A Flexible Advance Reservation Model for Multi-Domain WDM Optical Networks

BY

Eric He (Ding He)

B.S., Beijing Institute of Technology, 1994

M.S., Beijing Institute of Technology, 1997

THESIS
Submitted as partial fulfillment of the requirements
of the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2006

Chicago, Illinois

ACKNOWLEDGE

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AAA	Authorization, Authentication and Accounting
AFR	Alternate Fixed Routing
AR	Advance Reservation
AR-PDC	Advance Reservation enabled Photonic Domain Controller
AR-PIN	Advance Reservation enabled Photonic Interdomain Negotiator
BoD	Bandwidth on Demand
CERN	European Center for Nuclear Research
CR-LDP	Constraint-based Routing Label Distribution Protocol
CSPF	Constrained Shortest Path First
DRAGON	Dynamic Resource Allocation via GMPLS Optical Network
DWDM	Dense Wavelength-division Mutiplexing
ERO	Explicit Route Object
FARM	Flexible Advance Reservation Model
FR	Fixed Routing
GLIF	Global Lambda Integrated Facility
GMPLS	Generalized Multi-Protocol Label Switching
IR	Immediate Reservation
LFN	Long Fat Network
LHC	Large Hadron Collider
LLP	Least Load Path routing
LMP	Link Management Protocol
MEMS	Micro-Electro-Mechanical System
NIC	Network Interface Card
OBGP	Optical Border Gateway Protocol
ODIN	Optical Dynamic Intelligent Network
ODPP	On-Demand Parallel Probe

OGSA	Open Grid Service Architecture
OSPF-TE	Open Shortest Path First-Traffic Engineering
OXC	Optical Cross-Connect
QoS	Quality of Service
RBUDP	Reliable Blast User Datagram Protocol
RFORP	Robust Fast Optical Reservation Protocol
RPC	Remote Procedure Call
RSVF-TE	Resource ReSerVation Protocol with Traffic Engineering
RTT	Round Trip Time
RWA	Routing and Wavelength Assignment
SAGE	Scalable Adaptive Graphics Environment
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
STSD	Specified Starting time and Specified Duration
STUD	Specified Starting time and Unspecified Duration
TCP	Transmission Control Protocol
UCLP	User Controlled Light Path
UTSD	Unspecified Starting time and Specified Duration
UTUD	Unspecified Starting time and Unspecified Duration
VPN	Virtual Private Networks
WDM	Wavelength-division Mutiplexing
XML	Extensible Markup Language

SUMMARY

Grid is a computing architecture that consists of distributed clusters of computers interconnected by a network. Grid computing is used in a variety of data and compute-intensive scientific domains such as bioscience, nanotechnology, geoscience, high-energy physics. To facilitate the transportation of enormous (e.g. terabyte-sized) data-sets between Grid clusters, a new type of Grid computing architecture, called the LambdaGrid, has emerged. LambdaGrids are Grids that are interconnected by ultra-high-speed networks that can be directly controlled by applications. Typically the unit of control is a light path (often called a Lambda) in an optical network. In order for data-intensive applications to function efficiently, they need to be able to reserve enough bandwidth, through the allocation of these light paths. This thesis focuses on the problem of efficient scheduling of light paths between Grid clusters. Prior approaches use rigid scheduling schemes- i.e. resources are scheduled in terms of when the resources are needed and for how long. This thesis will show that contrary to obvious expectations, a flexible advance scheduling model can provide better overall resource utilization and user experience than a rigid scheduling scheme.

In this thesis I propose a Flexible Advance Reservation Model (FARM) and describe how to apply this model to the cross-domain lightpath reservation problem by incorporating Routing and Wavelength Assignment algorithms. Next, I present the architecture,

implementation and services of a coordinated Interdomain and Intradomain optical control plane called AR-PIN/PDC, which is capable of flexible advance reservations, and provides web services. The simulation results show that by relaxing the reservation time constraint, the acceptance rate and resource utilization can be improved dramatically. Through simulations, I also analyze the impact of advance reservations on immediate reservations and conclude that both AR and IR requests need admission control algorithms in order to let both types of reservations coexist and use resource properly. The AR-PIN/PDC software has been deployed in an international photonic testbed consisting of four domains. Over the testbed, I measure the components of end-to-end signaling latency during inter-domain reservation and claim processes. The results show that the major latency component during claim processes is the optical switching time and domain level parallelism can effectively reduce the claim latency. And the time slot granularity is the major factor affecting the reservation latency.

CHAPTER 1

INTRODUCTION

The purpose of this thesis research is to provide advanced networking services with massive bandwidth, Quality of Service and advance reservation, to high end applications. Specifically, this thesis seeks approaches of application-driven intra-domain and inter-domain layer 1 lightpath provision, with the capability of advance reservation. In this thesis, I created a Flexible Advance Reservation Model (FARM), applied this model to Routing and Wavelength Assignment (RWA) problem, and designed routing and signaling algorithms to implement a coordinated inter-domain and intra-domain optical control plane with advance reservation capability. Through simulations, I found that admission control of both Advance Reservation (AR) and Immediate Reservation (IR) are necessary in order to maintain a well-balanced AR/IR mixed situation. After the implementation and deployment, I did experiments over an international testbed to measure and analyze the end-to-end lightpath reservation and claim process, and proved the parallelism can effectively reduce the delay.

1.1 The Internet is Not Enough for Advanced Applications

The Internet has been an extremely successful technology innovation. Currently, there are approximately one billion users of the common Internet. The "killer application" that fostered the Internet into a global phenomenon was the World Wide Web. Developed in the late 1980s at the European Center for Nuclear Research (CERN) from research by Tim Berners-Lee, the Web was initially created to share data on nuclear physics. By using hyperlinks and graphical "browsing" technology, the Web greatly simplifies the process of searching for, accessing, and sharing information on the Internet, making it much more accessible to a non-technical audience.

However, advanced applications, represented by Grid applications, are raising network requirements of which the Internet can meet neither at present and even in the foreseeable future. A computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities [Foster01].

"It mainly concerns with coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. The sharing concerned is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. This sharing is, necessarily, highly controlled,

with resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form a virtual organization."

The Grid is a new type of infrastructure that builds upon, abstract and unify the innovations that originally arose from addressing the requirements of large-scale, resource-intensive science and engineering applications. One of the most important characteristics of the Grid is that it is distributed infrastructure. Therefore, from the earliest days of their design and development, Grids have always utilized networking services, especially those based on the Internet technologies. The shared Internet provides best-effort packet-forwarding service, which is not enough for scientific collaboration and Grid applications in several aspects: bandwidth is too low, quality of service is not guaranteed, and bandwidth cannot be reserved in advance. I will give advanced application examples to explain these aspects. (I will illustrate these aspects with an advanced application example.)

1.1.1 Low Bandwidth

The Large Hadron Collider (LHC) is a particle accelerator and collider located at CERN, near Geneva, Switzerland. Currently under construction, the LHC is scheduled to start operation in November 2007, when it will become the world's largest particle accelerator. The current estimates are that the major LHC experiments will store data onto permanent

storage at a raw recording rate of 0.1-1.5 GigaBytes/sec (GB/s). A single copy of the archive is estimated to grow at a rate of 5-8 PetaBytes (PB)/year. [Messina04] This data will be accessed and processed repeatedly by the worldwide collaborations searching for new physics processes.

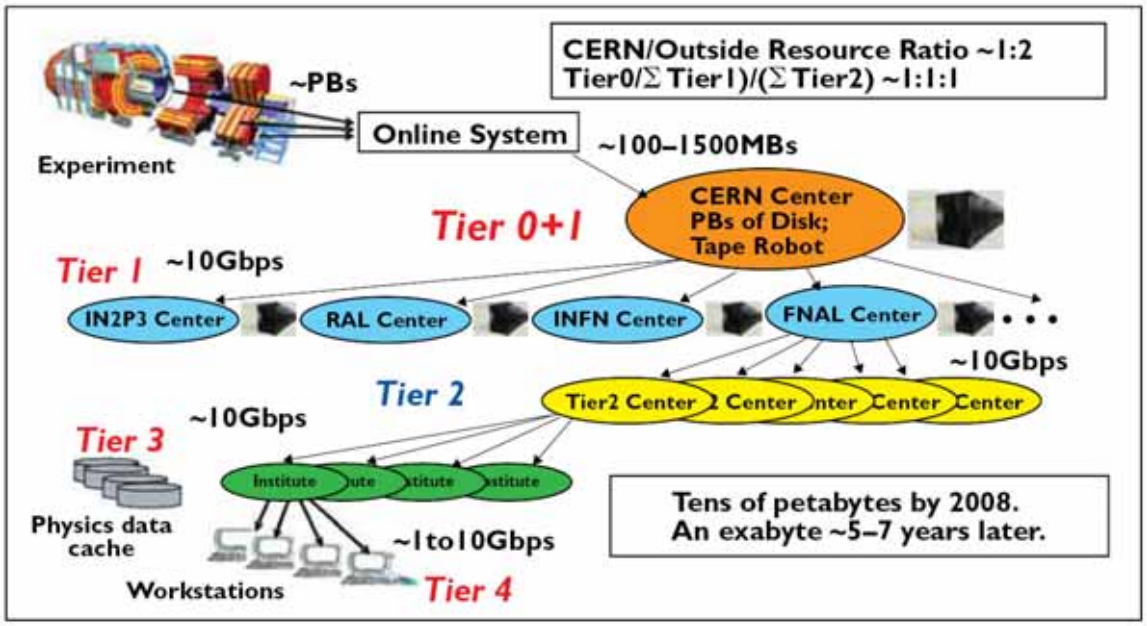

Figure 1-1. The Large Hadron Collider Data Grid Hierarchy

[Source: Harvey Newman, California Institute of Technology]

This Figure demonstrates that in the large hadron collider Data Grid hierarchy, the bandwidth requirements on the networks connecting the "Tier0", "Tier1" and "Tier2" should be in the order of 10Gbps, and the networks connecting the "Tier3" and "Tier4" should be in the order of 1Gbps.

Following initial processing and storage at the "Tier0" facility at the CERN laboratory site, the processed data is distributed through high-speed networks to ~10 to 20 national "Tier1" centers in the United States, leading European countries, Japan, and elsewhere. The data is there (then or The data there is…) further processed and analyzed and then stored at approximately 60 "Tier2" regional centers, each serving a small to medium-sized country, or

one region of a larger country. Data subsets are accessed and further analyzed by physics groups using one of hundreds of “Tier3” workgroup servers and thousands of “Tier4” desktops. [Newman03] As depicted in Figure 1-1, the bandwidth requirements on the networks connecting the “Tier0”, “Tier1” and “Tier2” should be in the order of 10Gbps, and the networks connecting the “Tier3” and “Tier4” should be in the order of 1Gbps. However, the most popular connection type to the Internet is digital subscriber line (DSL), which has only 1-10 Mbps. Even in the next 10 years, the Internet connection speed has little chance to reach 1Gbps. Cees de Laat classified the network users to A, B and C classes (can you say to class A, B and C), as shown in Figure 1-2. [DeLaat03] The first group, class A, includes typical home users with services provided by DSL or cable modems, who may have access at rates around 1Mbps, who use web browsing, emailing, etc. They need full Internet routing. Class B consists of enterprises, universities or Grid-based virtual organizations that operate at gigabit per second LAN speed. The majority of the traffic typically stays within the virtual organization. The class C represents a few hundred truly high-end applications currently being developed, which need transport capacities of multiple gigabits per second for a duration of minutes to hours, originating from a few places, destined for a few other places. Class C traffic often does not require Internet routing. However, it requires dynamic path provision because most of these applications require the gathering and utilization and releasing of resource at multiple sites.

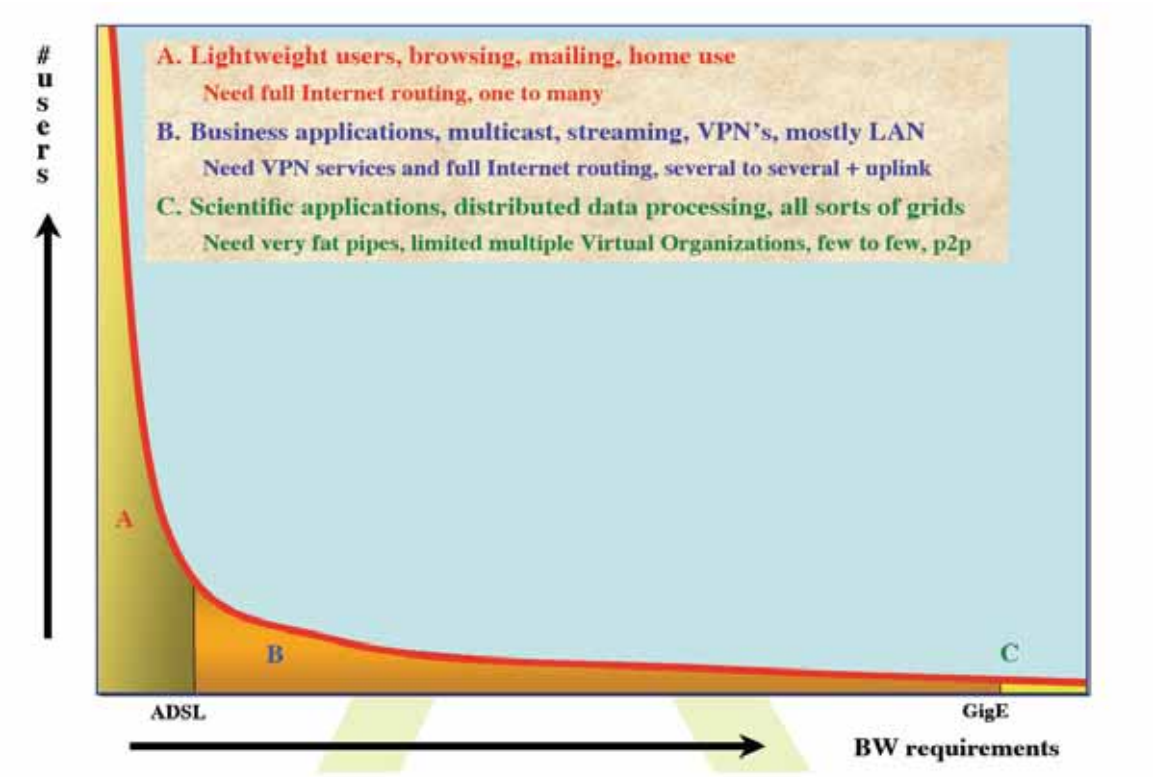

Figure 1-2. Number of Class A, B and C Users Compared with Their Bandwidth Appetite

[Source: Cees de Laat, University of Amsterdam]

This Figure compares the bandwidth and number of users of classes A, B and C users. Most network users belong to class A. They use little bandwidth but need full Internet routing. Very small percentage of users belong to class C, but they use huge amount of bandwidth and usually communicate in few-to-few fashion.

1.1.2 Lack of Quality of Service

As shown in Figure 1-3, Scalable Adaptive Graphics Environment (SAGE) is a specialized middleware for enabling data, high-definition video and extremely high-resolution graphics to be streamed in real-time from remotely distributed rendering and storage clusters to scalable displays over ultra high-speed networks. [Renambot04] Interactive ultra-high-resolution

LambdaGrid visualization applications routinely access remote datasets spanning over multiple terabyte and visualize the rendered pixels on high-resolution displays. The network bandwidth requirements for browsing these datasets or pushing the rendered pixels to remote displays are in the range of several tens to hundreds of gigabits per second. In addition to the huge bandwidth usage, these applications usually create hundreds of bidirectional streams between distant endpoints, each (what is “each”) with differing flow requirements operating over differing transport protocols. Table 1-1 quantifies the broad variety of flows that simultaneously emanate from SAGE. [Wang06]

When the Internet was first deployed many years ago, it lacked the ability to provide Quality of Service (QoS) guarantee because its infrastructure was too limited. It ran at default QoS level, or “best effort”. Integrated services tried to solve the problem by resource reservation. But it failed due to scalability issues. DiffServ took another way of prioritization and aggregation to limit the number of traffic classes in the backbone, but DiffServ (or backbone. However, DiffServ….) also failed because end-to-end QoS was difficult to guarantee by aggregating flows.

Figure 1-3. LambdaVision Driven by SAGE

This Figure shows the SAGE application running on the 55-panel LambdaVision. The LambdaVision is an ultra-high-resolution visualization and networking instrument designed to support collaboration among co-located and remote experts requiring interactive ultra-high-resolution imagery. SAGE enbles data, high-definition video and extremely high-resolution graphics coming from different applications and locations to display simultaneously on LambdaVision.

Type of Flow	Number of Flows	Bandwidth per Flow	Latency Sensitive	Jitter Sensitive	Reliability Requirement	Burstiness	Message Size	Protocol
Audio Stream	1 per user	Low 1Mbps	Yes	Yes	Med	Constant	Small	UDP-based
HD Video Stream	1 per user	Med to High 25Mbps-1.5Gbps	Yes	Yes	Med	Constant	Small to Med	UDP-based
Application Stream	1-100 per application	High 1-2.5Gbps	Yes	Variable	High	Application Dependent	Large	UDP-based
Bulk Data	1 per render node	High	No	No	High	Application Dependent	Large	UDP/TCP-based
Annotations/ Static Content	1-10 per user	Low 1Mbps	No	No	High	One Burst	Small	TCP-based
Control Channel	1 per rendering node + 1 per display	Low 64Kbps	No	Yes	High	Short Burst	Small	TCP-based
Synchronization Channel	1 per rendering node + 1 per display	Low 1Mbps	Yes	Yes	High	Constant	Small	TCP-based
SAGE UI	1 per user	Low 64Kbps	No	No	High	Short Burst	Small	TCP-based
VNC Streams	1 per user	Low 1Mbps	Yes	Yes	High	Small Burst	Small	TCP-based

Table 1-1 Network flows created by Ultra-High resolution Grid visualization applications

1.1.3 Absence of Advance Reservation

Advance reservation is required to guarantee the availability of network resources. The nature of resource reservations in Grid computing is quite different from those of telephone calls. For the latter, their durations are usually not known in advance and hence cannot be planned in advance. In contrast, resource allocations in Grid environments usually require a large number of different types of resources to be acquired simultaneously. Therefore, they have to be reserved in advance, in a manner similar to the reservation of hotels, airlines, and rental cars for vacation travel.

As I said (mentioned) in the last (or previous) section, the current Internet can only provide fairness-based best-effort services. Everyone gets a fair share from the available resources when he/she starts running the networking applications. There is no resource reservation service available in the Internet, let alone advance reservation services.

1.2 All-Optical Networks

Over the past ten years, the network bandwidth has been improved 240X from 155 Mbps to 40 Gbps. This growth rate has been outpacing that of disk speed/capacity and processing power during the same period. This has produced a major technology mismatch, perhaps best illustrated by the fact that most PCs today are sold with Gigabit Ethernet (GigE) interfaces,

even though typically available file transfer speeds across the shared Internet are only 10-20 Mbps. Even with Dense Wavelength-division multiplexing (DWDM) technology enabling immerse amounts of bandwidth on a single pair of fiber strands, today's networks are still clogged up and slow to a crawl. Alternatively, lambdas dedicated to individual researchers create the equivalent of high-occupancy-vehicle expressway lanes, delivering more reliable and predictable network performance. This has been considered the most promising mechanism to meet all these demands from Grid applications. 10 Gbps is common to be carried on each wavelength and 40 Gbps starts emerging. When 32 wavelengths are multiplexed, one fiber can carry more than 1 terabits per second! When the control plane provides the lightpath advance reservation capability, the applications can be guaranteed to achieve certain quality of service in specific time slots. When Grids are powered by dedicated lambdas and the networking is not the system bottleneck any more, we call them Lambda Grids.

The OptIPuter [Smarr03] is a National Science Foundation funded project to interconnect distributed storage, computing and visualization resources using photonic networks at tens of gigabits per second. The main goal of the project is to exploit the trend that network capacity is increasing at a rate far exceeding processor speed, while at the same time plummeting in cost. This allows one to experiment with a new paradigm in distributed computing - where the photonic networks serve as the computer's system bus and compute clusters, taken as a whole,

serve as the peripherals in a potentially planetary-scale computer. I consider photonic networks as all-optical networks comprised of optical fibers and 3D MEMS (Micro-Electro-Mechanical Systems) optical switching devices. There is no translation of photons to electrons in photonic networks, hence I can avoid electronic bottlenecks. MEMS optical switches are controlled by special control software that allows applications to request and acquire end-to-end lightpaths. This special software is called Photonic Domain Controller (PDC), which is one of the topics covered in this thesis.

Increasingly, research organizations are buying dark fiber or wavelengths, and they want to share their resources with each other in a manner similar to how they might share computing resources in Grid environments. A collection of Grid computing resources interconnected by an application-configurable network of lightpaths is called a LambdaGrid [DeFanti03]. This provides data-intensive applications with the necessary deterministic network bandwidth to transport data between grid instruments, high-performance storage systems, compute clusters and visualization systems, which is often needed for real-time interactive scientific exploration. An international virtual organization, GLIF, the Global Lambda Integrated Facility, was established to promote this paradigm [GLIF].

Photonic Interdomain Controller (PIN) is software that allows applications to provision for or (provide) and control multi-domain lightpaths [Yu04]. PIN specializes in the interdomain routing and signaling schemes over heterogeneous optical network domains. In a

multi-domain environment, security management and policy administration are also critical. Our collaborator, at the University of Amsterdam, has done some pioneering research on Authorization, Authentication and Accounting (AAA) and we are leveraging it within PIN software [Oudenaarde05].

1.3 Flexible Advance Reservation

As I mentioned in Section 1.1, advance reservation of lightpath resource is a critical functionality that the LambdaGrid needs to provide. For customers, the major performance parameter of resource reservations is *acceptance rate* or *blocking rate*, which is defined as the ratio of accepted (blocked) reservation requests of all submitted requests. For network operators, the major performance parameter is *resource utilization*, which is related directly to their revenue. In comparison to immediate reservations, *advance reservations* usually degrade the resource utilization and the acceptance rate due to the resource fragmentation [Burchard03]. In order to improve the network performance, fragmentation must be avoided. Allowing flexibility in defining the advance reservations can result in better resource utilization while offering greater convenience to users. In this paper I will examine, through simulations, the degree by which flexibility affects performance.

Incorporating flexible advance reservation into PIN/PDC is not trivial. As PIN/PDC is based on all-optical networks, one main problem that PIN/PDC has to solve is Routing and Wavelength Assignment (RWA). The RWA problem is a NP-hard problem. Usually it can be simplified by decoupling the problem into two sub problems: the routing problem and the wavelength assignment problem. The routing problem can be solved by Fixed Routing, Fixed Alternate Routing, or Adaptive Routing algorithms. Adaptive Routing is considered to be able to achieve the best performance by feeding the wavelength assignment status back to the routing algorithm [Zang00]. The flexibility of advance reservations introduces a new temporal dimension into the resource allocation problem. The wavelength resources along the path have to maintain both wavelength and temporal consistency.

For interdomain distributed control, the addition of a temporal dimension makes the resource state of each domain too large to disseminate to other domains. Therefore, only the relatively static topology summary information of each domain is disseminated to other collaborating domains. The Grid community consists of many Virtual Organization (VO) based collaborations, which means that the resource of each domain is usually not open for all the world, instead, each domain wants to define their own collaborators and individual access policy. I believe that the peer-to-peer publish/subscribe model is more effective in this regard and more scalable for interdomain topology exchange.

The multi-domain lightpath reservation problem is actually one type of meta-scheduling problem. Meta-scheduling can be defined as the act of locating and allocating resources for a job from a collection of distributed resources [Snell00]. The key to meta-scheduling is that the user need not be aware of where the resources are, who owns the resources, or who administers the resources in order to use them. Therefore, the meta-scheduler has to be in charge of probing, selecting and reserving the best set of resources by communicating with a bunch of local schedulers. This same methodology can be applied to the cross-domain lightpath reservation problem. The new version of PIN/PDC with flexible Advance Reservation (AR) capability is called Advance Reservation Photonic Interdomain Negotiator and Photonic Domain Controller (AR-PIN/PDC).

1.4 Contributions

Through the design and implementation of AR-PIN/PDC, I recognized and addressed numerous research problems in the control plane and data plane of optical networking. Specifically, this dissertation makes the following contributions.

- I created a Flexible Advance Reservation Model (FARM), applied this model to Routing and Wavelength Assignment (RWA), and designed algorithms to achieve interdomain and intradomain lightpath advance reservation. A peer-to-peer based

publish/subscription topology model is used to avoid huge amount of state flooding. The On-Demand Parallel Probe algorithm renders the periodic dissemination of time-based resource availability information unnecessary and hence makes the system more scalable.

- Through simulations, I found that by introducing some flexibility on the time parameters of advance reservations, the network performance can be improved dramatically. Also it is confirmed that both Advance Reservation (AR) and Immediate Reservation (IR) admission control are necessary in order to maintain a well-balanced AR/IR mixed environment.

- I implemented the fore-mentioned algorithms in the software AR-PIN/PDC. As a set of services, AR-PIN/PDC can be easily deployed in JBoss application server. Because it provides standard web service interfaces, writing clients is very easy in most platforms and environments.

- I deployed AR-PIN/PDC in four domains in US and Europe, making scheduling cross-continent lightpaths possible. In this testbed, I measured and analyzed the components in the end-to-end lightpath reservation and claim, and proved the parallelism of probing and claiming effectively reduces the delay, and the time slot granularity is the major factor affecting the computation time.

- Reliable Blast UDP (RBUDP) protocol was designed and implemented. This protocol is a very aggressive protocol designed for dedicated or QoS-enabled high bandwidth networks such as optical networks. For large bulk transfers, RBUDP can provide delivery at precise, user-specified sending rates. I provided an analytical model that provides a good prediction of RBUDP performance.

1.5 Organization

The remainder of the dissertation is organized as follows. In chapter 2, I describe related work. In chapter 3, a unified Flexible Advance Reservation Model (FARM) is described and I will show how to implement the model for the meta-scheduling problem. In chapter 4, I will elaborate on the architecture of AR-PDC and AR-PIN, especially the interdomain routing and signaling processes, algorithms and their functions. In chapter 5, comprehensive simulation results are shown on how flexibility improves network performance and the impact of advance reservations on immediate reservations. In chapter 6, I will describe the web services that AR-PIN/PDC will provide, data structures and compare two distributed web service modes. In chapter 7, several series of experiments were run to analyze the end-to-end signaling latency for inter-domain reservations and claims. In chapter 8, I will describe a high performance data transport protocol – RBUDP – which is suitable for transporting data over photonic networks. Then the dissertation is concluded by chapter 9.

CHAPTER 2

RELATED WORK

In this chapter, I will review previous work and literature in several aspects related to this dissertation: routing and wavelength assignment, interdomain routing and signaling, advance reservation, and deployed optical control planes.

2.1 Routing and Wavelength Assignment (RWA)

Most RWA approaches in wavelength-routed optical WDM networks have been reviewed and compared in (in what） [Zang00]. Because the combined routing and wavelength assignment are hard problems, all RWA approaches divided the entire problem into two subproblems: routing subproblem and wavelength assignment subproblem. Routing subproblem can be solved by Fixed Routing, Fixed-Alternate Routing, and Adaptive Routing. Fixed routing is the simplest but usually leads to high blocking probabilities. Fixed-alternate routing and adaptive routing provide significant benefits over fixed-shortest-path routing in terms of resource utilization and blocking rate. Of the eleven wavelength assignment heuristics summarized in Zang's paper, only four of them are for single fiber situations.

