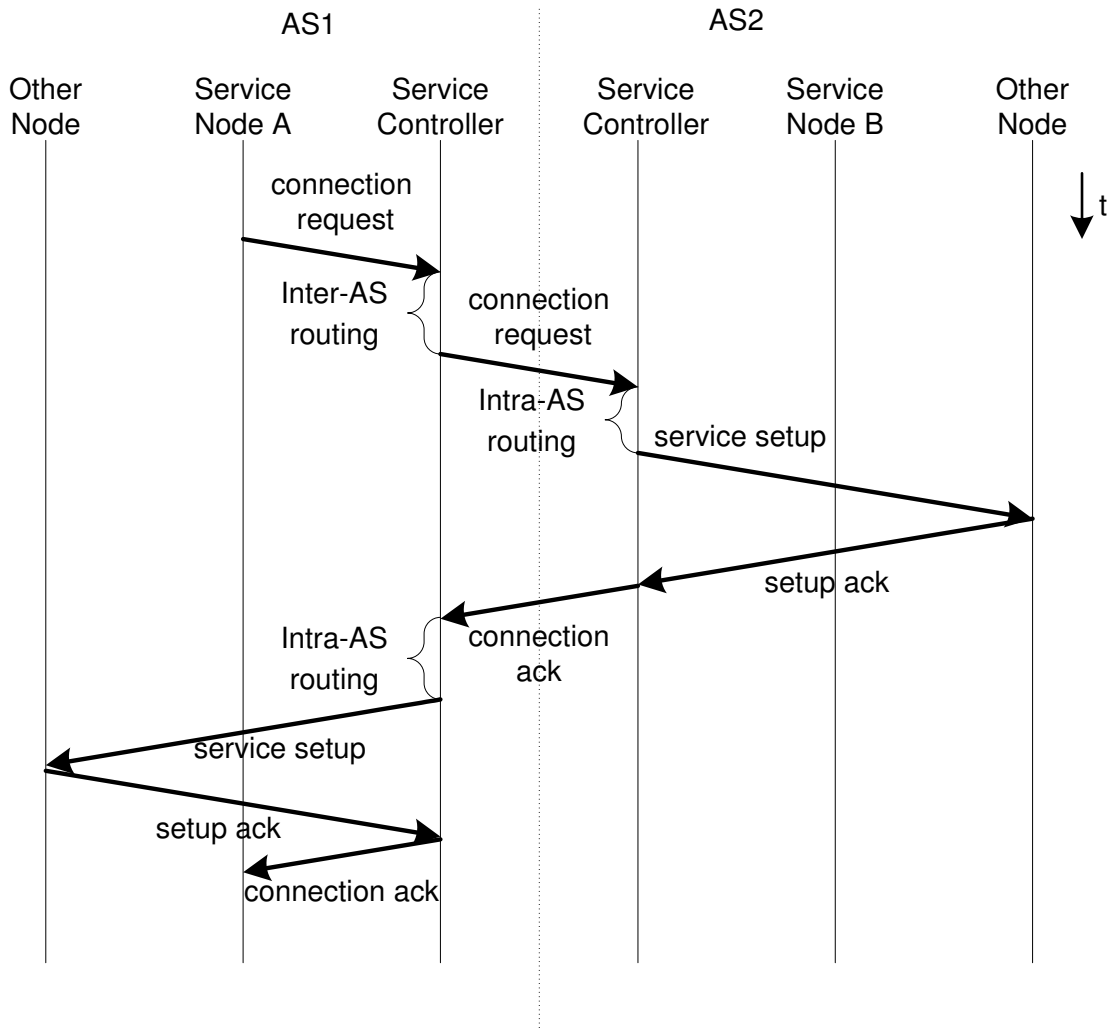


Figure 3.18. Space-Time Diagram for DSRP Connection Setup.

request, the controller immediately looks up the corresponding entry in the service matrix to determine the services to be performed within its AS and the next AS to which the remaining services are to be passed. This procedure is repeated at the next hop AS controller too, until the destination AS controller is reached. The destination AS controller performs the Intra-AS routing (centralized layered graph algorithm) to determine the local path within the AS and the mapping of services to the local service nodes. The local service nodes along the selected path are notified and hop-by-hop UDP connections are established between them. That is, each node is informed what service(s), if any, it needs to perform on the data and to which IP address and port number the (processed) data is to be sent. After setting up this path, the destination AS sends a connection acknowledgement to the (previous) controller, which sent the request. Upon receiving the acknowledgement, the previous AS controller too performs the Intra-AS routing, sets the path up within its AS, and sends a connection acknowledgement to its previous controller. This procedure continues till the connection acknowledgement reaches the source AS controller. The source AS controller too sets up its local service path, and the end-system is finally informed of the successful establishment of the connection through an acknowledgement. This marks the end of the setup stage. The end-system can now send its data traffic through the hop-by-hop UDP connection established and the data will be processed accordingly before reaching the intended destination.

3.7 Results and Analysis

In this section, we evaluate the performance of the approximate DSMR and demonstrate the scalability and efficiency of DSRP.

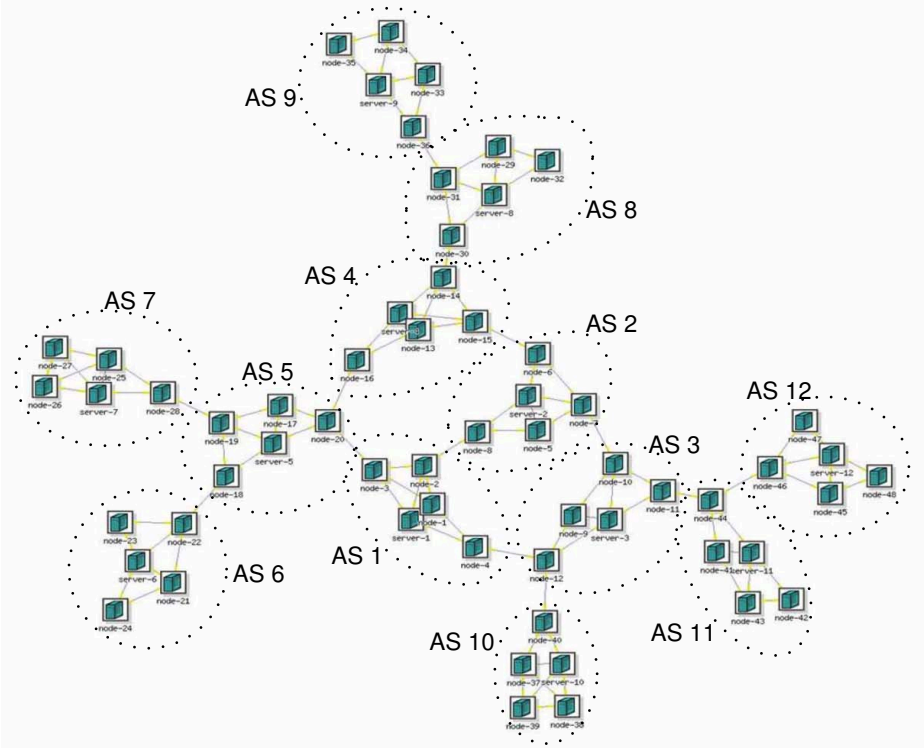


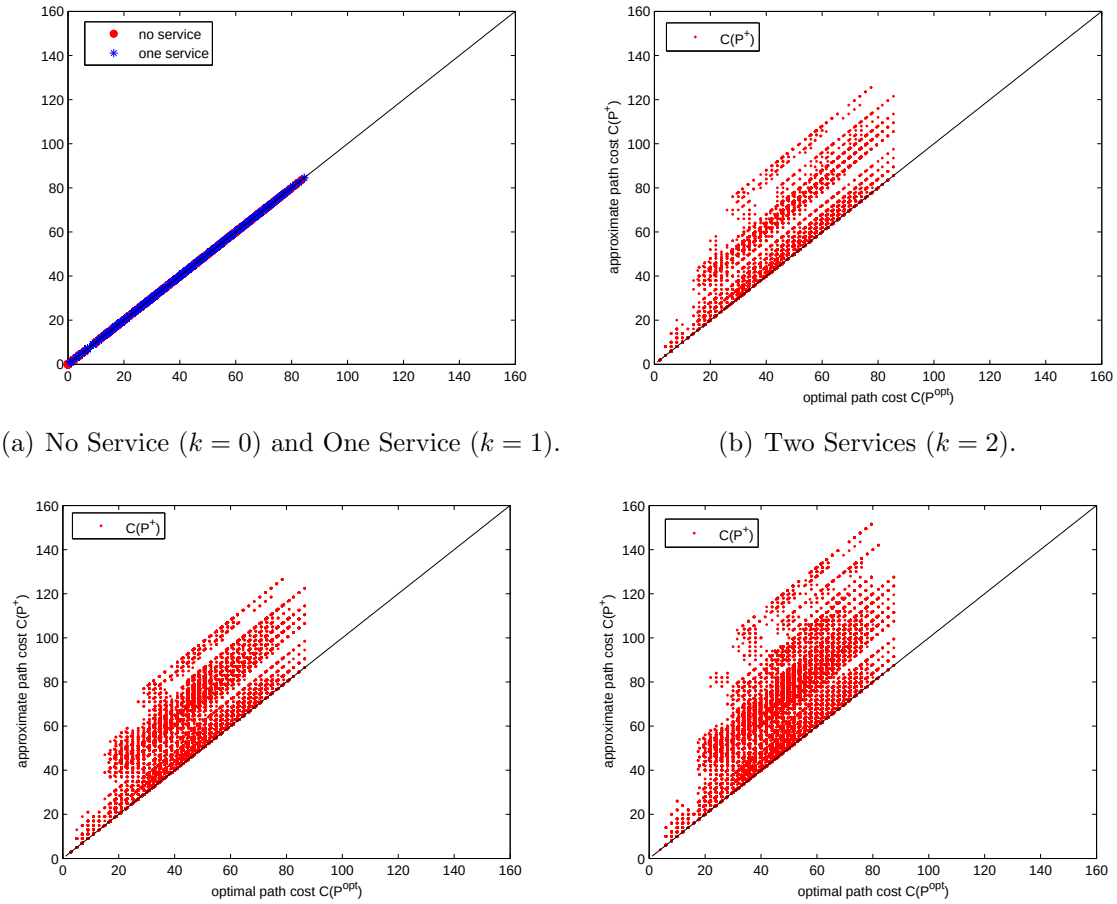
Figure 3.19. Emulab Prototype.

3.7.1 Prototype Implementation

We built a prototype of the service-centric network architecture described in Section 2.2, using Emulab and implemented DSRP. As shown in Figure 3.19, the prototype consists of 60 nodes organized into 12 ASs. Each AS consists of 1 service controller and 4 service nodes. The service nodes are capable of performing a maximum of four different services. We use the approximate DSMR to implement the Inter-AS routing and the centralized layered graph for the Intra-AS routing. The multi-threaded controllers and nodes in the prototype have been coded using C.

3.7.2 DSMR Algorithm

To quantify the quality of approximate DSMR compared to the optimal DSMR or layered graph, we use a simulation setup. We simulate a network topology with 56 nodes, each of which is capable of performing between zero and four different types of services. After the service matrix exchange has stabilized, we evaluate approximate



(a) No Service ($k = 0$) and One Service ($k = 1$).

(b) Two Services ($k = 2$).

(c) Three Services ($k = 3$).

(d) Four Services ($k = 4$).

Figure 3.20. Path Cost (Approximate DSMR vs. Layered Graph (Optimal Solution)).

DSMR by iterating through connection requests from all possible sources to all possible destination using all possible service combinations. The same requests are issued to the layered graph algorithm. This process allows us to obtain the cost of the upper bound, $C(P^+)$ (i.e., the cost of the path that approximate DSMR calculates), and the cost of the optimal path, $C(P^{opt})$ (i.e., the cost of the layered graph result).

3.7.2.1 Correctness of DSMR

Figure 3.20 shows the quality of approximate DSMR compared to optimal path (calculated from layered graph algorithm) for requests with different numbers of ser-

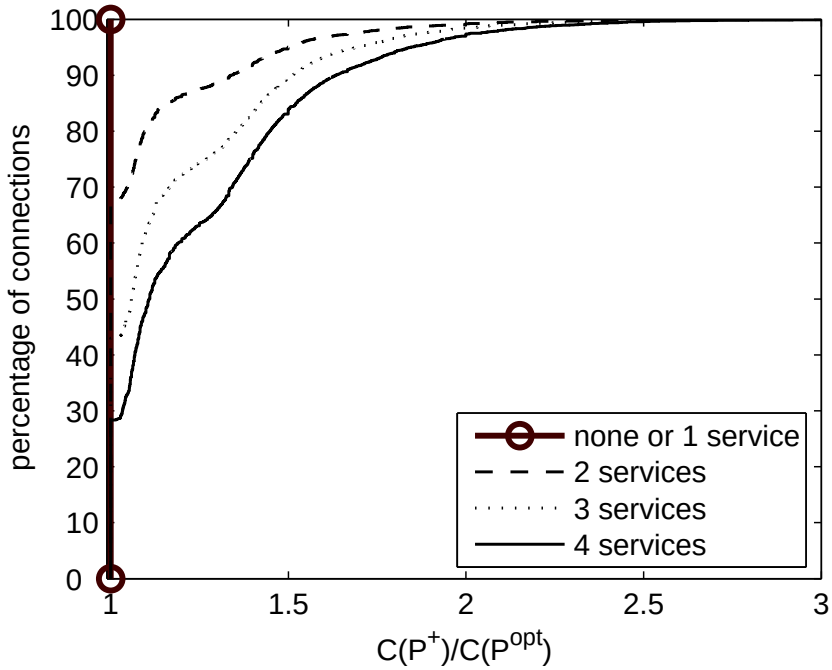


Figure 3.21. CDF of Relative Path Cost (Approximate DSMR vs. Layered Graph (Optimal Solution)).

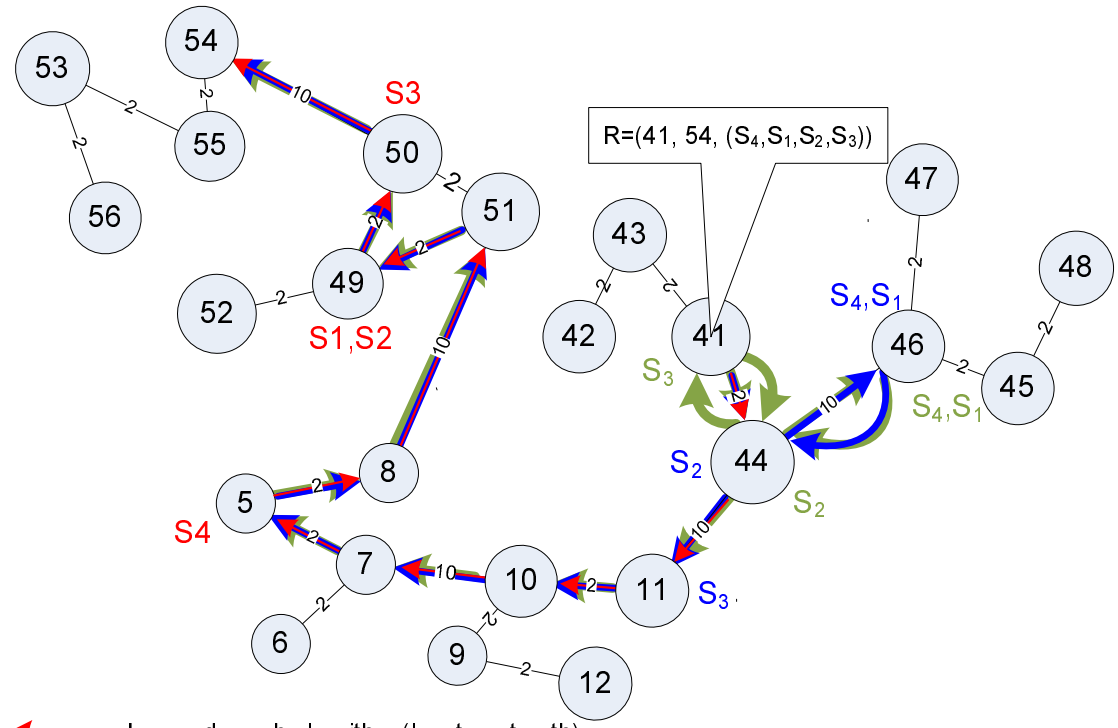
vices. The optimal path cost $C(P^{opt})$ is on the x-axis and the approximate DSMR path cost $C(P^{+})$ is on the y-axis. Each data point represents one connection request. In Figure 3.20(a), we can see that for $k = 0$ and $k = 1$, $C(P^{+}) = C(P^{opt})$ as all points fall on the diagonal. Thus, for zero or one service, approximate DSMR indeed provides the optimal solution as we expect. This results validates that the service matrix exchange (i.e., DSMR) is a correct substitute of the layered graph program.

3.7.2.2 Approximation at Source

Figure 3.20(b), 3.20(c), and 3.20(d) show the quality of approximate DSMR routes for two or more services ($k = 2$,$k = 3$, and $k = 4$) compared to the optimal centralized algorithm. As expected, we observe that $C(P^{+}) \geq C(P^{opt})$. With more services ($k = 4$), the distance from the diagonal increases for some solutions, indicating the approximation decreases in quality. However, numerous solutions remain close to the optimal path.

We can quantify how many requests are served how well by considering the cumulative distribution function (CDF) shown in Figure 3.21. The x-axis shows the relative cost of the approximate DSMR solution comparing to that of the optimal path (i.e., $C(P^+)/C(P^{opt})$). We can see that for zero or one service, approximate DSMR indeed provides the optimal solution since $C(P^+) = C(P^{opt})$ for all such connection requests. This result validates that the service matrix exchange (i.e., DSMR) is a correct substitute of the layered graph algorithm. We can also conclude from the figure that the approximation quality of the approximate DSMR deteriorates with the increase of number of services required in the connection request. However, for two services, around 68% of requests are routed with identical cost as the optimal solution. More than 95% of the requests are less than 50% more costly than the optimal path (i.e., $C(P^+)/C(P^{opt}) \leq 1.5$). For four services, these percentages drop as it becomes more difficult to find the optimal path by approximation. Still, nearly 30% of requests are routed optimally, and 95% are less than double the optimal cost.

It is important to note that from the above results, the majority (more than 95%) of the connection requests have an approximation of less than double of the optimal cost. For the current Internet architecture, an approximation of an optimal path that yields nearly 2× longer paths would be unacceptable, since only shortest path routing (without any services) is currently used and optimality in finding the path is expected. However, when multiple services are required for a connection, longer paths and less tight approximations are acceptable. It may be acceptable to route a connection along a longer path in order to avoid the heavy computational overhead that finding the optimal solution with the layered graph algorithm would entail. We should also note that although the entire network may provide tens of services, in practice, the number of services requested for any connection will only be a few.



Layered graph algorithm (least-cost path)
P^{opt} = $((e_{41,44}, e_{44,11}, e_{11,10}, e_{10,7}, e_{7,5}, e_{5,8}, e_{8,51}, e_{51,49}, e_{49,50}, e_{50,54}),$
$(S_4 \rightarrow 5, S_1 \rightarrow 49, S_2 \rightarrow 49, S_3 \rightarrow 50))$
$C(P^{opt})$ = 56;

Approximate DSMR at node 41
P^{+} = $((e_{41,44}, e_{44,46}, e_{46,44}, e_{44,41}, e_{41,44}, e_{44,11}, e_{11,10}, e_{10,7}, e_{7,5}, e_{5,8}, e_{8,51}, e_{51,49}, e_{49,50}, e_{50,54}),$
$(S_4 \rightarrow 46, S_1 \rightarrow 46, S_2 \rightarrow 44, S_3 \rightarrow 41))$
$C(P^{+})$ = 80

Approximate DSMR at node 44
P^{+} = $((e_{41,44}, e_{44,46}, e_{46,44}, e_{44,11}, e_{11,10}, e_{10,7}, e_{7,5}, e_{5,8}, e_{8,51}, e_{51,49}, e_{49,50}, e_{50,54}),$
$(S_4 \rightarrow 46, S_1 \rightarrow 46, S_2 \rightarrow 44, S_3 \rightarrow 11))$
$C(P^{+})$ = 76

Figure 3.22. Example Path: Layered Graph vs. Approximate DSMR.

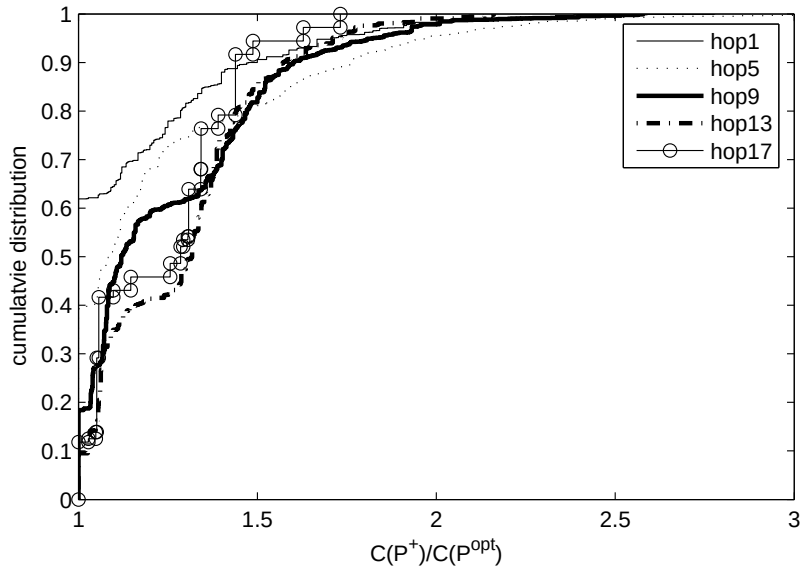


Figure 3.23. CDF of Approximate DSMR Path Cost to Optimal Path Cost for Different Path Lengths and four Services ($k = 4$).

3.7.2.3 Approximation Along Path

The above results consider the quality of the path when comparing the view of the source node with the overall optimum. However, connection requests are passed from node to node. Along this process, the path quality may be improved. This effect is illustrated in Figure 3.22. In the example, computation cost is 1 for all services. In the figure, the optimal path for a connection request is compared to two approximate DSMR solutions. The first DSMR approximation is the path determined by the source (i.e., node 41), which has a cost of $C(P^+) = 80$. When the connection request is passed to the next node (i.e., node 44), the new approximate DSMR computation yields a better path with $C(P^+) = 76$. Thus, the solution provided by approximate DSMR at the source really is a bound and may improve as requests traverse the network.

To further illustrate the improvement of routing as a request gets closer to the destination, Figure 3.23 shows the CDF for four services for different path lengths (counted in hops). We observe that short paths are more frequently close to the

optimal path, whereas longer paths are less likely to be optimal. However, in the vast majority of all cases, the relative cost is less than 2 (i.e., $(C(P^{+})/C(P^{opt}) \leq 2)$.

It is important to note that an approximation of an optimal path that yields nearly $2\times$ longer paths would be unacceptable in the current Internet. Since only shortest path routing (without any services) is currently used, optimality in finding the path is expected. However, when multiple services are required for a connection, longer paths and less tight approximations are acceptable. It may be acceptable to route a connection along a longer path in order to avoid the heavy computational overhead that finding the optimal solution with the layered graph algorithm would entail.

3.7.3 DSRP Protocol

In this section, we evaluate the scalability and the efficiency of DSRP using the results obtained from the Emulab prototype. The results include the evaluation of the service matrix convergence time across the entire network, the connection setup time, the amount of control information required for connection setup, and the amount of state information maintained in service controllers.

3.7.3.1 DSRP Service Matrix Convergence Time

As discussed in 3.4, routing is valid only when the service matrices converge to a consistent state. It should also be noted that the time for convergence is closely related to the size of the network. We hence measure this convergence time as one of the metrics to evaluate the efficiency and scalability of DSRP. For our experiments, we define the convergence time for each service controller to be the time between the first exchange (after initialization) and last update of its service matrix, assuming an unchanged topology.

Table 3.4 shows the maximum, average, and minimum service matrix convergence times among all controllers for networks of different sizes. We use the number of iterations as the unit of the convergence time since the absolute value (e.g., time

Table 3.4. Service Matrix Convergence Time.

Network Size		Convergence Time (iterations)		
topology	diameter(hops)	maximum	average	minimum
3 AS	1	2	1.33	1
6 AS	3	3	2.17	2
9 AS	5	4	2.78	2
12 AS	6	4	3.42	3

in seconds) depends on how often the service matrices are exchanged between the controllers (we set this interval as 20 seconds for our experiment). The size of the network is measured by network diameter, which is defined as the longest distance (in hops) between any two ASs in the network. From the table, we can observe that the time to converge increases as the size of the network increases from 3 AS (diameter of 1) to 12 AS (diameter of 6). Even in the worst case (maximum), the time to converge increases linearly. This indicates that the DSRP is indeed scalable and efficient with regards to service matrix convergence. Moreover, we observe that the service matrix convergence time is not only related to the network size, but also is affected by several other factors such as the topology and the distribution of the services among various service nodes.

3.7.3.2 DSRP Connection Setup

In this section, we evaluate the performance of the routing protocol in terms of the cost to set up a connection. The evaluation metrics are: connection setup time, amount of control information needed, and amount of state information maintained at controllers. We once again setup connections between all possible combinations of source and destination, with all possible permutations of the 4 different services in the prototype.

