UNIVERSITY OF CALIFORNIA, SAN DIEGO

Distributed Fair Bandwidth Sharing for Lambda Networks

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in
Computer Science

by

Xinran Wu

Committee in charge:

Professor Andrew A. Chien, Chair
Professor Bill Lin
Professor George Papen
Professor Alex C. Snoeren
Professor Geoffrey M. Voelker

2007

The dissertation of Xinran Wu is approved, and it is acceptable in quality and form for publication on microfilm:

Chair

University of California, San Diego

2007

For my parents, brother, and relatives who made this possible.

TABLE OF CONTENTS

LIST OF SYMBOLS

$\mathcal{V}$	Set of end nodes
$\mathcal{V}^S$	Set of sources
$\mathcal{V}^R$	Set of sinks
$\mathcal{K}$	Set of active sessions
d	Control interval
C	End node capacity
$\mathbf{x}^*$	The vector of global max-min rate allocation
$\mathbf{x}(t)$	The vector of sending rates at time t
$x_k(t)$	The sending rate of session k at time t
$\bar{\mathbf{x}}$	The vector of measured rates at time t
$\bar{x}_k(t)$	The measured rate of session k at time t
$\hat{x}_k^s(t)$	The source expected rate of session k at time t
$\hat{\mathbf{x}}$	The vector of expected rates calculated at the sink
$\hat{x}_k^r(t)$	The sink expected rate of session k at time t
rtt_k	The round-trip time of session k
α	Rate adaptation parameter
β	Rate adaptation parameter

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Professor Andrew A. Chien, for his continuous advice and encouragement for the past four years, for his patience and invaluable support, for guiding research and career directions through innumerable insightful discussions on various topics.

I also owe my gratitude to Professor Bill Lin, Professor George Papen, Professor Alex C. Snoeren and Professor Geoffrey M. Voelker for gladly agreeing to serve on my dissertation committee and providing helpful advice.

I was also fortunate to work with Professor Peter Marbach when I was studying toward my Master degree at the University of Toronto, and my mentors Dr. Matti A. Hiltunen, Dr. Richard D. Schlichting and Dr. Subhabrata Sen at AT&T Labs-Research, where I spent a wonderful summer in 2004.

I am grateful to everyone in the Concurrent Systems Architecture Group (CSAG). I thank those who went before me and gave me examples to follow: Xin Liu, Ju Wang and Huaxia Xia. I thank CSAG members on the OptIPuter project for their collaboration: Nut Taesombut, Justin Burke, Eric Weigle and Frank Uykeda. I have also learned much from my fellow CSAG members: Michela Taufer, Luis Rivera, Yang-Suk Kee, Richard Huang, Dionysios Logothetis, Jerry Chou, Ryo Sugihara, Han S Kim and Jing Zhu. Special thanks go to CSAG staffs Patricia Bladh, Alex Olugbile , Troy Chuang, Adam Brust and Jenine Combs for their generous support.

Lastly, I would like to express my profound thanks to my parents and brother for their unconditional love and support.

VITA

2000	B.S., Computer Science, Tsinghua University, Beijing, China
2002	M.S., Computer Science, University of Toronto, Canada
2007	Ph.D., Computer Science University of California, San Diego.

PUBLICATIONS

Nut Taesombut, Xinran (Ryan) Wu, Andrew A. Chien, et al, *Collaborative Data Visualization for Earth Sciences with the OptIPuter*, Journal of Future Generation Computer Systems, Pages: 955 - 963, Volume 22 , Issue 8 (October 2006).

Xinran (Ryan) Wu and Andrew A. Chien *Evaluation of End-node Based Protocols for Lambda Networks*, in Proceedings of the Fourth International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet2006), Nara, Japan, Feb 2-3, 2006

Xinran (Ryan) Wu, Andrew A. Chien, Matti A. Hiltunen, Richard D. Schlichting and Subhabrata Sen, *A High Performance Configurable Transport Protocol for Grid Computing*, in Proceedings of the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005).

Xinran (Ryan) Wu , and Andrew A. Chien, *Evaluation of Rate Based Transport Protocols for Lambda-Grids*, in Proceedings of the Thirteenth IEEE International Symposium on High-Performance Distributed Computing (HPDC), Honolulu, Hawaii, June 2004.

Xinran (Ryan) Wu, and Andrew A. Chien, *GTP: Group Transport Protocol for Lambda-Grids*, in Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), April 2004.

Xinran (Ryan) Wu, and Andrew A. Chien, *Evaluation of Rate Based Transport Protocols for Lambda-Grids*, the Second International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2004), Argonne, Illinois, Feb 2004.

Xinran Wu and Peter Marbach, *Pricing and Rate Adaptation in a Non-Cooperative Environment*, in Proceedings of the 42nd IEEE Conference on Decision and Control, Hawaii, December 2003.

Xinran Wu, *Pricing and User Adaptation for Smooth and Bursty Traffic Sources*, M.Sc. Thesis, University of Toronto, May 2002.

Xinran Wu, Clement Yuen, Yan Gao, Hong Wu, Baochun Li, *Fair Scheduling with Bottleneck Consideration in Wireless Ad-hoc Networks*, in Proceedings of

The 10th IEEE International Conference on Computer Communications and Networks(ICCCN01), Phoenix, Arizona, Oct 15-17, 2001.

Xinran (Ryan) Wu and Andrew A. Chien, *Distributed Fair Bandwidth Sharing for Lambda Networks* , In Submission.

Technical Reports

Xinran (Ryan) Wu and Andrew A. Chien, *A Distributed Algorithm for Max-Min Bandwidth Sharing: The Asynchronous Case*, UCSD Technical Report

Xinran (Ryan) Wu and Andrew A. Chien, *A Distributed Algorithm for Max-min Fair Bandwidth Sharing: The Synchronous Case*, UCSD Technical Report

Andrew A. Chien, Xinran (Ryan) Wu, Nut Taesombut, Eric Weigle, Huaxia Xia, and Justin Burke, *OptIPuter System Software Framework*, UCSD Technical Report CS2004-0786, May 2004.

FIELDS OF STUDY

Studies in Computer Networks and High-Performance Computing
Professor Andrew A. Chien

ABSTRACT OF THE DISSERTATION

Distributed Fair Bandwidth Sharing for Lambda Networks

by

Xinran Wu
Doctor of Philosophy in Computer Science
University of California, San Diego, 2007
Professor Andrew A. Chien, Chair

Dense Wavelength Division Multiplexing (DWDM), dedicated optical paths, high-speed switches and routers are giving rise to networks with plentiful bandwidth in the core. In such networks, bottlenecks and congestion are concentrated at the edge and at end nodes. Collaborative data-centric e-science applications and new multipoint-to-point and multipoint-to-multipoint communication patters raise the challenge of how to allocate bandwidth resources efficiently and fairly among active sessions.

In this dissertation, we consider how end nodes should efficiently and fairly manage capacity across multiple sessions with finite and unknown demands in such networks to support long-lived bulk data transfers. We propose a novel distributed end-node bandwidth sharing algorithm that controls each source and sink independently with local information, adaptively allocating its capacity to active sessions. This algorithm is given no knowledge of the desired session rates, but rather discovers them. We prove analytically that our distributed algorithm converges to the unique global max-min fair rate allocation from any initial or transitional states.

Simulations with different algorithm parameters, network topology and traffic patterns validate the convergence and fairness properties of our distributed algorithm. Simulations further show that the system convergence is in practice fast in networks of 32 to 1024 nodes, and our proposed approach achieves better efficiency and fairness than other high-speed transport alternatives.

We design and implement a prototype of our distributed bandwidth sharing algorithm. Experimental results on the OptIPuter networks and in emulated environments conform closely to analytical studies and simulation results, showing that the proposed approach is practically feasible.

I

Overview

Dense wavelength division multiplexing (DWDM), dedicated optical paths, high speed switches, and high speed routers are giving rise to networks with plentiful bandwidth in the core. In such networks, bottlenecks and congestion are concentrated at the edge and at end nodes. Collaborative data-centric e-science applications and new multipoint-to-point and multipoint-to-multipoint communication patters raise the challenge of how to allocate efficiently and fairly among active sessions. In this thesis, we consider how end nodes should efficiently and fairly manage capacity across multiple sessions with finite and unknown demands in such lambda networks to support long-lived bulk data transfers.

This dissertation proposes a novel distributed end-node bandwidth sharing algorithm that controls each source and sink independently with local information, adaptively allocating its capacity to candidate sessions. This algorithm is given no knowledge of the desired session rates, but rather discovers them. We prove analytically that our distributed algorithm converges to the unique global max-min fair rate allocation. Simulation and prototype experiments confirm this behavior, and further show that convergence is in practice fast in networks of 32 to 1024 nodes. Experiments also show that the approach proposed in this thesis achieves better efficiency and fairness than other high-speed transport protocols.

This chapter is organized as follows. In Section I.1, we describe the ad-

vances in optical networks that provide abundant network bandwidth. In Section I.2, we present the transport requirements of e-science applications. In Section I.3, we describe the challenges for achieving fair bandwidth sharing in lambda networks. In Section I.4, we present the thesis statement and our approach. In Section I.5, we outline the contributions of this dissertation, followed by a summary with a road map of the entire dissertation in Section I.6.

I.1 Lambda Networks Providing Abundant Network Bandwidth

Major recent technological breakthroughs and cost reductions in networking technology are driving rapid increases in feasible network bandwidths at densities of more than one terabit per optical fiber for both metro and long-haul networks. The exponential growth in bandwidth capacity over the past decade has surpassed the growth rate of CPU cycles predicted by Moore's Law and the growth rate of storage capacity. This is partially due to the use of new technology and parallelism in network architectures. A key driver in this growth is *dense wavelength division multiplexing* (DWDM), which allows each optical fiber to carry hundreds of lambdas. A lambda is a fully dedicated wavelength of light in an optical network with bandwidth speeds of 10-40 Gbps. These capabilities are being used to build high-speed, shared, routed internet networks, private dark fiber networks [65, 44], and based on new technologies for dynamic configuration, dynamic private networks.

These types of networks, with abundant, reconfigurable, end-to-end lambda paths, are known as *lambda networks*. Recent advances in network control plane [88, 80] enable dynamic provisioning of dynamic lambdas on demand, interconnecting Grid [42, 43] computing clusters via optical circuit-switched or packet-switched paths. They are capable of transferring data from 10 Gbps to 40 Gbps from end to end. For example, the OptIPuter project [76] interconnects national and international lambda networks through the StarLight [19] optical peering exchange

at Chicago, providing 10 Gbps connectivity between San Diego, Seattle, Chicago, Amsterdam and other partner sites around the world. Another example is TransLight [35], which is being implemented across research labs and universities in Europe, Japan and North America. Nowadays, National Lambda Rail [14] (Figure I.1) partners are providing at least 70 Gbps of electronically and optically switched circuits. Other examples of lambda networks include CANARIE [2], Netherlight [15], and NAREGI [13].

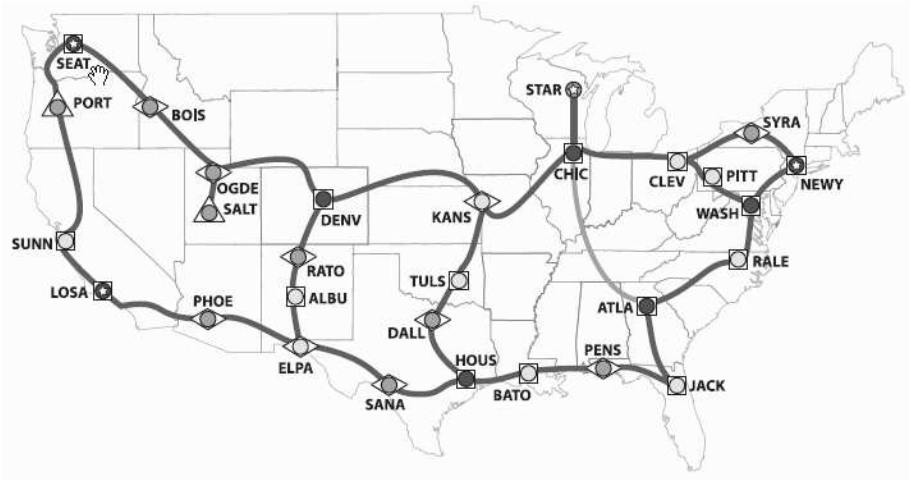


Figure I.1: National Lambda Rail Optical Network Topology

Unlike the traditional Internet, which offers commodity consumer services to hundreds of millions of end users with relatively low (modem or DSL) access speed, in a Grid environment powered by lambda networks, a limited set of high-end networked, middleware-enabled resources for computing, storage and visualization are tightly coupled by dynamically configurable light paths. Each end-node needs wide-area transport capabilities of multiple Gbps among a limited number of locations with long duration. Such capabilities can be easily supported by the dynamic allocation and scheduling of light paths. This type of Grid environment, in which the network bandwidth is one of many resources that must be allocated and scheduled, is known as a *lambda Grid*. In a lambda Grid, abundant network bandwidth moves data-intensive, e-science applications from a network-constrained world to a

network-rich world.

With this network bandwidth, the optical networks are even faster than the computing nodes to which they are attached. In such networks, the network bandwidth in the lambda core is high, matching or exceeding the speeds at which endpoints can source or sink network traffic. This environment poses new challenges in the efficient allocation and scheduling of network bandwidth resources.

I.2 Data-Centric E-science Applications with New Communication Patterns

E-science applications support operations on large scientific data sets, including data gathering from scientific instruments, data storage, backup, data-fetching to computing facilities, data processing and data visualization. Emerging e-science applications enable scientific collaboration over transoceanic distances with interactive fetching, sharing, processing and visualization of large-scale datasets from vast, distributed stores of data. Such applications have two noteworthy properties: a network-centric approach and multipoint communication patterns.

I.2.A From a Processor-Centric to a Network-Centric Approach

Traditional e-science and Grid applications deal with small-scale datasets at a single location. The traditional paradigm is to replicate and co-locate the computing resources with the storage systems. Many computationally intensive science applications are supported by parallel processing at a specific supercomputing center. After the processing is completed, highly distilled results are transmitted to remote visualization systems. Most computing centers are also data centers for this purpose. Under this old paradigm, high-speed, end-to-end network connections were a scarce commodity and therefore moving data from one location to another

was constrained by the limited network bandwidth. Consequently, different data handling phases could not be paralleled or interactively performed in real time. The bandwidth was limited by the end-node's access speed, shared network links with other public traffic, and/or the poor performance of data transport protocols. Therefore it was more efficient to co-locate the data with computing resources than to move it from distributed repositories on demand.

As the need for interactive e-science collaboration in real time increases, emerging data-intensive science applications must support the real-time acquisition of extremely high volumes of data from specialized instruments at distant locations. In addition, they must gather, distribute, process and visualize those large volumes of data as a collaborative initiative among colleagues around the world. One example of such a project is the EarthScope initiative[4], which aims to develop a national cyberinfrastructure to support the study of the structures and evolution of the Earth's crusts in North America. Digital seismic arrays produce high-resolution images of continental crust. Each image has more than 50 GB and the project produces 40 TB per year. The data must be processed and visualized on demand at participating sites. Another example is the Biomedical Informatics Research Network (BIRN) [1], which enables large-scale, distributed collaborations for medical research in neuroscience. High-resolution microscopes produce massive data sets in which each individual 2D brain image can be as large as one gigabyte, and each 3D image can be more than 100 GB. Those data sets must be visualized in real time at multiple remote locations to support collaboration among the researchers at different institutions around the world.

These data-centric requirements demand a new paradigm based on distributed data integration - how to gather, manage, explore, process and visualize distributed data sets in a synchronous and collaborative way. Lambda networks are a key element of this new paradigm. Supported by the superior end-to-end bandwidth, data movement is no longer a bottleneck for data processing. Lambda network infrastructure and bandwidth resources become a first-class Grid resource for

distributed parallel processing and visualization. Because the bandwidth provided by lambda networks exceeds end node capacities, bridging high-performance computing systems with lambda networks allows e-science applications to tightly couple distributed computing, visualization and storage resources without co-location.

I.2.B Novel Communication Patterns

The communication patterns of high-performance e-science application have evolved from point-to-point data movement to point-to-multipoint, multipoint-to-multipoint and even multipoint-to-multipoint data transfers. In traditional high-performance e-science applications, data is first moved to a supercomputing cluster for data processing, and then the results are sent back. Normally only point-to-point transfers are involved.

Emerging e-science applications that support collaboration among researchers physically located in different parts of the world require multiple, concurrent transfers for real-time collaboration and data handling on demand. These applications typically need to visualize certain data sets from a single scientific instrument or data repository on tiled displays at different locations. The subsequent collaboration and discussion requires synchronous updates in the visualization terminals, meaning that all end sites must receive the same data with only minimal delay. Such transfers display a point-to-multipoint transfer pattern. Another example of a multipoint communication pattern occurs when multiple network link paths are used for data fetching, to allow visualization of different parts of the dataset from multiple data repositories. This is a multipoint-to-point transfer pattern, which sometimes requires balanced (fair) transfers among the connections to avoid the situation in which, for example, part of a 3D image has been loaded for a long time while other parts are still in transmission. One can expect e-science applications with multiple, parallel data-handling phases that would require multipoint-to-multipoint transfer patterns.

These new communication patterns pose the question of how to efficiently

and fairly share network resources among different communication sessions. When fast flows converge on an endpoint, the challenges lie in efficiently and fairly utilizing each end node's limited capacity.

I.3 Challenges in Fair Bandwidth Sharing

In a network-rich world enabled by lambda networks, geographically distributed scientific instruments, storage services, computing services and visualization devices can be tightly coupled to form a *distributed virtual computer* (DVC)[78], in which network bandwidth resources are scheduled and managed in the same way as other computing and storage resources are scheduled and managed. Several challenges arise in network bandwidth allocation and control due to the fundamental differences between lambda networks and the traditional Internet.

Firstly, in a lambda network, the data-handling bottleneck is not located inside the network. Rather, the network is limited by the data-handling speed of different devices at end nodes. The disk speed or data-processing speed may not be able to meet the data transmission rate that lambda networks deliver. Therefore, network contention and congestion occur at end nodes due to their limited capacity.

Secondly, in a lambda network, improving the utilization of abundant and dedicated resources becomes important for e-science bulk data transfers. In the traditional Internet, the question of how to quickly utilize and fill up the pipe was not considered, as it was more important to find ways to be nice to other traffic.

Thirdly, the communication patterns for e-science applications require considering new transmission patterns (point-to-multipoint, multipoint-to-point and multipoint-to-multipoint data movement) instead of multiple, independent point-to-point transfers. This requires proper management across multiple, active sessions and presents the challenge of how to balance the transfers and guarantee fairness among transmission pathways.

Each of the aforementioned challenges in bandwidth sharing indicate that

when network resources become a first-class resource, the efficient utilization and fair distribution of network bandwidth become increasingly important. Consequently, the solution will differ from traditional rate- and congestion-control algorithms in several aspects.

First, the service target is different. Traditional transport protocols (e.g., TCP and its variants) deal with rate and congestion control on commodity Internet, which is shared among millions of end-users and suffers from throughput problems. For example, TCP is highly susceptible to packet loss. In order to achieve gigabit-level performance with standard TCP on a 4,000 km path with 40 ms round-trip time (RTT), the loss rate must not exceed one per 8.5×10^8 packets. TCP provides a packet-level solution using packet loss as a control signal. In lambda networks with a limited number of users and potentially no packet loss inside the network, the sharing solution does not need to be based on TCP or a loss-based approach.

Second, TCP and its variants all manage single flows, providing rate and congestion control functionalities. They do not explicitly consider interactions among flows. In lambda networks, it becomes more feasible to consider the interaction and fairness across flows.

Thirdly, traditional approaches assume shared networks with internal congestion. As a result, their focus is on managing congestive packet loss within the network. In contrast, for lambda networks, the key challenge is to fully utilize and fairly share the capacities at end nodes.

There are three critical objectives for bandwidth allocation and sharing in lambda networks: efficiency, fairness, and stability and convergence.

- Efficiency. It is needed to maximize the utilization of source and sink capacity in a timely and sustainable manner.

- Fairness. The bandwidth should be allocated to sessions fairly. This is to ensure that all active connections receive the same quality of network bandwidth, in regardless of different RTTs and network paths they may have.

- Stability and Convergence. The system supported by the transport service needs to be stable. And it can converge to a unique, optimal, and fair rate allocation from any initial or transitional state.

I.4 Thesis and Approach

Our research studies the feasibility of delivering efficient and fair bandwidth sharing to data-intensive applications in lambda networks. To address the challenges described above, our approach is guided by the following three observations. Firstly, most of the network congestion and contention occurs at end nodes in lambda networks. Secondly, each end node can easily obtain the rate information for all associated communication sessions, and this may help to ensure that each session gets a fair share of the bandwidth. Thirdly, for bulk data transfer in fast and long-delay networks, session-based rate allocation may manage the sessions more efficiently than do traditional packet-based and loss-triggered congestion control.

Therefore, we propose to use an end-node-based rate allocation and adaptation approach to achieve efficient and fair bandwidth sharing among active sessions for data-intensive applications in lambda networks. The thesis of our study is best stated as follows:

High efficiency and max-min fair bandwidth sharing can be achieved in lambda networks by using an end-node-based rate allocation and adaptation approach. By asynchronously managing the capacity of each end-node across its associated sessions, the end-node-based approach can achieve efficient sharing of network bandwidth resources. Simultaneously, the system always converges to a unique max-min fair rate allocation among active sessions with various RTTs and demand rates with a relatively fast speed.

The following is a summary of our approach.

- *Explicitly allocate network bandwidth at each source and sink node.* Each end node uses local information including its capacity, the number of associated

sessions and each session's current rate to explicitly assign its expected rate for each session. Then the real session rate is bounded by the expected rates of its source and sink.

- *Achieve max-min fairness across sessions.* We use a rate-estimation scheme to ensure that sessions with lower rates are allocated more room to increase their rates. We try to ensure that sessions with different RTTs are treated fairly in long-term, establishing max-min fairness among all sessions.

- *Achieve smooth transition with rate adaptation.* When new sessions join and others leave, a smooth transition is key to ensure system stability. We propose to use rate-adaptation schemes to make the transition phase as smooth as possible.

The proposed approach is studied analytically. We attempt to answer the following research questions in this dissertation:

- How can we formally model lambda networks and the bandwidth-sharing problem within those networks?

- How can we calculate the max-min fair rate allocation, given the topology and session demands of a lambda network?

- Can the proposed approach converge to such max-min fair rate allocation regardless of session's initial or transitional states?

We also conduct simulation and prototype experiments to validate the results obtained from the analytical studies, as well as to explore aspects of protocol design and parameter choices, and to conduct comparisons with existing solutions.

I.5 Contributions

Our work is one of the first studies to consider the allocation and sharing of network bandwidth resources as being similar to that of other grid resources.

Based on a formulation of the end-node-based rate allocation problem for lambda networks, we propose an asynchronous, distributed rate allocation algorithm. The distributed algorithm uses local information only to allocate capacity at each source and sink, assigning an expected rate to each session. The actual rate of each session is determined by the minimum of the expected rate at its source and sink and its desired rate (which is not generally known to the network). Our distributed algorithm converges rapidly to a max-min fair rate allocation and also adapts quickly when desired rates change. This study includes a thorough analytical model, analytical proofs, a comprehensive simulation, and experimental studies. The specific contributions of this dissertation are the following:

- To create a mathematical model that captures the characteristics of lambda networks and the differences between lambda networks and the traditional Internet.

- To identify the key challenges of bandwidth allocation and sharing for lambda networks and to provide a formulation of the bandwidth-sharing problem in lambda networks.

- To provide a global rate-allocation algorithm that calculates the max-min rate allocation given the topology of lambda networks and the constraints of a set of finite desired session rates, providing that such max-min rate allocation is unique.

- To propose an end-node-based rate allocation, control and adaptation approach. A distributed rate-allocation algorithm is presented, which allocates source and sink capacity among active sessions at each end node. The real session sending rate is determined by these estimates.

- To conduct an analytical study that proves that the distributed algorithm converges to the unique max-min fair rate allocation calculated by the global algorithm, from any initial or transitional state.

- To provide an evaluation of the distributed rate-allocation algorithm through simulations, which show that the proposed approach achieves efficient and fair allocations in distributed network environments with various parameters for size, latency, and synchrony.

- To conduct a simulated comparison study of the distributed rate allocation algorithm with TCP, its variants, rate-based protocols, and router- or switch-assisted rate-allocation schemes.

- To design and implement a prototype of the distributed rate allocation scheme, which will be used for comparison studies on real lambda-network test beds.

I.6 Organization

This dissertation is organized as follows. The current chapter presents an overview of the the challenges presented by lambda networks and the purpose of our work. In Chapter II, we introduce the background information and related works. In Chapter III, we formulate the bandwidth-sharing problem for lambda networks and present our thesis statement. In Chapter IV, we describe our end-node-based approach. In Chapter V, we present a global algorithm for calculating the optimum rate allocation based on global information. In Chapter VI, we describe the end-node-based approach, and give a proof of the stability and convergence properties of the proposed algorithm. In Chapters VII and VIII, we describe simulation studies to illustrate the convergence properties of the distributed algorithm under asynchronous distributed settings. We also compare our approach with existing end-node-based approaches and two variants of TCP. In Chapters IX, we present a prototype implementation, design issues and studies to verify the convergence properties with this implementation. We conclude in Chapter X with a summary and our recommendations for future works.