1. Random Wavelength Assignment (R)

This scheme first searches the space of wavelengths to determine the set of all wavelengths that are available on the required route. Among the available wavelengths, one is chosen randomly.

2. First Fit (FF)

The first available wavelength is selected. This scheme performs well in terms of blocking probability and fairness, and is preferred in practice because of its small computational overhead and low complexity.

3. Least-Used (LU)/SPREAD

LU selects the wavelength that is the least used in the network, thereby attempting to balance the load among all the wavelengths. The performance of LU is worse than Random.

4. Most-Used (MU)/PACK

MU is the opposite of LU in which it attempts to select the most-used wavelength in the network. It outperforms LU significantly.

In these methods, First-Fit is used mostly because of the algorithmic simplicity and good performance.

In [Chu04], upon the arrival of a lightpath request, if there is any link in the selected route which currently has no free wavelength, I can not set up the lightpath on this route. Otherwise, I should first try to find a common free wavelength on all the links along the

selected path. If there is no common free wavelength, I then check whether wavelength converters can help. A lightpath is divided into several segments by the intermediate OXCs which currently have free converters. Each segment still suffers the wavelength continuity constraint. A lightpath can be setup successfully if and only if every segment has common free wavelength(s). For each link, the first-fit wavelength assignment scheme is used.

[Yang05] designed a hybrid weighted shortest path first (HW-SPF) heuristic to find the route. Then the Fragmentation or Trace-back regenerator allocation strategy is used on the resultant route. The first-fit wavelength assignment is used by the regenerator allocation strategy. The HW-SPF heuristic is adapted to accept advance reservation requests in this thesis.

2.2 Interdomain Routing and Signaling

OBGP [Francisco01] extended the BGP routing protocol to support interdomain lightpath setup and management. Connectivity information in BGP is propagated through UPDATE messages. Each OBGP speaker contains complete autonomous system (AS) path information to reach a particular network. Signaling is also implemented through OBGP protocol. However, OBGP assumes that each OXC has the capability to convert wavelengths. OBGP also didn't consider the effect of physical layer impairments. These factors don't prevent us from using OBGP as a reachability disseminating protocol.

[Jukan04] used flooding-based protocol to transfer the user path request to the destination through all possible paths. The state vector composed by service-specific path quality attributes, such as physical layer impairments, reliability, policy, and traffic conditions, is updated at each hop. The flooding is stopped when the service cannot be met. If more than one messages arrive at the destination, the best path is selected based on some criteria. The signaling overhead is huge and increases exponentially with the number of hops.

In [Yang04], a domain gateway uses a local routing scheme to compute alternate local routes between itself and each interior node in the same domain as well as between itself and each neighboring domain gateway. Then a next-hop computation function is used to join the alternate local routes of this domain to the alternate routes of adjacent domains to form the next-hop interfaces leading to desired destinations. Finally, a hop-by-hop lightpath selection function uses the obtained local and next-hop routing information to establish interdomain end-to-end lightpaths.

2.3 Advance Reservation

Advance reservation has been widely studied in networks other than all-optical networks. Guerin and Orda [Guerin00] investigated the computational complexity of routing algorithms when supporting different models of advance reservations. Greenberg et al. [Greenberg99] proposed a call admission control algorithm that occasionally allows a call in progress to be

interrupted in order to efficiently share resources among book-ahead (BA) calls and non-BA calls. The Globus Architecture for Reservation and Allocation (GARA) is a toolkit used to implement advance reservations of grid resources in Globus software [Foster99, Curti05]. The performance issues of applying advance reservations to meta-scheduling problem have been examined by Snell et al. [Snell00].

[Zheng02] analyzed RWA algorithms for three types of advance reservations of lightpaths: Specified Starting Time and Specified Duration (STSD); Specified starting Time Unspecified Duration (STUD); Unspecified starting Time Specified Duration (UTSD).

2.4 Deployed Optical Control Planes

The UCLP (User Controlled Light Path) [Boutaba04, Wu05] software allows end users to self provision and dynamically reconfigure optical (layer one) networks within a single domain or across multiple independent management domains. Sometimes this is also referred to as user controlled traffic engineering. Users can also create daughter optical VPNs (Virtual Private Networks) and hand off control and management of these VPNs to other users. The UCLP software is designed to allow end users to create their own discipline or application specific IP network, particularly in support for high end grid applications. More importantly these networks can be dynamically reconfigured at any time without getting permission or signaling the optical network manager. The UCLP web services software is based on the Open

Grid Service Architecture (OGSA) using Globus Toolkit 3 and Java/Jini services. UCLP is now deployed across CA*net 4 networks.

The ODIN (Optical Dynamic Intelligent Network) service [Mambretti03, Mambretti06] is software being designed and developed iCAIR as an intermediary between high-performance distributed global applications and lower level network service layers. ODIN provides a single point of control for a defined set of network requests within a single administrative domain. This point of control is incorporated within a process that resides on a control server. The process has a complete “understanding” of the topology and current resource allocations within the administrative domain. ODIN was deployed on the OMNInet testbed.

Bandwidth on Demand (BoD) [Gommans03, Gommans06] service provides a QoS path based on Generic Authorization, Authentication, Accounting (AAA) towards a multi domain solution. As each administrative domain implements the authorization of its resources to an AAA server, more than one AAA server needs to communicate by means by AAA requests in order to authorize a QoS path. For each type of AAA request there exists a corresponding Driving Policy that instructs the AAA server how to deal with the request. Concrete resources are controlled by Application Specific Module (ASM), whereas generic actions are delegated to the generic part of an AAA server.

There are some other systems geared on provision circuit-switched end-to-end paths. [Veeraraghavan03, Veeraraghavan06] All of the aforementioned work assumes the physical

network is based on SONET or Ethernet segments and therefore do not incorporate RWA algorithms.

Generalized Multi-Protocol Label Switching (GMPLS) is a well-accepted control plane in optical networking industry. It extends MPLS to provide the control plane (signaling and routing) for devices that switch in any of these domains: packet, time, wavelength and fiber. The suite of GMPLS protocols consists of three separate protocols: OSPF-TE, RSVP-TE/CR-LDP, and LMP.

The routing protocol in GMPLS is usually Open Shortest Path First-Traffic Engineeting (OSPF-TE). [Kompella05] OSPF determines the shortest path from a source node to a destination node. The traffic engineering extensions provide a way of describing the traffic engineering topology (including bandwidth and administrative constraints) and distributing this information within a given OSPF area. In OSPF-TE, the route is usually computed at the source using Constrained Shortest Path First (CSPF).

Once the path is computed with OSPF-TE, the next thing is to establish the forwarding state along the path, as well as possibly to reserve resources along the path. There are two possible mechanisms to accomplish this: RSVP-TE(Resource ReSerVation Protocol with Traffic Engineering) and CR-LDP (Constraint-based Routing Label Distribution Protocol). [Ashwood03, Berger03] Both protocols use Explicit Route Object (ERO) to forward the path information from the source to other nodes along the explicit route.

Because data channels and control channels are separated in GMPLS, a mechanism is required to manage the data links, both in terms of link provision and fault management. This is accomplish by Link Management Protocol (LMP). [Lang05]

GMPLS is not designed for peer-to-peer networking architecture and more like a device management and control protocol for telco-provided networks. It is very difficult to incorporate multi-domain, advance reservation and Authentication, Authorization and Accounting (AAA) functions into it. Besides, GMPLS is a complicated suite of protocol. Implementing GMPLS is not trivial. Calient sells their photonic switches with GMPLS three times more expensive than those without GMPLS.

Robust Fast Optical Reservation Protocol (RFORP) [Yu04] discovers wavelength availability hop by hop along the pre-selected route. To minimize wavelength discovery failure, RFORP optimizes the use of wavelength conversion during discovery. When an OXC along the selected lightpath route does not have the common available wavelengths as the upstream OXCs, the wavelength discovery request will be rollback to the neighboring upstream OXCs to check if wavelength conversion could be deployed to resolve the wavelength-blocking problem. It attempts to recover from wavelength fails when the rollback request propagates back to the source OXC that is also non-converting. RFORP minimizes end-to-end reservation delay by employing parallel concurrent reservation. RFORP assumes that border switches are OEO switches and there is no wavelength continuity constraint between domains, and

therefore only considered RWA algorithms within domains. This assumption is not necessary valid, I extend the wavelength continuity constraint beyond domain borders in this paper.

The Dragon Project (Dynamic Resource Allocation via GMPLS Optical Networks) is trying to build an inter-domain lightpath resource management system and leverage the GMPLS protocol as the intra-domain control plane [Yang06, Lehman06]. Generally they use GMPLS as the intra-domain control plane. For those devices who can not talk GMPLS, a Virtual Label Swapping Router (VLSR) is employed to bridge GMPLS protocols and SNMP/TL1/CLI transactions. Between domains, they use Network Aware Resource Broker (NARB) to provide inter-domain service capability propagation, compute ERO using source routing, performs request authorization and book ahead reservations. In the process of state exchange, the topology or topology summary, Label Switching Path (LSP) reservation information, and AAA policy information of each domain will be disseminated to all other domains. This puts a huge amount of load on the control plane network which usually has relatively low bandwidth.

2.5 Comparison

In this dissertation, I describe a coordinated intra-domain and inter-domain control plane, taking into account both cross-domain RWA and flexible advance reservation. I propose a publish/subscribe route advertising model and On-Demand Parallel Probe (ODPP) algorithm to achieve the scalability of inter-domain information dissemination. The intra-domain control plane can work on not only GMPLS-enabled switches, but also bare MEMS switches that can

only talk TL1 language. In Table 2-1, I listed major related research and compared them to my research.

Through simulations, I found that flexibility in advance reservations can improve performance dramatically. I also explored the impact of introducing advance reservations schemes into an immediate reservation system. Through experiments, I found that the parallel probing and parallel claiming effectively reduce the end-to-end signaling time.

	RWA	Circuit-Switched Control Plane	Advance Reservation	Flexible AR	Multiple Domain	Deploy in Real Testbeds
[Yang04, 05]	X	X			X	
[Zheng02]	X		X	X		
GARA			X			X
UCLP		X			X	X
ODIN	X	X				X
BoD		X			X	X
RFORP	X	X				X
GMPLS		X				X
DRAGON		X	X		X	X
AR-PIN/PDC	X	X	X	X	X	X

Table 2-1 Feature comparison of related research to this thesis.

CHAPTER 3

FLEXIBLE ADVANCE RESERVATION MODEL

3.1 Flexible Advance Reservation Model (FARM)

I assume that Immediate Reservation (IR) requests use resources immediately upon arrival if they are admitted, without announcing their holding times. In contrast, Advance Reservation (AR) requests specify clearly a start time and an end time (or a holding time). The AR request holding time is usually an estimate or a safe upper bound. The resource used by this request will be made available to other requests when the customer finishes his/her job or the holding time expires, whichever happens first.

The specification of an Advance Reservation (AR) request consists of two types of parameters: time-related parameters and resource-related parameters. For fixed advance reservations, time-related parameters include reservation start time "t_{start}" and reservation end time "t_{stop}".

I believe that if I give some flexibility on the time parameters, the call acceptance rate will be improved. This is because flexibility tends to aggregate the reservations together thereby reducing the effect of fragmentation, and in turn enhancing resource utilization. This

hypothesis will be proven in the simulations in chapter 5. Zheng and Mouftah [Zheng02] classified advance reservations into three types: specified starting time and specified duration (STSD); specified starting time and unspecified duration (STUD); and unspecified starting time and specified duration (UTSD). Different wavelength assignment algorithms are used for each request type. There is, however, another possible condition where both starting time and duration are unspecified and only a time window is specified with an earliest time and latest time (UTUD). I wish (hope) to use this to express the notion of flexibility because all the other three reservation types can be expressed as UTUD with some constraints such as the earliest time or the longest time. It is possible the RWA algorithm will find that there are many candidate solutions. I want to put a limit on the maximum number of returned candidate solutions. Also we need to specify the criteria by which the resource agent can select the best candidate. The criteria could be the earliest or the longest.

The resource-related parameters are dependent on the type of the resource. In this paper, now that I am focusing on layer 1 lightpaths, the parameters should include a source node and a destination node. In all-optical circuit switching networks, the bandwidth granularity is a wavelength. A request which needs multiple wavelengths can be decomposed into multiple requests wherein each request provisions a single wavelength. Therefore, in my scheme, I consider only single wavelength reservations.

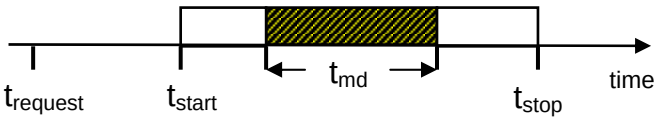

Figure 3-1. The Specification of Flexible Advance Reservations

This Figure shows that relation and meaning of the time-related parameters in flexible advance reservation specification. t_{start} is the earliest time, t_{stop} is the latest time, t_{md} is minimum duration.

Therefore, a flexible advance reservation is defined as follows:

$$R = (s, d, t_{start}, t_{stop}, t_{md}, c) \tag{3-1}$$

where s is the source node, d is the destination node, t_{start} is the earliest time, t_{stop} is the latest time, t_{md} is minimum duration, and c is the selection criteria. The time related parameters are shown in Figure 3-1.

Flexibility can also improve the user efficiency and satisfaction. For example, for fixed reservations, a user can only get the answer yes or no for a proposed reservation. When flexibility is introduced, the local resource manager will search a wider range of time slots when resources are available. This eliminates the need for the user to begin the process over again with another new proposed reservation. I will show how the flexibility improves the call acceptance rate by simulations in chapter 5.

3.2 FARM in Meta-scheduler

Meta-scheduling is the process that a super scheduler schedules resources across multiple sites by negotiating with corresponding local resource managers. [Weissman98] Usually the advance reservation in meta-scheduling is implemented by co-reservation. Figure 3-2 illustrates the steps to apply flexible advance reservations to meta-scheduling. These are:

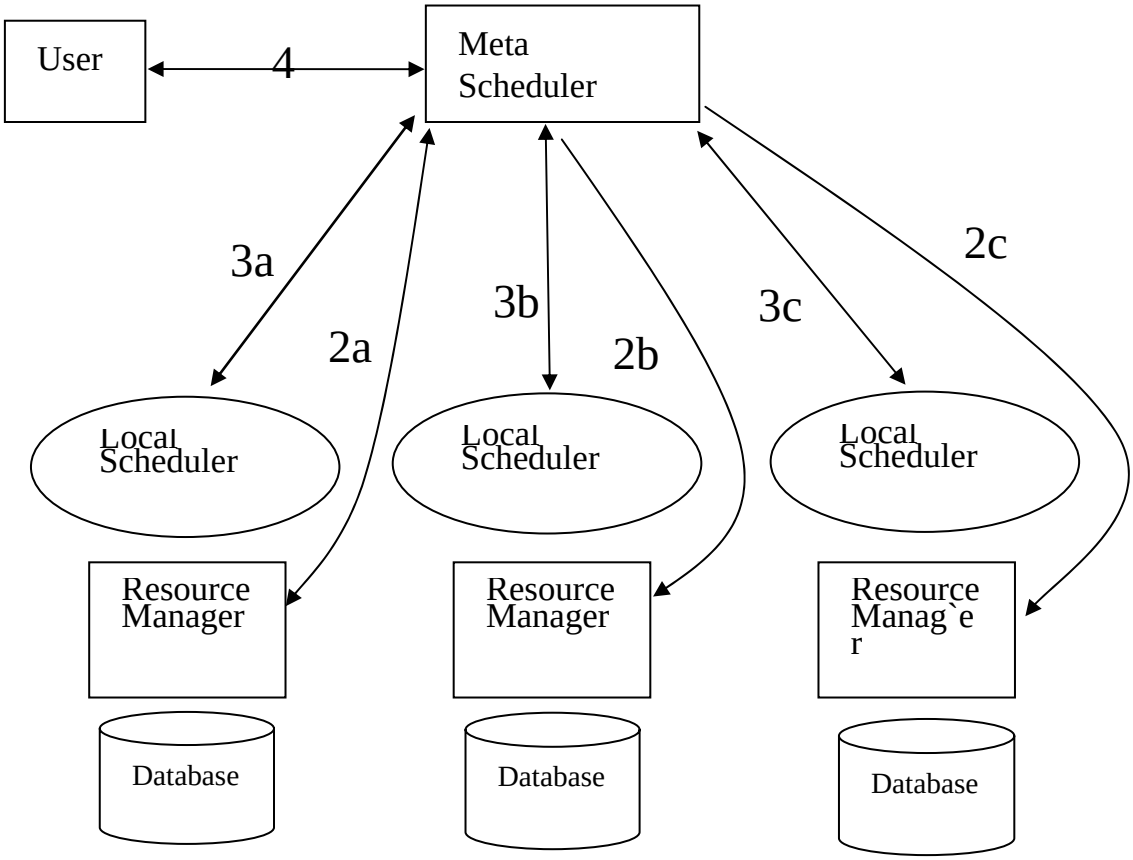

Figure 3-2. Apply FARM to Meta-Scheduler

Meta-scheduling is the process that a super scheduler schedules resources across multiple sites by negotiating with corresponding local resource managers. Usually the advance reservation in meta-scheduling is implemented by co-reservation. This Figure illustrates the steps to apply flexible advance reservations to meta-scheduling.

1). The user submits a flexible advance reservation request to the meta-scheduler. The meta-scheduler analyzes the request, computes a set of resources which can satisfy the user's request.

2). The meta-scheduler decomposes the original request into sub-requests and send them to their local resource managers. The resource managers fetch the resource availability information during the flexible time range from their scheduling database and sends it back to the meta-scheduler.

3). The meta-scheduler collects all the resource availability information. It ("It" refers to Information?) then determines the range of time that is common for all the resources to meet the original request. If there are more than one of them, the best one will be selected based on the user specified criteria. At this point, the flexible time parameters of the original request are fixed. Then the meta-scheduler sends the fixed advance reservation requests to all involved local schedulers to reserve the needed resources. If any reservation attempt fails, the meta-scheduler releases all existing reservations for this job.

4). If all involved local schedulers returns success to the meta-scheduler, then the meta-scheduler returns success to the user and sends back the reservation handle.

Cross-domain lightpath reservation is similar to meta-scheduling of multiple resources. However, the nature of wavelength resource makes the problem more complicated. The multiple domain all-optical lightpath reservation has one more constraint: wavelength

continuity. Therefore, both meta-scheduler and local schedulers have to maintain time continuity as well as wavelength continuity. I will discuss how I solved the problem and the algorithms in detail in chapter 4.

The flexibility can improve the call acceptance rate of local resource managers. The improvement is even more important when a meta-scheduler wants to co-reserve multiple resources from independent resource managers for the same period of time. For example, if the blocking rate of each resource is 0.05, then the blocking rate of meta-scheduling of 10 independent resources is $1-(0.95^{10}) = 0.401$. This high blocking rate is intolerable in most cases. If the blocking rate of each resource can be improved to 0.01 through flexible advance reservations, then the blocking rate of meta-scheduling can be improved to $1 - (0.99^{10}) = 0.096$, This blocking rate is acceptable. This makes it possible to schedule large scale scientific collaborations involving a lot of distributed resources. In other words, if the maximum blocking rate we can tolerate is 10%, then in the case of meta-scheduling of 10 independent resources, then the blocking rate of each individual local resource is required to be lower than 1%, assuming all resources having the same blocking rate. The relation of blocking rate of meta-scheduler and individual scheduler is depicted in Figure 3-3. Therefore, improvement brought by the flexible advance reservations has significant impact on meta-scheduling, also in cross-domain lightpath scheduling.

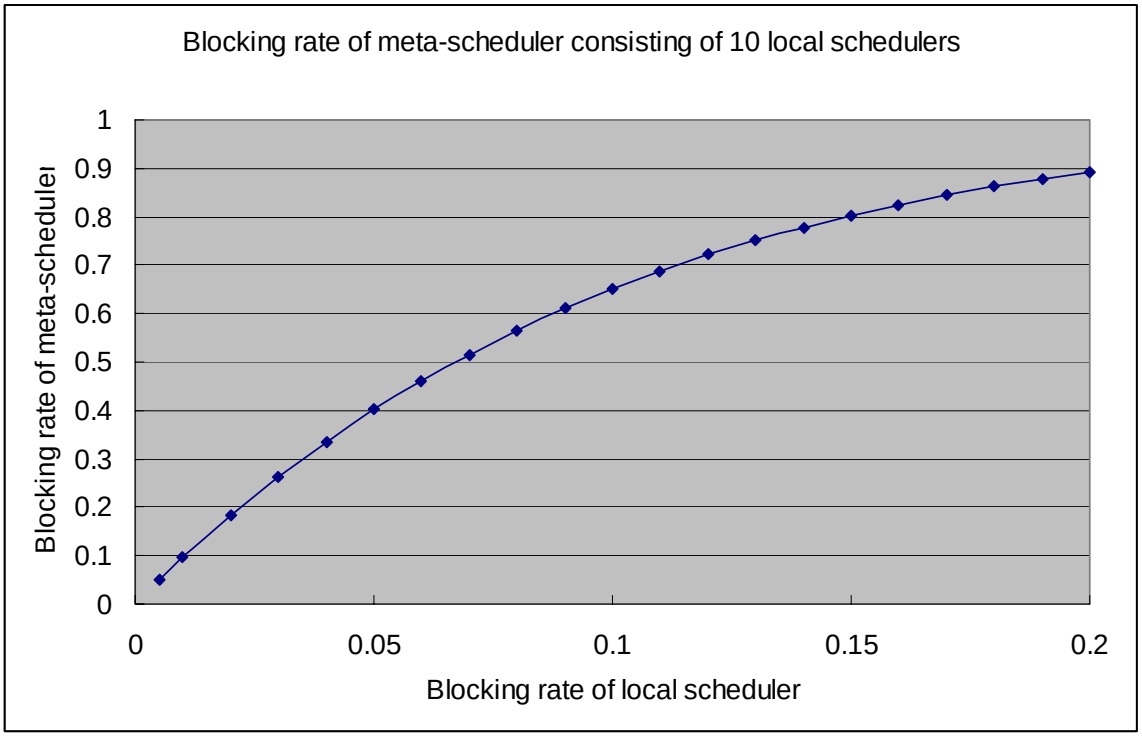

Figure 3-3. The Relation of Blocking Rate of Meta-Scheduler and Individual Local Schedulers

This Figure shows the relation of blocking rate of meta-scheduler having 10 resources and individual scheduler, assuming all resources having the same blocking rate. For example, if the maximum blocking rate we can tolerate is 10%, then in the case of meta-scheduling of 10 independent resources, then the blocking rate of each individual local resource is required to be lower than 1%. Therefore, improvement brought by the flexible advance reservations has significant impact on meta-scheduling..

CHAPTER 4

COORDINATED INTRADOMAIN AND INTERDOMAIN CONTROL PLANE

AR-PIN and AR-PDC are interdomain and intradomain lightpath control software that work together to enable advance reservations for end-to-end interdomain lightpaths.

The system architecture is shown in Figure 4-1. I use an example to show the sequence of interactions between users, AR-PIN and AR-PDC. The following steps will be executed when client A in domain “1” sends a reservation request to the AR-PIN/AR-PDC system:

Periodically, or based on topology changes, the collaborating domains exchange topology summary with each other.

1. Client A sends a lightpath reservation request to its local interdomain agent AR-PIN1.

2. AR-PIN1 computes the domain-level paths.

3. The source domain queries resource availability from each AR-PDC on the domain-level path.

4. Each queried AR-PDC checks its own AAA policy, resource database, then returns the timeslot-wavelength availability matrix.

5. All the returned timeslot-wavelength availability matrices are intersected at AR-PIN1. Based on the result, the best switch-level path is selected. Then the reservations of all involved domains are performed in parallel.

6. Within the reservation time window, the lightpath provision is triggered by committing the reservation. To do that, the device drivers send TL1 commands to switches to set up the end-to-end lightpath.

Next I will explain each major component of AR-PIN and AR-PDC in detail.

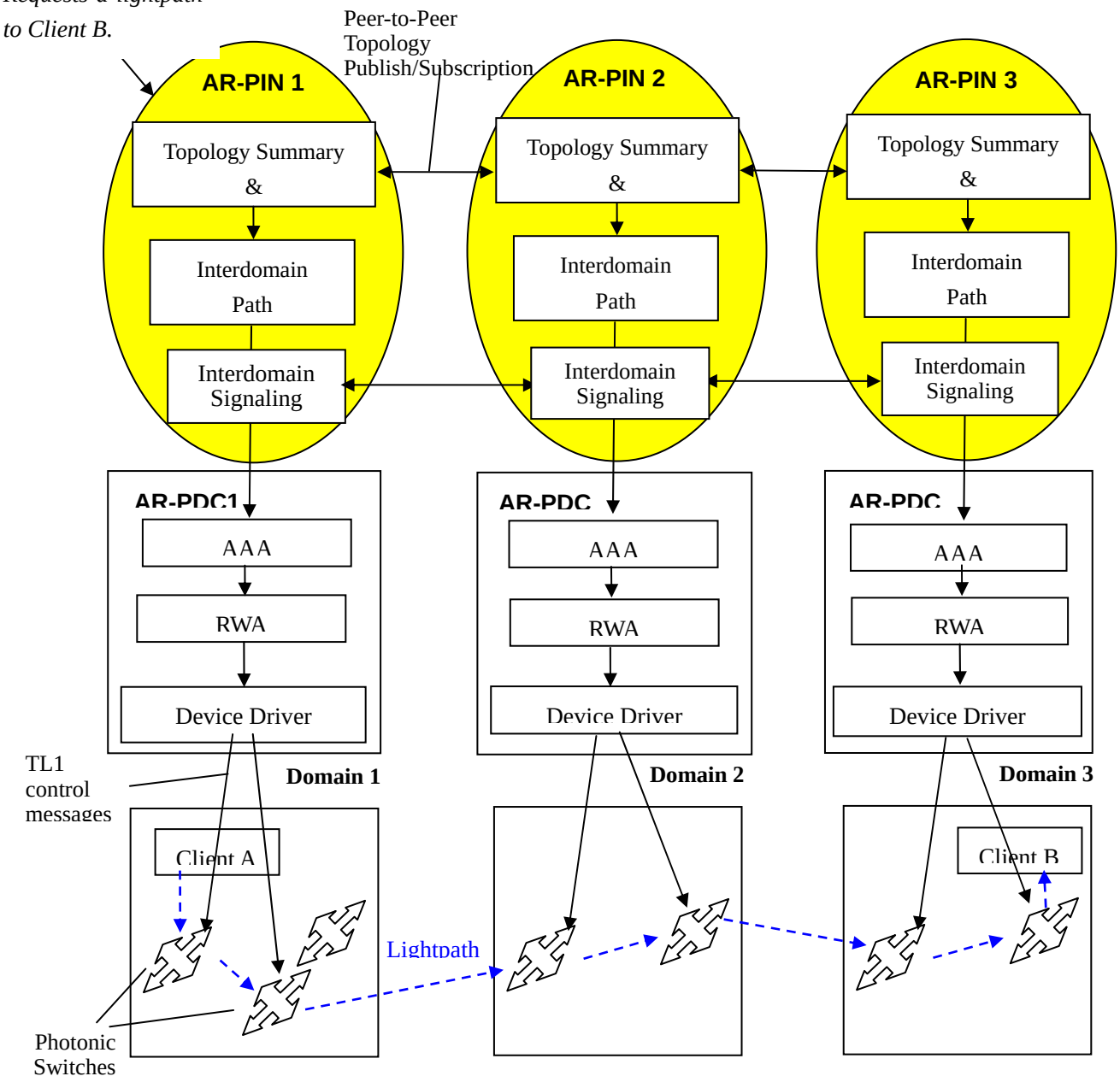

Figure 4-1. AR-PIN/AR-PDC System Architecture

This Figure shows that the major components of AR-PIN and AR-PDC and the interactions among them and photonic switches. Three domains are shown in the Figure.