3.7.3.2.1 Connection Setup Time

First, we evaluate the time it takes to setup the connection for each connection request. This is the time from the moment a node sends out a connection request

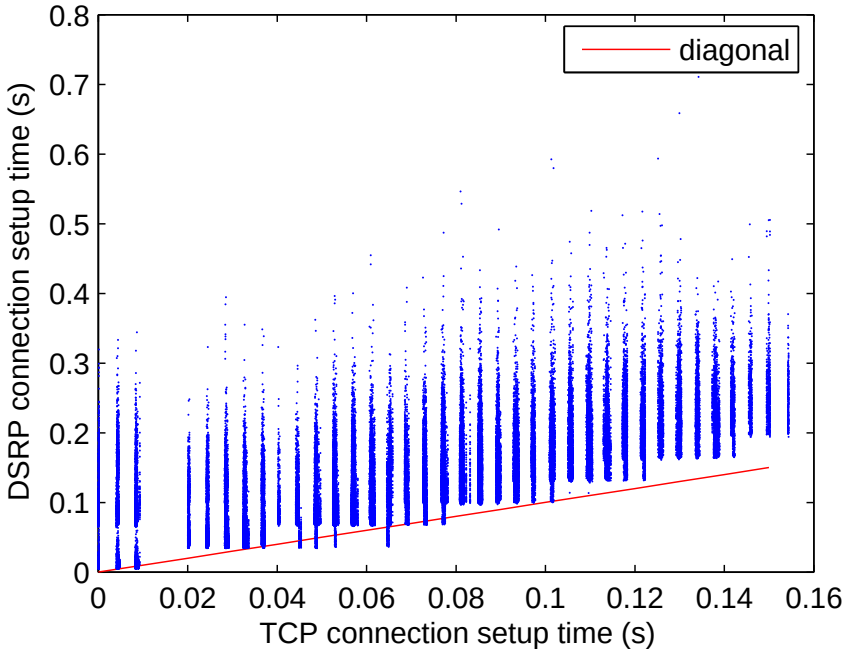


Figure 3.24. Connection Setup Time (DSRP vs. TCP).

to its controller to the moment it receives an acknowledgement for the successful setup of the connection. We use the time to establish a TCP connection between the corresponding source and destination pairs as the basis for our comparison.

In Figure 3.24, y-axis is the connection setup time for DSRP while x-axis is the corresponding TCP connection setup time. From the figure, we can notice that majority of the DSRP connection setup times vary from 1 to several times (within 10) the corresponding TCP connection setup times. To take a closer look, we plot the CDF of this data in Figure 3.25. We observe that all the TCP connections are established within 150 ms and 95% of that of DSRP are established within 250 ms. Figure 3.26 shows the CDF of the relative connection setup time of DSRP with respect to that of TCP for requests with different number of services. From the figure, we can see that the relative time increases only by a small margin with increase in the number of services. Even for 4 services, the setup times for 85% of the requests are less than 5 times that of TCP. The figures indicate that the DSRP is indeed efficient and

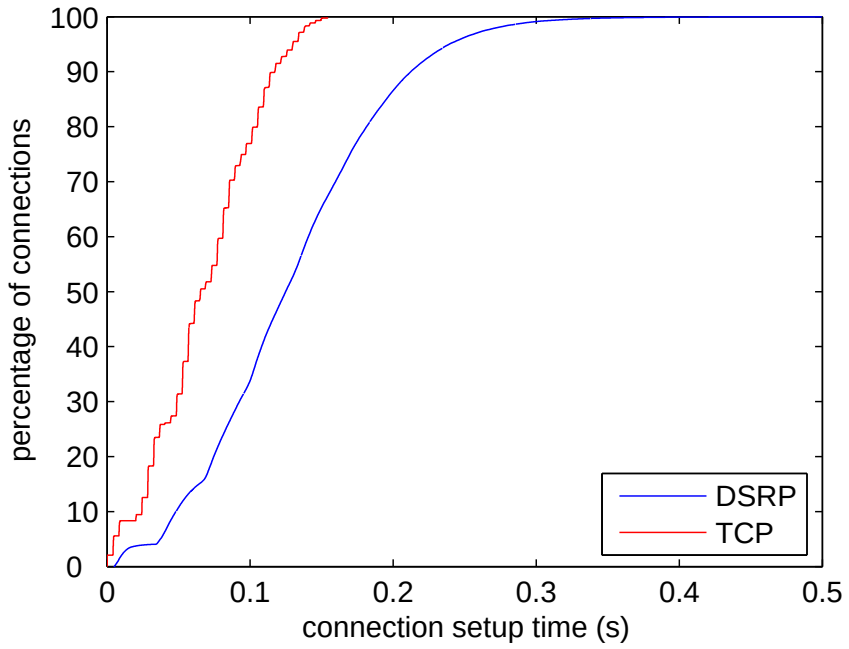


Figure 3.25. CDF of Connection Setup Time (DSRP vs. TCP).

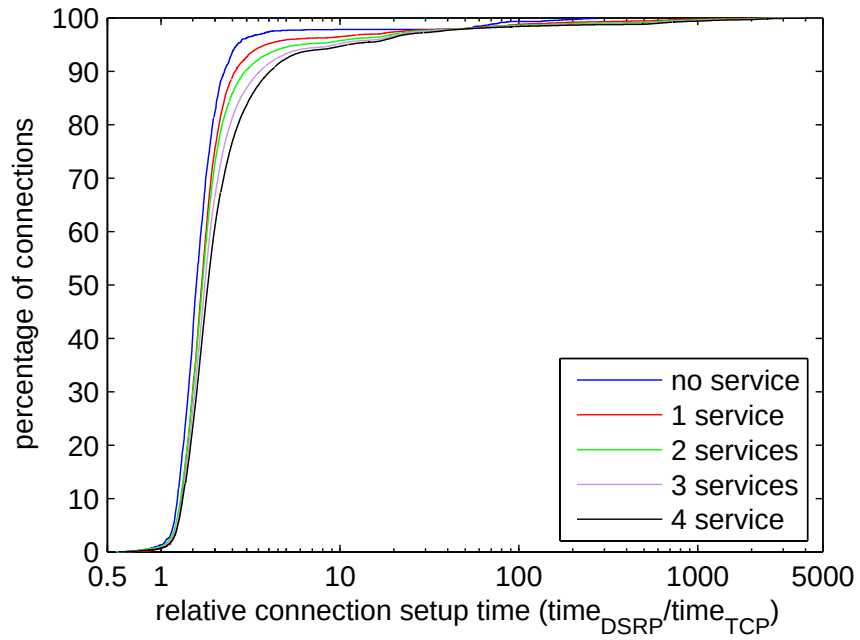


Figure 3.26. CDF of Relative Connection Setup Time.

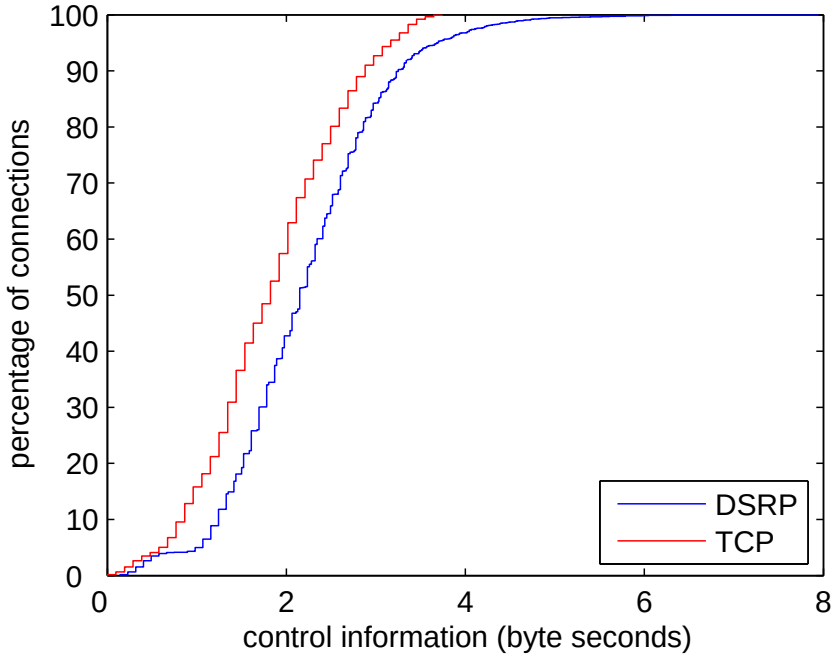


Figure 3.27. CDF of Control Information Size (DSRP vs. TCP).

scalable in terms of connection setup time even though so much more work is being done in the control plane for establishment of a connection with service placement.

3.7.3.2.2 Control Information

Next we evaluate the DSRP based on the amount of control information needed to setup a connection. This includes all the messages exchanged between nodes and controllers during various stages of the connection setup. Again, we use the TCP connection setup (messages exchanged during the three-way hand shake procedure) as the basis for comparison. We define a metric called "byte seconds" (i.e. the size of message multiplied by the time for delivering that message) to evaluate the amount of the control messages. Since this metric considers both the size of the message and the time (and hence distance) for transmitting the message, it gives a better picture of how much resource of the network is utilized by these control messages.

Figure 3.27 shows the CDF of the amount of control information used by the DSRP comparing to that used by the TCP. The CDF of the relative values is shown

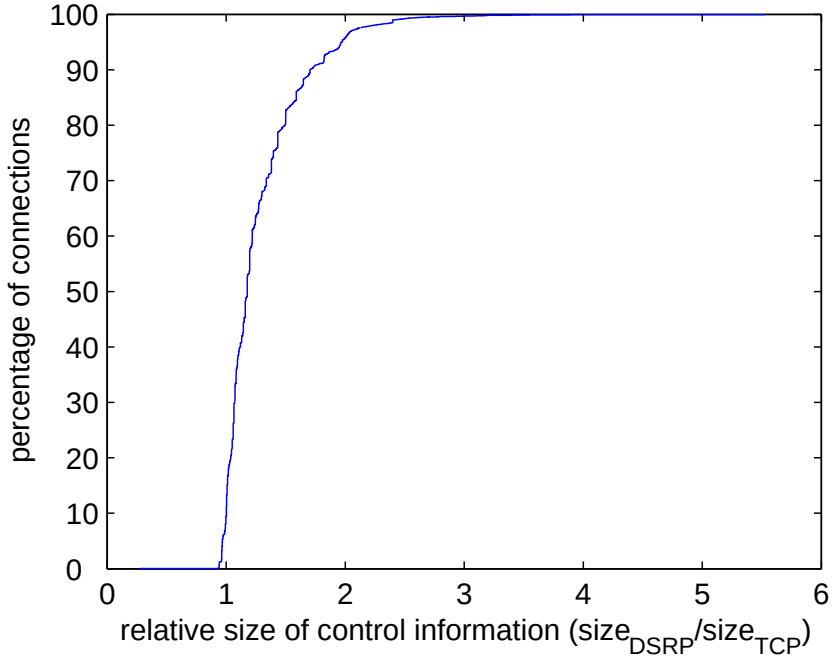


Figure 3.28. CDF of Relative Control Information Size (DSRP vs. TCP).

in figure 3.28. From the figures, as expected we observe that the DSRP in general uses more control messages for path setup than TCP. We can notice that the difference between the two sets of data is quite small. Figure 3.28 shows that for about 85% of the connections, the amount of control information used by the DSRP is less than 1.5 times that of TCP. This proves that the DSRP has low protocol overhead when setting up a connection.

3.7.3.2.3 State Information

Finally, we evaluate the amount of state information that needs to be maintained at the controllers for the duration of active connections. For each active connection, every time the connection goes through an AS there is a corresponding record (we call it "entry") maintained at the controller of the AS. The size of an entry is 24 bytes. The entry stores connection information such as flow_ID, the previous AS_ID, the next hop AS_ID, etc. Depending on the topology, routing, and the mapping of the services, there may be more than one entry at a controller for the same connection.

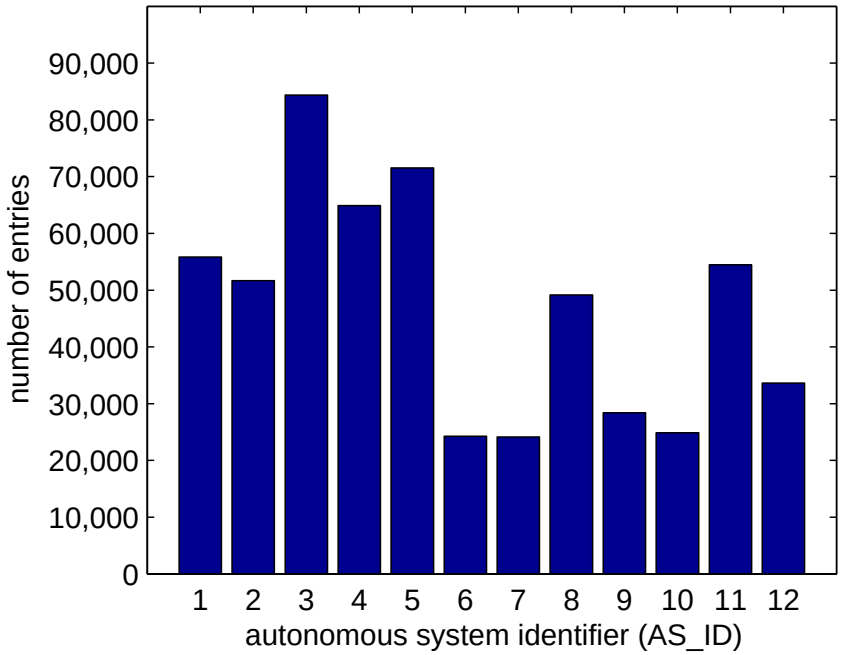


Figure 3.29. Number of Entries Maintained on All 12 Controllers.

In the experiment, we collect the entries from all the service controllers for all the 149,760 connection requests. Since the size of an entry is fixed, we use "number of entries" as the evaluation metric.

Figure 3.29 shows how many entries are maintained on the controllers when all the 149,760 connections are active at the same time. We observe that the number of entries on each controller depends on the position of the controller inside that network. The AS controllers at the edge of the network have lesser entries (controllers of AS-6,7,9,10.,12) while the AS controllers at the heart of the network have more entries. When all the 149,760 connections are active, the maximum number of entries needed at any controller (controller of AS3) is only 84,352, which is 0.5632 times of the number of active connections. This indicates that the DSRP achieves a balanced mapping for routing and service placement.

3.8 Related Work

The term "network service" has been used in a variety of different contexts ranging from application-layer end-system service (e.g., services in a grid computing environment [40] or Cisco Service-Oriented Network Architecture [27]) to packet processing services on routers (e.g., data-path services for the next-generation Internet [112] or programmable networks [88]). While we consider the latter context, our routing algorithm and protocol can be used for any type of service that is performed between communicating end-systems.

As indicated above, our focus in this work is on the control plane. Implementing packet processing services in the data plane has been studied extensively on different router architectures: workstations routers [57], programmable routers [88], virtualized router platforms [6], and Planetlab [12]. While this approach is distantly related to active networks [99], we consider more controllable programmable routers [111], where users can choose from predefined processing features (i.e., services).

To manage network services, an IETF working group has attempted to define Open Pluggable Edge Services (OPES) [9]. In such an architecture, end-systems can specify a set of data flow operations that are implemented on nodes throughout the network. TCP tunnels are used to create a multi-hop end-to-end stream. Also, several protocols for locating services have been proposed (e.g., Service Location Protocol [49]). Our work ties in with these efforts, but focuses on the routing aspect of service-centric architectures.

The service routing problem that we defined in Section 3.2 has received some attention. Choi et al. have developed layered graph algorithms [26]. This algorithm was later expanded to consider capacity constraints [25]. Ruf et al. consider service placement across network nodes and with multiprocessor packet processing systems [88]. They suggested to use randomized algorithm to solve the service mapping problem. Our service step search algorithm is a novel contribution for solving the service routing

problem with resource constrains, which is proven to be NP-complete [25]. Comparing to the earlier global algorithms (i.e., layered graph algorithm and randomized algorithm), it performs the best in most of our experiment scenarios. However, all of these global routing algorithms requires a complete view about the whole network and thus exhibit limited scalability. Our decentralized algorithms (i.e., DSMR and approximate DSMR) can scale to larger networks.

3.9 Summary

In this chapter, we presented a discussion of the service routing problem that occurs in networks where protocols and networking features (i.e., network services) are composed dynamically and the basic composition blocks are distributed across the network.

A global algorithm, service step search, is proposed to solve the service routing problem with resource constrain considerations. The performance of service step search algorithm are investigated and compared to that of another two algorithms, layered graph and randomized. The simulation results show that service step search performs best in terms of finding low delay path and minimizing connection drop rate. Another important conclusion from the results are the tradeoff between routing quality (end-to-end delay) and computation time. The randomized algorithm usually can figure out a valid routing very fast, while the routing quality is not guaranteed. The layered graph and service step search can get much better routing at the cost of computation time.

However, the global algorithms need to have a complete view about the whole network, including the nodes, the conductivities, the link costs, and the services. Thus, they are scalable enough to be used in practice. A decentralized algorithm, DSMR, is proposed to solve this scalability issue, followed by the description about an

approximation version to save the amount of routing information exchanged between neighboring nodes.

Based on above work, a matching protocol, DSRP, is designed to solve the service routing problem systematically in large-scale networks with data-path services. The DSRP uses decentralized algorithm (DSMR or approximate DSMR) for Inter-AS routing decision to achieve scalability. It uses global algorithm (layered graph or service step search) for Intra-AS routing to increasing routing quality (i.e., optimal or near-optimal routing in terms of end-to-end delay).

We have implemented a prototype on Emulab to compare the performance of different algorithms and to evaluate the efficiency of the proposed routing protocol. The results show that the approximate DSMR achieves a good balance between routing quality (i.e., end-to-end delay) and computation cost (i.e., calculation time and the amount of information exchange). Our prototype implementation also illustrates the capabilities of the routing protocol, DSRP, in terms of fast convergence, efficient connection setup, and low protocol overhead.

We believe our work on service routing forms a solid step towards efficiently managing data-path services in future networks.

CHAPTER 4

SERVICE COMPOSITION

4.1 Introduction

In our service-centric network architecture, communications are broken down into basic functional blocks that can be composed on demand. These basic blocks are network services. Dynamic composition of network services allows networks to establish connections with customized communication characteristics to satisfy various application requirements. The application here refers to all the network program entities (e.g., virtual networks, services, and applications) developed on the ITDS layer of the service-centric network architecture.

In service-centric networks, the combination of network services determines the functionality, quality, and performance of a communication. Thus, it is important to consider what choices (i.e., network services) are available within a network and how to pick a suitable service combination to satisfy the communication requirements of a given request. This is called *service composition problem* in our work.

The service composition problem is trivial in the scenario of the current Internet, since the choices are very limited. In this case, it typically comes down to a decision between TCP, which provides reliability at the cost of end-to-end delay, and UDP, which provides low latency with potential packet loss. However, with more and more advanced packet processing functions are proposed in the next-generation Internet, we assume that there will be a numerous number of network services available in the network and a huge difference between communication requirements from different

applications. When more services are available, the service composition problem becomes more complicated.

Furthermore, in the context of service-centric network, the service composition problem is closely related to routing decisions. Since the criteria for the a "good" combination should not only be the one which use the least number of services (steps) to satisfy the communication requirements, but the one which can achieve the least end-to-end routing delay (i.e., considering both communication delays and the processing delays). This problem is called as "service composition and routing problem" in our work. When the network becomes huge and more services are involved, it becomes extremely difficult to find the optimal solution.

In this chapter, we address the service composition problem faced in the service-centric networks. We answer one of the most fundamental questions in the context of service-centric networks:

> *How can we achieve optimal or near-optimal service composition for a connection requests according to its communication requirement?*

The specific contributions in this chapter are:

1. A formal definition of the service composition problem.
2. A general framework for solving the service composition problem in service-centric networks.
3. A formal definition of the service composition and routing problem.
4. A decision making system that can find optimal or near optimal solution for the service composition and routing problem.
5. A synthetic benchmark that allows a comprehensive evaluation of the performance of different solutions for the service composition and routing problem.

6. An extensive evaluation of our solution under several synthetic and practical network scenarios.

In this chapter, Section 4.2 first formalizes the service composition problem faced in service-centric networks and then introduces a solution for achieving the automated service composition. Section 4.3 extends the service composition problem to consider routing factors, and defines the service composition and routing problem in the context of service-centric networks. The design of a general framework and a novel decision making system are then discussed as a response to the problem. A synthetic benchmark, which provides a standard testing environment for evaluating the service composition and routing solutions, is presented in Section 4.4.1. In this section, simulation results are also analyzed and compared to evaluate the performance of proposed solutions. Finally, related work is discussed in Section 4.5. Section 4.6 summarizes this chapter. Parts of this chapter are published in [90] and [116].

4.2 Automated Service Composition

As described in the overall design of the service-centric network architecture [44, 112], protocol features and communication requirements are decomposed into basic network services that operate in the data path of the network. By providing different combinations of the network services along the data-path, the network can satisfy different communication requirements from the end-system applications. However, one of the major challenges of implementing such a network is how to decide the appropriate service composition for a particular communication requirement, which is called as service composition problem in our work.

In this section, we first define the service composition problem and then proceed to present a common framework that can be used to represent communication needs, services features, and system characteristics. This framework is later used to automate the composition process.

4.2.1 Service Composition Problem

The service composition problem can be stated as follows: Given a set of network services and a description of some particular communication requirement, check the validity of a given composition or even compute the correct composition to achieves the communication requirement.

Formally, assume $S = \{S_1, S_2, ..S_n\}$ is the set of services available inside the network.

1. Validity Check Problem: Given a communication request $R = (s, t, \{S_1, S_2, ..., S_k\})$. The request is valid if the sequence is valid. That is, $S_i \in S$ for $1 \leq i \leq k$ and $S_1 \gg S_2 \gg ... \gg S_k$.

2. Automated Composition Problem: Given a communication request R, find an valid sequence of services $(S_1 \gg S_2 \gg ... \gg S_k)$ such that $R \Leftarrow (S_1 \gg S_2 \gg ... \gg S_k)$.

4.2.2 Service Composition Framework

As described in [106], a intuitive solution for the service composition problem would be to first build a comprehensive dependency graph based on a set of rules and precedence constraints between the services. And then use this dependency graph to validate and correct the service composition sequence. However, this solution is not scalable or efficient because it requires a centralized view of all the available network services and an enumeration of all possible mutual constraints between any pair of network services.

We introduce a novel approach to solve the service composition problem in a more robust and distributed manner. Instead of focusing on the dependency relationship between services, we believe a description about the services is more essential. From a systematic description of the input and output characteristics of individual services,

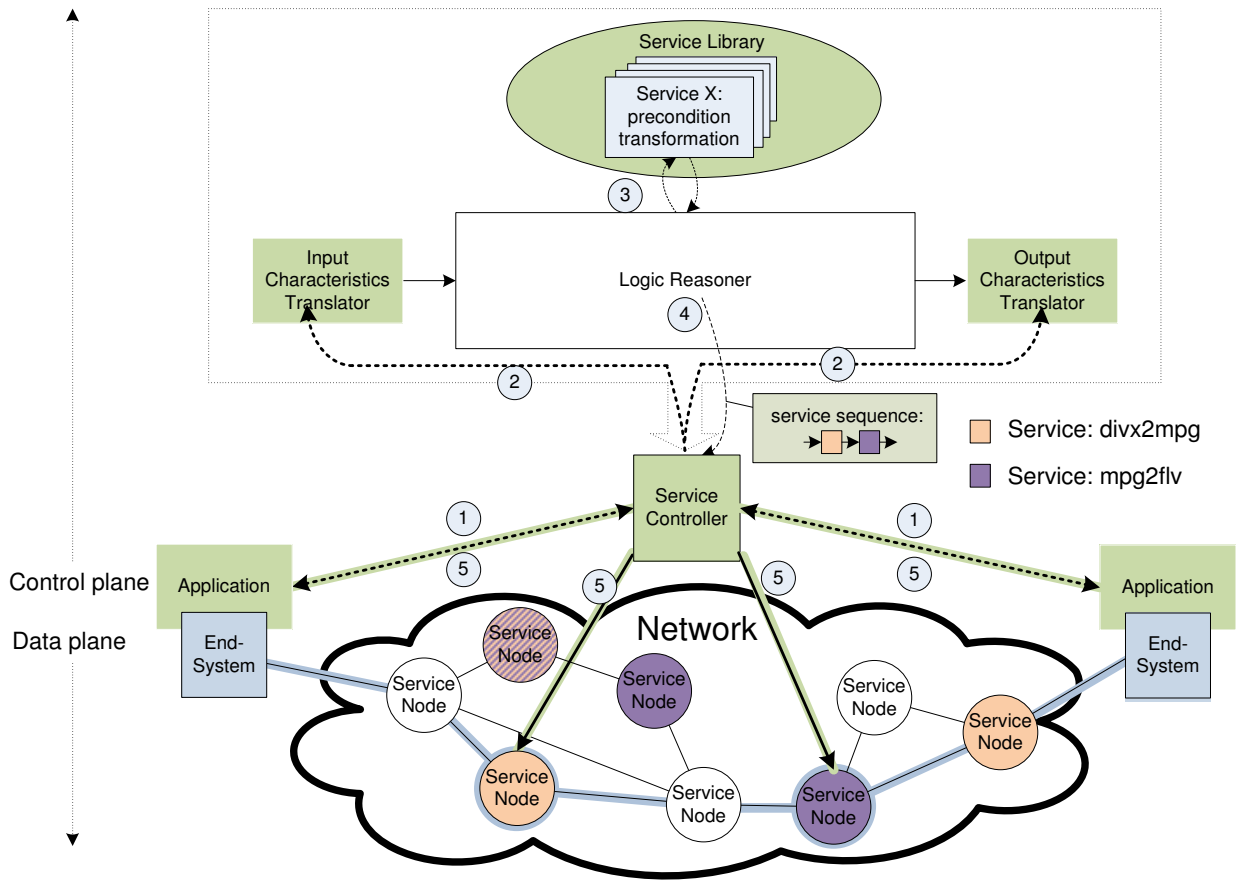


Figure 4.1. Service Composition Framework.

the composition dependencies between the services can be fully deduced. Followed by this logic, a general framework is designed to solve the service composition problem.

As shown in Figure 4.1, the framework is composed of three major parts: the input/output characteristics translator, the pre-defined service library, and the logic reasoner. The input and output characteristics translators are responsible for translating the communication requirements into the suitable input and output description that can be understood by the logic reasoner. The service library is a collection of descriptions (i.e., the input and output characteristics of the services) about the available services. The logic reasoner is a piece of software that can help to validate a sequence of service composition or to generate a valid sequence of service composition

according to the input and output requirements and the service descriptions. The design and implementation details are presented in Section 4.2.3 and Section 4.2.4.