II

Background and Related Work

In this chapter, we present background information regarding high performance data transport for data-intensive applications in lambda networks. We first introduce lambda networks and model the ways in which they work. We then describe typical data-intensive applications and their transport requirements. In Section II.3, we describe the OptIPuter's attempt to address the challenges posed by lambda networks and the demands of e-science applications. We then present the general challenges in achieving high-performance data transport and previous approaches that have been taken in Sections II.4 and II.5, respectively.

II.1 Lambda Networks: Architecture and Assumptions

Dense Wavelength Division Multiplexing is increasingly available as an efficient technique to exploit optical fiber bandwidth at terabit-per-second speeds. Each single fiber multiplexes large numbers of wavelengths, or *lambdas*. A *lambda* is a dedicated wavelength of light in an optical network, delivering between 10 and 40 Gbps of bandwidth. A lambda network is a network of end nodes connected by such lambdas from end to end. Lambda networks are distinguished from traditional, shared Internet protocol networks by their dynamic configuration and dramatically higher

performance. Lambda networks can be thought of as providing private networks for each application. Lambda networks are deployed by academic communities [19, 76], as well as major Internet companies, such as Google, Yahoo! and MSN [10].

Lambda networks are used to connect end-node resources in several ways. Figure II.1 [31] presents a representative physical architecture of a typical lambda network. The first form of connection is end-to-end dedicated lambdas, which directly connect the network interface of individual computing, storage or visualization devices with a private, dedicated connection running 1-10 Gbps or more. Network bandwidth is not shared with traffic from other end nodes, and generally each end node has an optical interface. The second type of connection is switch-to-switch lambdas that are shared through a border packet switch. One or multiple circuit-switched lambdas may terminate at a shared switch, allowing the connections to be efficiently shared by a set of endpoints. Further, the endpoints need no additional interfaces (optical or otherwise). The advent of cheap 10-Gbps packet switches and the pending availability of cheap 10-Gbps copper NIC interfaces make this approach attractive. High service quality can be achieved through VLANs or through simple over-provisioning. The third type of lamba-network connection, similar to the second type, is switch-to-switch lambdas that are shared through border routers. Only a router is used and can be controlled via *generalized multiprotocol label switching (GMPLS)* [24]. The challenge with this type of lambda is that router ports running at 10 Gbps are expensive. We expect that the second and third types of lambda networks will be widely deployed, reflecting the likely integration of optical circuit-switched and packet-switched networks.

Most lambda networks are operated as private networks with a limited number of end nodes and end users. The dynamically configured light path between two edge switches acts as a dynamically constructed bridge. Each cluster node may have two network interfaces, typically configured as the routed, shared internet and/or private optical circuit-switched networks. Lambdas are connected on demand, which allows applications to manage and control network resources

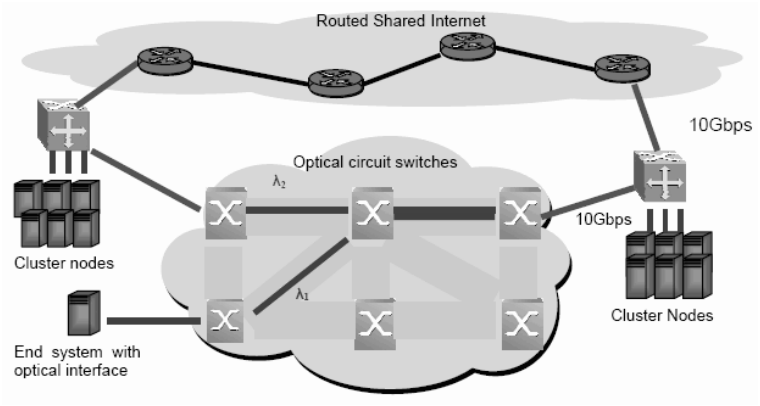


Figure II.1: Physical Architecture of the OptIPuter Networks

(e.g., to set up distributed computing resources and facilitate bulk data transfers). The light-path configuration can be static or dynamic. We anticipate lambda setup times of milliseconds or less [8].

Interfaces for controlling lambda networks are presented as control middleware or web services that accept as parameters the desired characteristics and capabilities of a lightpath for provisioning and monitoring. A typical procedure for establishing lambdas begins with an application communicating its network connectivity and resource needs, which may include end resources (storage and computing), along with a set of network paths and bandwidth requirements. The control middleware responds by matching the application requirements with the appropriate network paths, bandwidth resource (lambdas) and end-system resources. These lambdas are configured and reserved for dedicated use for the duration of the application. In this way, the bandwidth resource is fully controlled by the requesting application and is inaccessible to other applications. This mode of operation provides applications with network bandwidth resources as if it were a private resource in order to achieve high performance and guarantee high-quality service.

Lambda networks provide abundant bandwidth resources to the Grid environment, which is a set of distributed, networked, middleware-enabled instruments and computing, storage, and visualization resources. The resulting *lambda grids* are

collections of plentiful, geographically-distributed scientific instruments and computing, storage, and visualization resources, richly interconnected by lambda networks at tens of gigabits per second.

We have made the following observations about lambda networks, lambda grids, and their end systems.

- High-speed, dedicated links using one or more lambdas connecting a relatively small number of endpoints (e.g., 10^3 rather than 10^8). This is due to the fact that dedicated lambdas are used primarily by the high-performance computing community and are not used by most Internet users. This lets dedicated lambdas provide better QoS than the shared Internet does in terms of jitter and the rate of packet loss.

- Such lambda networks can exist with long delays between sites (e.g., 60ms RTT from the San Diego Super Computing Center [SDSC] to the National Center for Supercomputing Applications [NCSA]). Lambdas serve as the backbone network for remote collaboration between sites, and distributed grid applications are communicating through high-speed, long-distance lambda networks.

- End-to-end network bandwidth matches or exceeds the data-processing (computing or I/O processing) speeds of the attached systems. The abundant network resources create a relative scarcity of end-node resources, since lambda networks can be configured to accommodate any traffic rate requirement. As a result, the contention and congestion has been pushed from internal network links to the end nodes. When multiple connections flow from a single source or into a single sink, contention occurs at the source and congestion or possible packet loss occurs at the sink. Therefore, network congestion occurs primarily at the end nodes or at the "last switch" where one or multiple high-speed streams converge and terminate.

II.2 Data-Intensive Applications and Transport Requirements

Pioneering e-science applications supported by lambda networks are turning traditional grid applications into data-integration platforms with distributed data discovery, access and exploration technologies [1, 11, 12, 74, 5, 17, 7, 18, 3]. In this section, we first describe two e-science applications in detail. Then we analyze the transport requirements of those applications.

II.2.A Collaborative Data Visualization for Earth Sciences

The volume of data that geoscientists collect from their global-scale observational systems is dramatically increasing, producing data objects that can be visualized in multiple dimensions with very high resolution.

Researchers at the Scripps Visualization Center are using EarthScope [4] datasets to study geological activities at the San Andreas Fault in California. Each data file contains information about the 3D strain fields resulting from the deformation of the Earth's crust with spacing of 20 x 5 m. The raw data for an interferometric survey of California comprises 16 GB and can be as large as 50 GB. The size of the dataset increases not only with field size and sampling granularity, but also with timescales. Because raw data images can be produced on timescales ranging from days to decades, a visualization may involve datasets whose size is measured in terabytes.

The resolution of a standard display is sufficient to visualize only one data (scene) file. *Tiled displays*, which aggregate large numbers of smaller displays to achieve high resolution, is used to visualize such datasets over large timescales. Each display on a tiled display wall (usually with 55 (5x11) displays) is dedicated to a single data file as large as 50 Gbytes. The tiled display wall is driven by a set of cluster nodes and controlled by applications that switch among different datasets and timescales.

These displays create high demands on network infrastructure and data-transport services. The aggregate rate must be very high for data fetching from multiple, remote data centers. When the dataset size is too large to be cached locally, any command from the application to load a different set of scenes or timescales requires Gigabytes of data to be transferred for each tiled display. Furthremore, the transmission rate from different data centers must be balanced to minimize delay and ensure smooth transitions when displaying data visualizations or switching among the datasets. Additional demands are placed on the network when scientists collaborating at different locations share the same set of distributed data, because the visualization applications at different locations must provide the same quality of service regardless of their geographical location and/or asymmetric network delays.

II.2.B Visualization for Biomedical Informatics

Another e-science application is the Biomedical Informatics Research Network (BIRN) [1], which enables large-scale distributed collaboration for medical research in neuroscience. The goal is to create a distributed data repository that enables interactive visualization of 2D and 3D brain-mapping databases. One of the foci of the research is to conduct brain-mapping of human neurodegenerative diseases and build associated models based on mice brains. High-resolution 2D and 3D images are generated by high-energy electron microscopy, which produces multi-scale, multidimensional images ranging from subcellular structures to entire organs. Image magnification ranges from 20X to 5000X. In the past, the resolution of a single volumetric dataset was 2048x2048x512. Now, with the use of new energy-filtered microscopes, the resolution for single datasets can exceed 4Kx4Kx2K, and in the near future (within five years), it will exceed 12Kx12Kx2K. This means that the raw data in a 2D brain image can be at the Gigabyte level, and 3D images can be as large as 100 Gbytes.

Remote visualization and control applications allow scientists at different locations to remotely control electron microscopes and visualize the data, conduct-

ing collaborative microscopy experiments by correlating multi-scale, multi-modal datasets and the outputs of remote instruments. In this environment, the data is generated, transferred and visualized in real time and at the scales specified by interactive zoom or pan operations. This requires very high-speed network links between the data collection equipment, data repositories, and visualization stations. The average transmission speed needs to be high enough to support smooth zoom and pan operations.

Network bandwidth resources are a key element for constructing environments that allow researchers to disseminate, process, store and share data, as well as to visualize it concurrently at different sites.. For example, to simultaneously visualize and explore eight 3D images on a 5x11 tiled display wall, 64 Gbps of network bandwidth must be aggregated from different network paths to ensure smooth data display with zoom or pan operations at multiple locations. This kind of collaboration also requires equally high network quality at each visualization sites, regardles of the sites' geographical distances from the data sources. To minimize the delay at each visualization location, the data-fetching rate must be comparable at each site.

II.2.C Data Transport Requirements

The two types of e-science applications described above demand a high degree of network transport support in multiple dimensions. Specifically, they have the following data-transport requirements in common.

- High throughput and efficient utilization of network bandwidth to minimize the fetching time for bulk data.
- Fair sharing of network bandwidth resource to provide the same transfer rate from each data source and the same aggregated throughput to each data sink.

II.3 The OptIPuter Project: A System Middleware Approach

Much research has been conducted to address the challenges posed by lambda networks and the demands of emerging e-science applications. The work described in this dissertation is one part of the OptIPuter [76] project's research efforts. The OptIPuter project derives its name from its use of optical networking, Internet Protocol (IP), computer storage, and processing and visualization technologies [36]. It tightly couples computational resources over parallel optical networks using IP to form a distributed virtual meta computer [78]. For this type of distributed virtual computer, geographically distributed clusters serve as individual processors and are directly connected by optical links (lambdas). The massive storage systems are large, distributed repositories of scientific data and data feeds come from high-resolution scientific instruments. The main objective of the OptIPuter project is to develop software and middleware abstractions to provide data-delivery capabilities for the grid applications powered by lambda networks. The two e-science applications mentioned earlier are two such applications.

As OptIPuter networks scale up to have multiple 10-Gbps lambdas for end-to-end connections, the endpoints, too, must scale up to match the network's bandwidth. The OptIPuter's dedicated network infrastructure has significant advantages over the shared Internet, including high bandwidth, controlled performance (no jitter), a lower cost per unit of bandwidth, and higher security. The key question is how to efficiently discover, share, schedule, coordinate, and manage different types of distributed resources, including network topology, bandwidth, and resources for storage, computing, and visualization. Specifically, there are three major challenges:

(1) On-demand discovery, selection and configuration of supporting end resources and network resources;

(2) Construction of applications in heterogeneous, distributed environments; and

(3) Efficient sharing of network bandwidth resources to achieve high performance.

Multi-layered OptIPuter middleware technologies, including simple resource abstractions [78], dynamic network provisioning [49], robust storage [85], and novel data transport services [82, 83, 84, 79, 81] have been proposed to address aforementioned challenges. The work in this thesis addresses the third challenge: efficient sharing of network bandwidth resources.

II.4 Challenges in Delivering High-Performance Data Transport at High Speeds

Achieving high-performance bulk data transfer has been a long-standing research challenge for communication on high-bandwidth, long-delay links [37], as has controlling congestion at the single-flow level. In this section, we first describe the challenges presented by high bandwidth-delay links for data transport. Then we present the performance issues related to the standard Internet transport protocol (TCP). Finally, we describe general requirements for high-performance data transport. The improvements to TCP and other transport protocol approaches will be reviewed in the following section.

II.4.A Challenges in High Speed Data Transport

We have identified the following considerations as factors that make rate and congestion control more difficult in high bandwidth-delay product networks than in low bandwidth-delay product networks.

Huge Amount of Data in Transit

When data is transferred at full speed (link bandwidth), the total amount of data in transit is the number of bytes that has been dispatched for which no

acknowledgement has been received from the receiver, known as

$$B \cdot RTT, \tag{II.1}$$

where B is the link speed and RTT is the round-trip time between the sender and receiver. This is commonly known as the bandwidth-delay product. For example, on a San Diego-Chicago 10-Gbps link with an RTT of 60 ms, the bandwidth-delay product is 75 MB.

To maintain records of all in-transit packets and thereby make possible the subsequent retransmission of lost packets, the sender needs to maintain a buffer that is no smaller than the bandwidth-delay product of each active connection. The problems presented by a high bandwidth-delay product is the size of buffer needed for each connection, as well as the significant buffer that is needed for multiple concurrent connections. Most of the current operating systems (e.g., Linux, Windows) are configured by default to apply only small buffers. A typical value is 64 KB, but this is not sufficient for the high-speed, long-delay links. And without careful control of the buffer sizes and sending rates, the in-transit data can easily overflow the transmission queue at the routers, switches and/or end nodes, causing severe packet losses.

Low Link Error Rates are Significant at High Speed

The typical optical link-bit error ratio is between 10^{-9} and 10^{-11} including errors in optical fibers and the network equipment [64]. Although the error rate is low, it can prevent network flows from achieving high performance on high bandwidth-delay links. Consider the following example. On a 10 Gbps and 60 ms RTT link, the frequency with which errors occur can be expressed as:

$$10^{-9} \cdot 10\ Gbps = 1\ err/0.1\ s = 1\ err/1.67\ RTT,$$

$$10^{-11} \cdot 10\ Gbps = 1\ err/10\ s = 1\ err/167\ RTT.$$

In comparison, on a local link with 100 Mbps and 1 ms RTT, at the same link error rate, the results are:

$$10^{-9} \cdot 100\ Mbps = 1\ err/10\ s = 1\ err/10,000\ RTT,$$

$$10^{-11} \cdot 100\ Mbps = 1\ err/1000\ s = 1\ err/1,000,000\ RTT.$$

This example shows that high-throughput data transfer in high bandwidth-delay environments must tolerate much more frequent packet losses in terms of RTT than it does in low bandwidth-delay environments.

Transport schemes that reduce their rates upon packet loss experience a higher frequency of packet loss per RTT. Consequently, such schemes may not be able to achieve the same level of efficiency as they do in low bandwidth-delay environments.

High Bandwidth and Long Delays Yield Slow Responsiveness

One way to characterize the responsiveness of data-transport protocols is to measure the time needed to raise the sending rate from r_1 to r_2 ($r_2 > r_1$). In general, the time taken is an increasing function of r_2 - r_1. Therefore higher network bandwidth widens the range of r_2 - r_1. For control schemes that adjust their rate per RTT, a longer delay leads to longer control intervals for each step of the change in the rate change. Therefore, the total time required in such schemes to increase the sending rate from r_1 to r_2 is proportional to the product of $r_2 - r_1$ and RTT. Using the same numbers from the previous example, the total time needed to raise the rate from zero to full bandwidth becomes

$$\frac{10\ Gbps \cdot 60\ ms}{100\ Mbps \cdot 0.001\ s} = 6,000$$

times longer on the high bandwidth-delay links.

Slow responsiveness over longer RTTs also leads to unfair bandwidth distribution between flows with different RTTs. When there are large RTT differences among flows, flows with shorter RTTs may respond to changes in network status

much more rapidly than do flows with longer RTTs. In addition, as pointed out in the previous sub-section, packet loss per RTT is greater over long-delay links. Flows with longer RTTs may suffer more packet losses and therefore cut their rates more often than do flows with shorter RTTs. The net effect of these two factors is that flows with shorter RTTs may perform better than do long-haul flows.

High Speed Increases the Overhead Handling Per Packet and the End System Becomes the Bottleneck

When the transmission rate increases to one or more gigabits per second, the average time allowed for sending/handling each Ethernet packet decreases. For example, in order to send 1.5-KB packets at 10 Gbps, a packet needs to be sent every 2 us. As a reference point, when running at 100 Mbps, a packet is sent every 120 us. As we stated earlier, as the bandwidth increases, it is the end system that becomes the bottleneck - not the network bandwidth. On the one hand, the transmission rate may be limited by the I/O speed of the system. For example, a test at SuperComputing 2004 showed that the data transmission rate can reach only 7.2 Gbps on a 10 Gbps link, because it is limited by the speed of the PCI-X bus of the end system. On the other hand, the data-transport protocol is likely to compete with user-level applications for CPU cycles. Most of the issues mentioned above can be viewed as implementation challenges. Careful consideration of these issues at the protocol-design phase may help to eliminate some of the overhead handling and improve performance. We will discuss several possible optimization solutions in Chapter IX.

II.4.B TCP Performance Issues in High-Speed Networks

Delivering end-to-end high-performance communication in high bandwidth-delay product networks is a long-standing research challenge in wired, wireless and satellite networks [37, 89]. It is well documented that TCP, the widely used transport protocol for traditional internet, does not perform well in high-bandwidth,

Table II.1: TCP Reno Recovery Time with Different Rate

Rate	1 Mbps	10Mbps	100Mbps	1 Gbps	10 Gbps
Recovery Time	0.5 s	5 s	40 s	9 min	1 h 35 min

long-delay environments. The standard TCP exhibits several shortcomings when large bandwidth-delay product flows pass through a bottleneck. The performance of the TCP over high bandwidth-delay links is described below.

Slow increase in sending rate: In accordance with the AIMD control law, TCP increases its congestion window by only one segment per RTT during the congestion-avoidance stage. Consequently, TCP is slow to reach the full link bandwidth in high bandwidth-delay product environments. For example, on a link between San Diego and Chicago with 60 ms RTT and one Gbps bandwidth, a TCP flow with a packet size of 1.5 KB may need hours to climb to full bandwidth. Table II.1 lists the recovery time of TCP on a link with 60 ms RTT against different link bandwidths.

The sending rate of standard TCP increases exponentially during the slow-start phase, which is much faster than additive increases. However, this sometimes causes severe packet loss in the network, with the increment exceeding what the network could afford to transmit.

Unable to achieve high speed on lossy links: The average (sustained) throughput of TCP, r, is inversely proportional to the square root of the packet drop rate, p, and RTT, as described in [66]:

$$r \approx \frac{1.22s}{RTT\sqrt{p}}, \tag{II.2}$$

where s is the packet size. Therefore, in high bandwidth-delay product environments, the average packet loss ratio must be very low in order to maintain a high average TCP sending rate,. In our previous example, if we want to maintain an average transmission rate of 5 Gbps, the link packet loss ratio needs to be less than one packet per 3×10^9 packets. However, this is possible only in a clean network without competing and crossing background traffic.

Oscillation and instability caused by AIMD: As additive increase, multiplicative decrease (AIMD) interweaves steady rate increases with sharp rate decreases, the sending rate of TCP oscillates all the time. This is not a severe problem when the link speed is low, however it causes heavy oscillation in the size of the router queues and other peer TCP flows in high bandwidth-delay product networks. The analysis in [63] and [57] shows that as the network capacity and RTTs increase, commonly used router queue management schemes, such as Random Early Discard(RED) [41], Random Early Marking(REM) [23], and Adaptive Virtual Queue(AVQ) [57] all become prone to instability and oscillations

Low average link utilization: Due to the AIMD control law, the average throughput of single TCP flow can achieve no more than 75% of the link utilization when there is some packet loss on the link. This is because whenever the sending rate reaches full bandwidth, packet loss occurs, and the sending rate/window is halved, followed by the linear increase of TCP congestion window.

Unfair to flows with different RTTs: From Eq. II.2 we see that TCP's throughput is inversely proportional to the connection's RTT. Therefore, when flows with various round trip time are sharing the same set of network resources, they are not able to share the bandwidth equally. As a result, The TCP flows with shorter RTTs get higher throughput.

II.4.C General Requirements for High-Performance Data Transport

Having examined the performance of standard TCP in high bandwidth-delay product networks, we can now summarize a set of performance metrics and requirements for high-performance data transport in high bandwidth-delay networks.

- Quick ramp-up to utilize spare network bandwidth

  Because high-speed networks in the future will likely contain longer RTT links, it is important that the data-transport protocol be able to fill up network pipe

quickly (e.g., within several RTTs). This is especially critical for short flows that last for only a short period of time.

- High sustained throughput while tolerating link loss

  An ideal transport protocol should effectively recover from packet loss and quickly regain high throughput. Unlike traditional TCP, it should also operate at high speeds when the link loss is relatively high.

- Fair to flows with different RTTs

  As we have shown in Eq.II.2, the throughput of TCP is inversely proportional to its RTT. Therefore, flows with different RTTs are not treated in a fair manner. An ideal rate and congestion control scheme should treat flows with various RTTs fairly by following certain fairness criteria, such as Max-min fairness [25], proportional fairness [59], etc. A commonly used matrix is Jain's fairness index [53]. The fairness index, f, of n flows is defined as:

$$f = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2}.$$

- Stability and convergence

  If all the n flows are long-lived, the rate allocation vector $(r_1, r_2,..., r_N)$ should converge to a fixed rate allocation $(r_1^*, r_2^*,..., r_N^*)$, regardless of their initial states, flow arrival sequence, or other temporal details. The solution should also quickly react to the joins and terminations of flows, as well as lead flows converge to a fair allocation quickly from any transition state.

II.5 Previous Approaches for High-Performance Data Transport

Different approaches have been suggested to achieve data-transport performance higher than that which the standard TCP can offer. These approaches

are grouped in five categories, each of which is described in detail in the following subsections:

- tuning TCP's performance without changing its control algorithm;
- improving TCP's loss-based control algorithm;
- developing new delay-based control algorithms;
- building rate-based protocols, and
- designing switch or router-assisted protocols.

We elaborate each of them in detail, discussing their strengths and differences from the problem addressed in this thesis.

II.5.A Tuning TCP

Considering the fact that the standard TCP has been widely deployed, and realizing how difficult it is to deploy a new transport protocol all over the network, one research direction has been identified to improve the performance of the standard TCP.

End-system optimization

This includes fixing the improper configuration and implementation of TCP at end-systems. In [30] the authors conduct a set of TCP optimizations, including zero buffer-copying throughout the TCP stack, and reducing per-packet handling overhead. To make the TCP buffer size optimal for both high and low bandwidth-delay networks, Dynamic Right-Sizing [38] is developed to provide automatic tuning up of the TCP window size. This ensures support to high bandwidth-delay links. It also keeps windows small for low-bandwidth and low-latency connections so that they do not consume unnecessary amounts of the system memory.

Parallel TCP

Instead of tuning TCP for each connection, the performance can be improved by utilizing multiple concurrent TCP connections. When m parallel TCP connections are utilized, the linear increase phase of TCP gets fastened by m times. Thus when single TCP stream is not able to fill the pipe, adding multiple parallel TCP flows significantly improves the aggregate performance. Examples of such work include PSockets [75] and GridFTP [21]. There are however two issues associated with this approach. First, allowing multiple TCP flows between the same source and destination is unfair to other competing flows sharing the same network infrastructure [47, 22]. Second, the number of concurrent TCP connections needs to be carefully chosen, as this may incur extra CPU handling overhead for the system, and the performance may start decreasing when the number of connections keeps increasing.

TCP Proxies

As the performance of TCP is inversely proportional to RTT, another way to improve its performance involves the deployment of TCP proxies [27, 77] on long-delay links. TCP proxies route packets from the source to destination via two separate TCP connections. By doing that a long connection is split into two short connections. As the TCP proxy reduces the TCP round trip time by half for each segment, this approach could significantly improve TCP's performance over long-distance connections.

II.5.B Improving the TCP Control Law

As we stated earlier, the standard TCP (TCP Reno) uses the AIMD control law to control the rate and congestion. Given congestion window size w, the AIMD control law can be described as:

$$ACK : w \leftarrow w + 1/w; \tag{II.3}$$

$$Loss : w \leftarrow w - 0.5w, \tag{II.4}$$

where w is the congestion window size. The relation between the loss ratio q and window size w is

$$q = \sqrt{\frac{3}{2w}}.$$

As we have pointed out, a major problem with the AIMD control law is that the additive increase is too slow and halving windows by half upon packet loss is too aggressive in high bandwidth-delay networks. A set of improvements over the standard AIMD control law has been proposed, including HSTCP [39], Scalable TCP [60], BIC-TCP [86], etc.. They use the same window-based and loss-based framework, but end up with different window increase/decrease control schemes.