4.1 AR-PIN: Interdomain Control Plane

A domain is an independently managed network cloud exposing a set of ingress and egress points and links with service specifications. Each link is controlled and managed by a single domain. The separation points between neighboring domains are switches. These switches are called as **border switches**. Ports on border switches can terminate links of multiple different domains. Every border switch needs a globally unique address or name for addressing purposes.

When a domain advertises its topology information to other collaborating domains, it is not necessary to include the details such as internal switches and internal links. Instead, it will just send out a topology summary of its own domain consisting of only border switches and abstracted links. For example, the advertisement from domain A will be:

Switch 1-2: wavelength w1, w2, w3, w4.

Switch 1-3: wavelength w1, w2, w3, w4, w5, w6, w7, w8.

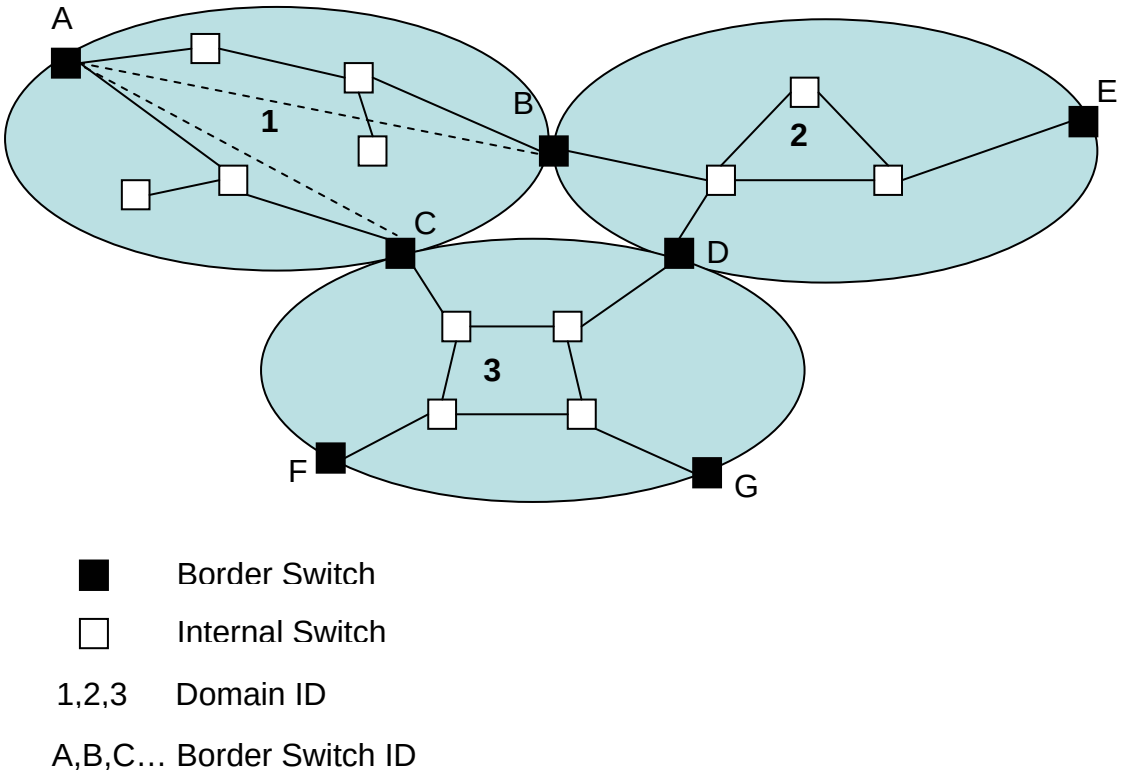

Figure 4-2. Structure of Multiple Photonic Domains

This Figure shows the switches and links of three routing domains. There are two catagaries of switches: border switches and internal switches. Topology summary of a domain consists of only border switches and abstract links. Only topology summaries are exchanged.

These two abstracted links are shown as dotted lines in Figure 4-2. The abstract link is actually an abstraction of a bunch of consecutive physical links in the same domain. The topology summary can be generated manually or automatically from the intradomain topology database. The topology summary generation is a maximum-flow problem and it can be solved by the Ford-Fulkerson method [Cormen01].

AR-PIN runs a peer-to-peer publish/subscribe based routing protocol to exchange topology summaries among different domains. The peer-to-peer exchange mode is more suitable than

blind flooding because it is possible that a domain may want to selectively advertise different sets of resources to different domains. The information exchange is based on the nature of the subscription. Every domain that wants to share its own wavelength resources maintains a list of collaborating domains (subscriber). The information exchange is triggered by any change of the interdomain topology of the domain. In other words, whenever the topology changes in a domain, the topology summary will be regenerated, the AR-PIN in this domain will update the new topology summary to all subscribed domains (push model) or just send a change notification and let the domains to request the update by themselves (pull model). The pull mode should always be supported to boot-strap newly started domains or out-of-sync domains. After receiving the topology summaries from all collaborating domains, each domain can compose its own global topology. Because each domain gets different topology summaries from different collaborating domains, every domain has its own unique global topology database. In this global view, each node is a border switch, and each link is an abstract link managed by a domain.

When a lightpath reservation request arrives, the local domain will compute a domain-level path based on its own view of global topology. This path includes only border switches. Source routing will be used to compute the path. There are several possible path computation algorithms such as Shortest Path First, Fixed Alternate, Least-Load-Path, etc. The detailed discussion of these algorithms is beyond the scope of this paper.

4.2 AR-PDC: Intradomain Control Plane

AR-PDC provisions intradomain lightpaths. Reservation requests may come from local domain users or its peering interdomain control plane AR-PIN. During the AR-PIN resource probing process, it (what is it) relies on AR-PDC to extend the domain-level path into a switch-level path and check the wavelength availability status.

Authorization, Authentication and Accounting (AAA)

When a reservation request comes from foreign domains, they need to go through the AAA mechanism to ensure the foreign user is authenticated. Then according to the identity of the user and the local access policy, the network resources will be filtered, and a virtual topology will be generated, which will be used in the following Routing and Wavelength Assignment (RWA) operation.

Intradomain Routing and Wavelength Assignment (IRWA)

AR-PDC does the RWA job at the switch level. I also divide the RWA problem into two sub problems: routing and wavelength assignment. For the routing problem, AR-PDC can use Fixed, Fixed Alternate or Adaptive algorithms – same as interdomain path computation. For the wavelength assignment problem, I execute a join operation on all hops from the ingress switch to the egress switch and return the resulting timeslot-wavelength availability matrix to PIN. When I use the Fixed Alternate algorithm, I can return the matrices of all paths to PIN

and let PIN choose the best one according to the intersection result with the matrix of the explored part of the path.

Device Driver

If a request gets reserved successfully, the user needs to claim the request when he/she wants to activate the reservation. Then each domain along the path will send TL1 commands to management ports of MEMS switches to set up cross connects. At the present time I have built device drivers for Calient DiamondWave PXC [Calient] and Glimmerglass Reflexion 3D [Glimmerglass] MEMS switches. Also a dummy switch is implemented for the purpose of emulation and debugging of high layer software. PDC software has unified interface to different types of MEM switches as shown in the class diagram Figure 4-3.

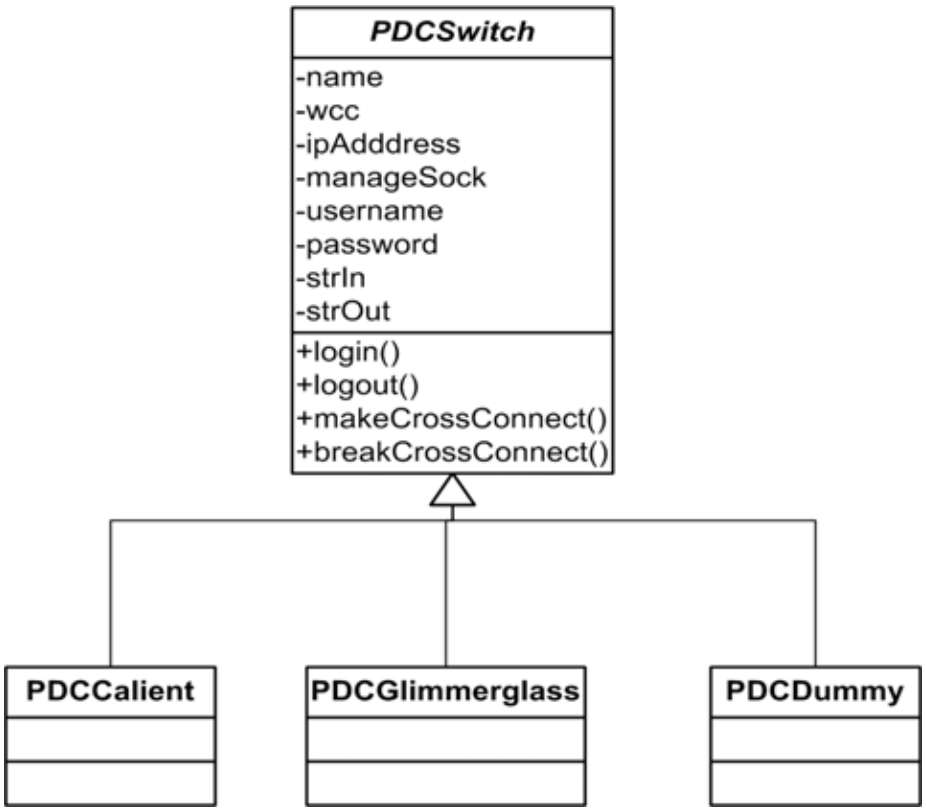

Figure 4-3. Class Diagram of Photonic Switches

This Figure shows that class diagram of photonic switches. PDCSwitch is the abstract parent class, it specifies the unified interface to different types of MEM switches. Currently three sub-classes are implemented for Calient switches, Glimmerglass switches and dummy switches respectively.

4.3 Apply FARM to AR-PIN/PDC

I described how to implement flexible advance reservation in meta-scheduling in chapter 3. The same principle can be applied to AR-PIN/PDC. In the context of cross-domain lightpath reservations, the meta-scheduler is implemented in AR-PIN, the local scheduler and the resource manager reside in AR-PDC.

I apply the FARM model to AR-PIN/PDC and reiterate the four steps shown in Figure 3-2. I call this algorithm **On-Demand Parallel Probe (ODPP)** because AR-PIN probes wavelength resources in each domain in parallel.

1). The user submits a flexible advance reservation request to AR-PIN, then AR-PIN computes domain level path based on its own global topology view, which has been described in section 4.1.

2.1). AR-PIN decomposes the original lightpath request into sub-requests and sends them to their local AR-PDCs. Each AR-PDC then computes its own local switch level path. Next the AR-PDC resource manager will operate a two-dimensional JOIN operation over the computed path, which is shown as Figure 4-4. The parameter t_{start} and t_{stop} from the original request specification (see equation 3-1) specifies the time range. The purpose of the JOIN is to remove unusable resources which cannot satisfy the wavelength and time continuities. Therefore the two dimensions are time slot and wavelength.

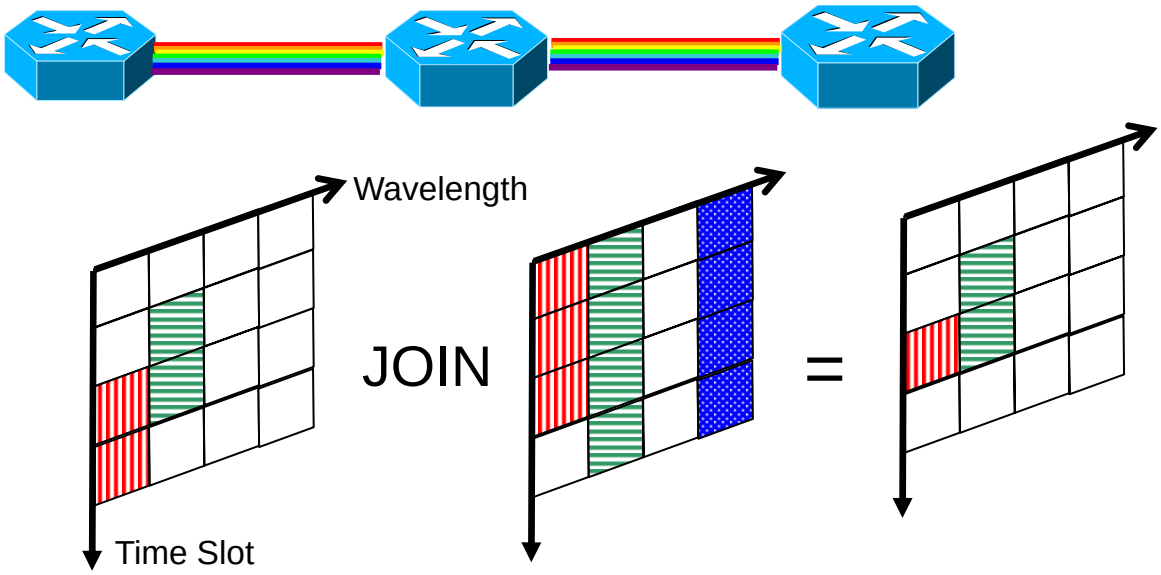

Figure 4-4. JOIN Operation in AR-PDC Resource Manager

An important operation during intra-domain and inter-domain resource probing is two-dimensional matrix join. Wavelength and time slot are the two dimensions. The purpose of the operation is to find slots maintaining the wavelength continuity and time continuity. It can be implemented by two dimensional bit-wise AND.

2.2). After the JOIN operation, the available time slots and wavelengths are found. The next operation is FILTER. The t_{md} in equation (3-1), minimum duration, is used to filter out those small time fragments whose duration is smaller than t_{md}. For example, if the t_{md} in Figure 4-4 is two time slots, then the red block in the right 2D plane will be filtered because its duration is only one slot. The final resulting 2D matrix is then sent back to the source domain's AR-PIN.

3.1). The AR-PIN collects timeslot-wavelength 2D matrices from all involved domains. It will operate JOIN and FILTER one more time in order to maintain the wavelength and time continuities of the end-to-end cross-domain lightpath. If there exists more than one candidate,

the best one will be selected based on user specified criteria, parameter c in the equation (3-1).

3.2). The AR-PIN needs to populate the selected timeslot-wavelength combination back to all involved domains. The resources may become unavailable due to other reservations. Two phase commit is adopted to make sure the reservations of all domains are all successful or all properly rolled back.

4). The AR-PIN will send back the reservation handle to the user if the two phase commit succeeds.

4.4 Algorithms

pdc-probe

During intra-domain probing process, the AR-PDC will be asked to find out the wavelength availability in the local domain. The ingress and egress switch are specified, a time range is given as well. The probe function initializes the wavelength-timeslot 2D availability matrix first. Then it uses some path computation algorithm such as Dijkstra's algorithm to compute the switch-level path at line 2. From line 4-7, for each hop on the switch-level path, all marked slots within the specified time range are found by querying database, and a wavelength-timeslot matrix is formed according to those marked slots. After that the matrix is joined together at line 6. Finally the result matrix is returned at line 8.

Let me analyze the complexity of the probe algorithm. The major time consumed by the

algorithm is database access. Therefore, I only consider the complexity of database access. The database access are at line 3 and 5. The store-db just perform once, and the find-mark-db just need one database operation to retrieve all marked slot on each link in the time range. Therefore, the complexity is O(h), h is the length of the switch-level path.

pdc-probe(resID, ingress, egress, range)

1. *init(matrix);*
2. *compute-switch-path(ingress, egress);*
3. *store-db(resID, path);*
4. ***for*** *each hop h on the path*
5. *marks = find-mark-db(h,range);*
6. *matrix$_l$ = compute-matrix(marks);*
7. *matrix = join(matrix, matrix$_l$);*
8. *return matrix;*

pdc-reserve

After the source PIN server selected the wavelength and fixed the time window, it starts the forward reservation process domain by domain. In each domain, pdc-reserve finds the correspond lightpath based on the reservation ID first. Then from line 2-4, the mark operation

needs to be performed for each time slot and each switch-level hop. In line 5-8, if the local domain is the source or destination domain, the switch port connecting to the client computer also need to be marked because I want to make sure the switch port is exclusively used by this reservation and this lightpath. The database access complexity is decided by line2-4. The mark operation will run h*t times, therefore the complexity is O(h*t);

pdc-reserve(resID, window, wavelength)

1. *lightpath = getLightpath(resID);*
2. ***for*** *each hop h in the lightpath*
3. ***for*** *each time slot t in window*
4. *mark-db(h, wavelength, t, resID);*
5. ***if*** *the local domain is the source*
6. *mark-db(srcClientPort, t, resID);*
7. ***if*** *the local domain is the destination*
8. *mark-db(destClientPort, t, resID);*

pin-odpp (On-Demand Parallel Probe)

The input to the interdomain lightpath reservation includes the source and destination

end-points, the time range in which the reservation is allowed, the minimal duration and the criterion for choosing the best solution. First the wavelength-timeslot matrix is initialized, the unique reservation ID is generated. Then in line 3 the domain-level path is computed based on the global topology view that the source domain has. From line 4-7, probing is performed in each domain on the selected domain-level path in parallel. All the wavelength-timeslot matrices are joined together to find the common wavelength and timeslot. Then from line 8-9, in the result matrix, the wavelength is selected based on the criterion, be the longest duration or the earliest duration, at the same time, the time window is fixed. Line 10-11 is real reservation process, domain by domain, the selected wavelength resource is marked in the slot database.

Because pin-reserve is a distributed process, I need to analyze both computation complexity and communication complexity. The computation complexity for the parallel probing is decided by the maximum of all domains, which is O(max(h)). The reservation process is domain by domain in sequential. So the complexity is O(d*h*t) considering the reservation in each domain is O(h*t), in which d is the number of domains on the domain-level path. For parallel probing, the total number of messages is 2(d-1) because the source domain will send a probeRequest message to each domain and receive a probeResponse message from each domain. For sequential reserving, the total number of messages is also 2(d-1). Therefore, the global communication complexity is O(h). However, the local communication complexity is

different for probing and reserving process. For probing, the local communication complexity of the source domain is O(h) because all messages are sent or received by the source domain. For reserving, all involved domain has communication complexity of O(1).

The parallelism of probing process effectively reduces the end-to-end interdomain signaling time. I will prove this by experiments in chapter 7.

pin-odpp(s,d,range,md,c)

1. *init(matrix);*
2. *generate(resID);*
3. *compute-domain-path(s,d);*
4. **for** *each domain d on the path p in parallel*
5. *find ingress and egress for* $domain_d$*;*
6. $matrix_d$*=probe(resID,ingress,egress,range);*
7. *matrix = join(matrix,* $matrix_d$*);*
8. *wavelength = select(matrix, c);*
9. *window = fix-window(matrix, c);*
10. **for** *(d=srcDomain; d<=destDomain;d->nextDomain)*
11. *d.pdc-reserve(resId, window, wavelength);*
12. *return result;*

pin-parallelClaim

Once the reservation instances and lightpath instances have been fixed and written into database during the reservation process, it is pretty straightforward to claim the reservation when the client application wants to use the lightpath resource. The parallelism of probing process effectively reduces the end-to-end interdomain signaling time. I will prove this by experiments in chapter 7.

pin-parallelClaim(resID)

13. ***for*** *each domain d on the path p in parallel*

14. *find all lightpath lList having resID;*

15. *for each lightpath l in lList*

16. *makeCrossConnects(l);*

17. *return result;*

4.5 AR-PIN/PDC Services

AR-PIN/PDC provides advance and immediate reservation service for applications or higher layer resource management systems. Their service interfaces should be defined clearly. As

described earlier AR-PIN accepts interdomain lightpath reservations while AR-PDC accepts intradomain lightpath reservations. Both in fact have similar interfaces. The interfaces are described as follows:

- **Reserve**: This function allows the application to submit a reservation with a specification of endpoints and time constraints to the reservation system. If the reservation succeeds, the system will reply with a unique reservation handle. This handle will be used for other operations such as modification and cancellation. The endpoints can be computing cluster nodes or photonic switch ports.

- **Cancel**: Before the reservation is bound, it can be cancelled.

- **Modify**: Before the reservation is bound, it can also be modified. For example, one can extend or shorten the reservation duration. If the modification request failed because part of the resources cannot be reserved, the original reservation should keep intact.

- **Bind**: When the application is ready to use a reservation, the resource manager may need to do some special processing for the application, or provide some run-time information to the application. For instance, in lightpath reservation systems, the control plane needs to set up the end-to-end lightpath for the application by make proper cross-connects in photonic switches. Also, the control plane may need to provide the IP addresses of end-points to application. This process is known as binding a reservation.

- **Unbind**: When a session of resource usage ends, the reservation should be unbound. After unbinding, the resource is still in reserved status and cannot be used by others. Therefore, if the application will not use the resource any more and the original reservation end time is in the future, it should cancel the reservation so that the resource can be returned to the pool of available resources.

- **Terminate**: This operation should be used when the reservation is in bound status. In fact Terminate is implemented as a combination of executing unbind and followed by cancel.

- **Query Reservation Status**: The client can discover the status of a reservation by polling it. The status includes whether the start of the reservation has begun and whether the reservation has been committed.

- **Query Reservation Attributes**: The client can discover the attributes associated with an existing reservation. These attributes include time-related or resource-related.

- **Subscribe Notification**: The client can subscribe to certain topics so that the resource manager can send messages when the status of the reservation changes or the reservation manager wishes to provide extra information to the application.

CHAPTER 5

SIMULATION

The simulation work I conducted in the thesis mainly consists of two parts. In the first part I validate that how the flexibility in advance reservations can improve acceptance rate and resource utilization. In the real world, immediate reservations co-exist with advance reservations. Therefore, in the second part the impact of advance reservations on immediate reservations is analyzed and it is concluded that both AR and IR requests need admission control algorithms in order to let both types of reservations live together and use the resources properly.

I ran simulations on the NSFNET topology with 14 nodes as shown in Figure 5-1. I assumed that each link is a single bi-directional fiber with 8 wavelengths. The entire topology was fully-optical without any wavelength converter. In the workload, the starting time of both advance and immediate reservations is a Poisson distribution and the reservation duration has a negative exponential distribution with mean duration of 30 minutes. All these distributions are mutually independent. For advance reservations, the book ahead time, $t_{start} - t_{reserve}$, is randomly selected between zero and the maximum allowed value. All requests try to reserve a lightpath with exactly one wavelength. I ran the simulations on five different randomly

generated workloads and took the average of the results. In all simulations, the path computation method employed is an adaptive routing algorithm using FIXED wavelength search since it can reach a reasonable balance of performance and complexity [Mokhtar98]. It searches through wavelengths in a fixed order until the available path is found. A standard shortest path algorithm is used to find a path on the effective topology.

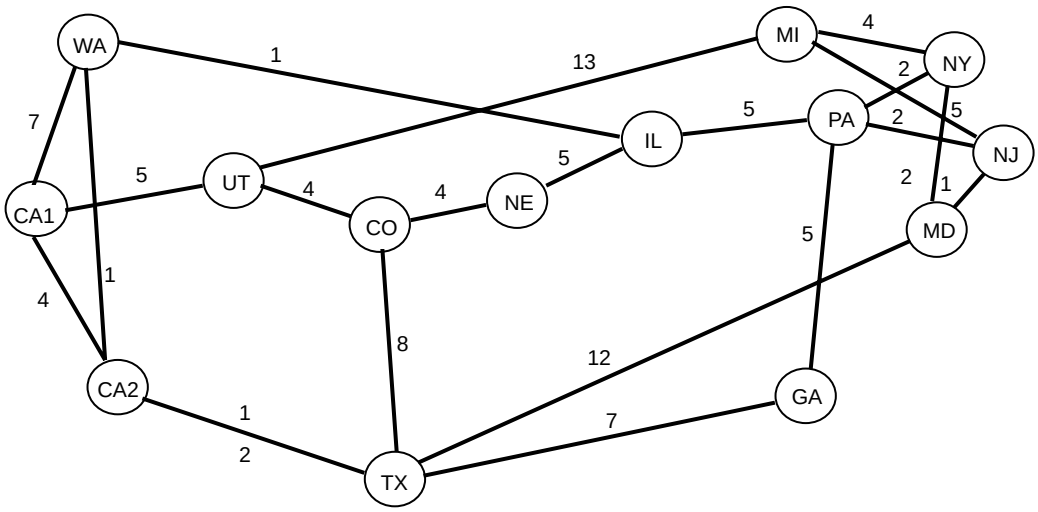

Figure 5-1. 14 Node NSFNET Topology.

This Figure shows the NSFNET 14 node model. The number on the links represent the relative distance.

For flexible advance reservations, the degree of flexibility is defined as:

$$flex = (t_{stop} - t_{start}) / t_{md} \qquad (5\text{-}1)$$

5.1 Flexibility Improves Both Acceptance Rate and Resource Utilization

The goal of the first set of experiments is to evaluate how flexibility affects the blocking rate and resource utilization of advanced reservations. I changed the degree of flexibility of the starting time of reservations from 0, 1, …, to 10. Figure 5-2 shows how the blocking rate varies with network load. The network load is denoted by the number of requests. The different curves represent the different degrees of flexibility. I can see that simply introducing 1 or 2 units of flexibility improves the performance considerably, but more flexibility does not help as much. For example, when the blocking rate is 5%, the system load is improved from 1100 requests to 1450 by introducing 1 unit of flexibility, and to 1720 by introducing 2 units of flexibility. Figure 5-3 shows how the relation of resource utilization vs. network load is affected by different flexibility. The maximum resource utilization can be improved from 47% to 57% by just introducing 1 unit of flexibility.

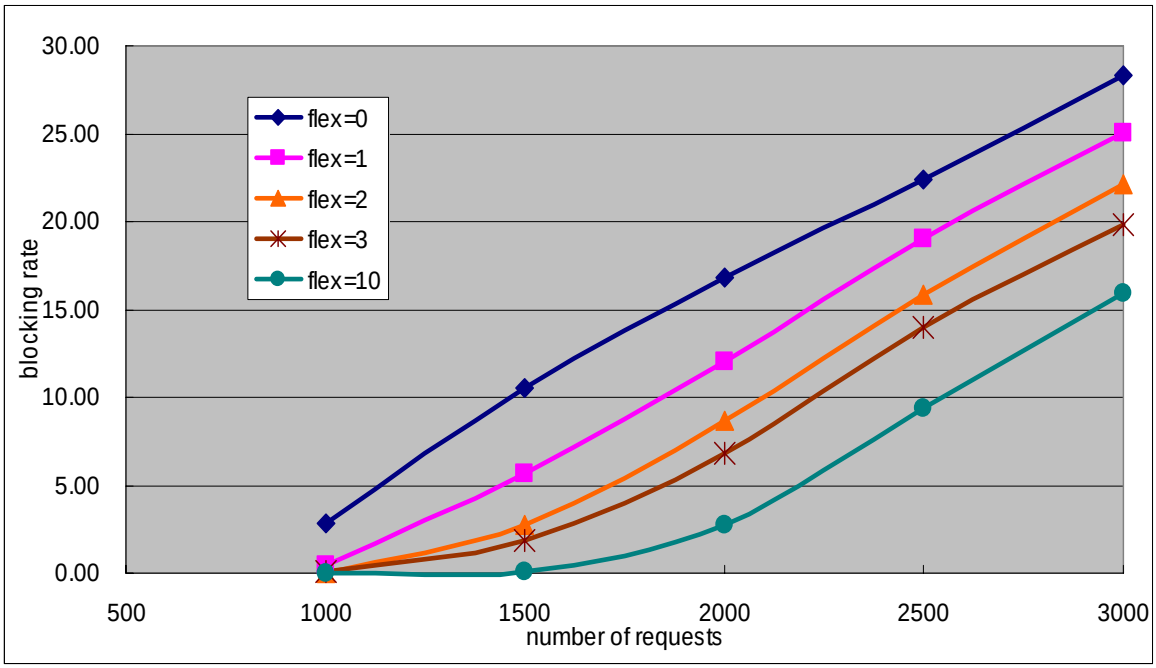

Figure 5-2. Blocking Rate under Different Flexibilities

This Figure shows how the blocking rate varies with network load. The network load is denoted by the number of requests. The different curves represent the different degrees of flexibility. We can see that simply introducing 1 or 2 units of flexibility improves the performance considerably, but more flexibility does not help as much.