The whole framework can be illustrated by the following scenario: Assume that the server of an online IPTV service is transferring a movie to an end-system. The source file format (i.e., the video stored on the server) is in *.divx* format. The end-system can only support and display videos in *.flv* format. Assume two related video transcoding services are provided in the network: divx2mpg and mpg2flv. First, the communication requests are sent to the service controller from the applications running on both the server and the end-system (Step 1). In step 2, the requests are sent to the input/output characteristics translators and translated to the input/output descriptions of the logic reasoner. In our scenario, the input characteristics should be: .divx video, and the output characteristics should be: .flv video. After getting the input and output, the logic reasoner consults the service library (step 3) and computes a valid service composition sequence as the results (step 4): divx2mpg (marked in yellow) $\gg$ mpg2flv (marked in purple). At last (step 5), the service controller will decide where to put the services and setup the whole path (marked in blue) for the connection. The routing protocol and algorithm are described in [53]).

4.2.3 Data and Communication Characteristics

One issue that occurs in the design of the framework is: what characteristics are important enough to be examined when describing the input and output characteristics of either the network or any individual service? After an extensive study of the services, applications, and communication paradigms in the current Internet, we designed a tree structure (Figure 4.2) to represent the data and communication characteristics. The tree representation includes the class hierarchies inherent in data and communication characteristics. For example, the different types of payloads (text, video, audio, images) can be classified as "type" under data characteristics.

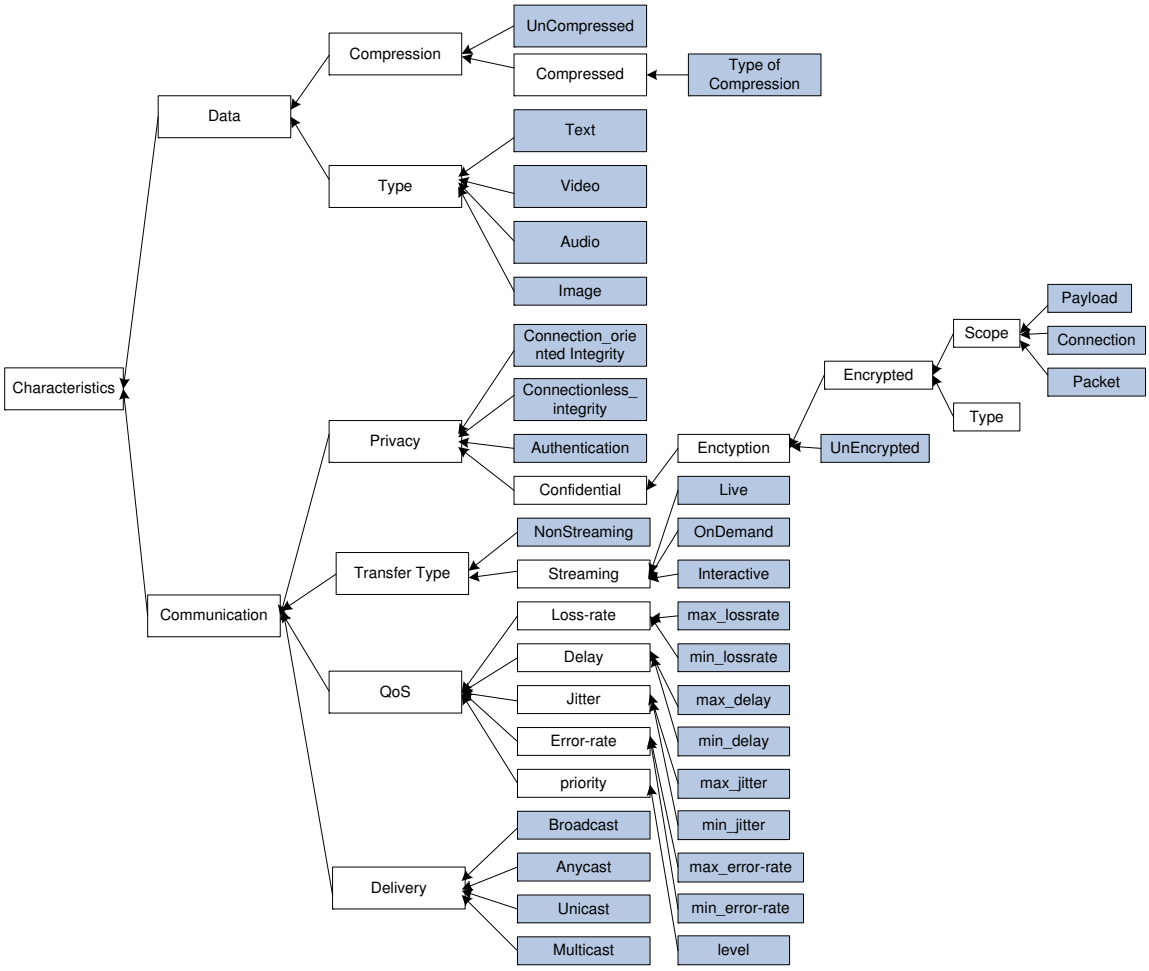


Figure 4.2. Semantics for Representing Characteristics.

The leaf-nodes in the tree structure represent parameters that can take on different values depending on the service the structure represents. For example, the Type of Compression node indicates the compression algorithm used, whereas max_delay and min_delay nodes under class Delay indicate the maximum and minimum acceptable values of delay. The data and communication characteristics tree is a general representation that can be extended to include new characteristics and can be described by semantic markup languages.

4.2.4 Automation of Service Composition

In this subsection, we describe how we make use of our service composition framework to automate the process of service sequence composition and sequence validity check.

4.2.4.1 Validity Check of Composition

Let $S = \{S_1, S_2, ..S_n\}$ be the set of services available inside the network. Let $p_1, p_2, ...p_n$ be the preconditions(input semantics) for services and $o_1, o_2, ...o_n$ be the output semantics produced by execution of the services $S_1, S_2, ...S_n$, respectively. Given an ordered sequence of services $(S_{x_1} \gg S_{x_2} \gg, ... \gg S_{x_k}), 1 \leq k \leq n$, with I being the semantics of data input into S_{x_1} and T being the semantics of data produced by the sequence (i.e. the target or goal to be reached), the sequence can be checked according to our framework to be valid if the following two rules hold:

1. The preconditions of each service in the ordered sequence are satisfied. That is, the output produced by a service should satisfy the preconditions of the service that follows in the sequence.

2. The precondition of the first service in sequence, S_{x_1} should be satisfied by input semantics I.

3. The output semantics of the last service in sequence S_{x_k} is the desired sequence output semantic T.

Each of these rules can be easily checked using our framework. The problem of validity check thus reduces to matching the preconditions of each service with the semantics of the data input to the service.

4.2.4.2 Automated Composition of Services

A straightforward solution to automating service composition problem would be the selection of the set of services dynamically out of the available services and order them in the sequence based on a set of rules and precedence constraints between the services so as to achieve the task. However, this solution involves the enumeration of all possible mutual constraints between pairs of network services.

We take a different approach to the problem of automated service composition, relying more on the connection and data semantics rather than on the precedence constraints between every possible service pair. The Data and service semantics (described in Section 4.2.3) enable us to compose complex services to achieve the desired goal automatically.

In our design, a service can be fully described by two characteristics:

- Preconditions - Preconditions are logical conditions that need to be satisfied in order for the service to be executed. The preconditions include both the protocol and data semantic requirements. For example, an input video stream to a video transcoding service that transcodes from MPEG to H.264 formats has to be MPEG, whereas the streaming protocol used can be either RTP, UDP or RTSP.

- Transformation - The function it performs and the transformations it causes in the input data semantics to produce an output with possibly different semantics. The service may or may not change the data stream. For example, a bandwidth

monitoring service will not alter the packets in anyway, whereas an encryption service will.

Our composition framework allows the automation of service composition as follows: Given a set of network services, the description of their preconditions (i.e., logical conditions that need to be satisfied in order for the service to be executed) and transformations (i.e., the effect of the service execution on the data semantics), initial and goal states of the data semantics, find a sequential composition of services that achieves the goal while maintaining the precondition requirements of each component service used in the sequence.

The above automated service composition problem can be reduced to the a planning problem wherein the services are regarded as **actions**, each with its own precondition and effect on global state: Given a set of actions that a planner can perform, A, a set of all possible data semantic states, Ψ, an initial state, $I \subset \Psi$, and a goal state, $T \subset \Psi$, that the planner attempts to achieve, the task is to attempt to achieve the goal state T from the initial state I by performing a series of actions. In the context of network service composition, Ψ is equivalent to the set of all possible semantic states of the data in the network. I is equivalent to the initial semantics of the data. T is the targeted semantics. The output of planning is a sequence of actions (services that should be executed in order) that form an optimal composition (in terms of total number of services in the output sequence) for the problem. Thus, the automation of service composition can be implemented by first formulating the problem into the corresponding planning problem and then solving it using an existing planner.

4.2.5 Examples

To validate our automated service composition framework design, we describe three test scenarios, along with the service requirements and results. In all cases, we have used LPG (Local search for Planning Graphs) [46] for planning and PDDL

(Planning Domain Definition Language) for describing services as actions. A more detailed description on LPG and PDDL is presented in Section 4.3.4.

4.2.5.1 Scenarios

The three scenarios used in this section are:

- Example scenario 1 – video transcoding service: Consider an IPTV distribution service that broadcasts HDTV-1080p videos. However, not all subscribers (e.g., PDAs and cellphones) to this service can receive and display high-quality HDTV video. In this case, a transcoding service needs to be executed in the data-path to transcode the video from HDTV-1080p to the format and resolution supported by the subscriber (say H.264 (176X208)).

- Example scenario 2 – encryption service: An end-system application requests an encrypted communication to upload some documents to a server. This scenario needs an encryption service to be executed in the data-path of the connection to satisfy the security requirement.

- Example scenario 3 – transcoding + encryption: Consider an end-system (PDA that supports only H.264 (176X208)) is requesting a secured Video-On-Demand from a server, which can only provide HDTV-1080p sources. In this case, two services, transcoding and encryption, need to be executed in the data-path and the transcoding service need to be done before the encryption service.

In all cases, the end-systems just need to specify the data and communication characteristics they intend to send and/or receive through a connection. The services are automatically selected based on this specification and a valid composition is constructed. In the next section, we present the results of the three scenarios discussed.

Table 4.1. Preconditions and Actions for the Example Services.

Service	Precondition	Action
Transcoder	Type-Video (HDTV-1080p)	Action (Converts to H.264.)
Encryption	Type-Text	Action (Encrypts payload.)

Table 4.2. Input and Output Semantics for Example Scenario 1.

Semantics	Input	Output
Type	Video-(HDTV-1080p)	Video-(H.264 (176X208))
Delivery	Broadcast	Broadcast
State	Stateless	Stateless
QoS	max_delay, min_delay	max_delay, min_delay
Unit	packets	packets

4.2.5.2 Results

Table 4.1 describes the services implemented along with their preconditions and actions. Tables 4.2, 4.3, 4.4 show the input and output semantics of the three example scenarios discussed in the previous subsection.

Tables 4.5 shows the results of the automated composition implementation of example scenarios discussed in the previous subsection. From the table, we observe that the requested service is selected by the sequential calculus based reasoning engine. The decision is purely based on the semantics of the data and communication and the service action. Observe from the last entry of Table 4.5, that the automatically composed sequence of services achieves the desired output - encryption of a H.264 video file - even though only the input and output data semantics are provided by the end-system. Moreover, the correct ordering of services is maintained (since an

Table 4.3. Input and Output Semantics for Example Scenario 2.

Semantics	Input	Output
Type	Text	Encrypted
Scope of Encryption	Payload	Payload
Delivery	Unicast	Unicast
State	Stateless	Stateless
Unit	packets	packets

Table 4.4. Input and Output Semantics for Example Scenario 3.

Semantics	Input	Output
Type	Video	Encrypted
Type	Video-(HDTV-1080p)	Video-(H.264 (176X208)
Scope of Encryption	Payload	Payload
Delivery	Unicast	Unicast
State	Stateless	Stateless
Unit	packets	packets

Table 4.5. Results for the 3 Example Scenarios.

Scenario	Result
Example Scenario 1	Transcoder
Example Scenario 2	Encryption
Example Scenario 3	Transcoder $\gg$ Encryption

encrypted video stream cannot be transcoded, transcoding should be done before encryption).

4.3 Automated Service Composition and Routing

4.3.1 Motivation

Now, we have known that there are two major technical challenges emerging in the context of service-centric networks: service composition and service routing. Service composition refers to the problem of finding the (preferably optimal) composition of services to satisfy the communication requirements of a given request. Service routing refers to the problem of finding the (preferably optimal) path for a given composition of services such that the required services will be executed in order along the path.

So far, these two topics have been studied separately. However, in a network with data path services, these problems cannot be considered independently since separate optimization may not lead to a global optimum. Instead, the overall problem is to find the optimal path for a given request, which combines both service composition and service routing. Thus, we need to consider a combined "service composition and routing" problem.

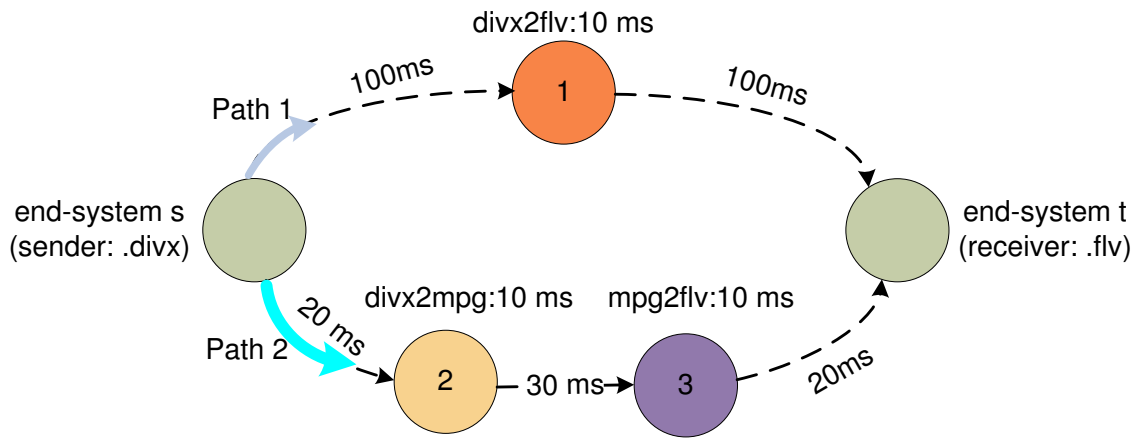


Figure 4.3. An Example for the Service Composition and Routing.

Here we present a brief example that illustrates why a combination of composition and routing can yield better results than considering these problems separately. The problem scenario is shown in Figure 4.3. The network services provided in this network are video transcoding functions. Here we use different colors to represent the services that are provided by nodes. Node names, link delays, services provided on the nodes, and the service processing delay are all annotated on the figure. For example, node1 provides video transcoding from *.divx* format to *.flv* format, with the processing cost of 10 ms.

Now node s wants to send a video stream in *.divx* format to node t. However, node t is only capable of playing video stream in *.flv* format. There are two different paths that can accommodate the service request:

Path1: sender s → node 1 → receiver t, with transcoding service divx2flv executed on node1. This solution uses only one service step along the path. The total cost for it is 210 ms.

Path2: sender s → node 2 → node 3 *to* receiver t, with service divx2mpg executed on node 2 and service mpg2flv executed on node 3. This solution uses two services steps for the video transcoding but with a total cost of 90ms.

Two approaches can be used to solve the problem. One is to consider the service composition and service routing problems separately. Another is to consider them together. When using different approaches, we obtain these differing solutions:

1. **Separate Service Composition and Routing (SSCR)**: This approach divide the problem into two sub-problems and solves them separately using the following two steps: 1) Solve the service composition problem, that is, to convert the video stream of divx into flv format with the minimized cost (service steps). The optimal answer to this problem has one service step: divx2flv. 2) Use the output of service composition as the input of service routing, and find the shortest path (least-cost in delay) for this service composition. The output for this step is path 1.

2. **Combined Service Composition and Routing (CSCR)**: This approach solves the service composition and routing problem as one problem, that is, to find the shortest path (least-cost in delay) from node s to node t, while at the same time can convert video format from divx into flv along the path. By considering both path 1 and path 2 and compare their cost, the output for this solution should be path 2.

In this simple example, the combined solution finds the optimal solution of the service composition and routing problem while the other approach does not. This indicates that it is important to solve this problem as a combined problem since the optimal solutions for sub-problems may not lead to the overall optimal solution.

4.3.2 Service Composition and Routing Problem

4.3.2.1 General Description and Assumptions

As illustrated in Figure 4.3, an end-system s (sender) of the video stream, only supporting divx format) wants to communicate with another end-system t (receiver

of the stream, only supporting flv format). In service-centric networks, the end-system s first sends out a connection request to the network, which specifies all its communication requirements: *s:video:divx→t:video:flv*. Then, the network determines a service composition that can satisfy the communication requirements, selects the least-cost route for the composition, and sets up the route for the connection. After that, data will be transferred along the connection until the connection is torn down.

Multiple compositions of services and paths may satisfies the same communication requirements. The goal of our work is to find the optimal service composition, which leads to the minimum routing cost (i.e., lowest end-to-end delay, which includes both link delay and service processing delay). Thus, the control plane of a service-centric network needs to support the setup of custom service requests. This setup involves several decisions:

- Choice of services. The service request may not fully specify all necessary services. Instead, it may only list the characteristics of source traffic and the characteristics of the traffic when it reaches the destination. Thus, the network needs to decide which services to instantiate along the way to achieve a translation of characteristics.

- Choice of service nodes. Since there are typically multiple nodes that can perform service processing, the network needs to decide which node is used to perform service processing for a particular connection.

- Choice of network links. To connection the end-nodes and the nodes that perform service processing, the network needs to set up routes. This is typically solved by simply using shortest path routing between the nodes.

This set of decisions leads to the overall problem that we address in this paper:

How can we achieve optimal or near-optimal setup of connections that involve data-path services?

Achieving optimality requires consideration of all aspects of the problem listed above.

We make the following assumption in our work:

- We assume that the network supports the establishment of connections with fixed paths. This is required since traffic needs to traverse a particular set of service nodes as determined by our solution.
- We assume that cost for communication and cost for service processing can be represented using a single metric (e.g., delay).

Despite these necessary assumptions, the problem remains general enough to be representative of scenarios that occur both in the current Internet as well in potential future networks.

4.3.2.2 Formal Problem Definition

The service composition and routing problem can be stated as: Given a network, a set of network services, their availability and distribution in the network and a description of communication requirements, find a service composition and the corresponding path that achieves the communication requirements with the minimum cost. In this paper, cost is measured in terms of link delay and service processing delay. The formally definition for the service composition and routing problem is as follows:

The network is represented by a weighted graph, $G = (V, E)$, where nodes V correspond to routers and end systems and edges E correspond to links. Each edge $e_{i,j}$ that connects nodes v_i and v_j is labeled with a weight $w_{i,j}$ that represents the communication cost (e.g., delay). Assume $S = \{S_1, S_2, ..S_n\}$ is the set of services available in the network, G. Each node v_m is labeled with the set of services that it can perform $u_m = \{S_k|\text{service } S_k \text{ is available on } v_m\}$ and the service processing cost $c_{m,k}$ (e.g., processing delay) of each service. A *connection request* is represented as $R = (s, t, I, T)$, where s is the source node, t is the destination node, and I,T is the

specification of communication requirements, which is represented as a pair of data formats at the source and the destination.

Given a network G, the service set S, and a request R, we need to find a path for the request such that the source and destination nodes are connected and the communication requirements are satisfied. The path is defined as $P = (E^P, M^P)$ with a sequence of edges, E^P, and services mapped to processing nodes, M^P: $P = ((e_{s,v_{i_1}}, \ldots, e_{v_{i_h},t}), (S_{k_1} \rightarrow v_{j_1}, \ldots, S_{k_l} \rightarrow v_{j_l}))$, such that $S_{k_x} \in S$ for $1 \leq x \leq l$ and $R \Leftarrow (S_{k_1} \gg S_{k_2} \gg ... \gg S_{k_l})$. To determine the quality of a path, we define the total cost $C(P)$ of accommodating the connection request, R as the sum of link cost and service processing cost: $C(P) = \left(\sum_{\{(i,j)|e_{i,j} \in E^P\}} w_{i,j}\right) + \left(\sum_{\{(k_x,m_x)|S_{k_x} \rightarrow v_{m_x} \in M^P\}} c_{m_x,k_x}\right)$.

In many cases, it is desirable to find the *optimal* connection setup. We view this optimality in terms of the least-cost path of a single connection request.

In this subsection, we introduce a novel decision making system to solve the service composition and routing problem. First, we will give an overview of the framework design in Section 4.3.3. Then, we will elaborate the design and implementation details in Section 4.3.4

4.3.3 Decision Making Framework

Our intention is to solve the service composition and routing problem in a robust and distributed manner. Instead of focusing on the dependency relationship between services, we believe a description about the services is more essential. From a systematic description of the input and output characteristics of individual services, the composition dependencies between the services can be fully deduced. Followed by this logic, a general framework is designed to solve the service composition and routing problem.

As shown in Figure 4.4, the whole system is composed of two major parts: the knowledge base and the reasoner. The knowledge base is a collection of descriptions

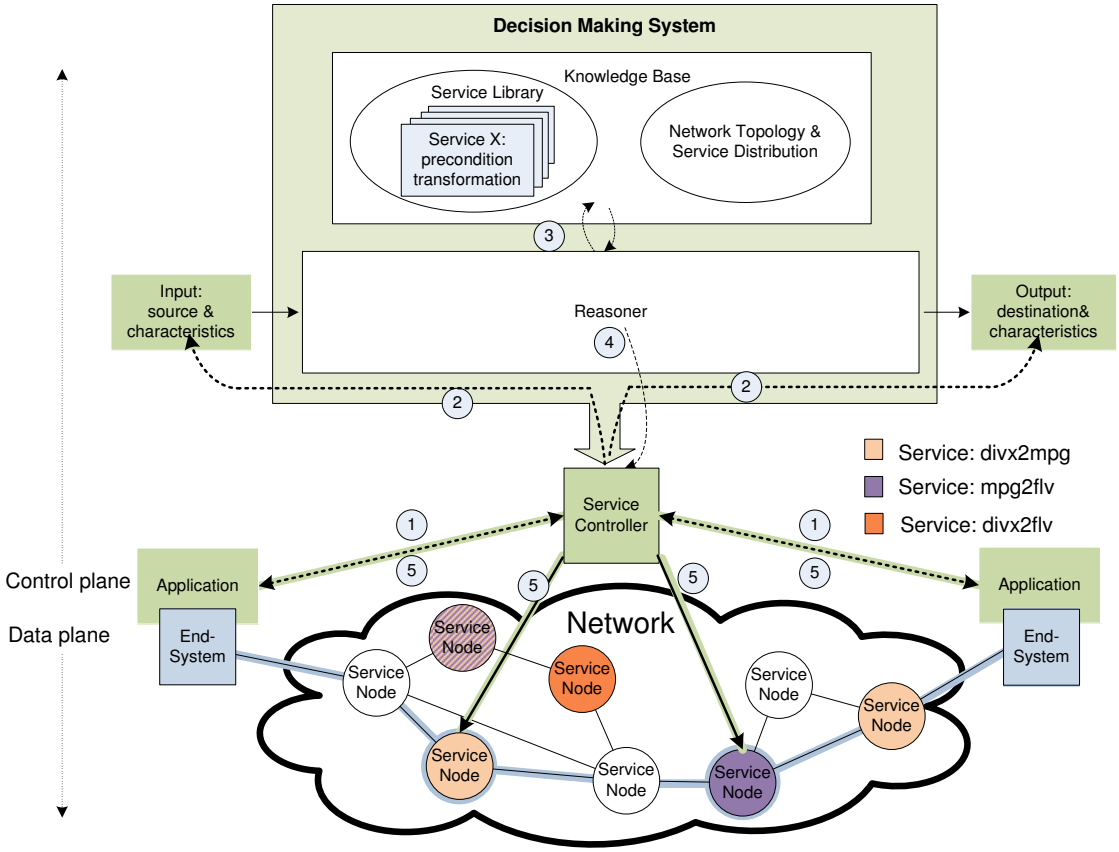


Figure 4.4. Decision Making System.

about the network topology, service availability and distribution, and the services (i.e., the input and output characteristics of the services). The reasoner is the core of the decision making system. It is a piece of software that can help figure out the optimal service composition and routing according to the input and output requirements and the information in the knowledge base. The design and implementation detail of the framework is presented in the following subsections.

The whole framework can be illustrated by the following scenario: Assume that the server of an online IPTV service is transferring a movie to an end-system. The source file format (i.e., the video stored on the server) is in *.divx* format. The end-system can only support and display videos in *.flv* format. Assume three related video transcoding services are provided in the network: divx2mpg, mpg2flv, and divx2mpg. First, the communication requests are sent to the service controller from the applications running on both the server and the end-system (Step 1). In step 2, the requests are sent to the input/output characteristics translators and translated to the input/output descriptions of the logic reasoner. In our scenario, the input characteristics should be: .divx video, and the output characteristics should be: .flv video. After getting the input and output, the logic reasoner consults the knowledge base (step 3) and computes the best path in terms of least-cost (marked in blue) as the result (step 4). Note that in order to calculate the best path for the input/out requirements, only the service library is not enough. The knowledge base also needs to include network topology and service distribution information. At last (step 5), the service controller will communicate with the nodes and setup the whole path for the connection.

The data and communication characteristics follow the same design as the automated service composition (Section 4.2.3).

4.3.4 Combined Service Composition and Routing

Our previous work on automated service composition (Section 4.2.4) leads to an intuitive solution to the service composition and routing problem, which breaks down the overall problem into two sub-problems and solves them separately. As shown in Figure 4.5, separate service composition and routing (SSCR) breaks down the overall problem into two parts: service composition and service routing. First, according to the services available in the network, a planner is used to find out the optimal service composition of the request (i.e., a sequence of services that can satisfy the communication requirements with the minimum number of steps) [90]. Then, the optimal routing (a path that can traverse the required services in order with the minimum delay) is calculated using the service routing algorithm presented in [53].

However, a fundamental limitation of the SSCR is that it handles the two sub-problems separately. *Separate optimization may not lead to a global optimum.* To address this issue, we present a solution that can figure out the optimal path for the service composition and routing problem by considering the two problems as a whole. As shown in Figure 4.5, the combined service composition and routing (CSCR) takes everything into consideration, including the input/output characteristics, service library, network topology, and service distribution, and combines all these factors to calculate the routing.

