ABSTRACT

IYER, MOHAN L. Providing Bandwidth on Demand Services using Optical Network Design and the SILO Network Architecture. (Under the direction of Dr. George N. Rouskas. and Dr. Rudra Dutta.)

This research concerns the issue of providing applications with direct access to bandwidth on demand services in cloud computing environments. Computationally intensive applications like scientific computing, data visualization and financial trading, require large amounts of network bandwidth and cannot tolerate variations in network performance. A cloud provider can address these concerns by reserving bandwidth of varying magnitude for application flows while minimizing network cost. We use a two-pronged strategy to address this issue. First, we design a provider network that satisfies long-term estimated traffic demands using traffic engineering approaches, minimizing the network deployment cost. Second, we accommodate application bandwidth requests of varying magnitude without exposing the details of the provider's network to the application.

Cloud providers use optical networks to connect their infrastructure that is distributed over geographically diverse locations. We design an optical network to accommodate a given set of traffic demands with the objective of minimizing the number of wavelength interfaces or optical ports used by the nodes. Advances in optical communications enable grouping of wavelengths into wavebands and augment the communication devices, like optical cross connects, to switch wavebands. We propose a hierarchical approach which uses waveband technology to establish optical connections for the considered set of traffic demands. Simulations indicate that the hierarchical approach produces scalable solutions in terms of different waveband and cluster sizes.

Customers of cloud providers pay only for the services and infrastructure they use; hence, cloud providers need to use customizable architectures to offer customer applications differentiated services. We augment the Services Integration and controL Optimization (SILO) network architecture to provide applications with bandwidth on demand services. The SILO architecture allows network stacks to be customized per application flow and enables easy service deployment for cloud providers. High bandwidth application flows are accommodated over multiple virtual circuits set up between the source and destination; multiple low bandwidth application flows are groomed into a single virtual circuit. Each virtual circuit operates at network link speed and uses the optical connections established during network design.

Providing Bandwidth on Demand Services using Optical Network Design and the SILO Network Architecture

by
Mohan L. Iyer

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2010

APPROVED BY:

Dr. Rudra Dutta Co-chair of Advisory Committee	Dr. Mihail Sichitiu
Dr. David Thuente	Dr. George N. Rouskas Chair of Advisory Committee

DEDICATION

To my parents and my sisters,
for their unwavering faith, limitless love and constant support.

BIOGRAPHY

Mohan Iyer is a Ph.D. candidate in the Department of Computer Science at the North Carolina State University (NCSU), Raleigh. He was born and brought up in Pune, India. He received his B.E. degree in Computer Engineering from the D. Y. Patil College of Engineering, Akurdi affiliated to the University of Pune in July 2003. He worked in Persistent Systems, Pune as a Member of Technical Staff from July 2003 to July 2005. He was associated with the Interdisciplinary School of Scientific Computing, University of Pune as a project guide for M. Sc. (Scientific Computing) students.

He joined NCSU in August 2005 as a M.S. (Computer Networking) student but converted to the Ph.D. (Computer Science) program in August 2007. During his time at NCSU, he did internships with the Solaris Networking team at Sun Microsystems (now Oracle Corporation) and the Optical Transmission Business Unit and the Core Routing Business Unit at Cisco Sysytems. He served as the Officer-At-Large and the Vice-President for the Computer Science Graduate Student Association at NCSU. He was honoured as the Outstanding Teaching Assistant by the Computer Science Department and nominated for the same by the Graduate School in 2009.

ACKNOWLEDGEMENTS

I am very lucky to have two great advisors, Dr. George Rouskas and Dr. Rudra Dutta who have been giving their expert guidance throughout my research; had it not been for them I would not have transitioned from the Masters to the Doctoral program. I am grateful to Dr. Ilia Baldine for his insightful advice on crucial aspects of my research. Besides my advisors, I would like to thank the rest of my thesis committee: Dr. Mihail Sichitiu, Dr. David Thuente and Dr. Khaled Harfoush, for their insightful comments and questions on my research. To Anjing Wang who worked with me on certain areas of my research and for the wide ranging discussions about the Internet and its future.

This work is dedicated to my parents and my sisters who encouraged me all through my graduate studies. I would like to thank my brothers-in-law and my nephews for giving me different perspectives in life. To my friends, without whom I would not have been able to enjoy my graduate study.

Finally, I would like to acknowledge the support received by National Science Foundation (under grant ANI-0322107) for funding my certain parts of my research.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Cloud Computing and Cloud Providers

With the explosion of information, the computing needs of organizations to process relevant information have greatly increased. The amount of infrastructure required to satisfy organizational computing needs and the resources needed to maintain this infrastructure have proportionally increased. Hence, organizations find it expensive to build and maintain computing infrastructure. This infrastructure may support applications involving information processing, data analytics, and computer simulations, amongst others.

Cloud providers address the issues related to building and maintaining computing infrastructure for various organizations; they also update their software and hardware infrastructure to reflect the current state of the art. Organizations can lease, for specific time durations the required amount of infrastructure from cloud providers to execute their computational tasks. Examples of cloud providers include Amazon Web Service's Elastic Cloud and Simple Storage Service [3], Microsoft's Windows Azure [66], and Google's App Engine [47] platforms.

Cloud providers typically offer computational power and data storage abilities to customer organizations. The architecture used to lease out computational power needs to provide resource allocation mechanisms, deterministic services, and high performance computing while hiding the underlying technology that is used. Similarly, data storage architectures are expected to provide for data protection from device failure and disaster recovery from catastrophic events like earthquakes. Within a cloud provider's infrastructure, also referred to as the "Cloud", applications are hosted in a software container which imitates the software components on which the application is designed to run; such containers are known as Virtual Machines [98]. Data storage is typically used to store data generated during or on completion of the execution of the applications.

Cloud providers use networking technology extensively to help organizations access their applications and data placed in the Cloud. Each customer request for computational power is satisfied by creating virtual machines; each virtual machine is dynamically assigned a globally unique identifier known as an Internet Protocol (IP) address. Networking technology is also used to transfer data from one storage location to another. In addition to this, Cloud providers maintain computing infrastructure at different locations to (**1**) ensure service availability to customers in case of calamities, (**2**) take advantage of subsidized electrical and cooling costs at diverse locations, and (**3**) be located close to customers to address concerns of latency and user experience. This distributed infrastructure is connected via high bandwidth networks.

1.2 Cloud Networking Challenges

Recently, a number of scientific computing applications have emerged which are computationally intensive, generate massive amounts of data, and require high bandwidth network access to remote peers or instrumentation devices. As an example, distributed applications associated with the Large Hadron Collider experimental project need to handle approximately 20 Petabytes of data annually [39]. [84] argues that this data is rarely co-located, so there is a need to address distributed data challenges; clouds are gaining acceptance as a means to address the concerns of the scientific computing community [30, 57]. Real world applications performing distributed data mining also need high bandwidth network access to distribute data sets which are terabytes in size [15, 29]; analysis of these distributed data sets is performed to build a data mining model and perform query processing.

In addition to the scientific computing and data mining applications, there exist applications like CineGrid [26], financial trading systems, and data visualization that are computationally intensive and require deterministic network latency; the networking requirements of such applications cannot be guaranteed by carrier networks [39]. A cloud provider can address these issues by building and maintaining a network infrastructure to connect its distributed sites. Application demands for network bandwidth can be satisfied by leasing out the networking infrastructure to customers in conjunction with computational power and/or data storage.

A cloud provider's networking infrastructure must (**1**) provide large amounts of bandwidth to accommodate numerous application data flows, (**2**) satisfy the individual bandwidth requirements of application flows, (**3**) automatically detect network failures, and (**4**) recover from network failures within a short time span [74]. Optical networks address all of these concerns and hence are used to connect different campuses of the cloud provider; characteristics of optical networks are described in Section 1.2.1.

Most cloud infrastructure consists of computing and storage services delivered through data centers. Data centers house computing and storage infrastructure in controlled environments under centralized management [64]. The computing infrastructure consists of compute servers, while storage infrastructure consists of databases and storage systems. Within each data center, the computing and storage resources are inter-connected using high speed local area networks (LANs) consisting of Gigabit Ethernet and Fiber Channel, amongst other technologies. Data centers are connected to each other using wide area network (WAN) technologies like optical networks; section 1.2.3 describes the underlying network technology used within data centers and the manner in which they connect to optical networks.

Each compute server hosts multiple applications whose data flows have different data processing and resource requirements [31]. Applications must have mechanisms to make the network architecture aware of their requirements so that the network can accommodate them. Cloud providers need architectures which enable the seamless addition of new network functionality to provide differentiated, value-added services to applications. The set of features the network architecture should provide to cloud environments is discussed in Section 1.2.2.

1.2.1 Clouds and Optical Network Design

Optical Dense Wavelength Division Multiplexing (DWDM) network technology is well suited to address the networking needs of cloud providers. Optical DWDM networks offer bandwidth on the order of Terabits per second, have a high signal to noise ratio, and extremely low power requirements [2]. They are optical, circuit switched networks and thus provide better security and deterministic network latency due to flow isolation. Optical networks have well designed and scalable survivability mechanisms to detect network failures and recover from them [71].

Optical DWDM technology is protocol, and bit-rate independent. Optical DWDM networks can carry different traffic types and may operate at varied speeds simultaneously; we provide more details about this characteristic in Sections 1.2.3 and 2.2. This feature is particularly attractive since optical DWDM technology can accommodate future evolution in the data link layer protocols.

Optical fiber and network devices are expensive to deploy and maintain, which inhibits their adoption by cloud providers. The deployment cost can be minimized by efficiently designing the optical network based on the estimates of long-term traffic demands. The efficiency of network design is dependent on the accuracy of traffic demand estimates; however, these estimates are difficult to perform. The physical network topology can be designed using the conservative estimates of long term traffic demand; additional network resources can be added incrementally based on present traffic demand and estimates of future traffic demand [45].

The network design mechanisms satisfy traffic estimates by establishing optical connections between source-destination pairs. Each optical connection is identified by the route it traverses from source to destination and the communication channel it uses. Network design mechanisms concentrate on performing intelligent routing and channel assignment of the optical connections using traffic engineering algorithms. The objective of *traffic engineering* is to "put traffic where network bandwidth and other resources are available" in an efficient and effective manner [68]; this approach focuses on minimizing the amount of network resources. The set of optical connections form a virtual network topology over the physical topology.

Advances in optical communication technologies have enormously increased the number of optical communication channels, or *wavelengths*, in a fiber. Hence, the number of wavelength interfaces or optical ports required at optical switches or cross connects in network nodes has also increased, which increases switch cost. Recently, mechanisms to group multiple wavelengths into *wavebands* [18] have been developed. The optical cross connect architecture requires extensions to identify wavebands as a single entity while switching; this has led to the development of *Multi-Granular Optical Cross Connects (MG-OXC)* which switch data at various granularities including wavelengths, wavebands, and whole fibers. *Waveband Switching* is the associated optical network design problem which aims to reduce the MG-OXC switch size through traffic engineering and appropriate use of wavebands.

1.2.2 Cloud Applications and Bandwidth on Demand Services

Cloud providers prefer to use customizable and flexible Service Oriented Architectures (SOA) to provide customers with differentiated services [92] enabling customer applications leverage the infrastructure effectively. Customers pay cloud providers only for the services and the amount of infrastructure they use [65]. A similar architecture is required for networking services, which satisfies varied requests from different applications without exposing the details of the network infrastructure.

Customer requests to host applications over cloud infrastructure come at different instances of time; these applications require varied amounts of bandwidth to share information with their peers. The networking architecture should be able to accommodate new applications at all instances of time satisfying their requests for varied bandwidth requirements. Customers should be billed for the amount of network bandwidth used by their applications; hence, the networking architecture should have good accounting mechanisms to determine the network resources used by application flows. From a cloud provider's perspective, the networking architecture should accommodate as many and as varied application requests for bandwidth as possible while minimizing the use of network resources.

The present Internet architecture does not provide applications with any mechanism to request network resources; the bandwidth used by an application flow is dependent on the other flows using the network, and the available bandwidth is shared by all flows in the network. Applications are provided with only two options to process their traffic, (**1**) The Transport Control Protocol(TCP) [52] which provides reliability, flow control, congestion control, and in-order delivery, amongst other features, and (**2**) The User Datagram Protocol(UDP) [73] which supports none of these transport functionalities. The inability to request network resources and the lack of feature customization are major performance inhibitors for applications which require large amounts of network bandwidth [31].

The Services Integration and controL Optimization (SILO) Architecture [35] is a customizable, service oriented networking architecture developed by researchers at the NC State University and the Renaissance Computing Institute (RenCI). The SILO architecture provides customized, per flow network stacks to applications and enables the seamless addition of new functionality. Customized, per flow network stacks allow applications to use only the required network functions for their flows; seamless addition of new functionality helps cloud providers provide differentiated, value-added services. These characteristics enable the easy development of bandwidth-on-demand services within the SILO architecture, and "customizing" the network bandwidth specific to each application flow without exposing the underlying network infrastructure to the applications.

1.2.3 Networking within Cloud Provider Centers

Computing and storage resources within data centers are connected to each other and the outside world using a high speed LAN as shown in Figure 1.1. The compute servers and storage systems are connected to the Access-Aggregation layer, which contains multi-layer switches to offer services like load balancing, firewalls, web caching, and Intrusion Detection Systems. These services are provided by the data center to address the security concerns of applications. The Core layer connects the Access-Aggregation layer to other centers using optical networks, and to the outside world through the Internet Edge; the Core layer may also serve other areas of the center.

The network within the data center is composed of Gigabit Ethernet (1 Gbps or 10 Gbps) links which connect end systems, access and core layer switches. Since Ethernet does not provide any bandwidth guarantees, four extensions to the Ethernet standard have been proposed by the IEEE 802.1 Data Center Bridging Task Group [48]. A brief description about these extensions is obtained from [48] and mentioned in verbatim below.

1. "*Priority-based Flow Control (PFC)* provides a link level flow control mechanism that can be controlled independently for different priorities. The goal of this mechanism is to ensure zero loss due to congestion in data center bridged networks. This is achieved by sending a PAUSE frame for a specific priority, which halts the transmission of the sender for a specified period of time. Each priority is determined by the IEEE 802.1p Class of Service to differentiate among traffic types. Priority-based Flow Control is an active project P802.1Qbb with the IEEE 802.1 working group."

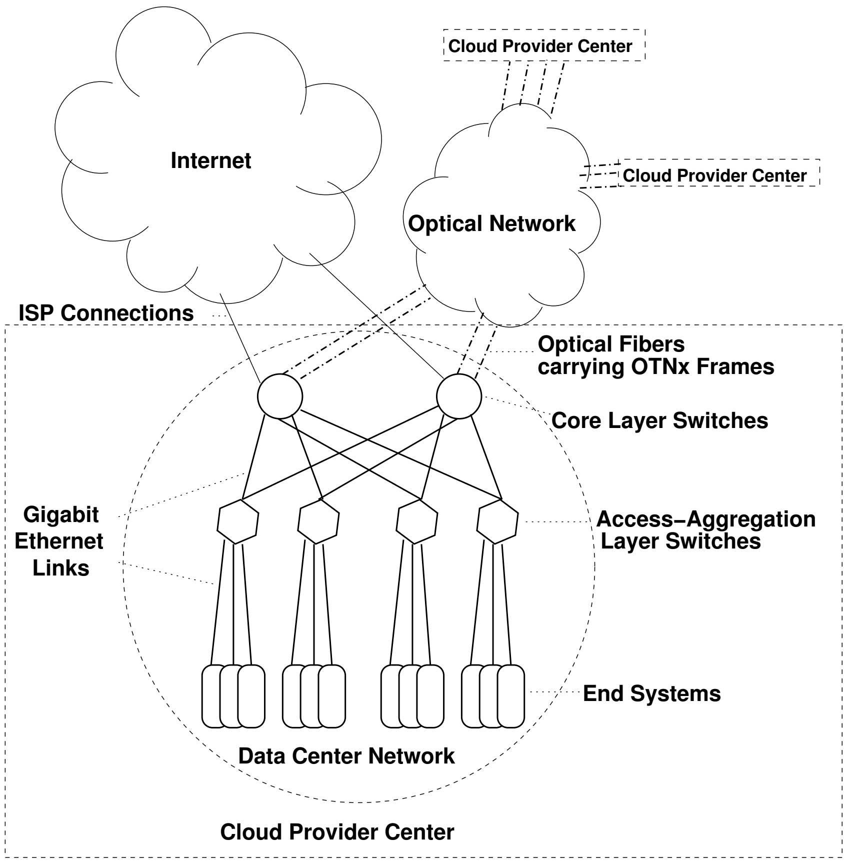


Figure 1.1: Data Center Network

2. "*Enhanced Transmission Selection (ETS)* provides a common management framework for assignment of bandwidth to traffic classes. When the offered load in a traffic class of higher priority does not use its allocated bandwidth, enhanced transmission selection will allow other traffic classes to use the available bandwidth. The bandwidth-allocation priorities will share bandwidth between bursty traffic loads while coexisting with the strict priority mechanisms carrying traffic requiring minimum latency. Enhanced Transmission Selection is an active project P802.1Qaz with the IEEE 802.1 working group."

3. "*Congestion Notification (CN)* provides end to end congestion management for protocols that do not already have congestion control mechanisms built in; e.g., Fiber Channel over Ethernet (FCoE). Congestion is managed by enabling bridges to signal congestion information to end stations; the end stations should be capable of transmission rate limiting, to avoid frame loss. Congestion Notification is an active project P802.1Qau with the IEEE 802.1 working group."

4. "*Data Center Bridging Capabilities Exchange Protocol (DCBX)*: A discovery and capability exchange protocol that is used for conveying capabilities and configuration of the above features between neighbors to ensure consistent configuration across the network."

Switches in the Access-Aggregation layer are expected to have these extensions; Cisco's Nexus Switch [27] series already incorporates these features. Switches ensure that no data frames are lost due to congestion in the data center network, using Congestion Notification and Priority Flow Control mechanisms. End systems are expected to use the notification from the switches to inform higher layer protocols about the congestion, enabling them take corrective actions.

The switches in the Core layer feed traffic from the data center to the optical network using the optical transport network protocol. The Optical Transport Network (OTN) protocol encapsulates Ethernet frames for transport over the optical network. OTN protocol has different levels which operate at different speeds. For example, OTN 1 operates at 2.66 Gbps, while OTN2 operates at 10.709 Gbps, and so on. OTN frames are carried over wavelengths within the optical networks; more information about OTN can be found in [69]. The mapping of different Gigabit Ethernet standards and their corresponding OTN levels is shown in Figure 1.2; it shows that two 1Gbps Ethernet links (shown by solid lines) map to one OTN1 link (shown by dotted-dashed lines), while eight 1 Gbps links map to OTN2, and so on. Please note that the figure is schematic, and the bandwidth numbers shown are not accurate; for more information on OTN interfaces, refer to ITU-T G.709 [1]. The option also exists of multiplexing data over different Gigabit Ethernet links into a higher speed OTN level or encoding the data directly onto the OTN level of the corresponding speed. The switching fabric at optical network nodes is unaware of the OTN level carried by the optical channels; this makes the optical network bit-rate independent and the switching fabric can accommodate optical channels performing transmission at different speeds simultaneously.

1.3 Thesis Contribution

This thesis enables cloud providers to deliver bandwidth on demand services of varying magnitude accessible by cloud applications over the cloud provider's network while minimizing the associated network cost. This vision is achieved using two-pronged strategy.

1. **Network Design:** We lower network deployment cost and accommodate a given set of long term traffic demands using network design mechanisms which consider node architecture and optical channel availability.

2. **Bandwidth on Demand Services:** We enable applications to request and obtain varying amounts of bandwidth from the networking architecture without exposing the under-

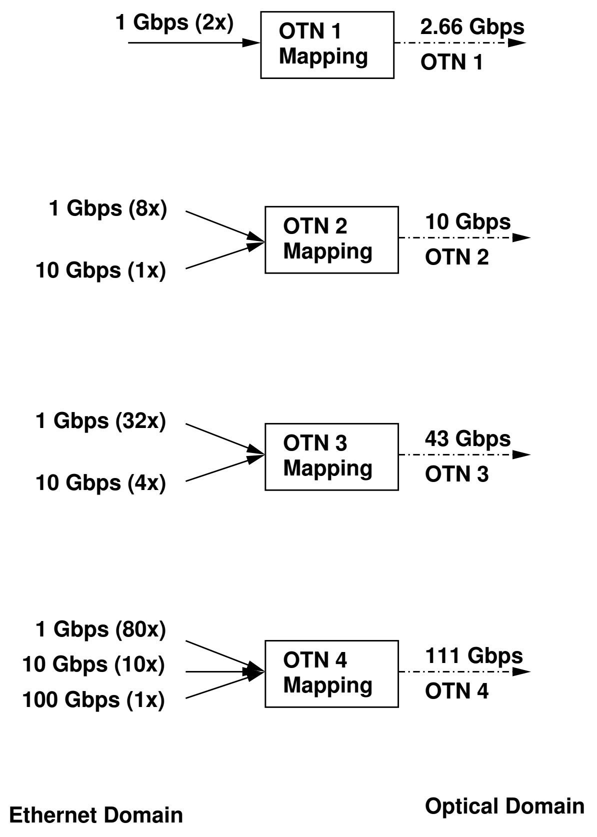


Figure 1.2: Mapping Ethernet Standards to the Optical Transport Network Protocol

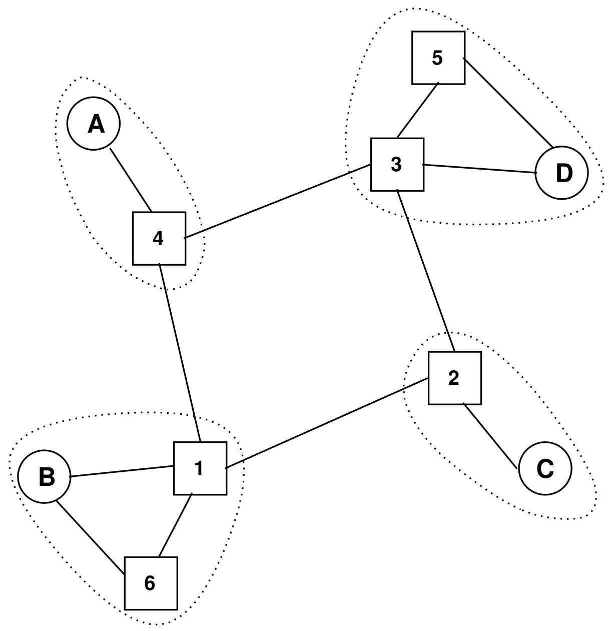


Figure 1.3: Optical Network Interconnecting Four Cloud Centers

lying network technology. The networking architecture utilizes the optical connections set up through the mechanisms of network design to satisfy application bandwidth requests.

We show the relevance of this work through an example shown in Figures 1.3-1.7. Figure 1.3 shows a cloud provider with four centers connected together by optical fibers; each fiber is capable of carrying four wavelengths, and traffic is fed into the optical network at 1 Gbps by all sites. We assume that the data center networks operate at 1 Gbps and are capable of automatic setup of relevant virtual circuits to connect end systems to the optical network; such automatic setup is possible under existing control architectures such as the Open Resource Control Architecture and Breakable Experimental Network (ORCA-BEN) [37]. The cloud provider still needs to determine the amount of network resources to provision for use; we consider the number of optical fibers and the amount of switching capacity at optical cross connects as the resources to be provisioned.

Centers A and C host data mining and video conferencing applications. The data mining application requires 700 Mbps network bandwidth for data transfer, while the video conferencing application requires 300 Mbps of bandwidth to connect remote campuses of an educational organization conducting Distance Education courses in real time. The cloud also hosts scientific computing applications from High Energy Nuclear Physics laboratories at centers B and D, which requires 3 Gbps of network bandwidth. The cloud provider is expected to provide computing power, data storage, and network bandwidth as required for these applications.

1.3.1 Optical Network Design

We consider the Waveband Switching network design problem of minimizing network cost while satisfying a given set of long-term traffic demands using wavebands and Multi-Granular Optical Cross Connects (MG-OXC). The network cost is determined by the total number of optical ports used and the number of wavelengths used per link in the network. We have designed efficient algorithms that address this problem over a hierarchical network; our solution considers any physical network that is partitioned into smaller, manageable clusters. We conduct a comprehensive performance evaluation using simulations and we present numerical results to demonstrate that our solution efficiently scales across different numbers of clusters and waveband sizes. This work was the first to study the effectiveness of different MG-OXC architectures based on the interactions of wavelength, waveband, and fiber.

In the example we consider, the cloud provider may use the hierarchical waveband switching mechanism over the network shown in Figure 1.3 to determine the minimum amount of resources required to service the long-term traffic demands. We consider the number of optical fibers and the number of optical ports used at optical cross connects to be resources; the number of optical ports required at an optical cross connect is determined by the number of optical connections that use the cross connect. The cloud provider needs to allocate one wavelength traffic demand from campus A to C and three wavelength traffic demands from campus B to D, represented on the figure by solid lines. The provider also estimates one additional traffic demand from campus A to C and campus B to D for future use, represented by the dashed lines.

The hierarchical approach considers a network partitioned into four clusters and groups two wavelengths into a waveband; two optical connections are established on each of the paths (Center A $\rightarrow$ 4 $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ Center C), (Center B $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ Center D), and (Center B $\rightarrow$ 1 $\rightarrow$ 4 $\rightarrow$ 3 $\rightarrow$ Center D) to service the estimated traffic demands. The solution is shown in Figure 1.4. It spreads the optical connections across multiple fibers to reduce the number of wavelengths used per link; two optical connections traverse the path (Center B $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ Center D), while another two traverse the path (Center B $\rightarrow$ 1 $\rightarrow$ 4 $\rightarrow$ 3 $\rightarrow$ Center D) to spread the link loads over the network even though their source-destination pairs are the same. In the final solution, two groups are formed, each containing two optical connections to enable MG-OXCs to switch the group as wavebands. The solid arrows represent resource allocation to satisfy the existing traffic demands, while the dotted arrows represent provisioning for future traffic demands.

The established optical connections form a logical topology over the physical network, as shown in Figure 1.5. The logical topology consists of four nodes representing the different campuses and edges representing the optical connections established between the campuses. This hierarchical approach to waveband switching is described in Chapter 5.

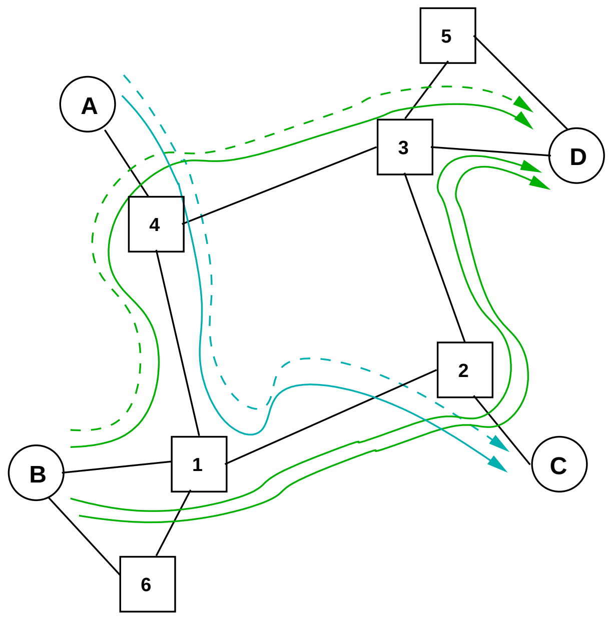


Figure 1.4: Lightpaths Routed Over the Optical Network

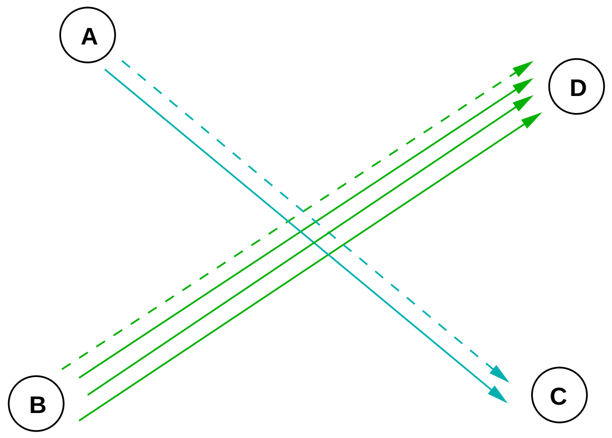


Figure 1.5: Logical Topology

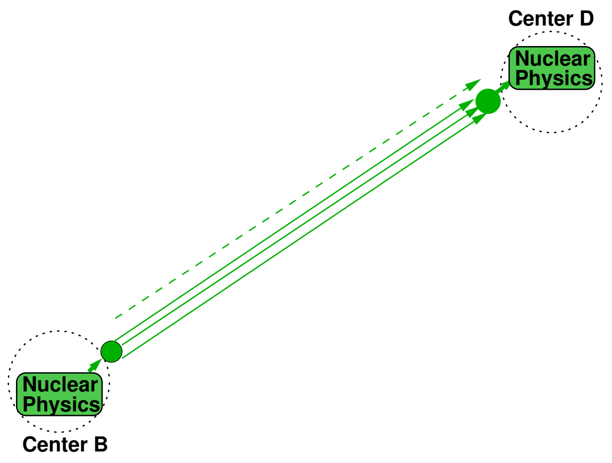


Figure 1.6: Virtual Concatenation

1.3.2 Bandwidth on Demand Services

We extend the SILO architecture [35] to provide applications with mechanisms to request bandwidth and enable end systems to process these requests based on the underlying network speed. We successfully hide the details of the network infrastructure from applications by augmenting the SILO architecture; the extensions enable the SILO architecture to efficiently accommodate application requests for bandwidth ranging from a fraction of the data center network link speed to multiples of that link speed.