High-speed TCP

High-speed TCP(HSTCP) [39] is designed to behave in the same way as the standard TCP in low bandwidth-delay environments, while achieving higher throughput than the standard TCP without requiring unrealistically low loss rates in high bandwidth-delay product networks. HSTCP achieves higher performance by increasing the window size faster than TCP Reno, and reducing the sending rate less aggressively when the packet loss occurs. HSTCP makes the increment to be a function of the current window size. The same applies to the decrement. The control law of HSTCP is expressed as:

$$ACK : w \leftarrow w + a(w)/w; \tag{II.5}$$

$$Loss : w \leftarrow w - b(w)/w, \tag{II.6}$$

where

$$a(w) = 1 \ if \ w \leq W,$$
$$a(w) = \frac{w^2 \cdot 2 \cdot b(w) \cdot p(w)}{2.0 - b(w)}, \ if \ w > W,$$

and

$$b(w) = 0.5 \ if \ w \leq W,$$

$$b(w) = (B - 0.5)\frac{log(w) - log(W)}{log(W_1) - log(W)} + 0.5,\ if\ w > W,$$

where W_1 is the corresponding congestion window size when the sending rate of TCP equals to the bottleneck capacity, and W is the threshold for switching between the normal TCP and aggressive behaviors. The values of parameters $a(w)$ and $b(w)$ are chosen to make the algorithm to yield the equilibrium. Under the equilibrium, the relationship between the end-to-end loss probability q and congestion window size w is defined in [40] as $q = \frac{0.0789}{w^{1.1976}}$.

Scalable TCP

Scalable TCP (S-TCP) [60] makes smaller changes to TCP Reno than HSTCP. It increases the TCP congestion window by a fixed increment whenever an acknowledgement is received (TCP Reno only increases the window size by $1/w$ segment). This is more aggressive than TCP Reno when the window size is large. When the packet loss happens, it only drops the congestion window by 1/8, which is much less than what is set in TCP Reno (1/2). The control function of S-TCP is:

$$ACK : w \leftarrow w + a; \tag{II.7}$$

$$Loss : w \leftarrow w - b \cdot w, \tag{II.8}$$

where a and b are constants, and $0 < a, b < 1$. The recommended values [60] are $a = 0.01$, and $b = 0.125$. The relationship between q and w is $q = \frac{a}{b \cdot w}(1 - \frac{b}{2})$.

BIC-TCP

In [86] the authors describe *BIC-TCP*, a TCP substitute, which achieves higher throughput during both slow-start and congestion avoidance stages. It also corrects the RTT unfairness problem of Scalable TCP and HSTCP. There are two congestion window thresholds set in BIC-TCP, low_win and $high\_win$. When the current congestion window size, $cwnd$, is smaller than low_win, BIC-TCP behaves just like TCP Reno. When $cwnd$ is between low_win and $high\_win$, BIC-TCP does

faster recovery of the window size:

$$cwnd = cwnd \cdot (1 - \beta), \tag{II.9}$$

where β is a multiplicative window decrease factor. It also conducts binary search between low_win and $high\_win$ to increase its congestion window when there is no loss. The parameter $high\_win$ is also updated based on the current value of $cwnd$. When $cwnd$ is larger than $high\_win$, it additively increases the window size with a larger increment than TCP Reno. The relationship between q and w is $q = \frac{1}{(log_2(\frac{w}{0.08})+1.75)\cdot w}$.

Discussion

The approaches of improving TCP's performance or changing TCP's AIMD control law does not target solving the problem we study in this thesis for lambda networks. The main difference is the following. Firstly, they still target general Internet, which is shared by millions of concurrent users. Therefore, loss-based approach is feasible to ensure that each flow along is not too aggressive. In comparison, with the assumption that lambda networks only serve a limited number of end nodes and concurrent flows, the transport scheme can be more aggressive to achieve better network bandwidth utilization. Secondly, achieving fairness for flows with different RTTs is not a major design goal for TCP various but must be considered for supporting e-science applications in lambda networks.

II.5.C New Delay-Based Protocols

Using the end-to-end queuing delay as an explicit congestion signal has been proposed as a way to provide better and earlier congestion-detection and congestion-avoidance. This is based on the observation that the queue size of the bottleneck link increases before packet loss occurs, such that the observed end-to-end delay rises as well. Another difference between the loss-based and delay-based approaches is that, because the packet drop is no longer a control signal, delay-based

approaches can operate with zero-packet loss in a steady state. The exact average throughput and convergence point depend on parameter settings. Early work in this area includes CARD [52], which suggests using delay as congestion signal and using AIMD to oscillate between empty queue and non-empty queue. The details of two recent work, TCP Vegas [28] and FAST TCP [54] are provided below.

TCP Vegas [28] was introduced as an alternative to the standard TCP. All the changes that TCP Vegas makes to the standard TCP are at the sender side. In contrast to the standard TCP, which uses the packet loss as the congestion signal, the TCP Vegas source anticipates the occurrence of congestion by monitoring the difference between the rate it is expecting to see and the rate it is actually realizing. TCP Vegas's strategy is to adjust the sending rate in an attempt to keep a small number of packets buffered in the routers along the path. For every RTT, the window is adjusted by comparing the current rate with the maximum observed rate, as:

$$if\ w/RTT_{min} - w/RTT < \alpha\ then\ w++, \tag{II.10}$$

and

$$if\ w/RTT_{min} - w/RTT > \beta\ then\ w--, \tag{II.11}$$

where RTT_{min} is the observed minimum RTT, and α and β are thresholds. In the slow start phase, to reduce overshoot, it increases congestion window only every other RTT. Upon a packet loss, TCP Vegas halves the window size, the same as what TCP Reno does.

FAST TCP [54] reacts both to the queuing delay and packet loss. It periodically measures the RTT, comparing with the base-RTT, and then adjusts the window size accordingly. The congestion window size is determined by

$$RTT : w \leftarrow w \cdot \frac{baseRTT}{RTT} + \alpha, \tag{II.12}$$

where $baseRTT$ is the minimum RTT observed by the sender, and the parameter α determines the fairness and convergence rate. The design of FAST-TCP is based on a utility-based primary-dual model, and the control signal is, but not limited to,

the round-trip delay. It has the same equilibrium properties as TCP Vegas, and it achieves weighted proportional fairness [59].

The parameters of Fast TCP need to be carefully chosen for different network topologies. Delay-based schemes also require precise measurements of end-to-end delays, which is difficult to implement in lambda networks, where end-system packet-processing speed becomes the bottleneck.

II.5.D User-Level Rate-Based Solutions

A general problem of the window-based control scheme in high bandwidth-delay product environments is that all the data within the sliding window is sent in bursts. This may introduce severe packet loss before the sender actually learns about the congestion on the link. In contrast, senders following rate-based schemes send data always at no higher than the specified rates. This is done by introducing a delay d between the sending time of two adjacent packets. The delay d can be expressed as

$$d = \frac{s}{r}, \tag{II.13}$$

where s is the packet size, and r is the required rate. The transmission rate is usually explicitly specified or negotiated by the sender and receiver.

Network Block Transfer (NETBLT) [34] was developed for high throughput bulk data transmissions. It is optimized to operate efficiently over long-delay links. The unit of transmission is a buffer, the size of which can be set during the two-way-handshake stage. The sending rate is limited by a negotiated rate, and either the sender or receiver could also limit the rate by not providing a buffer (similar to the idea of the sliding window). NETBLT selectively retransmits the loss packets to achieve reliability.

RBUDP [48] is a UDP-based point-to-point data transfer protocol, intended for dedicated (e.g. dedicated wavelength) or QoS-enabled network environments. RBUDP is a fixed-rate data transport protocol with selective retransmission. The RBUDP sender starts by transmitting all data blocks over UDP at a fixed speed,

which is specified by the user. The receiver maintains a bitmap to keep track of received/lost data blocks. After the sender finishes sending all the data, the receiver sends the updated bitmap back to the sender through TCP (TCP is used for the purpose of reliable transmission). The sender then re-sends the lost data blocks according to the bitmap in the next round. The above procedure repeats until the receiver successfully receives all the data blocks.

To avoid network congestion, both NETBLT and RBUDP require the explicit knowledge of the achievable bandwidth. The protocol overhead of RBUDP includes the delay between each round of transmission due to the bitmap transmission, which becomes expensive when the number of rounds of the transmission increases (due to heavy network congestion).

SABUL (Simple available bandwidth utilization library) is designed for data-intensive applications in high bandwidth-delay product networks with user level implementation and control. The newest version of SABUL (UDT) [45] combines rate-based and window-based control mechanisms to deliver high throughput and low loss data transmission. The rate control algorithm used in UDT is AIMD, where the increase parameter is related to the current available bandwidth in the network and decrease factor is a fixed constant:

$$w = w \cdot \alpha + R \cdot (RTT + SYN) \cdot (1 - \alpha), \tag{II.14}$$

where R is the observed packet arrival rate, and SYN is a constant interval equal to 0.01 second. UDT also deploys rate adjustments based on delay monitoring, providing improved performance over common AIMD control laws. However this also makes UDT sensitive to network and end system conditions and difficult to reach fairness across RTTs for multipoint-to-point transfers.

II.5.E Router or Switch-Assisted Rate and Congestion Control

The routers within the network can also play an important role to the network congestion control via queue management or rate allocation. Typically the *DropTail* scheme is used, which is a queueing discipline that drops an arrival to a full buffer. Other than DropTail, different Active Queue Management (AQM) [50] schemes have been invented. For example, Random Early Detection (RED) [41] provides an alternative way to generate congestion signals to TCP sources. Instead of dropping packets only upon a full buffer, RED maintains an exponential weighted queue length and drops packets with the probability that increases with the average queue length. No packet will be dropped when the average queue length is below a threshold.

These AQM schemes provide each individual flow with more information about the network condition via packet drops. This information can be used to control congestion at each router. But it is not clear whether it can be applied directly to lambda networks, in which network contention occurs only at the edge. Moreover, the AQM schemes do not explicitly allocate router capacity to each individual flow, partially due to the large number of flows sharing the same bottleneck router. In comparison, the amount of concurrent traffic is very limited for lambda networks.

The Explicit Control Protocol (XCP) [58] attempts to conduct rate allocation at routers and decouples the congestion control and rate allocation. It requires the cooperation between the routers and end-systems. Each router explicitly assigns the sending rate of each flow and notifies the senders. There are two control phases within XCP, *efficiency control* and *fairness adjustment*. During the efficiency control phase, XCP uses *Multiplicative Increase and Multiplicative Decrease (MIMD)* to achieve higher performance than TCP. It determines the aggregate traffic changes at each router, Δ, by

$$\Delta = \alpha \cdot S - \beta \cdot Q, \tag{II.15}$$

where α and β are constants, and they are set based on the stability analysis. The parameter Q is the persistent queue size, and S is the spare bandwidth defined as the difference between the input traffic rate and the link capacity. During the fairness adjustment phase, XCP adjusts each flow by AIMD, which helps it converge to a fair rate allocation across flows. More specifically, if $\Delta > 0$, it allocates Δ equally to all the flows. If $\Delta < 0$, it divides Δ between flows proportionally to their current rates. It also shuffles a small amount of the bandwidth to each flow equally, which leads flows with different RTT to eventually get the same share of the bandwidth.

Compared with previous schemes for router/switch rate allocation, XCP provides different control feedbacks, and thereby is able to avoid the severe overflow case described previously. Unmodified XCP is not directly applicable in lambda networks, which often do not include routers. In Chapter VIII we will study whether the XCP router functions can be moved to the sources and sinks. In this case, each end-node would be aware of its capacity and that of the associated sessions, and would conduct the same rate allocation, adaptation and feedback defined for the XCP router module. This study will allow us to investigate whether an XCP-like scheme can provide a solution for the bandwidth-sharing problem in lambda networks. To distinguish this study from the original XCP work, we refer to the modified scheme as *endpointXCP*.

XCP shares some ideas with the rate allocation solutions in ATM networks in the context of allocating ATM switch capacity for ABR traffic [29, 51, 70, 55, 26]. Charny et al. in [29] proposes the *Consistent Marking* scheme, which is proved to make session rates converge to the max-min fair rate allocation. Hou et al. in [51] study the problem of generalized max-min rate allocation, which recognizes that each session has constraints on the minimum rate and peak rate. Such rate-allocation schemes allow each switch to periodically calculate an advertised rate for each active session based on the switch capacity, session current rates and minimum- and peak-rate constraints. Under such schemes, each switch first identifies and marks the constrained sessions, which are likely bottlenecked by other switches, and

then calculates the advertised rate as the remaining bandwidth over the number of unmarked sessions.

These schemes cannot directly meet the data transport requirements in lambda networks, because in these networks the switches are not bottlenecks and therefore are not modelled. One way to extend such schemes is to move the switch functions to end nodes (sources and sinks). However, this is not a good solution for the problem we are studying for the following two reasons.

First, previous works (e.g. [51]) assume explicit knowledge about the session's minimum- and peak-rate constraints. However, for the problem we are studying, the session's desired rate (peak rate) is unknown to the end nodes and may change over time. This lack of global knowledge makes any centralized or coordinated rate-allocation scheme an unlikely solution.

Second, in most of the schemes described above, each switch sends the same advertised rate to traffic sources. As a result, sessions with a lower desired rate are allowed to send data at a much higher rate the moment that their demand increases. This can severely overflow the sinks. To illustrate this problem, consider this example of a multipoint-to-point transfer. In a lambda network, there are four sources and one sink, each with capacity 1. There is one active session between each source-sink pair. At equilibrium, the session's desired rates, equilibrium rates and advertised rates that are calculated by applying the distributed algorithm in [51] are shown in table 2.2. We see that each session uses the same advertised rate of 0.7 from the sink. If at any later time the three sessions that have a desired rate of 0.1 increase their desired rate to 1, each of them will begin sending at a rate of 0.7. This will result in aggregate traffic of 2.8 at the sink, much higher than the sink's capacity. This is an especially severe problem on long-delay networks, because at least one round-trip is needed to provide the feedback that will result in a lower advertised rate. Therefore, when the session's desired rates are unknown, assigning the same rate-control signal to all associated sessions may not work well in lambda networks.

Table II.2: 4-to-1 transfer

Session	Session demand	Eq. rate	adv. rate
1	1	0.7	0.7
2	0.1	0.1	0.7
3	0.1	0.1	0.7
4	0.1	0.1	0.7

II.6 Summary

In this chapter, we have described the unique characteristics of lambda networks, presented high-performance data transport requirement for supporting collaborative scientific applications. We have also summarized challenges for achieving high-performance transport in high bandwidth-delay product environment and reviewed previous approaches. Each of these approaches has different design goals and target different network environments. None of them is able to completely address the transport challenges we described for lambda networks. In this diserta-tion, we will study these challenges and present a novel approach to address these challenges.

III

Thesis Statement

Rate allocation and control in high-bandwidth, long-delay networks has long been an area of research interest. Our research studies an alternative approach to session-based rate control, in which end nodes asynchronously and adaptively allocate their bandwidth resources over active sessions to achieve high efficiency and fairness. Through this study, we analytically and experimentally develop distributed algorithms for this purpose. In this chapter, we outline the research context, define the research problem, and present the thesis statement.

III.1 Context

Emerging large-scale scientific applications have a critical need for efficient data transport over high-speed networks. Sharing and transferring information and allowing collaboration over such large datasets requires efficient management of active sessions in lambda network and lambda grid environments (see Figure III.1) Our study focuses on how to facilitate efficient and fair sharing of bandwidth resources in lambda networks for data-centric scientific applications.

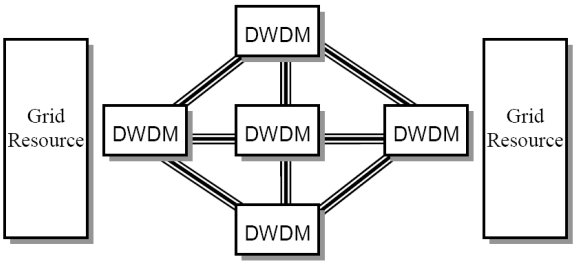


Figure III.1: Lambda Network and Grid Resources

III.1.A Targeted Applications

Emerging e-science, including geosciences, biomedical informatics, and nuclear physics, posits remote scientific collaboration within large-scale, distributed datasets. Such collaboration demands the ability to interactively and remotely share, process, and visualize large datasets. All such e-science applications share two characteristics: the need for efficient data transport and the presence of multipoint transfer patterns.

Firstly, all e-science applications have the need to transfer large scale datasets that are measured in gigabytes. With high-speed networks, the challenge of supporting data-intensive, collaborative science and engineering shifts from replicating and co-locating the computing resources with the storage systems to moving large-scale datasets to computational facilities on demand, without duplicating computation at data centers. Usually such transfers (e.g., FTP-like transfers and real-time streaming) require a relatively long period of time and differ from bursty traffic, such as that generated by web browsing.

Secondly, e-science collaboration that interactively explores multi-terabyte datasets requires the ability to fetch data from multiple, remote data repositories and visualize them concurrently at multiple locations. Consequently, the traffic in lambda networks behaves in a multipoint-to-multipoint pattern. Unlike traditional point-to-point traffic, multipoint-to-multipoint transfers must treat multiple sessions fairly, regardless of their geographical location and the propagation delay

differences between each sender/receiver pair. This is very different from current Internet transport protocols (e.g., TCP), which allow sessions with shorter round-trip delays to be delivered at much higher rates.

The traffic demands of these applications require a transport-level solution focused on long-term efficiency (high throughput) and fairness across all active sessions.

III.1.B Targeted Network Environment

Lambda networks are the focus of our research. As stated in Chapter II, such networks have sufficient DWDM-enabled bandwidth and the ability to provide dynamically network bandwidth (lambdas) on demand. Lambda networks differ from the traditional shared Internet in that there is essentially no packet loss due to congestion within the network, and any contention and congestion loss happens at end nodes (sources and sinks). Our lambda-network models are based on the following assumptions:

1. With plenty of network bandwidth provided by private, dynamically configured lambdas, there is no traffic congestion within the network. Because the capacity of lambda networks is higher than the desired rate of each session, there is no need to model network internals. The rate control and allocation problem is reduced to how to efficiently and fairly share the source and sink capacity among traffic sessions.

2. Each source and sink node has explicit knowledge regarding its capacity, network interface speed, local packet-processing capacity, and that of a shared link nearby. With this information, each node knows how much capacity it has to allocate across its sessions.

3. Each session has its own desired rate,or the peak rate that it can reach. The desired rate can be also interpreted as the maximum data-handling speed of

the applications using the network. The desired rate of each session is unknown to its source and sink and can vary over time.

Lambda networks switch data-centric applications from a network-constrained world into a network-rich world. The data-handling speed and end-node capacity are the new scarce resources. This change posits new approaches to solve the data-transport and bandwidth-sharing problems in lambda networks.

III.2 The Bandwidth Sharing Problem

The research problem we focus on is *how to* **efficiently** *and* **fairly** *share the capacity of each source and sink among active sessions.*

In this dissertation, we try to answer the following research questions concerned with this research topic.

- How can a fair bandwidth-sharing scenario be calculated in a lambda network?

  Given a set of steady and active sessions in lambda networks, we assume that all information regarding session demands, end-node capacity and network topology is known. Using this information, which fairness criteria should be used to determine whether a given rate-allocation is fair for all active sessions? Furthermore, how can such a fair rate allocation be calculated?

- Can a fair rate allocation be achieved through an asynchronous approach?

  Given that synchronous approaches may not be feasible due to the long delays experienced in lambda networks, is it possible to achieve a fair rate allocation through asynchronous-based approaches to provide efficient and fair utilization of network resources? Such an approach should assume no knowledge regarding network topology and session demands.

- Is the proposed approach reliable and can it always converge?

The proposed solution should work on any lambda-network topologies and converge to the same unique, fair rate allocation from any initial state. When sessions join or leave, or when desired session rates change, the proposed approach should be able to help the system converge to a new, fair rate allocation after a transition period.

- How does the proposed approach compare with other existing approaches?

  Comprehensive comparison studies must be conducted to compare the proposed approach with other existing session-based and router-assisted solutions.

III.3 Thesis Statement

Our research studies the feasibility of delivering efficient and fair bandwidth sharing to data-incentive applications in lambda networks by using an end-node based rate allocation and adaptation approach. The thesis statement is as follows:

High efficiency and max-min fair bandwidth sharing can be achieved in lambda networks by using an end-node-based approach to rate allocation and adaptation. By asynchronously managing the capacity of each end node across its associated sessions, the end-node-based approach can achieve efficient sharing of network bandwidth resources. The system always converges to a unique max-min fair rate allocation among active sessions with differing RTTs and relatively fast demand rates.

Subsidiary theses required to substantiate this result include the following.

- *Efficient utilization* of the source and sink capacities among active sessions in lambda networks can be achieved through distributed, asynchronous, end-node-based rate allocation;

- *Max-min fairness* among sessions with differing RTTs and demand rates can be achieved through end-node-based rate allocation, with session desired rates discovered through rate estimation techniques;

- *Smooth transition* can be achieved when some sessions join or leave, or when sessions' desired rates change, through rate adaptation;
- *Stability and convergence* can be achieved in fast networks with hundreds of end nodes.

To support the thesis, we first identify the max-min fair rate allocation, given the global topology and session information. We then investigate the possibility of an end-node-based approach by proposing an end-node-based rate allocation and adaptation scheme. Such a scheme uses only local information and operates asynchronously at each end node. We theoretically prove the soundness of the proposed scheme by showing that it always converges to the max-min fair rate allocation that is calculated by the global algorithm. We then conduct comprehensive simulation studies and prototype implementation studies to further prove the feasibility, efficiency, and convergence and fairness properties of the proposed scheme. We also compare the proposed approach with TCP variants and other router-based schemes under high-capacity lambda-network settings and differing round-trip times and traffic demands. Our results show that our end-node-based bandwidth sharing scheme better supports high-speed flows, yielding max-min fair allocation for all flows.

III.4 Solution Criteria and Metrics

III.4.A Feasibility

The rate allocation (bandwidth sharing) algorithm should efficiently utilize of the capacity of each source and sink while maintaining *feasibility*. Feasibility, simply put, requires that the aggregate rate of the sessions for each source and sink does not exceed its capacity, and each session's rate does not exceed its desired rate. Formally, *feasibility* can be defined as follows.

Definition 1. ***Feasibility.*** *A rate vector* $\mathbf{x}$ *is feasible, when for each node* $v \in \mathcal{V}$*,*

$$\sum_{k \in \mathcal{K}^v} x_k \leq C^v,$$

and for each session $k \in \mathcal{K}$ *we have that* $x_k \leq M_k$*.*

III.4.B Efficiency

The requirements on efficiency are as follows:

- *High link utilization.* An ideal distributed rate allocation and control scheme should achieve high resource utilization at each source and sink.
- *Avoidance of severe congestion.* It is of great importance that the rate control and allocation scheme does not incur severe congestion to end nodes during transitional states.
- *Quick reaction to flow dynamics.* The solution should quickly react to the joins and terminations of traffic sessions, as well as lead sessions converge to a state of equilibrium quickly from any transitional state.

III.4.C Fairness

The rate allocation and control algorithm should treat all active sessions fairly. We adopt *Max-min Fairness* [25] as the criteria, as it is widely used in wired and wireless networks [56, 68, 67]. Max-min fairness maximizes the rates of the sessions with lower rates, and shares the remaining bandwidth until the network is fully utilizes. Formally, max-min fairness is defined as follows.

Definition 2. ***Max-min Fairness.*** *A feasible rate vector* $\mathbf{x}$ *is max-min fair if it is not possible to increase the rate of a session* x_p *while maintaining feasibility, without reducing the rate of some session* x_q $(p \neq q)$ *with* $x_p \leq x_q$*. i.e. for any other feasible vector* y*,*

$$(\exists p \in \mathcal{K}) y_p > x_p \Longrightarrow (\exists q \in \mathcal{K}) y_q < x_q < x_p.$$

Max-min fairness criteria also implies that the ideal solution should treat sessions with various RTTs in a fair manner. For fairness criterions other than max-min fairness, we refer to single link fairness index [32] and most recently proportional fairness [59].

III.4.D Stability and Convergence

The distributed rate allocation algorithm should converge rapidly, reaching a unique rate allocation which is max-min fair. Consider under a discrete-time system, in time slot t, the rate vector $\mathbf{x}(t)$ is updated as

$$\mathbf{x}(t+1) = f(\mathbf{x}(t)),$$

where $f(\cdot)$ is the update function. The function $f(\cdot)$ may depend on session desired rates, the status of each source and sink, etc. For good network behavior, the rate allocation algorithm should be *stable* and *converge*.

Definition 3. ***Stability and Convergence.*** *An algorithm is stable if there is a unique equilibrium rate vector* $\mathbf{x}^* = (x_1^*, x_2^*, ..., x_K^*)$*, s.t.*

$$\mathbf{x}^* = f(\mathbf{x}^*).$$

An algorithm converges, if when all K *sessions are long-lived, the rate vector* $(x_1, x_2, ..., x_K)$ *over time approaches and achieves the unique equilibrium rate allocation* $\mathbf{x}^*$*, regardless of its initial state, session starting sequence, or other temporal details.*

The definition of convergence not only encompasses arbitrary initial state, but also changes in session desired rates during the convergence process.

IV

Approach

IV.1 Overview

We propose to use end-node-based, fair rate-control and rate-allocation schemes to achieve efficient and fair bandwidth-sharing in lambda networks.

In the traditional Internet, session-based approaches (e.g., TCP and its variants) are used to compete for traffic. As stated previously, it is difficult for such schemes to achieve bandwidth-sharing efficiency and fairness among flows with different RTTs in lambda-network environments. Based on the observation that in lambda networks the bottleneck of flow contention is not in the network, but rather pushed to the end-nodes, we propose to manage session rates at each end node to provide greater efficiency and guarantee long-term fairness.