Our framework design (Figure 4.4) enables us to reduce the service composition and routing problem to a planning problem. Both services and links can be regarded as **actions** and can be fully described by:

- Preconditions: logical conditions that need to be satisfied in order for the actions to be executed. In our system, the preconditions include both data semantic requirements and the location requirements. For example, a video transcoding service that transcodes from MPEG to H.264 can only be executed when the packet is in MPEG format and it is at a node capable of providing the service.

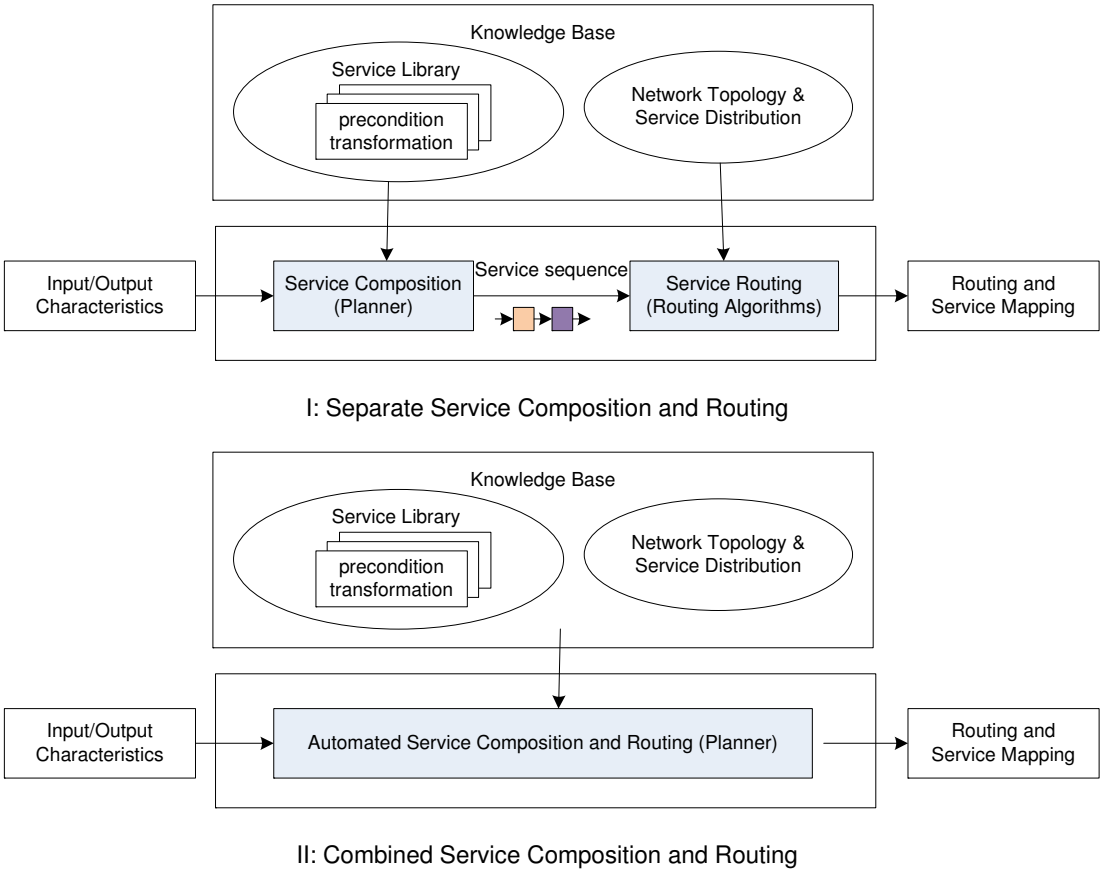


Figure 4.5. Two Solutions: SSCR vs. CSCR.

- Transformation: The function it performs and the transformation it causes. For example, it may refer to a link transferring packets from one node to another, a service causing changes in the data and communication formats, or an increase of the delay.

In general, the service composition and routing problem can be represented as the following planning problem: Given a set of actions A, a set of all possible states Ψ, an initial state $I \subset \Psi$, and a goal state $T \subset \Psi$ that the planner attempts to achieve, the task is to achieve the goal state T from the initial state I by performing a series of actions. In the context of networks with data-path services, A is the set of network services and links. Ψ is equivalent to the set of all possible states combining data semantics and the packet location. I is equivalent to the initial packet location and its semantics. T is the targeted destination and semantics. The output of planning is a series of actions (links and services) that form an optimal route for the service composition and routing problem.

The planner used in the proposed automated service composition and routing system is LPG [46], a fully automated domain-independent planner. It is based on local search and planning graphs and can solve both plan generation and plan adaptation problems. LPG uses the PDDL (Planning Domain Definition Language) for describing domains and problems. PDDL is an action-centric planning domain and problem description language with a LISP like syntax. The description of the semantics of actions, their preconditions and postconditions is greatly simplified in PDDL. A planning problem is described using a domain description that includes the actions, which essentially represent the behavior of the system, and a problem description, which specify the goals to be reached. The language is sufficiently structured to enable automation of domain and problem generation for our implementation.

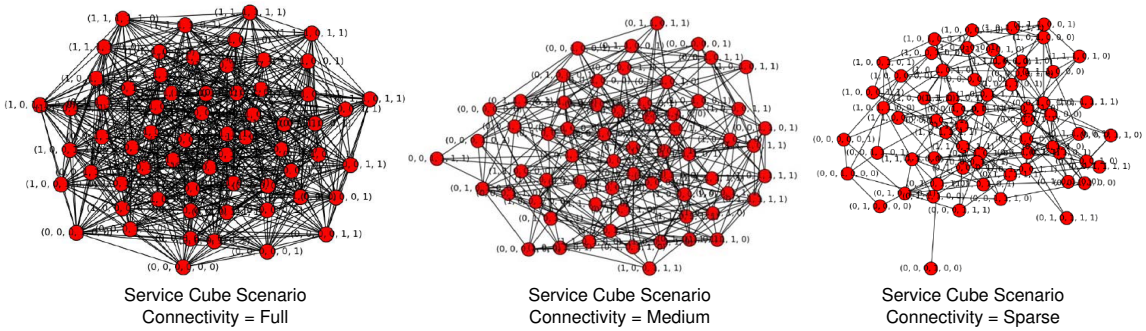


Figure 4.6. Synthetic Benchmark Scenarios. (L=6, h=2, r=0.5, $\Omega = \{0, 1\}$)

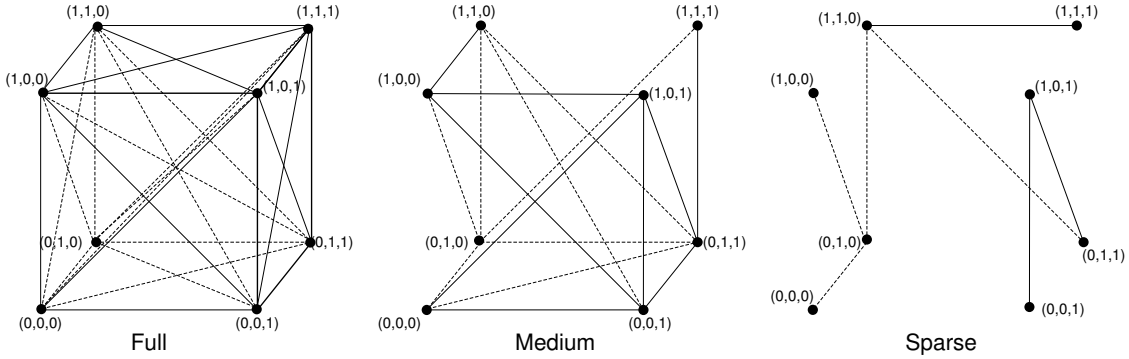


Figure 4.7. Synthetic Service Hypercube Graphs. (L=3, h=1, r=0.56, $\Omega = \{0, 1\}$)

4.4 Results and Analysis

In this section, we evaluate the performance of CSCR based on different synthetic and practical network scenarios, and compare the results with those of SSCR.

4.4.1 Synthetic Service Hypercube Benchmark

In order to fully evaluate algorithms and techniques for automated service composition, a benchmark is needed to provide a standard on which to develop and test new composition algorithms and validate existing ones. Such a benchmark should be flexible enough to generate various representative test scenarios. To address this need, we introduce Synthetic Service Hypercube Benchmark, which is later used in our evaluation to generate synthetic test scenarios. To the best of our knowledge, this is the first benchmark for automated service composition systems in the networking domain.

4.4.1.1 Synthetic Service Hypercube Graph

Let S be a set of atomic and composite services available in the system with every service represented by S_j, their preconditions, p_j and postconditions, o_j. Composite services are single services that can also be represented by combining multiple atomic services. Both p_j and o_j are represented by a tuple of L dimensions, $(\delta_{L-1}, ..., \delta_1, \delta_0)$ representing the leaf nodes in the semantic tree representation (Refer Fig. 4.2) and each δ_i where, $L - 1 \geq i \geq 0$ can take the set of values Ω_i for that dimension. The number of dimensions L, defines the service hypercube graph, with each node of the graph representing a tuple and the edges representing the services that change one or more dimensions in the tuple. The connectivity of the hypercube determines the relationship between the services.

Let $H_{V,E}$ be the service graph for L dimensions where V and E are the nodes and edges. The values of $|V|$ and $|E|$ depend on the set of values Ω_i that each dimension i can take. For example, L dimensions where each dimension takes on binary values results in a service graph with 2^L nodes. The value of $|E|$ is essentially decided by two parameters: the connectivity model and the hamming parameter. The hamming parameter decides the number of nodes of the service graph required to be traversed to reach the destination from the source. For example, a hamming parameter of $h = 2$ implies that any combination of source and destination nodes in the service graph can be reached by either using a single composite service or a combination of two atomic services.

4.4.1.2 Connectivity Models

We define three connectivity models for our benchmark:

- **Full Connectivity** service graphs are essentially small-world networks characterized by high connectivity, where the network diameter specifies the longest

shortest distance between the nodes. Thus, the destination node in the graph can be reached using a small number of services, either composite or atomic.

- **Medium Connectivity** service graphs are an example of random networks of N nodes, where each pair of nodes are connected with a probability, r, which decides the number of edges, $E(r) \subset E$ that can be included in the service graph. However, the edges are added such that the service graph is connected.

- **Sparse Connectivity** service graphs are similar to Medium Connectivity service graphs. The difference being the presence of services (edges) in the service graph, the removal of which renders the graph disconnected. This feature represents tuple dimensions that can only be achieved by one service.

Figure 4.7 shows the three connectivity models for $L = 3$ dimensions. The hamming parameter chosen for Full connectivity service graph is $h = 1$. For the medium connectivity service graph, the r parameter takes the value 0.56.

4.4.2 Synthetic Benchmark Scenarios

All of our synthetic scenarios are based on a network topology with 96 service nodes organized into 12 ASs such that each AS has 8 nodes. We simulate the decision making system using both CSCR and SSCR, and compare them under three different service scenarios. Figure 4.6 shows three synthetic scenarios generated by different connectivity models, with $L = 6$ dimensions and $\Omega = \{0, 1\}$ (i.e., data and communication characteristics are presented in 6-dimensions with each dimension taking on binary values). The full connectivity hypercube service graph represents networks with numerous redundant services, while the sparse connectivity hypercube graph is representative of networks with just enough services to satisfy the communication requirements.

For each scenario, network services defined by the synthetic benchmark are randomly distributed inside the network. And a total of 1000 connection requests are

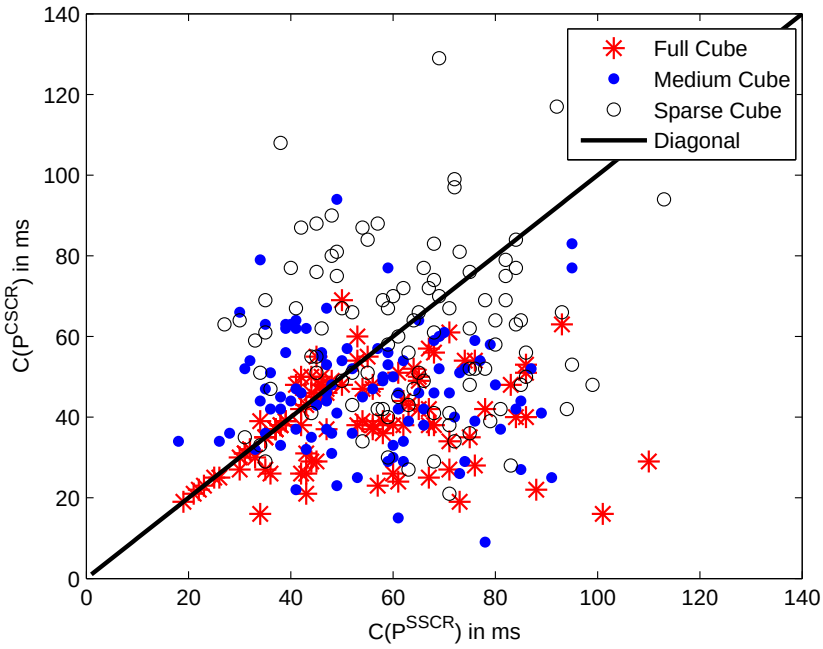


Figure 4.8. Path Cost under Synthetic Scenarios (CSCR vs. SSCR).

generated randomly. Two versions of the decision making system are implemented, based CSCR and SSCR, respectively. The same flow requests are sent to the two systems and total cost of their final routing decisions are compared. In our experiments, the total cost includes both link delays and service processing delays in milliseconds.

Figure 4.8 shows the path cost for SSCR $C(P^{SSCR})$ on the x-axis and the path cost for CSCR $C(P^{CSCR})$ on the y-axis. Each data point represents one connection request. Data points below the diagonal for a connection imply that CSCR has a better performance compared to SSCR. An interesting observation can be made from the performance comparison graph. In case of the Full Connectivity Synthetic scenario, CSCR performs better than SSCR most of the time. However, in the Sparse Connectivity Synthetic scenario, CSCR does no better than SSCR. The reason for the disparity in performance is due to the nature of the synthetic graphs themselves. Full connectivity implies large number of redundant services and multiple routes through

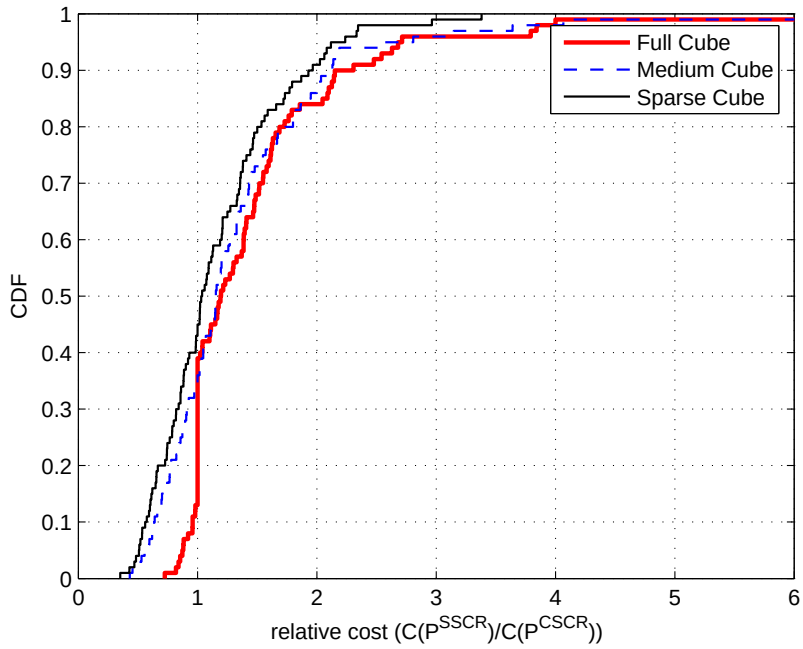


Figure 4.9. CDF of Relative Path Cost under Synthetic Scenarios (CSCR vs. SSCR).

the network, which translates to an increase in the probability of LPG finding the optimal or a near-optimal path.

The same trend can be seen more clearly from Figure 4.9, which represents the Cumulative Distribution Function (CDF) for the relative cost. The relative cost here is defined as the cost of the best path found by SSCR to that of CSCR. When the relative cost is larger than 1, it represents the situation where CSCR finds a lower cost path. The larger the value of x, the better the performance of CSCR is.

Figure 4.9 also compares the performance of the two techniques. For the full connectivity benchmark scenario, CSCR outperforms SSCR. However, in case of sparse connectivity, for about half of the cases, SSCR does better than CSCR.

There is a fair amount of disparity in the plan generation times between CSCR and SSCR. For the experiments, the average plan generation times for CSCR were 4.75ms, 2.00ms and 1.35ms for full, medium and sparse hypercube graphs, respectively. SSCR took 0.14ms, 0.09ms and 0.07ms for full, medium and sparse hypercube

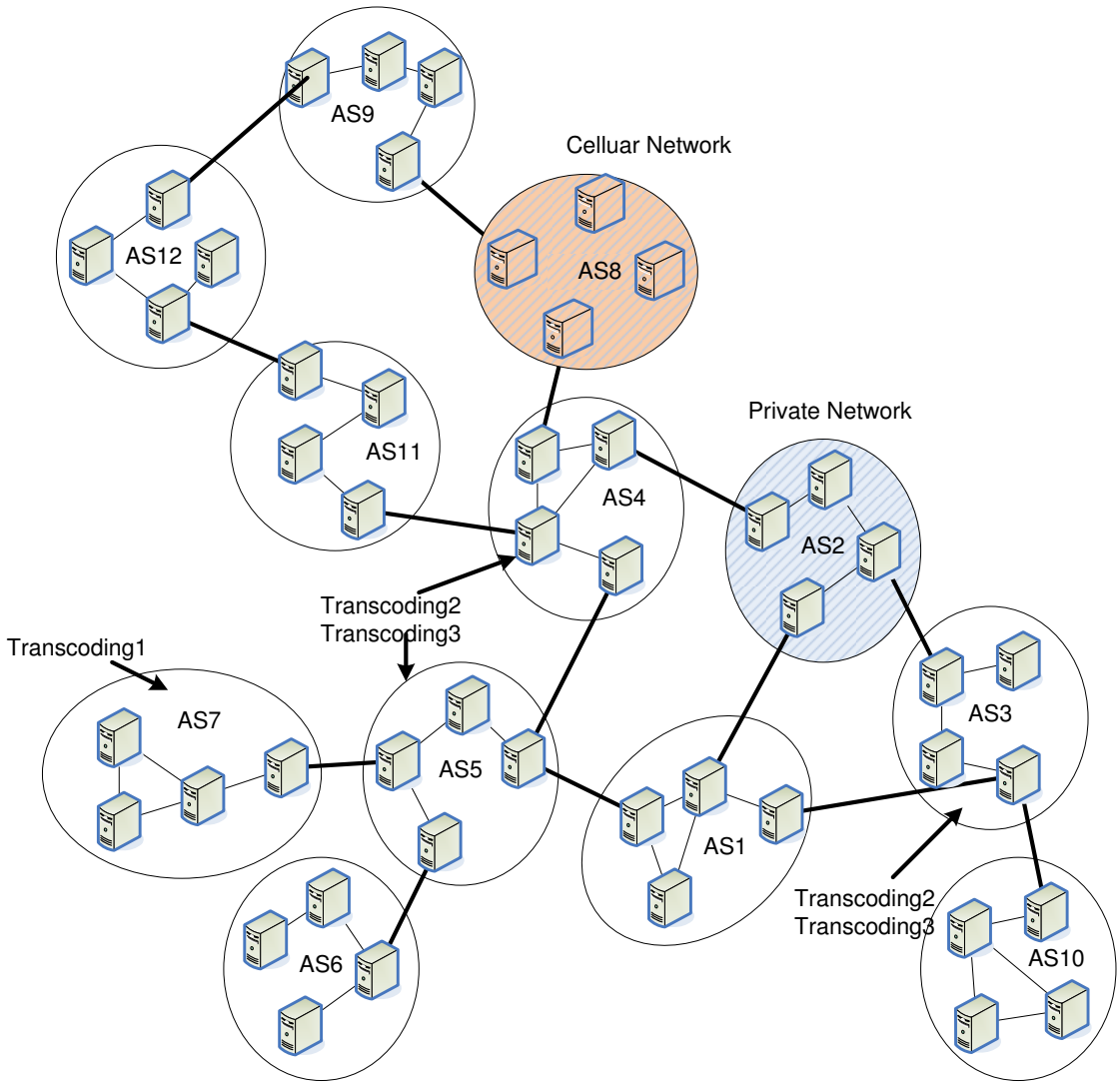


Figure 4.10. Practical Network Scenario.

graphs, respectively. This disparity is primarily because CSCR has to account for both the network topology and service cube graphs while generating the plan. The planner in the SSCR only needs to account for the service cube graph while the routing algorithm takes care of the service placement and routing.

4.4.3 Practical Scenarios

In this section, we further illustrate with practical scenarios the benefits of CSCR over SSCR.

The topology of the network is shown in Figure 4.10, and consists of 48 nodes within 12 ASs. Each AS has four service nodes. Among all the ASs, AS 2 is a private network, where every connection through is automatically tunneled through a private security channel (i.e., encryption and decryption services will be enforced on the packet traveling through it). AS 8 is a cellular network with bandwidth restrictions and therefore all packets through AS 8 has to be compressed and later decompressed. The decision making systems take into consideration these policies while formulating the least cost path. Three transcoding services, S_1, S_2 and S_3 have been implemented in the network. S_1 converts a video from format X to Z. S_2 converts a video from format X to Y. Similarly, S_3 converts a video from format Y to Z. Therefore, the decision making system, given the source format X and destination format Z, can choose between S1 as a single service to meet the requirement or combine S_2 and S_3 to meet the same. Also, S_2 and S_3 are redundant all through the network (e.g., in AS 3, AS 4, AS 5, etc.). However, S_1 is implemented in a pool of servers limited to AS 7.

We generated two types of requests connecting every possible source and destination pair in the network:

- Source node to destination node requests without any data-path services.
- Source node to destination node requests with transcoding from video format X to format Z.

Figure 4.11 compares the performance of CSCR with that of SSCR. Observe that CSCR consistently outperforms SSCR in most of the cases. This is observed in the CDF plots of Figure 4.12 as well. CSCR outperforms SSCR for 80% of the requests for requests that require transcoding. This is because the service composition stage does not account for delays while generating the service sequence. It only accounts for the number of steps needed to achieve the goal. This can result in a huge detour

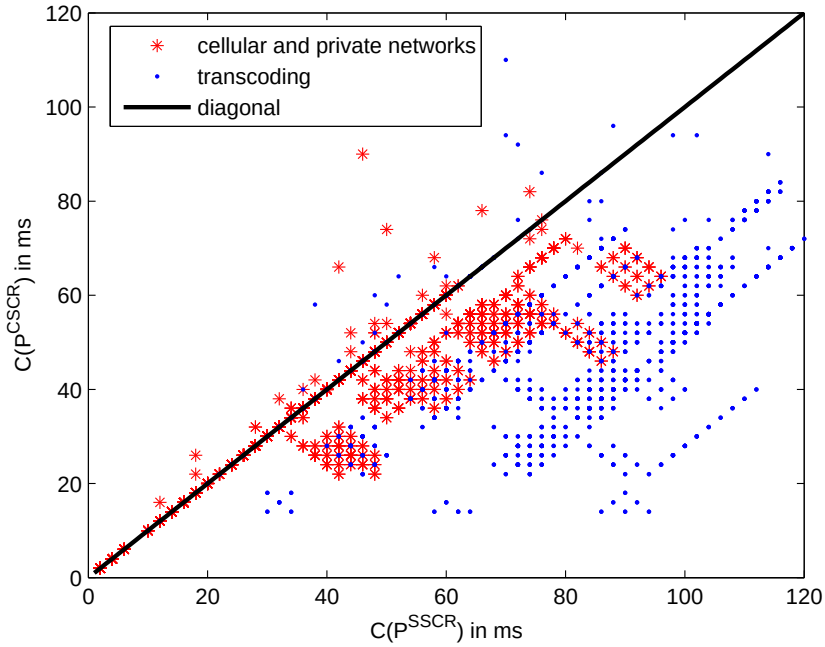


Figure 4.11. Path Cost under Practical Scenario (CSCR vs. SSCR).

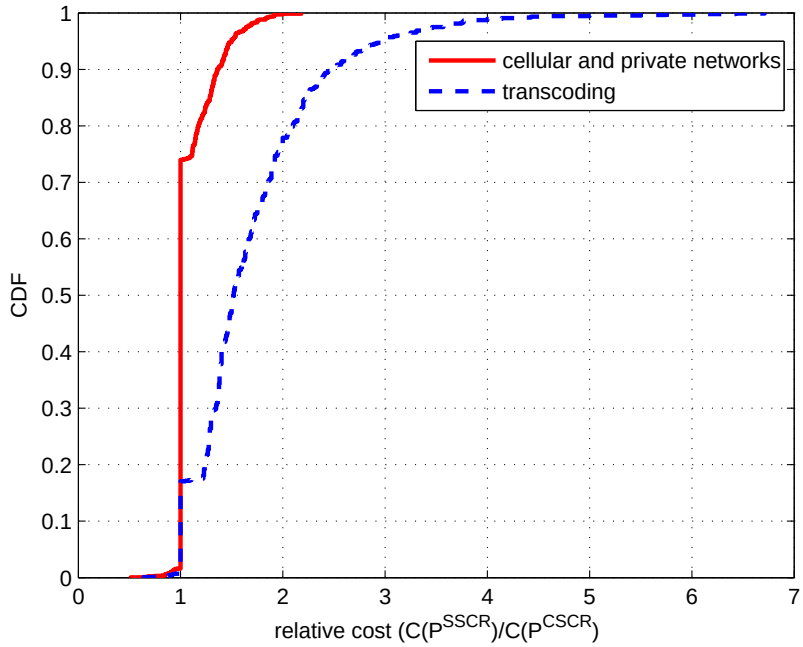


Figure 4.12. CDF of Relative Path Cost under Practical Scenario (CSCR vs. SSCR).

later in the routing stage. CSCR, on the contrary, takes into account both link delays and service processing delays and thus outperforms SSCR.