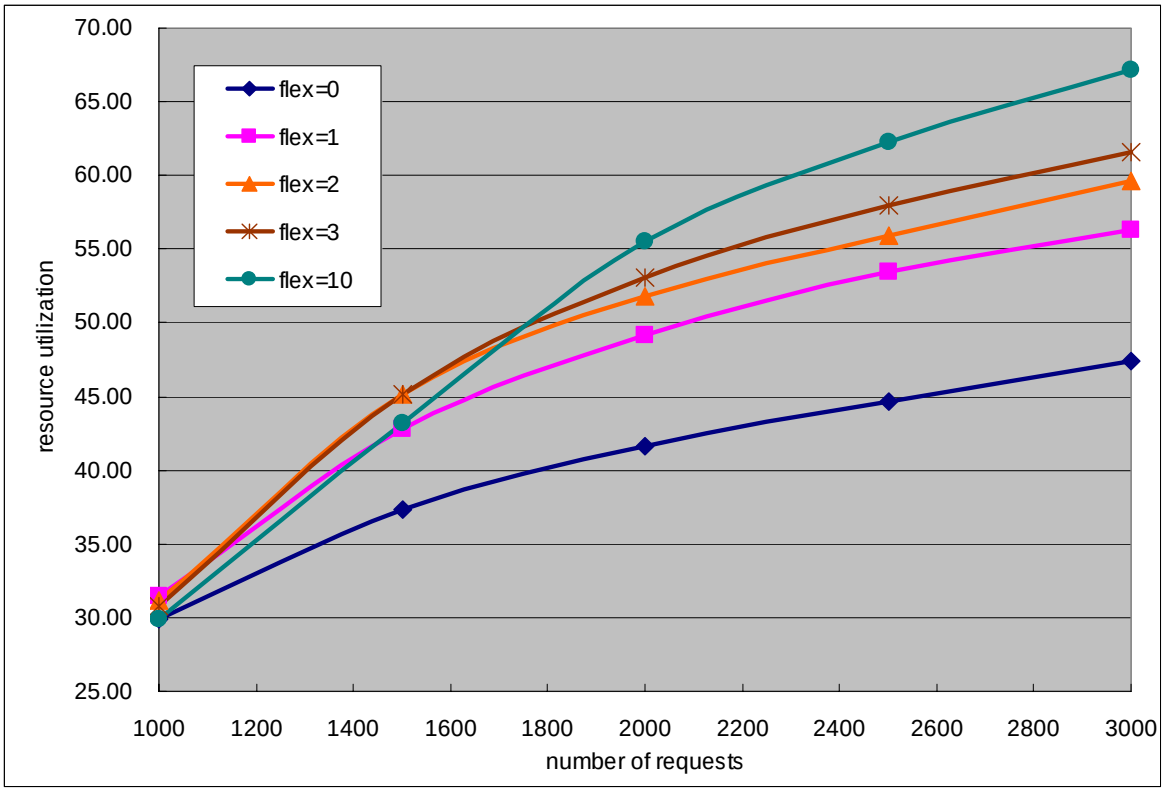

Figure 5-3. Resource Utilization under Different Flexibilities

This Figure shows how the relation of resource utilization vs. network load is affected by different flexibility. The maximum resource utilization can be improved from 47% to 57% by just introducing 1 unit of flexibility.

5.2 Comparison of Different Routing Algorithms

In another set of simulations, I wanted to evaluate how different routing algorithms (heuristics) perform under flexible advance reservations. The flexibility of starting time of reservations is 1 unit in simulations. I compared the three routing algorithms: Fixed Routing (FR), Alternate Fixed Routing (AFR), and Least Load Path Routing (LLP). LLP is one form of adaptive routing. In most prior RWA research in which flexibility is not considered, adaptive

routing showed superior performance to FR and AFR routing. However, from the simulation results in Figure 5-4, we can see the Alternate Fixed Routing (AFR) algorithm has the best performance and the Fixed Routing (FR) has the worst. The flexible advance reservation introduces a new temporal dimension into the resource allocation. The resource availability information is a three dimensional matrix of hop, wavelength and time slot. The reserved units scatter within this three dimensional matrix. The uncertainty of time parameters makes it difficult to filter out useful resource status and feed it back to the routing algorithm. This is why the LLP routing algorithm cannot perform well by just calculating average utilization of each wavelength within the reservation time window.

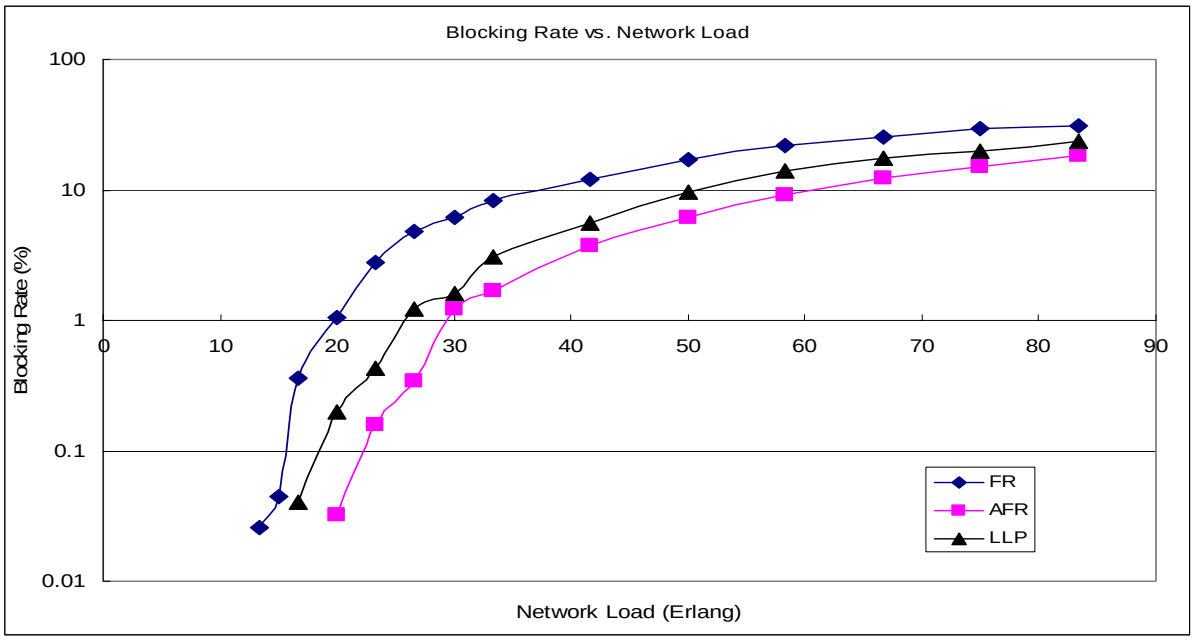

Figure 5-4. Blocking Rate of Different Routing Algorithms.

When Flexibility of Advance Reservations is 1T, the blocking rate of three routing algorithms: Fixed Routing (FR), Alternate Fixed Routing (AFR), and Least Load Path Routing (LLP) are compared. We can see the AFR algorithm has the best performance and FR has the worst.

5.3 Impact of Advance Reservations on Immediate Reservations

Certain extemporaneous activities can not or need not be planned ahead. As such our system should be able to take immediate reservations as well. There are three ways to share the wavelength resources between AR and IR requests: full sharing, partial sharing and strict partitioning.

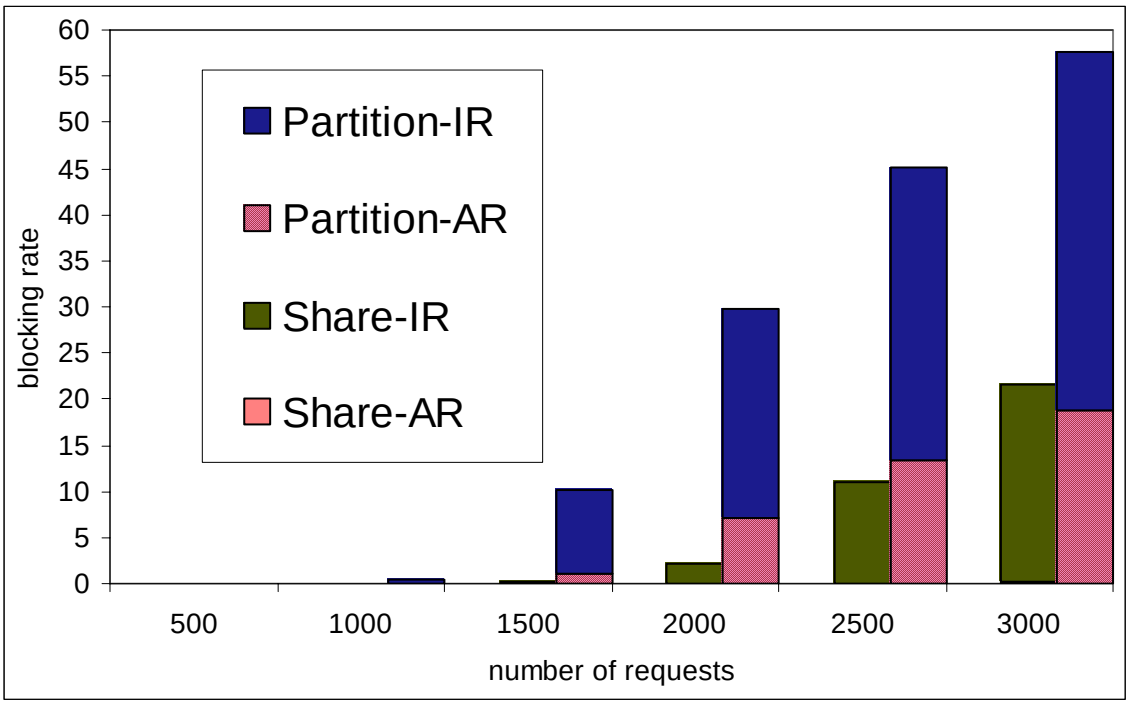

Figure 5-5. Comparison of Wavelength Sharing between ARs and IRs

I consider two situations: all AR and IR requests share all eight wavelengths (Share), or AR requests use four wavelengths and IR requests use the other four wavelengths (Partition). From this Figure, we can see that the Share case has much lower blocking rate than the Partition case. When the number of request is 3000, the blocking rate is 58% for strict-partitioning and only 21% for full-sharing case. The blocking rate of Share-AR is almost zero because it has time advantage over Share-IR.

In this simulation, there are eight wavelengths in the WDM network. I consider two situations: all AR and IR requests share all eight wavelengths (Share), or AR requests use four wavelengths and IR requests use the other four wavelengths (Partition). All AR and IR requests have independent identically-distributed Poisson distribution and occupy 50% of the entire load respectively. From Figure 5-5, we can see that the Share case has much lower blocking rate than the Partition case. When the number of request is 3000, the blocking rate is 58% for strict-partitioning and only 21% for full-sharing case. The blocking rate of Share-AR is almost zero because it has time advantage over Share-IR.

5.4 The Dropping Problem and IR Admission Control

Even though sharing brings about greater blocking rate performance, it also introduces a new problem: IR dropping. An admitted IR request may be dropped when the IR request conflicts with a reserved AR request. High and unpredictable dropping degrades the service satisfaction dramatically. I have two means to improve the user experience. Firstly, I could introduce one more parameter for IR requests: Minimum Duration (MD). The IR admission control algorithm will scan the future time slots to make sure the needed resources of this IR request are vacant within the Minimum Duration. Another measure is to notify the user when he/she is possible to be dropped in advance. When the IR request is admitted, I can continue search the future time slot table to find the next conflict point. After the conflicting point, when

the AR customer claims the AR request, the conflicting IR request will be dropped out after a short period. During the short period, the IR user can have time to gracefully stop his/her application. I can imagine that if I specifying a larger MD, the probability of blocking will be increased. This is confirmed in the simulation results shown in Figure 5-6.

In this simulation, the IR profile is fixed. With an increase in AR load, the blocking rate of IRs increases because more resources are pre-occupied by ARs. At the same time, the blocking probability is higher if the user specifies higher Minimum Duration for IRs. The average duration of IRs is 1800 seconds.

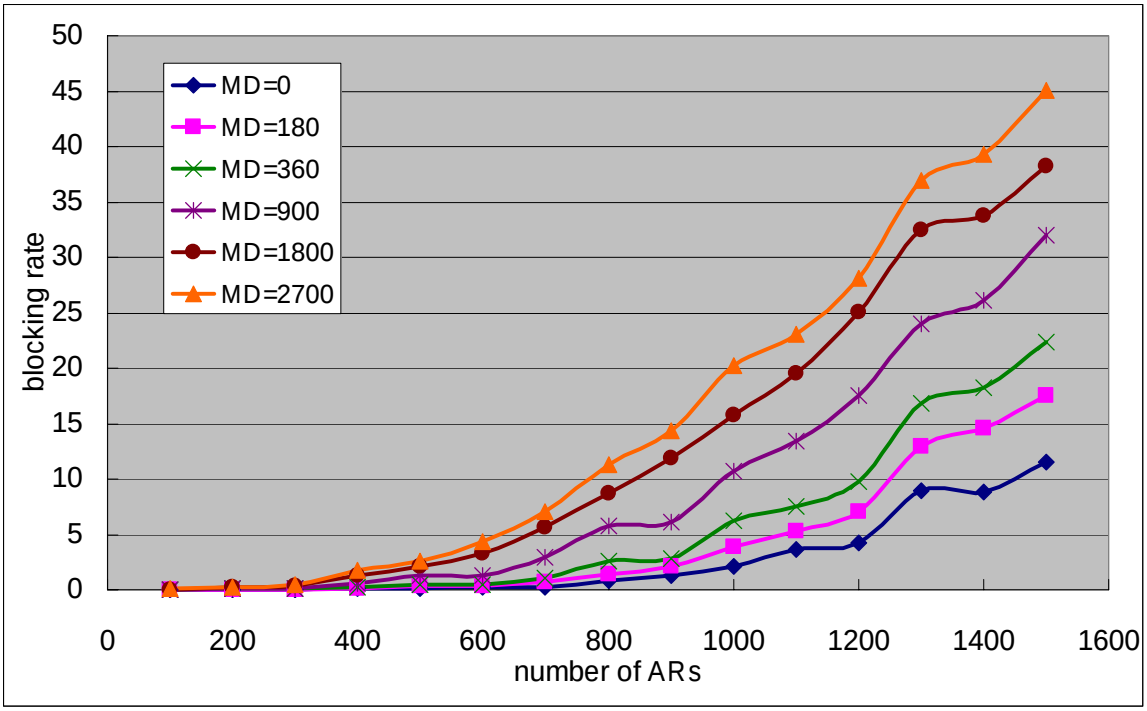

Figure 5-6. Blocking Rate of IRs for Different Minimum Durations

In this simulation, the IR profile is fixed. With an increase in AR load, the blocking rate of IRs increases because more resources are pre-occupied by ARs. At the same time, the blocking probability is higher if the user specifies higher Minimum Duration for IRs. The average duration of IRs is 1800 seconds.

5.5 AR Admission Control

Since AR requests book reservations relatively far ahead, this gives AR requests priority over IR requests. If there is no admission control for AR requests, it is possible that AR requests occupy majority of the resources, which causes a high blocking rate of IR requests or even starvation. In order to provide a certain level of service guarantees to IR requests, it is necessary to put an upper limit on admitted AR requests. From an economic perspective, the

charge of IR requests is usually more than AR requests. For example, the ticket fare is usually less expensive if you book earlier in airline reservation systems. At the same time, there are always some impromptu circumstances which cannot be anticipated. We need to keep this type of resource requests from starving.

The method I employed is to reserve partial wavelengths for IR requests only. For example, AR requests can only use the first five of the total eight wavelengths. From Figure 5-7, we can see that the AR admission control brings down the blocking rate of IR requests from 82% to 34% when the number of requests is 3000, while it increases the blocking rate of AR requests at the same time because of reduced available wavelength resources. Therefore I achieved a much better balance between AR and IR requests. The percentage of resources specially left to IR requests can be adjusted in run-time by the network administrator.

5.6 Summary

Through the simulation work, I found that the system performance can be improved dramatically by introducing some flexibility on the time parameters of advance reservations. IR minimum duration is necessary to have good Quality of Service and user experience. AR admission control is necessary in order to maintain a well-balanced AR/IR mixed environment.

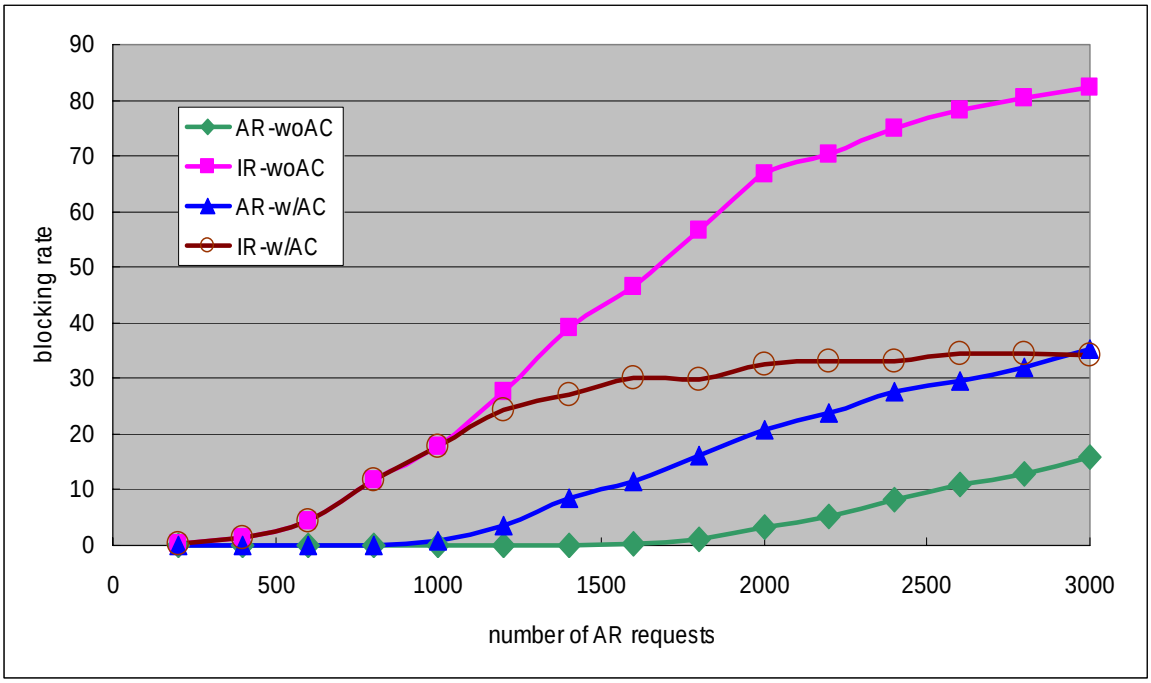

Figure 5-7. Effect of AR Admission Control

AR requests can only use the first five of the total eight wavelengths when admission control is employed. From this Figure, we can see that the AR admission control brings down the blocking rate of IR requests from 82% to 34% when the number of requests is 3000, while it increases the blocking rate of AR requests at the same time because of reduced available wavelength resources.

CHAPTER 6

IMPLEMENTATION OF AR/PIN-PDC

The AR-PIN/PDC (Advance Reservation enabled Photonic Inter-domain Negotiator and Photonic Domain Controller) software provides the lightpath reservation and provision services through a series of remote procedure calls. In order to accept diverse client types, I implement these remote procedure calls as web services. As long as the client can generate proper XML-based messages and send to the AR-PIN/PDC web services, the client code can be written by Java, C/C++, Python, etc. I chose JBoss as the web service hosting software because of its easy-to-use and stability. In this chapter, I will list all the interfaces that AR-PIN/PDC provides. Then I will expose the internal of AR-PIN/PDC implementation by describing the important data structures and how the algorithms described in Chapter ? are implemented in the distributed environment. After that I will describe the database tables and implementation. Finally I will briefly introduce JBoss software and the deployment of AR-PIN/PDC.

6.1 Service Description of AR-PIN/PDC

AR-PIN/PDC software exposes the following web services:

1. advanceReserveHH

Parameters:

String srcCluster, String srcNode, String srcNic,

String destCluster, String destNode, String destNic,

long start, long period, long minDuration, int criterion.

Return :

PDCReserveReturn

This function is used to reserve a lightpath between two end-points. The client needs to specify two types of information. The first type is end-point information. These parameters include the cluster name, node name and the name of the network interface card (NIC) of both source and destination. Because I assume that all lightpaths are bidirectional in this thesis, the source and destination are just names of two end-points. There is no difference to specify an end as the source or the destination. The other type of parameters is time-related. They include the time range defined by the start time and the period, the minimum duration and the selection criterion. It is possible that the RWA algorithm finds that more than one solution satisfy the user-specified request, the selection criterion is used to make the decision which

solution should be chosen. For now the criterion includes only two options: the earliest or the longest duration. This can be extended in the future. For example, some other options can be the lightest-loaded path, the shortest path or the lowest cost path. This function returns a *PDCReserveReturn* data structure. This data structure includes the following elements:

String **reservationId**: The reservation ID returned from the AR-PIN/PDC server. The client needs this ID to claim or cancel the reservation later on. If the reservation is failed, the returned ID will be null.

String **srcAddr**: The IP address of the source end-point. Applications will need IP addresses of end-points to communicate to each other.

String **destAddr**: The IP address of the destination end-point.

long **start**: The start time of the reservation.

long **end**: The finish time of the reservation.

String **message**: In case the reservation is failed, this message will show what problem it is.

2. advanceReserveSH

Parameters:

String srcSwitch, int srcPort,

String destCluster, String destNode, String destNic,

long start, long period, long minD, int criterion.

Return:

PDCReserveReturn

This function only works in local domain. It is used to reserve a lightpath between a photonic switch port and a computing end-point. The switch end is always defined as the source end because the lightpath is not directional. The optical switch parameters include the switch name and the port number. The time-related parameters and the returned data structure are same as the **advanceReserveHH** function.

3. advanceReserveSS

Parameters:

String srcSwitch, int srcPort,

String destSwtich, int destPort,

long start, long period, long minD, int criterion.

Return:

PDCReserveReturn

This function only works in local domain. It is used to reserve a lightpath between two photonic switch ports. There is no difference between source and destination because the lightpath is not directional. The optical switch parameters include the switch name and the port number. The time-related parameters and the returned data structure are same as the **advanceReserveHH** function.

4. claim

Parameter: String reservationId.

Return: int.

It is called to claim a reservation. The only parameter is the reservation ID. This function should be called within the valid reservation time window. When this function is called, the AR-PIN/PDC server tries to make proper cross connects of photonic switches to set up the reserved lightpath. It returns 1 if the claim is successful and 0 if it's failed.

5. unbind

Parameter: String reservationId.

Return: int.

It is called to unbind a reservation during a reservation session. The only parameter is the reservation ID. This function should be called within the valid reservation time window and the reservation is in bound status. The server will first tear down the lightpath by breaking the related cross connects, then change the reservation status to “reserved” in the database. It returns 1 if the termination is successful and 0 if it's failed

6. terminate

Parameter: String reservationId.

Return: int.

It is called to terminate a reservation. The only parameter is the reservation ID. If this function is called before the reservation is claimed, the AR-PIN/PDC server just nulls the reservation in the database without touching the photonic switches. If this function is called when the reservation is in active status, the server will first tear down the lightpath by breaking the related cross connects, then change the reservation status in the database. It

returns 1 if the termination is successful and 0 if it's failed

7. modify

Parameters: String reservationId, long start, long finish.

Return: PDCReserveReturn.

This function is used to modify existing reservations in the system. Claimed reservations can not be modified. Therefore, the reservations have to be in "reserved" status. The user needs to specify the original reservation ID, the new start time and finish time. The AR-PIN/PDC will return a *PDCReserveReturn* data structure.

8. renew

Parameters: String reservationId, long period.

Return: PDCReserveReturn.

This function tries to renew active reservations. The reservations have to be in "claimed" status. The user needs to specify the original reservation ID and the extra time he/she needs

after the original finish time. The AR-PIN/PDC will return a *PDCReserveReturn* data structure.

6.2 Data Structures

In this section, I will describe the important data classes defined in AR-PIN/PDC. All these data classes have a corresponding table in database. Persistence is very important for this kind of service providing software. When application is shut down for some unexpected reasons such as power off or hardware failure, all the data and status can be easily restore from database.

There is a paradigm mismatch between object-oriented classes and relational tables in databases. People have spent significant time of effort to bridge the object/relational paradigm mismatch. It is estimated that 30% of Java application code written is to handle this problem. [Bauer04] Therefore, automated object/relational mapping is being researched in the past years and Hibernate was emerged as the most promising solution. In AR-PIN/PDC, I used Hibernate to map the following classes to the corresponding tables in MySQL database:

Cluster

The client-side end points are abstracted as clusters because beowulf clusters consisting of commodity PCs are getting popular and becoming the mainstream platform of storage,

computing and visualization.

int id: Unique ID of the cluster.

String name: Name of the cluster.

int domain: Domain ID to which the cluster belongs.

ClusterNode

A computer cluster usually consists of multiple computer nodes. Each computer node can have one or more than one optical interface cards connecting to the photonic switches. This class abstracts a pair of a cluster node and network interface card (NIC).

int id: Unique numeric ID of the cluster node.

int cluster: Cluster ID to which the node belongs.

String node: Name of the cluster node.

String nic: Name of the network interface card (NIC).

int switch: Switch ID to which the cluster node/NIC connects to.

int port: Switch port number to which the cluster node/NIC connects to.

String ipaddr: IP address of the cluster node/NIC.