In contrast to a session-based approach, an end-node-based approach requires that each end node have knowledge of all the associated active sessions. By monitoring sessions' status, an end-node-based scheme allocates its bandwidth to sessions at the end of each control interval as "expected rates". The real sending rate of each session is then bounded by the "expected rates" of its sender and receiver, as well as its real demand. We describe our approach in greater detail in the following subsections.

IV.2 End-Node-Based Approach for Rate Allocation in Lambda Networks

As described earlier, lambda networks differ from the traditional shared Internet. There is essentially no packet loss due to congestion within the network and any contention and congestion occurs at end nodes (sources and sinks). The challenge of achieving high-performance data transport becomes determining how to efficiently manage end-node capacities. We propose to use end-node-based rate allocation to efficiently allocate the network resources (source and sink capacities) among active sessions. Our motivation arises from the following observations:

- Because contention among sessions occurs at end nodes (rather than within the network), it is sensible to pursue an end-node based control scheme rather than one that is session-based.
- Each end node has explicit knowledge about its capacity and the number of sessions associated with it, which makes end-node based allocation feasible.
- Explicit rate allocation among sessions may provide full capacity utilization while ensuring fairness.

End-node-based rate allocation uses only local information to allocate capacity to active sessions at each source and sink. Such local information includes end-node capacity, the number of associated sessions, and each session's current rate.

Based on such local information, each source and sink explicitly assigns an allocated rate for each session and the actual session rate is determined by the allocated rate at its source and sink, as well as its desired rate (generally unknown to the network).

IV.3 Achieving Max-min Fairness Across Sessions

One of our primary goals is the fair sharing of network capacities among sessions that may have differing round-trip delays. When TCP traffic sessions share a link and have different RTTs, the delivered bandwidth is inversely proportional to RTT, resulting in unfair sharing. We adopt *max-min fairness* [25] as our criterion, which maximizes the rates of lower-rate sessions, and fully utilizes the capacity by sharing the remaining bandwidth.

Two variables, demand rates and RTTs, create difficulty in achieving max-min fairness among sessions. Each session has its own desired rate (the peak rate that it can reach). The desired rate can also be interpreted as the maximum data-handling speed of the applications using the network. The desired rate of each session is unknown to its source and sink, and varies over time. This was not a concern in previous studies of the ATM and IP networks, because session demands were assumed to be infinite. With lambda networks, this is no longer a valid assumption, as the demand of some sessions might be much lower than the end-node capacity. To "discover" each session's demand, we propose to use a rate-estimation scheme to facilitate end-node-based rate allocation. This scheme will predict each session's demand rate based on its identifying statistics, and will help the rate allocation saturate any new session demands.

Sessions with shorter RTTs respond to control signals faster than do those with longer RTTs. This leads to unfairness under session-based control schemes. The end-node-based control scheme has explicit knowledge about the rates of all associated sessions, which makes it possible to adapt rates toward (and eventually achieve) the max-min fair rate allocation.

IV.4 Using Rate Adaptation and Session-Specific Rate Feedback to Achieve a Smooth Transition

A session's status may change over time, which results in a change in the number of active sessions (and/or the session demand rate) and produces the following challenges:

- How do we avoid severe congestion when new sessions join?
- How do we let the remaining sessions fully utilize the network bandwidth resource when some sessions leave?

We propose to use *rate adaptation* and *session-specific rate feedback* to achieve a smooth transition when there is a change in session status. Rate adaptation ensures that predefined step sizes bind the incremental changes in session rates, which avoids severe transitional congestion at end nodes. Our scheme also provides session-specific target rates that guarantee the session rate is always under the control of its source and sink. In contrast, most of the rate-allocation schemes in ATM networks provide all sources with the same target rate, which might lead to severe congestion at the sink if some sessions increase their desired rates.

IV.5 Using Analytical Studies to Explore Convergence Characteristics

We will conduct analytical studies to investigate the efficiency, fairness, and convergence properties of our end-node-based rate allocation and control scheme. Based on a mathematical model of lambda networks and a formulation of their bandwidth sharing problem, analytical studies answer the following questions:

- How do we calculate the max-min fair rate allocation?

- Is this max-min fair rate allocation unique?
- Is the max-min fair rate allocation a stable operation point for an end-node-based rate allocation scheme?
- Regardless of their initial or transitional state, will the decentralized end-node based rate allocations converge?

To answer these questions, we will use a global rate-allocation algorithm that calculates the optimum rate allocation given a lambda network's topology and a set of finite, desired session rates. We will then study the stability and convergence properties of the distributed algorithm rate allocation. In particular, we will show that (a) the distributed algorithm achieves the max-min fair allocation $\mathbf{x}^*$ as specified in the global algorithm as an equilibrium point and (b) the distributed algorithm causes the session rates to converge, regardless of their initial states.

IV.6 Evaluation through Simulations and Prototype Implementation

We will use simulation and prototype implementation to explore aspects of protocol design, parameter choice, and comparison studies. We will use discrete event simulation to construct the following network configurations:

- *Heterogeneity:* We will generate random lambda network topologies having as many as 1024 end nodes.
- *Bandwidth and latency:* Using network capacities of up to one gigabit and round-trip delays of up to hundreds of milliseconds, we will explore a wide range of network settings.
- *Communication patterns:* We will use point-to-point, multipoint-to-point, and multipoint-to-multipoint communication patterns to configure active sessions.

We will use existing testbeds (e.g., the TeraGrid [20] and the OptIPuter) to reflect real-world network configurations. For each configuration, we will evaluate the performance of the distributed rate-allocation scheme. More specifically, we will:

- Study how different parameter values affect performance,
- Verify convergence and stability properties with various initial states and parameter configurations, and
- Compare our approach and other rate-based approaches (e.g., XCP), and TCP variants.

IV.7 Summary

Our end-node-based approach allows each end node to allocate its bandwidth capacity to its associated sessions asynchronously. We will discuss the design and implementation of our approach in detail and demonstrate that it meets the requirements of efficient and fair data transport in lambda networks. The following key evidence will support our thesis:

- a mathematical model of lambda networks and a formulation of the bandwidth-sharing problem in lambda networks,
- a global rate-allocation algorithm that calculates the max-min rate allocation given the constraints of a set of finite desired session rates,
- a distributed rate-allocation algorithm that allocates source and sink capacity among active sessions,
- an analytical study that proves that the distributed algorithm converges to the unique max-min fair rate allocation calculated by the global algorithm,
- an evaluation of the distributed rate-allocation algorithm, showing that it achieves efficient and fair allocations in both synchronous and asynchronous distributed network environments (with varying size, latency, and synchrony),

- the design and implementation of the distributed rate-allocation scheme, which will be used for comparison studies on real lambda-network test beds, as well as to demonstrate the benefits observed with live application, and
- a comparison study of the distributed rate-allocation algorithm with TCP, its variants, rate-based protocols, and router- or switch-assisted rate-allocation schemes.

V

Global Algorithm for Bandwidth Sharing

This chapter presents an analytical study of the global max-min calculation. We extend the basic idea of max-min global computation found in [25, 51] to account for the limitations on session demand.

V.1 Notations

The following notations are used throughout this dissertation.

Consider a lambda network $\mathcal{G}$ with a finite set $\mathcal{V}$ of nodes and a set $\mathcal{K}$ of K active sessions. Each *active session* is an elastic traffic session (e.g. FTP transfer) between a source and sink nodes. An active session is also called a *flow*. Let $\mathcal{V}^S$ and $\mathcal{V}^R$ denote a set of source and sink nodes, respectively. Note that multiple sessions may start or terminate at the same source or sink node.

Let each session $k \in \mathcal{K}$ have a round-trip time rtt_k and be associated with a desired (peak) rate M_k. Let $x_k(t)$ be the instantaneous rate of session k at time t. Let $\mathbf{x}(t) = (x_1(t), x_2(t), ..., x_K(t))$ be the rate vector of all active sessions at time t.

Let each source and sink $v \in \mathcal{V}$ have a capacity C^v. It is either the sending or receiving capacity of node v, when $v \in \mathcal{V}^S$ or $v \in \mathcal{V}^R$. Let $\mathcal{K}^v$ denote the set of

sessions in which node v participates. Note that, in our model, each node v is either a sink or a source node, but not both.

Therefore $\mathcal{G}(\mathcal{V}, \mathcal{C}, \mathcal{K}, \mathcal{M})$ characterizes an instance of the bandwidth sharing problem in lambda networks.

V.2 The Global Algorithm

To motivate the distributed max-min fair rate allocation algorithm, we briefly review how to calculate max-min fair rates with global knowledge [25, 51, 67]. At each step, the algorithm assigns equal rate shares to sessions at the current bottleneck node. The bottleneck node is the one with the minimum average rate per session after its remaining capacity is assigned to the remaining sessions (also known as undetermined sessions). We now formally define the bottleneck node.

Definition 4. ***Bottleneck Node.*** *Given a partially determined rate vector* $\mathbf{x}$*, node* $v \in V$ *is called the bottleneck node, if we have undetermined sessions* $k \in \{j | x_j = 0, j \in \mathcal{K}\}$*, and remaining available capacity at node* v *divided by the number of undetermined sessions,* x_f *(also known as the bottleneck rate), satisfies*

$$x_f = \frac{C^v - \sum_{i \in \mathcal{K}^v} x_i}{|\{j | x_j = 0, j \in \mathcal{K}^v\}|} = \min_{v' \in V} \frac{C^{v'} - \sum_{i \in \mathcal{K}^{v'}} x_i}{|\{j | x_j = 0, j \in \mathcal{K}^{v'}\}|}, \tag{V.1}$$

where node $v' \in V$ *such that* $|\{i | x_i = 0, i \in \mathcal{K}^{v'}\}| > 0$*.*

The complete global algorithm is described in Algorithm 1. It extends the classic max-min algorithm [25] by considering the session's desired rate.

Algorithm 1. ***Global Algorithm***

Notations:

$\mathbf{x}^*$*: The resulting rate vector, and* $\mathbf{x}^* = (x_1^*, ..., x_K^*)$*.*

$b(v)$*: The function which computes the average rate of the undetermined sessions at node* $v \in V$*, denoted by*

$$b(v) = \frac{C^v - \sum_{i \in \mathcal{K}^v} x_i^*}{|\{j | x_j^* = 0, j \in \mathcal{K}^v\}|}. \tag{V.2}$$

If there are no undetermined sessions at node $v \in V$, set $b(v) = +\infty$.

The Algorithm:

1. set $\mathbf{x}^* = \mathbf{0}$*;*

2. ***repeat***

3 ***if*** $\min_{v \in \mathcal{V}} b(v) < \min_{k \in \mathcal{K}, x_k^* = 0} M_k$

4 $v =$ *arg* $\min_{v' \in \mathcal{V}} b(v')$*;*

5 $\forall\ k \in \mathcal{K}^v$*, if* $x_k^* = 0$ *then let* $x_k^* = b(v)$;

6 ***else***

7 $k =$ *arg* $\min_{k' \in \mathcal{K}, x_{k'}^* = 0} M_{k'}$*;*

8 $x_k^* = M_k$*;*

9 ***endif***

10 ***until*** $|\{j | x_j^* = 0, j \in \mathcal{K}\}| = 0$.

Note that in Algorithm 1, there may be multiple bottleneck nodes with the same remaining capacity for any given iteration. In that case, we pick an arbitrary node.

V.3 Properties

We show in Proposition 1 that the rate vector $\mathbf{x}^*$ calculated by the global algorithm is the unique max-min fair rate allocation.

Proposition 1. *1. The global algorithm terminates within K iterations.*

2. The rate allocation $\mathbf{x}^*$ computed by the global algorithm is max-min fair and unique.

Proof. As in the global algorithm (Algorithm 1) at least one undetermined session is assigned with a rate in each iteration, the total number of iterations required for Algorithm 1 is then no more than K iterations, where K is the total number of active sessions. This proves the correctness of the first statement.

The Proposition 3 in [67] shows that if a max-min fair vector exists, it is the unique leximin maximal vector. As it is straight forward that the calculated

rate vector $\mathbf{x}^*$ is feasible, we are to prove by contradiction that if there is another feasible rate vector $\mathbf{y}$, and $\mathbf{y} >^L \mathbf{x}^*$, then $\mathbf{y}$ is not feasible.

Suppose there exists $m \in [1, .., K]$, such that $y_{(m)} > x^*_{(m)}$ and $\tau_{m-1}(\mathbf{x}^*) = \tau_{m-1}(\mathbf{y})$.

We firstly show by induction on $p = 1..m-1$ that

$$x^*_{\phi(x^*_{(p)})} = y_{\phi(x^*_{(p)})}.$$

(1) When $p = 1$, let $i = x^*_{(1)}1$. As $y_{(1)} = x^*_{(1)}$, for all $k = 1..K$, we have

$$x_i = x^*_{(1)} \leq y_k.$$

Therefore either $x^*_i = y_i$ or $x^*_i < y_i$.

We consider the case when $x^*_i < y_i$. From global algorithm we know that $x^*_i = \min\{M_i, b(v)\}$, where v is the corresponding node that has the smallest $b(v)$ in the ith iteration.

If $x^*_i = M_i$ then $y_i > M_i$, which violates the feasibility. Now we consider $x^*_i = b(v)$. The global algorithm implies that for any j, $0 < j < i$, $x^*_i \geq x^*_j$. Then we have

$$x^*_i = b(v) \geq C^v/|\mathcal{K}^v|.$$

And as for any j $(1 \leq j \leq K)$, $y_j > x'^*_1$, then for the same node v we have that

$$\sum_{j, j \in \mathcal{K}^v} y_j \geq y_{(1)} \cdot |\mathcal{K}^v| > x^*_{(1)} \cdot |\mathcal{K}^v| \geq C^v,$$

which also violates the feasibility condition.

As it is impossible to have $x^*_i < y_i$ for a feasible $\mathbf{y}$, we obtain that $x^*_i = y_i$.

(2) Suppose that $\tau_p(\mathbf{x}) = \tau_p(\mathbf{y})$, and for all $j' \in [1, p-1]$, we have

$$x^*_{\phi(x^*_{(j')})} = y_{\phi(x^*_{(j')})}. \tag{V.3}$$

We show that $x^*_{\phi(x^*_{(p)})} = y_{\phi(x^*_{(p)})}$.

Let $i = \phi(x^*_{(p)})$. We show by contradiction that both Case (2.a) $y_i < x_i$ and Case (2.b) $y_i > x_i$ are not possible.

Case (2.a): If $y_i < x_i$ then $y_i < y_{(p)}$. This implies that there exits $q > 0$ and $q \neq p$, s.t. $i = \phi(y_{(q)})$. Then we have $q < p$. However from Eq. V.4 we obtain that $y_{\phi(x^*_{(1)})}..y_{\phi(x^*_{(p-1)})}$ are $p-1$ unique ones to $y_{(q)}$ and are all less than $y_{(p)}$. As a consequence there are p elements that are less than $y_{(p)}$, which is impossible according to the definition of lexmin mapping.

Case (2.b): If $y_i > x_i$, then from the global algorithm we know that

$$x^*_i = \min\{M_i, b(v)\},$$

where v is the corresponding node that has the smallest $b(v)$ in the ith iteration. If $x^*_i = M_k$, then we have $y_i > M_i$ which violates the feasibility condition. If $x^*_i = b(v)$, then we have

$$\begin{aligned}\sum_{j,j\in\mathcal{K}^v} y_{(j)} &= \sum_{j,j\in\mathcal{K}^v, x^*_{(j)}\neq 0} y_{(j)} + \sum_{j,j\in\mathcal{K}^v, x^*_{(j)}=0} y_{(j)} \\ &\geq \Big(\sum_{j,j\in\mathcal{K}^v, x^*_{(j)}\neq 0} y_{(j)}\Big) + y_{(j)} \cdot |\{j | x^*_{(j)} = 0, j \in \mathcal{K}^v\}| \\ &> \Big(\sum_{j,j\in\mathcal{K}^v, x^*_{(j)}\neq 0} x^*_{(j)}\Big) + x^*_p \cdot |\{j | x^*_{(j)} = 0, j \in \mathcal{K}^v\}| \geq C, \end{aligned} \tag{V.4}$$

which also violates the feasibility condition.

By induction from (1) and (2), for $p = 1..m-1$, we have

$$x^*_{\phi(x^*_{(p)})} = y_{\phi(x^*_{(p)})}.$$

It can be proved by applying similar arguments as the above that if $y_{(m)} > x^*_{(m)}$ then $\mathbf{y}$ is not feasible. This proves the second statement.

The second statement directly follows from the Proposition 1 in [67] (If a max-min fair vector exists on a set, then it is unique). □

V.4 Discussion

Although the global algorithm is simple and has many desirable properties, it has a number of important limitations which we relax in subsequent sections.

First, the global algorithm requires global information for the entire network to set the allocation for each node and session. Such global information can be available in a local (e.g., local cluster) and single-domain environment. However, if end nodes are located in different domains or in geographic locations with long RTTs, such global information may not be available or accurate.

Second, the global algorithm requires knowledge regarding desired session rates, which is typically difficult to extract from applications. The application's desired session rate is dependent on its data-handling speed. The data-handling speed at end nodes varies from time to time due to the influence of other jobs running on the same node or disk I/O conditions. Even the application itself may not be able to explicitly estimate its data-handling speed. Therefore, it is more realistic to assume that desired session rates are unknown to the rate-allocation algorithm.

In conclusion, a distributed algorithm that overcomes these limitations must be found. We propose such an algorithm in Chapter VI.

VI

Distributed Bandwidth Sharing

Due to the distributed nature of lambda networks, the global algorithm is difficult to realize for many practical circumstances. We proposed to develop a distributed algorithm with properties similar to those of the global algorithm. In this chapter we study how to conduct an end-node-based rate allocation approach to achieve global max-min fairness under asynchronous and continuous-time settings. We first give a brief introduction to motivations and our approach. We formulate the end-node rate-allocation model within which the end-node-based algorithm operates. It also defines what type of information is available to the end-node algorithms and what information can be fed back between source-sink node pairs. Then we present the end-node-based algorithm that combines rate allocation and adaptation, using a rate-estimation strategy to identify each session's potential rate. Next, we study the stability and convergence properties of the proposed algorithm by using non-linear programming techniques. We answer the following questions: Does the proposed algorithm converge from any initial state? Is the convergence point unique and the same as the one calculated by the global algorithm? Which algorithm parameters impact the convergence speed?

VI.1 Introduction

In Chapter V, we presented a global, synchronous algorithm. We now develop an asynchronous, distributed algorithm. More specifically, we study how end nodes can asynchronously manage their capacity without collaborating with other end nodes. We study the following questions: How can each end node allocate and adapt its rate allocation based on only local information? Will this asynchronous strategy converge to the same max-min fair rate allocation as that calculated by the global algorithm? If so, how quickly does it converge?

In additional to the assumptions about lambda networks, we make the following assumptions about active sessions and end nodes:

- Sessions are long-lived with varied RTTs. They can start and terminate at different time. Each session has a session desired rate, which is unknown to end nodes. And each session's rate can not exceed this session desired rate.
- Information propagation delay between the sink and the source is half RTT.
- Time is continuous and each node runs a source/sink rate allocation and adaptation algorithm.
- Each end node knows its bandwidth capacity and has the knowledge about sessions associated with it. There is no synchronization among end nodes.

Our objective is to fairly allocate and share each end node's capacity among active sessions. Considering the unique characteristics of lambda network model, we have the following observations over previous approaches.

The first observation is that packet loss is not a proper control signal for the problem we are to solve. As there are no inter-network congestions, packet loss only happens at the receivers. As a binary signal (loss or no loss), packet loss does not carry precise bandwidth sharing info and may not be helpful for reaching fairness over RTTs (e.g. TCP). Furthermore, as control schemes based on packet

loss must behave aggressively when packet loss happens, it is difficult to achieve high efficiency over long RTTs. Moreover, packet loss may not be a big concern for long-lived bulk data transfers, as reliable transfer can be guaranteed by loss retransmission or erasure coding [72].

The second observation is that most of the previous approaches are session-based. If not considering the feedback info from routers or switches, each session behaves independently and adjusts its rate based on control signals, such as packet loss, RTT delay, etc. This model fits for traditional Internet and make the system scalable. When we consider the bandwidth sharing problem for lambda networks, we only deal with a limited number of sessions , and we have the knowledge of end-node capacities. Intuitively, one may want to adjust sessions' rates at each end node coordinately to achieve both efficiency and fairness.

The third observation is that most of the previous router- or switch-based rate allocation schemes feed back the same control signal for all sessions. This is not helpful for achieving max-min fairness, as sessions with lower rates are not optimized. When routers and switches no longer play a role in our model, we consider establishing max-min fairness across sessions by giving higher priority to sessions with lower rates.

The last observation is that we also consider applying our bandwidth sharing solution to a wide range of scenarios with different rate adjustment frequency (RTT level, several seconds per adjustment for long-lived FTP session admission control or several minutes for sharing optical paths (lambdas)). A rate-based allocation scheme with adjustable control intervals satisfies this requirement.

Based on these observations, we come up with the approach of conducting rate allocation and adaptation at each end node asynchronously. Each end node explicitly assigns an *expected rate* for each session by allocating its capacity to its associated sessions. The real session rate is then bounded by the expected rates from its source and sink. As each end node periodically attempts to allocate its capacity to its associated sessions, *efficiency* is met when the aggregated session rates equal

the link capacity. As *fairness* involves the relative throughput of sessions, it can be achieved by trying to increase the sessions with lower rates first in the rate allocation scheme. As each session's desired rate is unknown, the rate allocation scheme can not just evenly allocate its capacity to all sessions. To avoid over-allocation and improve efficiency, we use a *rate adaptation* scheme when adjusting session's expected rates.

VI.2 Distributed Rate Allocation and Adaptation

In this section we describe the end-node rate-allocation model and how each source and sink node should allocate its capacity resources to associated sessions.

VI.2.A The End-Node (Distributed) Rate Allocation Model

When asynchronously allocating end-node bandwidth capacity to active sessions, the behavior of each end node is non-cooperative, meaning that each end node attempts to maximize only its own capacity usage, without any knowledge about other end nodes' behavior.

Now we consider the behavior of a typical sink node $v \in V^r$ that is associated with a set of active sessions from multiple sources, as shown in Figure VI.1. Sink nodes run a rate allocation algorithm at the end of each control interval with length d. Available information include the measured sending rate $\bar{\mathbf{x}}^r$, the number of active sessions $\mathcal{K}$ and the sink capacity C_r. More specifically, at the end of each control interval (time t_1) each sink node $v \in V^r$ knows the current measured rate $\bar{x}_k^r(t_1)$ of each session $k \in \mathcal{K}^v$, which is the measured average rate in the period of $(t_1 - d, t_1]$. Then each sink node allocates its capacity and sets expected rates $\hat{x}_k^r$ for each of its sessions for the coming time interval $t' \in (t_1, t_1 + d]$, using the status of all sessions terminating at the same sink:

$$\hat{\mathbf{x}}^r(t') = g(\bar{\mathbf{x}}^r(t_1)), \tag{VI.1}$$

where $\hat{\mathbf{x}}^r(t')$ is the sink expected rate vector and remains fixed in the next control

interval $(t_1, t_1 + d]$, and $g(\cdot)$ represents the rate allocation function. This expected rate information will be fed back to the sender. Note that due to the propagation delay between the source and the sink, the measured sending rate $\bar{\mathbf{x}}^r(t)$ is different from the real sending rate $\mathbf{x}(t)$ at time t.

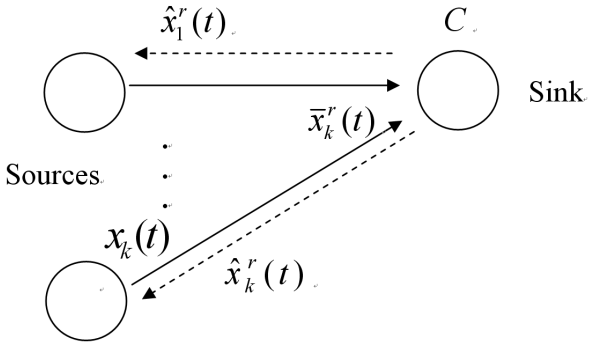


Figure VI.1: The Model

We note the following observations. First, as stated earlier, we do not independently consider the impact of packet loss. The measured sending rate is the sum of the measured actual receiving rate and the packet loss rate. In practice, reliable transfer can be achieved by adding a sequence number to each packet. Second, due to the propagation delay, the measured sending rate is only an estimate and may not be the same as the sender's current sending rate. Third, the observation of any session change will be delayed at the sink side due to the propagation delay.

The end node problem at the source side is similar. The algorithm's inputs are the measured sending rate in the previous control interval, the number of sessions, and the source node's capacity. The algorithm's output is allocated rates $\hat{\mathbf{x}}^r$ from the sender's side for each active session. More specifically, each source calculates the expected rate $\hat{x}_k^s$ for each of its sessions $k \in \mathcal{K}^v$:

$$\hat{\mathbf{x}}^s(t') = h(\bar{\mathbf{x}}^s(t_2)),$$

for any $t' \in (t_2, t_2 + d]$, where $\bar{\mathbf{x}}^s(t_2)$ is the average sending rate in the period

$(t_2 - d, t_2]$, and $\hat{\mathbf{x}}^s(t')$ is the source expected rate for the coming control interval $(t_2, t_2 + d]$. Function $h(\cdot)$ represents the rate allocation function.