4.5 Related Work

Composition of protocols and services has been studied in the context of the existing Internet as well as next-generation Internet. Configurable protocol stacks [15] and protocol heaps [19] have been proposed as a solution to statically compose novel protocol combinations. More dynamic approaches has been proposed in [112] and [35], where composition can be performed on a per-flow basis. The latter uses composition rules and constraints to determine valid compositions [106]. In our work, we attempt to determine valid compositions by input/output format characteristics of the data instead of explicitly enumerating all possible mutual constraints between pairs of protocol features.

In research related to automated service composition, various methods have been proposed in the area of web application composition. Most of these methods fall under the category of AI Planning and Theorem Proving. As far as we know, our work (presented in Section 4.2) is the first to use planning to automate the service composition in the context of networks with data-path services.

Routing in networks with data-path services becomes more complicated than the shortest path problem since it needs to consider both cost and service mapping. In our work, this problem is called as service routing problem in our work. Choi et al. first developed a global algorithm to solve the service routing problem [26]. This work was later extended to consider resource constraints [25], which has been proven to be a NP-complete problem. Two other global algorithms was proposed in Section 3.3 to solve the service routing problems with resource constraints. The performance and tradeoff of the algorithm are evaluated and canalized under different network scenarios. Moreover, a decentralized algorithm and a matching routing protocol is

designed and evaluated to solve the service routing problem in large-scale networks in Section 3.4.

As far as we know, our work in Section 4.3 is the first to consider the service routing and service composition jointly and to solve the problem using a planner based decision-making system.

4.6 Summary

This chapter presented a discussion of the service composition in service-centric networks. An important design principle of the service-centric network architecture design is to provide basic packet processing blocks, network services, inside the network. By arranging different service composition along the data-path of a connection, the network can setup customized connections to satisfy different communication requirements. Thus, how to select the "best" service composition according to service availability and connection requirement becomes an important technical challenge in implementing a service-centric network. We call this as service composition problem.

A simple version of the service composition problem is to define the "best" composition by the number of services in the output composition. Thus, the service composition problem is simply to find a shortest sequence of services that can satisfy the communication requirements of a given connection request. First, a common framework for expressing the communication requirements, available services, and service features are presented. Using this framework, the automation of service composition and validation can be implemented by reducing the problem into the corresponding planning problem. The effectiveness of this approach is illustrated by three practical examples.

However, in service-centric network, it is more common that the "best" composition refers to the one that can achieve the least-cost path in terms of end-to-end routing delay. This problem is actually an extended version of the previous service

composition problem, which includes the routing considerations into service composition. This problem is thus called service composition and routing problem in our work. We believe a better solution to this problem can be achieved by combining the service routing and service composition into one problem (CSCR) than by considering the two problems separately (SSCR). Following this logic, we designed a novel decision making system, which automates the process of finding the optimal or near-optimal solutions for the service composition and routing problem. We implemented the design based on the LPG planner and the PDDL language. A synthetic benchmark is also developed to help comprehensively evaluate the performance of our design. The results under both synthetic and practical scenarios validate our belief and show that the CSCR works better than the SSCR in most of the scenarios.

We believe our work in service composition is an important step toward make networks with advanced data-path services a practical reality.

CHAPTER 5

DATA-PATH RESOURCE MANAGEMENT

5.1 Introduction

Service-centric network architecture requires router-based processing functionality to permit services and flexibility inside the network. From the technology point of view, this requirement is being met by programmable routers, which is based on recent progress in embedded processing systems – Network Processors.

Network processors (NPs) provide programmable packet processing capabilities on modern routers. This ability to modify and customize the data path in these systems provides a vehicle for network services that go beyond simple packet forwarding. Routers can be enabled to provide QoS routing [119], advanced firewalling and intrusion detection [70], SSL termination [67], and numerous other functions that are more suitable for implementation inside the network than on end-systems. Our service-centric network architecture proposes to further expand processing capabilities on routers and make packet processing services a first-class networking function. These trends illustrate the importance of network processors as enabling platforms.

Current network processors are implemented as system-on-a-chip multiprocessors. The use of several to dozens of simple, parallel RISC processors provides the computational power to handle Gigabit per second data rates. The simplicity and repetitiveness of network processing tasks ensures that a high level of parallelism can be achieved. One of the main challenges in using network processors is to program them in a way that fully explores the capabilities of the system. The tight interactions between processors, co-processors, on-chip and off-chip memory, and other shared

components cause a lot of runtime issues that need to be considered by a programmer. Based on analytical models and simulation, workloads can be optimized for traffic scenarios by statically allocating sufficient resources.

The main problem with this approach is that network traffic changes dynamically. With changing proportions of different types of packets, processing requirements on the network processor change, too. In our paper [55], we show that using a static allocation of processing tasks for a realistic packet trace with four types of applications achieves only 20%–60% utilization. This becomes worse if more diverse types of processing are considered as it can occur on more advanced routers (i.e., tens of different packet processing options [36]). Thus, it is necessary to consider dynamic adaptation of processor allocations during runtime.

The concept of runtime support for network processors is not new [68,114]. As has been noted in this related work, there are considerable differences between runtime systems for network processors and conventional operating systems for workstations and server. In particular, the numerous parallel processor cores with limited instruction store make the dynamic adaptation problem complex. A runtime system needs to be able to allocate tasks to processing resources in such a way that current traffic patterns can be processed efficiently. Due to the inherent cost of reprogramming processing components, a particular configuration needs to be maintained for a certain amount of time. Thus, a given allocation needs to be somewhat "predictive" of a short window of future traffic.

Several approaches that attempt to solve the problem of allocating processing tasks to system resources have been published [68,88,114]. Unfortunately, it is very difficult to compare the performance of these runtime support solutions since they make a variety of assumptions on the underlying system structure and they use different traffic and benchmark applications for performance evaluation. This is a major problem as fair comparison is important for enabling progress in this area of research.

In this chapter, we propose a novel methodology to systematically evaluate runtime systems for network processors. Our contributions are:

1. **Definition of Dynamic Workloads**. We develop a model for describing workloads for network processors in scenarios where network traffic causes changes in processing requirements. We provide examples from realistic applications and network traces and provide a mechanism for generating synthetic workloads that can be used in benchmarks.

2. **Queuing Model for Analytical Evaluation of Runtime Systems**. We present a model that can determine the performance of different runtime support systems for network processors. We show how it can be used to determine a range of system performance metrics including throughput, processor utilization, and average number of packets in system.

3. **Comparison of Existing Mapping Algorithms**. We illustrate the generality of our model by comparing two existing runtime support approaches that have been published previously by two different research groups.

Parts of this chapter are published in [55] and [56].

5.2 Evaluation Methodology

Our overall methodology for evaluating runtime systems for network processors is outlined in Figure 5.1. It can be divided into three main components:

- **Dynamic Workload Characterization**. This aspect of the methodology consists of two components, a description of the individual processing steps ("applications") and a description of the variation in processing that depends on changes in network traffic.

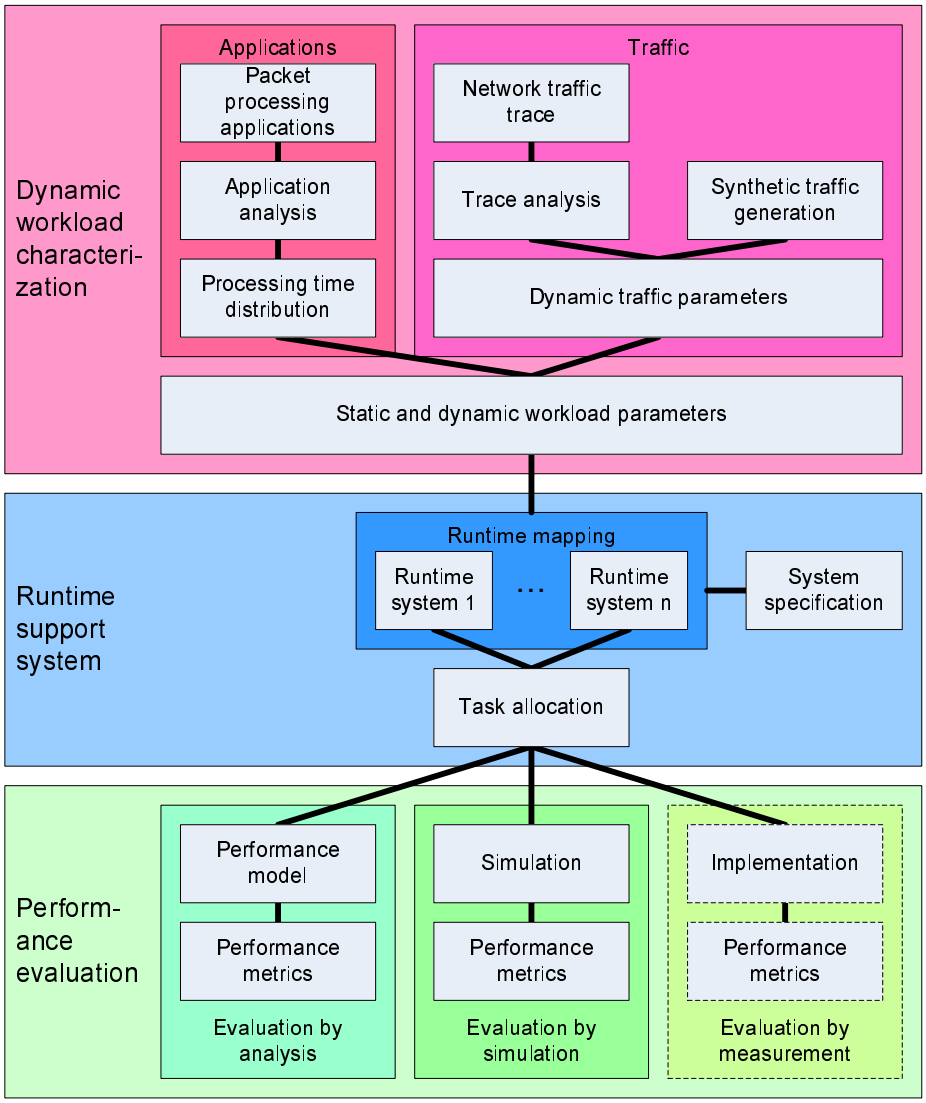


Figure 5.1. Components of Evaluation Methodology and Their Interactions.

- **Runtime Support System**. This component of the methodology handles the allocation of processing tasks to system resources as specified by different runtime systems. Typically, this process consists of a general system specification (e.g., number and capabilities of processors) and an algorithm that determines which application is assigned to which processor ("mapping algorithm"). The result of this process is a dynamic task allocation.

- **Performance Evaluation**. The task allocation obtained from the runtime support system needs to be evaluated for comparison between different runtime support systems and for comparison to the optimal performance. In principle, there are three methods for evaluation: analysis, simulation, and measurement. In this work, we focus on analysis and simulation. The other method is conceptually possible with our methodology, but is not further explored in this paper (and thus shown with dashed lines).

The main challenge in developing such a methodology lies in the differences in assumptions on workloads and system specifications between different existing runtime systems. We therefore focus on the following principles in our methodology:

- **Simplicity**. We do not want to make more assumptions than necessary. There are numerous details and special cases that could be considered in the evaluation process. In the current state of research on runtime support for network processors, a lot of fundamental questions about task allocation, dynamic remapping, etc. are not yet fully explored. We believe it more important to address these high-order issues before refining the methodology to consider less important special cases at the cost of generality.

- **Supports All Types of Evaluation**. As shown in Figure 5.1, runtime systems can be evaluated via analysis, simulation, and measurement. While we only pursue analytical evaluation and simulation in this paper, we have designed

the methodology in such a way that it can also be used in the context of measurement. For example, system performance data such as throughput, average queue length, and average packet delay could be measured on a prototype system. However, to explore fundamental design choices, analysis and simulation are more suitable.

- **Supports Hypothetical Scenarios**. Currently, there is no benchmark for NP runtime system. It is therefore important that a thorough evaluation methodology consider a broad range of scenarios, both existing (e.g., realistic network traffic) and hypothetical (e.g., extreme scenarios that may occur in the future).

In the following sections, we present the evaluation methodology that considers the above goals. First, we present a model to describe the workload and the system. Second, we show how to apply a queuing network model to estimate the performance of a particular configuration under different traffic scenarios. Then, we show results from a comparison of two specific runtime support systems that have been published previously by two different research groups. Finally, we use OPNET simulation to validate the result of the analytical queuing network models and explore the tradeoffs of these two different evaluation methods.

5.3 Workload Model

The workload of a network processor system can be divided into two categories. First, there is processing workload. In most NP systems these components are not one single monolithic piece of software, but a collection of smaller "applications" (e.g., packet classification, IP forwarding, intrusion detection, etc.). Second, there is network traffic that exercises the network processor system. Depending on the constitution of traffic, different applications are used more or less. Thus, representative traffic is an essential aspect of defining the workload.

5.3.1 Applications

The combination of all applications present in the NP system is represented by a workload graph $W = (\Theta, \Gamma)$ with vertices Θ, representing tasks, and edges Γ, representing transition probability. Each task $\theta \in \Theta$ corresponds to a set of processing instructions. We do not make any assumption about the granularity of these tasks – they may be as complex as entire network processing applications or as small as individual RISC instructions. The transition probability γ_{ij} between two tasks θ_i and θ_j gives the probability that task θ_j follows task θ_i in the execution of a particular packet.

While Γ determines the mix of the processing workload, it does not specify the packet rate that is injected into the system. We use Λ to represent that rate measured in packets per second.

5.3.2 Traffic

To introduce dynamic changes to the workload, we permit the edge matrix Γ and the system packet arrival rate Λ to change over time. We use the superscript t (i.e., Γ^t, Λ^t) to denote the state of Γ and Λ during the interval $[t, t + \Delta)$, where Δ is the duration of the interval. Note that the set of tasks, Θ, does not change over time. However, selected tasks can be activated and deactivated by adjusting Γ^t, which controls the amount of traffic that is sent to a given task.

5.3.3 Synthetic Traffic

It is important to obtain realistic values for the traffic characterization Γ^t as it defines how frequently different applications are used and how they are "chained together" when packets require multiple processing steps. As we show below, this information can be extracted from packet traces collected for network measurements. But as we have argued above, it is also important to generate scenarios that are

different from today's network traffic and illustrate requirements of next-generation networks. That leads to the need for generating "synthetic" traffic scenarios.

How to generate network traffic that has realistic properties has been explored extensively in the network simulation community. Our goals are somewhat simple since we do not need to generate network traffic with all levels of detail (e.g., correct source-destination pairs, IP addresses, and flow sizes as in [69, 94]). Instead, we just need to generate scenarios that determine realistic values for Γ^t (i.e., the proportions of packets that traverse different paths in the graph) and Λ^t (i.e., the packet rate injected into the system).

We base the generation of a synthetic traffic trace on Holt-Winter forecasting as applied to networking in [20]. Holt-Winter forecasting decomposes a time series into three time-variant components: a baseline $a(t)$, a linear trend $b(t)$, and a seasonal trend $c(t)$. The prediction of the next element $\hat{x}(t+1)$ is

$$\hat{x}(t+1) = a(t) + b(t) + c(t+1-m), \tag{5.1}$$

where m is the period of the seasonal cycle. The parameters of the estimation are computed using exponential smoothing:

$$a(t) = \alpha(x(t) - c(t-m)) + (1-\alpha)(a(t-1) + b(t-1)) \tag{5.2}$$

$$b(t) = \beta(a(t) - a(t-1)) + (1-\beta)b(t-1) \tag{5.3}$$

$$c(t) = \zeta(x(t) - a(t)) + (1-\zeta)c(t-m) \tag{5.4}$$

where α, β, and ζ are smoothing parameters for baseline, trend, and seasonal components ($0<\alpha$, β, $\zeta<1$). Note that $x(t)$ can represent number of packets or number of bytes depending on what is more suitable. In this paper, we assume $x(t)$ represents the number of packets in interval $[t, t+1)$. Also, we assume $x(t) \geq 0$ without explicit notation.

Table 5.1. Traffic Scenarios for Workload Characterization.

Scenarios	a	b	c	m	c_X
Static	any	0	0	n/a	0
Low variation	any	0	small	large	small
High variation	any	0	large	small	large
Transition	a_1 large	b_1<0	any	any	any
	a_2 small	b_2>0	any	any	any

By applying the concepts of Holt-Winter forecasting for generation, we can obtain a traffic sequence:

$$x(t) = a + b \cdot t + c \cdot S(t \bmod m) + N(\sigma), \tag{5.5}$$

where a, b, c, and m are static Holt-Winter parameters for baseline, the linear trend, and amplitude and period of the seasonal trend, respectively. Since Holt-Winter forecasting does not specify the "shape" of the seasonal trend, we introduce function $S() : [0, m) \rightarrow [-1, 1]$, which allows the specification of any type of function. The shape function should be normalized to produce values in the range $[-1, 1]$ with zero mean to allow the specification of the amplitude by parameter c. Randomness in the traffic is achieved by the "noise" function N, which has a mean of zero and is parameterized by the standard deviation σ. The distribution of $N()$ can be chosen arbitrarily and scaled appropriately with parameter σ.

We can use n generations functions $x_1(t) \ldots x_n(t)$ to describe n different classes of traffic, each with its own set of parameters $(a_i, b_i, c_i, m_i, S_i(), \sigma_i)$. The total traffic Λ^t is simply the sum of all classes: $\Lambda^t = \sum_{i=1}^{n} x_i(t)$. Given that S and N have zero mean, the expected rate at a given point in time t can be estimated by the baseline and linear parameters: $E[\![x(t)]\!] = (\sum_{i=1}^{n} a_i) + (\sum_{i=1}^{n} b_i) \cdot t$.

The set of parameters $(a_i, b_i, c_i, m_i, S_i(), \sigma_i)$ for each class of traffic gives us a large number of possible configurations. To ease the generation of traffic scenarios, we can make the following simplifications:

- **Identical Shape Functions**. While we allow different periods m_i for seasonal trends, we may simplify that all traffic classes follow the same seasonal shape $S() = S_1() = \ldots = S_n()$ (e.g., sinusoidal or step function).

- **Noise Proportional to Baseline**. The parameter for the standard deviation σ_i determines how much randomness is superimposed on the traffic. In many cases, we expect that traffic classes with a small baseline have a variation that is also small (i.e., proportional to the baseline). Thus, we can use a single parameter c_X, the coefficient of variation, to determine $\sigma_i = c_X \cdot a_i$. Note that c_X should not be confused with c_i. In cases where there is a significant linear component (i.e., large b_i), it may be more suitable to choose σ_i to be time-variant: $\sigma_i(t) = c_X \cdot (a_i + b_i \cdot t)$.

When generating workloads, we can create a number of different scenarios that are commonly observed in networking. The parameters for these scenarios are summarized in Table 5.1. We assume that the simplifications discussed above with a sinusoidal shape function $S()$ and the use of the coefficient of variation c_X to compute σ_i. The static scenario only requires parameters for the baseline a_i (the period can take any value since $c_i = 0$). For low variation, a small seasonal parameter with a long period and a small coefficient of variation is chosen. Accordingly, high variation is achieved with a large seasonal parameter with a short period and a large coefficient of variation.

5.3.4 Example Workload

In this section, we present an example of a workload with different real and synthetic traffic scenarios. The workload graph for one traffic scenario is shown in Fig-

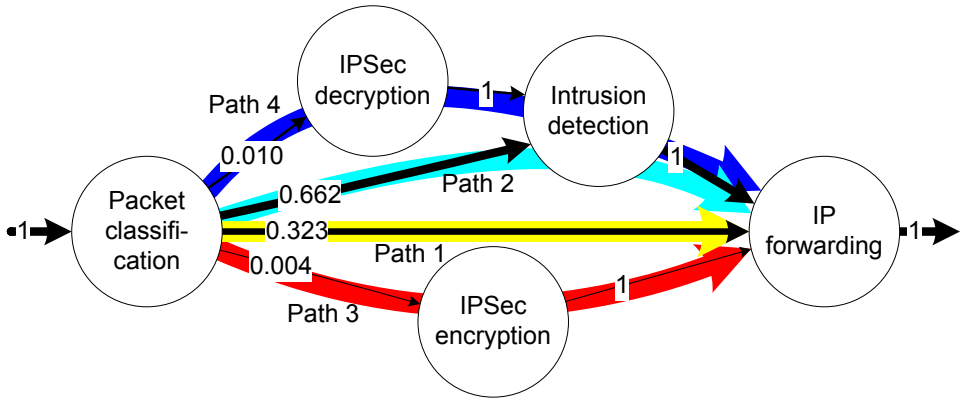
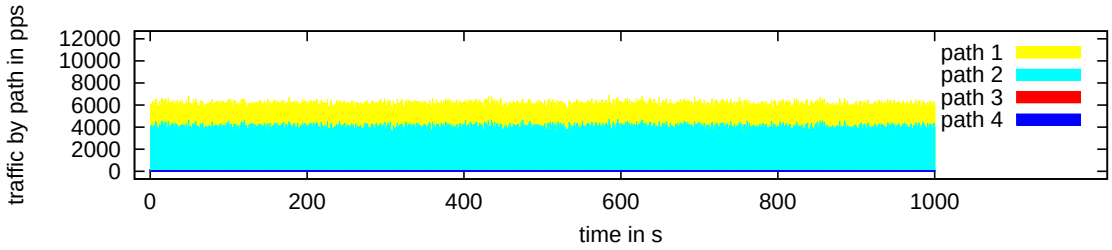


Figure 5.2. Example Workload Graph.

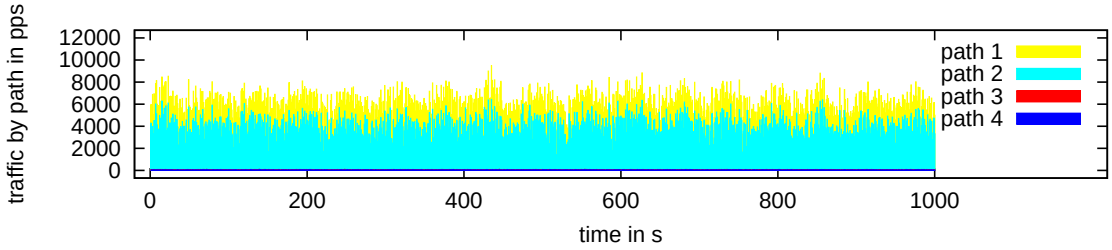
ure 5.2. There are five different packet processing tasks: packet classification, IP forwarding, intrusion detection, IPsec encryption, and IPsec decryption. The black arrows indicate the edges of the W with values that we have obtained from the first second of the measured traffic shown in Figure 5.3(e).

The colored arrows in Figure 5.2 show the paths that different packets take (i.e., the set of applications that are traversed when different packets get handled by the network processor). In our example, we consider four paths that correspond to the following four types of traffic:

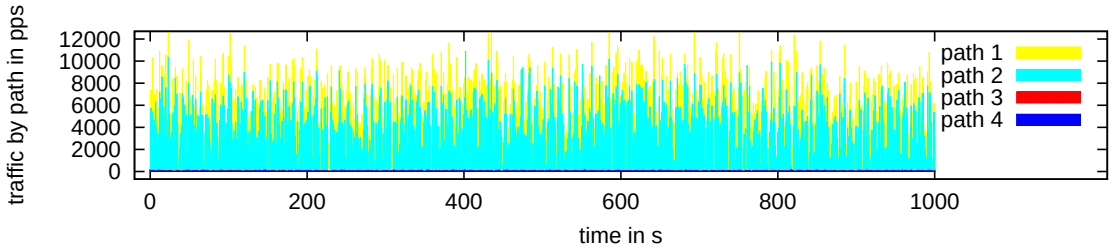
- **Path 1**: packet classification and forwarding is the default handling of packet.
- **Path 2**: packet classification, intrusion detection, and forwarding is a scenario where incoming packets on an edge router are scanned for malware.
- **Path 3**: packet classification, IPsec encryption, and forwarding is a scenario where some outgoing packets are tunneled via a VPN.
- **Path 4**: packet classification, IPsec decryption, intrusion detection, and forwarding is scenario where incoming VPN packets are decrypted and scanned for malware.



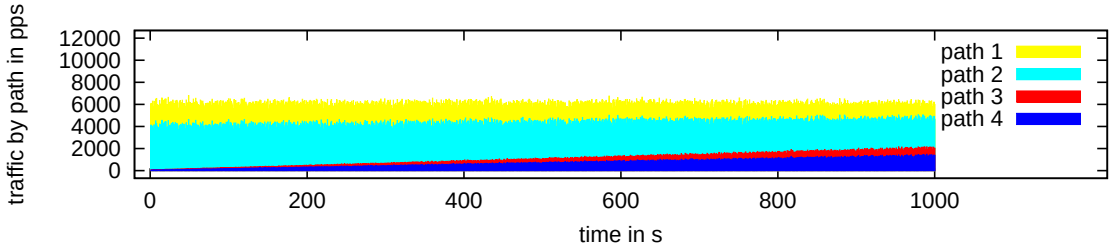
(a) Static (b_i=0, c_i=0, c_X=0.05)



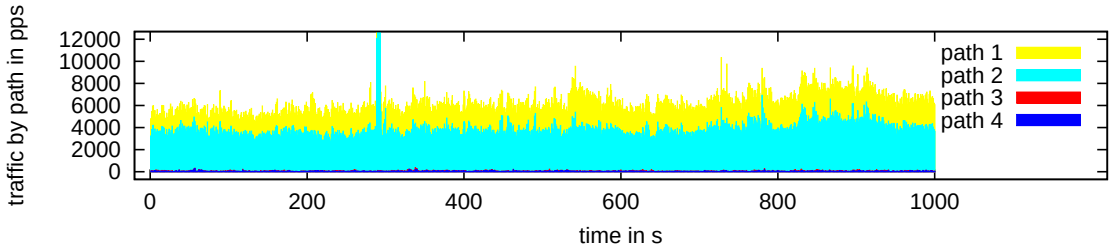
(b) Low Variation (b_i=0, c_i=.1a_i, m_i=40..70, c_X=0.2)



(c) High Variation (b_i=0, c_i=.5a_i, m=8..11, c_X=0.5)



(d) Transition (b_i=−1.33..1.33, c_i=0, c_X=0.05)



(e) Measured LAN Traffic

Figure 5.3. Packets per Second for each Path for Example in Fig. 5.2

The weight of the edges in the workload graph change as network traffic changes. Different traffic scenarios are shown in Figure 5.3 as stacked bar graphs. Packets that traverse the network processor on different paths are shown with the same colors as the paths in Figure 5.2. In the figure, packet counts are shown as cumulative for 1000 seconds. The baseline parameters a_i are chosen to match that of the measured LAN traffic in Figure 5.3(e). Figures 5.3(a)–(d) show instances of the four synthetically generated workloads discussed in Table 5.1. Figures 5.3(e) shows the traffic of a LAN trace. All traffic scenarios are adjusted to have the same long-term average packet rate.