We assume that virtual circuits are automatically established within the data center networks to service application flow. Virtual circuits are expected to use dedicated network links to service application flow at a deterministic rate. Switches at the Access and Core layers of the data center network also need to be provisioned with the required bandwidth and the path traversed by each flow; switches can satisfy the bandwidth requirement by tagging the flows with relevant IEEE 802.1p Class of Service and using Priority-based Flow Control. Virtual circuits can be identified with unique IEEE 802.1Q Virtual Local Area Network (VLAN) [51] tags or Multi-Protocol Label Switching (MPLS) [36] labels within the cloud provider's network. Thus, each virtual circuit will operate at the network link speed of the data center using dedicated network links and a proportional share of the switching fabric at the Access and Core layers; we assume that all links within data centers will operate at the same link speed.

Handling High Bandwidth Flows

Whenever an application requests an amount of bandwidth that is greater than the underlying network speed, the SILO architecture, as extended through this thesis, uses multiple virtual circuits to satisfy the request; the virtual circuits either originate with or terminate at the end

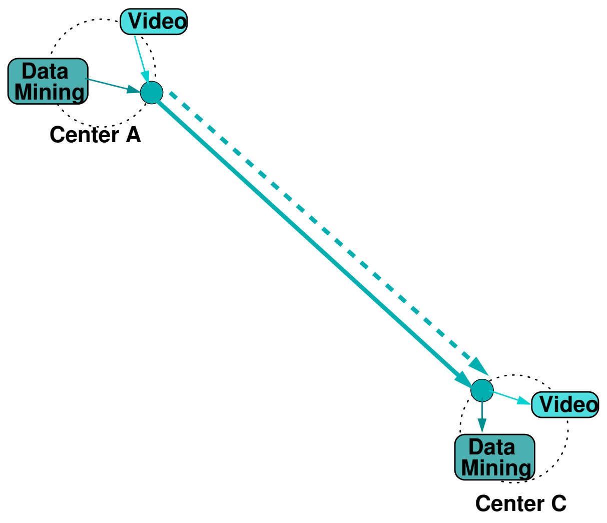


Figure 1.7: Traffic Grooming

system hosting the application. We designed mechanisms that enable applications to dynamically request and modify the amount of required bandwidth without knowing the underlying network details. These mechanisms separate the process of multiplexing from the policy used; this enables easy deployment of varied multiplexing policies like Round Robin, Priority Scheduling, and so on. We demonstrate these architectural extensions by implementing the SONET Virtual Concatenation (VCAT) feature; the application requests to vary the bandwidth are also accommodated by implementing a SONET Link Capacity Adjustment Scheme (LCAS). The VCAT/LCAS functionality implemented in SILO also provides support for handling network faults.

In Figure 1.6, the nuclear physics application flow of 3 Gbps is accommodated over three virtual circuits; these three virtual circuits use three of the four optical connections established from campus B to campus D. Data generated by the source application, is split into three flows of 1 Gbps, each of which is serviced by a virtual circuit; these virtual circuits connect the peer applications using the optical connections which are routed independently to campus D. The receiving application gets a 3 Gbps bandwidth flow, unified from the virtual circuits by the architecture at the destination. The process and entities which perform these tasks are described in Chapter 3.

Handling Low Bandwidth Flows

Whenever an application requests an amount of bandwidth that is less than the underlying network speed, the SILO architecture, as extended in this thesis, combines different application flows into one unified flow. The unified flow is serviced by a single virtual circuit; merged application flows may have different bandwidth requirements, but they originate from the same source and terminate at the same destination. Merging of application flows is performed to conserve network bandwidth, since virtual circuits use dedicated network links. We demonstrate these architectural extensions by implementing the different multiplexing policies of Round Robin and Deficit Round Robin [80]; dynamic addition of application flows into the optical connection is also accommodated by implementing the scheduling policies of Next Fit, First Fit, and so on.

In Figure 1.7, the Video conferencing and the data mining application flows of 300 Mbps and 700 Mbps, respectively, are provisioned over a single virtual circuit to reduce the requirement for network resources. The SILO architecture at the source grooms the application flows into a virtual circuit, in a bandwidth proportionate manner, without the applications realizing that such multiplexing is taking place. At the destination, the SILO architecture splits the data in the virtual circuit, directing each flow to the relevant application. The mechanisms and extensions to perform these tasks are described in Chapter 4.

1.4 Thesis Organization

This thesis is organized as follows. In this chapter, we have motivated the need for cloud providers to intelligently design their high bandwidth networks and satisfy application requests for varying amounts of bandwidth without exposing the underlying networking infrastructure. In Chapter 2, we briefly describe the SILO architecture and Optical Networking; we also discuss the research approaches related to Virtual Concatenation and Traffic Grooming. In Chapter 3, we propose extensions to the SILO architecture to accommodate high bandwidth application flows over multiple virtual circuits using optical network. We consider the inverse problem of provisioning low bandwidth flows in Chapter 4. In Chapter 5, we present a hierarchical approach to waveband switching which scales for different waveband and cluster sizes. Finally, we conclude the thesis and discuss future work in Chapter 6.

Chapter 2

The SILO Architecture and Optical Networks

In this chapter, we provide a brief introduction to the SILO Architecture, jointly developed at the North Carolina State University and the Renaissance Computing Institute, and Optical Dense Wavelength Division Multiplexing (DWDM) Networks. We describe the need for a new Internet architecture, the major components of the SILO architecture, and its prototype in Section 2.1. In Section 2.2, we describe essential optical networking concepts and devices before concluding with the routing and wavelength assignment problem in Section 2.2.

2.1 The SILO Architecture

2.1.1 Motivation for Network Architecture

The current Internet architecture was developed more than 25 years ago with the aim of interconnecting the existing networks of that time [28]. Network functions were broken down into layers, and protocols were designed or fitted into the layers based on their functionality. Today the Internet has successfully established itself as a modern communication infrastructure.

Over several decades, this architecture has evolved incrementally [33] to address the requirements arising arising as a result of the changing environment of heterogeneous users, service needs, economic structures, and threats. Typically, solutions to problems were (and are) engineered within the constraints of the current architecture. Such solutions are mostly suited to a specific context; for example, a number of Transmission Control Protocol (TCP) variants have been proposed for high bandwidth-delay networks [41, 96, 101], wireless networks [6, 14, 16, 102], and there are ongoing efforts towards cross-layer optimizations [63, 75, 81, 93, 99, 100]. In other cases, addressing broader needs, such as the Internet Protocol (IP) address shortage or security, has resulted in solutions which violate certain design principles or introduce new functionality to the architecture. For example, network address translation violates the end-to-end principle, while MPLS and transport layer security solutions were introduced at layers 2.5 and 3.5, respectively.

These workarounds for specific problems have raised concerns about the inflexible nature of the current architecture, the validity of fundamental design principles, and assumptions on which the current architecture was built. This ossification of the current architecture has resulted in calls for reconsidering the original principles and redesigning network architecture from scratch [4, 40]. There have been a number of initiatives in the United States (NSF FIND, DARPA NewArch) and Europe (FIRE) to address these concerns. Many are still skeptical of such efforts [33] and compare clean-slate approaches (revolution) with incremental modifications to the current architecture (evolution) through a biological metaphor. They make a case for the latter, claiming that evolution is more cost effective, competitive, and robust than revolutionary approaches.

The architects of the Services Integration and controL Optimization (SILO) architecture at the North Carolina State University and the Renaissance Computing Institute believe that although the current Internet architecture has been reasonably successful at addressing networking issues, it has three primary shortcomings. First, it does not allow the seamless addition of new functionality. Second, it does not support cross layer interactions. Last, implementation and experimentation with new networking algorithms and protocols requires substantial effort and is prohibitive for many researchers to undertake.

The SILO architecture is a new network architecture that supports the evolution of a flexible network architecture [35]. This framework enables evolution by (**1**) building blocks of fine-grained functionality, (**2**) explicit support for combining elemental blocks to accomplish highly configurable complex communication tasks, and (**3**) enabling control elements to facilitate or manage the architecture. We briefly describe the key components of the SILO architecture in Section 2.1.2. A prototype of the SILO architecture was built to demonstrate its feasibility; this prototype is described in Section 2.1.3. Section 2.1.4 describes existing and emerging approaches in the area of network architecture design and bandwidth on demand services.

2.1.2 SILO Architectural Components

The fundamental building blocks of the SILO architecture are *fine-grained services*; each service has a specific, reusable function that is performed on application data relevant to the specific communication task. This provides the architecture with more flexibility than current protocols, which typically embed complex functionality.

Each service may have some adjustable parameters specific to its functionality. Such parameters can be exposed by the service as *tuning interfaces or knobs* with a specified range

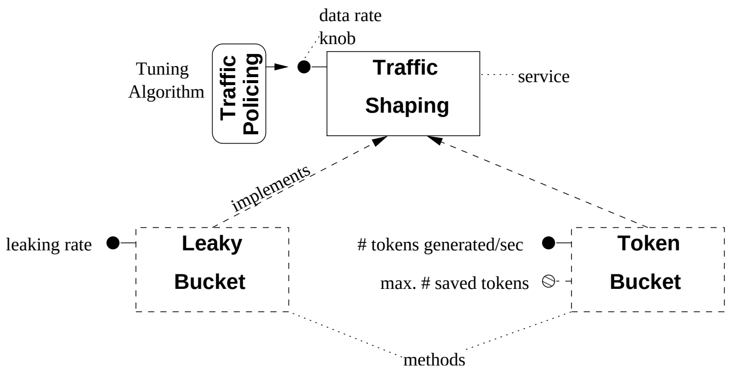


Figure 2.1: SILO Service Example

of values and a well defined relationship between these values and the perceived performance of the service. Hence, a service is defined by describing (**1**) the function it performs, (**2**) the interfaces it presents to other services, (**3**) any properties of the services that relate it to other services (required to establish partial ordering), and (**4**) the control parameters used to tune its performance, known as knobs.

To facilitate cross-layer interaction among services operating on a data flow and adjust network performance based on changing environmental conditions, the SILO architecture uses *tuning algorithms*, which monitor and adjust service performance via the manipulation of knobs.

In addition to this, SILO architecture differentiates between a service and its implementation or method. A *method* is an implementation of a service that uses a specific mechanism to carry out the functionality of the service. A method may also expose its method-specific control parameters as method-specific knobs. A method is described by (**1**) the service it implements, (**2**) the algorithm or mechanism used to implement the service, and (**3**) optionally method-specific control parameters or knobs.

To explain these concepts better, we consider the example of Traffic Shaping functionality, as shown in Figure 2.1. Since Traffic Shaping is a network function, it is considered a service. The traffic shaping service regulates the average rate of data transmission, which can be treated as a control parameter and exposed as a knob. Leaky bucket and token bucket are two different forms of traffic shaping and may be implemented as specific methods of the service. The data transmission rate knob can be interpreted by the leaky bucket method as the leaking rate; the token bucket method may interpret it as the number of tokens generated per second. To control the burstiness of the data flow, the token bucket may expose a method-specific knob representing the maximum number of saved tokens. The traffic policing algorithm may be implemented as a tuning algorithm which uses the knobs to monitor and “shape” the traffic according to the flow specifications.

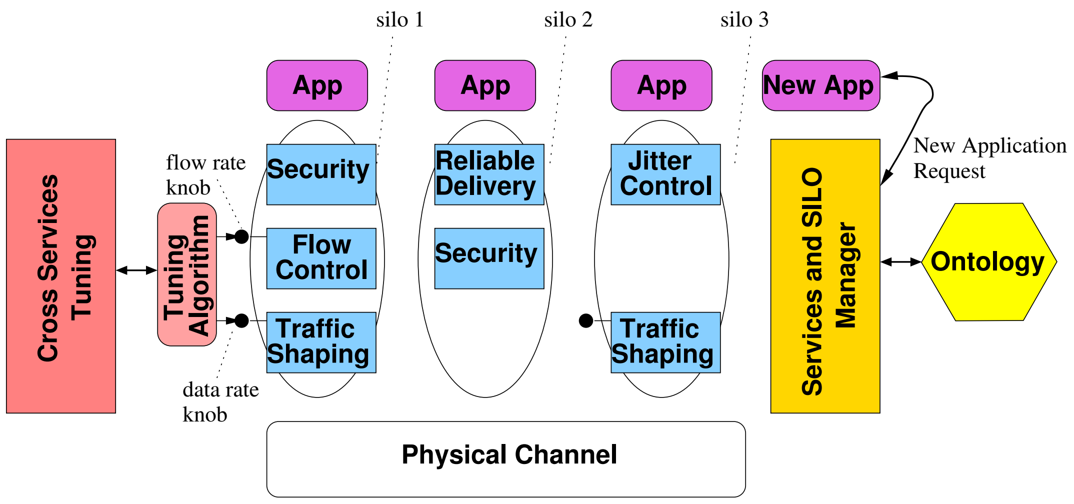


Figure 2.2: SILO Architecture

There are three major principles on which the SILO architecture is based.

1. *Generalization of the Layered Stack:* Services are composed vertically into silos so as to carry out an end-to-end communication task; *silos* can be thought of as a generalization of the protocol stack concept, and they are instantiated on a per-flow basis, allowing each flow (application) to customize its stack according to its own requirements and the properties of the underlying network. The silo concept decouples layers from services, allowing a service to be introduced at any point in the stack, wherever it is necessary rather than at a specific, predetermined layer. Figure 2.2 shows traffic shaping, flow control and transport layer security services placed vertically on top of each other, forming a silo. There are three applications, each with their own silo of services customized based on their requests. The decoupling of layers and service is also shown; the security service is placed in the fourth layer, implemented as Transport Layer Security, in silo 1 while it is placed in the third layer, implemented as Internet Protocol Security (IPSec), in silo 2.

2. *Cross-Layer Interactions:* The tuning algorithms and knobs enable cross-service or cross-layer interactions. The tuning algorithms consider jointly the behavior of the services in a silo and tune their knobs to the benefit of the communication task at hand. Tuning algorithms are managed by the cross services tuning agent. The *Cross Services Tuning* agent performs two tasks: (**1**) it selects an appropriate tuning algorithm based on the services in a silo at the time of its creation and (**2**) it schedules its execution at regular time intervals. In Figure 2.2, flow control service in silo 1 is placed above the traffic shaping service and exposes the maximum flow transmission rate as a knob. The tuning algorithm associated with silo 1 monitors the maximum flow transmission rate and varies

the data transmission rate of the traffic shaping service.

3. *Design for Change:* The SILO architecture provides mechanisms to introduce new services into the framework and compose them into silos. Specifically, the SILO architecture contains an ontology of services that stores service and tuning algorithm semantics (e.g., their function and data and tuning interfaces) as well as their relationships with each other in the form of constraints (e.g., relative ordering constraints). The SILO architecture also implements a composition algorithm that takes as input service requests (if any) from an application, the information and constraints in the ontology to construct a valid silo [91]. Consequently, introducing a new service is straightforward: a developer only needs to provide a description of the service and register it with the ontology for users to be able to include it in their silos.

Finally, the *Services and SILO Manager* is responsible for (**1**) the creation of a silo based on application requests, ontological constraints, and policies, and (**2**) scheduling the silos and executing the various service modules within each silo to process the application data flow. For additional information on the SILO architecture, the reader is referred to [7, 91].

2.1.3 Basic SILO Prototype

This section briefly describes the implementation details of the SILO architectural concepts; a comprehensive description is given in [7]. The SILO architecture primarily consists of four components, as shown in Figure 2.3. The *SILO Enabled Application* interacts with the SILO architecture to communicate with its peers using the *SILO Application Programming Interface (API).* The SILO API provides functions similar to the UNIX/Berkeley sockets interface for applications using the SILO architecture.

The *Universe of Services and Tuning Algorithm Storage (USTAS)* is a repository for services and tuning algorithms used within the SILO architecture. It contains (**1**) the relations between different services, their methods, and associated tuning algorithms which define the *SILO Ontology*, and (**2**) the policies and constraints under which they need to operate.

The *SILO Construction Agent (SCA)* uses the SILO Ontology and the prevalent policies to create a silo recipe which satisfies an application service request at the time of silo creation. The *SILO Management Agent (SMA)* uses the recipe created by the SCA to construct a silo using the object files stored within USTAS. In addition, it is responsible for maintaining the silo's state during communication sessions and for tuning services to optimize the performance of one or more silo.

As shown in Figure 2.3, the SILO components interact extensively with one another while handling requests for creating or destroying customized silos from SILO Enabled Applications.

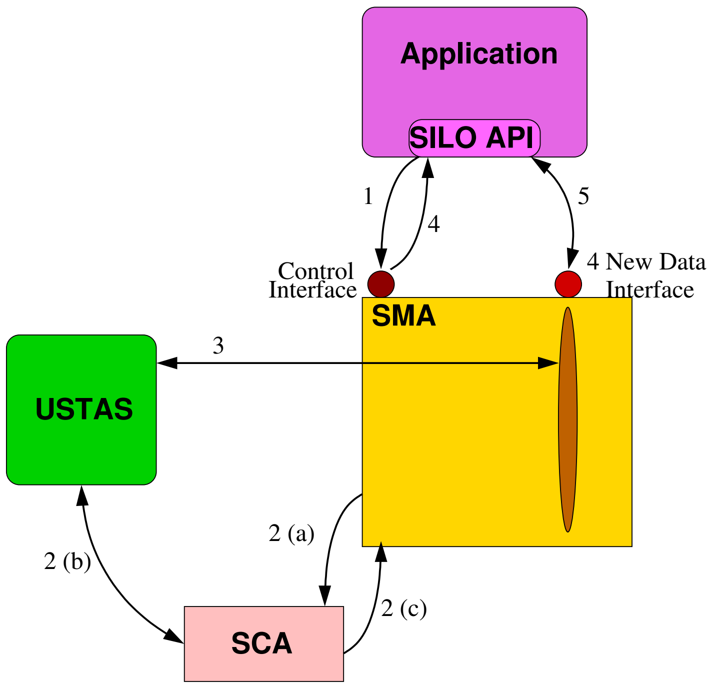


Figure 2.3: Prototype of the Original SILO Architecture

1. Each SILO Enabled Application creates a service request and forwards it to the SMA using the SILO API on a well known control interface (named pipes) exposed by the SMA.

2. Each application request is handled by the SMA in the following manner:

   (a) The SMA passes the application request to the SCA to verify the validity of the request.

   (b) The SCA consults the USTAS to process the application request based on composability constraints and enforced policies. If the request does not satisfy any policy or all composability constraints, a failure notification is sent to the SMA.

   (c) The SCA identifies any missing services or required tuning algorithms and adds them to the request. A silo recipe is created based on this request which contains the ordering of services within the silo.

3. Depending on the SCA's response, the SMA either gives a failure notification to the application or creates a silo using the object files of service methods and tuning algorithms in the USTAS.

4. On successful creation of the silo, the application is given a silo handle through the control

interface and notified of a newly created data interface (socket), associated with the new silo, for use by the application.

5. The application uses its data interface and silo handle for performing its communication tasks with its peers.

6. Once the application releases its silo handle, the SMA destroys the data interface of the application and the objects of the services and tuning algorithms associated with the relevant silo.

The process listed above is performed for every application requesting a silo from the SMA. An application may request multiple silos from the SMA; the SMA does not distinguish between requests made by different applications. For each new request processed, the application is given a new data interface and a new silo handle. This prototype implements the SMA and SCA as different processes; they communicate with each other using UNIX named pipes. The SMA is implemented in C++ using the event driven model, includes Cross Services Tuning, and is a part of the Service and SILO Manager. The SCA has been implemented in Python due to Python's string handling capabilities and is a part of the Service and SILO Manager.

2.1.4 Related Work

In this section, we initially describe new approaches to network architecture design similar to the SILO architecture. We later describe emerging mechanisms to provide network resource reservation mechanisms to applications using either network virtualization or optical networking.

Network Architecture Design Approaches

Architectural approaches such as x-kernel [50] provide a modular, generalized, but highly structured way for implementing network protocols. Each protocol is an object with service interfaces (for protocols at the same end) and peer interfaces (for protocols at the other end). x-kernel defines the syntax for interfaces of the protocols, not the semantics of protocols. The Coyote system [12, 11] extends x-kernel to introduce "microprotocols". A suite of microprotocols is composed to form a coyote service, which is also called a "composite protocol". Composite protocols are similar to traditional protocols, such as TCP and Open Shortest Path First (OSPF), which can be used to form a network subsystem. They are similar to protocols in x-kernel, which can have uniform interfaces, such as 'pop()' and 'push()'. Microprotocols can share variables, such as event handlers. The microprotocols are linked together statically to composite protocols.

While SILO may seem similar to x-kernel, there are several major differences: (**1**) x-kernel was an OS-centric effort in implementing existing network protocols as sets of communicating processes inside the novel kernel, while SILO is an attempt to introduce a network protocol meta-design that is independent of any assumptions about the underlying OS; (**2**) x-kernel made an early attempt at streamlining some of the cross-layer communications mechanisms, while SILO makes cross-layer tuning and optimization possible with an explicit focus on the framework, and finally (**3**) SILO is focused on the problem of automated, dynamic composition of protocol stacks based on individual application requests and module composability constraints, while the x-kernel protocols are pre-arranged statically at boot time.

Among recent architectural approaches, there are two projects whose scope extends to include the whole network stack and are hence most closely related to the SILO project. The first is work on Role-Based Architecture (RBA) [13], which considers a non layered approach to the design of network protocols, and organizes communication in functional units referred to as "roles". Roles are not hierarchically organized, and thus may interact in many different ways; as a result, the metadata in the packet header corresponding to different roles, forms a "heap", not a stack as in conventional layering, and may be accessed and modified in any order. The main motivation for RBA was to address the frequent layer violations that occur in the current Internet architecture and the unexpected feature interactions that emerge as a result [13], as well as to accommodate middle boxes.

The second is Recursive Network Architecture (RNA) [85], which introduces the concept of a "meta-protocol" that serves as a generic protocol layer. The meta-protocol includes a number of fundamental services, as well as configurable capabilities, and serves as the building block for creating protocol layers. Specifically, each layer of a stack is an instantiation of the same meta-protocol; however, the meta-protocol instance at a particular layer is configured based on the properties of the layers below it. The use of a single, tunable, meta-protocol module in RNA makes it possible for it to support dynamic service composition and facilitates coordination among the layers of the stack; both are design goals of our own SILO architecture, which takes a different approach in realizing these capabilities.

Emerging Approaches to Network Resource Reservations

Network virtualization techniques like Virtual Internet [87] and X-Bone [86] aim to reduce administrative burden by using overlay networks and multi-homing to virtualize networks, thereby abstracting out network complexity and providing isolation-based protection to encourage resource sharing. PlanetLab [9] is a geographically distributed overlay platform designed to support the deployment and evaluation of planetary-scale network services. Services within PlanetLab share infrastructure; resources are allocated to services based on the specification requested at service creation time. Veritas [10] aims to provide a testbed for network experimentation while simulating a realistic and controlled network environment; real Internet traffic helps to evaluate new services and controlled network events help to verify experiments in a repeatable manner.

Crossbow [88, 89] enables independent network stacks by slicing Network Interface Card (NIC) resources within a node into virtual NICs (vNICs). [62, 5] are prototypes which virtualize the data plane of physical routers, enabling them to factor in considerations for virtual networks. vNICs perform resource sharing in a fair manner between either applications or virtual networks; the bandwidth available to each network interface is partitioned amongst the vNICs by the network administrator. The Traffic Grooming mechanism we developed within the SILO architecture (Chapter 4) differs from vNIC based mechanisms ([88], [5] and [62]) in the following manner: (**1**) bandwidth sharing is performed per vNIC in these mechanisms ([88], [5] and [62]); Traffic Grooming performs bandwidth sharing per application flow, (**2**) vNICs are created and allocated bandwidth shares by the network administrator; Traffic Grooming allows flows to be dynamically added based on application requests, (**3**) multiple flows using a vNIC share bandwidth in an arbitrary manner; Traffic Grooming allows application flows to share available bandwidth in a deterministic manner, and (**4**) Traffic Grooming provides for easier deployment of varied multiplexing policies than vNIC based mechanisms.

IEEE 802.1AX Link Aggregation [42] mechanisms enable multiple physical links to be considered like a single logical link, with the aim of providing fault-tolerance and high speed links for higher layers to use. The set of physical links aggregated in a logical link is called a link aggregation group. All physical ports in the link aggregation group must reside on the same switch, resulting in a single point of failure; this limitation is overcome by Split Multi-Link Trunking [77], which is yet to be standardized. Link aggregation mechanisms aggregate one hop links, interconnecting hosts and switches, while the Virtual Concatenation mechanism we developed within the SILO architecture (Chapter 3) considers paths between source and destination hosts. The extensions developed for Virtual Concatenation can also be applied to Link Aggregation mechanisms.

2.2 Optical Networking

The current Internet is composed of three network tiers namely, access, metropolitan and backbone networks. Access networks feed traffic to metro networks from computers, cell phones, and so on; access networks use technologies like Digital Subscriber Loop, and Ethernet to provide Internet access to these end system devices. Metropolitan networks aggregate traffic from multiple access networks to feed traffic into the backbone networks; metropolitan networks typically span an organization's campus or a large metro area and use technologies like Metro Ethernet, and SONET amongst others. Backbone networks typically connect networks sepa-

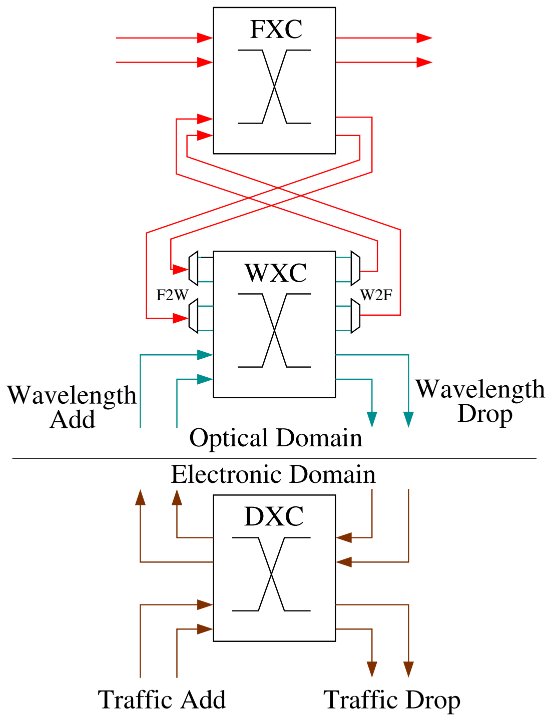


Figure 2.4: Edge Node Architecture

rated over geographically diverse regions and have high bandwidth capacity; they primarily use optical transmission technologies like Dense Wavelength Division Multiplexing (DWDM). In this section, we discuss characteristics of optical DWDM networks and the associated devices.

2.2.1 Optical DWDM Network

Nodes within an optical network are connected by optical fiber, which offers enormous bandwidth capacity and low transmission noise. *Dense Wavelength Division Multiplexing (DWDM)* enables the fiber to carry different optical signals in separate communication channels, known as wavelengths [82]. A *wavelength* is a communication channel operating in a dedicated frequency band and carries data as light pulses over the optical fiber. Each optical fiber can carry a fixed number of wavelengths in parallel; the current state of optical transmission technology and physical characteristics of the fiber determine the number of wavelengths that each fiber can accommodate. Commercial systems today are capable of carrying 120-150 wavelengths per fiber.

Presently, optical fibers can offer aggregate bandwidth of several Terabits per second; however, electronic devices are only able to process data in the order of Gigabits per second. The mismatch between the amount of data the fiber can transport and the amount electronic devices can process, is known as the *electro-optic bottleneck* [67]. The effect of the electro-optic bottleneck can be minimized by intelligently designing the optical network architecture based on the characteristics of optical network devices and fiber. An *Optical DWDM Network* is a network connecting nodes using optical fiber for data transmission and is based on an architecture that exploits the features of optical fibers and DWDM technology.

The architecture for wide area optical DWDM networks that is widely expected to form the basis for future optical infrastructures is built on the concept of *wavelength routing* [45]. A wavelength routing network consists of two kinds of nodes, namely, edge nodes and optical nodes. *Edge nodes* connect electronic devices, such as Internet Protocol (IP) routers, Asynchronous Transfer Mode (ATM) switches, or supercomputers to the optical network; the edge nodes have an electronic interface to interact with electronic devices and an optical interface to interact with other optical network nodes. *Optical nodes* are essentially composed of optical cross connects, and they switch traffic between different fibers in the network.

The architecture of an edge node is shown in Figure 2.4; it consists of an electronic domain and an optical domain. The electronic domain consists of a digital cross connect (DXC), which aggregates traffic from the electronic devices and feeds it to the optical domain for transmission as optical signals. Traffic is fed into or extracted from the optical domain using transceivers capable of identifying wavelengths; these transceivers are represented by W_{add} and W_{drop} arrows. The optical domain consists of a wavelength cross connect (WXC) and fiber cross connect (FXC), which together form the *Optical Cross Connect (OXC)*. The WXC switches optical signals extracted from incoming fiber's wavelengths to wavelengths which are a part of the outgoing fiber; the WXC also adds (drops) optical signals into (from) wavelengths at a node. The wavelengths are extracted from an incoming fiber by a demultiplexer (shown as F2W) and combined into an outgoing fiber using a multiplexer (shown as W2F). The FXC switches entire optical fibers.