With the regularly updated expected rate from sources and sinks, the actual sending rate of session $k \in \mathcal{K}$ at any time t is then determined by the minimum among the expected source rate $\hat{\mathbf{x}}_k^s(t)$, the delayed expected sink rate $\hat{\mathbf{x}}_k^r(t - rtt_k)$, and the session desired rate M_k, formally

$$x_k(t) = \min\{\hat{x}_k^s(t), \hat{x}_k^r(t - rtt_k/2), M_k\}. \tag{VI.2}$$

Due to the propagation delay between the source and sink, the expected rate from the sink is delayed for half RTT. Because we assume that each source and sink node updates at the end of each control interval, the newly calculated expected rates remain fixed until the time of the subsequent calculation. Although we assume fixed-length control intervals for all end nodes, in reality sources and sinks may conduct rate estimations asynchronously.

This end-node-based model differs significantly from a session-based model in which each session uses only the control signal from its own behaviors (e.g., packet loss or delay). In our end-node-based model, the allocated rates reflect the conditions at the sources and sinks where there is traffic contention.

VI.2.B Approximating Max-Min Rate Allocation

Under the end-node rate allocation model, the challenge becomes how to define the rete allocation functions $g(\cdot)$ and $h(\cdot)$ for each sink and source node. The goal is to approximate the max-min rate allocation and adapt each session's current rate to achieve that allocation. The key ideas of our approach include:

- Differentiate between sessions and give higher priority for considering sessions with lower rates.
- Approximate the global max-min rate allocation at each end node and calculate the *target rate* for each session.

- Assign each session an expected rate as an adaptation from its current rate toward the *target rate*.

At the end of each control interval, at each iteration, each node identifies the session with the minimum rate among all its undetermined sessions and assigns the expected rates based on the observed average rate of each session in the previous control interval and the calculated *target rate*. Intuitively, the *target rate* approximates the max-min fair allocation and is calculated by dividing the remaining capacity by the number of undetermined sessions. Therefore, this *target rate* represents the fair rate each undetermined sessions can get when they are not restricted by their desired rate.

In general, we try to maximize the session rates of sessions with lower rates by considering them earlier. In doing so, a session's rate continue to increase as long as its rate is the minimum rate. The rate stops increasing only when it is restricted by a peer node or by the desired rate, or when it reaches the *target rate* (defined in the following subsection). In the latter situation, the target rate is actually $C^v/|\mathcal{K}^v|$, which represents a fair share of bandwidth when no sessions are limited by peer nodes or by their desired rates.

We describe about the calculation of the target rate and the expected rate in details in the following two subsections.

VI.2.C Calculation of the Target Rate

As stated earlier, we try to approximate the global max-min rate allocation by assigning rates to sessions one at a time. We define the *target rate* as the fair rate that each session gets, which is the average rate for sessions sharing a certain capacity. If the distributed algorithm has assigned expected rates for a set of sessions, i.e. the expected rate $\hat{x}_j(t)$ is not equal to 0, then the *target rate* for the remaining sessions is the remaining capacity divided by the number of undetermined sessions (with expected rate 0). Formally, for any time $t' \in (t, t+d]$, the target rate x_f is

defined as:

$$x_f = \frac{C^v - \sum_{i \in \mathcal{K}^v, \hat{x}_i(t') \neq 0} \bar{x}_i(t),}{|\{j | \hat{x}_j(t') = 0, j \in \mathcal{K}^v\}|}. \quad \text{(VI.3)}$$

We update x_f during each iteration of the rate allocation to reflect changes in the remaining capacity and the number of undetermined sessions.

Note that the calculation of the remaining capacity uses the measured rate $\bar{x}_i(t)$ rather than the expected rate $\hat{x}_i(t)$. We will discuss about this in Section VI.2.F.

VI.2.D Calculation of the Expected Rates

Because each node v has no knowledge regarding each session's desired rate M_k, a source or sink cannot tell whether a session with small rate at t is constrained by the session's desired rate or by the capacity of the peer node (a complementary source or sink). This means that a static rate-estimation scheme without rate adaptation is unable to determine or adapt to changes in the desired session rate.

In our approach, we use rate estimation and adaptation to identify the desired rate of each session. The rate estimate allows sessions with lower rates to increase those rates, and thus helps the system to converge to the global max-min rate allocation.

The calculation of the expected rates consists of two steps. First, we calculate the *target rate* (x_f) the sessions with the lowest current rates, approximating the max-min fair allocation. Sessions with low rates are given higher priority. This conforms to the definition of max-min fairness. Second, we assign the expected rate of each session to adapt each session's observed average rate toward its target rate. If the current average session rate is less than the target rate , we adapt it toward the target rate, as

$$\hat{x}_k(t') = \bar{x}_k(t) + \alpha(x_f - \bar{x}_k(t)),$$

for any $t' \in (t, t + d]$, where $\alpha \in (0, 1)$ is the adaptation step size ($0 < \alpha < 1$).

When the minimum session rate of all undetermined sessions is greater than the *target rate* x_f, the remaining capacity is unable to provide sufficient bandwidth for the remaining sessions. In this scenario, we distribute the remaining bandwidth evenly to the remaining sessions, with a *target rate* of

$$\hat{x}_k(t') = x_f, \text{ if } x_k(t) \leq x_f.$$

Two properties of the expected rate calculation should be noted. First, all sessions with lower rates are treated with a higher priority and have the opportunity to receive a higher expected rate. Second, sessions with higher rates receive the same expected rate allocation and thus are treated fairly. These two properties help the distributed algorithm achieve max-min fairness in the long run.

VI.2.E The Algorithm

Formal descriptions of the sink algorithm is in Algorithm 2.

Algorithm 2. ***(Sink Algorithm)***

Input: $\bar{\mathbf{x}}^r(t)$, $\mathcal{K}$, V^R. ***Output***: $\hat{\mathbf{x}}^r(t')$.

1. ***foreach*** v, $v \in V^R$,

2. *Let* $\hat{x}^r_k(t') = 0$, $\forall k \in \mathcal{K}^v$.

3. ***repeat***

4. $\quad x_f = \frac{C^v - \sum_{k,k\in\mathcal{K}^v,\hat{x}^r_k(t')\neq 0} \bar{x}^r_k(t)}{|\{k|\hat{x}^r_k(t')=0,k\in\mathcal{K}^v\}|}$

5. $\quad k = arg\ \min_j \bar{x}_j(t)$, for all $j \in \mathcal{K}^v$ and $\hat{x}^r_j(t') \neq 0$;

6. $\quad$ ***if*** $\bar{x}^r_k(t) < x_f$

7. $\quad\quad \Delta x = \alpha \times (x_f - \bar{x}^r_k(t))$;

8. $\quad\quad \Delta x = max(\ \Delta x,\ \beta \cdot \alpha \cdot x_f\ )$;

9. $\quad\quad \Delta x = min(\ \Delta x,\ \alpha \cdot x_f\ )$;

10. $\quad\quad \hat{x}^r_k(t') = \bar{x}^r_k(t) + \Delta x$;

11. $\quad$ ***endif***

12. ***until*** $\quad |\{j|\hat{x}^r_j(t') = 0, j \in \mathcal{K}^v\}| = 0$ ***or*** $\bar{x}^r_k(t) \geq x_f$

13. ***for all*** $k \in \mathcal{K}^v$

14. ***if*** $\hat{x}_k^r(t') = 0$

15. $\hat{x}_k^r(t') = x_f$*;*

16. ***endif***

17. ***endfor***

The sink algorithm includes upper and lower bounds on the adaptation steps (α, β between 0 and 1). An upper bound ensures that the expected rate does not change too quickly, and limits overshoot. A lower bound supports faster convergence when current rates are close to the *target rate*.

The source algorithm calculates source-expected rates in fashion similar to the sink algorithm. We use a different notation $\hat{x}_k^s$ to denote the expected rate at the source. And the algorithm is described in Algorithm 2.

Algorithm 3. ***(Algorithm 3: Source Algorithm)***

Input*:* $\bar{\mathbf{x}}^s(t)$, $\mathcal{K}$, V^S. ***Output****:* $\hat{\mathbf{x}}^s(t')$.

1. ***foreach*** v, $v \in V^S$,

2. Let $\hat{x}_k^s(t') = 0$, $\forall k \in \mathcal{K}^v$.

3. ***repeat***

4. $x_f = \frac{C^v - \sum_{k,k\in\mathcal{K}^v,\hat{x}_k^s(t')\neq 0} \bar{x}_k^s(t)}{|\{k|\hat{x}_k^s(t')=0,k\in\mathcal{K}^v\}|}$

5. $k =$ *arg* $\min_j x_j(t)$, for all $j \in \mathcal{K}^v$ and $\hat{x}_j^s(t') \neq 0$*;*

6. ***if*** $\bar{x}_k^s(t) < x_f$

7. $\Delta x = \alpha \times (x_f - \bar{x}_k^s(t))$*;*

8. $\Delta x = max(\ \Delta x,\ \beta \cdot \alpha \cdot x_f\ )$*;*

9. $\Delta x = min(\ \Delta x,\ \alpha \cdot x_f\ )$*;*

10. $\hat{x}_k^s(t') = \bar{x}_k^s(t) + \Delta x$*;*

11. ***endif***

12. ***until*** $|\{j|\hat{x}_j^s(t') = 0, j \in \mathcal{K}^v\}| = 0$ ***or*** $\bar{x}_k^s(t) \geq x_f$

13. ***for all*** $k \in \mathcal{K}^v$

14. ***if*** $\hat{x}_k^s(t') = 0$

15. $\hat{x}_k^s(t') = x_f$*;*

16. ***endif***

17. ***endfor***

The only difference between sink and source algorithms is the notation used. We here assume that the source algorithm has no knowledge regarding the expected rate set by the sink thus allocate its capacity to associated sessions independently.

The actual sending rate of a session $k \in \mathcal{K}$ at any time t is the minimum among session k's desired rate, half-RTT delayed sink expected rate, and source expected rate, defined as

$$x_k(t') = min\{\hat{x}_k^s(t'), \hat{x}_k^r(t' - rtt_k/2), M_k\}. \tag{VI.4}$$

Intuitively, for flows with low session rates we have $x_f > x_k$, meaning that the *target rate* is higher than the current session, giving $\hat{x} > \bar{x}_k$. As a result, sessions with low rates have room to increase if desired by the application. If a session's M_k increases beyond the target rate, and there is available capacity, it will be allocated more rate.

VI.2.F Discussion

In this section, we discuss some of the design issues raised by the distributed end-node bandwidth-sharing algorithm.

In the proposed end-node bandwidth-sharing algorithm, we assume that the source algorithm has no knowledge regarding the sink's expected rate. The sink's expected rate is fed back only to the session level, which is separated from the source algorithm.

Due to the RTT delay, information about the observed sending rates at sinks is delayed. This generally suggests the control interval be not too short. Like similar parameters set in other protocols, the control interval should be greater than the maximum RTT of the active sessions. We will demonstrate that the choice of control interval length will not affect the algorithm's stability and convergence

properties, but will affect the transitional behavior of sessions and the speed of convergence.

The calculation of the remaining capacity uses the measured rate $\bar{x}_i(t)$ rather than the expected rate $\hat{x}_i(t)$. This is because the measured rate is reachable for each session and it may not be the case for the expected rate due to the unknown session desired rate. By doing so, we ensure that the expected rate for sessions with lower rates is higher than their current measured rate, thus give them room to increase their rates. However this may lead to over-allocation of the end-node capacity, in the worst case, by $\alpha \cdot C$. Potential packet loss can limited by setting $\alpha$to be a relatively small value and setting capacity C to be lower than the peak NIC speed the end node can reach.

Given RTT delay, the sink's new expected rate will not reach the source in half RTT, during which time the source still sends packets at the former rate. Therefore, packet loss may still occur at a ratio that is proportional to the parameter. In practice, we can eliminate possible packet loss by setting the capacity used to be less than the real node capacity

Parameters (i.e. α and β) are constant and are independent of the network topology (e.g., the number of sources and sinks, the delay and node capacities) and sessions' desired rates. We will use simulation studies to show how these values are set.

If there is only one active session, the parameter α controls the convergence speed. The relationship between α and the convergence speed, and the question of whether we can accelerate the convergence speed in this scenario, will be explored through the simulation studies.

Finally, our end-node bandwidth-sharing algorithm is also a rate-based scheme. It does not rely on per-packet acknowledgement and queueing delay signals. Implementing such a scheme does not require kernel level coding and can be done at the user level. We will discuss possible implementation designs in Chapter IX.

VI.3 Stability and Convergence

We study the stability and convergence properties of the distributed rate allocation algorithm. In particular, we show that (a) the distributed algorithm has the max-min fair allocation $\mathbf{x}^*$ in the global algorithm as an equilibrium point and (b) the distributed algorithm causes the session rates to converge, regardless of their initial states.

Let T_{max} be $(max_k\{rtt_k\} + 2 \cdot d)$, representing the maximal possible time legged between the rate updates from the source and sink. The stability and convergence results are formally described as follows.

Proposition 2. *(**Stability**) The max-min fair rate vector x^* calculated by the global algorithm is an equilibrium point for the distributed algorithm under continuous time system, i.e. if there exists $t_0 > T_{max}$, and for any t $(t_0 - T_{max} < t \leq t_0)$ it holds that*

$$\mathbf{x}(t) = \mathbf{x}^*$$

, then for any $t' > t_0$ we have that

$$\mathbf{x}(t') = \mathbf{x}^*.$$

Proposition 3. *(**Convergence**) From any initial rate vector $\mathbf{x}(0)$ (at time $t = 0$), under the distributed rate allocation algorithm the rate vector $\mathbf{x}$ converges to the final rate vector $\mathbf{x}^*$ computed by the global algorithm, i.e. there exists time $t_0 > 0$ and for any time $t > t_0$, we have that*

$$\mathbf{x}(t) = \mathbf{x}^*.$$

Before proving Proposition 2 and 3 we introduce two definitions: *leximin mapping* and *leximin ordering.*

Definition 5. ***(Leximin Mapping)**. The leximin order mapping $\tau : R^N \rightarrow R^N$ is the mapping that sorts vector $\mathbf{x}$ in non-decreasing order. We have that*

$$\tau(\mathbf{x}) = (x_{(1)}, ..., x_{(n)}),$$

where $x_{(1)} \leq x_{(2)} \leq ... \leq x_{(n)}$*, and we refer* $x_{(i)}$ *as the ith element of the leximin mapping of vector* $\mathbf{x}$*.*

We let $\tau_m(\mathbf{x})$ *denote the vector of first m elements from* $\tau(\mathbf{x})$*, which is* $(x_{(1)}, ..., x_{(m)})$*. We define the leximin reverse mapping function* $\phi(\cdot)$ *as*

$$\phi(x_{(j)}) = i,$$

where x_i *corresponds to* $x_{(j)}$ *after leximin mapping.*

Note that if multiple elements have the same value, we sort them during leximin mapping by their original index. This allows us to to compare two vectors in leximin order.

Definition 6. *(**Leximin Ordering**) Given that* $(x_{(1)}, ..., x_{(n)})$ *and* $(y_{(1)}, ..., y_{(n)})$ *are the leximin mappings of* $\mathbf{x}$ *and* $\mathbf{y}$*, we define the lexicographic ordering of* $\mathbf{x}$ *and* $\mathbf{y}$ *as* $\mathbf{x} \succ^L \mathbf{y}$*, if and only if* $(\exists i > 0) x_{(i)} > y_{(i)}$ *and* $(\forall 0 < j < i) x_{(j)} = y_{(j)}$*. We also define* $\mathbf{x} \succeq^L \mathbf{y}$*, if and only if* $\mathbf{x} \succ^L \mathbf{y}$ *or* $\mathbf{x} = \mathbf{y}$*; and* $\mathbf{x} \prec^L \mathbf{y}$*, if and only if* $\mathbf{y} \succ^L \mathbf{x}$*.*

We let $\equiv_n^L$ *denote a relation between two vectors* $\mathbf{x}$ *and* $\mathbf{y}$*, such that* $\mathbf{x} \equiv_n^L \mathbf{y}$ *if and only if* $\tau_n(\mathbf{x}) = \tau_n(\mathbf{y})$*, and* $x_{\phi(x_{(i)})} = y_{\phi(x_{(i)})}$*,* $i = 1..n$*.*

It is shown in [67] that if a max-min fair vector exists, then it is the unique leximin maximal vector. Thus the existence of a max-min fair vector implies the uniqueness of a leximin maximum.

We first show in Lemma 1 several leximin ordering properties of the expected rate calculation described earlier.

Lemma 1. *For the distributed algorithm described in Section IV, let* $\bar{\mathbf{x}}(t_0)$ *be the average measured rate of K sessions during the control interval* $(t_0 - d, t_0]$*. Let* $\mathbf{x}_{min}$ *be the minimum rate in leximin order within the control interval, i.e.*

$$\mathbf{x}_{min} = Min^L_{t \in (t_0 - d, t_0]} \{\mathbf{x}(t)\}.$$

Let $\hat{\mathbf{x}}$ *be the resulting expected rate and* $\mathbf{x}^*$ *be the equilibrium rate vector calculated by the global algorithm. Then the following relations hold.*

(a) It holds that $\mathbf{x}_{min} \preceq^L \bar{\mathbf{x}}$*;*

(b) If $\bar{\mathbf{x}} \prec^L \mathbf{x}^*$*, then* $\bar{\mathbf{x}} \prec^L \hat{\mathbf{x}}$*;*

(c) If $\mathbf{x}^* \prec^L \bar{\mathbf{x}}$*, and* $\mathbf{x}_{min} \prec \mathbf{x}^*$*, then* $\mathbf{x}_{min} \prec^L \hat{\mathbf{x}}$*.*

Brief proofs are as follows.

Proof. (a) Suppose $\bar{\mathbf{x}} \prec^L \mathbf{x}_{min}$. Then there exists $n < K$ s.t. for any $i \leq n$, $\bar{\mathbf{x}}_{(i)} = \mathbf{x}_{min,(i)}$ and $\bar{\mathbf{x}}_{(n+1)} < \mathbf{x}_{min,(n+1)}$. Let $j = \phi(\bar{\mathbf{x}}_{(n+1)})$, then it is obvious that $\mathbf{x}_j < \mathbf{x}_{min,j}$, which conflicts with the definition of $\mathbf{x}_{min}$.

(b) The distributed algorithm described in Section IV implies that for any $k \in K$, only when $\bar{\mathbf{x}}_k > \mathbf{x}_k^*$, we have $\hat{\mathbf{x}}_k < \bar{\mathbf{x}}_k$. It can be proved by using similar arguments as in (a) that it is impossible to have $\bar{\mathbf{x}} \succeq^L \hat{\mathbf{x}}$. Note that by combining (a) and (b) we can obtain that if $\bar{\mathbf{x}} \prec^L \mathbf{x}^*$, then $\mathbf{x}_{min} \prec^L \hat{\mathbf{x}}$.

(c) According to our distributed algorithm, for all $k \in K$, if $\bar{\mathbf{x}}_k > \mathbf{x}^*$, then $\hat{\mathbf{x}}_k > \mathbf{x}^*$; and if $\bar{\mathbf{x}}_k < \mathbf{x}^*$, then $\hat{\mathbf{x}}_k > \bar{\mathbf{x}}_k$. Suppose $\mathbf{x}_{min} \equiv^L \hat{\mathbf{x}}$ or $\mathbf{x}_{min} \succ^L \hat{\mathbf{x}}$. When $\mathbf{x}_{min} \equiv^L \hat{\mathbf{x}}$, $\mathbf{x}^* \equiv^L \bar{\mathbf{x}}$ which lead to confliction with the assumption. When $\mathbf{x}_{min} \succ^L \hat{\mathbf{x}}$, then there exists $n < K$ s.t. for any $i \leq n$, $\mathbf{x}_{min,(i)} = \hat{\mathbf{x}}_{(i)}$ and $\mathbf{x}_{min,(n+1)} > \hat{\mathbf{x}}_{(n+1)}$. Let $j = \phi(\bar{\mathbf{x}}_{(n+1)})$. It can be shown that $\hat{\mathbf{x}}_j = \mathbf{x}_j^*$ and $\mathbf{x}_{min} \succ^L \mathbf{x}^*$, which leads to contradiction with the assumption too.

□

We now provide a proof to Proposition 2.

Proof. We first to show that if the average rate $\bar{\mathbf{x}}(t_0)$ of the control interval ended at time t_0 is feasible, then for any session $k \in [1, ..K]$, we have that for any time $t' \in (t_0, t_0 + d]$,

$$\hat{x}_k(t') \geq \bar{x}_k(t_0). \tag{VI.5}$$

The above is true for both sources and sinks. Recall that in the distributed algorithm the expected rate $\hat{x}_k$ of session k depends on the *target rate* x_f and its average rate $\bar{x}_k(t_0)$. And only when x_f is less than the average rate $\bar{x}_k(t_0)$, the expected rate will be set to x_f, thus smaller than the current average rate:

$$x_k(t') = x_f < \bar{x}_k(t_0).$$

This only happens when the remaining capacity is less than the current aggregate rate of all undetermined sessions. In this case we have

$$\sum \bar{x}_k(t_0) > C$$

, indicating that the rate vector $\bar{\mathbf{x}}$ is not feasible. Therefore given any feasible rate vector $\bar{\mathbf{x}}$ in time period $(t_0 - d, t_0]$, the expected rates of each session k in the next time interval $(t_0, t_0 + d]$ satisfies Eq. VI.5.

Let t_1 be the earliest control interval ended after t_0 at the source and sink. The rate vector $\mathbf{x}(t')$ is determined by the latest rate estimates which are based on the rate vector $\mathbf{x}^*$ at each source and sink during $(t_0 - T_{max}, t_0]$. Then we have

$$\hat{x}_k^s(t') \geq x_k(t_0), \text{and } \hat{x}_k^r(t' - rtt_k/2) \geq x_k(t_0). \tag{VI.6}$$

We now show that indeed

$$x_k(t') = x_k(t_0) = x_k^*, \tag{VI.7}$$

for any $k \in [1...K]$ and $t' \in (t_0, t_1]$.

Recall that the rate of each session k at time t is determined by

$$x_k(t) = \min\{\hat{x}_k^s(t), \hat{x}_k^r(t - rtt_k/2), M_k\}.$$

For any $t' \in (t_0, t_1]$ we consider the following two separate scenarios:

(1) $x_k^* = M_k$,

(2) $x_k^* < M_k$ and $x_k(t') = \hat{x}_k^s(t')$ or $x_k(t') = \hat{x}_k^r(t' - rtt_k/2)$.

Case (1): $x_k^* = M_k$. We can get by Eq. VI.6 that

$$x_k(t') = \min\{\hat{x}_k^r(t' - rtt_k/2), \hat{x}_k^s(t'), M_k\} \geq x_k(t_0) = x_k^*.$$

As it is always true that $x_k(t') \leq M_k$, it directly follows that $x_k(t') = x_k^*$.

Case (2): $x_k^* < M_k$ and $x_k(t') = \hat{x}_k^s(t')$ or $x_k(t') = \hat{x}_k^r(t' - rtt_k/2)$. This implies that session k is constrained by its source or sink but not its desired rate. We now show that this also implies that $x_k(t') = x_k^* = x_f$, where x_f is the target rate given by the expected rate calculation. Let v be the corresponding node on

which $x_k(t')$ equals to its expected rate at time t'. We notice the following facts from the calculation of x^* and expected rates:

(a) If session k is not restricted by its desired rate, the aggregate expected rate at node v equals to C^v;

(b) the rate of session k equal to the highest session rate among all sessions at node v.

Therefore assuming that the most recent expected rate calculation at this bottleneck node happened at time t_2 ($t_2 \in (t_0 - T_{max}, t_0]$), from the calculation of x_f (Eq. VI.3) in the distributed algorithm, we can obtain that for any time $t'' \in (t_2, t_2 + d]$,

$$\begin{aligned} x_f &= \frac{C^v - \sum_{i \in \mathcal{K}^v, \hat{x}_i(t'') \neq 0} \bar{x}_i(t_2)}{|\{j | \hat{x}_j(t'') = 0, j \in \mathcal{K}^v\}|} \\ &= \frac{C^v - \sum_{i \in \mathcal{K}^v, \bar{x}_i(t'') < \bar{x}_i(t_2)} \bar{x}_k(t_2)}{|\{j | \bar{x}_j(t_2) \geq \bar{x}_k(t_2), j \in \mathcal{K}^v\}|} \\ &= \frac{C^v - \sum_{i \in \mathcal{K}^v, x_i^* < x_k^*} x_i^*}{|\{j | x_j^* \geq x_k^*, j \in \mathcal{K}^v\}|} = x_k^*. \end{aligned}$$

Thus we have

$$\hat{x}_k(t') = \hat{x}_k(t'') = \hat{x}_k(t_2) = x_k^*.$$

Therefore for any $t' \in (t_0, t_1]$, $x_k(t') = x_k^*$. The same arguments can also be applied for the time period $(t_1, t_3]$, where t_3 is the end of the next control interval among sources and sinks after t_1. By induction we have that for any $t' > t_0$,

$$\mathbf{x}(t') = \mathbf{x}^*.$$

We have shown that the rate vector calculated by the global algorithm is an equilibrium point for the distributed algorithm under the asynchronous system. By Proposition 1, the equilibrium point is max-min fair and unique. □

Before proving Proposition 3, we state several preliminary lemmas and give an outline of the proof for each of them.

Lemma 2. *Let $\mathbf{x}^*$ be the result of the global algorithm. Let $\mathbf{x}(t)$ be the rate vector of the distributed algorithm at time t. If there exists $t_0 \geq 0$, $n \in [1..K]$ such that for*

any $t \in (t_0 - T_{max}, t_0]$ *we have*

$$\mathbf{x}(t) \equiv_n^L \mathbf{x}^*, \tag{VI.8}$$

then for any $t' > t_0$*, it holds that*

$$\mathbf{x}(t') \equiv_n^L \mathbf{x}^*. \tag{VI.9}$$

Proof. We prove by showing the correctness of the following two statements.