5.4 Runtime System Model

It is very difficult to create an accurate model for a network processor system. Many network processors contain specialized components to accelerate common functions (e.g., hash computation, crypto-processing), I/O components to access network interfaces and memory, and different types of internal interconnects for communication between processors other system units. As we have discussed above, our work aims at obtaining a general understanding of tradeoffs between different runtime systems. It is therefore inherently necessary to generalize the network processor system model to a level that allows a comparison of different implementations.

5.4.1 Network Processor Model

In this paper, we represent the network processor system as a queuing network [18]. Queuing networks are queuing systems that consist of processing elements ("servers") and queues and allow feedback loops and different classes of traffic. Queuing networks exhibit the following properties that make them particularly suitable for obtaining the performance evaluation results that we are interested in:

- **Generality**. A queuing network can represent a wide range of system configurations as we show in Section 5.6.

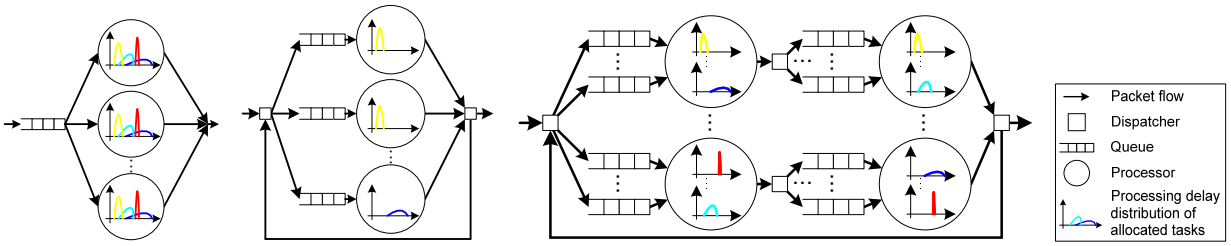
- **Suitable for Analytical Evaluation**. Very few system abstractions lend themselves for easy analytical evaluation. Queuing networks have been used extensively for analytical evaluation and are easy to simulate.

- **Representativeness**. Many network processor systems actually implement their processing tasks in form of processing steps that are interconnected with queues. For example, on the Intel IXP family of processors, scratch memory is used to queue packets between processing tasks.

We use the following notation: The network processor system is modeled by a graph $G = (\Phi, Q)$ with vertices Φ ("processors") and edges Q ("queues"). The size of the queue that connects processors ϕ_i and ϕ_j is given by q_{ij}. If $q_{ij} = 0$, no queue exists between the processors. In our queuing network model, we can support different classes of packets (e.g., packets that traverse different paths in Figure 5.2). If packets from these different classes are queued in independent queues, Q can be replicated for each of the m classes: Q^c, where $c \in \{1, \ldots, m\}$.

In addition to the structure of the queuing network, it is necessary to specify the "service time" that a packet experiences when being handled by a processor. Therefore, we define the execution time of a task θ_i on a processor ϕ_j as $D(\theta_i, \phi_j)$ ("delay"). Since processing is data dependent and thus can vary, we assume D to be a random variable with cumulative distribution function of $F_{D(\theta_i,\phi_j)}(x) = P[\![D \leq x]\!]$ and probability density function $f_{D(\theta_i,\phi_j)}(x) = dF_{D(\theta_i,\phi_j)}(x)/dx$.

This delay model is a very simple approximation of the processing cost on a real system, but can be derived easily [85]. Several aspects of real NP systems are not considered (e.g., memory accesses and multithreading). Some of these aspects can be approximated in the model. For example, $f_{D(\theta_i,\phi_j)}(x)$ can be adjusted for



(a) Ideal Scenario (b) Full Processor Allocation (c) Partitioned Application Allocation

Figure 5.4. System Model Instantiations for Different Runtime Systems.

different memory configurations and load levels. Integrating existing NP performance models [100] [113] or simulation-based approaches [47] with our queuing model can also be considered in the future.

5.4.2 Runtime Mapping

The main purpose of a runtime support system is to determine what part of the workload W^t is to be handled by what processing resources of the system. We refer to this allocation process as "mapping." The details of how and when this mapping process is performed is the main characteristic of a runtime support system and therefore the focus of our study. It should be noted that there are several second-order issues (e.g., placement of data in memories) that also need to be handled during runtime. However, these are usually strongly influenced by the decision where to place the processing tasks. We therefore focus on the mapping of tasks to processors.

The mapping function M places tasks Θ onto processing elements Φ. This relationship is dependent on the workload at time t and thus:

$$M^t : \Theta^t \rightarrow \Phi. \tag{5.6}$$

An allocation of tasks to processing elements causes packets to be sent between processors. The parameter $\lambda_{i,j}^{c,t}$ specifies the rate of class c ($1 \leq c \leq n$) packets sent

into queue $Q_{i,j}$ at time t. We can determine $\lambda_{i,j}^{c,t}$ based on the rate of traffic into the system Λ^t, workload W^t, and mapping M^t.

5.4.3 Example Runtime Systems

To illustrate the system model and runtime mapping process, we introduce three different mapping algorithms. These will be evaluated and compared in Section 5.6. The first algorithm is the ideal baseline case, which is not practical to implement. The second and third algorithm are runtime support systems that have been published by two different research groups. These systems are illustrated in Figure 5.4 (processors are represented as circles showing the processing delay distributions of allocated tasks).

Runtime System I: Ideal Allocation

This system assumes that all processors in the system can process all packets completely. The assumption that all applications are mapped to all processors is unrealistic for a real network processor system due to limitations in instruction store. However, it provides a baseline for the best possible performance that could be achieved in terms of packet delay, load balancing, and maximum data rate that can be sustained by the system. Actual runtime systems will perform less efficiently since task allocation may not match the exact current workload.

The ideal system is illustrated in Figure 5.4(a) In the case of the ideal system, all processor can process all types of packets. Thus, all packets are sent to the same queue, processed by any available processor, and then sent out of the system.

Runtime System II: Full Processor Allocation

This system is based on the runtime support system proposed by Kokku et al. [68]. The main idea of this approach is to allocate entire tasks to subsets of processors.

During runtime, the number of processors allocated to each task is adapted. The decision to adjust the allocation is based on the queue length of packets waiting to be processed. The algorithm also attempts to use as few processors as possible to reduce the power consumption of the network processor.

The queuing network that corresponds to this runtime system is illustrated in Figure 5.4(b). We assume a processor allocation granularity where each processor can only be assigned to process one type of task. The feedback from back to front is necessary to allow packets to traverse multiple processing steps as illustrated in Figure 5.2. Dispatcher nodes are used to direct packet flows accordingly.

Runtime System III: Partitioned Application Allocation

This system is based on the task allocation algorithm proposed by Wolf et al. [114]. In this scenario, tasks can be partitioned across multiple processors. The system is configured as a synchronous pipeline where packets move across stages in synchronous fashion. The mapping algorithm attempts to balance processing allocations across all processor to reduce overhead. The mapping is adjusted at regular intervals to adapt to changes in traffic.

This queuing system is shown in Figure 5.4(c). Each processor may have portions of different processing tasks installed and thus needs multiple queues for different classes of packets. Processing delay distributions in the same color and shape indicate that they are part of the same processing task. The feedback is necessary for packets that traverse multiple processing steps.

5.5 Analytical Evaluation

In the previous section, we have shown how to represent a runtime system as a queuing network where the workload W^t and mapping M^t get adapted over time. As shown in Figure 5.1, there are different approaches to evaluation this system model.

In this section, we present the basis of our analytical evaluation, which is used to derive the results shown in Section 5.6.

5.5.1 M/M/m–FCFS Model

We use the result of Baskett, Chandy, Muntz, and Palacios [11] (BCMP) to analyze the queuing network in our model. These BCMP networks may include several job classes, different queuing strategies, and service times with general distributions. The networks can be open, closed, or mixed. The assumptions for BCMP networks lead to four node types (using Kendall's notation): Type-1: –/M/m–FCFS, Type-2: –/G/1–PS, Type-3: –/G/∞ (IS), Type-4: –/G/1–LCFS PR. The arrival process is assumed to be one or multiple overlapping Poisson arrival streams.

The differences between these four node types lie in the queuing discipline and service time distribution. The queuing disciplines are First-Come-First-Serve (FCFS) for Type-1, Processor Sharing (PS) for Type-2, Infinite Server (IS) for Type-3, and Preemptive Resume (PR) for Type-4. The service time distribution for Type-1 is an exponential distribution and a general distribution for the other types. We have chosen Type-1 nodes for our analysis since network processors typically use FCFS queues between processing resources. An exponentially distributed service time for this node type is a reasonable assumption for analytical modeling.

For determining the equilibrium state probabilities, it has been shown that BCMP networks have product-form solution [11]. The steady-state probabilities π for states ψ have the following form:

$$\pi(\psi_1, \ldots, \psi_N) = \frac{1}{G(K)} d(\psi) \prod_{i=1}^{N} n_i(\psi_i), \tag{5.7}$$

where K is the number of jobs, $G(K)$ is a normalization constant, $d(\psi)$ is the product of arrival rates, and $n_i(\psi_i)$ is a function that depends on the type of nodes used in the BCMP queuing network. For the special case of an open BCMP network where

arrival and processing rates are load independent, we can simplify the steady state probabilities:

$$\pi(k_1, \ldots, k_N) = \prod_{i=1}^{N} \pi_i(k_i), \tag{5.8}$$

where $\pi_i(k_i) = (1 - \rho_i)\rho_i^{k_i}$, $k_i = \sum_{r=1}^{C} k_{ir}$, $\rho_i = \sum_{r=1}^{C} \rho_{ir}$, $\rho_{ir} = \Lambda_r \frac{e_{ir}}{\mu_i}$ (Type-1, $m_i = 1$), and $\overline{K}_{ir} = \frac{\rho_{ir}}{1-\rho_i}$. Details on these parameters can be found in [11] and [18] (with slightly different notation).

5.5.2 Performance Metrics

In our analysis, we focus on the following performance metrics, which are key indicators of the performance of a runtime system:

- **Processor Utilization** ρ. This metric is defined as the fraction of time that the processor is busy. Processor utilization indicates the efficiency at which the system operates.

- **Packets in System** K. This metric indicates how much the queues and processors are utilized in the system. A large value indicates that many packets are queued and that packets experience a large delay when traversing the system.

To obtain these metrics, from the above equations, we adapt the BCMP queuing network as follows.

For the Ideal Allocation system, we assume all the processing resources are identical and can process all types of traffic. Thus, the queuing network reduces to an M/G/m single server model, which we can easily evaluate.

For the Full Processor Allocation system and the Partitioned Application Allocation, we use the BCMP network introduced above with Type-1 nodes. In the Full Processor Allocation scenario, each processor can process one type of task with a given processing rate and deviation. In the Partitioned Application Allocation system, we partition tasks into subtasks and allocate them among different processors.

For simplicity, we assume that the service time distribution of each subtask is exponentially distributed (based on the number of instructions executed). Analyzing a system with subtasks from different tasks on the same processor is not easily possible with BCMP Type-1 network as they require all packet classes use the same service time distribution on a node. Therefore, we replicate each processor multiple times and reallocate subtasks such that only consecutive subtasks from a single task share a replicated processor. To obtain correct performance results, the parameter of the exponential distribution is adjusted such that the mean matches the delay that would have been encountered on the original system without replicas. This process allows us to analyze the performance of the system using the BCMP queuing network.

5.6 Results and Analysis

Using the analytical queuing models described above, we first evaluate the performance of the two runtime systems and the ideal case described in Section 5.4. Then, we compare and validate the analytical results against simulation results.

5.6.1 Setup

We use the following setup to evaluate the runtime systems:

- **System**: 16 processing engines, similar to an Intel IXP2800 network processor. The processing performance of each processor is assumed to be 100MIPS (corresponding to a 600MHz processor with CPI=6 due to memory access delay and other hazards). The queue lengths are assumed to be infinite to see the impact of queuing when observing the K metric.

- **Workload**: Applications are chosen as shown in Figure 5.2 and traffic scenarios are chosen according to Figure 5.3. The length of each trace scenario is 3026 seconds. To scale the overall data rate for some experiments, packet rates are scaled accordingly.

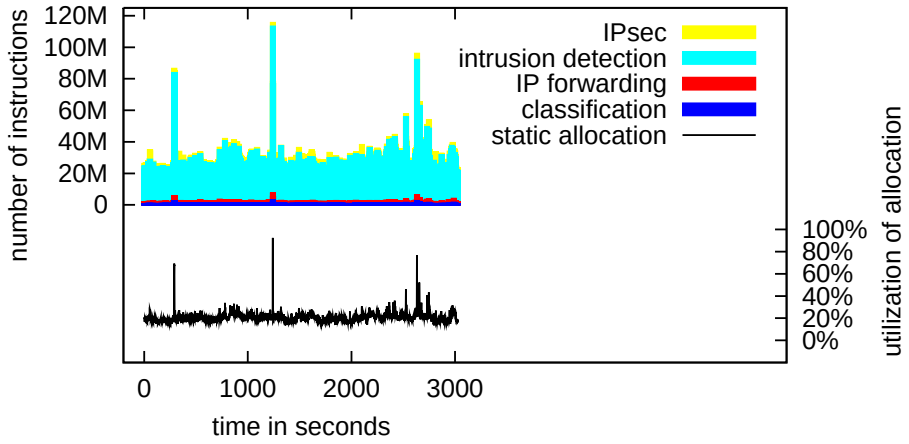


Figure 5.5. Effectiveness of Static Allocation.

- **Other Assumptions**: The partitioned applications for Runtime System III are split into 7–15 subtasks.

5.6.2 Analytical Results

The results presented here are a representative subset of all experiments that we have performed. First, we illustrate the importance of runtime systems by exploring the performance of a static allocation mechanism. Then, we show the behavior of runtime systems over time. Finally, we compare the performance of different runtime systems as data rates increase toward the maximum that a system can sustain.

5.6.2.1 Static Mapping

Figure 5.5 shows the processing demands that are encountered in the measured traffic trace. The packet rate for each application is multiplied by the average number of instructions that are executed, which is directly proportional to processing time. The left y-axis shows the demand for instructions over time as a stacked bar graph.

Assuming a static allocation (i.e., there are enough processing resources allocated to handle the worst case demand for each application), processing resources capable of processing 125.1 million instructions per second are necessary. Since this allocation

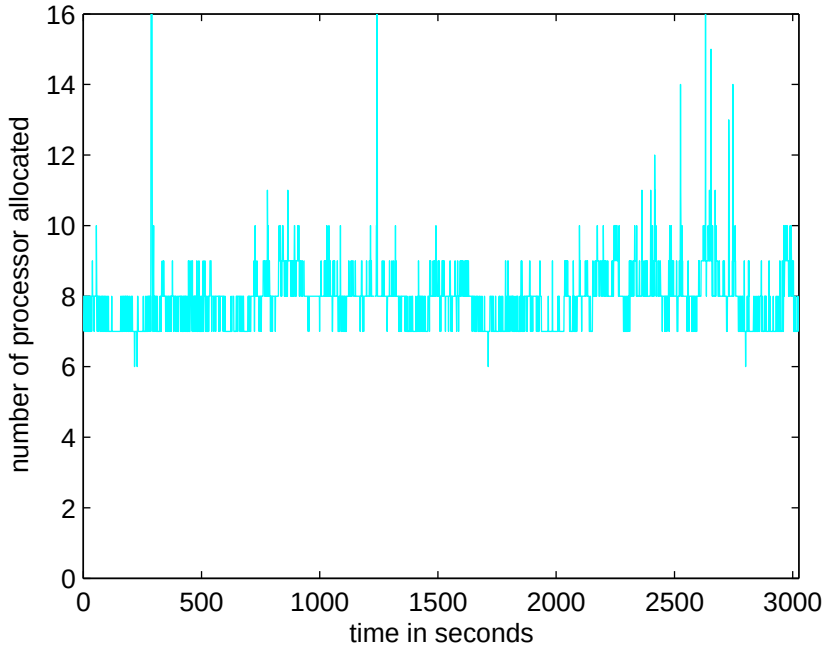


Figure 5.6. Processors Allocated over Time in Full Processor Allocation System.

assumes the worst case, most of the time the system is under-utilized (as shown in Figure 5.5). A utilization of 100% may not even be reached if worst case demands of different applications do not coincide.

These results show that it is indeed important to adapt processing configurations during runtime to make full use of the available hardware (or to achieve comparable data rates with fewer processors). Especially with an increasing number of different services and applications on network processors, this adaptation will become more important.

5.6.2.2 Performance over Time

To illustrate the behavior of a runtime system, it is easiest to consider the case of the Full Processor Allocation system. Figure 5.6 shows the total number of processors that are allocated over time. Note that this is different from the utilization that is achieved for a given allocation. Depending on the processing demand (see Figure 5.5), the allocation spikes during high-demand periods. As the demand de-

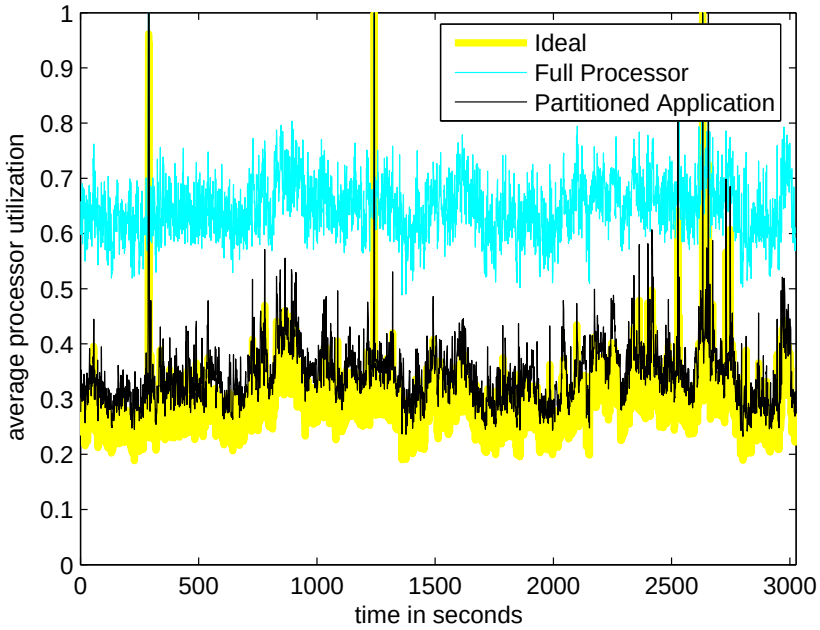


Figure 5.7. Processor Utilization over Time.

creases, processors are not utilized anymore. The allocation among different packet classes also changes accordingly (not shown). For the Ideal system and the Partitioned Application system all processors are in use at all times, but are only partially utilized.

This difference in utilization can be seen in Figure 5.7, where average utilization of the processors is shown over time. In the Partitioned Application Allocation case, applications are reallocated at fixed intervals (as proposed in [114]) of 1 second and the granularity at which traffic demands are estimated is 1%. As the figure shows, the Full Processor Allocation shows a higher utilization on the processors that are allocated to processing. The Ideal system achieves the lowest utilization because all processors are utilized and the allocation allows all processor to process any type of packet. The Partitioned Application Allocation cannot equal the Ideal case due to overhead of fragmented subtasks.

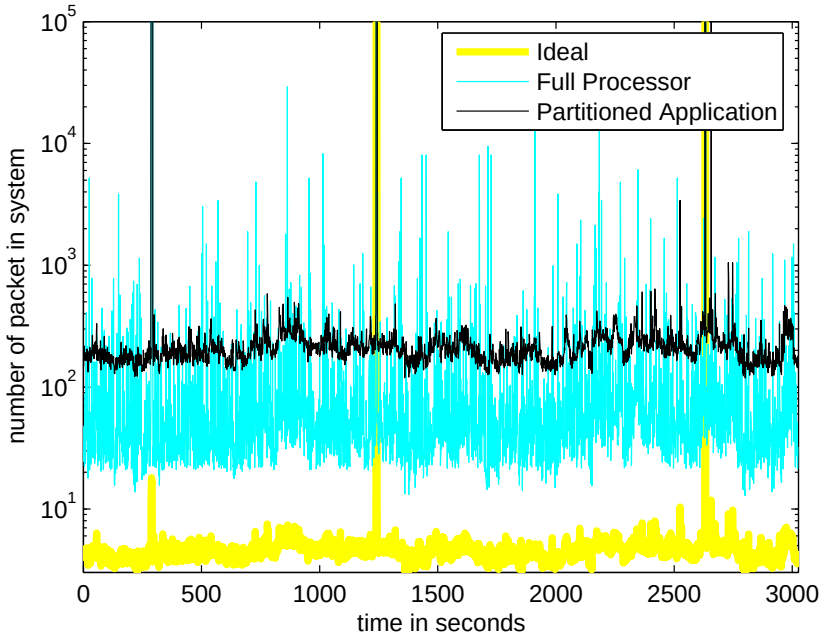


Figure 5.8. Packets in System over Time.

The effects of the runtime system can also be seen when exploring the number of packets in the system as shown in Figure 5.8. The higher utilization in Full Processor Allocation causes packets to queue up. Runtime adaptation is triggered when the queue length exceeds a threshold. The average number of packets in the system for Full Processor Allocation increases and decreases with this threshold value. Although Partitioned Application Allocation has a lower processor utilization than the Full Processor Allocation, the potentially unbalanced allocation of subtasks and the associated synchronization overhead increases the number of packets that are present in the system. However, the partitioning of applications into subtasks allows for a finer granularity of processor resource allocation, which in turn leads to less variation in the number of packets in the system. In Figure 5.8, the standard deviation of number of packets in the system for Full Processor Allocation is 1048.6 compared to 85.4 for Partitioned Application Allocation and 0.9 for Ideal Allocation. Less variation in

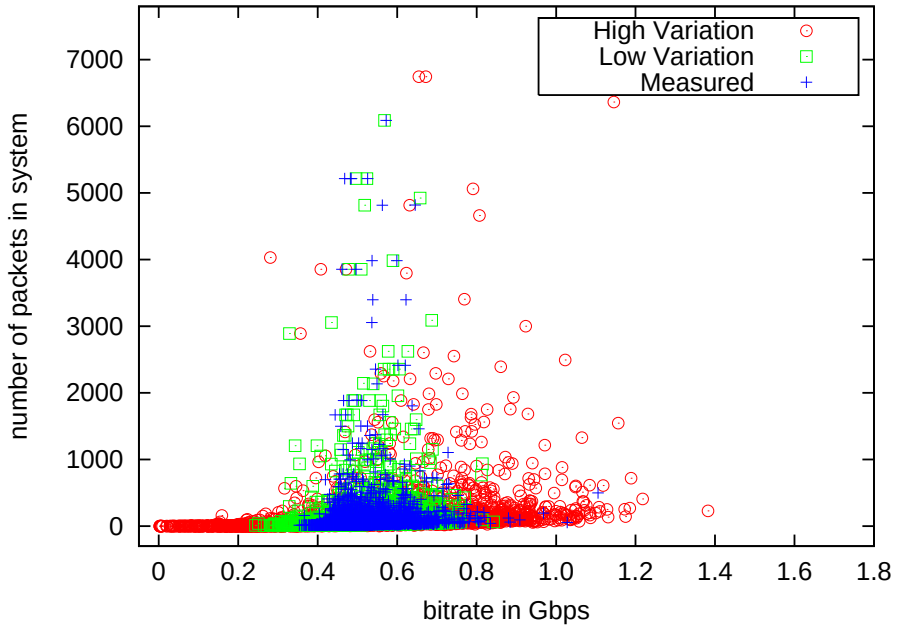


Figure 5.9. Packets in System under Different Traffic Scenarios.

the number of packets in the system is generally preferable as it leads to less variation in the processing delay and thus less jitter in the network.

5.6.2.3 Impact of Different Traffic

The characteristics of the traffic that is processed by a system has a big impact on the overall queuing behavior. Figure 5.9 shows a scatter plot of bit rates at 1-second interval of traffic and the observed number of packets in the system for Full Processor Allocation. Different colors indicate data points from different traffic scenarios. The High Variation scenario generates traffic with a wide range of data rates. The adaptation cannot happen infinitely fast, and thus the system may have to queue packets. With higher variation, more packets need to be queued on average than for the Low Variation scenario. The measured scenario is also shown as reference.

5.6.2.4 Performance for Different Data Rates

Finally, we explore the behavior of all three runtime systems for increasing data rates. Figure 5.10 shows the average number of packets in the system for different

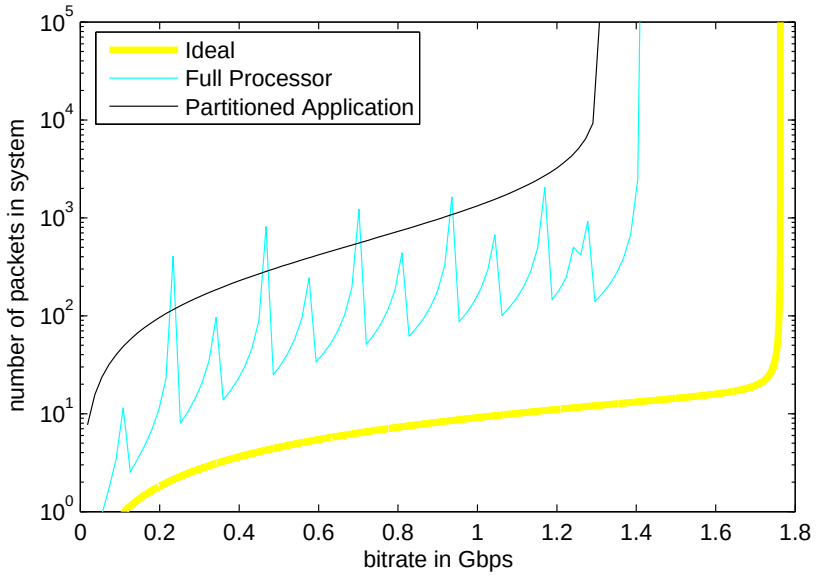


Figure 5.10. Packets in System over Different Data Rates.

data rates. The Ideal system can process packets with negligible delay up to the maximum possible data rate that can be sustained for the given system configuration and workload.