2.2.2 Lightpath and Wavelength Routing

Edge nodes communicate with each other using logical connections known as *lightpaths*. Lightpaths are essentially clear optical connections between any two edge nodes and can span across one or more hops. Data transmitted over a lightpath always remains in the optical domain; hence, intermediate optical nodes perform only switching of lightpaths and do not perform any processing on the data they transport. Since data processing capability is not required at any optical nodes, the optical network can simultaneously accommodate data flows operating at different speeds within lightpaths. Figure 2.5 shows an optical DWDM network with edge nodes (A, B, C and D) and optical nodes (1, 2, 3 and 4) that accommodates traffic demands from node A to C and node A to B. In Figure 2.5, both of the traffic demands are satisfied using

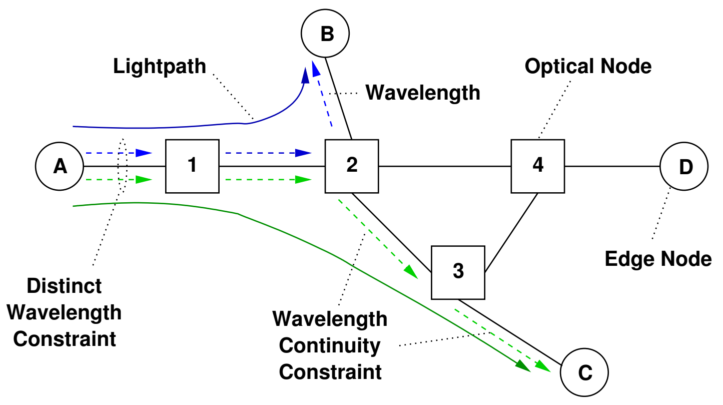

B
Lightpath
Wavelength
Optical Node
A
1
2
4
D
3
Edge Node
Distinct Wavelength Constraint
Wavelength Continuity Constraint
C

Figure 2.5: Optical DWDM Network

lightpaths represented by solid lines, while wavelengths are represented using dashed lines.

Each lightpath is identified by the path it traverses from the source to the destination and the wavelength it uses on each link in the path. Figure 2.5 shows the green-colored lightpath originating from source node A and terminating at destination node C, traversing the path A $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ C while using the light green-colored wavelength. A blue-colored lightpath traverses from A to B on the path A $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ B using light blue-colored wavelength.

Since the lightpath never uses the electronic domain, it must use the same wavelength on all the fiber links it traverses on the path from source to destination; this constraint is known as the *Wavelength Continuity Constraint*. In Figure 2.5, green- and blue-colored lightpaths use the light green- and light blue-colored wavelengths respectively, on each link they traverse.

To identify every lightpath traversing any particular link, all lightpaths using a link must be allocated distinct wavelengths on that link; this constraint is known as the *Distinct Wavelength Constraint*. In Figure 2.5, two lightpaths (green and blue) use different wavelengths (light green and light blue) on every common link (A-1 and 1-2) in their paths.

Network Design using Wavelength Routing

The wavelength continuity and distinct wavelength constraints indicate a tight coupling between routing and wavelength assignment. Thus, establishing a lightpath between any two nodes requires routing (selecting a suitable path) and wavelength assignment (allocation of an available wavelength) to be performed simultaneously. This problem is known as Routing and Wavelength Assignment (RWA) [104], and it occurs primarily in two forms: (**1**) a static version, which is applicable when the traffic requirements are predetermined, and (**2**) a dynamic version in which the lightpaths requests arrive in some arbitrary manner. We discuss the static version of this problem below; for details regarding the dynamic version of the RWA problem, the reader is

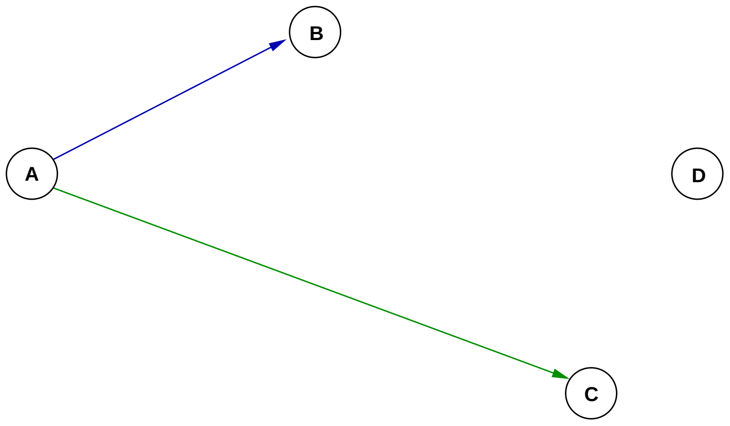


Figure 2.6: Logical Topology

referred to [78].

The objective of the static RWA problem is to determine the minimum amount of network resources required to satisfy a given set of traffic requirements; the distribution of resources across the network is also determined. The network resources typically considered are the network link load, and network latency, amongst others. In this context, the network link load is measured in terms of the number of wavelengths being used on a link, while the network latency is typically measured in terms of the hop count.

Since traffic requirements are known in advance and variations in traffic occur only over long time scales, the static RWA problem establishes long-term lightpaths which create a logical topology between the edge nodes. This logical topology is constrained by the physical network topology, which is composed of optical fiber links and network nodes. The vertices in the logical topology represent the edge nodes, while the topology links represent the lightpaths connecting the edge nodes. Figure 2.6 shows the logical topology for the optical DWDM network shown in Figure 2.5. The logical topology is used by backbone network service providers to route data flows between edge nodes, as well as by cloud providers to interconnect their centers.

Chapter 3

Bandwidth on Demand Service - Virtual Concatenation

3.1 Introduction

The SILO architecture gives applications the freedom to customize the network stack or silo that operates on its data flow. The application can list these services relevant to its data flow and the order in which the services operate on the flow. The architecture may also choose to include some services in the application's silo based on the services requested. Additions to or modifications of existing services and their implementation methods in the SILO architecture can be performed seamlessly by the cloud service provider; this increases the breadth of service and method offerings available for use, as relevant to the application requirements or the underlying network technology.

The flexibility offered to customize silos is desirable, but the applications also need to reserve the network resources, like bandwidth, that are used by their data flows; resource reservations are required to provide applications with consistent network performance, independent of other data flows using the network. Reservations are preferred by cloud service providers since they enable better accounting mechanisms for tracking the usage of network resources by applications. Resources reservation requests by applications can be serviced by the architecture based on their availability, network administration policy, and priority of the application. As mentioned in Section 1.2.1, cloud providers are expected to use optical networks as a result of their high bandwidth capacity and their bandwidth reservation and protection mechanisms to support mission-critical, on-demand simulations [97].

Cloud providers will aim to satisfy diverse requests of varying magnitude made by numerous applications, including requests for resources such as storage capacity, computational power, or network bandwidth. Cloud providers would like to satisfy application requests for network resources without revealing the underlying networking technology they use [72]. For the SILO architecture to address both these concerns, it needs mechanisms for the flexible sharing of network resources between different applications.

We augment the current SILO architecture with mechanisms that satisfy application requests for bandwidth of varying granularities using features provided by the underlying network. In this chapter, we consider application bandwidth demands which are greater than the underlying network link speed; we have designed and implemented application interaction and flow aggregation mechanisms to satisfy such application requests. We demonstrate these mechanisms using SONET's Virtual Concatenation (VCAT) feature, described in Section 3.1.1. Application interaction and flow aggregation extensions to the SILO architecture are described in Section 3.2; Section 3.3 presents the prototype implementation of these extensions. Section 3.4 describes the experimental setup, verification, and validation of these extensions, and Section 3.5 provides a summary of this work. In Section 2.1.4, we discussed the research related to work described in this chapter.

3.1.1 Overview of Virtual Concatenation

Virtual Concatenation (VCAT) [90] enables high bandwidth application flows to utilize medium bandwidth network links by establishing multiple non-contiguous, virtual tributaries or virtual circuits. Each *Virtual Tributary* is essentially a virtual circuit connecting the source and the destination operating at link speed.

At the sender, the high bandwidth flow is split into multiple medium bandwidth flows, each of which is accommodated in a virtual circuit. These medium bandwidth flows are routed using virtual tributaries and reassembled at the receiver into a high bandwidth flow.

The set of virtual tributaries used to accommodate the high bandwidth flow is called a virtual concatenation group and is identified by a *VCAT Group Identifier (GID)*. Since virtual tributaries in any VCAT group need not use the same path, each medium bandwidth flow in the VCAT group may experience a different network delay based on the path they use. The difference in network delay among various virtual tributaries is called differential delay; a *Multi-Frame Indicator (MFI)* field is used by the destination to accommodate the differential delay. The *Sequence Number (SQ)* field is used to label the different virtual tributaries within the VCAT group. The MFI and SQ fields help the destination to ensure packets are in-order after reassembly.

Since application demands for bandwidth can vary over time, the capacity of the VCAT group may not be used efficiently. To improve this utilization we have implemented the *Link Capacity Adjustment Scheme (LCAS)* protocol [53], which provides mechanism to adjust the size of the VCAT group.

The LCAS protocol ensures that virtual tributaries may be added or deleted without any data loss but requires them to carry LCAS control signals, like Control, Member Status, and Resequence Acknowledgement. The *Control (CTRL)* field is used to add or delete virtual tributaries dynamically based on the application requests or network faults. The *Member Status (MST)* field is used to inform the peer about the status of the virtual tributaries. Lastly, the *Resequence Acknowledgement (RS-Ack)* field is used to resequence the virtual tributaries upon modification of the VCAT group size.

Whenever virtual tributaries need to be added (deleted), the network administrator needs to provision (tear down) the virtual circuits connections between the source and destination before (after) informing the VCAT/LCAS module about the addition (deletion).

3.2 Extensions for Virtual Concatenation

The current SILO architecture allows for an application to request a customized, per flow network stack with cross layering enabled through tuning algorithms which improve performance based on changing network environments. The SONET VCAT functionality primarily deals with splitting (merging) a flow into (from) smaller constituent subflows, and the LCAS protocol enables varying the flow's network bandwidth based on its needs. To implement VCAT/LCAS functionality within the SILO architecture, extensions are needed that (**1**) enable applications to modify their flow characteristics based on a changing environment and (**2**) provide mechanisms for aggregating multiple flows into a single flow and vice-versa. We have developed both of these extensions, which are described below.

3.2.1 Enabling Interactions between an Application and its Flow

This extension provides the SILO architecture with new mechanisms to adapt the network performance to an application's changing needs. Modifying an application's flow characteristics or parameters enables the SILO architecture to deliver network performance as required by an application. Since the SILO architecture handles each application flow separately, we can easily limit the impact of an application request on other flows in the system based on global policies maintained by the Service and SILO Manager.

Since every flow is operated by a silo, we can modify the flow characteristics or parameters by tuning the knobs exposed by services to obtain the desired network performance from the silo. To satisfy the application request, we need to identify the relevant flow parameters, the services which control these parameters, and the tuning algorithm which converts abstract application requests into actionable tasks like setting knobs to control service behavior or requesting resources from the Service and SILO Manager.

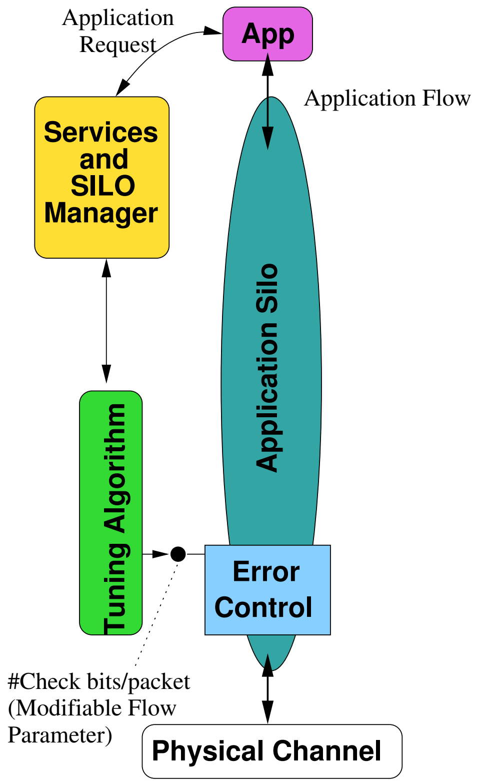


Figure 3.1: Application and Flow Interactions

The tuning algorithm and service designers may choose the flow parameters that can be modified by the application. The Services and SILO Manager enforces networking resource control policies system-wide. Each request can be satisfied based on its feasibility, resource availability, prevailing network conditions, and policies. While this extension provides applications with significant flexibility, it cannot (**1**) change the composition of services or tuning algorithms associated with its silo or (**2**) control the behavior of service instances which do not belong to silos operating on its flow.

In the example shown in Figure 3.1, the error control service and the tuning algorithm allow the application to modify the number of check bits per packet. This enables the application to increase the number of check bits before transmitting critical data to the receiver.

3.2.2 Enabling Flow Aggregation

The original SILO architecture allowed only one silo to operate on application flow. This limitation is cumbersome, as it results in duplication of services between silos and increases the complexity of managing all the flows. Figure 3.2 shows two applications with their customized silos performing data transmission using an Ethernet Interface. Each application requests a

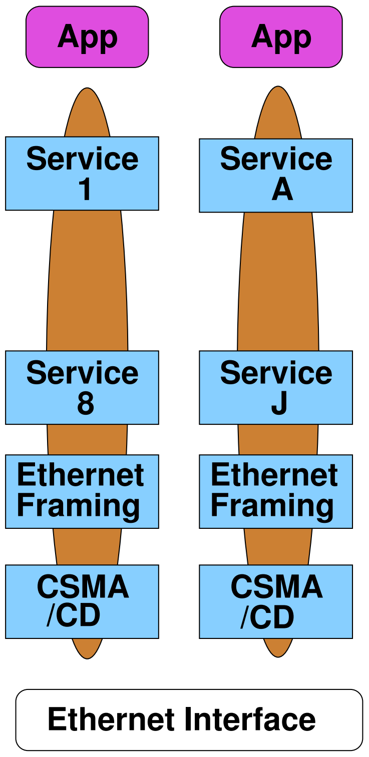


Figure 3.2: Duplication of Services in the original SILO Architecture

different set of services, represented by Service 1 to Service 8 and Service A to Service J, respectively, to operate in their customized silos. However, the Services and SILO Manager adds the Ethernet Framing and CSMA/CD services to the application silos, since data transmission occurs over the Ethernet Interface. This example shows that duplication of services which are not requested by the application can occur within the original SILO architecture.

Flow aggregation mechanisms enable the SILO architecture to be more flexible, addressing both of these concerns. Using flow aggregation mechanisms, one data flow can be created by merging multiple data flows. Conversely, one data flow can be split into multiple constituent data flows. Each data flow is operated by a silo until it merges with other flows, splits into multiple flows, or terminates at the application or physical channel. The lifetime of a silo depends on the lifetime of the flow associated with it; removal of the flow results in destruction of the associated silo. Similarly, a new silo is created to operate new flows added by applications. Thus, application data may be operated on by multiple silos, depending on the aggregation mechanisms used over the application's flow.

To implement flow aggregation mechanisms, we introduce the concept of a *Connection Point* where two or more silos can bind with each other and operate on the flow. When more than

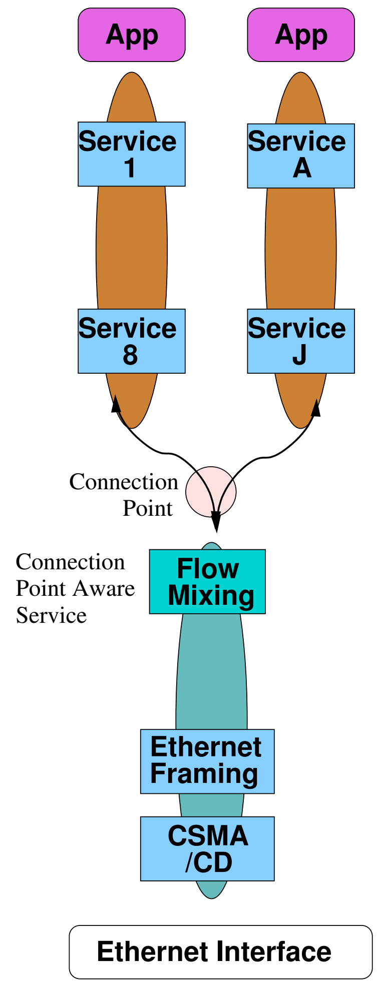


Figure 3.3: Flow Aggregation in the SILO Architecture

two silos are bound to a connection point, the data flow can be split into (or merged from) constituent subflows with a silo operating over each flow and subflow. A connection point cannot exist with just one bound silo, as we need at least two silos to ensure that data transmission takes place. A connection point uses *Insertion Points* to differentiate between the various silos bound to it; these insertion points enable silos to direct their processed data to specific silos.

The flow aggregation mechanism should allow for the implementation of different aggregation policies (like round-robin, priority queuing, etc.) based on the requirements of the flow. It should also allow dynamic variation in the number of subflows aggregated into (segregated from) a flow. To address both of these requirements, we introduce *Connection Point Aware Services*, which implement the desired policy and control the number of constituent subflows of a flow. They are placed as the last or the first service in their silos, and they are responsible for management of the connection point and associated silos. These services identify silos operating on subflows based on their insertion points. The insertion points are used to multiplex data into subflows and vary the number of subflows associated with a flow.

Figure 3.3 shows how flow aggregation mechanisms eliminates the duplication of services which were not requested by the applications, as shown in Figure 3.2. Both applications get their customized silos with the set of services they requested. All services performing network interface specific processing, such as Ethernet Framing and CSMA/CD, are placed in a silo, which aggregates the application flows. All three silos shown in Figure 3.3, are bound to each other by a connection point. The flow mixing service in the systems silo is a connection point aware service and combines the application flows based on the policy it implements.

3.3 Prototype and Implementation of Virtual Concatenation

We now describe the prototype which was built to demonstrate the capabilities of the SILO architecture and the extensions we propose in this report.

3.3.1 Prototype Modifications for Virtual Concatenation

The updated SILO prototype currently implements the extensions mentioned in Section 3.2. Interactions between an application and its flow are limited to modifying flow parameters; to accommodate these interactions, the silo services associated with the flow need to expose relevant control information as knobs. The relevant tuning algorithms handle the application requests by tuning these knobs. We have extended the SILO API to accommodate flow parameters, enabling the application to interact with the relevant tuning algorithm which handles the application requests. In addition to this, an application can request that its flow have certain parameters along with the services it requests at the time of silo creation.

Since modification requests to an application's flow are handled only by the concerned tuning algorithms, the SMA does not modify these requests, instead passing them onto the tuning algorithm using a function interface defined by the SMA and implemented by the tuning algorithm. To satisfy certain requests, the tuning algorithm may request additional resources from the SMA. The SMA may accept or deny this request based on the policy it follows and the amount of resources used by the application flow. This prevents applications from abusing the system. For instance, the SMA limits the number of subflows an application flow can be split into (or merged from), since‘ each flow and subflow has to be operated upon by an individual silo. Each silo is considered to be a resource, since it needs to be maintained by the SMA.

Flow aggregation mechanisms are implemented using connection points and connection point aware services. Connection points allow two or more silos to bind to them and operate on a flow, while connection point aware services are used to control the silos that are bound to the connection point. Internally, each connection point has an insertion point for each silo bound to it and one queue to maintain all the data passing through it. Connection points expose and implement an interface which can be used by connection point aware services to control the connection point.

Each connection point is controlled by a silo, typically operating on the flow which splits into or merges from its constituent subflows; this is done using a connection point aware service to regulate the number of constituent subflows. The SMA notifies the connection point about its controlling silo and notifies the services within the controlling silo about the connection point and the identifier of the controlling silo. The connection point aware services use the silo identifier while interacting with the connection point for security purposes. These connection point aware services primarily use the insertion points exposed by the connection point interface to identify the subflows and their associated silos.

The connection point aware services, based on the policy they implement, can use the connection point interface to multiplex data into the subflows. These services may also request the addition or deletion of silos from the SMA to vary the number of constituent subflows servicing the flow. The SMA may service this request differently, depending on the number of silos already associated with the relevant connection point. The SMA uses the same recipe for all silos operating over subflows, while using a different silo recipe for aggregated flows. This enables the SMA to duplicate silos operating over subflows whenever more subflows are requested by the connection point aware service. Whenever the number of subflows has to be decreased, the connection point aware service has to ensure that no data flows through them since the SMA deletes the relevant silos immediately.

3.3.2 Implementation of Virtual Concatenation

To incorporate the SONET VCAT/LCAS functionality, we implemented a new service and a new tuning algorithm which enable applications to request VCAT/LCAS functionality at silo creation time. Figure 3.4 provides an overview of the implementation details of VCAT/LCAS functionality.

The VCAT functionality splits a high bandwidth application flow into medium bandwidth virtual circuit capacity flows, routed to the destination using virtual tributaries belonging to a VCAT group. Thus, VCAT operates on the data path of the flow and is implemented as a service. The VCAT service is placed as the last service within the application defined silo and is a connection point aware service. It multiplexes the high bandwidth application flow in a round-robin manner into data flows operated upon by silos which provide virtual tributary processing services. Each VCAT service is initialized with the group identifier field (GID) at silo creation time by the Services and SILO Manager based on the number of pre-existing VCAT

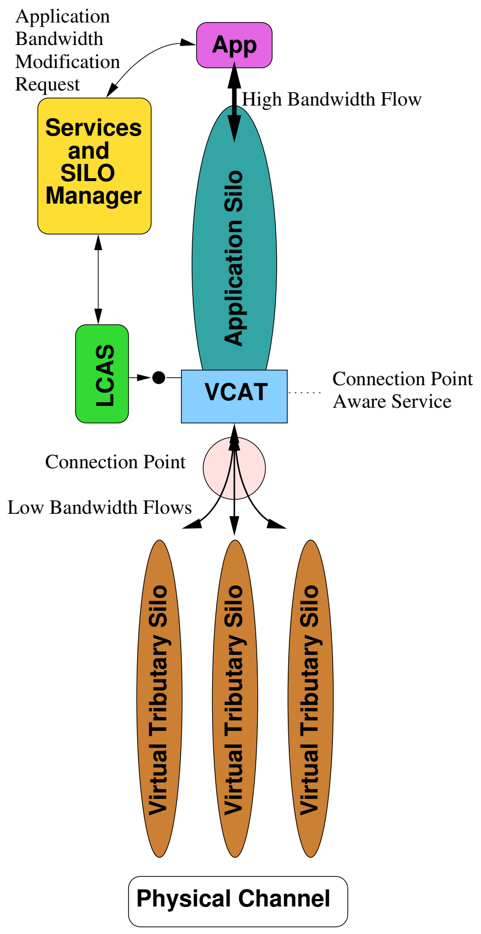


Figure 3.4: Virtual Concatenation Implementation

groups in the system. The initial values of Multi-Frame Indicator (MFI) start from zero, and the Sequence Number's (SQ) range from zero to one less than the VCAT group size.

The LCAS protocol varies the number of virtual tributaries used in the VCAT group based on dynamic application requests. LCAS thus operates on the control path of the flow and is implemented as a tuning algorithm. All application requests to vary the bandwidth are passed on to the Service and SILO Manager, which in turn invokes the relevant LCAS tuning algorithm. LCAS converts these bandwidth requests into the number of virtual tributary silos that need to be added or deleted; the VCAT service takes appropriate action based on this information. In addition to this, LCAS also takes care of network faults affecting the virtual tributaries using signaling messages to temporarily deactivate the silos of the failed virtual tributaries.

For LCAS to operate in conjunction with VCAT, the LCAS control signals, CTRL, MST, and RS-Ack, need to be added by VCAT to the data flow. These control signals are exposed as

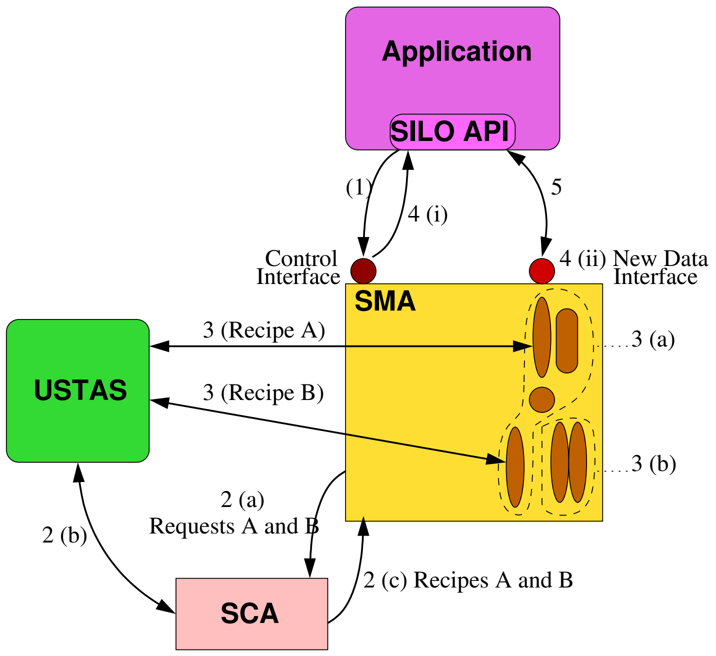


Figure 3.5: Prototype of Virtual Concatenation

knobs by the VCAT service, enabling the LCAS tuning algorithm to modify the VCAT service behavior to get the desired result. Since CTRL and MST signals are specified for each virtual tributary, these signals are exposed as special knobs called Vector Knobs. *Vector Knobs* are a set of related control variables in an array which can be exposed by services as knobs to enable a tuning algorithm to analyze all of them before modifying the service behavior. The LCAS tuning algorithm uses the CTRL and MST vector knobs to signal its peer to add or delete virtual tributary silos based on application requests and to activate or deactivate virtual tributary silos based on network faults and restorative actions.

3.3.3 Operation of Virtual Concatenation

Although the prototype for SILO extensions does not introduce any new software modules, there are significant modifications within the SMA for handling application requests involving flow modification and aggregation. Figure 3.5 shows the interactions of the SMA with various SILO components, using Virtual Concatenation as an example, along with the steps enumerated below.

1. The SILO Enabled Application requesting VCAT service needs to mention the amount of bandwidth it requires when requesting a silo from SMA. It can request the SMA to include some services in virtual tributary silos (Figure 3.4), in addition to configuring the

silo operating over the high bandwidth flow.

2. VCAT service requests are handled by the SMA in the following manner:

   (a) Upon receiving the application request, the SMA splits this request into separate requests for high bandwidth (Request A) and medium bandwidth flows (Request B) to be processed by the SCA.

   (b) The SCA consults the USTAS to process these requests based on the composability constraints and cross services tuning algorithms needed.

   (c) The SCA provides silo recipes (Recipes A and B) for both flows without any knowledge of the association between the recipes. The VCAT service and LCAS tuning algorithm are placed in Recipe A.

3. The SMA then creates two silos, Silo A for the high bandwidth flow, using Recipe A, and Silo B for the medium bandwidth flow, using Recipe B.

   (a) The LCAS tuning algorithm and VCAT service are placed in Silo A, which is connected to Silo B using a Connection Point.

   (b) The SMA informs the LCAS algorithm about the application's bandwidth requirement, and the latter replicates Silo B to satisfy the application's bandwidth requirements using the VCAT service.

4. The application receives a silo handle and a new data interface is associated with the set of silos that were created to handle its flow.

5. The application uses the silo handle and data interface to communicate with its peer.

6. Whenever the application releases the silo, the SMA destroys the relevant silos associated with the Connection Point used to handle the application flow.

7. All bandwidth modification requests from the application are forwarded to the LCAS algorithm using the control interface of the SMA. The LCAS tuning algorithm requests the SMA to add or delete silos based on the bandwidth modification requests from the application.

The process listed above does not take into account failure scenarios; the application is given a failure notification if the SCA is not able to construct both recipes (Recipes A and B) in step 2. Bandwidth modification requests are not handled if the application requests either, elimination of all virtual tributary silos or the addition of virtual tributary silos in excess of the limit set by the SMA.

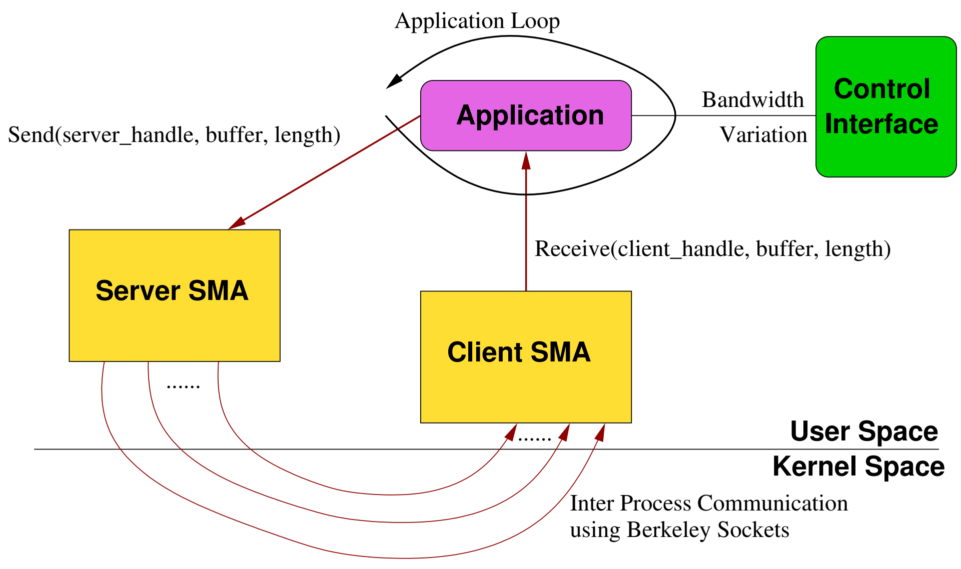


Figure 3.6: Virtual Concatenation Demo Application

3.4 Demonstration and Experimentation of Virtual Concatenation

We demonstrate the VCAT/LCAS functionality implemented in the SILO architecture using client-server configuration on the same host. The client-server configuration of SILO architecture requires that one server SMA and one client SMA be started with different control interfaces; two SCAs are also needed, one for each SMA. This prototype uses UNIX sockets for communication between client and server SMAs.