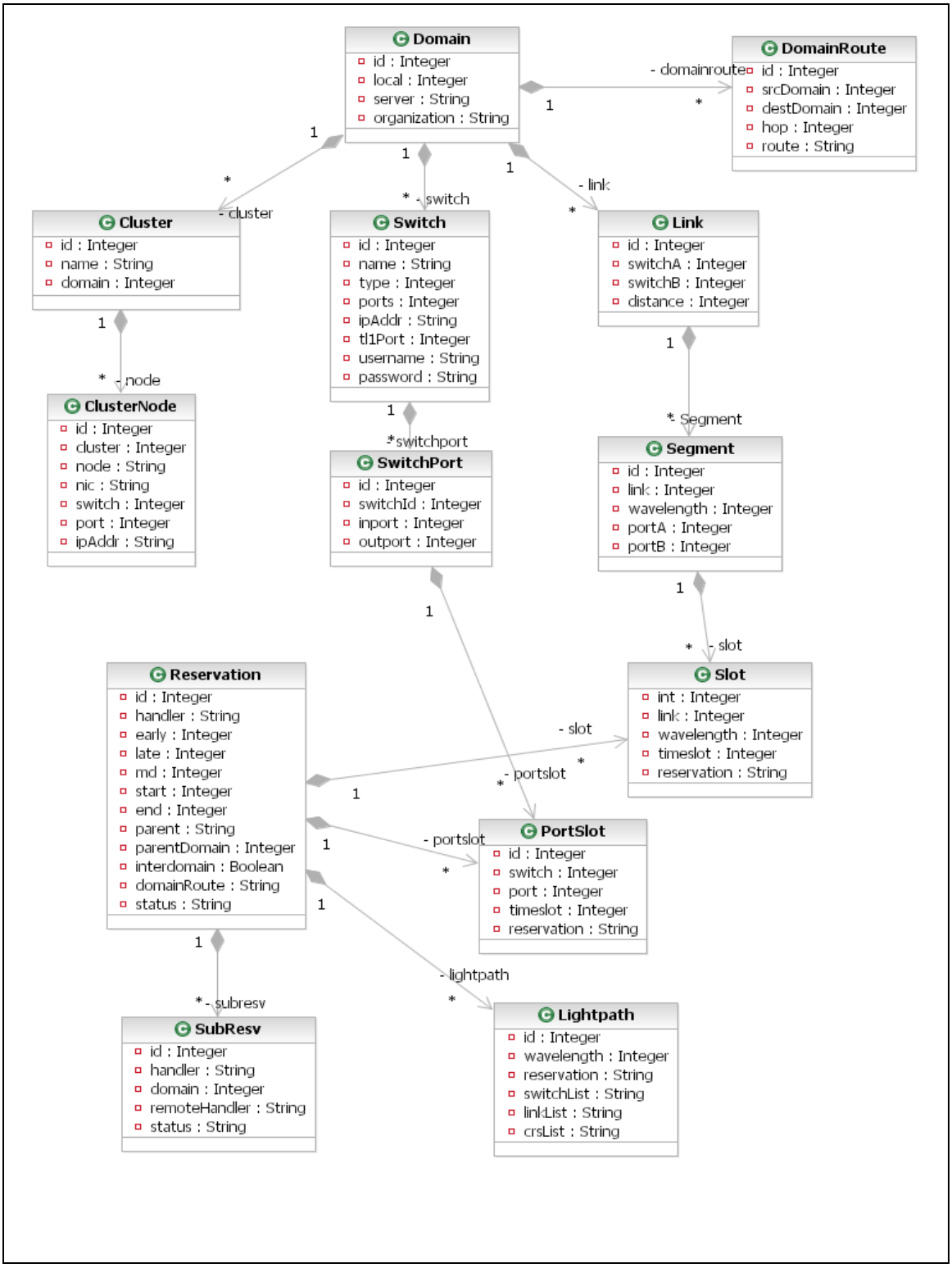

Figure 6-1. Class Diagram of Basic Data Structures in AR-PIN/PDC

Domain

A domain is an independently managed network cloud exposing a set of ingress and egress points and links with service specifications. In this thesis, each domain runs one instance of AR-PIN/PDC server.

int id: Numeric ID of the domain. This ID is unique and consistent in all domains globally.

int local: Specify if this domain is a local domain.

String server: Name of the local AR-PIN/PDC server.

String organization: Name of the organization that runs this domain.

DomainRoute

For interdomain lightpath requests, the first step is to find the domain-level route. DomainRoute class keeps this information. As far as how to generate this information, there are several ways. It can be specified statically by administrators, or it can be computed dynamically by some path computation algorithms such as Shortest-Path First, based on the topology information received from other collaborative domains.

int id: ID of the domain-level route.

int srcDomain: Source domain ID.

int destDomain: Destination domain ID.

int hop: Number of domain-level hops. It equals to the number of all involved domains minus

one.

String route: A string to express the domain-level route. The string consists of domain IDs and border switch IDs listed alternatively and separated with commas, starting from the source domain ID and ends with the destination domain ID. For example, in “1,A,2,B,3”. 1,2,3 are domain IDs and A,B are border switches although they are actually numbers.

Lightpath

This class keeps the detail information of lightpaths. For intradomain lightpaths, the end-to-end information of switches, ports and links is stored. For interdomain lightpaths, only local information is stored, i.e., from ingress switch to egress switch.

int id: ID of the lightpath.

int wavelength: Wavelength ID of the lightpath.

String reservation: The reservation ID to which the lightpath belongs. One reservation can have multiple lightpaths.

String switchList: A string to express the switch-level route. The string consists of switch IDs ordered from the source/ingress switch to the destination/egress switch and separated with commas.

String linkList: A string to express the switch-level route together with switchList. The string consists of link IDs with the same order as switchList, separated by commas. The number of

items in linkList should be that of switchList minus one.

String crsList: A string to express the ports on the switch-level route. The string consists of incoming port and outgoing port of each switch on the route, with the same order as switchList, separated by commas. The number of items in crsList should be as twice as that of the switchList.

Reservation

Every advance reservation will have an instance of this class. The handler, sometimes I call it reservation ID, is a unique String representing this reservation. For interdomain reservation, a sub-reservation will be created for each foreign domain. In this system, I specify absolute time using a milliseconds value represents the number of milliseconds that have passed since January 1, 1970 00:00:00.000 GMT.

int id: Unique internal numeric ID.

String handler: A string to represent this reservation uniquely.

long early: The earliest time of the allowed time window for flexible advance reservations.

long late: The latest time of the allowed time window for flexible advance reservtions.

long md: The milliseconds value of minimal duration that the user requested.

long start: After the AR-PIN/PDC reserves the resource successfully, the flexible time window will be fixed. This parameter is the start time of the fixed window.

long end: The end time of the fixed window after reservation.

String parent: If this reservation is a child reservation of another interdomain reservation, this parameter stores the handler of the parent reservation, otherwise it is null.

int parentDomain: If this reservation is a child reservation of another interdomain reservation, this parameter stores the ID of the domain where the parent reservation is located.

int interdomain: If this reservation is interdomain and it is the parent, this parameter is 1, otherwise it is 0. Please note, for child reservations of interdomain, this is also 0.

String domainRoute: The domain-level route retrieved from the database.

String status: The status of the reservation. It could be

"reserved": The reservation has been made successfully, but not claimed yet.

"claimed": The reservation has been claimed successfully and the lightpaths are set up.

"terminated": A "claimed" reservation is terminated before it expires.

"cancelled": A "reserved" reservation is cancelled before it is claimed.

"expired": A "claimed" reservation will be terminated by the system when the end time arrives, after that the status becomes "expired".

Subresv

For interdomain reservation, a sub-reservation will be created for each foreign domain.

Each sub-reservation corresponds to a reservation in a remote domain.

int id: Numeric ID.

String handler: The handler of the parent reservation.

int domain: The ID of the remote domain.

String remoteHandler: The hander of the corresponding reservation in the remote domain.

String status: The status of the remote reservation.

Switch

This class abstracts physical photonic switches.

int id: Numeric ID.

String name: Unique name in String format.

int type: Currently AR-PIN/PDC supports three types of switches: 1 – Calient, 2 – Glimmerglass, 3 – dummy.

int ports: Number of ports.

String address: IP address of the management interface.

int tl1port: The TL1 communication TCP port of the management interface.

String username: User name of TL1 management interface.

String password: Password of TL1 management interface.

int border: This parameter specifies if this switch is a border switch.

SwitchPort

This class abstracts a port on a photonic switch.

int id: Numeric ID.

int switchId: The ID of the photonic switch to which the port belongs.

int inport: Together with outport, they specify the cross connect status. The inport specifies which output port this input port connects to. 0 if not connected.

int outport: The outport specifies which inport port this outport port connects to. If the connection is bidirectional, inport and outport should be in pair.

Link

This class represents the DWDM optical link connecting two photonic switches. It has no direction.

int id: Numeric ID.

int switchA: ID of one of connecting photonic switches.

int switchB: ID of another of connecting photonic switches.

double distance: The physical distance between the two connected switches.

Segment

A DWDM optical link consists of multiple wavelengths and connects two photonic switches. The Segment class represents one wavelength within a link.

int id: Numeric ID.

int link: The ID of Link to which the segment belongs.

int wavelength: The wavelength ID.

int portA: The port number of the wavelength on the switcha in the corresponding Link.

int portB: The port number of the wavelength on the switchb in the corresponding Link.

Slot

This is the slot table of wavelengths. The slot table is three-dimensional, consisting of link, wavelength and timeslot. The database only keeps the reserved units in the three-dimensional space. Please note AR-PIN/PDC keeps two important constants. The first constant is the time slot granularity SLOT_GRANULARITY. AR-PIN/PDC divides the continuous time range into discrete time slots of fixed size. Thus, reservations can be made for a number of consecutive slots. The slot granularity is defined by the duration covered by a single slot. If the slot granularity is too small, the number of slots needed by a reservation will be very large, which results in too many database operations. The other constant is the largest number of time slots the system supports MAX_SLOT. If this number is too large, the

AR-PIN/PDC server has to consume excessive memory and the database may be very large. The product of SLOT_GRANULARITY and MAX_SLOT is the latest time when the finish time of reservations can be set. For example, if SLOT_GRANULARITY is one minute, and MAX_SLOT is 60*24*30 = 43200, it means that I can reserve wavelengths to as late as 30 days from now.

int id: Numeric ID.

int link: The ID of the link.

int wavelength: The ID of the wavelength.

long timeslot: The absolute slot number, i.e., the absolute time divided by SLOT_GRANULARITY.

String reservation: The handler of the reservation to which the time slot belongs.

PortSlot

The main function of AR-PIN/PDC is manage end-to-end lightpaths. When I say end-to-end lightpaths, that means from NIC to NIC. In other words, the lightpath not only includes the wavelengths on the way and the switch ports where these wavelengths connect, but also the switch ports where client NICs connect to. Therefore, these ports are also resources clients need to reserve. PortSlot class manages the time slot of these switch ports. This slot table is two-dimensional, consisting of switch port and time slot.

int id: Numeric ID.

int switch: The numeric ID of the photonic switch to which the port belongs.

int port: The photonic switch port number.

long timeslot: The absolute slot number, i.e., the absolute time divided by SLOT_GRANULARITY.

String reservation: The handler of the reservation to which the time slot belongs.

6.3 Two Web Service Modes: Synchronous and Asynchronous

6.3.1 Web Services

Why is the Internet so successful? An important reason is that it uses Internet Protocol as the only protocol in Layer 3. Under IP, different data link protocols such as ATM, Ethernet or PPP can be used to transport data; above IP, different applications such as web browsing, email, file transfer or Voice or IP can be built upon. All these protocols speak the same language – IP – so that they can understand each other and support each other and conglomerate into a huge Internet society. I can say it is the IP that makes all data within the Internet be able to talk to each other. The distributed applications have the same situation. The diverse distributed systems need a common “language” to communicate to each other. XML-based web services are believed to be a good candidate.

Service Oriented Architecture (SOA) is a component-based architecture, it divides a distributed application into a number of separate services that, individually, perform a specific function, but when put together make up the components of a larger application. The rational of SOA is not new. Component-based distributed systems have been around for years. Three famous architectures are DCOM (Distributed Component Object Model), CORBA (Common Object Request Broker Architecture) and Java/RMI (Remote Method Invocation). DCOM is Microsoft specific, Java/RMI is Java specific, CORBA is both platform and language independent, but its complexity and lack of security and versioning make it hard to be accepted by the industry [Henning06]. XML-based Web service is a promising technology to push SOA to a success.

The potential users of AR-PIN/PDC are the scientists from different fields such as astronomy, biology, geographer, physics etc. The application and technologies they used are very diverse. For example, their applications may use different operating systems, different programming languages. What if their applications all want to reserve lightpaths, including a wide range of legacy application? Obviously we should use a standardized way to provide the lightpath service – XML-based web services.

In web service world, XML is the universal data format. This saves us a lot time and effort to “marshalling” data for RPC calls among distributed systems because almost all modern systems provides decent XML processing engine. However, use of XML-RPC was

limited as SOAP (Simple Object Access Protocol) evolved rapidly and offered richer semantics than XML-RPC.

There are two paradigms in terms of the remote method invocations among distributed systems: RPC style and Message-Centric style. RPC is a good model for function-centric invocations. It works well when I have clearly defined functions and associated parameters. RPCs are typically associated with synchronous functional invocations and do not support messaging semantics such assured delivery. In Message-Centric style, data is exchanged in a prescribed message format that both the sender and the receiver can understand. This programming model is often used for loosely coupled integration with messages and events flowing back and forth. One of the strengths of the Message-Centric model is support for asynchrous invocations.

6.3.2 Synchronous vs. Asynchronous Web Services

There are two principle architectures for Web Service interfaces: synchronous Web Services and asynchronous Web Services. There two architectures are distinguished by their request-response handling. With synchronous services, clients invoke a request on a service and then suspend their processing while they wait for a response. With asynchronous services, clients initiate a request to a service and then resume their processing without waiting for a response. The service handles the client request and returns a response at some later point, at

which time the client retrieves the response and proceeds with its processing.

Because the client suspends its own processing after making its service request, synchronous services are best when the service can process the request in a small amount of time or when applications require a more immediate response to a request. Web services that reply on synchronous communication are usually RPC-oriented.

With asynchronous services, the client invokes the service but does not – or cannot – wait for the response. Often, with these services, the client does not want to wait for the response because it may take a significant amount of time for the service to process the request. The client can continue with some other processing rather than wait for the response. Later, when it does receive the response, it resumes whatever processing initiated the service request.

When I implemented AR-PIN/PDC software, I used both synchronous and asynchronous communications. For some requests such as inter-domain reservation request, it takes long time to run the probing process in parallel and then the reservation process domain by domain. Therefore, asynchronous mode is more proper for clients because they do not wait for the long time. For some requests such as reservation claim request, the client application may want to wait for the response from the server to make sure the lightpaths have been set up, then they can send data over the lightpaths. For some operations such as domain-by-domain reservation, the synchronous mode will be extremely inefficient because the source domain has to wait for the response from each domain before it can send reservation to the next

domain. In the next two sections, I will take the Inter-domain reservation as an example to compare different communication scenarios of synchronous and asynchronous mode.

6.3.3 Synchronous Mode Interdomain Reservation

In this scenario, all AR-PIN/PDC web services are provided in synchronous mode. In another word, all web service calls are RPC based. When the client calls the local AR-PIN/PDC server, it will be blocked to wait for the return from the local AR-PIN/PDC server, i.e., to wait for the entire probing and reservation process to finish. During probing process, the local AR-PIN/PDC server can not execute parallel probing in RPC mode unless it creates new threads for every remote PIN peers. As shown in Figure 6-2, the local AR-PIN server probes the remote AR-PIN1 server first, waits for the probing response, then probes the remote AR-PIN2 server, etc. This is quite inefficient. During the reservation process, it can not be implemented in domain-by-domain forward reservation style using RPC style web services. It is the same as the probing process, the local AR-PIN server has to reserve each remote domain in serial fashion.

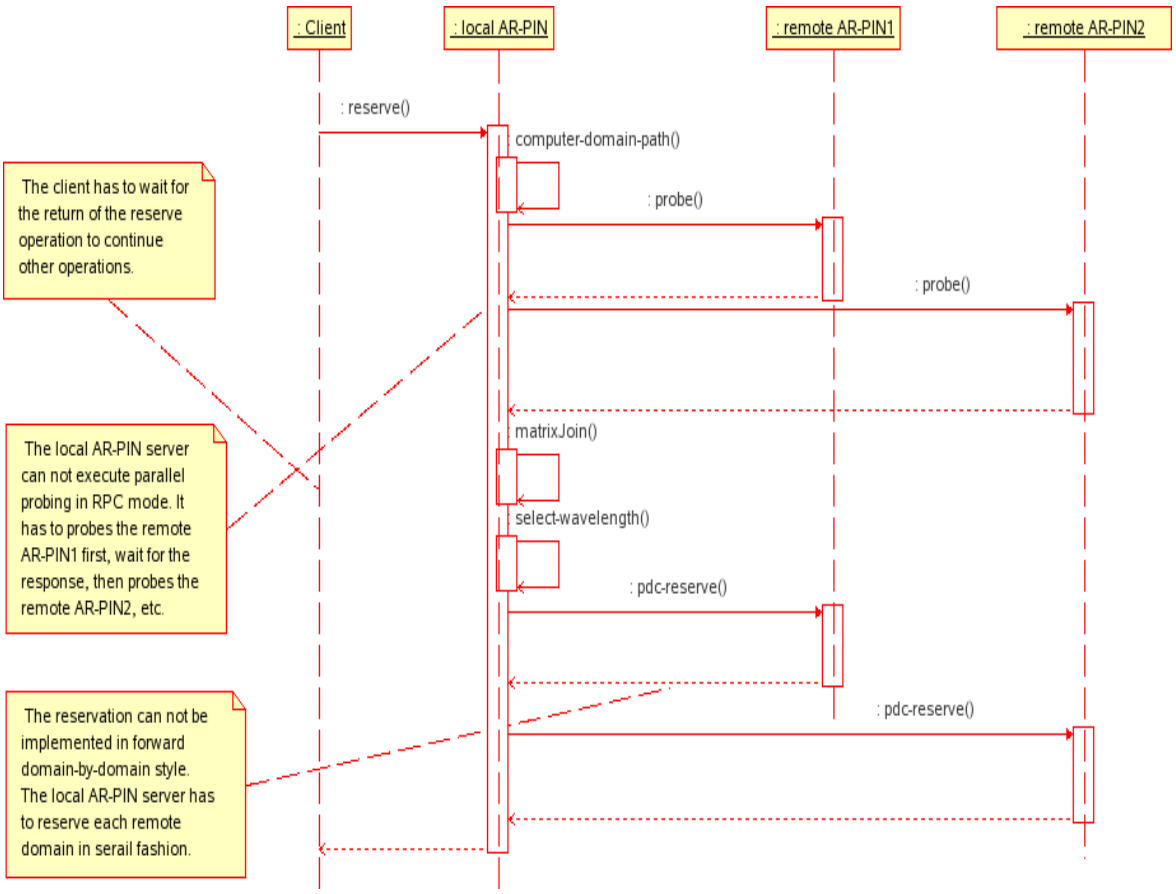

Figure 6-2. AR-PIN in Synchronous Web Service Mode

When the AR-PIN works in Synchronous mode, The local AR-PIN server can not execute parallel probing in RPC mode. It has to probe the remote AR-PIN1 first, wait for the response, then probes the remote AR-PIN2, etc. For the same reason, the reservation can not be implemented in forward domain-by-domain style. The local AR-PIN server has to reserve each remote domain after it receives the response from the previous domain.

6.3.4 Asynchronous Mode Interdomain Reservation

In Figure 6-3, I implemented all the communications between AR-PIN servers using asynchronous web services. We can see that parallel probing can be achieved by sending

probing messages to all remote domains. An advantage is that the local AR-PIN server doesn't have to be blocked and wait for the responses from remote servers, it can continue other operations, for example, accepting next request from clients. The domain-level forward reservation can also be easily implemented using message-based web services. One requirement on this kind of message-centric web service invocation is that the application server needs to ensure the robust delivery of messages.

For the interaction between the client and the local PIN-server, if the web service client wants to receive asynchronous the response from the server, it needs to provide a callback endpoint capable of receiving and processing response messages. It is possible for some complicated client applications to include a light-weight message server or application server. However, most time the client wants to avoid the message server and the complicated callback mechanism, then a technique know as polling can be used as an alternative. This technique requires the client to periodically call the server to check for the status.

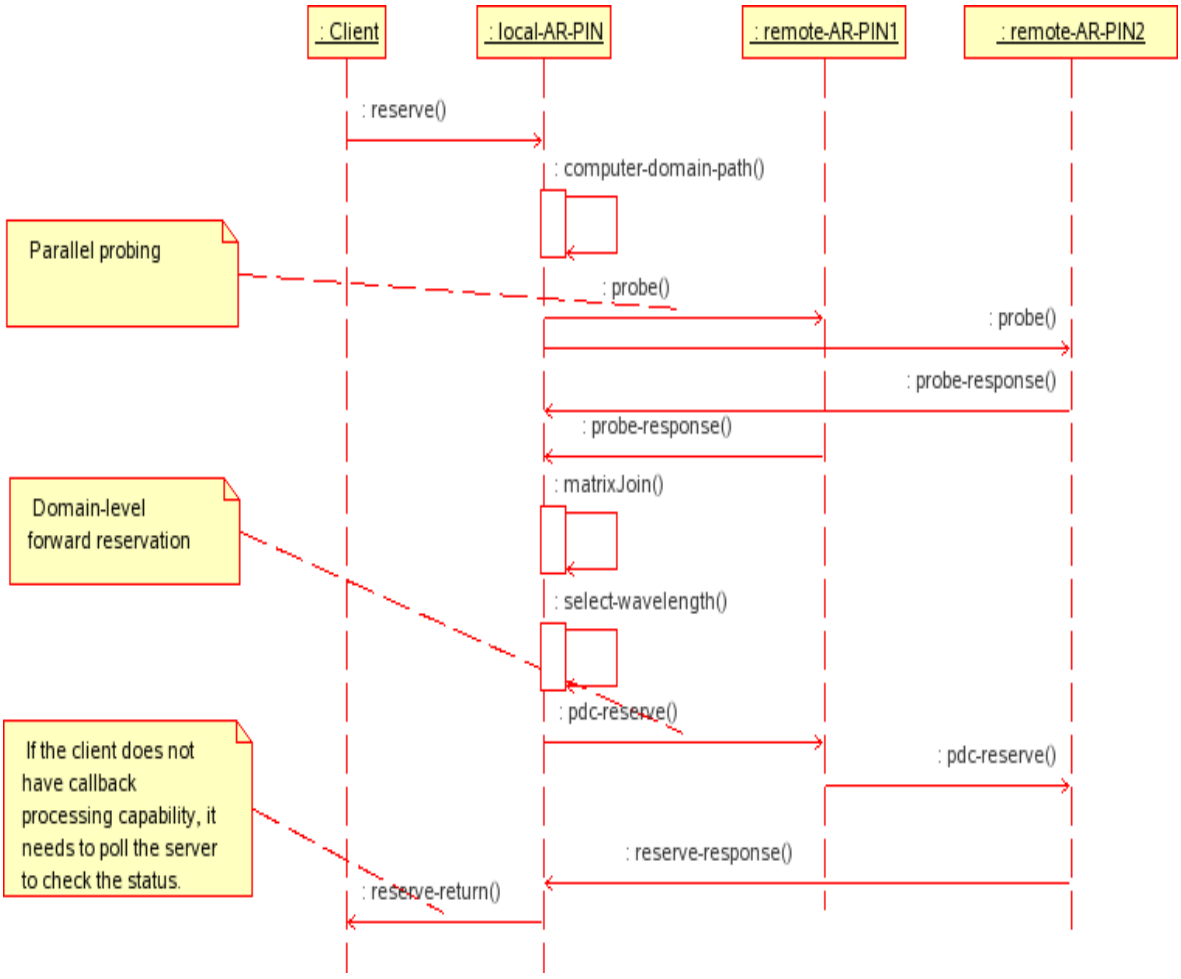

Figure 6-3. AR-PIN in Asynchronous Web Service Mode

When AR-PINs are implemented in asynchronous mode, the web services are fired by sending and receiving messages between peers. In this mode, the parallel probing and domain-level forward reservation can be implemented.

6.4 AR-PIN/PDC Web Interface

Other than web services that the AR-PIN/PDC software provides, a web interface is also written for interactively reserve lightpaths and view the reservation status. Figure 6-4 is the main interface to reserve a lightpath. The user selects the source and destination cluster, node and NIC, specifies the time range within which the reservation should be made, and the minimum duration, then a reservation message will be sent to AR-PIN/PDC. After the reservation is made successfully, the reservation ID and the endpoint IP addresses will be returned and printed on screen. Figure 6-5 shows the interface that you can see the status of all the reservations. Also it is the interface that you can claim and terminate a reservation.

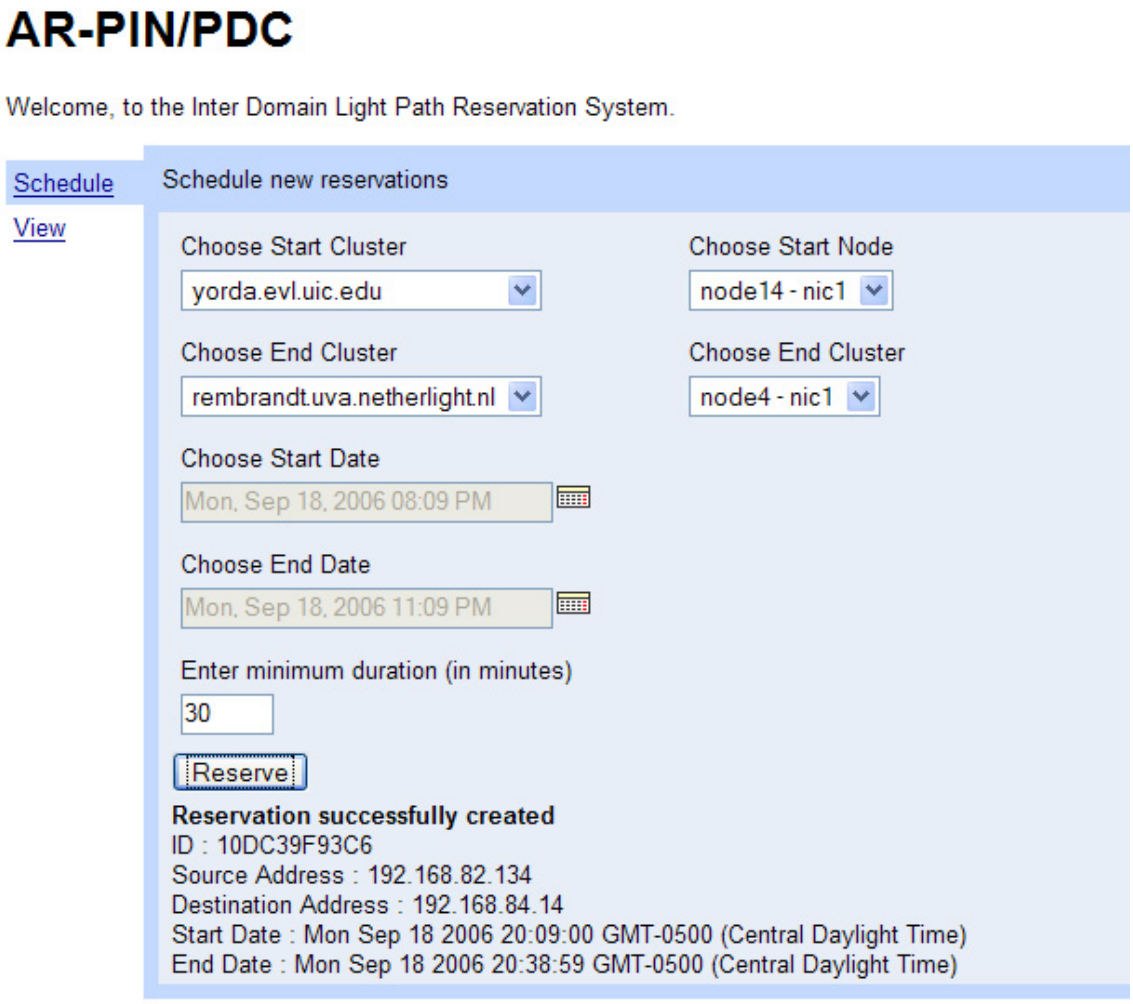

Figure 6-4. Lightpath Reservation WebInterface of AR-PIN/PDC.