The Full Processor Allocation system shows longer queues for some data rates. This occurs when a given allocation is just barely sufficient to process the offered load. When the load increases a bit, an additional processor is allocated and the number of packets in the system drops to nearly zero.

The Partitioned Applications system shows an increasing number of packets even for low data rates. It does not show the effects of reallocation as the Full Processor Allocation system since all processors are in use at all times. However, due to the partitioning of applications, finding a good mapping is difficult. As a result, the system is fully utilized at a lower data rate than the other two scenarios.

The point at which the queue length increases to infinity is important as it marks the maximum data rate that a particular system can sustain. For the scenario shown in Figure 5.10, Full Processor Allocation achieves 1.41Gbps / 1.77Gbps = 79.6%

and Partitioned Application Allocation achieves 1.33Gbps / 1.77Gbps = 75.1%. Of course, these numbers change for different workloads and system configurations.

5.6.3 Simulation Results

To validate the analytical models, we present performance results from simulations for each type of runtime system. We also explore the difference and tradeoffs of these two evaluation methods.

5.6.3.1 OPNET Simulation Setup

We use the OPNET simulator to model our queuing networks with three components: (1) Sources that generate packets of constant size (535 bytes – approximately the average packet size in the Internet) using the OPNET *simple_source* process model. The packet interarrival time can be chosen to be exponentially distributed (for memoryless source model) or taken from a trace file (to simulate realistic network traffic). (2) Sinks that absorb packets and collect statistics using the OPNET *sink* model. (3) A Queueing Network that implements the queuing models shown in Figure 5.4 based on the OPNET acb_fifo_ms processor model (each processor is assumed to process 100MIPS). The service time of each processor can be set to be constant, exponentially distributed (for memoryless service model), or taken from a random variable generation process specified through a probability mass function (for service model based on real processing time measurements). The default queue lengths are simulated to be infinite (within the limits of OPNET).

When simulating each runtime system, we can choose from several combinations of arrival and service distributions:

- M/M/m Simulation: Uses exponentially distributed packet interarrival times and processing times. This configuration matches closely to queuing models with memoryless arrival and service distributions.

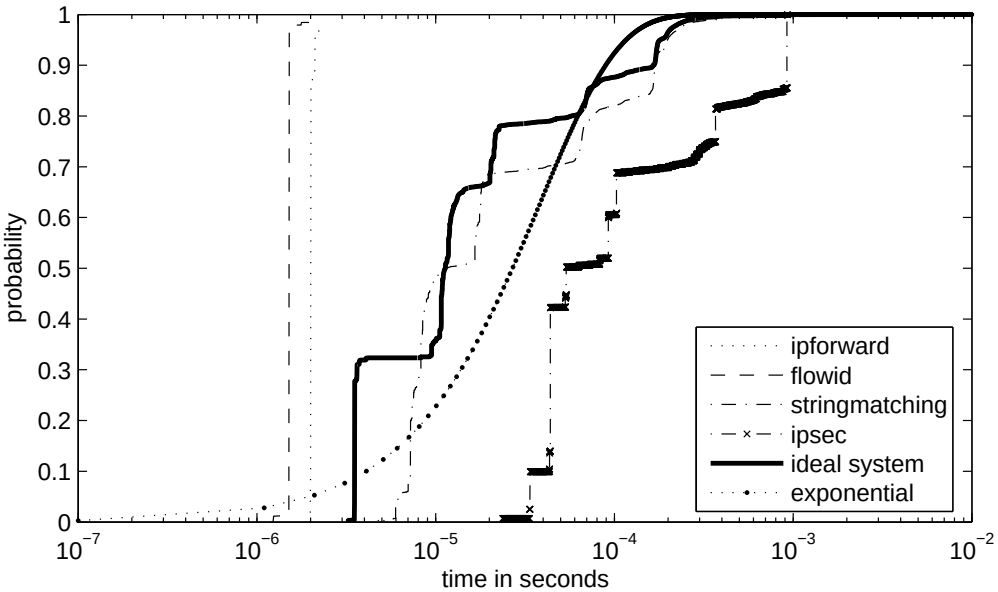
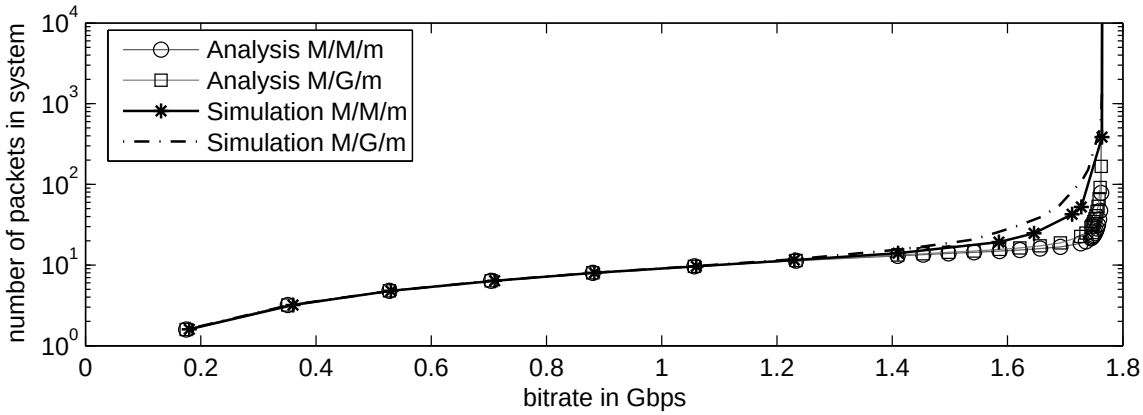


Figure 5.11. Service Time CDF of All Applications, the Ideal System, and an Exponentially Distributed Random Variable.

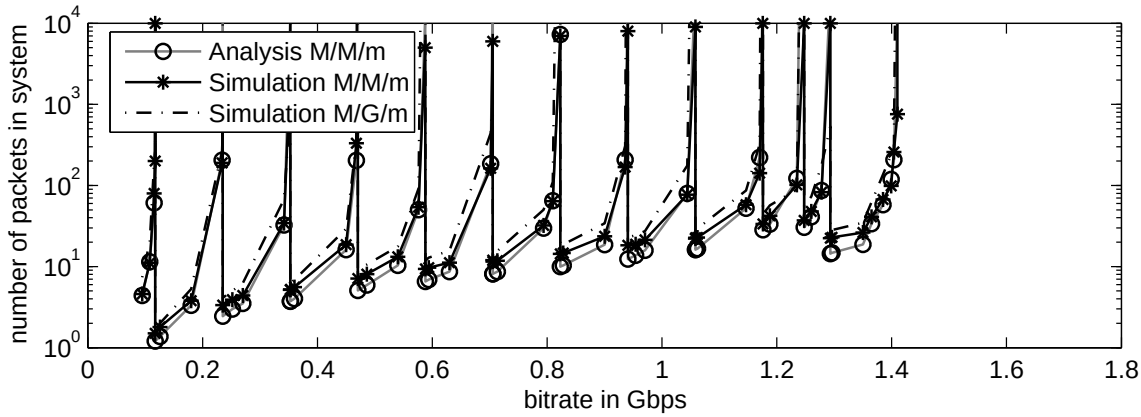
- M/G/m Simulation: Uses exponentially distributed packet interarrival times and realistic processing times. The distribution of packet processing time is obtained by analyzing packets processing traces using PacketBench [85]. The cumulative distribution function for all applications of our workload (see Figure 5.3(e)) is shown in Figure 5.11.

- G/G/m Simulation: Uses realistic distributions of packet interarrival times and packet processing times. The interarrival times are obtained from a network traffic trace. The packet processing times are obtained as described for the M/G/m simulation above.

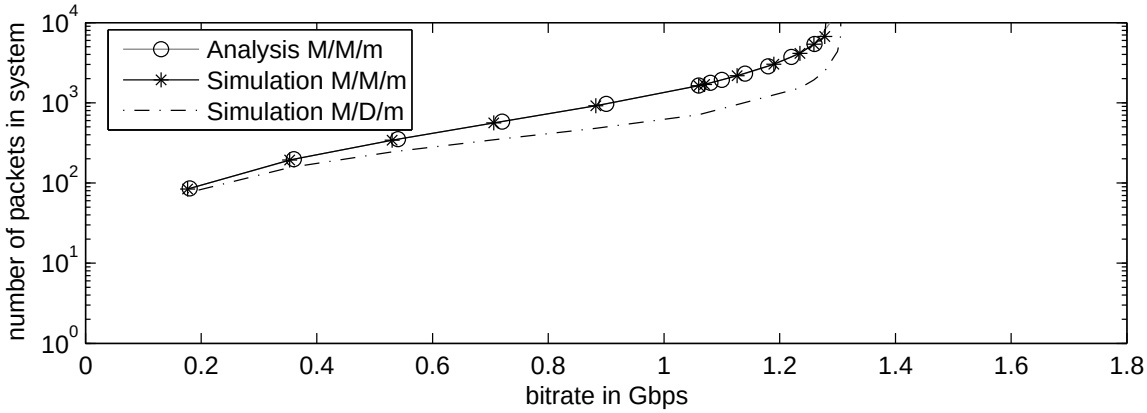
For each runtime system, we simulate the configurations that are feasible. The M/M/m simulation matches most closely to our analytical evaluation while the G/G/m simulation reflects a more realistic system configuration.



(a) Ideal System



(b) Full Processor Allocation System



(c) Partitioned Application Allocation System

Figure 5.12. Packets in System over Different Data Rates.

5.6.3.2 System Performance for Different Data Rates

In Figure 5.12, we show the simulation results for each of the three runtime systems. Each subfigure shows the analytical results (some of which are shown in Figure 5.10) and simulation results for one runtime system.

For the Ideal system (Figure 5.12(a)), we can compare the M/M/m and M/G/m models. As expected, the analytical results on number of packets in the system for both models are practically identical to those obtained from simulation except for very high data rates (and therefore high processor utilization). A difference of up to a factor of 2 can be observed when comparing the results from M/G/m and M/M/m for high bit rates in both analysis and simulation. This difference is due to the general distribution considering the coefficient of variation in addition to the service time mean. This leads to a higher number of packets in the system as the service time distribution shown in Figure 5.11 has a higher variance than the exponential distribution used in the M/M/m model.

A similar trend can be observed for the Full Processor Allocation system in Figure 5.12(b). Here, analysis can only provide results for M/M/m. The simulation results for M/M/m and M/G/m track the analytical results closely. Again, M/G/m provides higher (and more accurate) results for the number of packets in the system because of the underlying service time distribution.

The Partitioned Application Allocation system results are shown in Figure 5.12(c). The simulation results are based on M/M/m (as are the analytical results) and on M/D/m. The M/D/m model is a special case of the M/G/m model with deterministic processing times. This is applicable in this case since in a partitioned application, instruction basic blocks are always executed with constant processing time. This determinism is reflected in the lower number of packets in the system.

We observe that for all three systems, analysis and simulation yield very similar results. The only considerable difference occur for high bit rates and for high processor

utilization in the Full Processor Allocation scenario. Nevertheless, analysis can be used to get a good estimate of number of packets in the system.

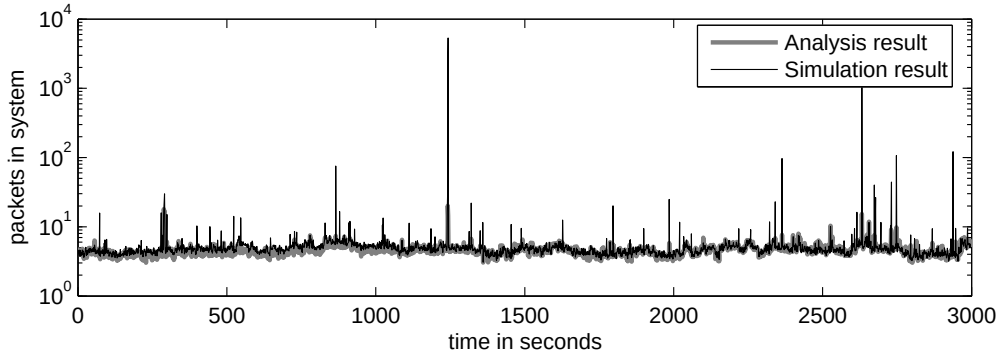
5.6.3.3 System Performance Over Time

The above simulation results do not consider a G/G/m model with realistic network traffic. In this section, we explore simulation results that are based on a real traffic trace (scaled to the appropriate bitrate) and consider the "memory effect" between reallocations of processing resources (i.e., packets that are queued but not yet processed remain in the system). For the Full Processor Allocation system, we set the high/low threshold that triggers reallocation to 10/0 (Simulation 1) and 20/15 for tasks with high processing requirements and 5/3 for tasks with lower processing requirements (Simulation 2). The queue length is limited in this simulation.

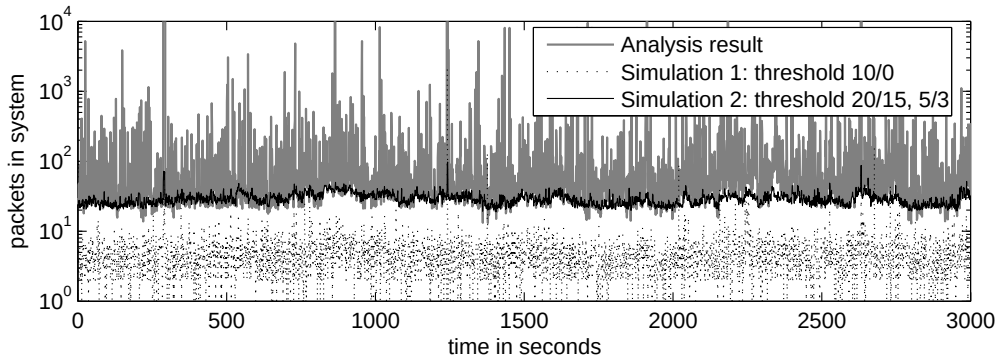
The simulation results of packets in system over time using our sample traffic are shown in Figure 5.13. The simulation results closely track the behavior of the analytical results.

For the Ideal Scenario, the results are practically identical and show that the general arrival time distribution does not yield different results. For Full Processor Allocation, simulation shows a lower number of packets in the system than analysis due to the settings of high/low queue length thresholds, which trigger the allocation of more or less processing resources. For the Partitioned Application Allocation, the simulation provides a slightly lower number of packets in the system due to the use of a deterministic service time distribution in the simulation.

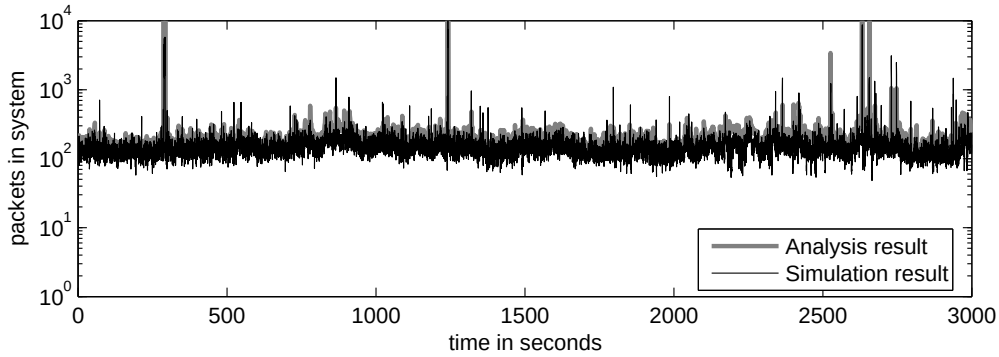
Overall, the simulation results show that our analytical models match closely the operational behavior of network processor runtime systems, even when considering realistic packet arrival and processing time distributions. This indicates that it is possible to use analytical performance analysis when exploring different runtime system design, which is much easier to do than implementing complex simulation setups. It



(a) Ideal System



(b) Full Processor Allocation System



(c) Partitioned Application Allocation System

Figure 5.13. System Performance over Time.

is also possible to use analytical performance modeling within a runtime system to evaluate the current state of the system and make short-term performance predictions.

5.6.4 Implications for Runtime System Design

The above results yield several observations and conclusions with regards to runtime system design for network processors:

- Static allocations are not effective under varying traffic conditions. Figure 5.5 shows that only 20–60% of utilization would be achieved in such a scenario.

- Runtime systems that allocate only some of the processors to conserve power cause longer packet delays due to longer packet queues (see Figure 5.8). It is also possible to encounter longer delays when the system is not fully utilized (see spikes in Figure 5.8). It is therefore important to design the trigger mechanism for allocating additional processors to be sensitive enough to avoid that these effects become problems.

- High variation in traffic causes longer packet delays (see Figure 5.9). This is inherent to all runtime system that allocate some tasks to some subset of processors. Only the Ideal system, where all processors can process all applications, can avoid this problem. It may be desirable to investigate if such an Ideal system can be implemented as it performs well and would be trivial to manage during runtime.

- The analytical result can estimate the trend of the system performance nearly as accurately as simulation. This enables runtime system designers to utilize analytical performance modeling for design-space exploration and runtime performance estimation.

5.7 Related Work

Several network processors are commercially available and have been used in research and development: Intel IXP [58], Hifn PowerNP [4], EZchip NP-1 [38], AMCC np7510 [5]. These systems have been used for a broad range of networking applications, ranging from general programmable router component [95] to overlay networks [45] and content-based switching [118].

Programming of network processors is challenging due to the complex interactions between the multiple processor cores, memory, and other shared components. Different programming abstractions have been proposed by Goglin et al. [47] and Shah et al. [89] for static workload scenarios. Dynamic scenarios have been considered by Memik et al. [74] and Teja [97]. While both of these approaches provide a thin network processing operating system to simplify programming, neither provides the ability to quickly adapt multiple applications during runtime. Other scheduling algorithms have been proposed by Franklin and Datar [41]. Plishker et al. [83] have proposed mapping based on their domain specific language for network processors. Kokku et al. [68] have proposed an algorithm for adapting allocations of entire applications to processor cores to reduce power consumption. We use this algorithm as one of the evaluation scenarios in Section 5.6. Wolf et al. [114] have proposed runtime support that also considers the partitioning of applications across multiple processor cores. This is another approach that is evaluated in the Section 5.6.

In order to evaluate the performance of a given workload scenario on a network processor system, a number of different models have been proposed. These have typically been applied to design space evaluation by Thiele et al. [100], Crowley and Baer [29], Gries et al. [48], and Wolf and Franklin [113]. We use a less detailed approach to evaluating system performance and base our results on simulation results as proposed by Ramaswamy and Wolf [85].

Lu and Wang [35] have proposed queuing networks for modeling network processor systems. Their work addresses the performance evaluation of a single static network processing application. While their model is more detailed and considers memory accesses, it does not address the issue of dynamic workload scenarios and runtime adaptation, which is the main contribution of our work.

5.8 Summary

In summary, this chapter presents a methodology for evaluating runtime task allocation solutions for multi-core network processor systems. The motivation for this work lies in the need for a standardized evaluation process for runtime task mapping solutions on network processor systems that are currently being developed as the essential basis for network programmability.

The presented methodology considers the workload of the system in terms of processing characteristics and traffic characteristics. To exercise evaluated system with a wide range of traffic patterns, we present a mechanism for generating synthetic traffic traces for different scenarios. The system model uses a queuing network abstraction to represent different runtime systems and allows for an analytical performance evaluation.

Both analysis and simulation are used for system performance evaluation. First, the proposed system is used to evaluate the performance of two existing runtime systems that have been published in literature. The results are compared to a static configuration and an ideal system. The results show that runtime adaptation is necessary but causes overhead over an idealized system. Our methodology allows us to quantify this overhead for different workload scenarios. Then, the accuracy of the proposed evaluation methodology is validated through simulation. The results show that in most cases this analytical-model-based system can provide performance estimations that are comparable to those obtained from a widely-used discrete-event

simulator. However, our analytical-model-based evaluation methodology takes much less time than evaluations based on discrete-event simulators. Therefore, our analytical evaluation provides an easy way to estimate system performance trend and compare the performance of different runtime systems.

We believe that our work is an important step towards developing a benchmark for runtime resource management on network processor systems, and thus contributes to the implementation of programmable routers and the efficient management of datapath processing capability in future networks.

CHAPTER 6

SUMMARY AND FUTURE WORK

6.1 Summary

This work presents an extensive discussion of implementing issues associated with a service-centric network architecture designed for the next-generation Internet.

Modern networks need to be able to adapt to evolving security threats, provide connectivity to a variety of different types of end-systems, and support emerging networking paradigms ranging from peer-to-peer networks to content networks. This need for flexibility is not only realized through the use of novel protocols, but also through a fundamental shift in the architecture: instead of limiting networks to provide simple store-and-forward packet processing, more complex "network services" are moved from end-system to the network. Such services provide a variety of processing capabilities that go well beyond packet forwarding (e.g., content inspection, payload transcoding, etc.).

One important question emerging with this evolvement trend is: how to manage such services in a scalable fashion? As the answer to this question, a service-centric network architecture is proposed. Its idea is to use router-based programmability to provide packet processing services inside the network. Communications are decomposed into these service blocks. By providing different compositions of services along the data path, the network can customize connections to satisfy various communication requirements from different applications. Rather than viewing routers as simple electronic devices connecting end-systems, this new architecture design regards the routers as an integral part of a distributed computing system. It helps to overcome the

flexibility constraints of the current store-and-forward Internet design and to expand the capabilities of the network to meet its next-generation challenges.

Three major technical challenges for implementing such service-centric network architecture design networks are studied in this work:

1. **Service Routing**: Routing in the service-centric network becomes a complex problem that goes beyond simply finding the shortest path between two end-systems. It is called service routing in this paper. Service routing needs to consider both service availability and path cost (the sum of link delay and processing delay). When more services are required by a connection, the problem becomes increasingly difficult. When resource constraints are considered, service routing becomes an NP-complete problem. A global routing algorithm and a novel decentralized routing algorithm are presented to solve the service routing problem. The global algorithm provides optimal solution for the routing problem without resource constraints and near-optimal solution for the problem with resource constraints. However the global algorithm needs a complete view of the network. The decentralized algorithm is scalable and efficient but at the cost of quality (i.e., path cost). Based on these algorithms, a routing protocol is designed to implement the proposed algorithms in large-scale service-centric networks. The experimental results from a Emulab-based prototype validate our design and shows the tradeoffs between different algorithms. The results also show that the proposed routing protocol is efficient in connection setup time and control massage overhead.

2. **Service Composition**: In service-centric networks, the basic functional building blocks, network services, are deployed and enabled on service nodes distributed inside the network. Different service nodes provide different network services, depending on the deployment configuration. Given a communication request, the network needs to dynamically compose network services to satisfy

the communication requirements. And in most of the cases, an optimal service composition that satisfies the communication requirements with the minimum cost is desired. A decision making framework is presented, which helps to deduce the service composition problem to a planning problem and automates the composition of services according to specific communication requirements of the connections. This framework is further extended to combine service composition and service routing into one problem. A synthetic benchmark is also developed to help comprehensively evaluate the performance of our design. The results show that combing service routing and service composition problems yields better solutions than solving them separately.

3. **Data-path Resource Management**: Programmable routers provide an ideal platform for implementing the service-centric network. Current programmable routers are based on the Network Processor (NP) technology, which utilizes multi-core system-on-a-chip architecture to achieve a good balance between programmability and performance. Several runtime task mapping approaches have been proposed for NP systems. Unfortunately, it is very difficult to compare the performance of these solutions since they make a variety of assumptions on the underlying system structure and they use different traffic and benchmark applications for performance evaluation. An evaluation methodology is presented to solve this problem. Based on queuing network abstraction, the methodology can systematically model the NP systems and analyze the performance of different runtime task mapping algorithms under different network traffic scenarios. The results from the proposed evaluation methodology is validated and compared with those from a widely-used discrete event simulator. The results show that this analytical-model-based system can provide performance estimation at a comparable level while takes much less time than evaluations based on discrete-event simulators.

6.2 Future Work

There are two main directions in which future work can be pursued. One is to solve the other issues faced from implementing the service-centric network architecture. The other is to integrate our design with other emerging technologies/works in the domain of next-generation Internet.

Implementing such a service-centric network architecture is important and challenging. Except the problems (e.g., service routing, service composition, and data-path resource management) that have been studied in our work, there are many other interesting implementation issues. Multi-policy issue is an example. In the service-centric network, multi-parties (e.g., the sender, the receiver, network administrators, and service providers) may enforce different policies (i.e., service requirements) to the same connection. This makes the service composition problem more complicated and opens a number of questions. How to integrate these enforced policies into the original service composition? How to check the compatibility of the policies? How to handle the conflicting policies? Another interesting problem is the programming interface. That is, to build a unified interface similar to BSD TCP/IP sockets to ease the communication between different systems inside a service-centric network. In a network where various complex packet processing services are available in the data path, how could the servers be notified about the futures of these services (e.g., the input and output description of a service)? How could a service nodes report to the server about its processing capability (e.g., services availability and processing time)? How could an end-system application specifies their communication requirement in the connection request? Finding the answers to these questions will help improve the design of the service-centric network and bring it to maturity.

Another equally important topic is the applications of our service-centric network architecture design in the context of next-generation Internet. Section 2.3 presented one possible application example. The service-centric network architecture could be

encapsulated into a communication abstraction and work as a comprehensive interfacing framework to ease the development of novel networks and applications. It would be interesting to implement such an interfacing framework on one of the available programmable/virtualzied platforms, such as Emulab, PlanetLab, etc. Measurement could be another interesting application area for the service-centric network architecture. The current Internet architecture design limits processing power on the end-systems, which makes network measurement a tough topic. The Internet performance like a "black box," all the measurement has to be performed outside it. However, in the service-centric network, measurement functionality could be embedded inside the network as a special service. How to build a unified measurement framework to manage all the measurement functionalities provided by the network? How could the distributed measurement nodes cooperate with each other to satisfy a specific measurement request? How the framework manages the measurement resources to maximize the overall performance? Similarly, how the service-centric network architecture could help in improving network security is another potential topic.