3.4.1 Experimental Setup

To vary the flow's bandwidth, the sender and the receiver need to increase or decrease the bandwidth in coordination with each other. This coordination is achieved by using one application for both the sender and receiver sides. As shown in Figure 3.6, the application connects to both SMAs, the server and the client, and obtains silo handles for sending and receiving data; this enables it to vary the bandwidth at the sender and receiver sides simultaneously. The application exposes a control terminal to read in the data, varying the application bandwidth as desired. All three tasks (that is, reading bandwidth modification requests from the control interface, sending data to the server SMA, and receiving data from the client SMA) are performed by the application sequentially, in a loop.

We also implemented a logging service, which records the data of all packets processed by a silo into a log file. This service is placed in the virtual tributary silos operating over the medium bandwidth flows, which can be added or removed by the LCAS tuning algorithm according to the variation in application bandwidth requirements. Thus, we can verify the variation in flow bandwidth in real-time using the UNIX 'tail' command or by analyzing the log file after data transmission. Since the logging service is placed in silos operating over the medium bandwidth flows, it also records the VCAT/LCAS control packets that are exchanged between the peers, enabling verification of the LCAS protocol implementation.

3.4.2 Experiments and Validation

To verify the operation of the VCAT/LCAS implementation, we analyzed the data in the log files generated by the logging service placed in the virtual tributary silos. We used the application described above to send equal-sized data packets containing a sequence number specified by the application. The log files of the virtual tributary silos operating over the medium bandwidth virtual tributaries (VT 0 to VT 5) contain two pieces of information, a packet number for each data packet flowing through the silo (Flow Packet Number) and the sequence number of the packet as specified by the application (Application Sequence Number). Figures 3.7, 3.8, and 3.9 plot the Flow Packet Number versus the Application Sequence Number to help us understand the behavior of the VCAT/LCAS implementation.

Handling the Application Data Flow

Figure 3.7 demonstrates how the Virtual Concatenation service handles the application data packets; this experiment was run with four virtual tributaries (VT 0 to VT 3) processing application data flow.

There are two things to note in Figure 3.7. First, in each virtual tributary, the Application Sequence Number increases by four whenever the Flow Packet Number increases by one; this implies that each virtual tributary processes one in every four data packets. Second, for a specific Flow Packet Number, the Application Sequence Number increases across the set of virtual tributaries; this implies that the ordering of virtual tributaries which process the application packets is predetermined. Both these inferences help us verify the round robin policy used by the VCAT service when multiplexing the application flow into the virtual tributaries.

Handling Variation in Bandwidth Requirements

Figure 3.8 demonstrates the effect of varying the number of virtual tributaries operating on the application flow. Similar to the previous experiment, four virtual tributaries (VT 0 to VT 3) process the application flow. The VCAT/LCAS functionality operates in the same manner until the Application Sequence Number reaches forty.

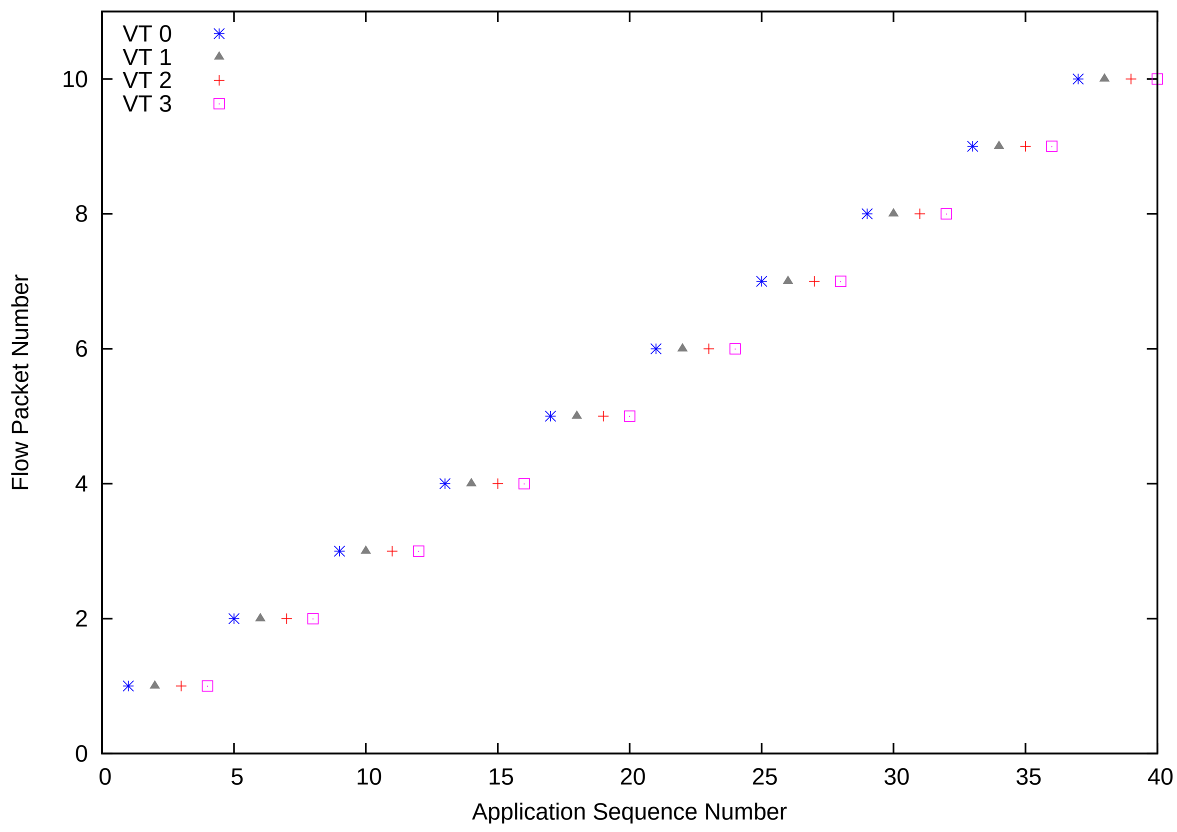


Figure 3.7: Handling Application Flow

When the Application Sequence Number reaches forty, the number of virtual tributaries is increased to six, resulting in the creation of two new virtual tributaries (VT 4 and VT 5). The LCAS algorithm handles the application request and directs the VCAT service to add two silos to the associated connection point. Again, there are two things to observe here. First, the Application Sequence Number increases by six whenever the Flow Packet Number increases by one. This implies that one in six data packets are processed by each virtual tributary. Second, once the virtual tributaries VT 4 and VT 5 become active and start processing data packets, the curves VT 4 and VT 5 are placed lower than others since they were created after increasing the number of available virtual tributaries. The curves of VT 4 and VT 5 are parallel to the curves of VT 0 to VT 3 when VT 4 and VT 5 are active. This implies that the newly created virtual tributaries also process one in six data packets. These observations lead us to infer that the VCAT/LCAS functionality has successfully scaled to accommodate the increase in available virtual tributaries for processing the application flow.

Conversely, when the Application Sequence Number is eighty, the number of virtual tributaries is decreased to two, resulting in the deletion of virtual tributaries VT 2 to VT 5. As a

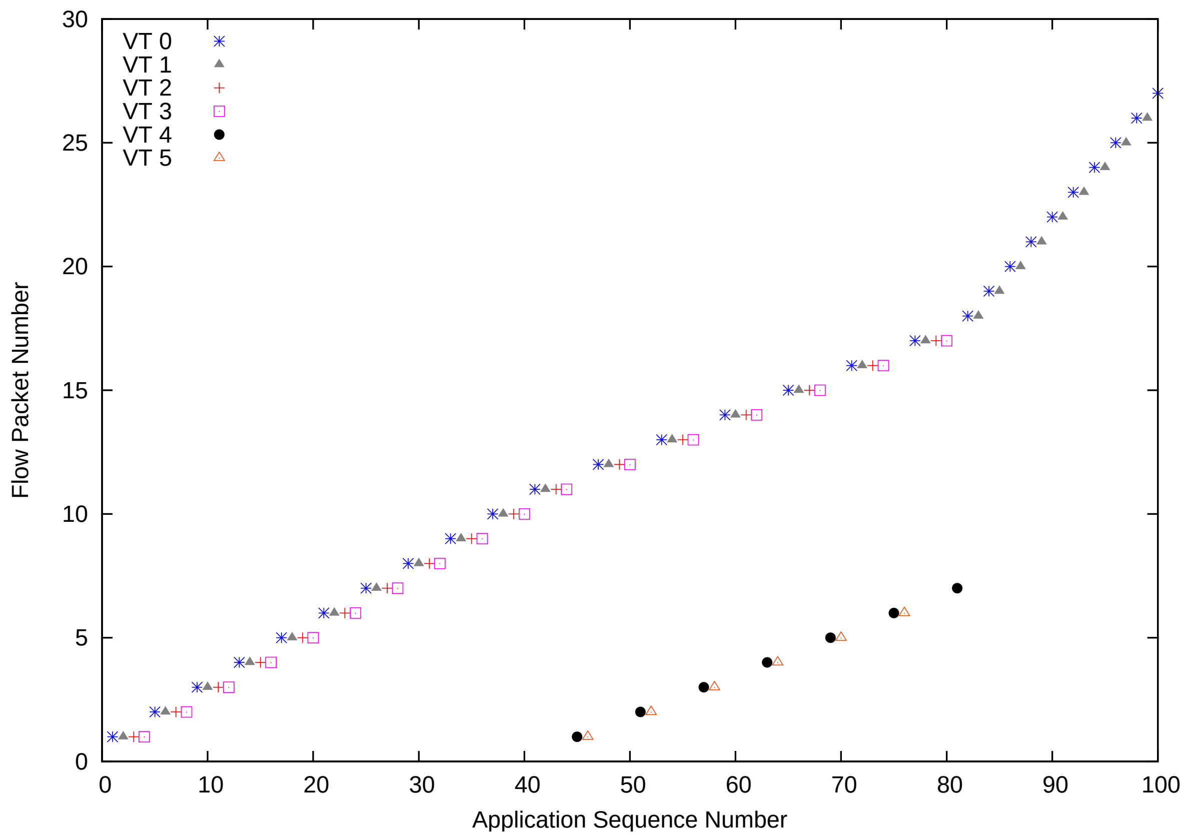


Figure 3.8: Handling Application Bandwidth Modification Requests

result, the LCAS algorithm deletes four silos from the system, leaving only silos for VT 0 and VT 1 to service the application flow. There are two things to observe here. First, the curves for virtual tributaries VT 2 to VT 5 do not progress after eighty, verifying that these virtual tributaries no longer process the application flow. Second, the Application Sequence Number increases by two whenever the Flow Packet Number increases by one; this implies that each virtual tributary processes one in two packets from the flow. Again these observations help us infer that the VCAT/LCAS functionality scaled successfully to accommodate the reduction in the number of available virtual tributaries.

Handling Network Faults

Figure 3.9 demonstrates how the VCAT/LCAS functionality handles network faults. For this experiment, we used three virtual tributaries and simulated a network fault. To simulate the network fault, the receiver transmits a member status (MST) failure notification on VT 0 to the sender when the Application Sequence Number reaches eighteen. The LCAS algorithm in

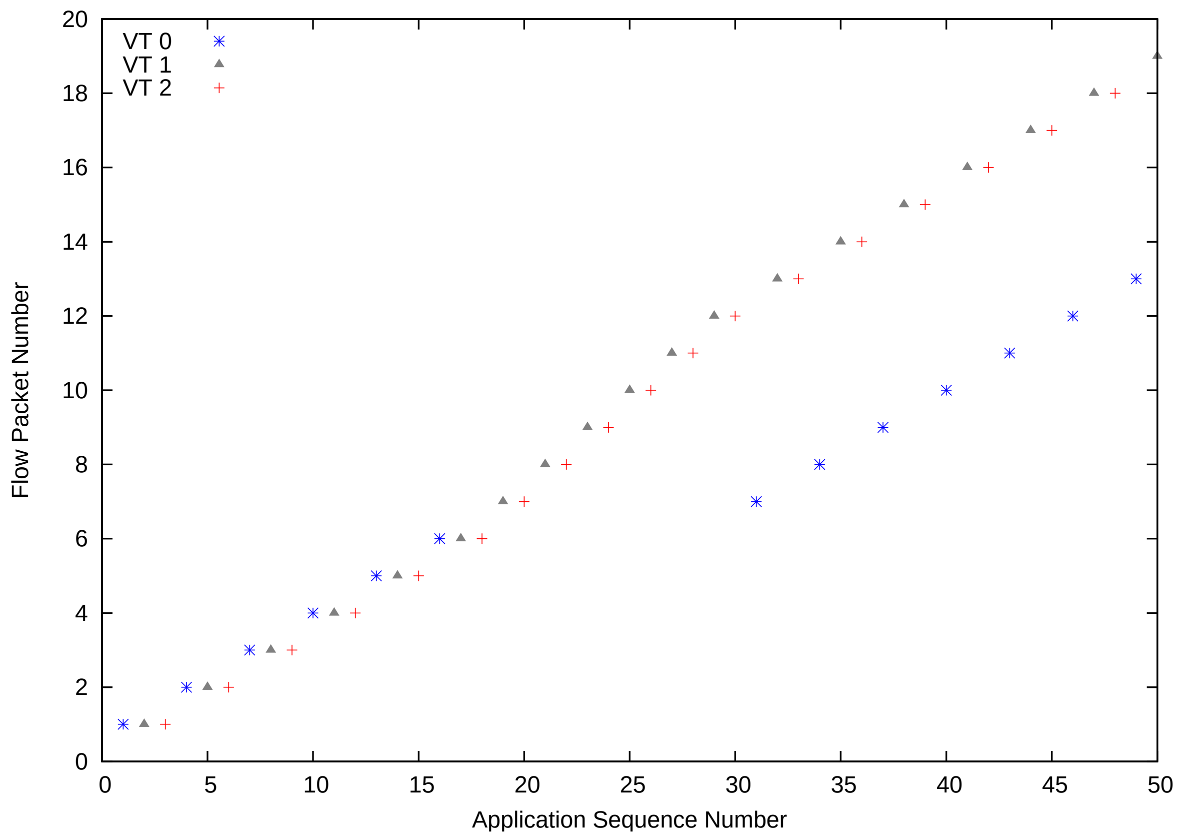


Figure 3.9: Handling Network Faults

the sender marks VT 0 as inactive until it receives a member status recovery notification from the receiver. The member status recovery notification is transmitted by the receiver when the Application Sequence Number is thirty.

The network fault results in two things. First, data flow through VT 0 stops, and second, VT 1 and VT 2 handle the entire application data flow since they are the only active virtual tributaries in the system. As a result, the Application Sequence Number increases by two instead of three, for an increase of one in the Flow Packet Number in VT 1 and VT 2. Since VT 0 is now inactive, the Flow Packet Number of VT 0 remains until it is reactivated.

The network recovery notification re-establishes communication on VT 0. As a result, data resumes flowing through VT 0, and the Flow Packet Number starts to increase again. However, the Application Sequence Number is much higher due to the data transfer performed using VT 1 and VT 2 while VT 0 was inactive. This is shown by the significant interval between the points signifying when the network fault occurred and recovered, on VT 0's curve. In addition to this, since one more virtual tributary is available, the Application Sequence Number increases by three for each increase in Flow Packet Number, and data is switched between the three available virtual tributaries.

These experiments verify the implementation of VCAT/LCAS functionality within the SILO Architecture. The results obtained from these experiments help us understand three things. First, this VCAT/LCAS implementation handles splitting of the high bandwidth application flow into various medium bandwidth flows. Second, it scales well to modification in the bandwidth requirements of applications. Last, it handles network faults in a robust manner, resulting in a graceful degradation of the service levels provided to applications.

3.5 Summary

In this chapter, we reported on the implementations of extensions to the SILO architecture that enable it to support flow aggregation mechanisms and handle application requests for modification of its flow characteristics. Flow aggregation is performed by introducing Connection Points to which silos can bind to, and Connection Point Aware Services which control the degree of flow aggregation and the multiplexing policy for packets using the flow and its subflows.

Application requests are handled by tuning algorithms, services which expose flow parameters as knobs, and the Services and SILO Manager. The tuning algorithm interprets abstract application requests and tunes the relevant service knobs to control flow parameters. Whenever additional network resources are requested by the application, the tuning algorithm and the Services and SILO Manager service the request based on the amount of resources being used by the application flow, the amount of available resources, and the global, system wide policy used by the Services and SILO Manager to allocate resources.

The SONET Virtual Concatenation functionality, along with the Link Capacity Adjustment Scheme, use these extensions to aggregate medium bandwidth virtual tributaries into a higher bandwidth application flow and to dynamically modify the amount of bandwidth used by the application. These extensions and SONET VCAT/LCAS functionality within the SILO architecture were demonstrated using a proof-of-concept, prototype implementation. We also validated the behavior of VCAT/LCAS functionality in this prototype using a logging service and a customized application to verify the correctness of their implementation.

Chapter 4

Bandwidth on Demand Service - Traffic Grooming

4.1 Introduction

There are a number of applications, such as audio and video streaming, which require their flows to have low jitter though their bandwidth consumption is as low as 300 kbps [95]. Such applications can get very low to no jitter from the network by reserving bandwidth for their traffic. To host such applications within the Cloud, cloud providers need to provide them the flexibility to request resources of low granularity. It is possible to satisfy such requests for low bandwidth by allocating a dedicated virtual circuit; however, this comes at the cost of wasting the residual bandwidth capacity in the virtual circuit.

Wasting residual bandwidth capacity is not preferred, since the excess may be used to carry other application flows within the virtual circuit. This inefficiency can be overcome by Traffic Grooming. *Traffic Grooming* is the process of grouping multiple low bandwidth flows into larger units that can be processed as single entities by networking devices. Typically, low bandwidth flows are grouped or groomed into a higher bandwidth virtual circuit; intermediate network nodes switch virtual circuits without differentiating between the subflows they contain.

In this chapter, we provide mechanisms for grooming multiple application flows at the source based on their varied bandwidth requirements. Application flows, bound to the same destination, are groomed on-the-fly into a higher bandwidth virtual circuit. The extensions to the SILO architecture, described in Chapter 3, enable cloud providers to process application requests for high bandwidth. In this chapter, we extend the SILO architecture to *efficiently* accommodate low bandwidth flows over medium bandwidth virtual circuits, so as to support different applications requesting bandwidth of varying granularities.

Using the application request mechanism and flow aggregation features, this can be achieved provided we can (**1**) uniquely identify each application flow while it is being transferred over the virtual circuit, (**2**) provide mechanisms to accommodate the bandwidth requirements of applications, and (**3**) conserve the network resources used to satisfy such low bandwidth requests. The extensions to support these features in the SILO architecture are discussed in Section 4.2. The implementation details of the SILO prototype are discussed in Section 4.3. Section 4.4 describes the experimental setup, verification, and validation of end systems grooming within SILO, and a summary of this work is given in Section 4.5. In Section 2.1.4, we discussed the prior research related to the work described in this chapter.

4.2 Extensions for Traffic Grooming

Since traffic grooming requires the multiplexing of application flows into a unified flow, we need mechanisms to (**1**) accommodate low bandwidth flows within the available bandwidth virtual circuit whenever an application request arrives and (**2**) multiplex data from different flows into the virtual circuit, in a bandwidth proportionate manner and at specified time intervals. In this section, we describe the extensions to the SILO architecture to perform these tasks. Extensions discussed in Section 4.2.1 enable low bandwidth flows to be accommodated whenever application requests arrive. Those in Sections 4.2.2 map flows to source and destination applications, while those in Section 4.2.3 satisfy the bandwidth requirements of application flows based on the bandwidth capacity of virtual circuits.

4.2.1 Network Interface Silos

Network Interface Cards (NICs) process multiple flows, each of which use the NIC for data transmission. To enable faster processing, each NIC offers features like hardware buffers, checksum offloading, and so on; such features can be exploited by implementing NIC specific services which process flows using these features. If each flow using the NIC were to have its own silo which duplicates NIC specific services, it raises synchronization issues due to the sharing of NIC resources among different instances of the NIC services.

Synchronization issues are known to cause performance degradation and can be avoided by creating an *Interface Silo*, which processes all data flows using a NIC. This silo will contain all the NIC specific services; application silos will offload NIC specific processing to this silo. Since NIC specific services will not be duplicated, the need for synchronization mechanisms is eliminated, avoiding the corresponding drop in performance.

The interface silo needs to merge the application flows, using the relevant NIC, into a unified flow before performing NIC specific processing on the unified flow. Merging of flows is performed using a connection point and a connection point aware service. All application silos use the connection point to transfer data to the interface silo. The manner in which application flows

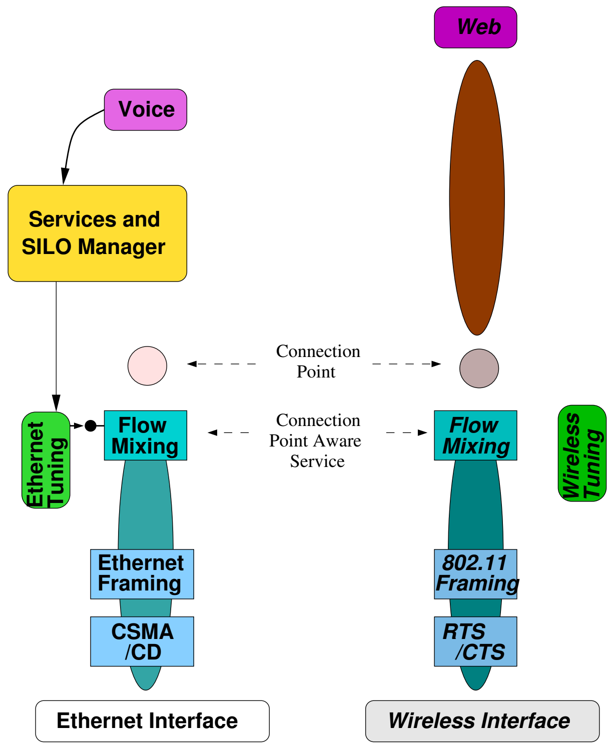


Figure 4.1: Network Interface Silo

are multiplexed depends on the flow aggregation policy used by the connection point aware service.

In the original SILO architecture, the lifetime of a silo is dependent on the data flow it processes. When the application data flow ceases to exist, the associated silos are destroyed, as they have outlived their utility. However, the interface silo operates on the multiple flows that use the NIC. The NIC remains active and operates on flows even after some of the flows are destroyed, since the data flows yet to be created in the future have to be processed. Since the interface silo has to process multiple flows which start and end at different instances of time, its lifetime is determined by the time duration the NIC remains active and not by the lifetime of the flows it is currently processing.

Since multiple flows will use the NIC at various points in time, applications need a mechanism to associate and disassociate with the relevant interface silo. We use the tuning algorithm to enable the dynamic addition or deletion of flows to the group of those operated on by the interface silo.

Figure 4.1 shows the operation of interface silos over Ethernet and Wireless Network Interface Cards. An Ethernet interface is associated with a silo composed of Ethernet specific services like Ethernet Framing, Checksum Calculation, and CSMA/CD. The wireless interface is associated with a silo composed of services like 802.11 Framing, Checksum Calculation, and Rate Adaption, amongst others. Flow Mixing is a connection point aware service common to both interface silos and performs the task of flow aggregation using a single queue in the connection point. An application silo can bind to the relevant interface silo after being added by the tuning algorithm, Ethernet or Wireless Tuning in this case, using the associated connection point. In Figure 4.1, a voice application request to use the Ethernet interface is processed by the associated tuning algorithm, while the Web browser's data is processed by the silo connected to the Wireless Interface silo using the connection point.

4.2.2 Identification and Routing of Application Flows

Whenever an application flow operates at network speed, it directly uses a virtual circuit without interacting with other flows; its flow can be identified by determining the virtual circuit it uses. This is applicable even when a high bandwidth flow uses multiple virtual circuits, provided that the network architecture knows the set of virtual circuits used by the flow. However, when a low bandwidth flow is merged with other such flows, it loses its identity, since multiple flows now share a virtual circuit. Each low bandwidth flow needs to maintain its identity within a virtual circuit, since it needs to be directed to the appropriate application at its destination.

In the current Internet architecture, each application specifies its remote peer by specifying (**1**) the Internet Protocol (IP) address of the machine on which it is hosted, and (**2**) the port number used exclusively by the peer to listen for connections. The tuple of IP address and port number is a globally unique *Endpoint*, since IP addresses are unique across all networks and port numbers are unique within each host, each endpoint can uniquely identify an application. Each application at connection setup mentions (**1**) the destination endpoint to identify the peer application to which its data flow is destined, and (**2**) the source endpoint to help the peer identify the source application of the data flow it received.

The combination of source and destination endpoints can uniquely identify each low bandwidth flow within the virtual circuit, since they uniquely identify the source and destination applications of the flow. At the sender, the application data is enclosed within a data frame; the frame header contains the source and destination endpoints. At the receiver, the relevant application can be identified by inspecting the frame header; this header should be stripped off at the receiver before handing the data over to the appropriate application.

Identifying the Relevant Network Interface

Each host can have multiple interfaces through which it connects to numerous other hosts on the network. When requesting a connection, the application does not specify the interface to use but instead, specifies the destination endpoint; hence, the appropriate network interface

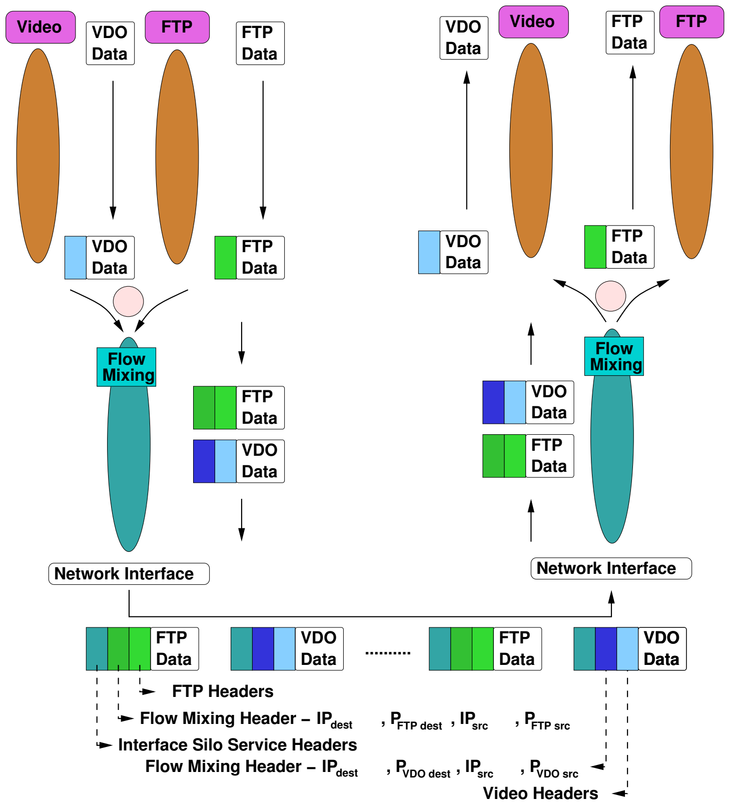


Figure 4.2: Identifying Application Flows

needs to be identified to satisfy the application's request.

The relevant network interface can be identified using a forwarding table, the destination IP address, and network administration policies. Whenever an application request arrives, the relevant interface silo is determined by the forwarding mechanism; the application silo is then created and bound to the connection point after the application flow is accommodated in the lightpath by the tuning algorithm of the chosen interface silo.

Figure 4.2 shows the mechanism for identifying the application data flows while using traffic grooming. The FTP source client with the endpoint (IP_{src}, P_{FTPsrc}) specifies the FTP destination server to which it wants to connect using the destination endpoint (IP_{dest}, $P_{FTPdest}$). Similarly, a Video source with endpoint (IP_{src}, P_{VDOsrc}) connects to its peer with destination endpoint (IP_{dest}, $P_{VDOdest}$). The Services and SILO Managers on hosts IP_{src} and IP_{dest}, create the FTP and Video application silos and connect them to the appropriate interface silos, since both flows use the same source-destination pair.

At IP_{src}, the services within the FTP and Video silos add headers relevant to their flow processing; these services are unaware that the data flow is groomed. The grooming service in the interface silo adds the relevant source and destination endpoint information to the application data flows. The interface silo may introduce additional headers based on the data frames.

At IP_{dest}, the frame headers are stripped off by the interface silo. The grooming service strips off the added endpoints and directs the FTP data flow to the relevant silo based on the destination endpoint (IP_{dest}, $P_{FTPdest}$) mentioned in the data packets. Similarly, the video flow is directed to the relevant video application silo. Both applications ultimately receive their data for further processing without accessing any communication headers that might have been added by any of the services which processed the data flow.

4.2.3 Scheduling Execution of Interface Silos

Applications using bandwidth reservations are unaware of the underlying network link speed. Since the network interfaces can only guarantee bandwidth at a certain granularity, support for bandwidth demands lower than this granularity will need to be provided by the architecture. While grooming low bandwidth application flows to conserve network resources, the SILO architecture must be able to efficiently use the link bandwidth.

The interface silo needs to process sufficient data to operate the NIC at link speed while providing the bandwidth promised to each application flow. If the link speed is R bytes per second, the NIC should transmit R bytes of data within one second to operate efficiently. If the interface silo can process R bytes of data every second, the NIC should be able to utilize all available link bandwidth. If the interface silo processes data in excess of R bytes, the NIC will be overwhelmed; hence, the interface silo should process not more than R bytes per second. To guarantee bandwidth of r_i bytes per second to an application flow f_i, the interface silo should process r_i bytes of data from f_i every second. The interface silo has to ensure that its obligation to both the NIC and all M application flows are met. It also should not process application data in excess of the amount of data that can be handled by the NIC. In other words, the interface silo should satisfy the constraint $\sum_{i=0}^{M} r_i \leq R$.