The users need to select the cluster and node of the two endpoints, choose the reservation time range, minimum duration, and hit the "Reserve" button. After the reservation is fulfilled by AR-PIN server successfully, the reservation ID, IP addresses of endpoints will be returned and printed on the screen.

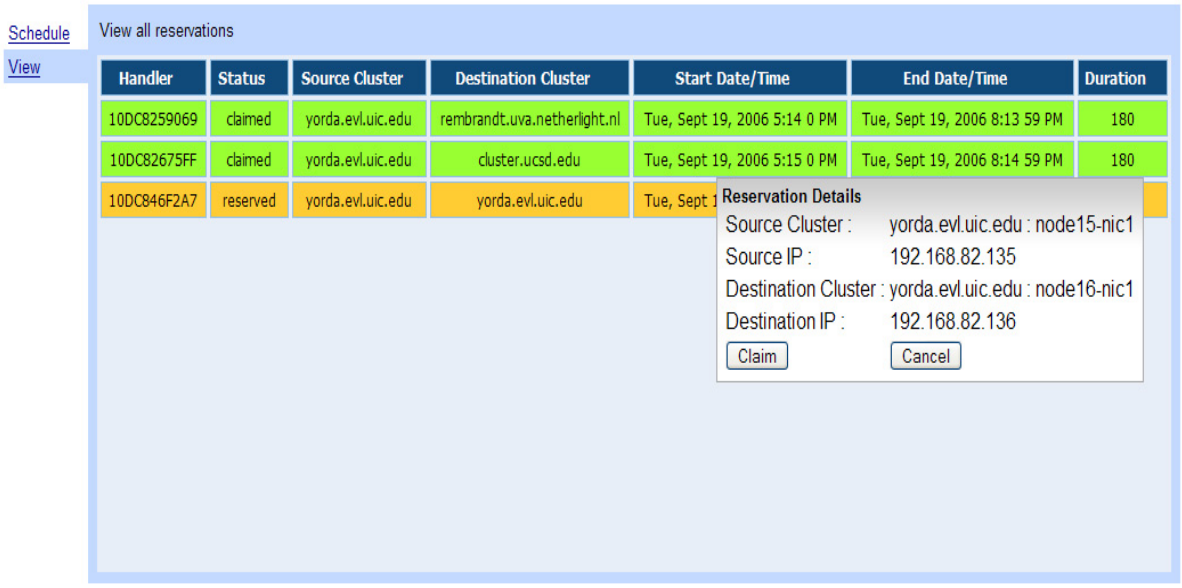

Figure 6-5. Lightpath Status Viewing Interface of AR-PIN/PDC.

In View screen, all active reservations in the local AR-PIN will be listed. When users click a reservation, a popup window showing the endpoint information will be brought up. On the window, users can click buttons to claim or terminate reservations if they are in "reserve" status, cancel or unbind reservations if they are in "claimed" status.

CHAPTER 7

DEPLOYMENT AND EXPERIMENTS

The AR-PIN/PDC software has been deployed to four sites in different continents to control four domains: University of Illinois at Chicago, Northwestern University, University of California at San Diego, and University of Amsterdam. All four AR-PIN/PDC servers have been set up to control real photonic switches to provide real lightpaths. In this chaptor, I will describe the photonic testbed in detail. Then I will show some experiment results I have done on this testbed. The objectives of the experiments are mainly two: one is to analyze the different components of the end-to-end signaling latency and compare different algorithms; the other goal is to find what are the major computational factors affecting the end-to-end signaling latency.

7.1 Testbed Deployment

The muli-domain photonic testbed consists of four domains. Each domain has one photonic switch. The detail of the four domains is listed in the table 7-1 and the topology of the testbed is depicted in Figure 7-1. There are two types of 3D MEMS switches. One is

Calient DiamandWave® PXC photonic switch, [Calient] the other type is manufactured by Glimmerglass Networks. Other than the StarLight domain having a Calient switch, all three domains contains a Glimmerglass switch. [Glimmerglass]

The AR-PIN/PDC servers are deployed on Jboss 4.0.4 application servers. JBoss is an

Domain ID	AR-PIN/PDC server	Organization	Photonic Switch
1	iching.evl.uic.edu	Electronic Visualization Laboratory University of Illinois at Chicago (EVL)	Glimmerglass
2	gjall.sl.startap.net	StarLight, downtown Chicago Northwestern University (SL)	Calient
3	calit2-host8.optiputer.net	University of California at San Diego (UCSD)	Glimmerglass
4	remrandt0.uva.netherlight.nl	University of Amsterdam (UvA)	Glimmerglass

Table 7-1 Detail of four domains in the photonic testbed.

Domain	Cluster Name	Node Name	NIC Name
EVL	yorda.evl.uic.edu	node11-node16	nic1
EVL	scylla.evl.uic.edu	node11-node16	nic1
SL	charybdis.sl.startap.net	node1	nic1, nic2
SL	atlas.sl.startap.net	node1	nic1, nic2
UCSD	cluster.ucsd.edu	node1-node4	nic1
UvA	rembrandt.uva.netherlight.nl	Node3-node6	nic1

Table 7-2 Detail of computing clusters in the photonic testbed.

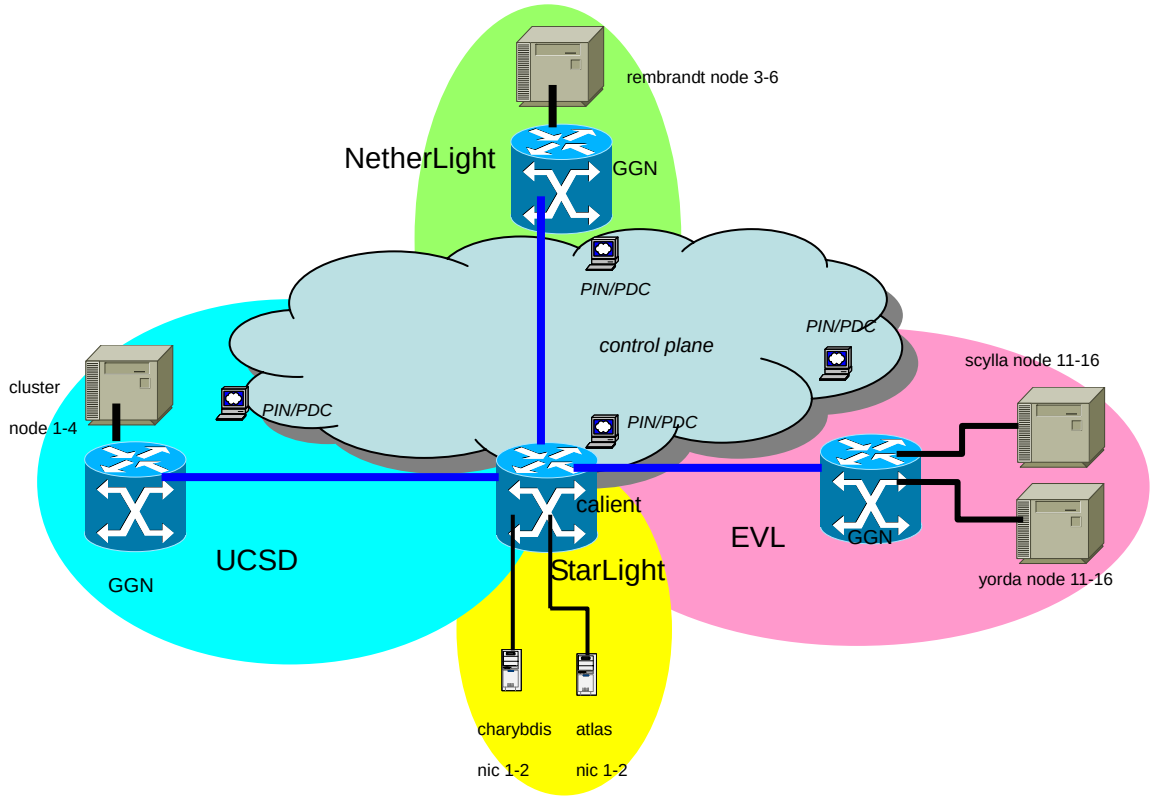

Figure 7-1. AR-PIN/PDC Multi-domain Photonic Testbed Topology.

The muli-domain photonic testbed consists of four domains: EVL/UIC, StarLight at Chicago downtown, UCSD and UvA. Each domain has one photonic switch. The detail of the four domains is listed in the table 7-1 and the topology of the testbed is depicted in this Figure. There are two types of 3D MEMS switches. One is Calient DiamandWave® PXC photonic switch, the other type is manufactured by Glimmerglass Networks. Other than the StarLight domain having a Calient switch, all three domains contains a Glimmerglass switch. There are one or more computing clusters connected to each photonic switch. One AR-PIN/PDC server is running on each domain.

open source Java EE-based application server implemented in Java. I chose JBoss application server because it's very stable, the services can be hot-deployed, and it provides robust Java messaging service. All servers run Linux operating systems with 2.6.0 or above kernel version, although they have different flavors such as SUSE, Debian or ROCK.

7.2 Experimental Results

As I mentioned in section 6.3, AR-PIN/PDC can be implemented in synchronous or asynchronous mode. The asynchronous mode can be much more efficient and lower end-to-end latency than the synchronous mode. The experiments performed in this section are based on the asynchronous implementation. In section 7.2.1 and 7.2.2, I will measure different components in the end-to-end latency of inter-domain reservation process and inter-domain claim process repectively. In section 7.2.3, I will investigate how the time slot granularity affects the computation time, in turn the end-to-end latency.

7.2.1 Components of Inter-domain Reservation Latency

The components of inter-domain reservation can be divided into two categories: propagation time and processing time. Propagation time is the period from when the sending server sends the control message to when the receiver receives the message. Actually it

includes the time on wire and the marshalling/unmarshalling time. In Figure 6-3, I showed the sequence diagram of inter-domain reservation process in asynchronous mode. I define the components of end-to-end inter-domain reservation latency ordered by time as follows:

(1) Propagation time from the client to the local AR-PIN server t_{cs}.

(2) Time to check domain-level path and prepare probing messages: t_{proc1}.

(3) Propagation time of the probing message from the local AR-PIN server to the remote AR-PIN server *i* during probing process : $t_{p\text{-}prop\text{-}f}[i]$.

(4) Probing time at remote AR-PIN server *i* : $t_{probing}[i]$.

(5) Propagation time of the probe-response message from the remote AR-PIN server *i* to the local server : $t_{p\text{-}prop\text{-}b}[i]$.

(6) Time to join all returned matrices and find the reservation solution, reserve the local domain: t_{proc2}.

(7) Propagation time of the reserve message from the AR-PIN server *i-1* to next hop *i* : $t_{r\text{-}prop\text{-}f}[i]$.

(8) Reservation time at remote AR-PIN server *i* : $t_{resv}[i]$.

(9) Propagation time of the reserve-response message from the destination AR-PIN server to the source server : $t_{r\text{-}prop\text{-}b}$.

(10) Propagation time from the local AR-PIN server to the client : t_{sc}.

Assume one domain-level route consisting of N domains (i=1 for the source domain and i=N for the destination domain) including source and destination domains, the end-to-end reservation delay can be expressed as follow:

$$T_{resv} = t_{cs} + t_{proc1} + Max(t_{p-prop-f}[i] + t_{probing}[i] + t_{p-prop-b}[i]) + t_{proc2} + \sum_{i=2}^{N}(t_{resv}[i] + t_{r-prop-f}[i]) + t_{r-prop-b} + t_{sc}$$

(7-1)

I did four sets of experiments to measure the components of end-to-end latency. They differ in the domain-level path:

(1) EVL-SL

(2) EVL-UCSD

(3) EVL-UvA

(4) EVL-SL-UCSD-UvA

All AR-PIN/PDC servers run NTP protocol to have time synchronized during measurements. I run 5 times on each case, and I took the average as results. The result is shown in Table 7-4 and depicted as a diagram in Figure 7-2. Table 7-3 shows the round trip time between each pair of AR-PIN/PDC servers.

Link	EVL-SL	EVL-UCSD	EVL-UvA	SL-UCSD	SL-UvA	UCSD-UvA
RRT(ms)	1.0	60.4	104.0	58.1	104.0	163.0

Table 7-3 The round trip time between each pair of AR-PIN/PDC servers

ms	EVL-SL	EVL-UCSD	EVL-UvA	EVL-SL-UCSD-UvA
t_{cs}	1281	1288	1292	1437
t_{proc1}	135	74	73	215
$MAX(t_{p\text{-}prop\text{-}f}[i])$	31	572	910	967
$MAX(t_{probing}[i])$	88	118	227	212
$MAX(t_{p\text{-}prop\text{-}b}[i])$	26	588	920	1031
t_{proc2}	628	835	708	833
$t_{r\text{-}prop\text{-}f}[2]$	39	772	1338	62
$t_{resv}[2]$	226	205	220	212
$t_{r\text{-}prop\text{-}f}[3]$	N/A	N/A	N/A	867
$t_{resv}[3]$	N/A	N/A	N/A	361
$t_{r\text{-}prop\text{-}f}[4]$	N/A	N/A	N/A	1602
$t_{resv}[4]$	N/A	N/A	N/A	207
$t_{r\text{-}prop\text{-}b}$	25	630	1211	1353
t_{sc}	84	69	78	71

Table 7-4 Inter-domain reservation measurements

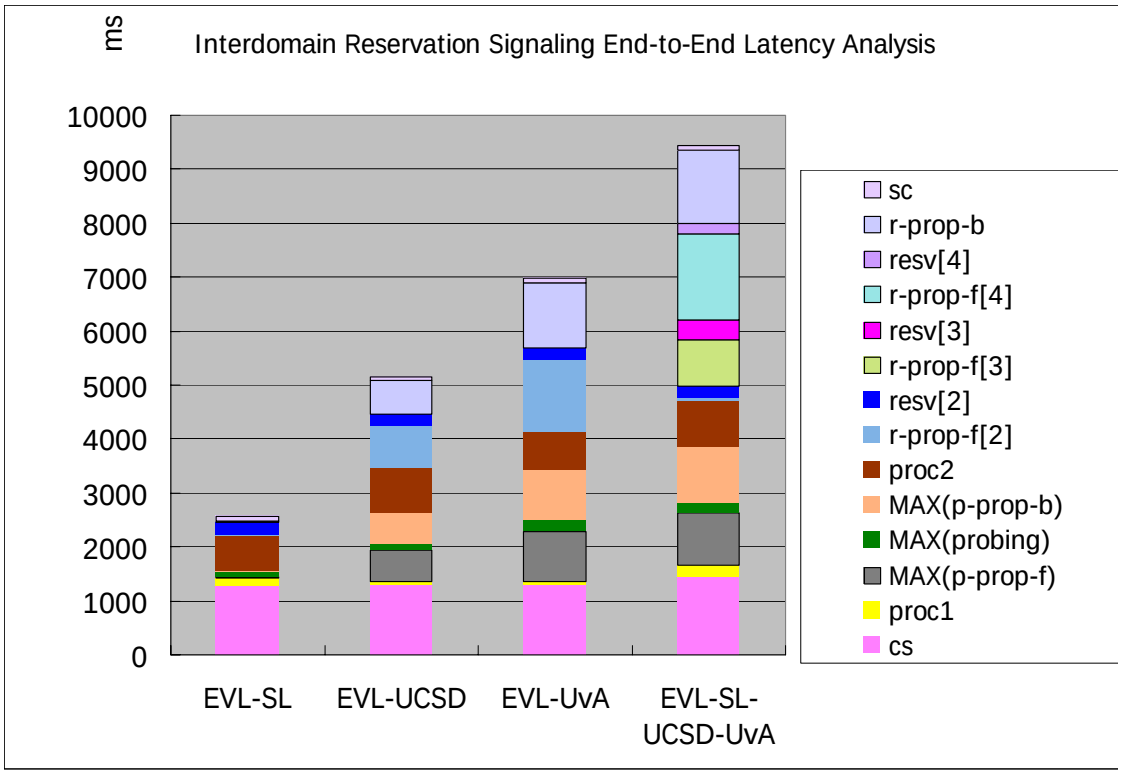

Figure 7-2. Interdomain Reservation Signaling End-to-End Latency Analysis

The Figure shows the components of inter-domain reservation signaling latency. The four sets of experiments run on EVL-SL, EVL-UCSD, EVL-UvA, and EVL-SL-UCSD-UvA lightpaths respectively. The upward diagonal strip parts are propagation delay between AR-PINs, the downward diagonal strip parts are propagation delay between client and AR-PIN server. The solid parts are processing delay. From the Figure, the major delay is propagation delay. The propagation delay is proportional to Round Trip Time between parties. The entire end-to-end delay divides into two parts: probing process and reservation process. The probing process is parallel, therefore the four domain case has similar delay to the two domain case. The reservation process is serial, therefore the four domain is much longer.

If I add $t_{p\text{-}prop\text{-}f}[i]$ *and* $t_{p\text{-}prop\text{-}b}[i]$ together, noted as $t_{p\text{-}prop}[i]$ the sum should have a relation with the round trip time from the local server to the remote server $t_{rrt}[i]$. When I compare these two sets of values in Table 7-5, I find that the actual value is much large than ping RTT time. I guess that there are two factors. One is a close to constant marshalling and

unmarshalling time *a*. The other factor is that each message delivery needs multiple RTTs, assuming *b* RTTs. The relation should be

$$t_{p-prop}[i] = a + b * t_{rtt}[i] \quad (7\text{-}2)$$

If I try to fit the curve to the real experimental values, I get a=18.5, b=40ms, which means each marshalling/unmarshalling takes about 20 ms and each message delivery in JBoss takes about 9 round trip times.

Another phenomenon I noticed is that the t_{cs} is very large for all four cases, the average is about 1.3 seconds, I don't know how to explain this yet. The only thing I can guess right now is that JBoss application server takes significant time to load the enterprise Java beans.

Path	$t_{p\text{-}prop}[i](ms)$	$t_{rrt}[i](ms)$
EVL-SL	57	1
EVL-UCSD	1160	60.4
EVL-UvA	1998	104

Table 7-5 Comparison of actual and theoretical RTTs.

7.2.2 Components of Inter-domain Reservation Claim Latency

I define the components of end-to-end inter-domain reservation claim latency ordered by time as follows:

(1) Propagation time from the client to the local AR-PIN server t_{cs}.

(2) Propagation time of the claim message from the local AR-PIN server to the remote

AR-PIN server *i* during probing process : $t_{c\text{-}prop\text{-}f}[i]$.

(3) Claiming time at remote AR-PIN server *i* : $t_{claiming}[i]$.

(4) Propagation time of the claim-response message from the remote AR-PIN server *i* to the local server : $t_{c\text{-}prop\text{-}b}[i]$.

(5) Propagation time from the local AR-PIN server to the client : t_{sc}.

Assume one domain-level route consisting of N domains (i=1 for the source domain and i=N for the destination domain) including source and destination domains, the end-to-end reservation delay can be expressed as follow:

$$T_{claim} = t_{cs} + Max(t_{c-prop-f}[i] + t_{claiming}[i] + t_{c-prop-b}[i]) + t_{sc} \quad (7\text{-}3)$$

I did the same set of experiments for the interdomain claiming process. The result is shown in Table 7-6 and Figure 7-3.

People usually are not in hurry during reservation process while they hope claiming process as short as possible, because the claiming process is often initiated by applications such as remote visualization. They hope the lightpaths can be set up quickly and data can start to flow on them early. I can see the parallel claim effectively shorten the end-to-end latency comparing to the serial reservation process. Usually the end-to-end claim latency is dominated by the slowest or the farthest optical switch. From Figure 7-3, we can see the StarLight domain always has much faster response comparing to other domains. That's because Calient optical switches has much faster switching speed than Glimmerglass

switches. Specifically, the Calient switches have about 500 ms switching time and the Glimmerglass switches have about 1300 ms switching time. Both types of switches are controlled by communicating TL1 commands over the management port. I was informed by Glimmerglass engineers that they have C library which supports much faster switching than TL1 interface. After preliminary experiments, I got very positive result on their C library. The switching time can be reduced from 1300 ms over TL1 to 44 ms over C library. It is not trivial work to incorporate the C library into the JBoss architecture. But this will be one of our future work.

ms	EVL-SL	EVL-UCSD	EVL-UvA	EVL-SL-UCSD-UvA
t_{cs}	182	182	182	263
$t_{c\text{-}prop\text{-}f}[1]$	5	4	4	5
$t_{claiming}[1]$	1924	2029	2102	1943
$t_{c\text{-}prop\text{-}b}[1]$	43	19	42	38
$t_{c\text{-}prop\text{-}f}[2]$	45	N/A	N/A	39
$t_{claiming}[2]$	330	N/A	N/A	363
$t_{c\text{-}prop\text{-}b}[2]$	33	N/A	N/A	48
$t_{c\text{-}prop\text{-}f}[3]$	N/A	528	N/A	519
$t_{claiming}[3]$	N/A	1303	N/A	1211
$t_{c\text{-}prop\text{-}b}[3]$	N/A	539	N/A	544
$t_{c\text{-}prop\text{-}f}[4]$	N/A	N/A	1195	1234
$t_{claiming}[4]$	N/A	N/A	1047	1050
$t_{c\text{-}prop\text{-}b}[4]$	N/A	N/A	912	954
t_{sc}	80	79	79	80

Table 7-6 Inter-domain reservation claim measurements

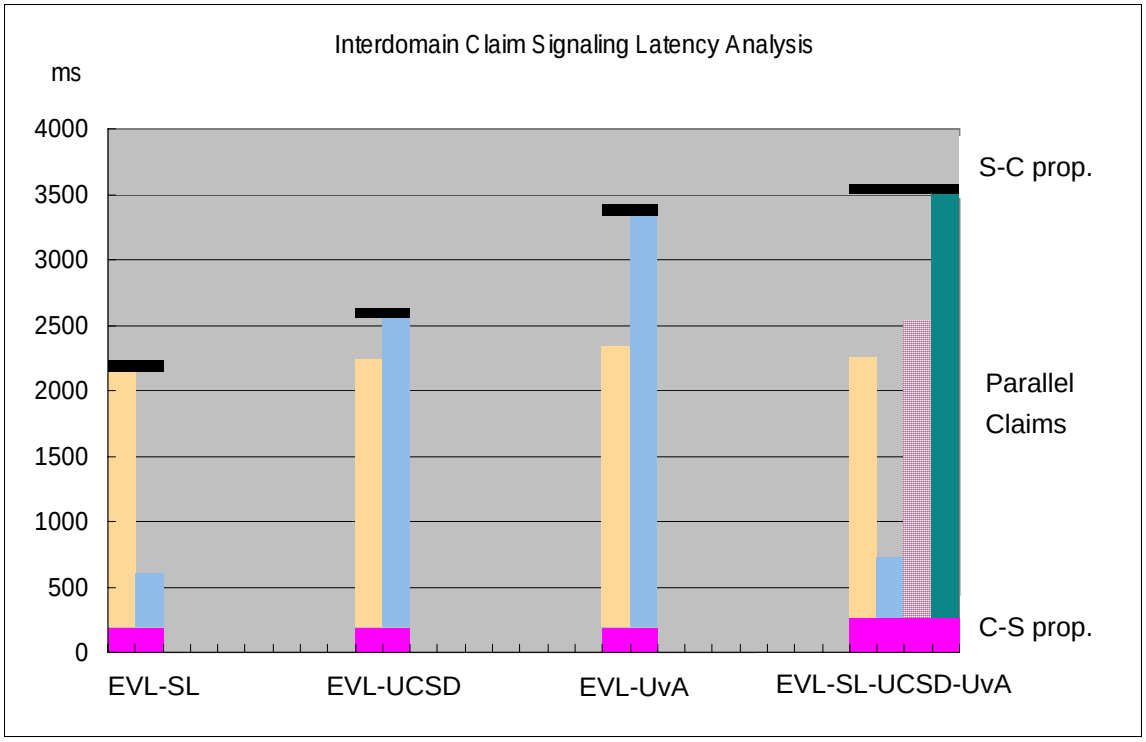

Figure 7-3. Interdomain Reservation Claim Signaling End-to-End Latency Analysis

This Figure shows the components of inter-domain claim signaling latency. Apparently, the dominant component is the parallel claim, which is the sum of the time during which the photonic switches set up the cross connects and the propagation delay. The parallelism effectively reduces the end-to-end delay. The claim time of StarLight is much shorter because the calient switch responses much faster.

7.2.3 Effect of Time Slot Granularity

In section 4.4, I analyzed the computation complexity of different algorithms, and mentioned that database access will be the most time-consuming part in the algorithms. The database access complexity of pdc-probe() is O(h), in which h is the number of hops. The database access complexity of pdc-reserve() is O(h*t), in which h is the number of hops and t is

the number of time slots in the reservation window. Usually the number of hops won't be large in reality, but the number of time slots could be very large when the time slot granularity is small. I ran a set of experiments to compare the time consumed by pdc-reserve() under different time slot granularities. The result is shown in Figure 7-4. We can see that the client-server propagation and the server-client propagation is pretty stable but the server processing time is almost proportional to the time slot granularity. Specifically, when the slot granularity is 1, i.e., one time slot is one minute, the server processing time is 0.36 seconds; when the slot granularity is 60, i.e., one time slot is one second, the server processing time is 13.8 seconds. This result matches our earlier algorithm analysis pretty well. Therefore, I set a maximum time slot granularity in AR-PIN/PDC to avoid the excess reservation time.

7.3 Summary

Through the experiments, I draw two main conclusions. Firstly, the parallelism in reservation and claim processes effectively reduce the end-to-end signaling latency. Secondly, the time slot granularity is the major factor affecting the computation time.