If Eq. VI.8 is true for time period $(t_0 - T_{max}, t_0]$, then for time $t \in (t_0, t_1]$, where t_1 is the end if the first control interval after t_0 among sources and sinks. We have that (a) For any $j \in [1..n]$, let $k = \phi(x_{(j)})$. Then it holds that

$$x_k(t) = x_k(t_0);$$

and (b) For any $j \in [n+1..K]$, it is true that

$$x_{(j)}(t) \geq x_{(n)}(t_0).$$

We prove (a) by induction on $j = 1..n$.

(1) When $j = 1$, let $k = \phi(x_{(1)})$. Then we have

$$x_k(t) = x_k^* = x_{(1)}.$$

We consider two cases:

(1a): $x_k(t_0) = M_k$;

(1b): $x_k(t_0) < M_k$ and either $x_k(t) = \hat{x}_k^s(t)$ or $x_k(t) = \hat{x}_k^r(t - rtt_k/2)$.

Similar arguments used in the proof of Proposition 2 can be directly applied, and it can be proved that

$$x_k(t) = x_k(t_0) = x_k^*.$$

(2) Suppose (a) is true for $1..j-1$ $(j > 0)$. We need to show that the same is true for j, i.e. if $k = \phi(x_{(j)})$ then we have for any $t \in (t_0, t_1]$,

$$x_k(t) = x_k(t_0).$$

Notice the fact that the calculation of $\hat{x}_k^r(t - rtt_k/2)$ and $\hat{x}_k^s(t)$ is only based on the source/sink capacity of session k and sessions with lower average rate than

$\bar{x}_k$ at the same source or sink node. This is also true when calculating x^*_k in the global algorithm. Applying the same technique used in the proof of the previous proposition, it can be proved that

$$x_k(t) = x_k(t_0) = x^*_k.$$

Now we show the correctness of (b). Suppose that there exists $j \in [n+1, K]$, and let $k = \phi(x_{(j)})$, s.t.

$$x_k(t) < x_{(n)}(t_0) = x^*_{(n)}. \tag{VI.10}$$

As by the definition of leximin ordering, $x_k(t) \geq x_{(n)}(t_0)$, we then have

$$x_k(t) < x_k(t_0). \tag{VI.11}$$

As stated before, Eq. VI.11 is only valid when $x_k(t) > x_f$, where x_f is the *target rate* at the source or sink of session i when calculating its expected rate. However from the fact that

$$\mathbf{x}(t) \equiv^L_{j-1} \mathbf{x}^*,$$

we have

$$x_f = \frac{C - \sum_{k,k\in\mathcal{K}^v,\hat{x}_k\neq 0} \bar{x}_k(t)}{|\{k|\hat{x}_j = 0, k \in \mathcal{K}^v\}|} \geq \frac{C^v - \sum_{i=1}^{n-1} x^*_{(i)}}{K-n+1} \geq x^*_{(n)}.$$

Therefore $x_k(t) > x^*_{(n)}$, which contradicts with Eq. VI.10.

Using the same induction on t, we have that for any $t > t_0$ both statements (a) and (b) are true. □

Lemma 2 implies that if a subset of rates which are the lowest ones of $\mathbf{x}$ converges to the subset of lowest rates of $\mathbf{x}^*$, then these rates will stay unchanged afterward.

Lemma 3. *Let $\mathbf{x}^*$ be the result of the global algorithm. Let $\mathbf{x}(t)$ be the rate vector of the distributed algorithm at time t. Let Min^L denote the minimum vector in leximin orders among a set of vectors. If for $t \in (t_0 - T_{max}, t_0]$,*

$$Min^L_{t\in(t_0-T_{max},t_0]}\{\mathbf{x}(t)\} \prec^L \mathbf{x}^*,$$

then for any $t' > t_0$ we have that

$$Min^L_{t\in(t_0-T_{max},t_0]}\{\mathbf{x}(t)\} \prec^L \mathbf{x}(t'). \tag{VI.12}$$

Proof. Let $\mathbf{x}(t_2)$ be the minimum vector value in leximin order, where $t_2 \in (t_0 - T_{max}, t_0]$, i.e.

$$\mathbf{x}(t_2) = Min^L_{t\in(t_0-T_{max},t_0]}\{\mathbf{x}(t)\}.$$

We still let t_1 ($t_1 > t_0$) be the end of the first control decision interval at sources or sinks. Then the source and sink expected rates $\hat{x}$ for the time period $(t_0, t_1]$ are based on the average rate during time periods of d within $(t_0 - T_{max}, t_0]$. The minimum rate vector of each control interval should be no smaller than $\mathbf{x}(t_2)$ in leximin order per definition of $\mathbf{x}(t_2)$.

We let the source and sink estimation be $\hat{x}^s(t_3)$ and $\hat{x}^r(t_4)$, respectively, where t_3 and t_4 are the end of latest two control intervals at the source and sink, where $t_3, t_4 \in (t_0 - T_{max}, t_0]$.

Suppose there exists $n \in [0, .., K]$ and for any time $t \in (t_0 - T_{max}, t_0]$, $\mathbf{x}(t) \equiv^L_n \mathbf{x}^*$. Then $\mathbf{x}(t_2) \equiv^L_n \mathbf{x}^*$, and $\mathbf{x}_{n+1}(t_2)$ is the smallest rate element when excluding the first n elements. By applying Lemma 2 and the same technique used in the proof of Lemma 1 and 2, we can prove the following facts:

(1) for $k = 1, ..., n$ and for any $t' \in (t_0, t_1]$, we have $x_{(k)}(t') = x^*_{(k)}$;

(2) for any $t' \in (t_0, t_1]$ and for any $j = n + 1, ..., K$, we have $x_{(j)}(t') > x_{(n+1)}(t_2)$.

We here omit the detailed proof. Based on (1) and (2) and using induction on t', for any $t' > t_0$ we obtain that Eq. VI.12 holds.

□

The next lemma uses the results from the previous two lemmas to show that the actual rate vector gradually converges.

Lemma 4. *Let $\mathbf{x}^*$ be the result of the global algorithm. Let $\mathbf{x}(t)$ be the rate vector of the distributed algorithm at time t. If for time $t \in (t_0 - T_{max}, t_0]$, $\mathbf{x}(t) \equiv^L_n \mathbf{x}^*$,*

where $0 \leq n < K$*, and there exist time* $t' \in (t_0 - T_{max}, t_0)$*,* $x_{(n+1)}(t') \neq x^*_{(n+1)}$*, then there exits* $t_1 > t_0$*, s.t. for any time* $t'' > t_1$ *we have that*

$$\mathbf{x}(t'') \equiv^L_{n+1} \mathbf{x}^* \tag{VI.13}$$

Proof. Let $\mathbf{x}(t_2)$ be the minimum value in leximin order, where $t_2 \in (t_0 - T_{max}, t_0]$, i.e.

$$\mathbf{x}(t_2) = Min^L_{t\in(t_0-T_{max},t_0]}\{\mathbf{x}(t)\}.$$

Given that for time $t \in (t_0 - T_{max}, t_0]$, $\mathbf{x}(t) \equiv^L_n \mathbf{x}^*$, we consider the following two scenarios regarding $\mathbf{x}(t_2)$: (1) $\mathbf{x}(t_2) \succeq^L \mathbf{x}^*$, and (2) $\mathbf{x}(t_2) \prec^L \mathbf{x}^*$

Case (1): The relations $\mathbf{x}(t) \equiv^L_n \mathbf{x}^*$ and $\mathbf{x}(t_2) \succeq^L \mathbf{x}^*$ imply that for any $j \in [n+1, K]$, it holds that

$$x_{(j)}(t_2) \geq x^*_{(n)}.$$

Therefore it can be easily proved that for any $j \in [n+1, K]$, let $i = \phi(x_{(j)})$ and let t_3 be the next rate update time after t_0, then for any $t'' \in (t_0, t_3]$, we have

$$x_i(t'') \geq x^*_{(n)}.$$

We can also show by using similar technique as in the proof of Lemma 1 that when $i' = \phi(x_{(n+1)})$, then we have

$$x_{i'}(t'') = x^*_{n+1}.$$

Together with the fact that $\mathbf{x}(t'') \equiv^L_n \mathbf{x}^*$ (by Lemma 2), we have

$$x_{(n+1)}(t'') = x^*_{(n+1)},$$

and Eq. VI.13 is valid.

Case (2): By directly applying Lemma 3, we obtain that if t_3 is the next rate update time after t_0, then for any $t'' \in (t_0, t_3]$, $\mathbf{x}(t_2) \prec^L \mathbf{x}(t'')$ when $\mathbf{x}(t_2) \neq \mathbf{x}^*$.

Also from the fact that the smallest increment in the distributed algorithm is $\alpha \cdot \beta \cdot C^v/|\mathcal{K}^v|$. Therefore within finite time intervals, i.e. there exits time $t_1 > 0$, there is no unconverged session with rate lower than $x^*_{(n+1)}$. We then have that

$$\mathbf{x}(t_1) \succeq^L \mathbf{x}^*.$$

This becomes the same as case (1).

To conclude, in both cases (1) and (2), Eq. VI.13 becomes valid within finite time. Lemma 2 ensures that Eq. VI.13 is always valid afterwards. □

Lemma 3 shows that if the smallest n ($n \in [0, .., K-1]$) elements in the rate vector converges, then the $(n+1)$th element will converge to its corresponding equilibrium rate within finite time. Based on these lemmas we now give an outline of the proof for Proposition 3.

Proof. Proposition 3 can be proved by applying Lemma 4 through induction on n from 0 to K-1, i.e. we can show the following:

(1) It holds that there exits $t_1 \geq 0$ and for any $t \geq t_1$ we have that $\mathbf{x}(t) \equiv_1^L \mathbf{x}^*$. This can be directly obtained from Lemma 4 by setting $n = 0$.

(2) Suppose there exits $t_{n-1} \geq 0$ and for any $t \geq t_{n-1}$ we have that $\mathbf{x}(t) \equiv_{n-1}^L \mathbf{x}^*$. Then there exits $t_n \geq t_{n-1}$ and for any $t \geq t_n$ we have that $\mathbf{x}(t) \equiv_n^L \mathbf{x}^*$. This is true by applying Lemma 4 and Lemma 1.

Combining (1) and (2), we conclude that there exits $t_K \geq 0$ and at any time $t \geq t_K$ we have that

$$\mathbf{x}(t) \equiv_K^L \mathbf{x}^*.$$

This proves the proposition. □

VI.4 Summary and Discussions

We have proposed a distributed end-node bandwidth-sharing algorithm that allocates each end node's capacity and adapts each session's rate. Unlike the global algorithm described in Chapter V, the algorithm does not require global information and there is no synchronization among the end nodes. We have proved that a unique equilibrium state exists for the distributed bandwidth-sharing scheme, that it is a max-min rate allocation, and that the rate allocation is the same as that computed by the global algorithm. Further, we have shown that the distributed

rate-allocation algorithm converges within a finite number of steps from any initial or transitional state.

While our approach achieves the same "generalized max-min fair" rate allocation as described in [51], it differs from other ATM rate allocation schemes (e.g., [51, 29, 87]) in the following aspects:

- Previous studies provide the same rate control signal from a router or switch to all associated sessions. We consider providing each session with an explicit control signal according to its specific status. This ensures achieving max-min fairness over long-delay links.

- Most studies of ABR traffic do not consider rate adaptation, which might result in severe and instant network congestion. We will use rate adaptation to smooth the transitions

- Our rate adaptation is conducted during the centralized rate allocation stage at each node. This is not the per-session rate-adaptation scheme observed at each source under other studies (e.g., EPRCA [87])

VII

Simulation Studies

In this chapter, we discuss simulations that illustrate the stability and convergence properties of the proposed distributed end-node bandwidth-sharing algorithm. We study the impact of design parameters such as step sizes, the desired session rates, and control intervals. We use an NS-2 [16] simulator to generate different network topologies and traffic patterns, including large and small point-to-point, point-to-multipoint, and multipoint-to-point networks.

VII.1 Methodology

Simulation studies allow us to isolate the performance of algorithms from the noise introduced by real network systems. To understand the convergence and stability properties of the end-node bandwidth-sharing algorithm, we study the performance of the bandwith-sharing algorithm in a set of experiments in a simulated network environment under different design parameters, network topology and traffic patterns.

VII.1.A Simulation Topologies

We use an NS2 simulator to generate a set of network topologies ranging from point-to-point to multipoint-to-multipoint, including

- Single link case. In this case, there is only one source node and one sink node and they have equivalent capacity. This simple topology tests how a single session behaves in the absence of other competing sessions. When there are multiple concurrent sessions, experiments validate whether source and sink capacity are equally shared by the sessions.

- Multiple sources and single sink. In this case, the aggregate capacity of the sources exceeds the end-node capacity at the sink. Therefore, network contention exists whenever multiple sessions merge at the sink. Experiments with different parameters in this case give insight into how the sink shares its capacity among sessions with different demands.

- Single source and multiple sinks. In this case, the single source needs to allocate its bandwidth to multiple sessions terminating at different sinks. Sessions may have different desired rates and different parameters (e.g., RTT and starting/ending time). This scenario validates whether the end-node bandwidth-sharing algorithm can share the source bandwidth fairly among sessions.

- Multiple sources and multiple sinks. In this networked case, sessions from sources terminate at different sinks. The number of nodes ranges from 8 to 1,024. Experiments in this environment validate whether the end-node bandwidth-sharing algorithm can achieve fairness and stability under various complex network topologies.

In all of these experiments, the node capacity is set at 1,000 Mbps. The rate is normalized to 1 when studying session-convergence properties. A packet switch is placed in front of each source- and sink-node to reflect the real lambda-network settings.

VII.1.B Simulation Parameters

Using the various network topologies mentioned in the previous sub-section, we validate the convergence properties of the end-node bandwidth-sharing algorithm while varying a set of protocol and simulation parameters. We divide these parameters into two categories: protocol-related parameters and simulation-related ones.

The protocol-related parameters that we vary during the experiments are as follows.

- The step size α. By varying α we can tell how different values of α affect the convergence rate and stability under each network topologies. The default value of α is set as 0.15. And step size α can vary from 0.025 to 0.2.

- The step bound β. By varying β, experiments show what is the influence of different values of β during the convergence process. The default value of β is set as 0.2, and it can vary between 0.05 to 0.3.

- The desired session rates. Sessions with different desired rates may lead the system to converge at different equilibrium points. We validate this by assigning sessions with different rates.

- The control interval. In our end-node bandwidth-sharing algorithm, each end node runs the rate-allocation algorithm at every control interval. By varying the control interval for each node, we can study influence of the control interval on the convergence rate.

The simulation scenario-related parameters include:

- The link RTT. We vary the link RTT for different source and sink pairs to validate whether the distributed end-node bandwidth-sharing algorithm is fair to sessions with different RTTs.

- The number of concurrent sessions. We vary the number of concurrent sessions at one or multiple sources or sinks to study whether the system converges under different traffic loads.

- Different starting/ending time. In order to understand the transitory behavior of the distributed end-node bandwidth-sharing algorithm, we consider scenarios in which sessions join and leave at different times.

VII.1.C Performance Metrics

To validate the convergence and stability properties observed in each experiment, we adopt two performance metrics. The first one is the length of time that is required for session rates to converge from any initial or transitional state. This is a simple yet effective measurement for simple point-to-point or multipoint-to-multipoint scenarios. Depending on the parameter variables, the total length of time can be expressed in terms of seconds, the number of RTTs, or the number of control intervals.

The second performance measurement, used to measure the algorithm's convergence properties for large networks, is the distance between the current rate vector $\mathbf{x}(t)$ and the max-min fair equilibrium $\mathbf{x}^*$ (calculated for example by the global algorithm).By experimenting with various distance metrics, we found that 2-norm distance, defined as follows, works well:

$$D(t) = \Big[\sum_{i=1}^{K}(x_i(t) - x_i^*)^2\Big]^{1/2}.$$

Therefore in the equilibrium state, we have $D = 0$. This distance metric gives insight into whether session rates are converging or diverging at any point in time.

VII.2 Dynamics of a Single Session

The simplest case is a single session on a single link, as shown in Figure VII.1. We omit the packet switch at both the source and sink in all figures. This simple point-to-point setting helps to illustrate the basic behavior of the adaptation used in the distributed end-node bandwidth sharing algorithm.

Table VII.1: Default values for experiment parameters

Parameter	Default Value
α	0.15
β	0.2
RTT	50 ms
Ctrl. Interval	RTT
Link Capacity	1 Gbps

The link bandwidth and session demand are both set at 1 Gbps. In the following four experiments, we change one of the parameters (α, β, RTT or the control interval), while holding the other parameters at fixed values, as shown in Table VII.1.

In each scenario, there is no global synchronization and the experiment begins with each node having a slight difference in time. The rate trajectories of the single session is depicted in Figure VII.2.

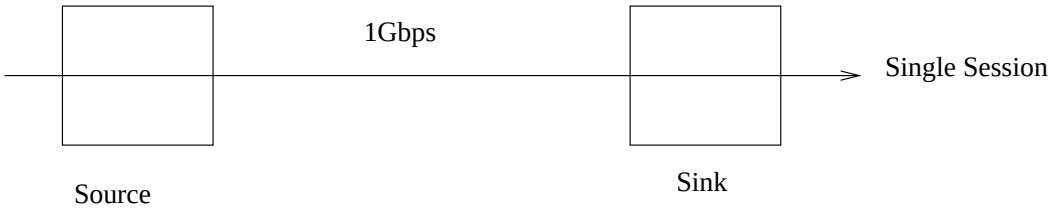
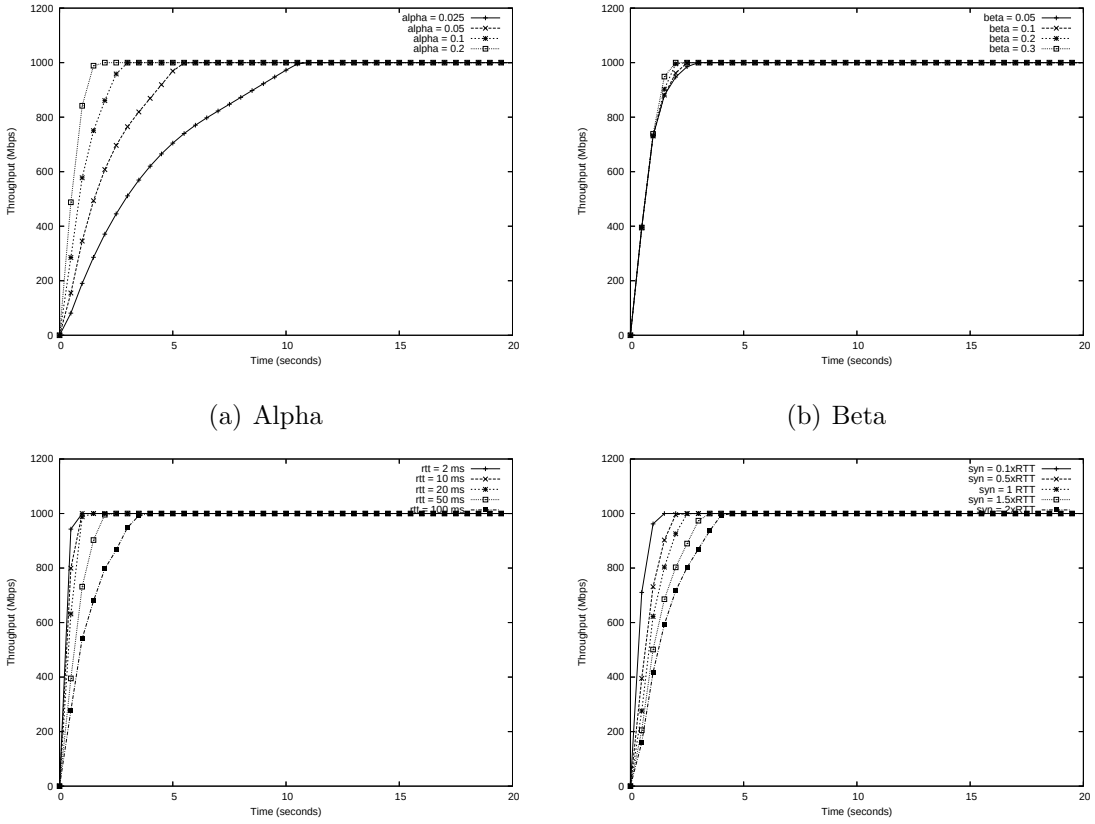


Figure VII.1: A single sink, single source Topology

As shown in Figure VII.2(a), with step sizes α varying from 0.025 to 0.2, the time taken to reach the peak rate varies. It takes up to 10s (or 200 RTT) for the session to reach full capacity. This is because the parameter α determines how fast each session can increase its rate during each control interval. Therefore, α needs to be a relatively larger value to achieve a higher convergence speed. We set the default value of α to be 0.15 in most of our simulations.

Figure VII.2(b) shows the rate trajectories under different β, from which we can tell that the value of *beta* has little effect on the general convergence rate. However, it determines how fast the rage gets to converge when it is close to the



(a) Alpha

(b) Beta

(c) RTT

(d) Control Interval

Figure VII.2: Trajectories of a single session under different parameter values: (a) Various α (b) Various β (c) Various RTT (d) Various Control Interval

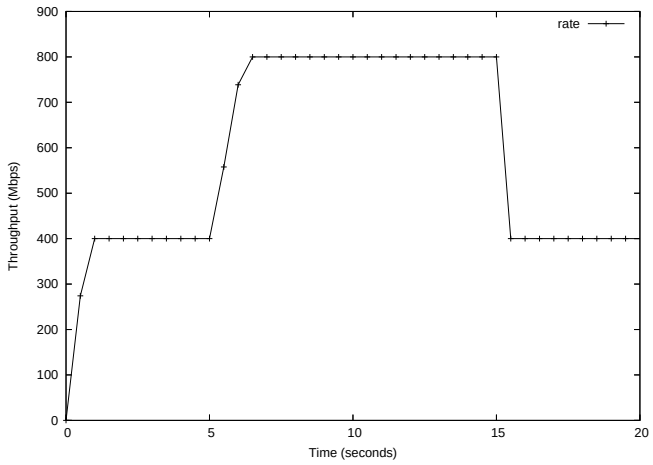


Figure VII.3: Trajectory of a single session with changing session desired rate

peak rate. This confirms with the design goal of setting parameter *beta* to control the "last-minute" speed.

Figure VII.2(c) depicts rate trajectories under different RTTs between the source and sink with fixed control interval. The convergence time goes longer when RTT is increasing. And in Figure VII.2(d) we held the RTT constant but varied the control interval. The figure shows that the rate will converge to the peak rate regardless of the value of the control intervals. A shorter control interval yields a faster convergence.

For all four of the scenarios, the single session experiences a "start-up" period, which reflects the rate-adaptation stage in the proposed algorithm. In next Chapter, we will discuss how this compares with TCP and how the "start-up" period can be improved.

In the last scenario of point-to-point experiments, we simulate the change in session demand. The link capacity remains 1 Gbps. The initial session desired rate is set to 400 Mbps. It changes to 800 Mbps and back to 400 Mbps at time $t = 4s$ and $t = 8s$, respectively. All other parameters are set with default values. Figure VII.3 depicts the trajectory of the session rate, showing that the session rate is bounded by and adapts to the session demand.

VII.3 Multiple Sources or Sinks

In this section we study the multipoint-to-point and point-to-multipoint scenarios.

As shown in Figure VII.4, there are 5 sources, each with a session terminating at a single sink. We firstly let all sessions start at the same time. Similar to what we have done for the point-to-point case, we vary the values of α, β and RTT while keeping other parameters at default values. The trajectories of five sessions for each scenario are shown. We let five sessions start at the same time. and the trajectories of all five sessions are shown in Figure VII.5, Figure VII.6 and Figure VII.7. These trajectories demonstrate the convergence properties of our proposed approach. And the system converges for a mixed set of sessions having different RTTs and even algorithm parameter α and β. Sessions with shorter RTT responses to the state changes faster. Normally we use the same α and β parameters for all sessions to make sure the algorithm is fair for each session in the system and no session takes advantage of faster adaptation pace.