The present SILO architecture invokes relevant silos as soon as there is unprocessed data available, either from the application or from the underlying networking interface. The interface silo can ensure that the bandwidth promises to applications are met only if it is able to keep track of the time and amount of data processed for each flow. The architecture can aid the interface silo in tracking time by invoking it after a periodic time interval. The interface silo can process the right amount of data from each application flow based on its knowledge of the bandwidth requirements of each application flow and the time elapsed since its last invocation. The connection point aware service requests that the Service and SILO Manager schedule the associated interface silo at periodic time intervals; the Service and SILO Manager is responsible for ensuring that no more than one outstanding request per interface silo exists at any given time.

4.3 Prototype and Implementation of Traffic Grooming

4.3.1 Prototype Modifications for Traffic Grooming

The prototype for Traffic Grooming (Grooming) augments the prototype built for Virtual Concatenation (VCAT). It reuses the extensions made for VCAT, namely Flow Aggregation and Application Flow Interactions. Each application can request the SILO architecture to ensure that its data flow maintains certain characteristics. For example, the application can request that the SILO architecture ensure that its data flow be given certain amount of bandwidth; this feature introduced in VCAT, is reused by Grooming. The SILO architecture determines whether to use Grooming or VCAT mechanisms to service the application request. If the amount of bandwidth requested is less than the bandwidth capacity of the underlying network link, the SILO architecture uses Grooming, otherwise, it uses VCAT.

Grooming also uses flow aggregation to groom different application flows into one unified flow requiring bandwidth sufficient to be carried over an entire virtual circuit. A connection point and its associated connection point aware service are used to assist in grooming. Unlike VCAT, the connection point aware service cannot replicate or delete silos while grooming, in order to prevent it from deleting or duplicating application silos.

The connection point module was introduced for VCAT with a single queue to handle all the data going through it. The connection point aware service will be unable to groom application data flows with different bandwidth requirements using a single queue. Hence, the connection point has been extended to accommodate one queue per bound silo. Presently, the connection point can be created in single queue or multiple queue mode.

The application silos can enqueue data packets at the connection point after processing them. The connection point aware service can then dequeue these data packets at its convenience to perform further processing. Each queue has an upper bound on the amount of data that it can contain in order to restrict the size of the buffer used by each application. If the application data overflows, the data is dropped from the queue, following a tail-drop queue management policy.

SILO Prototype on Multiple Machines

The SILO prototype consists of an SMA, an SCA, and the SILO enabled application. The network architecture of each machine consists of one instance of the SMA and the SCA; multiple SILO applications interact with the SMA to establish a communication mechanism with peer applications interacting with different instances of the SMA. In the SILO prototype described in Sections 2.1.3 and 3.3.1, the SMA is able to interact with the other SMA instances only if they are placed on the same machine.

Each SMA exposes a well known UNIX pipe for other SMAs to use; whenever an SMA needs to communicate with another SMA, the source SMA writes data into the the relevant UNIX pipe of the destination SMA, which processes the data after it has been read. This limitation is undesirable, since it does not reflect the real world scenario in which applications hosted on different machines communicate with each other.

We have modified the SILO prototype to enable communication between SILO enabled applications hosted on different physical machines. The SMA instances communicate with each other using UNIX Berkeley sockets. Every SMA instance listens on a well known UNIX port to receive messages from other SMA instances; hence, only one SMA instance can operate per machine. An SMA instance directs messages to other SMA instances using the IP address of the machine hosting the destination SMA and the UNIX port used by SMA instances; SMAs communicate over UDP/IP.

At connection setup, each application is required to state the destination IP address identifying the machine which hosts the peer application in addition to the local and remote silo ports. The source SMA attaches a header detailing the local and remote silo ports to the application data flow; the remote silo port enables the destination SMA instance to identify the relevant application to which the incoming flow is destined, while the local silo port identifies the application from which the flow originated.

4.3.2 Implementation of Traffic Grooming

Figure 4.3 shows the implementation of Traffic Grooming within the SILO architecture, which introduces a new service and a tuning algorithm. Application flows are groomed by the interface silos if their bandwidth requirement is less than the bandwidth granularity of the underlying network. The application is thus unaware that its data flows are being processed by the Traffic Grooming entities.

Grooming as a Service

Grooming of different data flows into a unified flow needs to performed in the data plane. Services within the SILO architecture operate in the data plane, so we introduce *Grooming* as a silo service. This grooming service is placed as the topmost service in the interface silo and is a connection point aware service.

The Grooming service schedules data placed in the different queues within the connection point based on the link scheduling discipline it implements. Priority scheduling [94], round robin [38], and weighted fair queuing [83], amongst others, are the link scheduling mechanisms

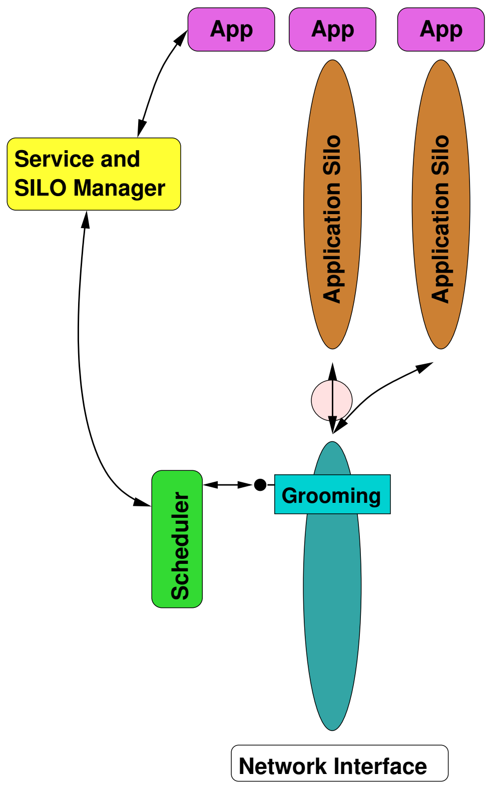


Figure 4.3: Traffic Grooming Implementation

that may be implemented. The Grooming service adds a frame header to each flow's data, uniquely identifying the flow. This service is also responsible for periodically processing the application data flows, so it requests the Service and SILO Manager to schedule the interface silo at periodic intervals.

This implementation performs packet switching [76], which multiplexes packets of varying length at periodic intervals. Bandwidth requirements of different application flows are roughly met if the constituent packets are of same or similar sizes. An adaptation service can service all application data flows being groomed to ensure data packets are of uniform size. The uniformity of the packet size provides a measure of the amount of data processed (in bytes) by the Grooming service, based on the number of packets processed. Limiting the number of packets processed by the Grooming service every time it is invoked prevents it from overwhelming the NIC.

At the receiving side, the Grooming service inspects the destination endpoint from the frame's header to determine which application silo is the destination of the data. The service directs the data to the appropriate application silo using the associated insertion point maintained by the connection point.

Scheduler as a Tuning Algorithm

Determining whether an application flow can be allowed access to a virtual circuit is a control function; the tuning algorithm can process application requests to use a virtual circuit since it operates on the control aspects of the unified flow. Each application request contains the source and destination endpoints, which are used to determine the relevant interface silo.

The tuning algorithm can include additional application flows into the virtual circuit if the number of packets processed by the Grooming service is less than the upper bound on the number of packets as determined by the link speed and packet size. An application's request to add its data flow is accepted if the amount of additional packets to be processed by the Grooming service can be accommodated within the residual capacity of the virtual circuit. The residual capacity of the virtual circuit is determined by a positive difference in the maximum number of packets that the Grooming service can process and the number of packets it is currently processing. The number of additional packets the Grooming service needs to process to service the flow is determined by the bandwidth requested by the application.

Deletion of an application flow is trivial, since it involves freeing a network resource. Since the tuning algorithm determines the placement of the application flow it is called the *Scheduler* tuning algorithm. It can employ First Fit, Best Fit, or Next Fit mechanisms, depending on the request turnaround time, flow fragmentation policy, or other needs.

The Scheduler tuning algorithm uses a Vector Knob called *Slot Distribution* to inform the Grooming service about the placement of data flows in their respective slots. In addition to this, the Scheduler informs the Grooming service about any variations in the number of considered flows and the associated source and destination endpoints. Whenever the interface silo is scheduled, the Grooming service dequeues data from the queues according to the order maintained by the Slot Distribution vector.

4.3.3 Operation of Traffic Grooming

The prototype for the Traffic Grooming extensions does not introduce any new modules, but similar to Virtual Concatenation, there are significant modifications within the SMA which involve creating interface silos, scheduling silos using services, and identifying remote application endpoints over groomed flows. Figure 4.4 shows the interactions of various SILO components using Traffic Grooming with the steps enumerated below:

1. As soon as the SMA is started, it creates an interface silo for each active network interface that is reserved for Traffic Grooming, using the following steps:

   (a) The SMA forms a request for each interface silo to be created based on the configuration parameters of the relevant network interfaces. Each silo request contains a list

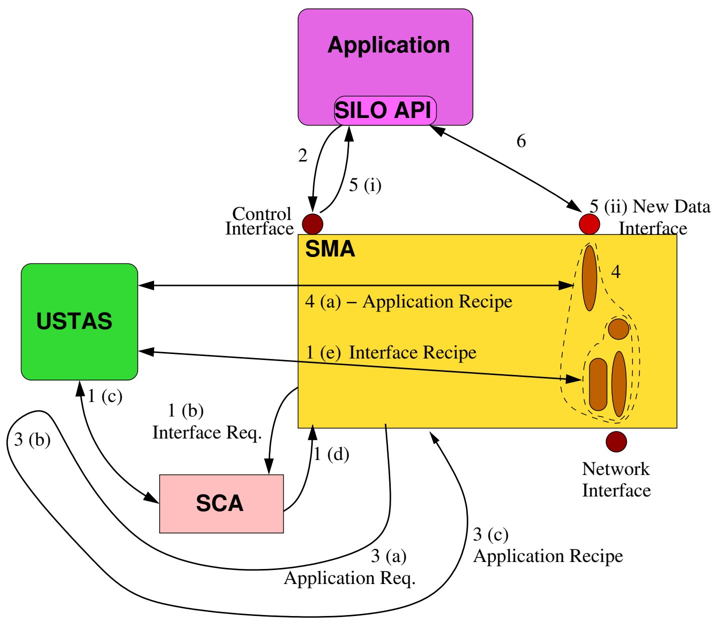


Figure 4.4: Prototype of Traffic Grooming Implementation

of services and tuning algorithms required to process data flows using the relevant network interface; all requests include the Grooming service and Scheduler tuning algorithm.

(b) Once the SCA is started and bound to the SMA, the SMA passes on the requests to the SCA for validity checks.

(c) The SCA checks the correctness of these requests based on the composability constraints maintained in the USTAS.

(d) If the request is valid, the SCA creates the necessary recipe for interface silo creation without any knowledge of the relationship between these silo recipes and the network interfaces.

(e) On receiving the interface silo recipes from the SCA, the SMA creates the interface silos using the object files of services and tuning algorithms. The SMA also creates a connection point that is controlled by the Grooming service; the SMA is now ready to groom different application flows into a unified flow.

2. The SILO Enabled Application requests bandwidth allocation for its data flows using the control interface of the SMA. If the amount of bandwidth requested is lower than the network speed, Grooming is used; otherwise, Virtual Concatenation is used. We look at Grooming requests in this chapter; handling of Virtual Concatenation requests was covered in Section 3.3.3.

3. After choosing Grooming as the appropriate mechanism, the application request is validated in a manner similar to the interface silo request.
    (a) The SMA passes the application silo request onto the SCA for validity checks.
    (b) The SCA consults the USTAS to check these requests for correctness based on the composability constraints and required cross services tuning algorithms.
    (c) If the application request is valid, the SCA provides application silo recipes to the SMA.
4. The SMA processes the application silo recipes with the following steps:
    (a) The SMA creates the application silo using the object files from USTAS.
    (b) Once the application silo is created, the SMA determines the appropriate interface silo to groom the newly created application flow based on the destination IP address, which is mentioned in the application request. The SMA maintains a forwarding table, which provides a mapping of interface silo identifiers to a list of reachable hosts using the relevant network interfaces.
    (c) The application request is then processed by the Scheduler algorithm associated with the chosen interface silo. The Scheduler algorithm determines whether it can accommodate the application flow and places the flow based on its bandwidth requirement.
    (d) Once the Scheduler accommodates the application flow, the related silo is bound to the interface silo using the connection point.
5. Once the application silo is bound to the relevant interface silo, the application receives a silo handle and data interface to communicate with its peer.
6. The application uses the silo handle for all further interactions with the SMA and uses the data interface to communicate with its peer.
7. Whenever the application releases the silo, the SMA destroys the application silo and requests that the appropriate Scheduler algorithm to free the bandwidth used by the application's flow.
8. The interface silo exists as long as its associated network interface is active.

The process listed above does not take into account failure scenarios; the SMA and SCA will terminate if creation of any of the interface silos fails. The application is given a failure notification under three scenarios: (**1**) if the application silo cannot be created, (**2**) the destination IP address is not known or is incorrect, and (**3**) if the application flow cannot be accommodated by the relevant interface silo in the unified flow.

4.4 Demonstration and Experimentation of Traffic Grooming

We demonstrate the Grooming functionality implemented in the SILO architecture using the peer-to-peer configuration on different hosts. The peer-to-peer configuration of the SILO architecture requires one SMA and SCA to be started on each host. The SMAs hosted on different hosts communicate using UDP sockets; the SCA and SMA on a given peer interact with each other through UNIX pipes.

4.4.1 Experimental Setup

For the purposes of these experiments, virtual circuits are considered to operate at a hundred units of bandwidth; the grooming service creates five slots of twenty bandwidth units to service application flows. The grooming service can accommodate up to five different application flows at a time, each flow occupying one service slot. The number of service slots occupied by each application flow can range from zero to five depending on its bandwidth requirements and the availability of slots.

The experiments require two SILO Enabled applications hosted on different machines and serviced by the SMA and SCA for each machine. Experiments consider a unidirectional flow to validate the functionality of Grooming; this requires one of the applications to send data while the other application receives data. The sender and receiver applications should request the same amount of bandwidth in order to transfer data between each other. The application flow is groomed with other flows only if its bandwidth requirements are less than one hundred bandwidth units.

The functionality of the grooming service has been extended to log information about all packets it processes. This enables us to verify the operation of the multiplexing scheme and to monitor variation in the number of application flows processed over a period of time. The log files are opened whenever an instance of the grooming service is created along with the relevant silo; these files are closed whenever the grooming service instance is destroyed.

4.4.2 Experiments and Validation

To verify the operation of the Grooming service and Scheduler tuning algorithm, we analyzed the data placed in the log files by the Grooming service. The log files essentially contain three pieces of information: (**1**) the packet number for each data packet flowing through the interface silo (Flow Packet Number), (**2**) the number of times the Grooming service has serviced each of the associated application flows (Round Number), and (**3**) the sequence number of the packet as specified by the application (Application Sequence Number).

We claim in Section 1.3.1 that different variations of multiplexing mechanisms and schedul-

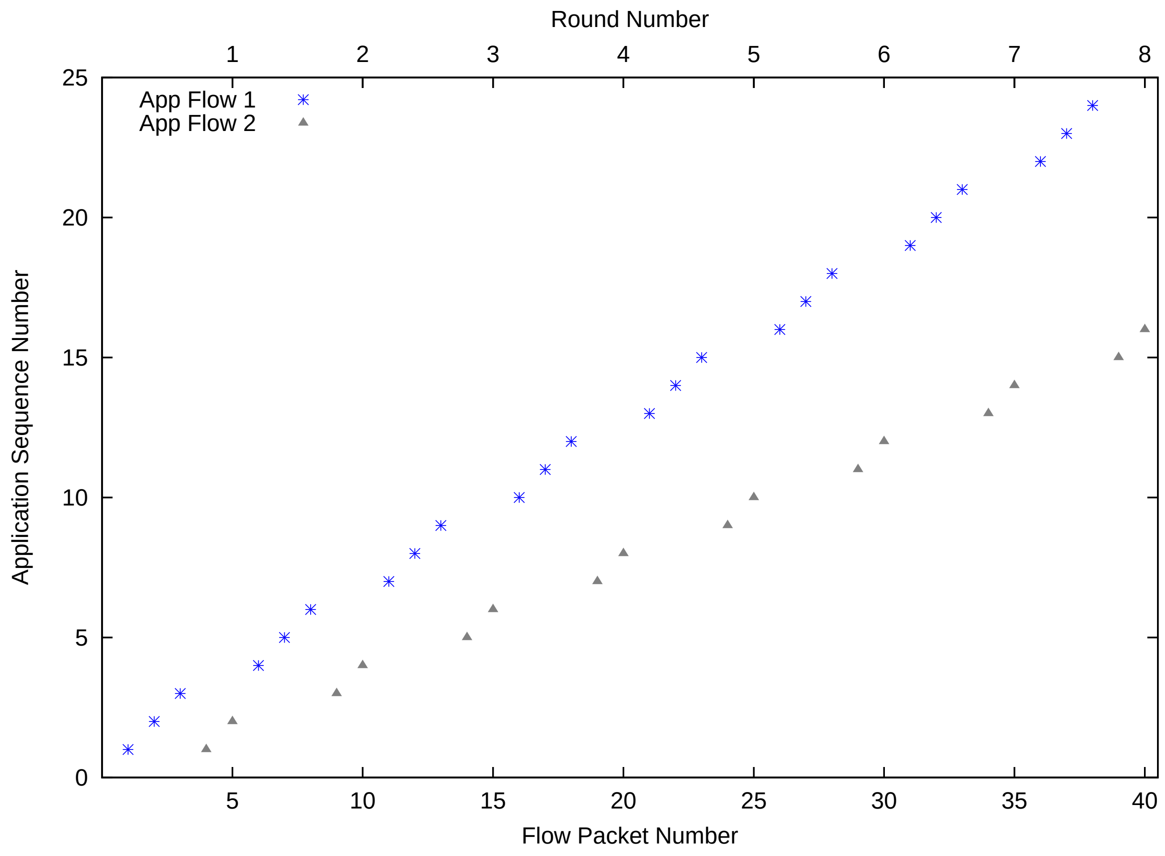


Figure 4.5: Operation of Traffic Grooming Service

ing schemes can be easily accommodated within the grooming functionality. To validate this claim, we implemented Round Robin [38] and Deficit Round Robin [80] multiplexing mechanisms within the Grooming service, and we implemented First Fit and Next Fit [58] scheduling schemes within the Scheduler tuning algorithm.

Verifying the operation of Grooming Service

Figure 4.5 demonstrates how the Grooming service handles application flows with varied bandwidth requirements. In this experiment, we consider an application flow (App Flow 1) using three packet service slots and another (App Flow 2) using two packet service slots; both applications send packets that are of the same size. The Grooming service processes five service slots per round and one packet per service slot, totaling five packets per service round.

There are two things to note in Figure 4.5. First, in each round of servicing five packets, the Application Sequence Number for App Flow 1 increases by three per round, while that of App Flow 2 increases by two; this implies that three packets of App Flow 1 and two packets of App

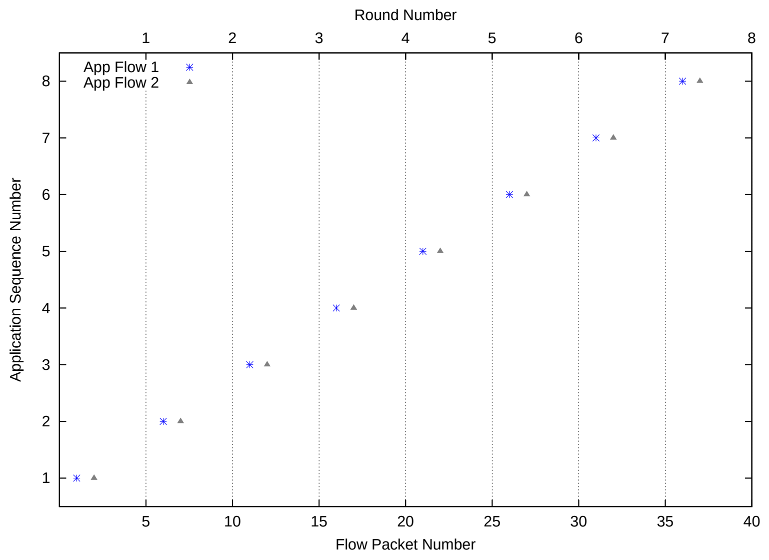


Figure 4.6: Round Robin Grooming

Flow 2 are processed per round, thus satisfying the bandwidth requirement. Second, all packets belonging to the same flow are grouped together; this implies that App Flow 1 is serviced before App Flow 2 in each round. These observations help us to verify that the Grooming service processes application flows with varied bandwidth requirements in a particular manner that repeats after every round.

Comparison between Multiplexing Schemes

In this experiment, the Round Robin (Figure 4.6) and Deficit Round Robin (Figure 4.7) schemes are implemented within the Grooming service. The Round Robin scheme services flows in circular order. All flows are treated equally without consideration for priority or packet size; as a result, flows with small packet size get less than their fair bandwidth share compared to flows with large packet sizes. Deficit Round Robin addresses this limitation of Round Robin by using a deficit counter to track the bandwidth used by each flow per service round. If a flow uses more bandwidth than its fair share within a service round, it receives proportionally less bandwidth in the next service round.

The Grooming service processes five service slots per round, as before. We consider one application flow (App Flow 1) having double the packet size of the other flow (App Flow 2); each of the application flows use only one service slot per round. As shown in Figures 4.6 and 4.7, markers for only two flows are visible in each round; this implies that the three remaining service slots are available for use by other flows.

There is one observation to consider about the Round Robin scheme. The Application

Sequence Number for both flows increases by one in each service round; this implies that packets from App Flow 1 and App Flow 2 are serviced sequentially in each round, thus verifying the Round Robin behavior. Since the packet size of App Flow 1 is double that of App Flow 2, App Flow 1 gets twice the amount of bandwidth than does App Flow 2, even though both flows requested the same amount of bandwidth.

For the Deficit Round Robin scheme, there is also one observation to consider. The Application Sequence Number for App Flow 2 increases by one in each service round, while that for App Flow 1 increases by one every alternate service round; this implies that both flows receive their fair share of bandwidth, in contrast to, the Round Robin scheme.

Implementing both these schemes demonstrate that different multiplexing schemes can easily be accommodated in the Grooming functionality.

Comparison between Scheduling Schemes

In this experiment, the First Fit (Figure 4.8) and Next Fit (Figure 4.9) schemes are implemented within the Scheduler tuning algorithm. The First Fit scheme "fits" the application flow's bandwidth requirements by scanning from the beginning of slots to the end, until a set of available slots which can accommodate the relevant flow, is found. The Next Fit scheme scans from the current slot to search for the next available slot, by contrast to the First Fit scheme; if it reaches the end, it starts again from the beginning of slots until either available slots are found or a cycle is complete.

The Grooming service processes five service slots per round, and any slot which is free does

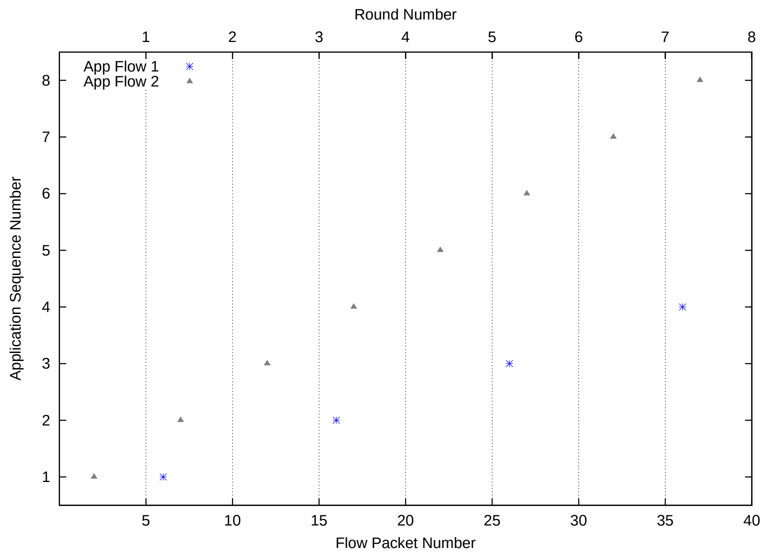


Figure 4.7: Deficit Round Robin Grooming

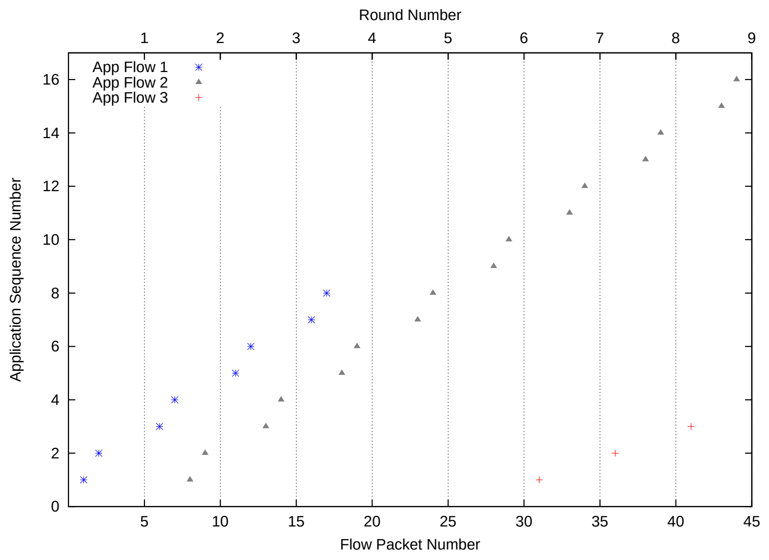


Figure 4.8: First Fit Scheduling

not have a corresponding marker. We consider three application flows (App Flow 1, App Flow 2, and App Flow 3), all of them sending packets of the same size.

As shown in the figures, initially, only one application flow (App Flow 1) is operating, using two service slots per round. When the Flow Packet Number is eight, the scheduler gets an application request to process App Flow 2 using two service slots. Since there are three service slots available, the scheduler accommodates this request; this is shown in the figures for both First Fit and Next Fit scheme. There are two things to note that are common to both schemes. First, the scheduler allocates two slots immediately following App Flow 1, conforming to the behavior of the First Fit and Next Fit schemes. Second, the Application Sequence Numbers for App Flow 2 are lower than those of App Flow 1, implying that the service of App Flow 2 packets has just started. Both these inferences demonstrate the successful addition of an application flow into the virtual circuit.

When the Flow Packet Number is seventeen, the application with flow App Flow 1 terminates; the scheduler frees the service slots that were allocated to App Flow 1. This action is common, again, to both First Fit and Next Fit schemes. The deletion of App Flow 1 from the virtual circuit is shown by the absence of its corresponding markers in Figures 4.8 and 4.9. The Grooming service continues to process App Flow 2 at its scheduled slot intervals. This demonstrates the successful deletion of an application flow from the virtual circuit.

When the Flow Packet Number is thirty, the scheduler receives another application request to process its flow (App Flow 3) with one service slot. Since there are three available slots, the scheduler can accommodate the request; the two slots previously occupied by App Flow 1 are available, at the beginning of the service round and one at the end of the service round. In Figure 4.8, we see a new marker placed at the beginning of the service round, implying that the corresponding slot is being used by App Flow 3, and the scheduler implements the First Fit scheme. In Figure 4.9, we see a new marker placed slot immediately following App Flow 2, that is, at the end of the service round, implying that the corresponding slot is being used by App Flow 3 as the scheduler implements the Next Fit scheme.

4.5 Summary

In this chapter, we reported on the implementation of extensions to the SILO architecture that enable it to support interface silos and the scheduling of silos at periodic time intervals. Interface silos are created when the Services and SILO Manager is started based on pre-configured parameters. Interface silos process all data flows using a particular NIC and contain services which perform NIC specific processing on these data flows. Each interface silo is associated with a connection point to perform flow aggregation and contains a connection point aware service which is used to multiplex application flows.

The interface silos can be scheduled to execute at periodic intervals by the connection point aware service. The duration of each period and number of bytes processed within one period is determined by the speed of the associated NIC. The SILO and Services Manager prevents the interface silo from scheduling itself more than once in each time period.

The Traffic Grooming functionality along with the Scheduling tuning algorithm uses these

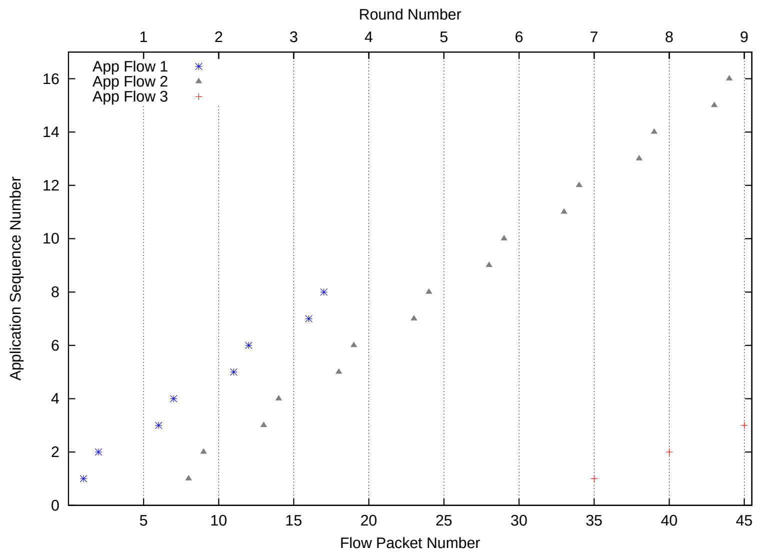


Figure 4.9: Next Fit Scheduling

extensions in addition to flow aggregation and application request handling. New applications request that the Scheduler tuning algorithm accommodate their data flows into the merged flow. The Scheduler determines the order of multiplexing application flows. The Grooming service performs multiplexing based on whichever policy it implements. These extensions and Traffic Grooming/Scheduler functionality were demonstrated with the SILO architecture as a proof-of-concept prototype. We also validated the behavior of Grooming and Scheduler functionality by extending the Grooming service to log information, verifying the results to determine the correctness of their implementation.