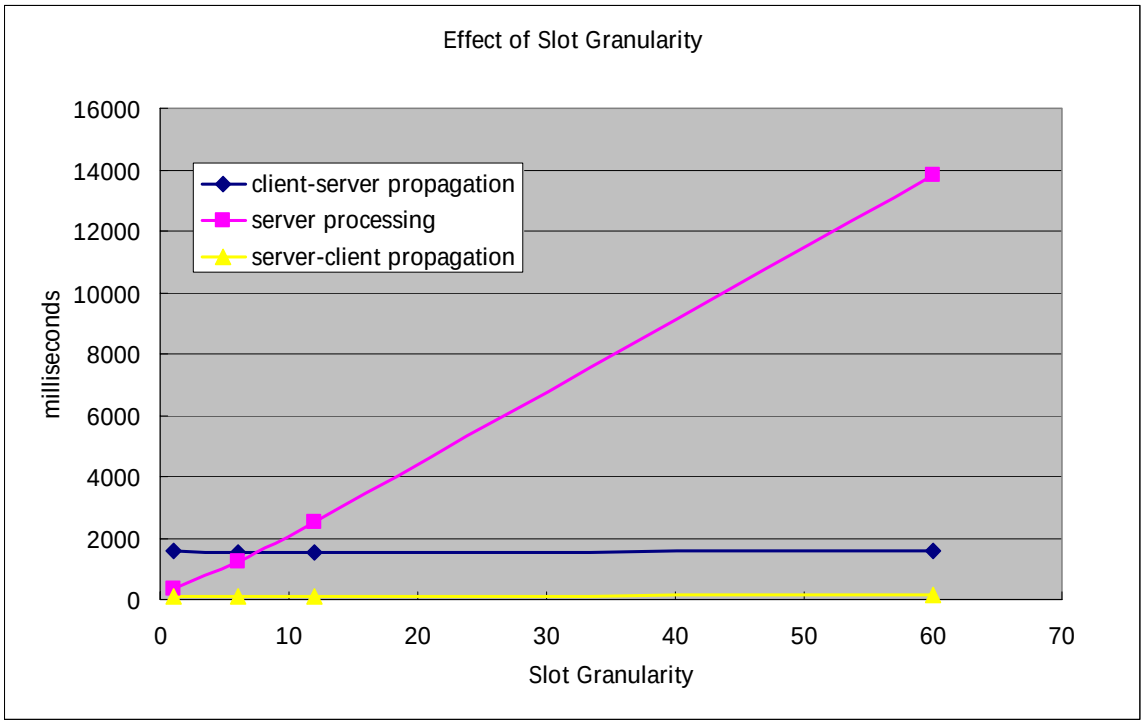

Figure 7-4. Effect of Time Slot Granularity on Reservation Processing Time

This Figure shows different latency components with the increasing of the time slot granularity. The propagation parts are quite constant while the server processing time is almost proportional to the time slot granularity. This conforms to the algorithm complexity analysis before.

CHAPTER 8

Reliable Blast UDP – an Advance Data Transmission Protocol over Photonic Networks

With AR-PIN/PDC, high performance applications can have dedicated lightpaths with multiple gigabits of bandwidth available. Full use of this bandwidth is the goal of new generation of transport protocols. At the transport layer, there is already consensus among network researchers that the current TCP implementations are not suitable for long distance high performance bulk data transfer. Either TCP needs to be modified radically or new transport protocols should be introduced. Reliable Blast UDP (RBUDP) was developed to fill the gap. [He02]

8.1 The problem of Bulk Data Transfers

Even if networked applications could make Gigabit "lambda reservations," it does not however guarantee that they will be able to make full use of that bandwidth. This problem is particularly evident when one attempts to perform large bulk data transfers over long distance, high speed networks (often referred to as "long fat networks" or LFNs) [Stevens94].

LFNs such as those between the US and Europe or Asia have extremely high round-trip latencies (at best 120ms). This latency results in gross bandwidth *under*-utilization when TCP is used for data delivery. This is because TCP's windowing mechanism imposes a limit on the amount of data it will send before it waits for an acknowledgement. The long delays that occur over international networks means that TCP will spend an inordinate amount of time waiting for acknowledgments, which in turn means that the client's data transmission will never reach the peak available capacity of the network. Traditionally this is "remedied" by adjusting TCP's window and buffer sizes to match the *bandwidth * delay* product (or capacity) of the network. For example, for a 1Gbps connection between Chicago and Amsterdam, with an average round trip time of 110ms, the capacity is 1024*0.11/8 = 14.1 Mbytes. Adjusting TCP window size is problematic for several reasons: firstly, on some operating systems (such as IRIX for the SGI,) the window size can only be modified by building a new version of the kernel- hence this is not an operation a user-level application can invoke. Secondly, one needs to know the current capacity of the network in order to set the window size correctly. The current capacity varies with the amount of background traffic already on the network and the path to the destination.

Several alternative solutions are possible. One solution is to provide TCP with better estimates of the current capacity of a link. This is the approach of the WEB100 Consortium [Web100]. The consortium is developing techniques to modify router operating systems to

report available bandwidth over a network link. Furthermore they are modifying operating systems kernels to allow better monitoring of TCP performance. Another solution is to use striped (or parallel) TCP [Park00, Leigh01, Allcock01]. In parallel TCP, the payload is divided into N partitions which are delivered over N TCP connections. Both Leigh (in CAVERNsoft) and Allcock (in GridFTP) have shown that parallel TCP can provide throughput as high as 80% of a network's available bandwidth, however its performance is unstable when excessive numbers of sockets are used. Furthermore it is difficult to predict the correct number of sockets to use.

In this research I take a more aggressive approach by using UDP augmented with aggregated acknowledgments to provide a reliable bulk data transmission scheme. I call this Reliable Blast UDP (RBUDP). A similar scheme called NetBLT was first proposed in 1985 (RFC969) by Clark et al [Clark87]. I extend Clark's work by providing both analytical and experimental results to show that RBUDP can provide the performance predictability that is lacking in parallel TCP. Furthermore I will provide an equation similar to TCP's *bandwidth*delay* product to allow one to predict RBUDP performance. This prediction will be useful in the future, for network resource reservation on the Grid.

It is important to remember that I intend aggressive protocols such as parallel TCP and Reliable Blast UDP for high speed dedicated links or links over which quality of service is available. I do not intend these protocols for use over the broader Internet.

8.2 Reliable Blast UDP

Reliable Blast UDP has two goals. The first is to keep the network pipe as full as possible during bulk data transfer. The second goal is to avoid TCP's per-packet interaction so that acknowledgments are not sent per window of transmitted data, but aggregated and delivered at the end of a transmission phase. Figure 8-1 below illustrates the RBUDP data delivery scheme. In the first data transmission phase (A to B in the figure), RBUDP sends the entire payload at a user-specified sending rate using UDP datagrams. Since UDP is an unreliable protocol, some datagrams may become lost due to congestion or an inability of the receiving host from reading the packets rapidly enough. The receiver therefore must keep a tally of the packets that are received in order to determine which packets must be retransmitted. At the end of the bulk data transmission phase, the sender sends a DONE signal via TCP (C in the figure) so that the receiver knows that no more UDP packets will arrive. The receiver responds by sending an Acknowledgment consisting of a bitmap tally of the received packets (D in the figure). The sender responds by resending the missing packets, and the process repeats itself until no more packets need to be retransmitted.

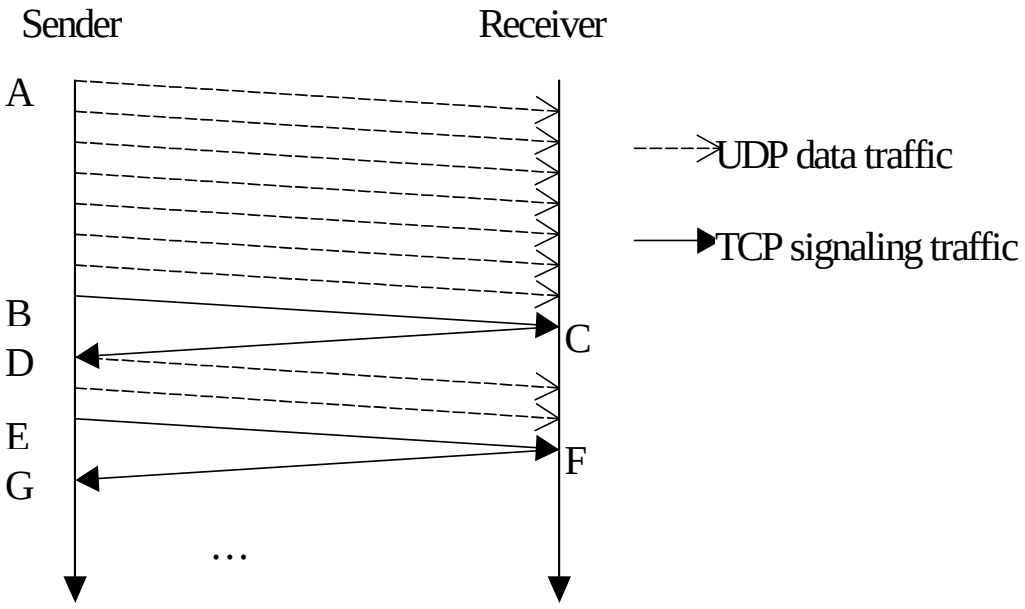

Figure 8-1. The Time Sequence Diagram of RBUDP

In RBUDP, the most important input parameter is the sending rate of the UDP blasts. To minimize loss, the sending rate should not be larger than the bandwidth of the bottleneck link (typically a router). Tools such as Iperf [Iperf] and netperf [Netperf] are typically used to measure the bottleneck bandwidth. In theory if one could send data just below this rate, data loss should be near zero. In practice however, other factors need to be considered. In our first implementation of RBUDP, I chose a send rate of 5% less than the available network bandwidth predicted by Iperf. Surprisingly this resulted in approximately 33% loss! After further investigation I found that the problem was in the end host rather than the network. Specifically, the receiver was not fast enough to keep up with the network while moving data from the kernel buffer to application buffers. When I used a faster computer as the receiver,

the loss rate decreased to less than 2%. The details of this experiment are further discussed in Section 5.

The chief problem with using Iperf as a measure of possible throughput over a link is that it does not take into account the fact that in a real application, data is not simply streamed to a receiver and discarded. It has to be moved into main memory for the application to use. This has motivated us to produce app_perf (a modified version of iperf) to take into account an extra memory copy that most applications must perform. I can therefore use app_perf as a more realistic bound for how well a transmission scheme should be able to reasonably obtain. In the experiments detailed in Section 8.4, I however include both iperf and app_perf's prediction of available bandwidth.

Three versions of RBUDP were developed:

1. RBUDP without scatter/gather optimization – this is a naïve implementation of RBUDP where each incoming packet is examined (to determine where it should go in the application's memory buffer) and then moved there.

2. RBUDP with scatter/gather optimization – this implementation takes advantage of the fact that most incoming packets are likely to arrive in order, and if transmission rates are below the maximum throughput of the network, packets are unlikely to be lost. The algorithm works by using readv() to directly move the data from kernel memory to its predicted location in the application's memory. After performing this

readv() the packet header is examined to determine if it was placed in the correct location. If it was not (either because it was an out-of-order packet, or an intermediate packet was lost), then the packet is moved to the correct location in the user's memory buffer.

3. "Fake" RBUDP – this implementation is the same as the scheme without the scatter/gather optimization except the incoming data is never moved to application memory. This was used to examine the overhead of the RBUDP protocol compared to raw transmission of UDP packets via Iperf.

Experiments that compare these versions of the protocol, and an analytical model of RBUDP, will be presented in Section 8.3 and 8.4 respectively.

8.3 Analytical Model for RBUDP

The purpose of developing an analytical model for RBUDP is two-fold. Firstly I wanted to develop an equation similar to the "bandwidth * delay product" equation for TCP, to allow us to predict RBUDP performance over a given network. Secondly I wanted to systematically identify the factors that influenced the overall performance of RBUDP so that I can predict how much benefit any potential enhancement in the RBUDP algorithm might provide.

First of all, all variables are defined as follows:

$B_{achievable}$ = achievable bandwidth

B_{send} = chosen send rate

S_{total} = total data size to send (ie payload)

T_{total} = total predicted send time

T_{prop} = propagation delay

$T_{udpSendi}$ = time to send UDP blast on ith iteration.

N_{resend} = number of times to resend (depends on loss%)

T_{ack} = time to acknowledge a blast (at least 1 ACK is always needed)

L_i = % packet loss on ith iteration

In our model I am attempting to predict the achievable bandwidth ($B_{achievable}$)of RBUDP:

$$B_{achievable} = \frac{S_{total}}{T_{total}} \tag{8-1}$$

Following the RBUDP algorithm, I estimate T_{total} as:

$$\begin{aligned} T_{total} &= \left(T_{prop} + T_{udpSend_0}\right) \\ &+ \left(\sum_{i=1}^{N_{resend}} \left(T_{prop} + T_{udpSend_i}\right)\right) \\ &+ \left(\left(N_{resend} + 1\right) * \left(T_{ack} + T_{prop}\right)\right) \end{aligned} \tag{8-2}$$

In (8-2), the first term is the time to send the main payload, the second term is the time to transmit missing packets, called T_{resend}, the last term is the time to send each acknowledgement.

Specifically:

$$T_{udpSend0} = \frac{S_{total}}{B_{send}}$$

$$T_{udpSendi} = \frac{L_{i-1} * S_{udpSend-1}}{B_{send}}$$

$$T_{ack} = \frac{S_{ack}}{B_{send}}$$

$$S_{ack} = \left(\frac{S_{total}}{S_{packet}} \right) / 8 \qquad T_{ack} = \left(\frac{S_{total}}{8 * S_{packet}} \right) / B_{send}$$

S_{packet} = 1.5Kbytes

Consequently:

$$T_{total} = \left(T_{prop} + \frac{S_{total}}{B_{send}} \right) + \left((N_{resend} * T_{prop}) + \sum_{i=1}^{N_{resend}} \frac{L_{i-1} * S_{udpSend_{i-1}}}{B_{send}} \right) + \left((N_{resend} + 1) * \left(\frac{S_{total}}{8 * S_{packet} * B_{send}} + T_{prop} \right) \right) \quad (8\text{-}3)$$

Given this equation, let us consider two possible situations - one where no loss occurs, and one where loss does occur. If no loss occurs, I can eliminate the middle term so that the best achievable performance can be computed using:

$$T_{best} = \left(T_{prop} + \frac{S_{total}}{B_{send}} \right) + \left(\frac{S_{total}}{8 * S_{packet} * B_{send}} + T_{prop} \right)$$

$$B_{best} = \frac{S_{total}}{\frac{S_{total}}{B_{send}} + \frac{S_{total}}{8 * S_{packet} * B_{send}} + 2T_{prop}} \quad (8\text{-}4)$$

In the denominator, $\frac{S_{total}}{8 * S_{packet} * B_{send}}$ is very small compared to other factors and can be omitted.

I can then derive the ratio of B_{best} and B_{send} as:

$$\frac{B_{best}}{B_{send}} = \frac{1}{1 + \frac{RTT * B_{send}}{S_{total}}} \quad (8\text{-}5)$$

where:

$2*T_{prop}$ is RTT (Round Trip Time).

This ratio shows that in order to maximize throughput, I should strive to minimize $\frac{RTT * B_{send}}{S_{total}}$ by maximizing the size of the data I wish to deliver. For example, given T_{prop} for Chicago to Amsterdam is 55ms, and B_{send} is 600 Mbps, and if I wish to achieve a throughput of 90% of the sending rate, then the payload, S_{total} needs to be at least 74.25 Megabytes.

In Section 8.4 (Figure 8-2) I will use equation 8-3 to compare the theoretical best rate B_{best} against experimental results, over a variety of send rates (B_{send}).

Furthermore I will compare B_{best} against experimental results with varying payload sizes (S_{total}) (Section 8.4, Figure 8-4).

Now let us turn to consider the situation where loss does occur. I will take a simplifying assumption that a constant loss rate of L occurs at every pass of the algorithm. I realize that in a real network subsequent losses in the retransmit phases should be smaller, rather than constant, because I will be retransmitting a significantly smaller payload at each iteration. However to estimate that accurately would require us to develop a model for the buffer in the intervening routers too. Hence I can take our simplifying assumption as a worst-case

estimate.

So, given loss rate L, retransmits will occur until the amount of data left is less than 1 packet. Therefore the number of retransmits required can be estimated as:

$$N_{resend} = \lfloor \log_L(S_{packet} / S_{total}) \rfloor \quad (8\text{-}6)$$

The data size of all retransmits is therefore:

$$S_{resend} = S_{total} * \frac{L(1 - L^{\lfloor \log L(Spacket / Stotal) \rfloor})}{1 - L} \quad (8\text{-}7)$$

I can now plug (8-6) and (8-7) back into equation (8-3) to produce our new estimate of $B_{achievable}$ given constant loss rate L. In Section 8.4 (Figure 8-4) I will put this prediction to use comparing an experimental situation where packet loss was observed.

8.4 Experimental Results

The testbed network consisted of an OC-48 link (2.5 Gbps) brought by SURFnet from Amsterdam to the StarLight facility in Chicago. There was little-to-no traffic on the link when the experiments were performed. Linux PCs were placed at each end of the link. The specifications of each PC is shown in **エラー! 参照元が見つかりません。** below. Wgsara (in Amsterdam) was the slower PC, Charybdis (in Chicago) was the faster one. The network bottleneck resides in the Gigabit Ethernet cards of host computers.

Host Name	CPU	Memory Size	System Bandwidth
wgsara2.phys.uu.nl (Amsterdam)	Pentium III 800 MHz	512M Bytes	238 MBytes/s
charybdis.sl.startap.net (Chicago)	XEON 1.8 GHz	512M Bytes	844 MBytes/s

Table 8-1 Specification of host PCs in the experimental testbed

In the first set of experiments, data was sent via RBUDP from the faster PC to the slower PC (from Chicago to Amsterdam). In the second set of experiments data was sent in the opposite direction. This allowed us to examine the performance of RBUDP when the bottleneck was either at the processor or in the network. The result was compared against predicted results from our analytical model. A third set of experiments examined RBUDP throughput for different payload sizes.

8.4.1 From the Fast PC to the Slow PC (Chicago to Amsterdam) – when the Bottleneck is in the Receiving Host Computer

In this experiment, Iperf measured maximum available bandwidth at 878 Mbps, and app_perf measured maximum possible throughput at 643 Mbps. In Figure 8-2 I plot these thresholds as lines across the top of the graph. Plotting the achieved throughput at various sending rates for the fake and real RBUDP algorithms, I notice that at sending rates below the network capacity, RBUDP performs well - i.e., RBUDP gives the application exactly what the application asks for. I also notice that as the sending rates approach the capacity of the

network, Fake RBUDP achieves almost the same throughput as Iperf, and the real RBUDP begins to hurt in performance because the underpowered CPU is unable to keep up with handling the incoming packets. However, as real RBUDP is able to match the maximum performance of app_perf, this means that RBUDP is making as much use of the network for useful data transfer as the CPU will allow. Finally, notice that there is a close match between our experimental results and our prediction from equation 8-4 (which estimated RBUDP performance when loss rate is zero.)

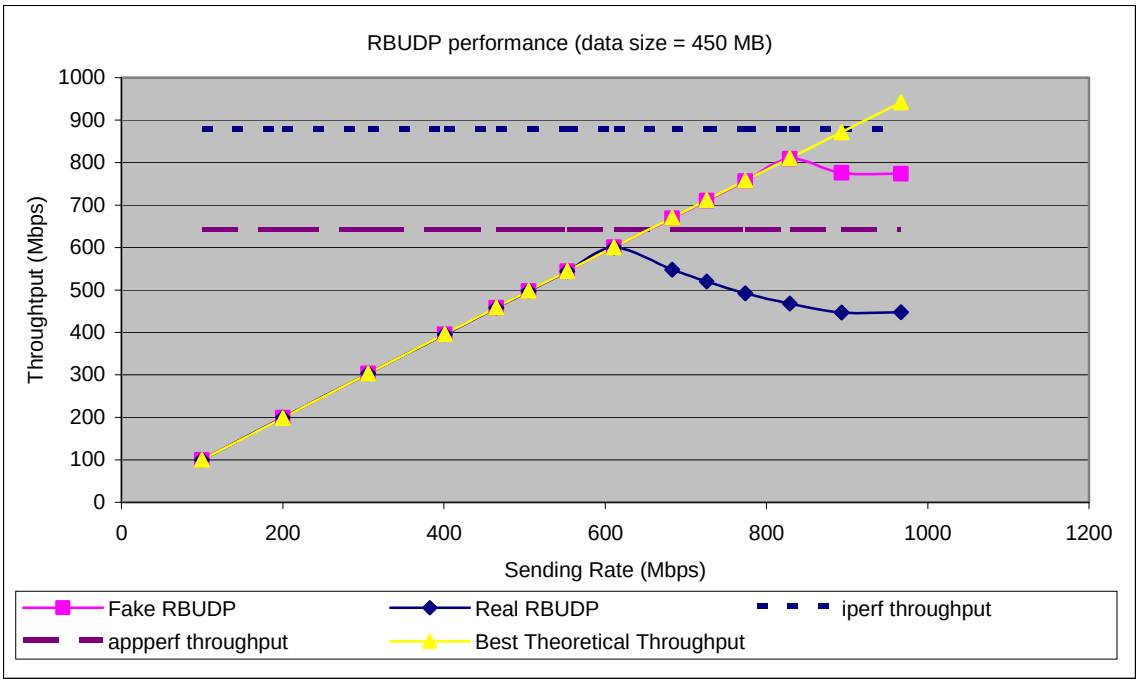

Figure 8-2. RBUDP throughput from Chicago to Amsterdam

Payload is 450MB. Bottleneck is in the receiving host. The lines indicating iperf and app_perf throughput show the maximum performance when the tools are sending at the network's full data rate. App_perf is a more realistic indication of the rate at which an application can absorb incoming data packets as it takes into account the additional overhead involved in most applications that need to take the data off the network and use it.

8.4.2 From the Slow PC to the Fast PC (Amsterdam to Chicago) – when the Bottleneck is in the Sending Host Computer

I repeated the experiment in the opposite direction. This time the bottleneck was in the sending PC rather than in the receiving PC. Figure 8-3 shows that when the host computer is fast enough, iperf and app_perf performances match, as do the different implementations of RBUDP. Fake RBUDP is able to reach the maximum performance obtained by iperf; and Real RBUDP is able to reach the maximum performance obtained by app_perf- again confirming RBUDP's ability to maximize bandwidth utilization for useful data delivery.

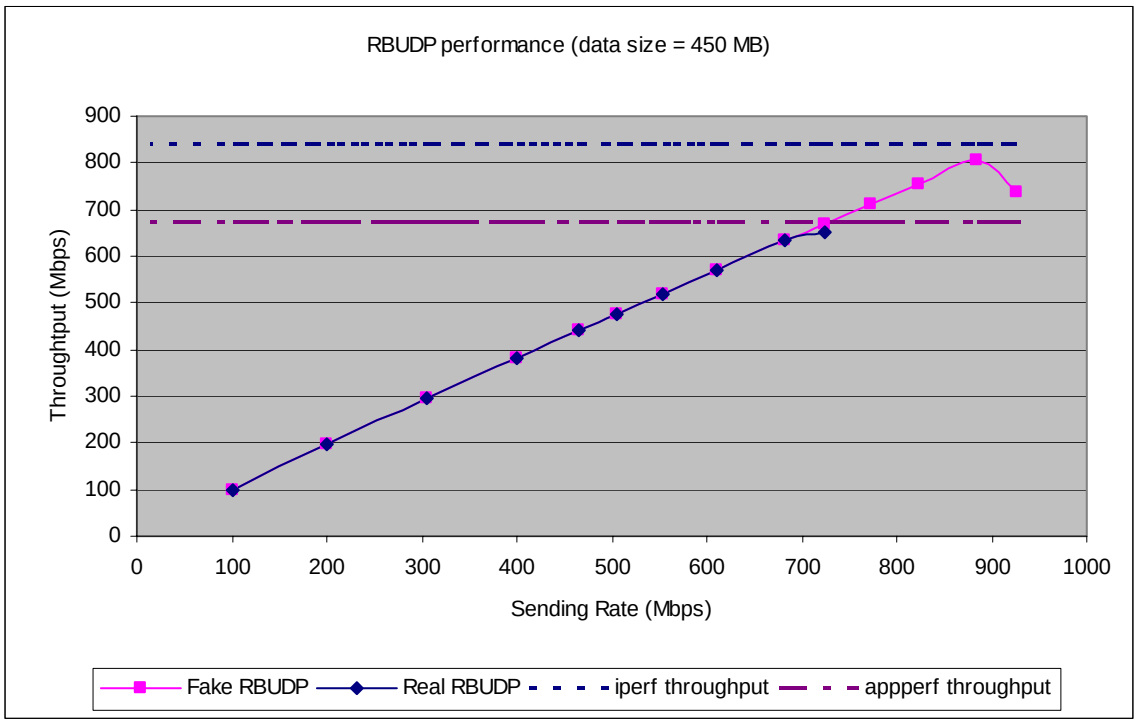

Figure 8-3. RBUDP throughput from Amsterdam to Chicago

Payload is 450MB. Bottleneck is in the sending host. The maximum of the sending rate is 725Mbps. See Figure 8-2 for an explanation of the iperf and app_perf lines in the graph.

8.4.3 Effect of Payload Size on Throughput

From the analysis in Section 8.3, I know that the propagation time is the primary factor affecting RBUDP overhead. For smaller payloads, the time spent in the acknowledgement phase is almost constant while the time spent blasting UDP packets decreases. In Figure 8-4 I compare an experimental situation where I send data at 611Mbps (experiencing no loss) against our theoretical prediction, which assumes no loss (equation 8-3.) Furthermore I compare an experimental situation sending data at 682Mbps experiencing 12% loss, against our theoretical prediction where I assume a constant 12% loss per iteration.

Firstly, the results show that RBUDP performs best for large payloads. Secondly, the results show that a 12% packet loss does not impact throughput greatly for large payloads. Finally, our analytical models provide good boundaries for our experimental results for 0% loss and 12% loss.

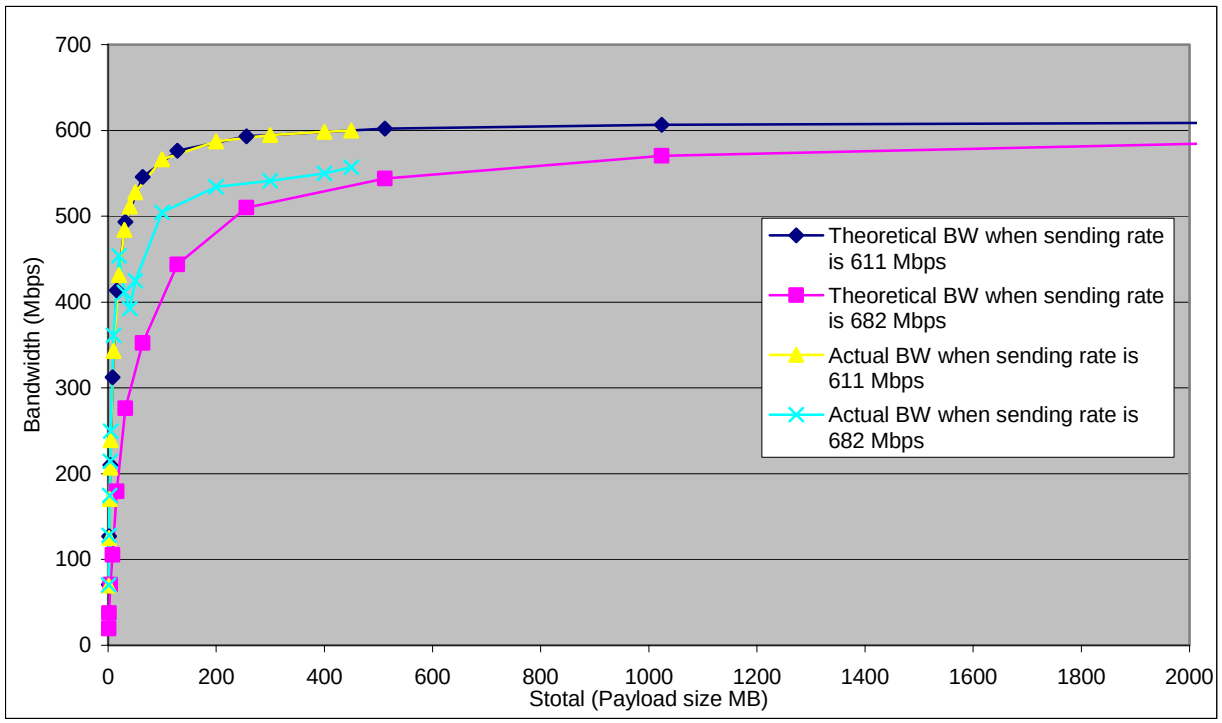

Figure 8-4. Throughput vs. Payload Size.