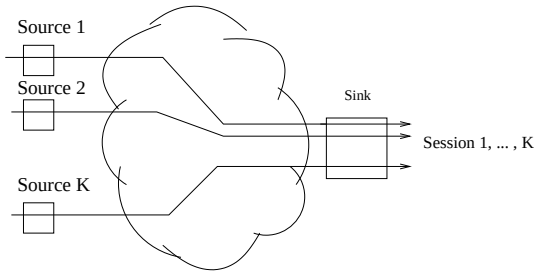


Figure VII.4: A single sink, multiple source Topology

We then change the start and end time of sessions with different RTTs. We add a new session join every 10 seconds. And when all sessions are present, one session terminates every 10 seconds. As we use sessions with different RTTs (100ms, 75ms, 50ms, 25ms and 1ms), two different sequences with RTT in descending and ascending orders are used. The rate trajectories of each session are shown in Figure and Figure VII.9. We observe that in either case, after each new session joins,

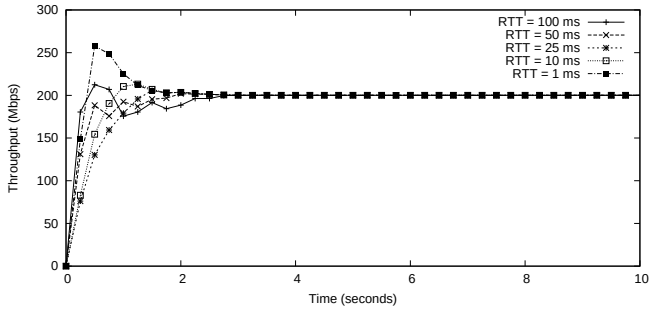


Figure VII.5: A multipoint-to-point case with different session RTTs

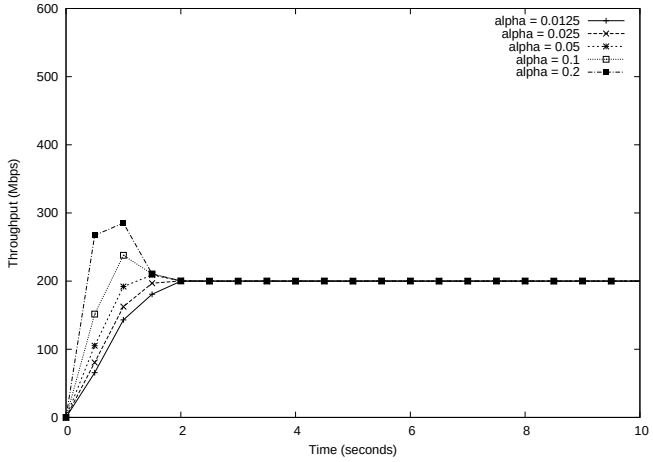


Figure VII.6: A multipoint-to-point case with different α's

the system converges to a new equilibrium state, in which all sessions get the same share of the sink bandwidth.

In this last scenario, we assign five sessions with different demand of 50 Mbps, 100 Mbps, 200 Mbps, 300 Mbps and 400 Mbps. The trajectories of five sessions are depicted in Figure VII.10, which shows that session rates converge to their desired rates and the session with highest demand (400 Mbps) get the rest of capacity share (350 Mbps).

Point-to-multipoint transfer (Figure VII.11) generally incurs the contention among sessions at the source. We conduct similar simulations as the multipoint-to-point case with varied values of α, β and RTTs as shown in Figure VII.12, VII.13,

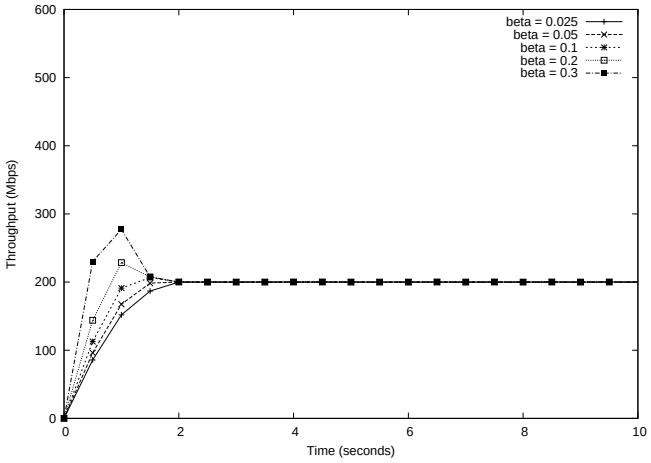


Figure VII.7: A multipoint-to-point case with different β's

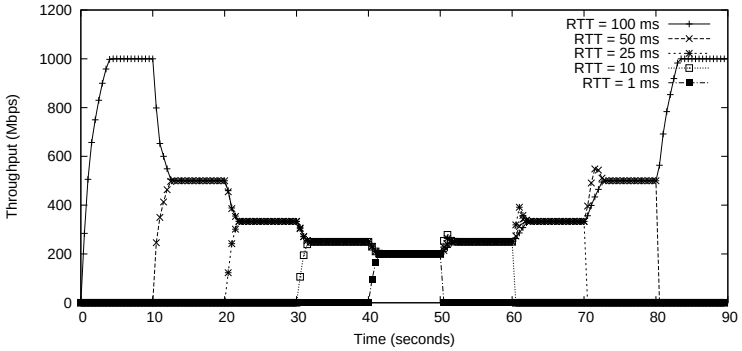


Figure VII.8: A five-to-one case: sessions with different RTTs join and leave one by one, in RTT descending order.

VII.14. In all the cases, all sessions converge to a max-min fair rate allocation.

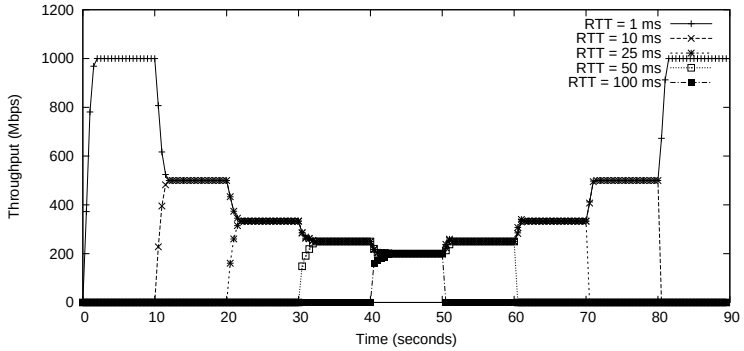


Figure VII.9: A five-to-one case: sessions with different RTTs join and leave one by one, in RTT ascending order.

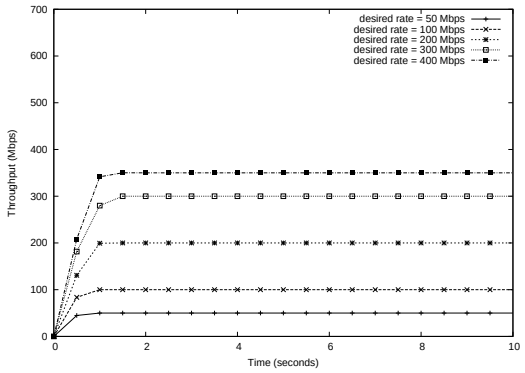


Figure VII.10: A five-to-one case: sessions with different desired rates of 50 Mbps, 100 Mbps, 200 Mbps, 300 Mbps and 400 Mbps

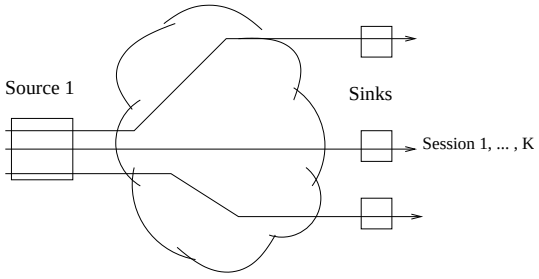


Figure VII.11: A single sink, multiple source Topology

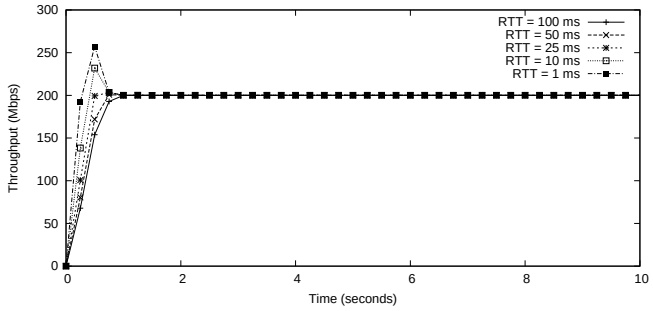


Figure VII.12: A one-to-five case: sessions with different RTTs of 100 ms, 50 ms, 25 ms, 10 ms and 1 ms.

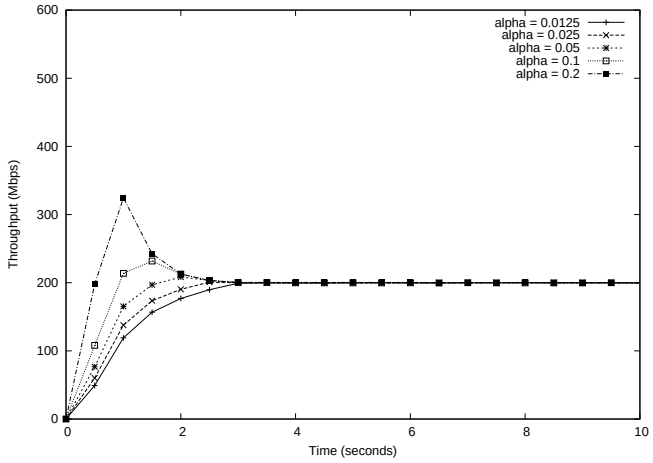


Figure VII.13: A one-to-five case: sessions with different *alpha*'s

VII.4 A Four-to-four Case

We now show the rate trajectories and convergence results when there are four sources and four sinks. We setup one session between each source and sink nodes, for 16 sessions in total. The sources and sinks have the same capacity, which is 500Mbps, and normalized to 1. The initial rate of each session is 0. The RTT is randomly generated with uniform distribution between 1ms and 100ms. The control interval of each node varies between 10ms and 100ms. Step size parameters α is equal to 0.1. The trajectories of each sessions are depicted in Figure VII.15, with

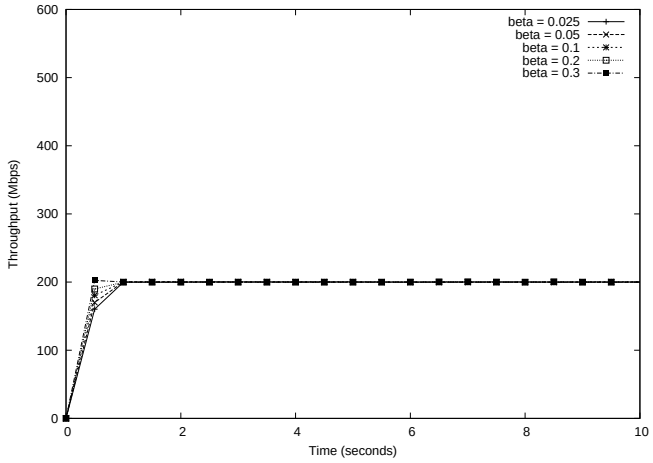


Figure VII.14: A one-to-five case: sessions with different β's

two different values of parameter $\beta = 0.2$ and 0.05. We see that the rates of all 16 sessions converge to 0.25, and a smaller parameter β leads to longer time for the system to converge.

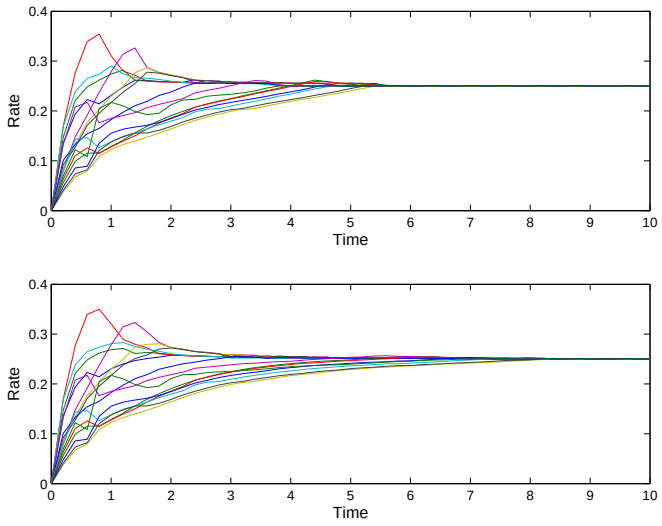


Figure VII.15: The trajectories of 16 sessions in a 4-to-4 case. Top: Step size $\beta = 0.2$; Bottom: Step size $\beta = 0.05$.

Table VII.2: Value ranges for different parameters

Parameter	Value Range
α	0.05 - 0.15
β	0.05 - 0.15
RTT	1 - 100 ms
Control Interval	10 ms - 100 ms

VII.5 Larger Networks

We now consider network sizes of 32, 128, and 1024 end nodes. Each end node has the same capacity, which is normalized to 1. For session rate behavior, we used to plot session rates for particular flows versus time. However, this visualization quickly becomes impractical to show individual session rates for large numbers of flows. Instead, we use the session distance vector defined previously, as:

$$D(t) = \Big[\sum_{i=1}^{K}(x_i(t) - x_i^*)^2\Big]^{1/2}.$$

The trajectory of this rate distance vector reflects whether session rates are converging or departing from the equilibrium state at any time.

In each scenario, half of the nodes are sources and the other half are sinks. We let each source participate in 4 sessions with randomly picked sinks. We generate 30 test cases for each of the three scenarios with randomly picked parameters following uniform distributions, as shown in Table VII.2

Figure VII.16, VII.17 and VII.18 plot the 2-norm distance between current rates and the equilibrium rates over time of the 30 testing cases for the three scenarios of 32, 128 and 1024 nodes. To show the impact of randomly generated round-trip time, in each figure we also plot three 2-norm distance trajectories in synchronous cases in which each session has fixed round-trip time of 1ms, 50ms, and 100ms.

From all three figures we see that the distributed algorithm causes the 2-norm distance to decrease quickly from initial states, reaching the equilibrium in no more than 6 seconds. The 2-norm distance is also bounded between two synchronous

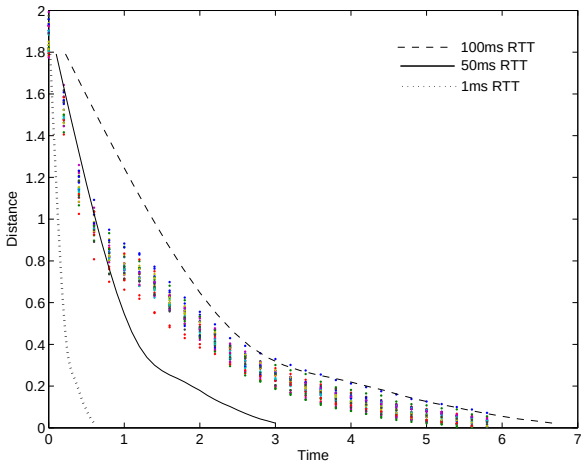


Figure VII.16: The 32-node network case: a comparison between the 2-norm distances of 30 test cases in the asynchronous case and the 2-norm distance trajectories in the synchronous case with various time slot size (1ms, 50ms and 100ms).

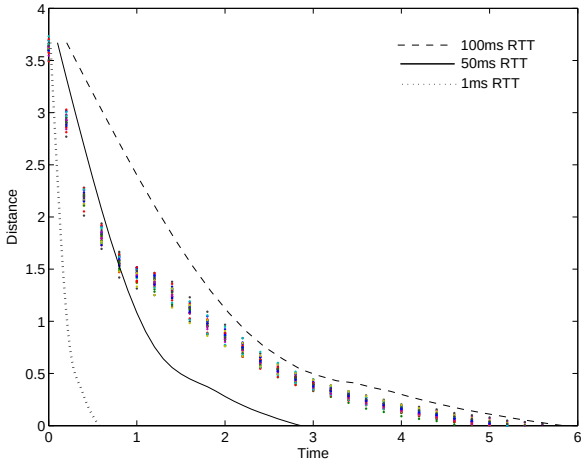


Figure VII.17: The 128-node network case: a comparison between the 2-norm distances of 30 test cases in the asynchronous case and the 2-norm distance trajectories in the synchronous case with various time slot size (1ms, 50ms and 100ms)

cases with minimum (1ms) RTT and maximum (100ms) RTT. Figure VII.19 shows the trajectories of all 64 sessions in the case with 32 nodes.

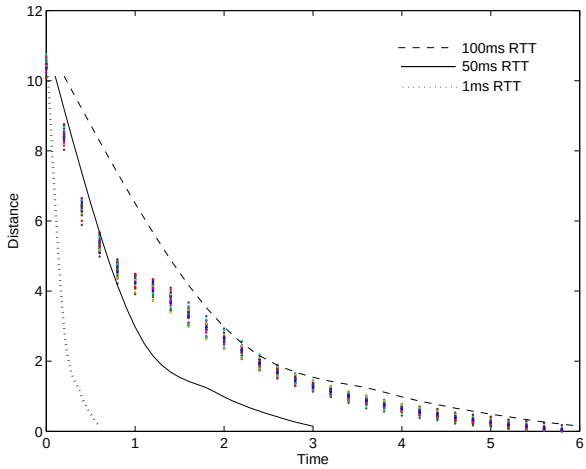


Figure VII.18: The 1024-node network case: a comparison between the 2-norm distances of 30 test cases in the asynchronous case and the 2-norm distance trajectories in the synchronous case with various time slot size (1ms, 50ms and 100ms)

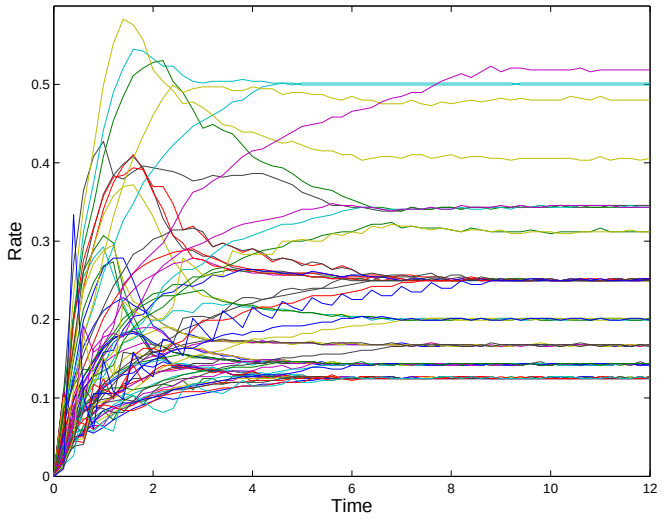


Figure VII.19: The trajectories of 64 sessions in an asynchronous 32-node network case

VII.6 Summary

Through simulation experiments, we have validated the convergence and stability properties of our distributed end-node bandwidth-sharing algorithm. We

have shown that the system converges, with various traffic patterns, protocol parameters, numbers of end nodes, numbers of sessions, and RTTs. These experimental results provide strong evidence that our distributed end-node bandwidth-sharing algorithm makes session rates converge from any initial or transitional states.

VIII

Comparison with Other Protocols

In this chapter we describe the simulation studies that we conducted to compare our approach with those that target high-performance data transfers. We first compare the behavior of our approach, referred as GTP (Group Transport Protocol), with traditional TCP and study how GTP and TCP could co-exist. We then compare GTP with high-speed TCP variants and investigate whether existing router-based schemes can be used to solve the bandwidth-sharing problem in lambda networks.

VIII.1 Comparison and Interaction with TCP

TCP is the standard transport protocol widely used for the traditional Internet. In using our distributed end-node-based bandwidth-sharing algorithm (GTP) to provide transport-level service, we must address the coexistence of GTP and TCP. This coexistence is challenging for two reasons. First, because TCP is the default transport protocol, most distributed applications and services are based on TCP. When GTP is running in lambda networks, there may be some TCP background traffic. Second, GTP targets only long-lived bulk data transfer. For small message-passing and streaming-data services that do not require high transmission rates, TCP may still be the best option. Therefore, it is study the

coexistence of GTP and TCP from several different aspects. More specifically, we study two important questions:

- If there is only one session in a point-to-point scenario, what is the performance difference between GTP and TCP? TCP uses a loss- and per-flow-based rate-control scheme, while GTP performs rate allocation across sessions, so it is important to demonstrate how GTP's rate adaptation compares with that of TCP's AIMD scheme in a point-to-point scenario.

- Given that GTP genearlly behaves more aggressively than does TCP, is it possible for GTP and TCP to coexist? Since GTP targets only bulk date transfer and manages long-lived sessions, applications may continue to use TCP for small message passing and other purposes. Therefore it is important to come up with a scheme that will permit GTP and TCP to coexist.

We explore the answers to these two questions in the following two sub-sections.

VIII.1.A Comparison with TCP: The Single-Session Case

We first compare the start-up behavior of GTP and TCP in the single-link case (Figure VII.1). The RTT is 60 ms and the default values of $\alpha = 0.1$ and $\beta = 0.2$ are used for GTP. Figure VIII.1 shows that TCP begins slowly, but quickly reaches full capacity. This conforms with the improved start-up phase, during which the observed rate increases exponentially when there is no packet loss on the link. In contrast, GTP's increase in speed is determined by the parameter α, and the incremental increase is defined as $\alpha \cdot \Delta r$, where Δr is the difference between the link capacity and the current rate. This linear adaptation scheme leads to a faster rate increase initially, but a slower rate increase as the rate approaches the convergence rate. This prevents severe rate oscillations when the system is close to the convergence state.

The slower start-up speed is not a serious concern, because we are more interested in achieving long-term fairness and convergence. However, several measures could be undertaken to improve GTP's start-up behavior in the single-session scenario:

- Adopt a greater linear adaptation rate (α) in the initial stage. Figure VIII.2 shows that larger values of α yield faster rate increases.
- Use schemes similar to those used in TCP, exponentially increasing the rate to a certain threshold and then reverting to the original linear adaptation schemes.

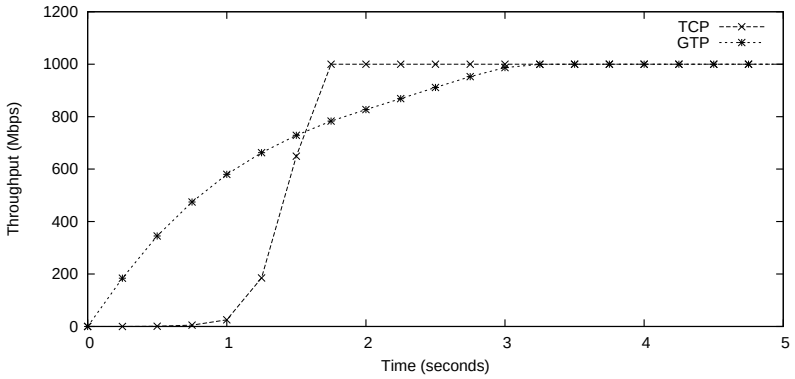


Figure VIII.1: The start-up behavior of TCP and GTP

With multiple sessions, the start-up behavior can be improved by directly assigning each session an initial rate that is equal to the fair share of bandwidth that each session receives.

The experiment assumes a clean path without any background traffic or random link loss. When some random link loss occurs, the performance of traditional TCP will suffer from a slow AIMD recovery phase in high-bandwidth, long-delay networks. This is illustrated in Figure VIII.3, which shows that TCP requires significant time to recover whenever packet loss occurs on a lossy link. In contrast,

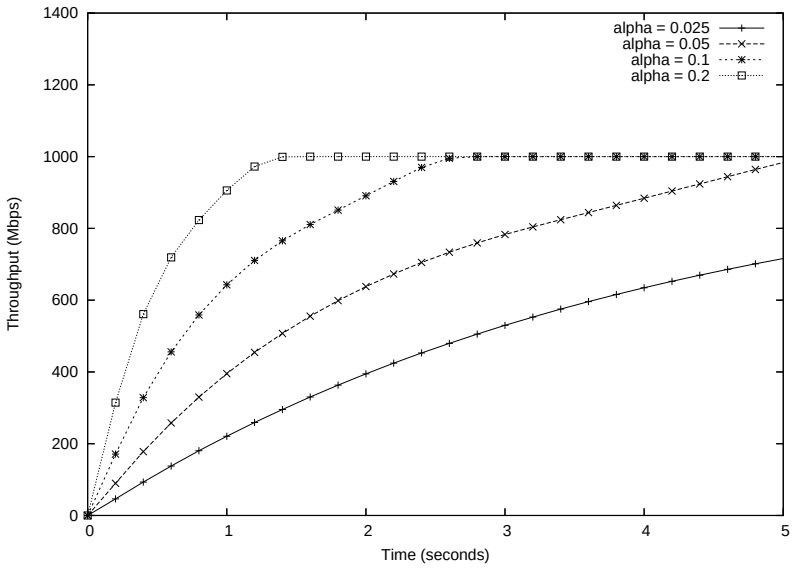


Figure VIII.2: The Start-Up Behavior of GTP Under Various α Values

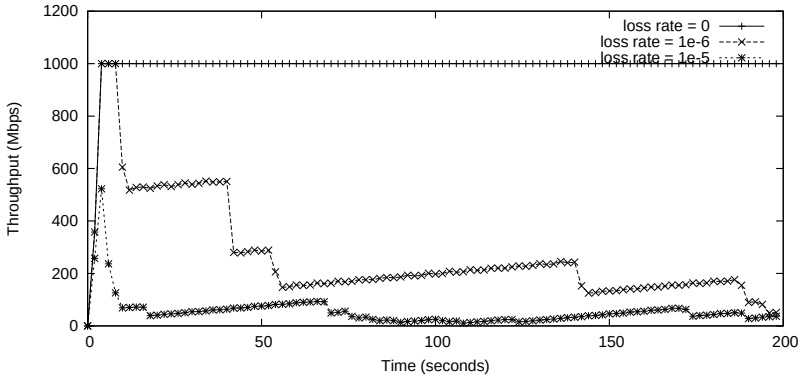


Figure VIII.3: Trajectories of single TCP session under different link loss ratio

GTP does not use packet loss as a control signal and therefore is not influenced by random packet loss. Thus, the GTP session rate is maintained at full capacity.

We discuss scenarios with multiple sessions in Section VIII.2, in which we compare GTP with other TCP variants. Our simulation results show that when there are multiple sessions, TCP is unable to achieve high efficiency and is unfair to sessions with different RTTs. In contrast, GTP achieves both efficiency and fairness across sessions.

VIII.1.B Interaction with TCP

Although GTP behaves more efficiently than does TCP in a lambda network environment, TCP may nevertheless be used by many applications in lambda networks as previous discussed. Therefore, it is necessary to understand how GTP interacts with TCP traffic. We also need to study whether GTP impacts TCP's performance and how the two protocol can co-exist.