Chapter 5

Hierarchical Network Design for Multi-Granular Optical Networks

5.1 Introduction

The emergence of *Dense Wavelength Division Multiplexing* (DWDM) has helped utilize the enormous bandwidth available in optical networks. Each traffic demand is routed to its destination based on a specific wavelength allocated along the path to its destination. Recent advances in optical communication have increased the capacity of individual wavelengths and the number of wavelengths carried by optical fibers. However, the increase in traffic demands between source-destination pairs has not kept pace with the increases in individual wavelength capacity. Thus, each sub-wavelength traffic demand occupies one whole wavelength when routed, resulting in wastage of residual wavelength capacity. An increase in the number of wavelengths in each fiber also increases the number of ports required at each switch in network nodes, which increases the size of the switch and the complexity of switching logic and control planes.

Optical networks are primarily used in Internet backbone networks and in some metropolitan networks. Switches in such optical networks consist of Digital Cross Connects (DXC) in the electronic domain and Optical Cross Connects (OXC) in the optical domain. The DXC switches traffic from the metropolitan or access networks and to the OXC. The OXC then routes these traffic demands as lightpaths, taking advantage of the low switching cost in the optical domain. *Lightpaths* are multi-hop optical connections between source and destination nodes.

Future optical networks are expected to carry traffic demands that range in size from sub- to super-wavelength. To ensure that resources are utilized efficiently, traffic demands must be aggregated, and carried over the network in a cost-effective manner. Assuming that the fiber infrastructure exists, the network cost is typically taken as a function of the total number of switching ports across all network nodes. When all demands are sub-wavelength in size, the network cost is related to the number of *electronic DXC* switching ports required to combine the various traffic components into wavelength-capacity lightpaths. The area of research concerned with the cost-effective transport of sub-wavelength traffic over optical networks is referred to as "traffic grooming." The reader is referred to [34] for a comprehensive survey and classification of research on traffic grooming.

More recently, it has been recognized that combining multiple wavelengths into logical containers called "wavebands" [18] can lead to a significant reduction in the number of *optical OXC* switching ports in the network, since intermediate nodes only need a single port to switch a waveband (instead of one port for each of the constituent wavelengths). This observation has led to the development of Multigranular Optical Cross-Connects (MG-OXCs), which are capable of switching optical signals at multiple granularities, including single wavelengths, single wavebands, or whole fibers.

5.1.1 Prior Work

MG-OXCs can be single- or multi-layered [18]. Multi-layer MG-OXCs consist of separate layers of cross-connects that switch traffic at fiber (FXC), waveband (BXC), or wavelength (WXC) granularity. Multi-layer MG-OXC architectures have been proposed and studied in [49], [70], [32], and their merits in terms of modularity, cross-talk, and complexity are discussed in [49]. [32] proposes a framework extending the GMPLS protocol suite to cover multi-granular aspects of optical networks. A discussion of the cost-performance trade-offs using band aggregation overhead and band contention in MG-OXCs can be found in [56], where an analytical model was also developed to demonstrate the benefits, in terms of port cost, of MG-OXCs over plain OXCs. Single-layer MG-OXCs consist of a unified layer which can connect fibers, wavelengths, and wavebands, as proposed in [55, 103]. The study in [19] concludes that single-layer MG-OXCs provide a greater reduction in switch size under a static traffic model, while multi-layer MG-OXCs provide lower blocking probability under a dynamic traffic model.

With the advent of MG-OXCs, a new network design problem has emerged, namely, the problem of grooming wavelength demands onto wavebands and routing these wavebands over a multi-granular optical network so as to minimize the number of optical switching ports. This is referred to as the *waveband grooming problem.* Variants of this problem have been studied in several contexts, and it has been found that the number of optical ports required to carry a given traffic matrix is affected by the composition of wavebands and the manner in which lightpaths are grouped into wavebands. The work in [43] establishes an equivalence between the waveband routing and wavelength assignment and logical topology design problems. The Logical Topology Design problem deals with designing a topology that interconnects nodes, abstracting out the details of the optical layer. A Routing and Wavelength Assignment (RWA) algorithm determines the route and wavelength(s) for traffic requests based on the logical topology with the objective of minimizing the number of wavelengths. The effect of uniform vs. non-uniform waveband size on switching cost is studied in [24], while [55] proposes non-uniform wavebands and addresses the issues of waveband size selection and wavelength assignment. A waveband RWA algorithm to minimize wavelength conversion is presented in [59].

Wavelength banding techniques in regional and metropolitan optical access networks are discussed in [79]. The merits and shortcomings of using waveband routing and its application to access ring topologies are discussed in [44]. [54] proposes heuristics for ring topologies. A destination-based lightpath grouping mechanism for mesh networks is described in [60], and [20] presents a balanced path routing algorithm for forming wavebands in networks of general topology. According to the study in [61], same-destination-intermediate grouping of lightpaths works slightly better than end-to-end grouping in mesh networks, while [70] finds that intermediate grouping performs better than end-to-end grouping as the number of network nodes increases. Waveband routing under dynamic traffic loads has also been studied; [17] proposes an algorithm which routes lightpaths along the path with the maximum links in common with lightpaths already used, while [21] proposes an algorithm which assigns wavelengths by grouping the lightpaths and using wavelength converters to satisfy new requests.

5.1.2 Our Contribution

The above studies regard the network as a flat entity for the purposes of lightpath grooming, waveband routing, and wavelength assignment. However, it is well-known, that in existing networks, resources are typically managed and controlled in a hierarchical manner. The levels of the hierarchy either reflect the underlying organizational structure of the network or are designed in order to ensure scalability of the control and management functions. Based on this observation, a hierarchical approach was developed to address traffic grooming that emulates the hub-and-spoke model used by the airline industry to "groom" passenger traffic onto connecting flights, and this research demonstrated that the hierarchical approach is effective in minimizing the electronic port cost [22]. With an increase in the number of entities that need to be controlled, a hierarchical framework for managing wavebands is even more warranted in multigranular optical networks.

Optical fibers carry a number of distinct wavelengths but are limited by the fiber's physical characteristics and the state of optical technology in combining wavelengths onto or splitting them from the fiber [78]. Also, installing and maintaining new optical fibers is prohibitively expensive. Hence, the number of wavelengths used to satisfy traffic demands is an important measurement in network design.

The objective of our algorithm is to develop a hierarchical solution which is scalable across different waveband sizes and number of entities or clusters while satisfying a given set of traffic demands and keeping the number of optical ports low. We assume that a uniform waveband size is used throughout the network, all network nodes are MG-OXCs, and none of the nodes have wavelength conversion ability.

We study three different MG-OXC switch architectures, depending on the interactions of three layers (fiber, waveband, and wavelength cross-connects) with one another. In all the models considered, the waveband layer interacts with the fiber layer but does not necessarily interact with the wavelength layer; similarly, the fiber layer does not necessarily interact with the wavelength layer. An analysis of the different models provides the upper and lower bounds to the cost of bypassing, adding, or dropping one fiber, waveband, or wavelength.

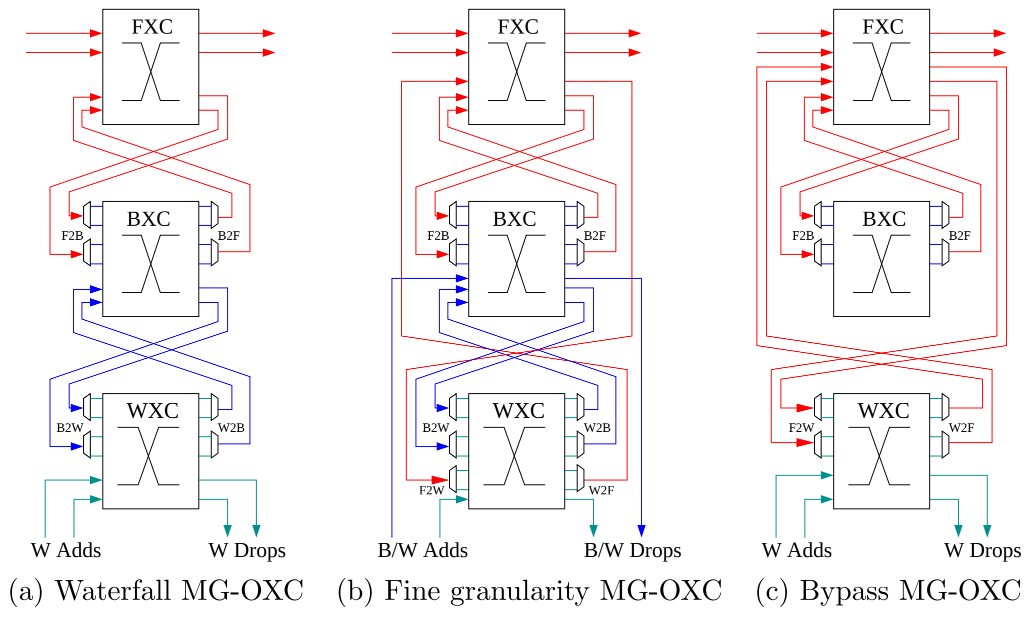
In this chapter, we extend the hierarchical approach in [22] to perform routing and "coloring" of wavebands in a scalable and efficient manner. At the first level of the hierarchy, the network is partitioned into clusters, and one node in each cluster (referred to as the *hub*) is responsible for grooming intra-cluster lightpaths, as well as inter-cluster lightpaths originating or terminating locally. At the second level of the hierarchy, each hub grooms lightpaths routed to a specific remote cluster into wavebands, and routes the latter directly to the remote cluster's hub. Finally, the routing and coloring of wavebands on the underlying physical topology is performed using a standard RWA algorithm. Our approach provisions only a few nodes (the hubs) for grooming lightpaths they do not originate or terminate, and efficiently packs lightpaths among remote clusters onto wavebands between the corresponding hubs.

The chapter is organized as follows. In Section 5.2, we describe the switch model and define the network design problem. In Section 5.3, we present the hierarchical algorithm for lightpath grooming and waveband routing; the corresponding numerical results are presented in Section 5.4. We conclude in Section 5.5.

5.2 Optical Switch Architectures

We consider a general topology network with N nodes interconnected by links consisting of one fiber per direction. Fiber links carry D wavebands and each waveband consists of L consecutive wavelengths; hence, the total number of wavelengths on each link is $W = D \times L$. We assume the existence of a traffic demand matrix $T = [t^{(sd)}]$, where integer $t^{(sd)}$ denotes the amount of (forecast) long-term traffic, in multiples of the wavelength capacity, to be carried from node s to node d; any changes in the demand matrix take place over long time scales, and, for the purposes of this work, the matrix T is assumed to be fixed.

Let us define a *bandpath*, a generalization of the lightpath concept, as a (waveband, path) pair that is associated with a number of lightpaths equal to the band size L. The waveband is an integer in the range $1, \cdots, D$, where D is the total number of wavebands in a fiber, and can



(a) Waterfall MG-OXC (b) Fine granularity MG-OXC (c) Bypass MG-OXC

Figure 5.1: Switch Architectures (F: fiber, B: waveband, W: wavelength)

be thought of as the "color" of the bandpath, while the path may span multiple links. Hence, a bandpath uniquely identifies the path over which the traffic on the associated set of L lightpaths will be carried, as well as the wavelengths of these L lightpaths (since each waveband consists of a unique set of L wavelengths).

Each network node is equipped with an MG-OXC. MG-OXCs are characterized by the switching and grooming capabilities they provide. Specifically, we consider the following capabilities, listed from finer to coarser granularity:

- *wavelength switching* refers to the ability to switch individual wavelengths optically;
- *waveband grooming* is the ability to demultiplex a waveband into its constituent wavelengths, and to add or remove wavelengths from the waveband;
- *waveband switching* is the capability to optically switch individual wavebands, that is, to switch all wavelengths in a waveband as a group;
- *fiber grooming* refers to the ability to demultiplex a fiber into its constituent wavebands, and to add or remove wavebands from the fiber; and
- *fiber switching* is the capability to switch a whole fiber from an input to an output port.

In general, there is a trade off between the switching/grooming granularity and the cost of an MG-OXC [56], with finer granularity implying greater flexibility, but also higher cost.

There are three different kinds of MG-OXC we consider. The MG-OXC depicted in Figure 5.1(a) can switch traffic at the fiber, wavebands or wavelength levels using the corresponding fiber, waveband or wavelength cross-connects. It is also capable of grooming traffic at the waveband and fiber levels; however, traffic can be added(dropped) from(to) the digital cross-connect (DXC) only at the wavelength level. Similarly, the waveband cross-connect (BXC) can add (drop) wavebands from(to) the wavelength cross-connect (WXC) and fiber cross-connect (FXC) can add(drop) fibers from the waveband cross-connect. Since traffic demands terminating at a node forces its corresponding fiber, waveband, and wavelength to be dropped, this model is called the *Waterfall MG-OXC*.

In addition to having the capabilities of the Waterfall model, the *Fine granularity MG-OXC* depicted in Figure 5.1(b) can add(drop) wavebands from(to) the digital cross-connect (DXC) at the waveband level. This is useful if waveband sized traffic is routed together over some or all the links until their respective destinations. However, the wavebands still need to be split into(merged from) constituent wavelengths before entering(after leaving) the DXC. This necessitates additional interfaces at the DXC. Also, the FXC can add(drop) fibers from(to) the WXC. This also adds an additional interface at the wavelength cross-connect.

The *Bypass MG-OXC* depicted in Figure 5.1(c) does not have waveband grooming capability but can switch fibers, wavebands, and wavelengths. Traffic can be added or dropped only at the wavelength level. The FXC can add(drop) fibers from(to) waveband and wavelength cross-connects.

Additional interfaces at the DXC and WXC that are required by the Fine granularity and Bypass MG-OXC Switches incur additional cost in terms of switch size, equipment, and maintenance, but if RWA is done intelligently, the cross-connect size can be reduced substantially, especially in the case of the Fine Granularity model.

In this work we assume that all the OXCs in the network have exactly the same capabilities (i.e., the network is homogeneous in terms of switching and grooming capability). We also assume that the cost of a node is determined by the cost of the optical ports of its MG-OXC; this is a reasonable assumption that is commonly adopted in the literature.

We quantify the cost of switching, adding, or dropping wavelengths, wavebands, or fibers based on the OXC architecture considered. Let us assume there are D wavebands in a fiber, with each waveband having L wavelengths. The cost of a port in a WXC switch is w, in a BXC is l and in a FXC is f. The cost to drop a wavelength w_{drop}, waveband l_{drop}, and fiber f_{drop} are given by

w_{drop} is the sum of the cost to demultiplex a wavelength into the electronic domain and the cost of a port in a WXC switch, w.

l_{drop} is the sum of the cost to demultiplex a waveband into wavelengths and the cost of a port in a BXC switch, l.

f_{drop} is the sum of the cost to demultiplex a fiber into wavelengths and the cost of a port in a FXC switch, f.

The cost to bypass a wavelength is $w_{bypass} \leftarrow 2 \times w$, to bypass a waveband is $l_{bypass} \leftarrow 2 \times l$ and to bypass a fiber is $f_{bypass} \leftarrow 2 \times f$. *The costs to bypass, add, or drop a wavelength, waveband, or fiber is assumed to be the same.*

The maximum cost to drop wavebands from a fiber using only BXC occurs when *only one waveband* is dropped while all the rest of the wavebands are bypassed at the node and is given by

$$Maximum(L_{drop}^{BXC}) = l_{drop} + (D-1) \times l_{bypass} + f_{drop} + f_{add} \tag{5.1}$$

The minimum cost to drop wavebands from a fiber using only waveband switching occurs when *all the wavebands* are dropped at the node and is given by

$$Minimum(L_{drop}^{BXC}) = l_{drop} + f_{drop}/D \tag{5.2}$$

The maximum cost to drop wavelengths from a fiber using only WXC occurs when *only one wavelength* is dropped while all the rest of the wavelengths are bypassed at the node and is given by

$$Maximum(W_{drop}^{WXC}) = w_{drop} + (L-1) \times D \times w_{bypass} + f_{drop} + f_{add} \tag{5.3}$$

The minimum cost to drop wavelengths from a fiber using only WXC occurs when *all the wavelengths* are dropped at the node and is given by

$$Minimum(W_{drop}^{WXC}) = w_{drop} + f_{drop}/(D \times L) \tag{5.4}$$

The cost for adding or dropping a wavelength through BXC and WXC is similar to the above equations and is given as

$$Maximum(W_{drop}^{BXC}) = w_{drop} + (L-1) \times w_{bypass} + l_{drop} + (D-1) \times l_{bypass} + f_{drop} + l_{add} + f_{add} \tag{5.5}$$

$$Minimum(W_{drop}^{BXC}) = w_{drop} + l_{drop}/L + f_{drop}/(D \times L) \tag{5.6}$$

All equations are applicable to the Fine-Granularity Model, while equations (5.3) and (5.4) are applicable to the Bypass Model and equations (5.5) and (5.6) are applicable to the Waterfall Model. These equations give us the basis to calculate the number of ports used by the solutions in the hierarchical model.

5.3 Hierarchical Approach to Waveband Switching

Given the forecast traffic demands $\{t^{(sd)}\}$, our objective is to carry the traffic matrix in its entirety while minimizing the total optical port cost and wavelength cost in the network. This problem involves the following conceptual sub problems (SPs):

1. *logical topology SP:* find a set of lightpaths to carry the traffic demands $\{t^{(sd)}\}$;

2. *lightpath grooming SP:* groom the lightpaths into wavebands; and

3. *routing and wavelength assignment SP:* assign a waveband and path over the physical topology to each bandpath; as we mentioned earlier, lightpaths within each bandpath will also be assigned a wavelength as a result.

This is only a conceptual decomposition that helps in understanding and reasoning about the problem; in an optimal approach, the sub problems would be considered together by the solution. Note that, if the first two sub problems above have been solved, the third sub problem reduces to the classical routing and wavelength assignment (RWA) problem [25], with the difference that the entities to be routed and colored are lightpaths considered in units of bandsize. Hence, the above optimization problem is NP-hard.

5.3.1 Algorithms for the Hierarchical Model

We now present an extension of the hierarchical model developed in [22] to tackle this lightpath grooming and bandpath routing problem. We assume that all nodes in the network are equipped with a one of the MG-OXCs as shown in Figure 5.1. All switch models add (or drop) wavelength, while the fine granularity model in Figure 5.1(b) can also add (or drop) wavebands.

In our approach, we assume that the network is partitioned into clusters (or islands) of nodes, where each cluster consists of nodes in a contiguous region of the network. The clusters may correspond to independent administrative entities (e.g., autonomous systems), or may be created solely for the purpose of simplifying resource management and control functions. For the purposes of grooming and routing, we designate one node within each cluster as the *hub*.

The hierarchical algorithm solves the waveband switching problem in two steps, as shown in the *Waveband Routing and Wavelength Assignment Algorithm* (Algorithm 4). In the first step, the inter-cluster traffic demands are identified and connections are established between hubs. These connections are extended to originate from (and terminate at) nodes within the source (and destination) clusters. Thus inter-cluster traffic demands are satisfied and lightpaths are established using these connections. In the last step, the intra-cluster traffic demands are satisfied.

Our hierarchical model is explained below:

1. **Clustering and hub selection.** In this phase, we use the modified K-center algorithm [23] based on the 2-approximation algorithm [46] for the K-center problem to partition the network into K clusters. We select one node having the maximum nodal degree in each cluster as the hub. It is assumed that all-pair shortest paths have been pre-computed and recorded as input matrix distance.

2. **Routing and Wavelength Assignment.** As mentioned before, this phase consists of the two steps listed below.

   (a) *Inter Cluster Routing and Wavelength Assignment*

      i. *Connection Establishment between Hubs.* The inter-cluster traffic $T = [t^{(s,d)}]$ between nodes s and d in clusters C_i and C_j having hubs h_i and h_j, respectively, are aggregated into $H = [h^{(s,d)}]$ according to equation (5.7).

$$h^{(h_i,h_j)} = \lceil (\sum_{s \in C_i, d \in C_j} t^{(s,d)})/L \rceil \quad \forall 1 \leq i, j \leq K, i \neq j \tag{5.7}$$

      A set of a bandsize number of connections are established between hubs to help satisfy inter-cluster traffic demands using the *Routing and Waveband Assignment Algorithm* 1. This algorithm uses the shortest path between hubs ($\mathcal{P}$) on the least waveband number (B) available, thus, minimizing the number of wavelength W.

      ii. *Connection Extension within Clusters.* Once a set of connections are established, they are extended to non-hub nodes (if required) in the source and destination clusters. For each wavelength λ in the waveband B, the *Destination Cluster RWA Algorithm* (Algorithm 2) identifies a node in the destination cluster that has traffic originating from the source cluster and is farthest from its hub. The connection corresponding to wavelength λ from the set is extended from destination hub h_d to the node d along $\mathcal{P}'$. Similarly, the *Source Cluster RWA Algorithm* (Algorithm 3) extends a connection corresponding to wavelength λ from a node s in the source cluster that has traffic terminating at d and is farthest from its hub h_s on a path $\mathcal{P}''$ between s and h_s.

      If no path is found on wavelength λ in the source cluster, its equivalent connection in the destination cluster between d and h_d on λ along $\mathcal{P}'$ is freed. If no path is found in the destination cluster, connection extension is performed on the next wavelength in the set. If none of connections in the set established in step 2(a)i are extended, all of them are freed.

   (b) *Intra-Cluster Routing and Wavelength Assignment.* In this step, only intra-cluster traffic demands remain to be satisfied. Bandpaths are established between source and destination nodes using the Routing and Waveband Assignment Algorithm along path $\mathcal{Q}$ on waveband B.

```
Algorithm 1 Routing WaveBand Assignment
  Input: Mesh DWDM network ($G_P$), Waveband size $L$, $W$ wavelengths/link, Traffic Matrix $T = [t^{(sd)}]$.
  Output: Waveband sized Set of Connections identified by $\mathcal{P}$ between a source destination pair $(s, d)$, such that $t^{(s,d)} > 0$, and waveband number $B$.
  begin
    1. Let $B \leftarrow 0$.
    2. Identify a source-destination pair $(s, d)$ which are farthest from each other, such that $t^{(s,d)} > 0$.
    3. Find the shortest path $\mathcal{P}$ over $G_P$ between $s$ and $d$ collectively on the wavelength graphs corresponding to wavelengths $B \times L, \cdots, ((B + 1) \times L) - 1$.
    4. If $\mathcal{P}$ is not found, increment $B$ by one and repeat (2-3). If $B$ exceeds $W/L$, exit with failure.
    5. Remove edges on wavelength graphs corresponding to wavelengths $B \times L, \cdots, ((B + 1) \times L) - 1$ along $\mathcal{P}$.
    6. Return $(\mathcal{P}, B, s, d)$.
  end
```

Hub nodes are the only nodes that perform any grooming of lightpaths into wavebands. All lightpath grooming is performed in the optical domain without using wavelength conversion.

5.3.2 An Example

To further explain the hierarchical model, we use an example depicted in Figure 5.2. Let us assume single wavelength traffic demands exist from nodes 1 to 4, 2 to 1, and 2 to 7, and no other traffic demands exist. Upon invoking the *Clustering* algorithm over the network (shown in Figure 5.2(a)) with $K = 2$, we get a network partitioned into two clusters, C_3 and C_5 with nodes 3 and 5 as hubs since they have the maximum nodal degree within their cluster. The resulting network is shown in Figure 5.2(b).

In the *Connection Establishment* stage, after $t^{(1,4)} = 1$ and $t^{(2,7)} = 1$ are aggregated into inter-cluster traffic demands H by equation 5.7, we get $h^{(3,5)} = 2$, implying two traffic demands

Algorithm 2 Destination Cluster RWA

```
Input: Mesh DWDM network ($G_P$), Traffic Matrix $T = [t^{(sd)}]$, Source Cluster $C_{h_s}$ with its hub $h_s$, Destination Cluster $C_{h_d}$ and its hub $h_d$, Wavelength to be routed on $\lambda$.
Output: Lightpath identified by the path $\mathcal{P}'$ from $h_d$ to $d$ and wavelength $\lambda$ with $d$ having originating traffic in $C_{h_s}$.
begin
    1. Mark all nodes within the destination cluster as not visited.
    2. Consider an unvisited node $d$ farthest away from $h_d$, such that $t^{(s,d)}_{s \in C_{h_s}, d \in C_{h_d}} > 0$.
    3. Find the shortest path $\mathcal{P}'$ over $G_P$ from $h_d$ to $d$ on the wavelength graph corresponding to wavelength $\lambda$.
    4. If $\mathcal{P}'$ is not found, mark $d$ as visited and repeat (2-3) until either $\mathcal{P}'$ is found to some other node in $C_{h_d}$ or all nodes in cluster $C_{h_s}$ are visited.
    5. Return ($\mathcal{P}'$, $d$).
end
```

Algorithm 3 Source Cluster RWA

```
Input: Mesh DWDM network ($G_P$), Traffic Matrix $T = [t^{(sd)}]$, Source Cluster $C_{h_s}$ with its Hub $h_s$, Destination Node $d$, Wavelength to be routed on $\lambda$.
Output: Lightpath identified by the path $\mathcal{P}''$ from $s$ to $h_s$ and wavelength $\lambda$ with $s$ having traffic terminating in $d$.
begin
    1. Mark all nodes within the source cluster as not visited.
    2. Consider an unvisited node $s$ farthest from $h_s$, such that $t^{(s,d)}_{s \in C_{h_s}} > 0$.
    3. Find the shortest path $\mathcal{P}''$ over $G_P$ from $s$ to $h_s$ on the wavelength graph corresponding to $\lambda$.
    4. If $\mathcal{P}''$ is not found, mark $s$ as visited and repeat (2-3) until either a $\mathcal{P}''$ is found to some other node in $C_{h_s}$ or all nodes in cluster $C_{h_s}$ are visited.
    5. Return ($\mathcal{P}''$, $s$).
end
```

Algorithm 4 Waveband Routing and Wavelength Assignment

```
Input: A mesh DWDM network ($G_P$) partitioned into $m$ clusters $C_1, \cdots, C_m$ having hubs
$h_1, \cdots, h_m$, Waveband size $L$, $W$ wavelengths/link, Traffic Matrix $T = [t^{(sd)}]$.
Output: Set of Lightpaths in the Logical Topology and routing of the traffic components
$t^{(sd)}$, Number of ports used at each network node.
begin
  1. Initialize the inter-cluster traffic $H = [h^{(sd)}]$ according to Equation 5.7.
  2. While Inter-Cluster traffic exists do
     (a) $(\mathcal{P}, B, h_s, h_d) \leftarrow RoutingWaveBandAssignmentAlgorithm(G_P, L, W, H)$
     (b) For $\lambda = B \times L$ to $((B+1) \times L) - 1$ do
         i. $(\mathcal{P}', d) \leftarrow DestinationClusterRWA(G_P, T, C_s, C_d, \lambda)$.
         ii. If $\mathcal{P}'$ is found then
             A. Remove edges on wavelength graph corresponding to $\lambda$ along $\mathcal{P}'$.
             B. $(\mathcal{P}'', s) \leftarrow SourceClusterRWA(G_P, T, C_s, d, \lambda)$.
             C. If $\mathcal{P}''$ is found, then remove edges on the wavelength graph corre-
                sponding to $\lambda$ along $\mathcal{P}''$ and decrement $t^{(s,d)}$, $h^{(s,d)}$ by one. Else, if $\mathcal{P}''$
                is not found, then insert edges on wavelength graph corresponding to
                $\lambda$ along $\mathcal{P}'$.
     (c) If no connections are extended, then free all the connections using $B$ along $\mathcal{P}$.
  3. While Intra-Cluster traffic exists do
     (a) $(\mathcal{Q}, B, s, d) \leftarrow RoutingWaveBandAssignmentAlgorithm(G_P, L, W, T)$
     (b) If $t^{(sd)} > L$ then decrement $t^{(sd)}$ by $L$, else decrement $t^{(sd)}$ by one.
  4. Calculate and print wavelength and number of ports used based on routing information.
end
```

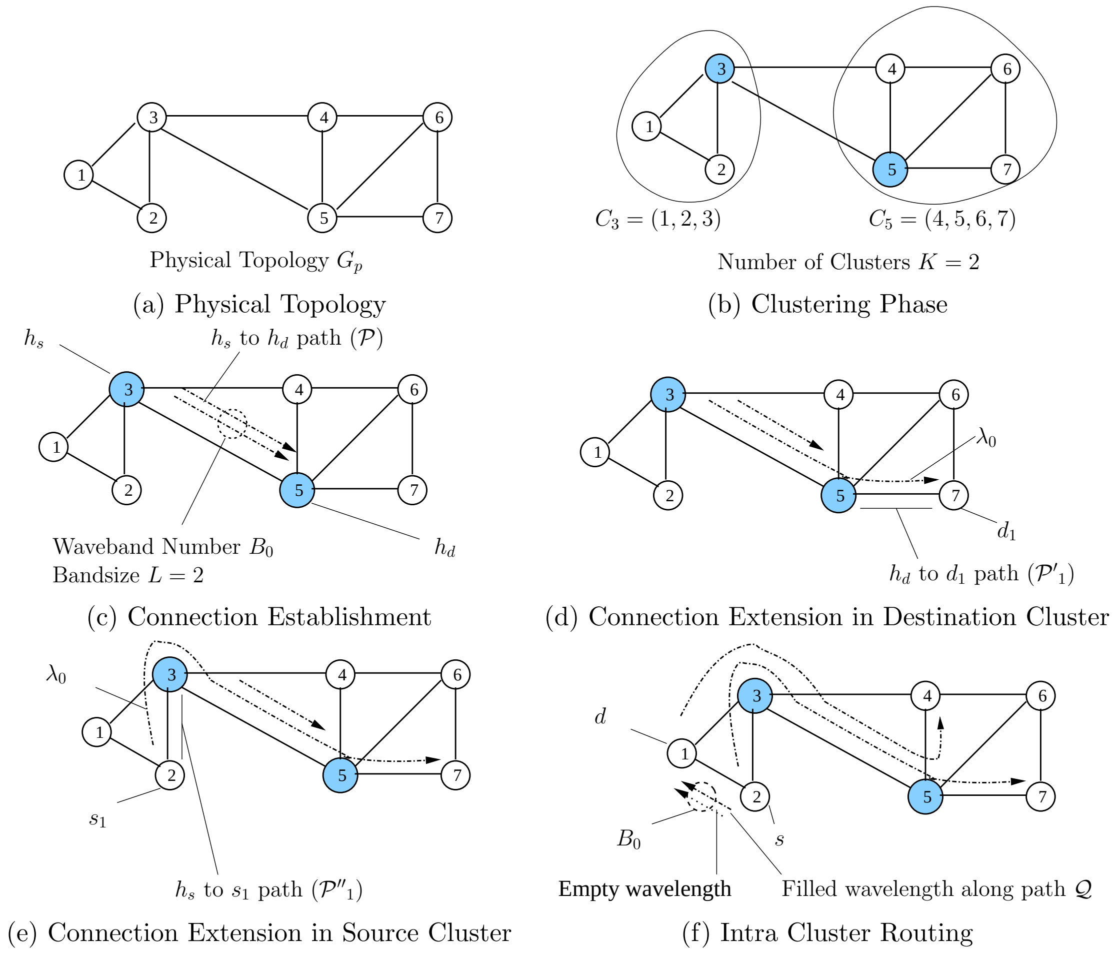

Physical Topology G_p
(a) Physical Topology
$C_3 = (1, 2, 3)$
$C_5 = (4, 5, 6, 7)$
Number of Clusters $K = 2$
(b) Clustering Phase
h_s
h_s to h_d path ($\mathcal{P}$)
Waveband Number B_0
Bandsize $L = 2$
h_d
(c) Connection Establishment
λ_0
d_1
h_d to d_1 path ($\mathcal{P}'_1$)
(d) Connection Extension in Destination Cluster
λ_0
s_1
h_s to s_1 path ($\mathcal{P}''_1$)
(e) Connection Extension in Source Cluster
d
B_0
s
Empty wavelength
Filled wavelength along path $\mathcal{Q}$
(f) Intra Cluster Routing

Figure 5.2: An example of Hierarchical Model

from cluster C_3 to C_5. As there is no traffic from cluster C_5 to C_3, $h^{(5,3)} = 0$. Nodes 2, 1 belong to the same cluster $(2, 1 \in C_3)$; hence, $t^{(2,1)}$ will not be represented in H and all other entries in H remain zero. Using the Routing and Waveband Assignment Algorithm, we obtain a routed waveband connection of size $L = 2$ along path $\mathcal{P} = (3, 5)$ on waveband number B_0 between nodes $h_s = 3, h_d = 5$. Thus, we establish a bandsized set of connection between the hubs, as shown in Figure 5.2(c).