In summary, intergrading with the emerging technologies such as network programmability and network virtualization, the service-centric network architecture design opens a broad range of potential research topics in the clean-slate design of next-generation Internet.

BIBLIOGRAPHY

[1] *PlanetLab*. `http://www.planet-lab.org/`.

[2] *VINI: a Virtual Network Infrastructure*. `http://www.vini-veritas.net/`.

[3] Alan, Sheltzer, Robert, Hinden, and Mick, Brescia. Connecting differeent types of networks with gateways. In *Data Communications* (Aug. 1982).

[4] Allen, James, Bass, Brian, Basso, Claude, Boivie, Rick, Calvignac, Jean, Davis, Gordon, Frelechoux, Laurent, Heddes, Marco, Herkersdorf, Andreas, Kind, Andreas, Logan, Joe, Peyravian, Mohammad, Rinaldi, Mark, Sabhikhi, Ravi, Siegel, Michael, and Waldvogel, Marcel. IBM PowerNP network processor: Hardware, software, and applications. *IBM Journal of Research and Development 47*, 2/3 (2003), 177–194.

[5] AMCC. *np7510 10 Gbps Network Processor*, 2003. `http://www.amcc.com`.

[6] Anderson, Thomas, Peterson, Larry, Shenker, Scott, and Turner, Jonathan. Overcoming the Internet impasse through virtualization. *Computer 38*, 4 (Apr. 2005), 34–41.

[7] Apostolopoulos, George, Aubespin, David, Peris, Vinod, Pradhan, Prashant, and Saha, Debanjan. Design, implementation and performance of a content-based switch. In *Proc. of IEEE INFOCOM 2000* (Tel Aviv, Israel, Mar. 2000), pp. 1117–1126.

[8] B., Cain, S., Deering, I., Kouvelas, B., Fenner, and A., Thyagarajan. *IANA Guidelines for IPv4 Multicast Address Assignments*. Network Working Group, Oct. 2002. RFC 3376.

[9] Barbir, Abbie, Reinaldo, Penno, Chen, Robin, Hofmann, Markus, and Hilarie, Orman. An architecture for open pluggable edge services (OPES). RFC 3835, Network Working Group, Aug. 2004.

[10] Barham, Paul, Dragovic, Boris, Fraser, Keir, Hand, Steven, Harris, Tim, Ho, Alex, Neugebauer, Rolf, Pratt, Lan, and Warfield, Andrew. Xen and the art of virtualization. In *Proc. of the nineteenth ACM Symposium on Operating Systems Principles (SOSP)* (Bolton Landing, NY, Oct. 2003), pp. 164–177.

[11] Baskett, Forest, Chandy, K. Mani, Muntz, Richard R., and Palacios, Fernando G. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM 22*, 2 (Apr. 1975), 248–260.

[12] Bavier, Andy, Bowman, Mic, Chun, Brent, Culler, David, Karlin, Scott, Peterson, Larry, Roscoe, Timoty, Spalink, Tammo, and Wawrzoniak, Mike. Operating system support for planetary-scale network services. In *Proceedings of the 1st Symposium on Network System Design and Implementation (NSDI '04)* (San Francisco, CA, Mar. 2004).

[13] Bavier, Andy, Feamster, Nick, Huang, Mark, Peterson, Larry, and Rexford, Jennifer. In VINI veritas: realistic and controlled network experimentation. In *SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Pisa, Italy, Aug. 2006), pp. 3–14.

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

[15] Bhatti, Nina T., and Schlichting, Richard D. A system for constructing configurable high-level protocols. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication* (Cambridge, MA, Aug. 1995), pp. 138–150.

[16] Blumenthal, Marjory S., and Clark, David D. Rethinking the design of the Internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology 1*, 1 (Aug. 2001), 70–109.

[17] B.M., Leiner, D.L., Nelson, and F.A., Tobagi. Issues in packet radio network design. *Proceedings of the IEEE 75* (Jan. 1987), 6–20.

[18] Bolch, Gunter, Greiner, Stefan, de Meer, Hermann, and Trivedi, Kishor S. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley & Sons, Inc., New York, NY, Aug. 1998.

[19] Braden, Robert, Faber, Ted, and Handley, Mark. From protocol stack to protocol heap: role-based architecture. *SIGCOMM Computer Communication Review 33*, 1 (Jan. 2003), 17–22.

[20] Brutlag, Jake D. Aberrant behavior detection in time series for network monitoring. In *Proc. of the 14th Systems Administration Conference* (New Orleans, LA, Dec. 2000), pp. 139–146.

[21] Caesar, Matthew, and Rexford, Jennifer. Building bug-tolerant routers with virtualization. In *Proc. of the ACM workshop on Programmable Routers for Extensible Services of TOmorrow (PRESTO)* (Seattle, WA, Aug. 2008), pp. 51 – 56.

[22] Cerf, Vinton G., and Kahn, Robert E. A protocol for packet network intercommunication. *IEEE Transactions on Communications COM-22*, 5 (May 1974), 637–648.

[23] Chandra, Prashant R., Yavatkar, Raj, Bock, Tony, Cabot, Mason, and Mathew, Philip. Benchmarking network processors. In *Proc. of First Network Processor Workshop (NP-1) in conjunction with Eighth IEEE International Symposium on High Performance Computer Architecture (HPCA-8)* (Cambridge, MA, Feb. 2002), pp. 10–17.

[24] Charlie, Wiseman, Jonathan, Turner, John, DeHart, Jyoti, Parwatikar, Ken, Wong, and David, Zar. Using the netfpga in the open network laboratory. In *Proceedings of the 1st NetFPGA Developers Workshop* (2000).

[25] Choi, Sumi Y., and Turner, Jonathan S. Configuring sessions in programmable networks with capacity constraints. In *Proc. of IEEE International Conference on Communications (ICC)* (Anchorage, AK, May 2003).

[26] Choi, Sumi Y., Turner, Jonathan S., and Wolf, Tilman. Configuring sessions in programmable networks. *Computer Networks 41*, 2 (Feb. 2003), 269–284.

[27] Cisco Systems, Inc. *Cisco Unified Communications and Service-Oriented Network Architecture*, July 2007.

[28] Clark, David D. The design philosophy of the DARPA Internet protocols. In *Proc. of ACM SIGCOMM 88* (Stanford, CA, Aug. 1988), pp. 106–114.

[29] Crowley, Patrick, and Baer, Jean-Loup. A modelling framework for network processor systems. In *Proc. of First Network Processor Workshop (NP-1) in conjunction with Eighth IEEE International Symposium on High Performance Computer Architecture (HPCA-8)* (Cambridge, MA, Feb. 2002), pp. 86–96.

[30] D., Waitzman, C., Partridge, and S., Deering. *Distance Vector Multicast Routing Protocol*. Network Working Group, Nov. 1988. RFC1075.

[31] David, Erickson, Glen, Gibb, Brandon, Heller, David, Underhill, and Nick, McKeown. A demonstration of virtual machine mobility in an openflow network. In *in Proc. of ACM SIGCOMM (demo)* (Seattle, WA, aug 2008).

[32] David D., Clark. Window and acknowledgement strategy in tcp nic-rfc-813. In *DDN Protocol Handbook* (July 1982), vol. 3.

[33] Decasper, Dan, Dittia, Zubin, Parulkar, Guru, and Plattner, Bernhard. Router Plugins - a modular and extensible software framework for modern high performance integrated services routers. In *Proc. of ACM SIGCOMM 98* (Vancouver, BC, Sept. 1998), pp. 229–240.

[34] Decasper, Dan, Parulkar, Guru, Choi, Sumi, DeHart, John, Wolf, Tilman, and Plattner, Bernhard. A scalable high performance active network node. *IEEE Network 13*, 1 (Jan. 1999), 8–19.

[35] 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.

[36] Eatherton, Will. The push of network processing to the top of the pyramid. In *Keynote Presentation at ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (Princeton, NJ, Oct. 2005).

[37] Egevang, Kjeld Borch, and Francis, Paul. The IP network address translator (NAT). RFC 1631, Network Working Group, May 1994.

[38] EZchip Technologies Ltd. *NP-1 10-Gigabit 7-Layer Network Processor*. Yokneam, Israel, 2002. `http://www.ezchip.com/html/pr_np-1.html`.

[39] Floyd, Sally, and Jacobson, Van. Random early detection (RED) gateways for congestion avoidance. *IEEE/ACM Transactions on Networking 1*, 4 (Aug. 1993), 397–413.

[40] Foster, Ian, and Kesselman, Carl, Eds. *The Grid – Blueprint for a New Computing Infrastructure*, 2nd ed. Morgan Kaufmann, 2004.

[41] Franklin, Mark A., and Datar, Seema. Pipeline task scheduling on network processors. In *Proc. of Third Network Processor Workshop (NP-3) in conjunction with Tenth IEEE International Symposium on High Performance Computer Architecture (HPCA-10)* (Madrid, Spain, Feb. 2004).

[42] Franklin, Mark A., and Wolf, Tilman. A network processor performance and design model with benchmark parameterization. In *Proc. of First Network Processor Workshop (NP-1) in conjunction with Eighth IEEE International Symposium on High Performance Computer Architecture (HPCA-8)* (Cambridge, MA, Feb. 2002), pp. 63–74.

[43] Franklin, Mark A., and Wolf, Tilman. Power considerations in network processor design. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth IEEE International Symposium on High Performance Computer Architecture (HPCA-9)* (Anaheim, CA, Feb. 2003), pp. 10–22.

[44] Ganapathy, Sivakumar, and Wolf, Tilman. Design of a network service architecture. In *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (Honolulu, HI, Aug. 2007), pp. 754–759.

[45] Gavrilovska, Ada, Schwan, Karsten, Nordstrom, Ola, and Seifu, Hailemelekot. Network processors as building blocks in overlay networks. In *Proc. of Hot Interconnects* (Stanford, CA, Aug. 2003), ACM, pp. 83–88.

[46] Gerevini, Alfonso, and Serina, Ivan. Lpg: a planner based on local search for planning graphs. In *Proc. of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS)* (Toulouse, France, 2002).

[47] Goglin, Stephen D., Hooper, Donald, Kumar, Alok, and Yavatkar, Raj. Advanced software framework, tools, and languages for the IXP family. *Intel Technology Journal 7*, 4 (Nov. 2003), 64–76.

[48] Gries, Matthias, Kulkarni, Chidamber, Sauer, Christian, and Keutzer, Kurt. Exploring trade-offs in performance and programmability of processing element topologies for network processors. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth IEEE International Symposium on High Performance Computer Architecture (HPCA-9)* (Anaheim, CA, Feb. 2003), pp. 75–87.

[49] Guttman, Erik. Service location protocol: Automatic discovery of IP network services. *IEEE Internet Computing 3*, 4 (July 1999), 71–80.

[50] He, Jiayue, Zhang-Shen, Rui, Li, Ying, Lee, Cheng-Yen, Rexford, Jennifer, and Chiang, Mung. Davinci: Dynamically adaptive virtual networks for a customized internet. In *Proc. of the 4th ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)* (Madrid, Spain, Dec. 2008).

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

[52] Huang, Xin, Ganapathy, Sivakumar, and Wolf, Tilman. A distributed algorithm for network service placement. In *Proc. of Seventeenth IEEE International Conference on Computer Communications and Networks (ICCCN)* (St. Thomas, USVI, Aug. 2008).

[53] Huang, Xin, Ganapathy, Sivakumar, and Wolf, Tilman. A scalable distributed routing protocol for networks with data-path services. In *Proc. of 16th IEEE International Conference on Network Protocols (ICNP)* (Orlando, FL, Oct. 2008).

[54] Huang, Xin, Ganapathy, Sivakumar, and Wolf, Tilman. Evaluating algorithms for composable service placement in computer networks. In *Proc. of the IEEE International Conference on Communications (ICC)* (Dresden, Germany, June 2009).

[55] Huang, Xin, and Wolf, Tilman. A methodology for evaluating runtime support in network processors. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (San Jose, CA, Dec. 2006), pp. 113–122.

[56] Huang, Xin, and Wolf, Tilman. Evaluating dynamic task mapping in network processor runtime systems. *IEEE Transactions on Parallel and Distributed Systems 19*, 8 (Aug. 2008), 1086–1098.

[57] Hutchinson, Norman C., and Peterson, Larry L. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering 17*, 1 (Jan. 1991), 64–76.

[58] Intel Corporation. *Intel Second Generation Network Processor*, 2005. `http://www.intel.com/design/network/products/npfamily/`.

[59] J., Saltzer, D., Reed, and D.D., Clark. End-to-end arguments in system design.

[60] Jad, Naous, Glen, Gibb, Sara, Bolouki, and McKeown, Nick. Netfpga: Reusable router architecture for experimental research. In *in Proc. of the 1nd ACM SIGCOMM workshop on Programmable routers for extensible services of tomorrow* (Seattle, WA, aug 2008).

[61] John M., McQuillan, and David C., Walden. The arpa network design decisions. In *Computer Networks* (Aug. 1977), vol. 1.

[62] Jonathan, Postel. User datagram protocol. RFC 768, Information Sciences Institute, Aug. 1980.

[63] Jonathan B., Postel. Internetwork protocol approach. In *IEEE Transactions on Communications* (Apr. 1980), vol. 28, pp. 605–611.

[64] Jonathan B., Postel, Carl A., Sunshine, and Danny, Cohen. The arpa internet protocol. In *Computer Networks* (July 1981), vol. 5, pp. 261–271.

[65] Karp, Richard M. An introduction to randomized algorithms. *Discrete Applied Mathematics 34*, 1-3 (Nov. 1991), 165–201.

[66] Keller, Eric, and Green, Evan. Virtualizing the data plane through source code merging. In *Proc. of the ACM workshop on Programmable Routers for Extensible Services of TOmorrow (PRESTO)* (Seattle, WA, Aug. 2008), pp. 9–14.

[67] Khan, Esam, El-Kharashi, M. Watheq, Ehtesham Rafiq, A.N.M., Gebali, Fayez, and Abd-El-Barr, Mostafa. Network processors for communication security: a review. In *Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing 2003. PACRIM. 2003 IEEE Pacific Rim Conference onCommunications, Computers and signal Processing (PacRim 2003)* (Waikiki, HI, Feb. 2003).

[68] Kokku, Ravi, Riché, Taylor, Kunze, Aaron, Mudigonda, Jayaram, Jason, Jamie, and Vin, Harrick. A case for run-time adaptation in packet processing systems. In *Proc. of the 2nd Workshop on Hot Topics in Networks (HOTNETS-II)* (Cambridge, MA, Nov. 2003).

[69] Lakhina, Anukool, Papagiannaki, Konstantina, Crovella, Mark, Diot, Christophe, Kolaczyk, Eric D., and Taft, Nina. Structural analysis of network traffic flows. *SIGMETRICS Performance Evaluation Review 32*, 1 (June 2004), 61–72.

[70] Liu, Rong-Tai, Huang, Nen-Fu, Chen, Chih-Hao, and Kao, Chia-Nan. A fast string-matching algorithm for network processor-based intrusion detection system. *Transactions on Embedded Computing Systems 3*, 3 (Aug. 2004), 614–633.

[71] Lockwood, John W., McKeown, Nick, Watson, Greg, Gibb, Glen, Hartke, Paul, Naous, Jad, Raghuraman, Ramanan, and Luo, Jianying. NetFPGA–an open platform for gigabit-rate network switching and routing. In *MSE '07: Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education* (San Diego, CA, June 2007), pp. 160–161.

[72] Lum, Wai Yip, and Lau, Francis C.M. A context-aware decision engine for content adaptation. In *Proc. of IEEE Pervasive Computing* (2002), vol. 1, pp. 41–49.

[73] Mcauley, Anthony J., and Francis, Paul. Fast routing table lookup using cams. In *IEEE INFOCOM* (1993), pp. 1382–1391.

[74] Memik, Gokhan, and Mangione-Smith, William H. NEPAL: A framework for efficiently structuring applications for network processors. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth IEEE International Symposium on High Performance Computer Architecture (HPCA-9)* (Anaheim, CA, Feb. 2003).

[75] Mike, Hibler, Robert, Ricci, Leigh, Stoller, Jonathon, Duerig, Shashi, Guruprasad, Tim, Stack, Kirk, Webb, and Jay, Lepreau. Large-scale virtualization in the emulab network testbed. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference* (Berkeley, CA, USA, 2008), USENIX Association, pp. 113–128.

[76] Mogul, Jeffrey C. Simple and flexible datagram access controls for UNIX-based gateways. In *USENIX Conference Proceedings* (Baltimore, MD, June 1989), pp. 203–221.

[77] Muhammad, Bilal, and Nick, Feamster. Building a fast, virtualized data plane with programmable hardware. In *in Proc. of the 1st ACM workshop on Virtualized Infrastructure Systems and Architectures* (Barcelona, Spain, aug 2009).

[78] National Science Foundation. *Global Environment for Network Innovation.* `http://www.geni.net/`.

[79] Neda, Beheshti, Sara, Bolouki, Yashar, Ganjali, and Nick, McKeown. Experimenting with programmable routers in real networks. In *in Proc. of ACM SIGCOMM (demo)* (Seattle, WA, aug 2008).

[80] Nick Feamster, Lixin Gao, and Rexford, Jennifer. How to lease the internet in your spare time. In *Editorial Zone of ACM SIGCOMM Computer Communications Review* (Jan. 2007), pp. 61 – 64.

[81] Pandey, Sonal, Somani, Arun, and Tyagi, Akhilesh. A reliable protocol for processing within IP-routed networks. In *Proc. of Eleventh International Conference on Computer Communications and Networks (ICCCN)* (Miami, FL, Oct. 2002), pp. 84–89.

[82] Peterson, Larry, Anderson, Tom, Culler, David, and Roscoe, Timothy. A blueprint for introducing disruptive technology into the Internet. In *Proc. of HotNets–I* (Princeton, NJ, Oct. 2002).

[83] Plishker, William, Ravindran, Kaushik, Shah, Niraj, and Keutzer, Kurt. Automated task allocation for network processors. In *Proc. of Network System Design Conference* (Oct. 2004), pp. 235–245.

[84] Qiang, Wu, and Tilman, Wolf. Design of a network service processing platform for data path customization. In *in Proc. of the 2nd ACM SIGCOMM workshop on Programmable routers for extensible services of tomorrow* (Barcelona, Spain, aug 2009).

[85] Ramaswamy, Ramaswamy, and Wolf, Tilman. PacketBench: A tool for workload characterization of network processing. In *Proc. of IEEE 6th Annual Workshop on Workload Characterization (WWC-6)* (Austin, TX, Oct. 2003), pp. 42–50.

[86] R.E., Kahn, S.A., Gronemeyer, and J., Burdifiel amd E. V., Hoversten. Advances in packet radio technology. In *Proceedings of the IEEE* (Nov. 1978), vol. 66.

[87] Rina, Panigrahy, and Samar, Sharma. Reducing tcam power consumption and increasing throughput. In *IEEE 10th Symposium on High Performance Interconnects HOT Interconnects (HotI'02)* (Standford, CA, 2002), pp. 107–114.

[88] 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).

[89] Shah, Niraj, Plishker, William, and Keutzer, Kurt. NP-Click: A programming model for the Intel IXP1200. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth IEEE International Symposium on High Performance Computer Architecture (HPCA-9)* (Anaheim, CA, Feb. 2003), pp. 100–111.

[90] Shanbhag, Shashank, Huang, Xin, Proddatoori, Santosh, and Wolf, Tilman. Automated service composition in next-generation networks. In *Proc. of The International Workshop on Next Generation Network Architecture (NGNA)* (Montreal, Canada, June 2009).

[91] Shanbhag, Shashank, and Wolf, Tilman. Implementation of end-to-end abstractions in a network service architecture. In *Fourth Conference on emerging Networking EXperiments and Technologies (CoNEXT)* (Madrid, Spain, Dec. 2008).

[92] Snoeren, Alex S., Partridge, Craig, Sanchez, Luis A., Jones, Christine E., Tchakountio, Fabrice, Kent, Stephen T., and Strayer, W. Timothy. Hash-based IP traceback. In *Proc. of ACM SIGCOMM 2001* (San Diego, CA, Aug. 2001), pp. 3–14.

[93] Soltesz, Stephen, Poetzl, Herbert, Fiuczynski, Marc E., Bavier, Andy, and Peterson, Larry. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proc. of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems* (Lisbon, Portugal, Mar. 2007), pp. 275 – 287.

[94] Sommers, Joel, and Barford, Paul. Self-configuring network traffic generation. In *Proc. of the 4th ACM SIGCOMM conference on Internet measurement (IMC)* (Taormina, Italy, Oct. 2004), pp. 68–81.

[95] Spalink, Tammo, Karlin, Scott, Peterson, Larry, and Gottlieb, Yitzchak. Building a robust software-based router using network processors. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)* (Banff, AB, Oct. 2001), pp. 216–229.

[96] Stephen, Deering, Deborah, Estrin, Dino, Farinacci, Van, Jacobson, Chinggung, Liu, and Liming, Wei. The pim architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking 4* (1996), 153–162.

[97] Teja Technologies. *TejaNP Datasheet*, 2003. `http://www.teja.com`.

[98] Tennenhouse, David L., Smith, Jonathan M., Sincoskie, W. David, Wetherall, David J., and Minden, Gary J. A survey of active network research. *IEEE Communications Magazine 35*, 1 (Jan. 1997), 80–86.

[99] Tennenhouse, David L., and Wetherall, David J. Towards an active network architecture. *ACM SIGCOMM Computer Communication Review 26*, 2 (Apr. 1996), 5–18.

[100] Thiele, Lothar, Chakraborty, Samarjit, Gries, Matthias, and Künzli, Simon. Design space exploration of network processor architectures. In *Proc. of First Network Processor Workshop (NP-1) in conjunction with Eighth IEEE International Symposium on High Performance Computer Architecture (HPCA-8)* (Cambridge, MA, Feb. 2002), pp. 30–41.

[101] Turner, Jonathan S. A proposed architecture for the GENI backbone platform. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (San Jose, CA, Dec. 2006), pp. 1–10.

[102] Turner, Jonathan S., Crowley, Patrick, DeHart, John, Freestone, Amy, Heller, Brandon, Kuhns, Fred, Kumar, Sailesh, Lockwood, John, Lu, Jing, Wilson, Michael, Wiseman, Charles, and Zar, David. Supercharging PlanetLab: a high performance, multi-application, overlay network platform. In *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications* (Kyoto, Japan, Aug. 2007), pp. 85–96.

[103] Turner, Jonathan S., and Taylor, David E. Diversifying the Internet. In *Proc. of IEEE Global Communications Conference (GLOBECOM)* (Saint Louis, MO, Nov. 2005), vol. 2, pp. 755–760.

[104] Ulana, Legedza, David, Wetherall, and John, Guttag. Improving the performance of distributed applications using active networks. In *IEEE INFOCOM* (1998).

[105] Ulana, Legedza, and John, Guttag. Using network-level support to improve cache routing. In *In Proceedings of the 3rd International WWW Caching Workshop* (1998).

[106] Vellala, Manoj, Wang, Anjing, Rouskas, George N., Dutta, Rudra, Baldine, Ilia, and Stevenson, Dan. A composition algorithm for the SILO cross-layer optimization service architecture. In *In Proc. of the Advanced Networks and Telecommunications Systems Conference (ANTS)* (Mumbai, India, Dec. 2007).

[107] Vinton, Cerf, Yogen, Dalal, and Carl, Sunshine. Transmission control protocol. RFC 793, DARPA, Sept. 1981.

[108] Wang, Yi, Keller, Eric, Biskeborn, Brian, van der Merwe, Jacobus, and Rexford, Jennifer. Virtual routers on the move: Live router migration as a network-management primitive. In *Proc. of the ACM SIGCOMM* (Seattle, WA, Aug. 2008), pp. 231 – 242.

[109] Wetherall, David J., Guttag, John, and Tennenhouse, David L. ANTS: A toolkit for building and dynamically deploying network protocols. In *Proc. of IEEE OPENARCH 98* (San Francisco, CA, Apr. 1998), pp. 1–12.

[110] Wiseman, Charlie, Turner, Jonathan, Becchi, Michela, Crowley, Patrick, DeHart, John, haitjema, Mart, James, Shakir, Kuhns, Fred, Lu, Jing, Parwatikar, Jyoti, Patney, Ritun, Wilson, Michael, Wong, Ken, and Zar, David. A remotely accessible network processor-based router for network experiementation. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (San Jose, CA, Nov. 2008), pp. 20–29.

[111] Wolf, Tilman. Challenges and applications for network-processor-based programmable routers. In *Proc. of IEEE Sarnoff Symposium* (Princeton, NJ, Mar. 2006).

[112] 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.

[113] Wolf, Tilman, and Franklin, Mark. Performance models for network processor design. *IEEE Transactions on Parallel and Distributed Systems 17*, 6 (June 2006), 548–561.

[114] Wolf, Tilman, Weng, Ning, and Tai, Chia-Hui. Design considerations for network processor operating systems. In *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)* (Princeton, NJ, Oct. 2005), pp. 71–80.

[115] Wolf, Tilman, Weng, Ning, and Tai, Chia-Hui. Run-time support for multi-core packet processing systems. *IEEE Network 21*, 4 (July 2007), 29–37.

[116] Xin, Huang, Shashank, Shanbhag, and Tilman, Wolf. Automated service composition and routing in networks with data-path services. In *submitted to the IEEE INFOCOM 2010* (San Diego, CA, Mar. 2010).

[117] Z., Albanna, K., Almeroth, D., Meyer, and M., Schipper. *IANA Guidelines for IPv4 Multicast Address Assignments*. Network Working Group, Aug. 2001. RFC 3171.

[118] Zhao, Li, Luo, Yan, Bhuyan, Laxmi, and Iyer, Ravi. Design and implementation of a content-aware switch using a network processor. In *Proc. of 13th International Symposium on High Performance Interconnects (Hot-I05)* (Stanford, CA, Aug. 2005).

[119] Zhou, Wenjiang, Lin, Chuang, Li, Yin, and Tan, Zhangxi. Queue management for QoS provision build on network processor. In *Proc. of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)* (San Juan, PR, May 2003), p. 219.