Larger payloads produce better network utilization

8.4.4 Adapting RBUDP for High Speed Data Streaming

Even though the initial motivation of RBUDP is for bulk data transfer over long distance, some applications require high performance reliable streaming transport. In Section 8.3, I showed that in order to achieve fairly high throughput, the payload needs to be large. In streaming applications, if the size of objects to be streamed is small, I combine multiple objects to form a large payload. However this will cause end-to-end latency to increase because of the buffering needed to form the large payloads. Based on our analytical model, I can determine the minimum sending rate needed to ensure a desired object throughput rate, given the maximum delay the application is able to tolerate.

Let:

S_{obj} = size of streamed objects.

N_{obj} = number of objects per payload.

B_{obj} = required throughput of objects. (number of objects per second)

For example, in the case of graphics streaming, object throughput rate is measured in frames per second.

D = the maximum extra delay the applcation can tolerate.

Then the size of a payload is:

$$S_{total} = S_{obj} * N_{obj} \quad (8\text{-}8)$$

where:

$$N_{obj} = B_{obj} * D \quad (8\text{-}9)$$

The required raw bandwidth is:

$$B_{best} = B_{obj} * S_{obj} \quad (8\text{-}10)$$

Assuming we are operating over an over-provisioned network, we plug (8-8), (8-9) and (8-10) back in equation (8-5) to compute the rate at which RBUDP needs to send data to achieve the application's requested throughput:

$$B_{Send} = \frac{S_{obj} * B_{obj}}{1 - \frac{RTT}{D}} \quad (8\text{-}11)$$

Hence, using a graphics streaming application as an example: given that RTT is 100ms, S_{obj} is 800*600*3, (assuming image resolution of 800x600 and 3 bytes color information for each pixel), if I want to achieve a frame rate B_{obj} of 20 frames/second, the maximum extra delay introduced will be 0.5 seconds, the sending rate needs to be at least 288 Mbps and each payload must encapsulate 10 image frames. RBUDP was deployed as an important component in software Quanta. [He03] During IGrid 2002, Luc Renambot applied Quanta's RBUDP to a parallel graphics streaming application called ***Griz***. Using our analytical model and the parameters from the above example, I was able to predict the number of animation frames that Griz had to package into a single payload to achieve full utilization of the Amsterdam-Chicago Starlight link [Renambot02].

8.5 Conclusions

RBUDP is a very aggressive protocol designed for dedicated- or QoS-enabled high bandwidth networks (such as our aforementioned DiffServ and IP-over-DWDM testbeds). It eliminates TCP's slow-start and congestion control mechanisms, and aggregates acknowledgments so that the full bandwidth of a link is used for pure data delivery. For large bulk transfers, RBUDP can provide delivery at precise, user-specified sending rates. RBUDP performs at its best for large payloads rather than smaller ones, because with smaller payloads the time to deliver the payload approaches the time to acknowledge the payload. The

scatter-gather algorithm to reduce memory copies, provides better performance over the non-scatter-gather algorithm for slower CPUs when the loss rate is not very high. This benefit is expected to increase for faster networks.

I have provided an analytical model that provides a good prediction of RBUDP performance. This prediction can be used as a rule of thumb in a manner similar to the *bandwidth * delay* product for TCP. Furthermore this prediction can be used to estimate how future ideas for improving the algorithm might impact RBUDP performance.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

In order for data-intensive distributed applications to function efficiently, they need to be able to reserve enough bandwidth, through the allocation of lightpaths. This dissertation addresses the problem of efficient scheduling of lightpaths between Grid clusters. Specifically, this dissertation seeks approaches of application-driven intra-domain and inter-domain layer 1 lightpath provisioning, with the capability of advance reservation. In this dissertation, I proposed a Flexible Advance Reservation Model (FARM) and described how to apply this model to inter-domain and intra-domain lightpath reservation problem by incorporating Routing and Wavelength Assignment algorithms. Through simulations, I found that the flexibility on time parameters of advance reservation requests can improve the system performance dramatically. And in order to maintain a well-balanced AR/IR mixed environment, both ARs and IRs need admission control. AR-PIN/PDC (Advance Reservation enabled Photonic Inter-domain Negotiator and Photonic Domain Controller) is an implementation of my design and algorithms. I deployed AR-PIN/PDC in four domains internationally. Over the testbed, some experiments were performed to measure the components of end-to-end signaling

latency. I found that the parallelism can reduce the latency effectively and the major factor affecting the computation time is the time slot granularity.

9.1 Contributions

Through the design and implementation of AR-PIN/PDC, I recognized and addressed numerous research problems in the control plane and data plane of optical networking. Specifically, this dissertation makes the following contributions.

- I created a Flexible Advance Reservation Model (FARM), applied this model to Routing and Wavelength Assignment (RWA), and designed algorithms to achieve interdomain and intradomain lightpath advance reservation. A peer-to-peer based publish/subscription topology model is used to avoid huge amount of state flooding. The On-Demand Parallel Probe algorithm renders the periodic dissemination of time-based resource availability information unnecessary and hence makes the system more scalable.

- Through simulations, I found that by introducing some flexibility on the time parameters of advance reservations, the network performance can be improved dramatically. Also it is confirmed that both Advance Reservation (AR) and Immediate

Reservation (IR) admission control are necessary in order to maintain a well-balanced AR/IR mixed environment.

- I implemented the fore-mentioned algorithms in the software AR-PIN/PDC. As a set of services, AR-PIN/PDC can be easily deployed in JBoss application server. Because it provides standard web service interfaces, writing clients is very easy in most platforms and environments.

- I deployed AR-PIN/PDC in four domains in US and Europe, making scheduling cross-continent lightpaths possible. In this testbed, I measured and analyzed the components in the end-to-end lightpath reservation and claim, and proved the parallelism of probing and claiming effectively reduces the delay, and the time slot granularity is the major factor affecting the computation time.

- Reliable Blast UDP (RBUDP) protocol was designed and implemented. This protocol is a very aggressive protocol designed for dedicated or QoS-enabled high bandwidth networks such as optical networks. For large bulk transfers, RBUDP can provide delivery at precise, user-specified sending rates. I provided an analytical model that provides a good prediction of RBUDP performance.

9.2 Future Work

This dissertation aims to let application reserve and set up layer 1 lightpaths on demand. However, in layered network model, application is layer 7 and lightpath is layer 1. In order for applications to utilize the layer 1 ligthpaths smoothly and effectively, a lot of work needs to be done from layer 2 to 6. For example, typical LambdaGrid applications have hundrends to thousands of parallel flows with different Quality of Service requirements. These flows emanate from network interfaces in the endpoints (i.e. the compute clusters) to communicate with other endpoints over multiple lightpaths. It is a challenging research problem how to multiplex and optimize m flows into n lightpaths (m>>n).

Based on functionalities, the communication network can be divided into data plane, control plane and management plane. This dissertation addresses how to set up lightpath in control plane and how to transmit bulk data over data plane. However, the management plane is also important, especially the monitoring function. The applications always want to know when the network resource is ready to use and be notified if fault occurs. The monitoring can occur in multiple layers. For example, optical layer monitoring detects the health of lightpaths; IP layer monitoring decides if the end-to-end connections can be established; transport layer monitoring judges how much bandwidth is available for applications to use.

REFERENCES

[Allcock01] W. Allcock, J. Bester, J. Bresnahan, et al., "Data Management and Transfer in High-Performance Computational Grid Environments," *Parallel Computing*, 2001.

[Ashwood03] P. Ashwood-Smith, L. Berger, "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Constraint-based Routed Label Distribution Protocol (CR-LDP) Extensions", *IETF RFC 3472*, January 2003.

[Bauer04] Christian Bauer, Gavin King, *Hibernate in Action*, Manning Publication, 2004.

[Berger03] L. Berger, "Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions", IETF RFC 3473, January 2003.

[Boutaba04] Raouf Boutaba, Wojciech Golab, Youssef Iraqi, and Bill St. Arnaud, "Lightpaths on Demand: A Web-Services-Based Management System", *IEEE Communications magazine*, July 2004, pp 2-9.

[Burchard03] Lars-Olof Burchard, Hans-Ulrich Heiss, Cesar A. F. De Rose, "Performance Issues of Bandwidth Reservations for Grid Computing", *Proc. Of the 15th Sym. On Computer Architecture and High Performance Computing (SBAC-PAD'03)*, 2003.

[Calient] http://www.calient.net

[Chu04] Xiaowen Chu, Jiangchuan Liu, Zhensheng Zhang, "Analysis of Sparse-Partial Wavelength Conversion in Wavelength-Routed WDM Networks", *IEEE INFOCOM 04*, Hongkong, March, 2004.

[Clark87] D. Clark, M. Lambert, L., Zhang, "NETBLT: A High Throughput Transport Protocol," *ACM* SIGCOMM, 1987.

[Cormen01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, "*Introduction to Algorithms*," Second Edition, The MIT Press, 2001.

[Curti05] C Curti, T Ferrari, L Gommans, S Van Oudenaarde, et al, "On advance reservation of heterogeneous network paths", *Future Generation Computer Systems*, Vol. 21, No. 4, Page 525-538, Apr. 2005.

[DeFanti03] T. DeFanti, M. Brown, J. Leigh, O. Yu, E. He, J. Mambretti, D. Lillethun, J. Weinberger, "Optical Switching Middleware for the OptIPuter," *IEICE Transactions on Communications, invited paper in special issue on Photonic IP Network Technologies for Next Generation Broadband Access.* Vol. E86-B, No. 8, pp. 2263.

[DeLaat03] Cees de Laat, E. Radius, and S. Wallace, "The Rationale of the Current Optical Networking Initiatives," *Future Generation Computer Systems*, Special Issue: iGrid 2002, Vol. 19, Page 999-1008.

[Foster99] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation**",** *International Workshop on Quality of Service*, 1999*.*

[Foster01] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, 15(3), 2001.

[Francisco01] M. Francisco, S. Simpson, L. Pezoulas, C. Huang, I. Lambadaris, and B. St. Arnaud, "Interdomain Routing In Optical Networks," *Proceedings of SPIE Opticomm 2001*, Denver, Aug. 2001.

[GLIF] http://www.glif.is.

[Glimmerglass] http://www.glimmerglass.com

[Gommans06] Leon Gommans, Freek Dijkstra, Cees de Laat, Arie Taal, Alfred Wan, Universiteit van Amsterdam, Bas van Oudenaarde, Tal Lavian, Inder Monga, Franco Travostino, "Applications Drive Secure Lightpath Creation across Heterogeneous Domains", *IEEE Communications Magazine*, vol. 44, no. 3, March 2006.

[Greenberg99] AG Greenberg, R Srikant, W Whitt, "Resource sharing for book-ahead and instantaneous-request calls", *IEEE/ACM Transactions on Networking*, Vol. 7, No. 1, Feb 1999.

[Guerin00] R Guerin, A Orda, "Networks with Advance Reservations: The Routing Perspective", *IEEE INFOCOM 2000*, Tel-Aviv, Israel, March 26-30, 2000.

[He02] Eric He, Jason Leigh, Oliver Yu, Thomas A. DeFanti, "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," *Proceedings of IEEE Cluster Computing 2002*, Chicago, September 24-26, 2002.

[He03] Eric He, Javid Alimohideen, Joshua Eliason, Naveen Krishnaprasad, Jason Leigh, Oliver Yu, Thomas A. DeFanti, "Quanta: A Toolkit for High Performance Data Delivery over Photonic Networks," *Journal of Future Generation Computer Systems (FGCS)*, Elsevier Science Press, Volume 19, Issue 6, August 2003, pp. 919-934.

[Henning06] Michi Henning, "The Rise and Fall of CORBA," *ACM Queue*, Vol. 4, No. 5, June 2006.

[Iperf] http://dast.nlanr.net/Projects/Iperf/

[Jukan04] Admela Jukan and Gerald Franzl, "Path Selection Methods With Multiple Constraints in Service-Guaranteed WDM Networks," *IEEE/ACM Transactions on Networking*, Vol. 12, No. 1, Feb. 2004.

[Kompella05] K. Kompella, Y. Rekhter, "Routing Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS)", IETF RFC 4202, October 2005.

[Lang05] J. Lang, "Link Management Protocol (LMP)", IETF RFC 4204, October 2005.

[Lehman06] Thomas Lehman, Jerry Sobieski, Bijan Jabbari, "RAGON: A Framework for Service Provision in Heterogeneous Grid Networks,"*IEEE Communications Magazine*, vol. 44, no. 3, March 2006.

[Leigh01] J. Leigh, O. Yu, D. Schonfeld, R. Ansari, et al., "Adaptive Networking for Tele-Immersion," Proc. *Immersive Projection Technology/Eurographics Virtual Environments Workshop (IPT/EGVE)*, May 16-18, Stuttgart, Germany, 2001.

[Mambretti03] Mambretti, J., Weinberger, J., Chen, J., Bacon, E., Yeh, F., Lillethun, D., Grossman, B., Gu, Y., and Mazzucco, "The Photonic TeraStream: enabling next generation applications through intelligent optical networking at iGRID2002", *Future Gener. Comput. Syst.* 19, 6 (Aug. 2003), 897-908.

[Manbretti06] Joe Mambretti, David Lillethun, John Lange, Jeremy Weinberger, "Optical Dynamic Intelligent Network Services (ODIN): An Experimental Control-Plane Architecture for High-Performance Distributed Environments Based on Dynamic Lightpath Provision," *IEEE Communications Magazine*, vol. 44, no. 3, March 2006

[Messina04] P. Messina, "Challenges of the LHC: The computing challenge," *The European Physical Journal C – Particles and Fields*, Springer Berlin/Heidelberg, Vol. 34, No. 1, May, 2004, Page 67-75.

[Mokhtar98] Ahmed Mokhtar, Murat Azizoglu, "Adaptive Wavelength Routing in All-Optical Networks", *IEEE/ACM Trans. Networking,* Vol. 6, No. 2, pp. 197-206, April, 1998.

[Netperf] http://netperf.org/netperf/NetperfPage.html

[Newman03] Harvey B. Newman, Mark H. Ellison, John A. Orcutt, "Data-Intensive E-Science Frontier Research," *Communications of the ACM*, Special Issue: Blueprint for the future of high-performance networking, Vol. 46, No. 11, 2003, Page 68-77.

[Oudenaarde05] S Van Oudenaarde, Z Hendrikse, F Dijkstra, L. Gommans, C. de Laat, R. Meijer, "Dynamic paths in multi-domain optical networks for grids," *Future Generation Computer Systems*, Vol. 21, No. 4, Page 539-548, Apr. 2005.

[Park00] K. Park, Y. Cho, N. Krishnaprasad, C. Scharver, M. Lewis, J. Leigh, A. Johnson, "CAVERNsoft G2: A Toolkit for High Performance Tele-Immersive Collaboration," *Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2000*, October 22-25, 2000, Seoul, Korea, pp. 8-15

[Renambot02] L. Renambot, T. V. D. Schaaf, H. E. Bal, D. Germans, H. J. W. Spoelder, "Griz: Experience with Remote Visualization Over Optical Grids," *Proc. IGrid 2002*, Oct, 2002.

[Renambot04] L. Renambot, et al., "SAGE: the Scalable Adaptive Graphics Environment," *Proceeding of WACE 2004,* Sep. 23-24, 2004.

[Smarr03] Larry L. Smarr, Andrew A. Chien, Tom DeFanti, Jason Leigh, Philip M. Papadopoulos, "The OptIPuter", *Communications of the ACM*, Volume 46, Number 11 (2003), Pages 58-67.

[Snell00] Quinn Snell, Mark Clement, David Jackson, Chad Gragory, "The Performance Impact of Advance Reservation Meta-Scheduling", *IPDPS 2000 Workshop, JSSPP 2000*, Cancun, Mexico, May 2000.

[Stevens94] W. R. Stevens, "*TCP/IP Illustrated*," vol. 1: Addison Wesley, 1994, pp. 344-350.

[Wang06] X. Wang, V. Vishwanath, B. Jeong, R. Jagodic, E. He, L. Renambot, A. Johnson, J. Leigh, "LambdaBridge: A Scable Architecture for Future Generation Terabit Applications," *Broadnets 2006 – Third International Conference on Broadband Communications, Networks, and Systems,* San Jose, CA, Oct. 2006.

[Yang04] Xi Yang, Byrav Ramamurthy, "Interdomain Dynamic Wavelength Routing in the Next-Generation Translucent Optical Internet", *OSA Journal of Optical Networking*, Vol. 3, No. 3, Mar. 2004.

[Yang05] X. Yang and B. Ramamurthy, "Sparse Regeneration in Translucent Wavelength-Routed Optical Networks: Architecture, Network Design and Wavelength Routing," *Springer Journal of Photonic Network Communications*, pp. 39-53, July 2005.

[Yang06] Xi Yang, Tom Lehman, Chris Tracy, Jerry Sobieski, Payam Torab, Shujia Gong, Bijan Jabbari, "Policy-Based Resource Management and Service Provision in GMPLS Networks", *Adaptive Policy-Based Management workshop*, Barcelona, Spain, April 28, 2006.

[Yu04] O. T. Yu, T. A. DeFanti, "Collaborative User-Centric Lambda-Grid over Wavelength-Routed Network," *In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, Nov 06 - 12, 2004, Washington, DC.

[Veeraraghavan03] M Veeraraghavan, X Zheng, H Lee, M Gardner, W Feng, “CHEETAH: Circuit-switched High-speed End-to-End Transport ArcHitecture”, *Proc. of Opticomm 2003*, 2003.

[Veeraraghavan06] Malathi Veeraraghavan, Xuan Zheng, Zhanxiang Huang, "On the Use of Connection-Oriented Networks to Support Grid Computing", *IEEE Communications Magazine*, vol. 44, no. 3, March 2006.

[Web100] http://www.web100.org

[Weissman98] J.B. Weissman, “Metascheduling: A Scheduling Model for Metacomputing Systems,” page 348, *Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC-7 '98)*, 1998.

[Wu05] Jing Wu, Michel Savoie, Scott Campbell, Hanxi Zhang, Gregor V. Bochmann, Bill St. Arnaud, “Customer-managed end-to-end lightpath provision”, *International Journal of Network Management*, Vol. 15, No. 5, September 2005.

[Zang00] Hui Zang, Jason P. Jue, and Biswanath Mukherjee, “A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks,” *SPIE Optical Networks Magazine*, vol. 1, no. 1, Jan. 2000.

[Zheng02] Jun Zheng and Hussein T. Mouftah, “Routing and Wavelength Assignment for Advance Reservation in Wavelength-Routed WDM Optical Networks”, *IEEE International Conference on Communications (ICC),* 2002.

APPENDIX A

AR/PIN-PDC WSDL FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PDCService" targetNamespace="http://localhost:8080/pdc-ws"
    xmlns:tns="http://localhost:8080/pdc-ws" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:ns2="http://localhost:8080/pdc-ws/types" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <schema targetNamespace="http://localhost:8080/pdc-ws/types" xmlns:tns="http://localhost:8080/pdc-ws/types"
        xmlns:soap11-enc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="PDCReserveReturn">
        <sequence>
          <element name="destAddr" type="string" nillable="true"/>
          <element name="finish" type="long"/>
          <element name="message" type="string" nillable="true"/>
          <element name="reservationId" type="string" nillable="true"/>
          <element name="srcAddr" type="string" nillable="true"/>
          <element name="start" type="long"/></sequence></complexType></schema></types>
  <message name="PDCEndpoint_advancereserveHH">
    <part name="String_1" type="xsd:string"/>
    <part name="String_2" type="xsd:string"/>
    <part name="String_3" type="xsd:string"/>
    <part name="String_4" type="xsd:string"/>
    <part name="String_5" type="xsd:string"/>
    <part name="String_6" type="xsd:string"/>
    <part name="long_7" type="xsd:long"/>
    <part name="long_8" type="xsd:long"/>
    <part name="long_9" type="xsd:long"/>
    <part name="int_10" type="xsd:int"/></message>
  <message name="PDCEndpoint_advancereserveHHResponse">
```

<part name="result" type="ns2:PDCReserveReturn"/></message>
<message name="PDCEndpoint_breakPorts">
<part name="int_1" type="xsd:int"/>
<part name="int_2" type="xsd:int"/>
<part name="int_3" type="xsd:int"/></message>
<message name="PDCEndpoint_breakPortsResponse"/>
<message name="PDCEndpoint_claim">
<part name="String_1" type="xsd:string"/></message>
<message name="PDCEndpoint_claimResponse">
<part name="result" type="xsd:int"/></message>
<message name="PDCEndpoint_connectPorts">
<part name="int_1" type="xsd:int"/>
<part name="int_2" type="xsd:int"/>
<part name="int_3" type="xsd:int"/></message>
<message name="PDCEndpoint_connectPortsResponse"/>
<message name="PDCEndpoint_getHostName"/>
<message name="PDCEndpoint_getHostNameResponse">
<part name="result" type="xsd:string"/></message>
<message name="PDCEndpoint_getNumberOfPorts">
<part name="String_1" type="xsd:string"/></message>
<message name="PDCEndpoint_getNumberOfPortsResponse">
<part name="result" type="xsd:int"/></message>
<message name="PDCEndpoint_getSwitchOutPort">
<part name="int_1" type="xsd:int"/>
<part name="int_2" type="xsd:int"/></message>
<message name="PDCEndpoint_getSwitchOutPortResponse">
<part name="result" type="xsd:int"/></message>
<message name="PDCEndpoint_init"/>
<message name="PDCEndpoint_initResponse"/>
<message name="PDCEndpoint_terminate">
<part name="String_1" type="xsd:string"/></message>
<message name="PDCEndpoint_terminateResponse">
<part name="result" type="xsd:int"/></message>
<message name="PDCEndpoint_unbind">
<part name="String_1" type="xsd:string"/></message>
<message name="PDCEndpoint_unbindResponse">
<part name="result" type="xsd:int"/></message>
<portType name="PDCEndpoint">
<operation name="advancereserveHH" parameterOrder="String_1 String_2 String_3 String_4 String_5 String_6

```
        long_7 long_8 long_9 int_10">
      <input message="tns:PDCEndpoint_advancereserveHH"/>
      <output message="tns:PDCEndpoint_advancereserveHHResponse"/></operation>
    <operation name="breakPorts" parameterOrder="int_1 int_2 int_3">
      <input message="tns:PDCEndpoint_breakPorts"/>
      <output message="tns:PDCEndpoint_breakPortsResponse"/></operation>
    <operation name="claim" parameterOrder="String_1">
      <input message="tns:PDCEndpoint_claim"/>
      <output message="tns:PDCEndpoint_claimResponse"/></operation>
    <operation name="connectPorts" parameterOrder="int_1 int_2 int_3">
      <input message="tns:PDCEndpoint_connectPorts"/>
      <output message="tns:PDCEndpoint_connectPortsResponse"/></operation>
    <operation name="getHostName">
      <input message="tns:PDCEndpoint_getHostName"/>
      <output message="tns:PDCEndpoint_getHostNameResponse"/></operation>
    <operation name="getNumberOfPorts" parameterOrder="String_1">
      <input message="tns:PDCEndpoint_getNumberOfPorts"/>
      <output message="tns:PDCEndpoint_getNumberOfPortsResponse"/></operation>
    <operation name="getSwitchOutPort" parameterOrder="int_1 int_2">
      <input message="tns:PDCEndpoint_getSwitchOutPort"/>
      <output message="tns:PDCEndpoint_getSwitchOutPortResponse"/></operation>
    <operation name="init">
      <input message="tns:PDCEndpoint_init"/>
      <output message="tns:PDCEndpoint_initResponse"/></operation>
    <operation name="terminate" parameterOrder="String_1">
      <input message="tns:PDCEndpoint_terminate"/>
      <output message="tns:PDCEndpoint_terminateResponse"/></operation>
    <operation name="unbind" parameterOrder="String_1">
      <input message="tns:PDCEndpoint_unbind"/>
      <output message="tns:PDCEndpoint_unbindResponse"/></operation></portType>
  <binding name="PDCEndpointBinding" type="tns:PDCEndpoint">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="advancereserveHH">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></input>
      <output>
        <soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></output></operation>
    <operation name="breakPorts">
```

<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></input>
<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></output></operation>
<operation name="claim">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></input>
<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></output></operation>
<operation name="connectPorts">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></input>
<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></output></operation>
<operation name="getHostName">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></input>
<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></output></operation>
<operation name="getNumberOfPorts">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></input>
<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></output></operation>
<operation name="getSwitchOutPort">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></input>
<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></output></operation>
<operation name="init">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws"/></input>

<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws/"/></output></operation>
<operation name="terminate">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws/"/></input>
<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws/"/></output></operation>
<operation name="unbind">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws/"/></input>
<output>
<soap:body use="literal" namespace="http://localhost:8080/pdc-ws/"/></output></operation></binding>
<service name="PDCService">
<port name="PDCEndpointPort" binding="tns:PDCEndpointBinding">
<soap:address location="http://localhost:8080/pdc-ws/PDCService"/></port></service></definitions>

APPENDIX B

AR/PIN-PDC JAVA CLIENT EXAMPLES

Because AR-PIN/PDC provides web services, the client could be Java-based, C-based or Python-based. In this Appendix, I will show two Java client examples The first example reserves a lightpath, and the second one claims a lightpath.

Reserve client:

```
package edu.uic.evl.pdc.client;
import java.util.*;

public class reserve {

	public static void main(String [] args) {
		String resvID, srcIP, destIP, error;
		Date now = new Date();
		long start = now.getTime() + 0;
		// Reservation window end time is 10 minutes later.
		long period = 10*60*1000;
		// The mininum duration is also 10 minutes.
		long md = 10*60*1000;
		long realStart;
		long realFinish;
		Date startDate;
		Date finishDate;
		// Source endpoint information.
		String c1 = "yorda.evl.uic.edu";
		String n1 = "node11";
```

```
                String i1 = "nic1";
                // Destination endpoint information.
                String c2 = "yorda.evl.uic.edu";
                String n2 = "node12";
                String i2 = "nic1";
                PDCReserveReturn ret = null;

                try {
                        PDCService service = new PDCServiceLocator();
                        PDCEndpoint endpoint = service.getPDCEndpointPort();
                        ret = endpoint.advancereserveHH(c1, n1, i1,
                                                        c2, n2, i2,
                                                        start, period, md, 1);
                        resvID = ret.getReservationId();
                        srcIP = ret.getSrcAddr();
                        destIP = ret.getDestAddr();
                        realStart = ret.getStart();
                        realFinish = ret.getFinish();
                        error = ret.getMessage();
                        System.out.println("Reservation ID: " + resvID);
                        System.out.println("Source IP address: " + srcIP);
                        System.out.println("Destination IP address: " + destIP);
                        System.out.println("Error Message: " + error);
                        startDate = new Date(realStart);
                        finishDate = new Date(realFinish);
                        System.out.println("Reservation Start Time: " + startDate.toString());
                        System.out.println("Reservation Finish Time: " + finishDate.toString());
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Claim client:

```
package edu.uic.evl.pdc.client;
import java.util.*;

public class claim {
	public static void main(String [] args) {
		try {
			PDCService service = new PDCServiceLocator();
			PDCEndpoint endpoint = service.getPDCEndpointPort();
			System.out.println(endpoint.claim(args[0]));
		}
		catch(Exception e) {
			e.printStackTrace();
		}
	}
}
```