In the *Connection Extension* stage, we iterate through each wavelength (λ_0, λ_1) belonging to waveband B_0. Initially, we consider the destination cluster C_5 and extend a connection from the destination hub 5 to any node (in this case 7 or 4) having originating traffic (not yet satisfied) from source cluster C_3. We choose to extend the connection on wavelength λ_0 along path $\mathcal{P}'_1 = (5, 7)$, as shown in Figure 5.2(d). Next, we consider the source cluster C_3 and extend the connection on wavelength λ_0, choosing any node that terminates at 7. Since only 2 has such traffic, the connection is extended along path $\mathcal{P}''_1 = (2, 3)$, as seen in Figure 5.2(e). We repeat this procedure until $t^{(1,4)}$ is also satisfied.

Lastly, we set up the intra-cluster traffic demands like $t^{(2,1)}$ using the Routing and Waveband Algorithm, which returns the path $\mathcal{Q} = (2, 1)$ on waveband B_0 and nodes $s = 2, d = 1$. Since $t^{(2,1)} = 1$ and the waveband has size $L = 2$, one of the wavelengths is wasted. The resultant routing is shown in Figure 5.2(f).

5.4 Numerical Results

We now present the results of an experimental study to evaluate the performance of the hierarchical lightpath grooming algorithm described in Section 5.3. For comparison, we also implemented the balanced path routing with heavy traffic first (BPHT) algorithm [20]. BPHT was one of the first algorithms developed to groom lightpaths into bandpaths for routing over a network of switches with waveband grooming and switching capabilities. Unlike our approach, the BPHT algorithm regards the network as a flat entity for the purpose of forming bandpaths. Specifically, BPHT computes a small number of shortest paths between each source-destination pair, and then searches these paths, assigning lightpaths to bandpaths, encouraging band merging and splitting at intermediate nodes, in a greedy manner. Whenever bands operating along different links are merged, they need to be dropped to the WXC to allow them to combine into a single band. Similarly. when a band is split, it needs to be dropped to the WXC in order to break up into smaller bands going out on different links. This increases the number of ports required at the nodes where band merge or split occurs. For more details on the operation of this algorithm, please refer to [20].

We conducted our experiments based on a 47-node, 96-link network topology used in [8]. In order to apply our hierarchical algorithm, we partitioned this network into K clusters, $K = 4, 8, 12$, using the algorithm in [46] for the K-center problem. The traffic matrix T for each problem instance is generated by drawing $N(N-1)$ random numbers (where $N = 47$ is the number of nodes) from a Gaussian distribution with mean t and standard deviation of $0.1t$; if the numbers are greater than t, they are rounded up to the next lowest integer, otherwise they are rounded down to the next highest integer. Any negative numbers are set to zero. The generated matrix represents the wavelength demands between all source-destination pairs.

We consider two performance metrics in our study: the optical port cost of the network (over all nodes), and the number of wavelengths required to establish all bandpaths and lightpaths. The port cost includes the cost of all fiber, waveband, and wavelength ports at each node in the network. We discuss the performance of our algorithm primarily with respect to the Fine Granularity MG-OXC model (Figure 5.1(b)) and compare the performance of all models

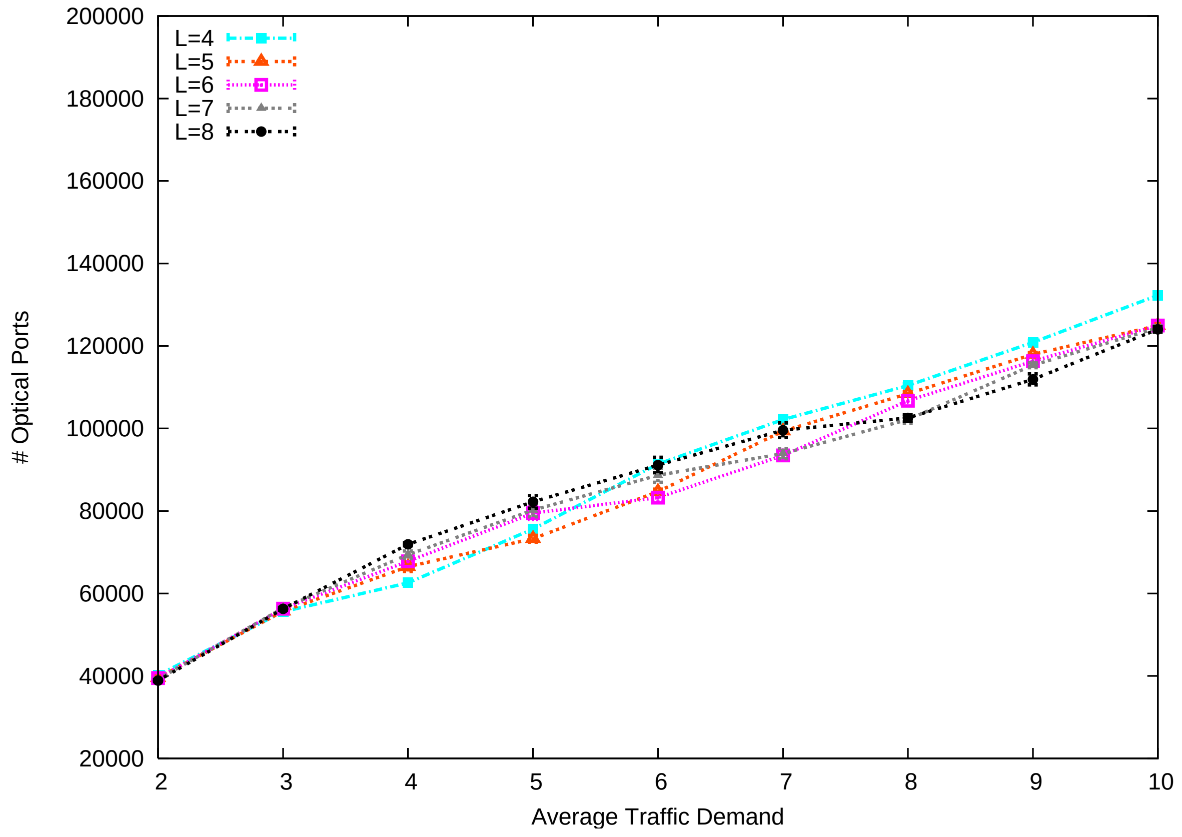


Figure 5.3: Port Cost, Hierarchical Approach, Varying Bandsize, $K = 8$

(i.e. Waterfall, Fine granularity, and Bypass MG-OXC) with one another. All the experiments consider a single fiber scenario except the last experiment (Figures 5.9 and 5.10), where we considered 40 wavelengths to be contained in a fiber.

For the results presented in this section, we have varied the mean t of traffic demand. Each point plotted in the figures represents an average over 30 problem instances (i.e., 30 random traffic matrices) for the stated values of the network and traffic parameters. We have considered a confidence interval of 95%.

5.4.1 Variation in Waveband Size

Figure 5.3 plots the number of optical ports required against the mean value t of the traffic components, using Algorithm 4 for various band sizes when the network is partitioned into $K = 8$ clusters. There are two important observations we can make regarding the trend of the curves. First, the number of ports generally increases with the traffic load, as expected. Second, a larger band size implies lower optical port cost; again, this behavior is expected, as larger

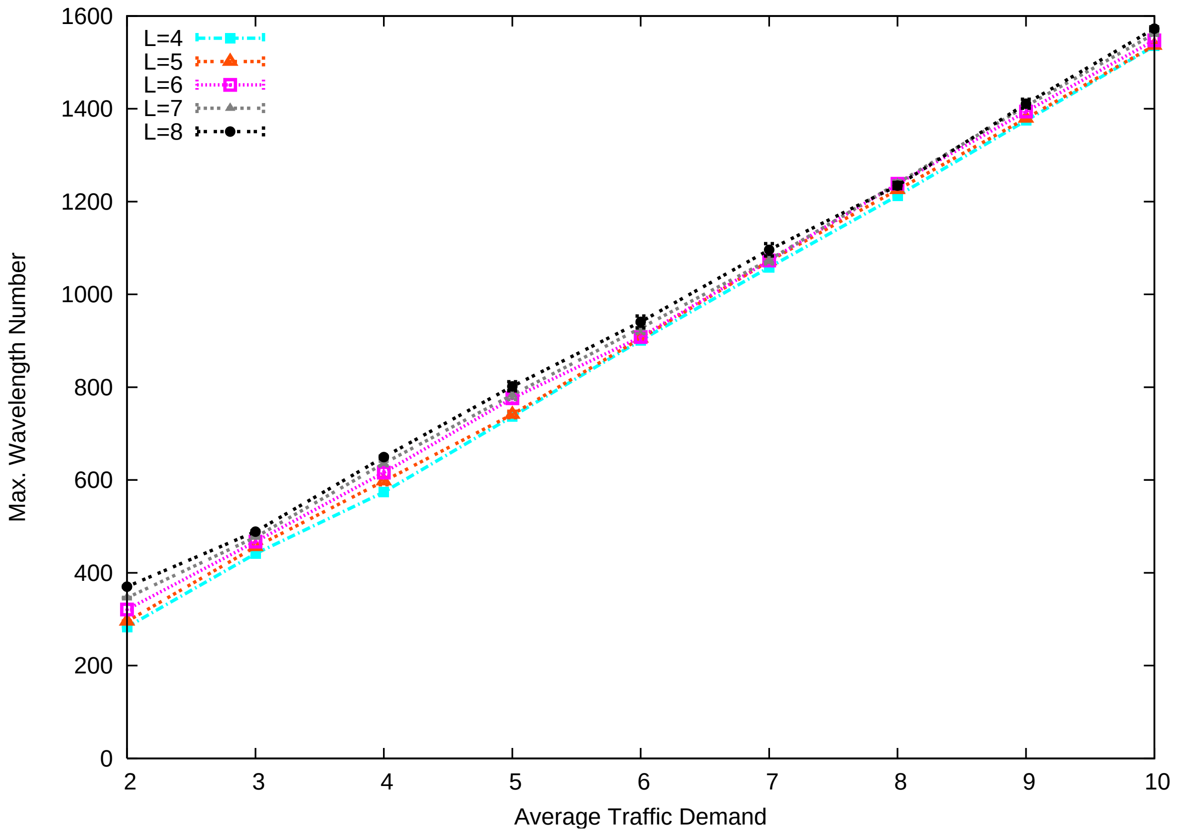


Figure 5.4: Wavelength Cost, Hierarchical Approach, Varying Bandsize, $K = 8$

bands can accommodate more lightpaths, decreasing the number of optical switching ports required at intermediate nodes. (*Note:* the fact that the number of ports is in the order of tens of thousands is due to the large number of traffic demands that need to be carried; specifically, the minimum (respectively, maximum) number of wavelength capacity traffic demands that need to be accommodated is equal to $47 \times 46 \times 2 = 4,324$ for average traffic demand $t = 2$ (respectively, $47 \times 46 \times 10 = 21,620$, for $t = 10$).)

Figure 5.4 plots the wavelength cost against the mean value t of the traffic components, for various band sizes. Again the network is partitioned into $K = 8$ clusters. We observe that the wavelength cost generally increases with traffic demands, as expected. Also, larger band size implies higher wavelength cost. This can be explained as follows, larger band sizes results in more lightpaths being routed together to use band ports at intermediate nodes. However, this requires a contiguous set of wavelengths to be free along the path on which the lightpaths are routed together, increasing the wavelength number over the links in this path.

Figure 5.5 shows that the observations applied for the hierarchical approach can be applied for BPHT as well. However, the number of ports used by BPHT is much higher than the

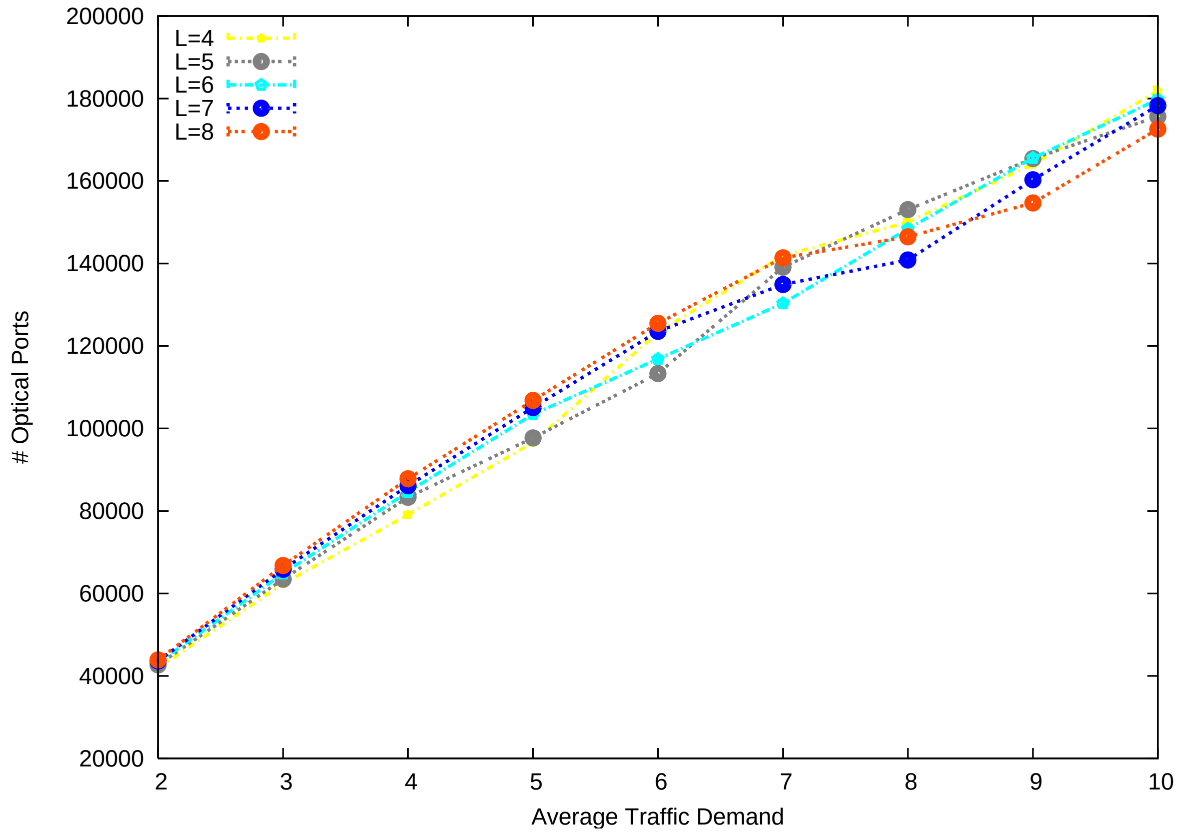


Figure 5.5: Port Cost, Balanced Path Heavy Traffic (BPHT), Varying Bandsize

hierarchical approach, this can be explained as follows. Since BPHT encourages band merges and splits, a number of bands are dropped to the WXC layer before being added back to the BXC layer again. This increases the number of ports needed at the nodes, where band merges and splits take place. However, in the hierarchical approach, band merges and splits are allowed only at the hubs. The bands are routed between source and destination hubs (from the respective source and destination) without modification.

Figure 5.6 shows BPHT using lesser number of wavelengths than the hierarchical approach. This can be explained by observing that, BPHT encourages band merges and splits, in the process more tightly packing the lightpaths together as bandpaths, as compared to the hierarchical approach.

5.4.2 Variation in Cluster Size

Figure 5.7 plots the optical port cost against the traffic load when the band size $L = 8$ and when the network is partitioned into $K = 4, 8$, and 12 clusters. We see that the results of our

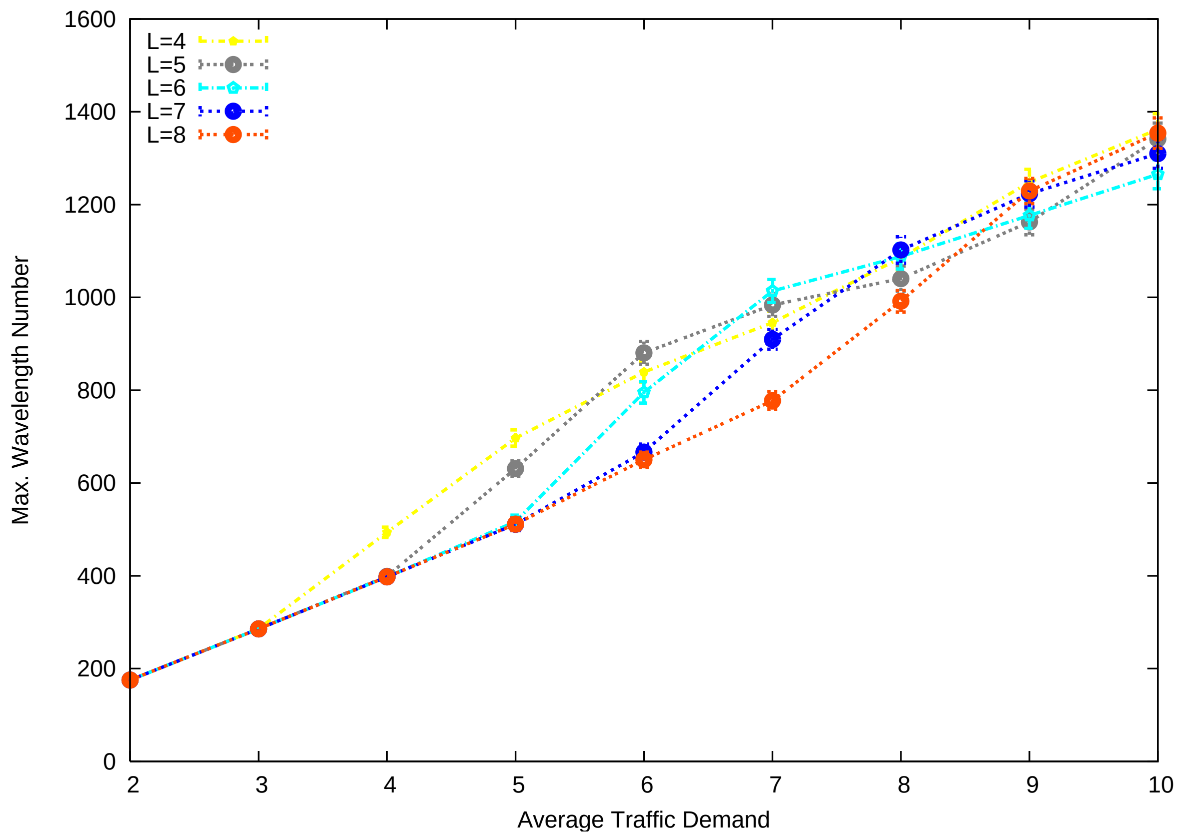


Figure 5.6: Wavelength Cost, Balanced Path Heavy Traffic (BPHT), Varying Bandsize

algorithm depend on the number K of clusters. We observe that using 8 clusters requires much fewer ports than 4 clusters, as each node tends to be closer to its local hub, hence, carrying the traffic to and from the hub requires fewer ports. However, using 12 clusters results in slightly more ports than 4 clusters. This is due to hub-to-hub connections conflicting with non-hub node to hub connections forcing the latter to take longer route. Again, we observe BPHT requires more ports than hierarchical approach for all the clusters considered. Figure 5.8 plots the number of wavelengths as a function of traffic load for $L = 8$. We observe that a larger number of clusters requires fewer wavelengths. This result can be explained by the fact that for a small number of clusters, each hub has to transmit (receive) a large amount of traffic to (from) other hubs; hence, the links directly connected to hubs tend to become congested requiring many wavelengths. With 8 clusters, each hub handles less traffic, alleviating the congestion on its links and leading to fewer wavelengths. However, using 12 clusters results in the use of higher wavelengths as a result of the conflict between hub-hub connections and non-hub node to hub connections. This conflict results in lightpaths being more loosely packed into bandpaths, as compared to the 8 clusters.

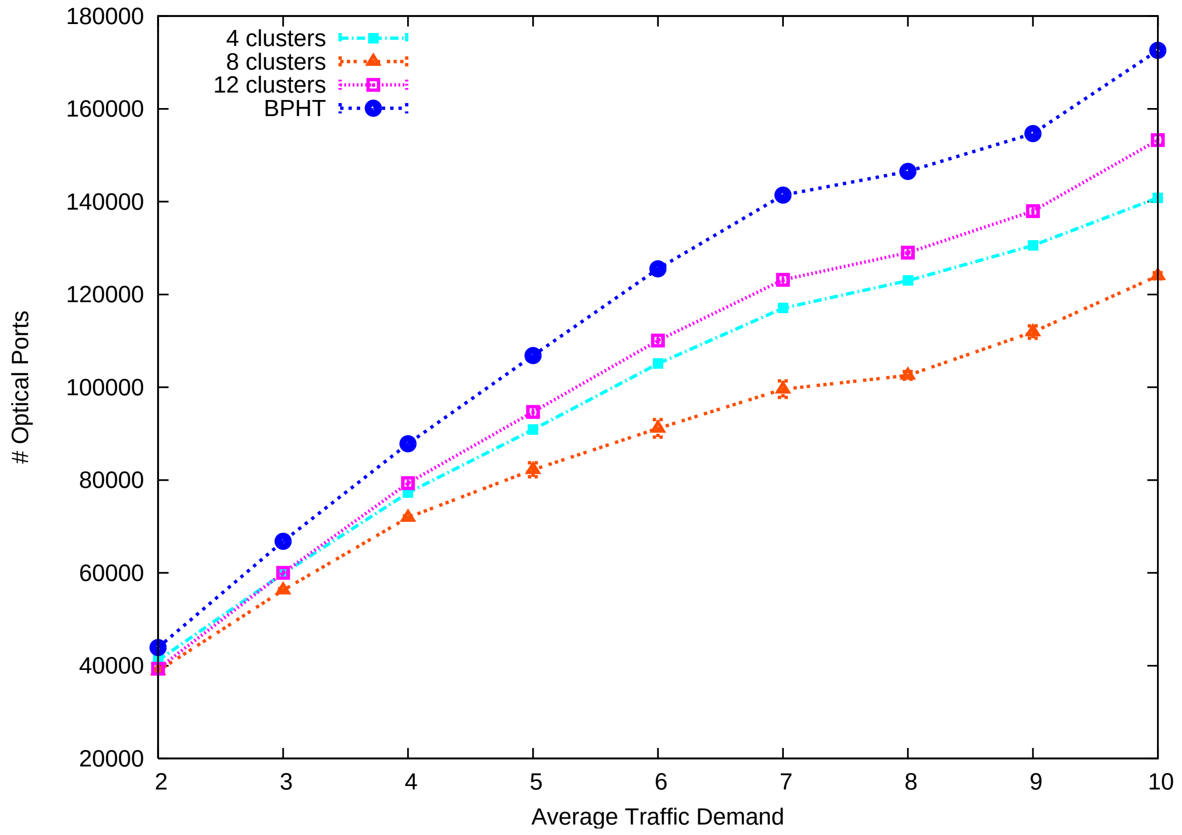


Figure 5.7: Port Cost, Hierarchical Approach and BPHT, Varying Number of Clusters, $L = 8$

5.4.3 Comparison of Different Models

Finally, Figure 5.9 plots the number of optical ports when different models are used for band size $L = 7$ when the network is partitioned into $K = 8$ clusters. We find that the Fine Granularity MG-OXC requires the least number of optical ports; this is expected due to the flexibility it offers, helping us to take advantage of waveband grooming and waveband adds and drops. We note, however, that entire fibers are rarely dropped to (added from) the WXC due to the large amount of inter-cluster bypass traffic at the nodes. The Waterfall Model can add or drop only wavelengths and hence requires more optical ports than Fine Granularity Model. The Bypass Model does not have waveband grooming ability forcing entire fibers to be dropped (added) at either the BXC or the WXC. Fibers are dropped (added) to BXC only if none of the traffic demands in the fiber require wavelength switching, adds, or drops at the concerned node. At all other times they are dropped to the WXC, thus, the Bypass MG-OXC performs worst and uses significantly more ports. Our studies (Figure 5.10) indicate that all models have identical wavelength costs, since the routing and wavelength assignment of traffic demands is

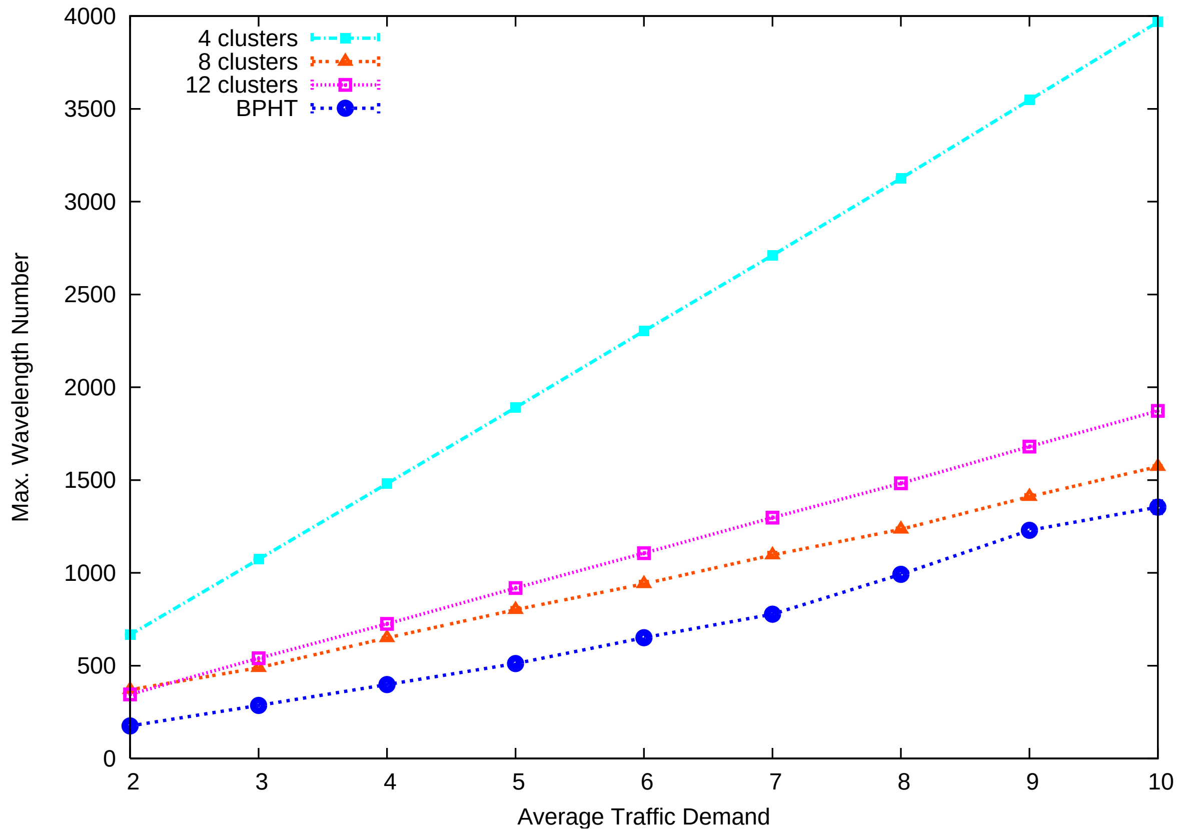


Figure 5.8: Wavelength Cost, Hierarchical Approach and BPHT, Varying Number of Clusters, $L = 8$

independent of the models used in the network.

Overall, these results indicate that the hierarchical algorithm is very scalable, not only in terms of bandsize, but also in terms of the number of clusters considering its utilization of network resources, including optical ports and wavelengths.

5.5 Conclusions

We have presented a hierarchical approach to grooming lightpath traffic and routing wavebands over a multigranular optical network. In our model, the network is partitioned into clusters, and one node in each cluster is designated as the hub. Inter-cluster traffic is routed through the hubs in a manner conducive to the formation of wavebands, thus reducing bypass traffic overhead at the intermediate nodes. Our algorithm easily scales across bandsizes and different cluster sizes, the considering optical port cost and number of wavelengths.

The present solution considers long-term traffic forecasts to minimize the number of ports

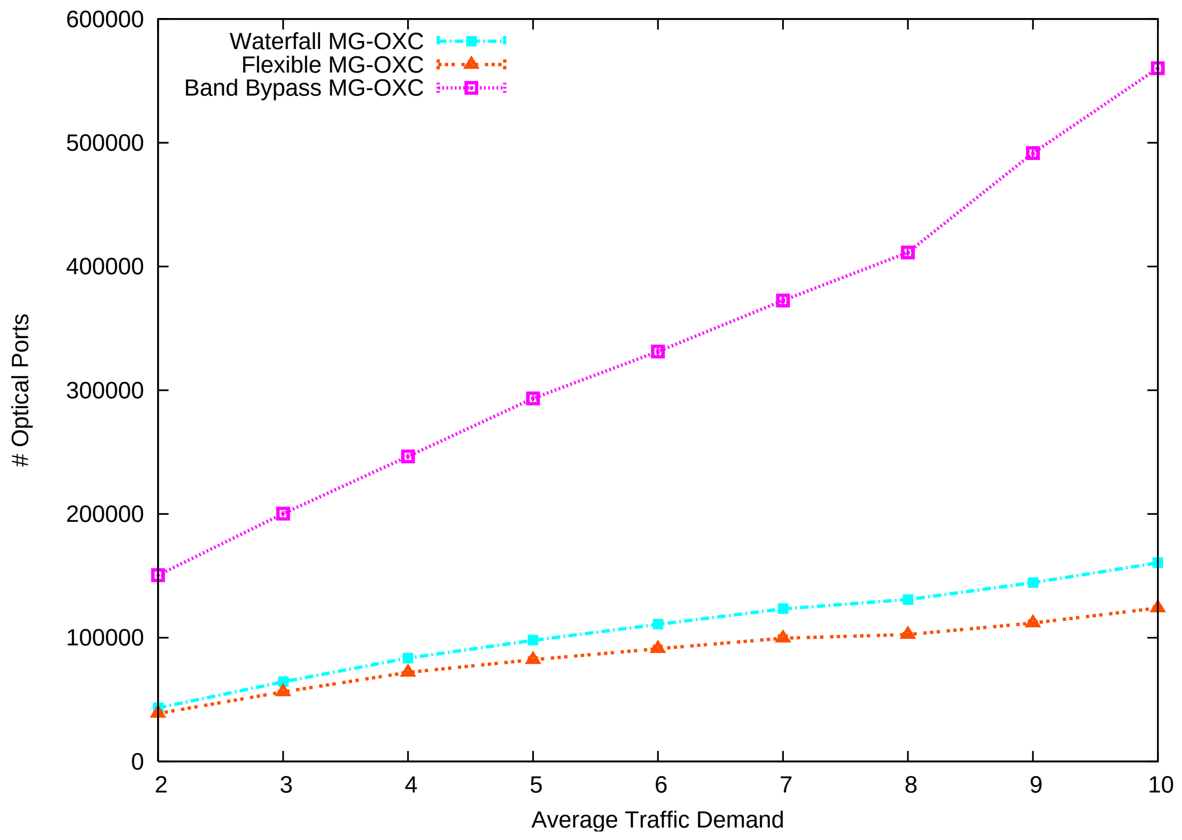


Figure 5.9: Port Cost, Varying Switch Architectures, $K = 8, L = 8$

used over the network. Changes in traffic forecasts cannot be accommodated by the present solution. We have also considered a homogeneous network where all nodes have the same MG-OXC. Our approach needs to be extended to take advantage of heterogeneity, if any is present in the network.

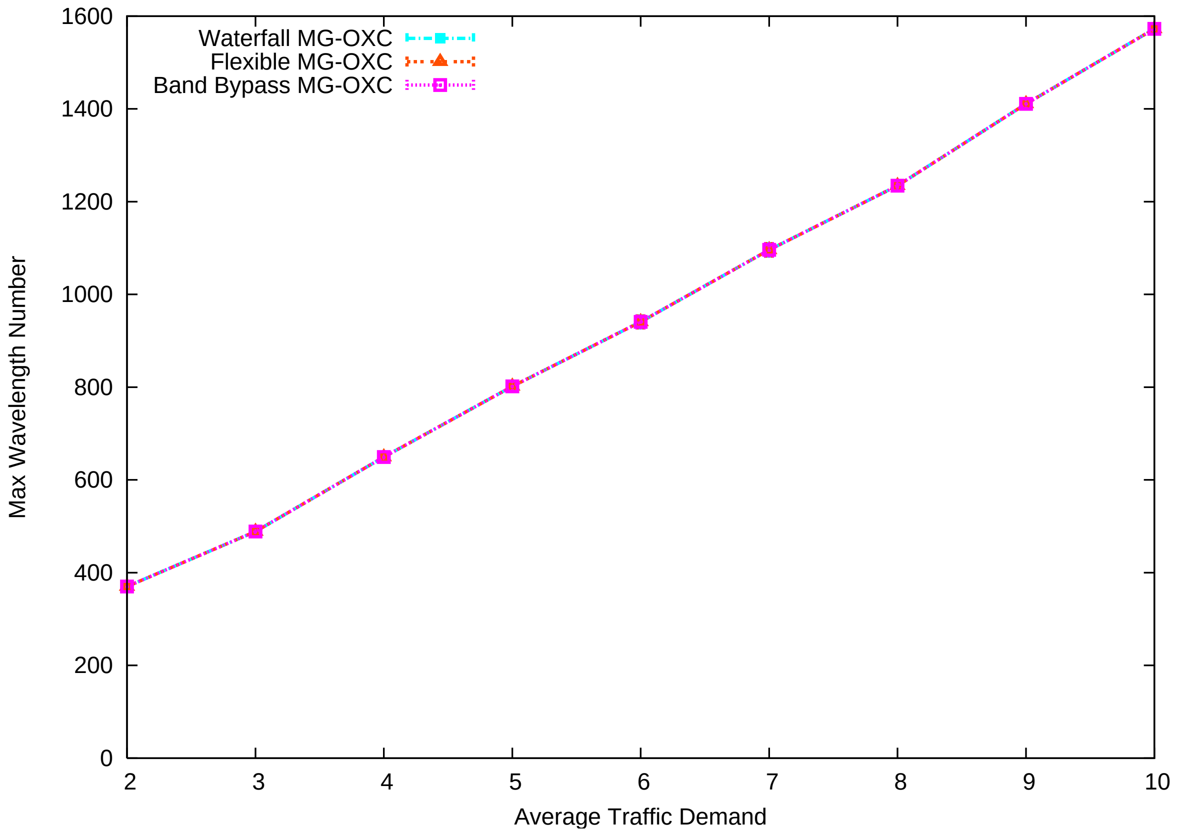


Figure 5.10: Wavelength Cost, Varying Switch Architectures, $K = 8, L = 8$

Chapter 6

Summary and Future Work

In this work, we proposed mechanisms to implement bandwidth on demand services provisioned by a cloud provider, accessible by applications while lowering the cost of delivering these services over the provider's optical network. We minimize the number of optical ports and wavelengths required to satisfy long-term traffic demand estimates using a hierarchical approach. The hierarchical approach is conceptually simple and establishes lightpaths to satisfy traffic demand estimates. Experimental results show that this approach easily scales across different waveband sizes and numbers of clusters in a given network.

We provide cloud applications with mechanisms to request network bandwidth, using the SILO network architecture, without exposing the details of the cloud provider's networking infrastructure to the application. The SILO architecture is used primarily as a result of its ability to provide customized, per-flow network stacks to applications, as well as enabling the easy deployment of new network functions as services by the cloud provider.

Application requests for bandwidth greater than the underlying network speed are accommodated by splitting the high bandwidth application flow into multiple virtual circuits; each virtual circuit operates at the network link speed and is carried over lightpaths established during network design. Flow aggregation is performed using connection points and its associated connection point aware service; implementation of different flow aggregation policies like Round Robin, Priority Scheduling and so on, can be done within the Virtual Concatenation service. Bandwidth modification requests made by applications, are handled by the Link Capacity Adjustment Scheme tuning algorithm; the SILO and Service Manager ensures that the applications do not monopolize available network resources.

Low bandwidth application flows are multiplexed with other flows in a bandwidth proportionate manner into a single virtual circuit, all flows within the virtual circuit originate from the same source and terminate at the same destination. Interface silos are established to service these virtual circuits at system startup, and each virtual circuit uses one network interface and services multiple application flows with shorter lifetimes than itself. The Grooming service can implement different flow aggregation policies, while the Scheduling tuning algorithm accommodates application flows starting at different instances of time. The Grooming service schedules the interface silo to process data into the virtual circuit at periodic time intervals.

6.1 Future Work

The contributions of this thesis are placed primarily within optical network design and end system architectures; however, there are several additional directions in which this work can be extended. These directions are discussed below.

1. *Accommodating Survivability Concerns of Low Bandwidth Flows:* The Virtual Concatenation functionality has a mechanism to handle network failures. However, such protection or recovery mechanisms are presently unavailable for low bandwidth flows. It would be useful to augment the Grooming functionality of the current SILO prototype to provide path-based protection or recovery mechanisms.

2. *Automation of Network Management functions:* Our work for this thesis has assumed that virtual circuits can be dynamically established between data centers and optical networks. In reality, data center networks and optical networks use different control planes that may not be compatible. We would like to investigate mechanisms to automate this process using common control plane technologies to reserve network resources and establish end-to-end virtual circuits between source and destination nodes.

3. *Joint Resource Allocation Mechanisms for Network and Computing Resources:* Current mechanisms for the resource allocation of networking and computing resources operate independently, which can lead to sub-optimal solutions. We would like to devise joint resource allocation mechanisms for applications requesting both computing and networking resources.

4. *Performance Characterizations of the SILO Architecture:* Until now, we have concentrated on designing the architectural capabilities of the SILO architecture. We believe that performance analysis of the SILO components is important, if it is to be accepted by the networking community. We would like to develop system optimization techniques based on a performance analysis of the SILO architecture.

REFERENCES

[1] ITU-T G.709 (01/03). Interfaces for the optical transport network (OTN).

[2] G. P. Agarwal. *Nonlinear Fiber Optics.* Academic Press, third edition, 2001.

[3] Amazon. Web services. http://aws.amazon.com/.

[4] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. *Computer*, 38(4):34 – 41, april 2005.

[5] Muhammad Bilal Anwer and Nick Feamster. Building a fast, virtualized data plane with programmable hardware. *SIGCOMM Comput. Commun. Rev.*, 40(1):75–82, 2010.

[6] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Trans. Netw.*, 5(6):756–769, 1997.

[7] I. Baldine, M. Vellala, Anjing Wang, G. Rouskas, R. Dutta, and D. Stevenson. A unified software architecture to enable cross-layer design in the future internet. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 26 –32, 13-16 2007.

[8] P. Baran. On distributed communications networks. *Communications Systems, IEEE Transactions on*, 12(1):1 –9, march 1964.

[9] Andy Bavier, Mic Bowman, Brent Chun, David Culler, Scott Karlin, Steve Muir, Larry Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. Operating system support for planetary-scale network services. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[10] Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. 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*, pages 3–14, New York, NY, USA, 2006. ACM.

[11] Nina T. Bhatti, Matti A. Hiltunen, Richard D. Schlichting, and Wanda Chiu. Coyote: a system for constructing fine-grain configurable communication services. *ACM Trans. Comput. Syst.*, 16(4):321–366, 1998.

[12] Nina T. Bhatti and Richard D. Schlichting. A system for constructing configurable high-level protocols. In *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 138–150, New York, NY, USA, 1995. ACM.

[13] Robert Braden, Ted Faber, and Mark Handley. From protocol stack to protocol heap: role-based architecture. *SIGCOMM Comput. Commun. Rev.*, 33(1):17–22, 2003.

[14] Kevin Brown and Suresh Singh. M-TCP: TCP for mobile cellular networks. *SIGCOMM Comput. Commun. Rev.*, 27(5):19–43, 1997.

[15] A. Parker. C. Kamath. Mining data for gems of information., 2000.

[16] R. Caceres and L. Iftode. Improving the performance of reliable transport protocols in mobile computing environments. *Selected Areas in Communications, IEEE Journal on*, 13(5):850 –857, jun 1995.

[17] Xiaojun Cao, V. Anand, and Chunming Qiao. A waveband switching architecture and algorithm for dynamic traffic. *Communications Letters, IEEE*, 7(8):397 – 399, aug. 2003.

[18] Xiaojun Cao, V. Anand, and Chunming Qiao. Waveband switching in optical networks. *Communications Magazine, IEEE*, 41(4):105 – 112, april 2003.

[19] Xiaojun Cao, V. Anand, and Chunming Qiao. Multilayer versus single-layer optical cross-connect architectures for waveband switching. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1830 –1840 vol.3, 7-11 2004.

[20] Xiaojun Cao, V. Anand, Yizhi Xiong, and Chunming Qiao. A study of waveband switching with multilayer multigranular optical cross-connects. *Selected Areas in Communications, IEEE Journal on*, 21(7):1081 – 1095, sept. 2003.

[21] Xiaojun Cao, Chunming Qiao, V. Anand, and Jikai LI. Wavelength assignment in waveband switching networks with wavelength conversion. In *Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE*, volume 3, pages 1943 – 1947 Vol.3, 29 2004.

[22] B. Chen, G.N. Rouskas, and R. Dutta. A framework for hierarchical traffic grooming in WDM networks of general topology. In *Broadband Networks, 2005. BroadNets 2005. 2nd International Conference on*, pages 155 – 164 Vol. 1, 3-7 2005.

[23] Bensong Chen, R. Dutta, and G.N. Rouskas. On the application of k-center algorithms to hierarchical traffic grooming. In *Broadband Networks, 2005. BroadNets 2005. 2nd International Conference on*, pages 1218 –1224 Vol. 2, 7-7 2005.

[24] L.-W. Chen, P. Saengudomlert, and E. Modiano. Uniform vs. non-uniform band switching in WDM networks. In *Broadband Networks, 2005. BroadNets 2005. 2nd International Conference on*, pages 204 – 213 Vol. 1, 3-7 2005.

[25] I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: an approach to high bandwidth optical WAN's. *Communications, IEEE Transactions on*, 40(7):1171 –1182, jul 1992.

[26] *CineGrid*. http://www.cinegrid.org/.

[27] Cisco. Data center switches. http://www.cisco.com/en/US/products/ps9441/Products_-Sub_Category_Home.html.

[28] D. Clark. The design philosophy of the DARPA internet protocols. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 106–114, New York, NY, USA, 1988. ACM.

[29] H. Mannila. D. J. Hand, P. Smyth. *Principles of Data Mining.* MIT Press, 2001.

[30] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[31] Tom DeFanti, Cees de Laat, Joe Mambretti, Kees Neggers, and Bill St. Arnaud. Translight: a global-scale lambdagrid for e-science. *Commun. ACM*, 46(11):34–41, 2003.

[32] E. Dotaro and *et al.* Optical multi-granularity architectural framework. draft-dotaro-ipo-multi-granularity-02.txt, 2002.

[33] Constantine Dovrolis. What would darwin think about clean-slate architectures? *SIGCOMM Comput. Commun. Rev.*, 38(1):29–34, 2008.

[34] R. Dutta and G.N. Rouskas. Traffic grooming in WDM networks: past and future. *Network, IEEE*, 16(6):46 – 56, nov/dec 2002.

[35] R. Dutta, G.N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson. The silo architecture for services integration, control, and optimization for the future internet. In *Communications, 2007. ICC '07. IEEE International Conference on*, pages 1899 –1904, 24-28 2007.

[36] R. Callon. E. Rosen, A. Viswanathan. RFC 3031, multiprotocol label switching architecture., 2001.

[37] Jeff Chase et. al. Cloud network infrastructure as a service: An exercise in multi-domain orchestration, March 2010. In submission.

[38] Thomas H. Cormen et. al. *Introduction to Algorithms.* MIT Press, 2009.

[39] G. Karmous-Edwards F. Travostino, J. Mambretti, editor. *Grid Networks: Enabling Grids with Advanced Communication Technology.* John Wiley & Sons Ltd., 2006.

[40] Anja Feldmann. Internet clean-slate design: what and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, 2007.

[41] S. Floyd. Highspeed TCP for large congestion windows, 2003.

[42] Link Aggregation Task Force. 802.1ax link aggregation control protocol. IEEE, 2000. http://www.ieee802.org/3/ad/.

[43] S. Ganguly, A.C. Varsou, and R. Izmailov. Waveband routing on logical topologies constructed over a ring network. In *High Performance Switching and Routing, 2004. HPSR. 2004 Workshop on*, pages 126 – 133, 2004.

[44] O. Gerstel, R. Ramaswami, and W. Wang. Making use of a two stage multiplexing scheme in a WDM network. In *Proceedings of OFC 2000*, page ThD1, 2000.

[45] H.G. Perros. G.N. Rouskas. *A Tutorial on Optical Networks.*, volume 2497 of *Networking 2002 Tutorials*, pages 155–193. Lecture Notes in Computer Science, 2002.

[46] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[47] Google. App engine. http://code.google.com/appengine/.

[48] IEEE 802.1 Working Group. Data center bridging task group.
. http://www.ieee802.org/1/pages/dcbridges.html.

[49] K. Harada, K. Shimizu, T. Kudou, and T. Ozeki. Hierarchical optical path cross-connect systems for large scale wdm networks. In *Optical Fiber Communication Conference, 1999, and the International Conference on Integrated Optics and Optical Fiber Communication. OFC/IOOC '99. Technical Digest*, volume 2, pages 356 –358 vol.2, 1999.

[50] N.C. Hutchinson and L.L. Peterson. The x-kernel: an architecture for implementing network protocols. *Software Engineering, IEEE Transactions on*, 17(1):64 –76, jan 1991.

[51] IEEE. 802.1q - virtual lans., 2007. http://www.ieee802.org/1/pages/802.1Q.html.

[52] Univ. of South. Cal. Information Sciences Institute. RFC 793, transmission control protocol., 1981.

[53] International Telecommunication Union (ITU). ITU-T G.7042, link capacity adjustment scheme (LCAS) for virtually concatenated signals., 2004.

[54] R. Izmailov, S. Ganguly, Y. Suemura, I. Nishioka, Y. Maeno, and S. Araki. Waveband routing in optical networks. In *Proceedings of IEEE ICC 2002*, pages 2727–2733, 2002.

[55] R. Izmailov, Samrat Ganguly, V. Kleptsyn, and A.C. Varsou. Nonuniform waveband hierarchy in hybrid optical networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1344 – 1354 vol.2, 30 2003.

[56] R. Izmailov, A. Kolarov, R. Fan, and S. Araki. Hierarchical optical switching: a node-level analysis. In *High Performance Switching and Routing, 2002. Merging Optical and IP Technologies. Workshop on*, pages 309 – 313, 2002.

[57] Tim Freeman Kate Keahey. Science clouds: Early experiences in cloud computing for scientific applications. In *In Cloud Computing and its Applications (CCA)*, 2008.

[58] D.E. Knuth. *The Art of Computer Programming, Vol. I: Fundamental Algorithms.* Addison-Wesley, 1968.

[59] A. Kolarov and B. Sengupta. An algorithm for waveband routing and wavelength assignment in hierarchical WDM mesh networks. In *High Performance Switching and Routing, 2003, HPSR. Workshop on*, pages 29 – 36, 24-27 2003.

[60] Myungmoon Lee, Jintae Yu, Yongbum Kim, Chul-Hee Kang, and Jinwoo Park. Design of hierarchical crossconnect WDM networks employing a two-stage multiplexing scheme of waveband and wavelength. *Selected Areas in Communications, IEEE Journal on*, 20(1):166 –171, jan 2002.

[61] M. Li, W. Yao, and B. Ramamurthy. Same-destination-intermediate grouping vs. end-to-end grouping for waveband switching in wdm mesh networks. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, volume 3, pages 1807 – 1812 Vol. 3, 16-20 2005.

[62] Yong Liao, Dong Yin, and Lixin Gao. Pdp: parallelizing data plane in virtual network substrate. In *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 9–18, New York, NY, USA, 2009. ACM.

[63] R. Madan, Shuguang Cui, S. Lall, and A. Goldsmith. Cross-layer design for lifetime maximization in interference-limited wireless sensor networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1964 – 1975 vol. 3, 13-17 2005.

[64] Maurizio Portolani. Mauricio Arregoces. *Data Center Fundamentals.* Cisco Press, 2004.

[65] Rean Griffith et. al. Michael Armbrust, Armando Fox. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, University of California, Berkeley, 2009.

[66] Microsoft. Windows azure. http://www.microsoft.com/windowsazure/.

[67] B. Mukherjee. *Optical Communication Networking.* McGraw-Hill, 1997.

[68] Biswanath Mukherjee. *Optical WDM Networks.* Springer, 2006.

[69] ITU Study Group 15 Optical Transport Networks and Technologies. Optical transport network (OTN) tutorial.

[70] L. Noirie, M. Vigoureux, and E. Dotaro. Impact of intermediate traffic grouping on the dimensioning of multi-granularity optical networks. In *Optical Fiber Communication Conference and Exhibit, 2001. OFC 2001*, volume 2, pages TuG3–1 – TuG3–3 vol.2, 2001.

[71] C. (Sam) Ou and B. Mukherjee. *Survivable Optical WDM Networks.* Springer, 2005.

[72] Scuola Superiore. Piero Castoldi. Challenges for enabling cloud computing over optical networks. In *OFC/NFOEC 2009, International Workshop on the Cloud/Grid/Utility Computing over Optical Networks, San Diego, CA.*, 2009.

[73] J. Postel. RFC 768, user datagram protocol., 1981.

[74] Chunming Qiao. Grid/cloud computing over optical networks: Opportunities & research issues. In *OFC/NFOEC 2009, International Workshop on the Cloud/Grid/Utility Computing over Optical Networks, San Diego, CA.*, 2009.

[75] V.T. Raisinghani and S. Iyer. Cross-layer feedback architecture for mobile device protocol stacks. *Communications Magazine, IEEE*, 44(1):85 – 92, jan. 2006.

[76] L.G. Roberts. The evolution of packet switching. *Proceedings of the IEEE*, 66(11):1307 – 1313, nov. 1978.

[77] Richard Mcgovern Roger Lapuh, Dinesh Mohan. Split multi-link trunking (SMLT). Internet Engineering Task Force, 2008. http://tools.ietf.org/id/draft-lapuh-network-smlt-08.txt.

[78] George N. Rouskas. Routing and wavelength assignment in optical WDM networks. In John Proakis, editor, *Wiley Encyclopedia of Telecommunications*. John Wiley & Sons, 2001.

[79] A.A.M. Saleh and J.M. Simmons. Architectural principles of optical regional and metropolitan access networks. *Lightwave Technology, Journal of*, 17(12):2431 –2448, dec 1999.

[80] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *Networking, IEEE/ACM Transactions on*, 4(3):375 –385, jun 1996.

[81] V. Srivastava and M. Motani. Cross-layer design: a survey and the road ahead. *Communications Magazine, IEEE*, 43(12):112 – 119, dec. 2005.

[82] T. E. Stern and K. Bala. *Multiwavelength Optical Networks*. Prentice Hall, 2000.

[83] Dimitrios Stiliadis and Anujan Varma. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *IEEE/ACM Trans. Netw.*, 6(5):611–624, 1998.

[84] Alexander Szalay. Science in the cloud. In *In Cloud Computing and its Applications (CCA)*, 2008.

[85] J.D. Touch and V.K. Pingali. The RNA metaprotocol. In *Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on*, pages 1 –6, 3-7 2008.

[86] Joe Touch. Dynamic internet overlay deployment and management using the x-bone. volume 36, pages 117–135, New York, NY, USA, 2001. Elsevier North-Holland, Inc.

[87] Joe Touch, Yu-shun Wang, Lars Eggert, and Gregory Finn. A virtual internet architecture, 2003.

[88] Sunay Tripathi, Nicolas Droux, Thirumalai Srinivasan, and Kais Belgaied. Crossbow: from hardware virtualized nics to virtualized networks. In *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 53–62, New York, NY, USA, 2009. ACM.

[89] Sunay Tripathi, Nicolas Droux, Thirumalai Srinivasan, Kais Belgaied, and Venu Iyer. Crossbow: a vertically integrated qos stack. In *WREN '09: Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 45–54, New York, NY, USA, 2009. ACM.

[90] Huub van Helvoort. *Next Generation SDH/SONET: Evolution of Revolution?* John Wiley & Sons Ltd., 2005.

[91] Manoj Vellala, Anjing Wang, George Rouskas, Rudra Dutta, Ilia Baldine, and Daniel Stevenson. A composition algorithm for the silo cross-layer optimization service architecture. In *ANTS 2007, In Proceedings of*, 2007.

[92] M.A. Vouk. Cloud computing: Issues, research and implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on*, pages 31 –40, 23-26 2008.

[93] Jiantao Wang, Lun Li, S.H. Low, and J.C. Doyle. Cross-layer optimization in TCP/IP networks. *Networking, IEEE/ACM Transactions on*, 13(3):582 – 595, june 2005.

[94] Yun Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, pages 328 –335, 1999.

[95] WebEx. Network bandwidth whitepaper, 2003. http://www.webex.com/pdf/wp_bandwidth.pdf.

[96] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. Fast TCP: Motivation, architecture, algorithms, performance. *Networking, IEEE/ACM Transactions on*, 14(6):1246 –1259, dec. 2006.

[97] ADVA Optical Networking Whitepaper. Optical networks for cloud and grid computing., 2009. http://www.advaoptical.com/ApplicationNotes/2009/AN_Optical_Networks_-Grid_Cloud.pdf.

[98] VmWare Whitepapers. Virtualization overview.

[99] R. Winter, J.H. Schiller, N. Nikaein, and C. Bonnet. Crosstalk: cross-layer decision support based on global knowledge. *Communications Magazine, IEEE*, 44(1):93 – 99, jan. 2006.

[100] Y. Wu, P.A. Chou, Qian Zhang, K. Jain, Wenwu Zhu, and Sun-Yuan Kung. Network planning in wireless ad hoc networks: a cross-layer approach. *Selected Areas in Communications, IEEE Journal on*, 23(1):136 – 150, jan. 2005.

[101] Lisong Xu, K. Harfoush, and Injong Rhee. Binary increase congestion control (bic) for fast long-distance networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514 – 2524 vol.4, 7-11 2004.

[102] G. Xylomenos, G.C. Polyzos, P. Mahonen, and M. Saaranen. TCP performance issues over wireless links. *Communications Magazine, IEEE*, 39(4):52 –58, apr 2001.

[103] S. Yao and B. Mukherjee. Design of hybrid waveband-switched networks with OEO traffic grooming. In *Optical Fiber Communications Conference, 2003. OFC 2003*, pages 357 – 358 vol.1, 23-28 2003.

[104] Hui Zang and Jason P. Jue. A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks. *Optical Networks Magazine*, 1:47–60, 2000.

APPENDIX

Appendix A

List of Abbreviations

API	Application Programming Interface
App Flow	Application Data Flow
ATM	Asynchronous Transfer Mode
BPHT	Balanced Path, Heavy Traffic
BXC	Waveband Cross Connect
CN	Congestion Notification
CTRL	Control (Virtual Concatenation)
DCBX	Data Center Bridging Capabilities Exchange Protocol
DWDM	Dense Wavelength Division Multiplexing
DXC	Digital Cross Connect
ETS	Enhanced Transmission Selection
FIND	Future Internet Design
FIRE	Future-Generation Internet Architecture
FTP	File Transfer Protocol
FXC	Fiber Cross Connect
GID	Group Identifier (Virtual Concatenation)
IP	Internet Protocol
IPSec	Internet Protocol Security
LCAS	Link Capacity Adjustment Scheme
MFI	Multi Frame Indicator (Virtual Concatenation)
MG-OXC	Multi Granular Optical Cross Connect
MPLS	Multiprotocol Label Switching

MST	Member Status (Virtual Concatenation)
NewArch	Future-Generation Internet Architecture
NIC	Network Interface Card
ORCA-BEN	Open Resource Control Architecture - Breakable Experimental Network
OSPF	Open Shortest Path First
OTN	Optical Transport Network Protocol
OXC	Optical Cross Connect
PFC	Priority-based Flow Control
RBA	Role Based Architecture
RNA	Recursive Network Architecture
RS-Ack	Resequence Acknowledgement (Virtual Concatenation)
RWA	Routing and Wavelength Assignment
SCA	SILO Construction Agent (SILO: Services Integration and controL Optimization)
SILO	Services Integration and controL Optimization
SMA	SILO Management Agent (SILO: Services Integration and controL Optimization)
SOA	Service Oriented Architectures
SONET	Synchronous Optical Network
SQ	Sequence Number (Virtual Concatenation)
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
USTAS	Universe of Services and Tuning Algorithm Storage
VCAT	Virtual Concatenation
VDO	Video Application
VLAN	Virtual Local Area Network
vNIC	Virtual Network Interface Card
VT	Virtual Tributaries
WXC	Wavelength Cross Connect