We first consider a simple case, in which one TCP session shares a link with a GTP session. The trajectories are depicted in Figure VIII.4. Both sessions start at time $t = 0$ s. The trajectories show that TCP traffic has an exponential start-up, but then quickly drops to zero. In this scenario, the GTP sharing scheme is set to a full capacity of 1000 Mbps. As shown in Figure VIII.4, GTP traffic is not significantly affected by TCP traffic and it occupies all of the bandwidth share.

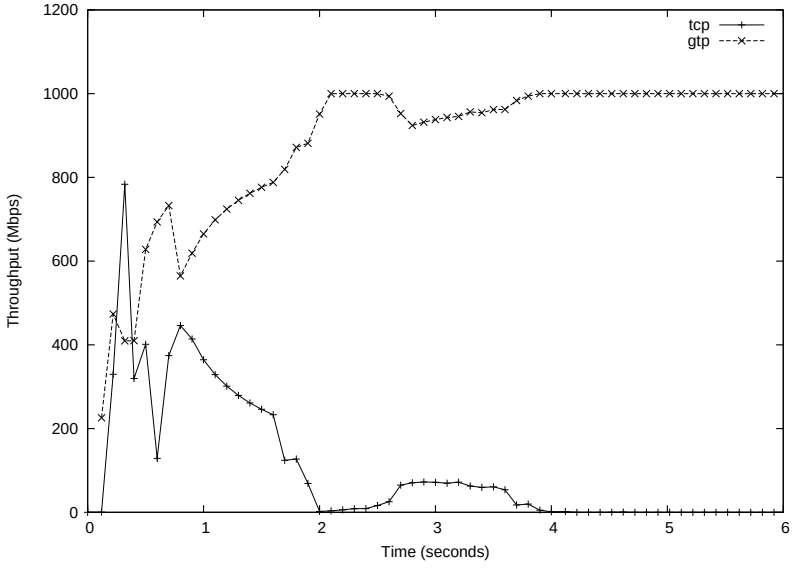


Figure VIII.4: Trajectories of single TCP session and single GTP session. GTP session completes at t = 6s

The example described in Figure VIII.4 shows that the rate-based sharing in GTP is unfair to TCP traffic. This unfairness results from different traffic-sharing and design objectives. Our proposed approach deals more with the scenario in which

a limited number of long-life sessions share network and end-node capacities, whereas TCP tries to manage thousands of sessions sharing a network path with multiple traffic sizes.

We can take several approaches to improve the unfairness of GTP to TCP. First, we can reserve a bandwidth share for TCP by adjusting the total capacity available to GTP at each end node. To that end, we repeat the experiment described above, but this time we lower the total GTP capacity to 800 Mbps. The resulting trajectories are redrawn in Figure VIII.5. In this scenario, the TCP session operates within the remaining capacity of 200 Mbps. Note that in all the cases, the RTT is set to a small value of 4 ms and the control interval is 20 ms.

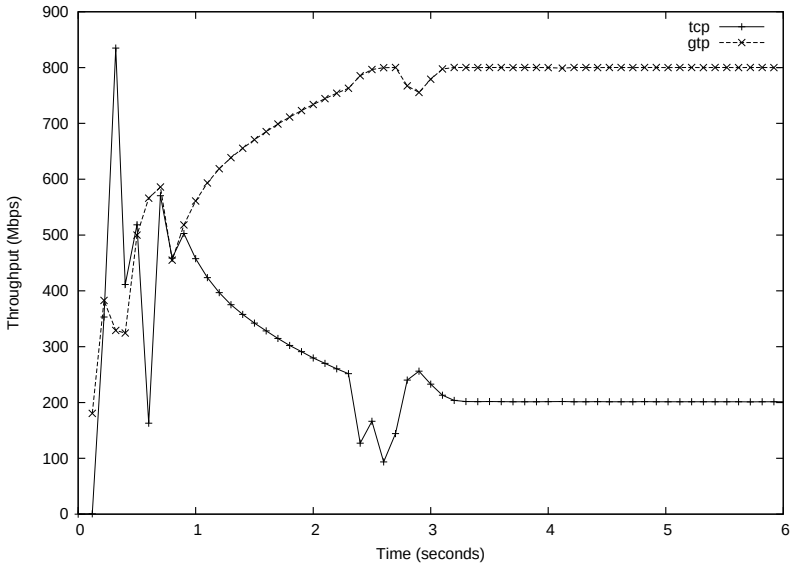


Figure VIII.5: Trajectories of single TCP session and single GTP session with 80% capacity allocated

Two additional examples further illustrate this sharing feature. In Figure VIII.6, the number of TCP sessions is increased to five. As expected, all five TCP sessions share the bandwidth that is unallocated to the GTP session. In Figure VIII.7, we increase the number of GTP sessions to two. the number of GTP sessions is increased to two. Each GTP session has a bandwidth share of 400 Mbps

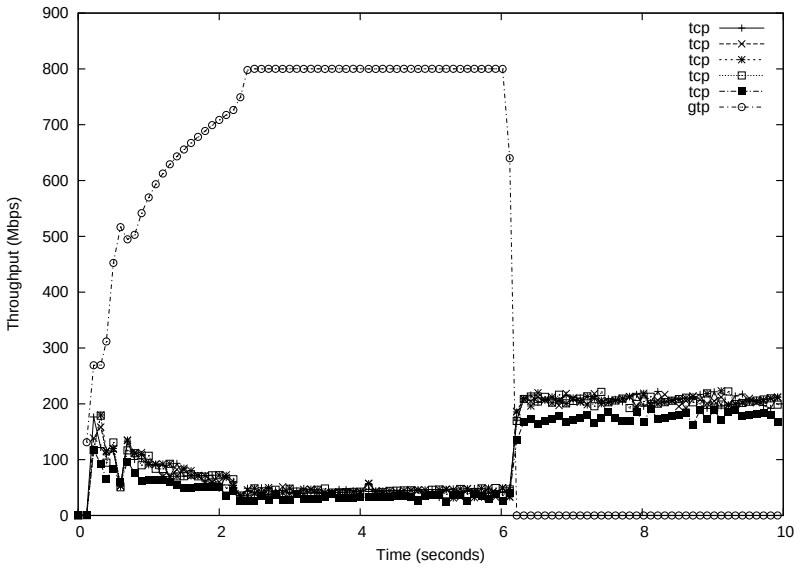


Figure VIII.6: Trajectories of five TCP sessions and single GTP session with 80% capacity allocated

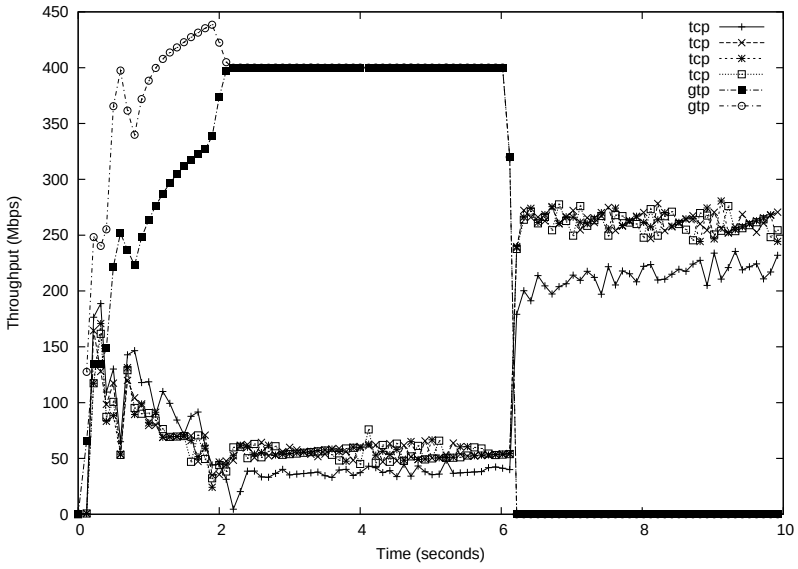


Figure VIII.7: Trajectories of four TCP sessions and two GTP sessions with 80% capacity allocated to GTP

and the four TCP sessions share the remaining bandwidth resource equally.

The second solution to correct the unfairness is to dynamically monitor the number of active sessions and the current aggregate rate of the TCP sessions and adjust the GTP capacity accordingly. Ignoring the exponential start-up phase, TCP behaves in accordance with AIMD. This makes feasible the estimation of TCP traffic, and the dynamic, adaptative approach gives TCP traffic a higher priority. This may help to maximize capacity utilization, especially when TCP traffic remains low. However, the stability properties that were proven previously may not hold at these low levels of TCP traffic and system oscillations may occur due to the unstable nature of TCP traffic.

In conclusion, experiments in this section show that GTP outperforms TCP in the long run, in terms of (1) being fair to sessions with different RTTs and (2) tolerating packet loss. Because GTP is more aggressive than TCP, we must take measures to ensure that they can coexist.

VIII.2 Comparison with High-Speed Variants

As we have shown in the previous section, GTP yields better efficiency and fairness in lambda networks than does TCP. However, there are many high-speed TCP variants and other protocols that are entirely distinct from TCP. Each protocol has different design goals and targets different models of network use. These other protocols may not have the same goals as does our proposed approach, but we nevertheless wish to briefly compare GTP with several high-speed variants to answer the follow questions:

- How does GTP compare with TCP alternatives that have more aggressive AIMD control schemes? Such schemes usually share a design goal with TCP, in that they aim to provide reliable and scalable service for general Internet usage.

- How does GTP compare with router- or switch-assisted schemes? Because the

Table VIII.1: Comparison among protocols

Protocol	Target Network	Design Goals
TCP Reno	Internet	Avoid packet loss
HSTCP, HTCP	Internet	Avoid packet loss; Improve Efficiency
XCP	Router-Assisted	Efficiency and Fairness
GTP	Lambda networks	Efficiency and Fairness

functions of routers and packet switches are similar to those of the end nodes in GTP, we want to investigate whether such router- or switch-based schemes can be directly applied to end nodes.

In this section, we compare our approach with a modified end-point-only version of XCP (endpointXCP) [58], which moves the XCP router functions into the receiver (endpointXCP), and with two TCP variants, TCP NewReno and Highspeed TCP [39] with SACK1. We investigate their efficiency and their properties of convergence, stability, and fairness under various network configurations (multipoint-to-point and multipoint-to-multipoint) and session demands (desired peak rates). We summarize the information regarding protocols in Table VIII.1. The default parameter values of XCP remain unchanged in endpointXCP. Highspeed TCP is configured with a *window_option* of 8, and the parameter *max_ssthresh* is set to 50.

We examine the behavior of these protocols under multipoint-to-point and multipoint-to-multipoint traffic patterns. We also vary the following experiment parameters.

- Session RTTs. We assign a range of RTTs to different sessions to investigate whether the fairness across sessions is ensured.
- Session demand. We assign different session demands to investigate whether these protocols can saturate sessions with low demands and let sessions with high demands acquire the remaining capacity.

- Number of concurrent sessions. We vary the number of concurrent sessions to investigate the relationship between the aggregate utilization and the number of sessions.

- Packet loss. We introduce packet loss to study the resulting transport performance of these protocols.

In the rest of this section, we introduce the modified version of XCP and then present the experimental results of different scenarios.

VIII.2.A End-Node-Based XCP

The Explicit Control Protocol (XCP) is a novel transport protocol that decouples efficiency control from fairness control and makes each router periodically allocate its capacity to traffic sources. XCP generalizes traditional Explicit Congestion Notification (ECN) [69] by placing multiple bits in the information that is fed back to the traffic sources. The efficiency controller in XCP uses MIMD to calculate the desired change in the aggregate traffic rate in a control interval, based on the persistent queue length and the available bandwidth. MIMD assists in achieving fast convergence and high utilization. The fairness controller uses AIMD and "capacity shuffl" to compute the desired increase or decrease in the aggregate traffic rate for each individual session, which guarantees constrained max-min fairness [62]. Unlike many other schemes for router- and/or switch-based rate allocation, XCP gives different control feedbacks to each session, thereby avoiding the severe overflow that was described previously.

Unmodified XCP is not directly applicable in lambda networks, which often lack routers (much less XCP-enabled routers). To make a fair comparison, we modify XCP, creating endpointXCP which moves the router functions to the sources and sinks. Each end node is aware of its capacity and that of all the associated sessions, and conducts the same rate allocation, adaptation, and feedback as those defined for the XCP router module. This allows us to investigate whether an XCP-like scheme

can solve the bandwidth-sharing problem in lambda networks. To distinguish this modified scheme from the original XCP, we refer to it as endpointXCP.

VIII.2.B Experimental Results: Multipoint-to-Point Traffic

We first study a simple scenario with five sources and one sink node. The five sessions have RTTs that vary from 10 ms to 90 ms. The capacity of each source and sink node is 500 Mbps. We let each session have an infinite desired rate, and the resulting session trajectories for different protocols are depicted in Figure VIII.8. EndpointXCP and GTP are able to achieve full utilization and fairness within 10 seconds, much faster than the TCP variants. EndpointXCP has the fastest rate to convergence.

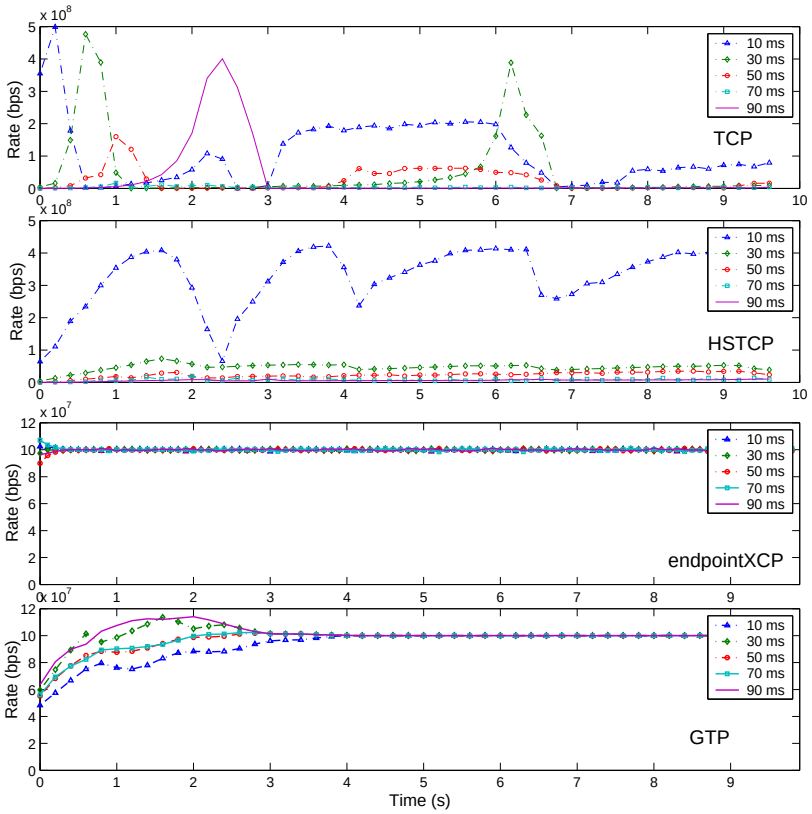


Figure VIII.8: The 5-to-1 Case: Five sessions have different round-trip times: 10 ms, 30 ms, 50 ms, 70 ms, 90 ms. The sink capacity is 500 Mbps.

We then consider a scenario in which the sessions have different desired rates. We let four of the five flows have the same low desired rate of 25 Mbps and 10 ms RTT. The fifth session's desired rate is 500 Mbps with an RTT of 50 ms. In Figure VIII.9, we observe that only endpointXCP and GTP achieve high throughput in the first 10 seconds, and that only the GTP sessions achieve 100% link utilization. The fifth endpointXCP session cannot fully utilize the remaining bandwidth.

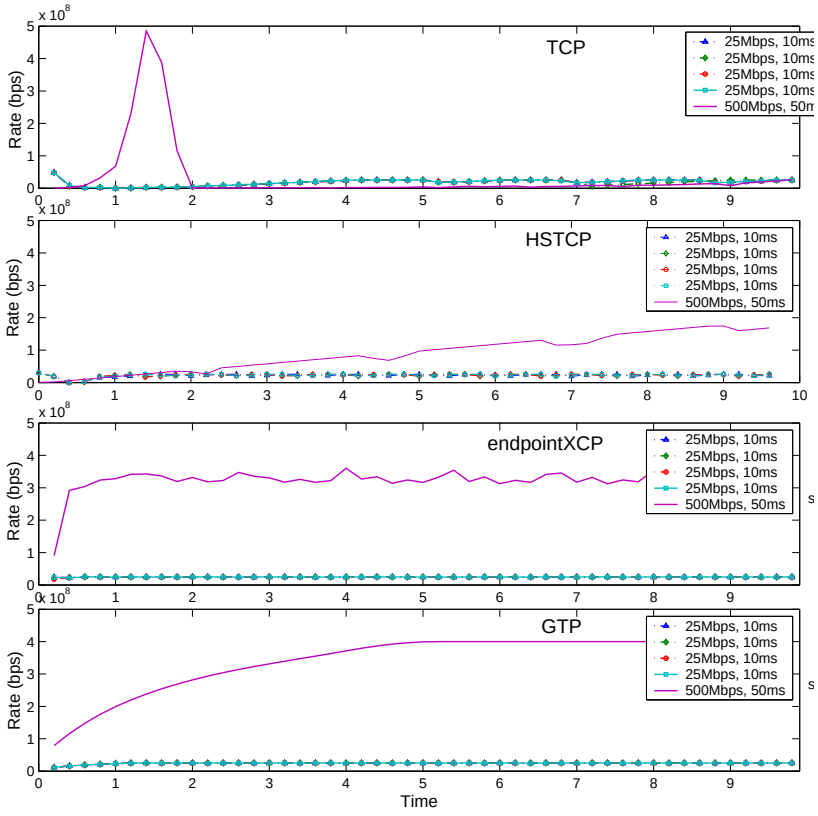


Figure VIII.9: The 5-to-1 Case: Five sessions have different round-trip times: 10 ms, 10 ms, 10 ms, 10 ms, 50 ms. Four sessions have low desired (peak) rates of 25 Mbps. The fifth session has a desired rate of 500 Mbps. The sink capacity is 500 Mbps.

To further illustrate this difference in bandwidth-utilization at convergence between GTP and endpointXCP, we construct a scenario in which one session with a high desired rate shares a single end node with low-desired-rate peer sessions.

The aggregate desired rate from the peer sessions that have low desired rates is equal to half of the sink capacity. Figure VIII.10 shows the remaining link-capacity utilization of the high-desired-rate session, demonstrating that the utilization ratio of the endpointXCP session declines as the number of low-desired-rate peer sessions increases. In other words, endpointXCP becomes less efficient. This is due to the "bandwidth-shuffle" effect of XCP, which underlies how XCP normally achieves fairness for sessions with different RTTs. We refer to [62] for formal studies on the fairness constraints of XCP. Unlike XCP, GTP can optimize capacity utilization over multiple peer sessions.

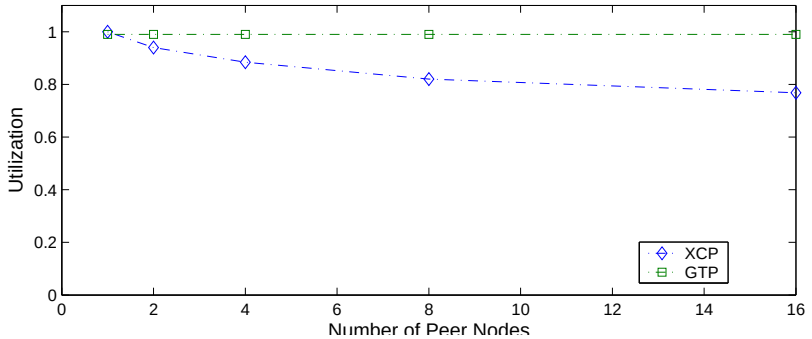


Figure VIII.10: Comparison between endpointXCP and GTP: The link utilization of one "fat" session while sharing with different number of "thin" peer sessions.

Before considering multipoint-to-multipoint scenarios, we study how packet loss could impact the performance of XCP. We run the same 5-to-1 experiment, but with uniformly random packet loss on the access link of two senders. The loss ratio is set to 0.025%. The session trajectories are depicted in Figure VIII.11 and VIII.12 for XCP and GTP, respectively. From Figure VIII.11, we observe that XCP traffic has a very low average transmission rate on the lossy link . Although the control signal fed back is based on queue size and aggregate rate, each XCP sender nevertheless adjusts its rate based on detected packet loss. In contrast, GTP does not consider the packet loss ratio in its rate adjustments and therefore can achieve almost the same convergence properties as when there is no random link loss. Although GTP

is primarily a bandwidth-sharing scheme, it can also be extended to adapt to severe packet loss, as we discuss in Chapter X.

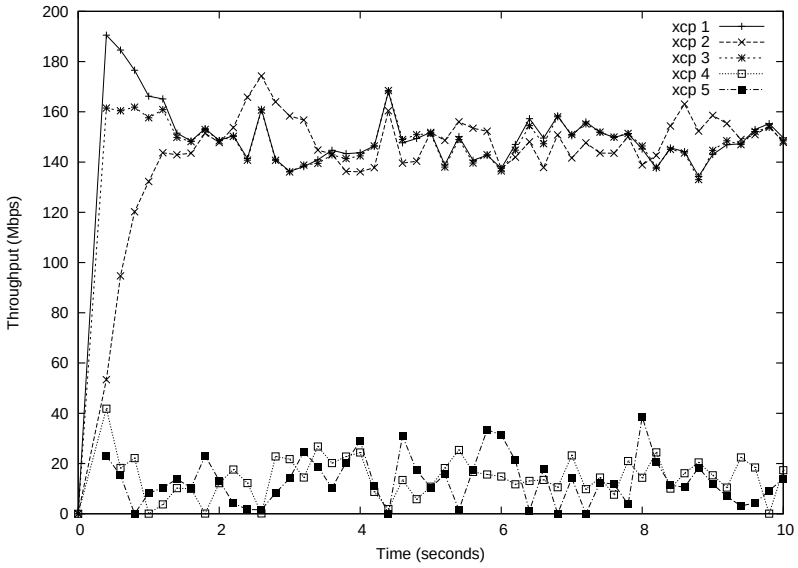


Figure VIII.11: The Trajectories of five endpointXCP sessions in the case of 5-to-1 transfer with 0.025% random link loss.

VIII.2.C Experimental Results: Many-Way Traffic (Network)

We have evaluated the convergence properties of endpointXCP and GTP over a range of network sizes and traffic patterns. Now we show how endpointXCP and GTP behave in a 16-node lambda network. We let 8 nodes be traffic sources and 8 nodes be sinks. Each source node has 4 active sessions with randomly selected sink nodes, as shown in Table VIII.2. The RTT for each session is randomly generated between 1 ms and 100 ms, and the capacity of each node is 500 Mbps. We assume infinite desired session rates in this scenario. This setup is similar to that required for data visualization from data repositories at different locations. And, because the endpoints are directly connected with bottlenecks at the end nodes, this setup resembles a lambda network.

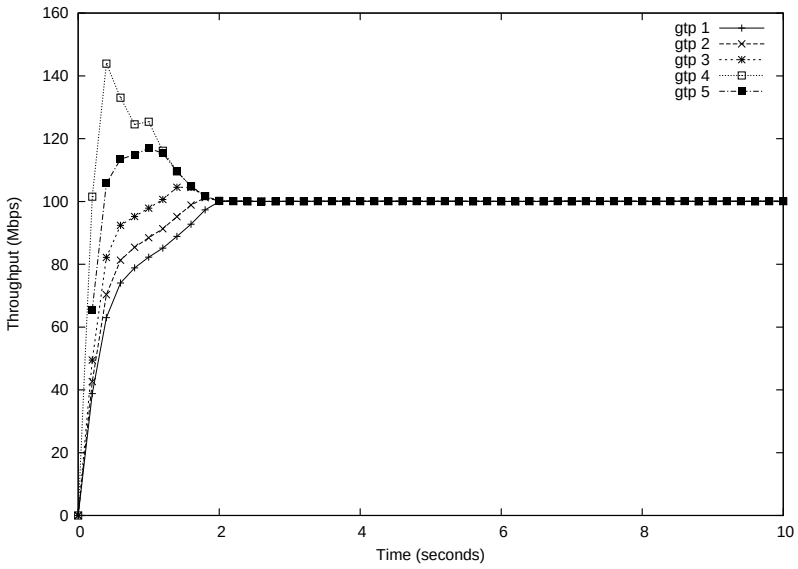


Figure VIII.12: The trajectories of five GTP sessions in the case of 5-to-1 transfer with 0.025% random link loss.

Table VIII.2: Connections between sources and sinks

Source	Sink	Source	Sink
1	1, 2, 2, 7	5	1, 3, 4, 5
2	1, 2, 3, 4	6	2, 2, 3, 4
3	2, 3, 4, 4	7	2, 5, 8, 8
4	3, 4, 4, 8	8	1, 4, 5, 6

Figure VIII.13 shows the trajectories of endpointXCP and GTP for all 32 sessions. We see that both endpointXCP and GTP sessions converge to steady states. However, because some of the endpointXCP sessions do not reach optimum rates, the aggregate rate for endpointXCP sessions is 3.41 Gbps, compared with the max-min fair allocation of 3.50 Gbps in GTP sessions. The four highest sessions rates generated by endpointXCP and GTP are listed in Table VIII.3, showing that endpointXCP may not achieve max-min fairness in certain complex, networked scenarios, even when all the desired session rates are infinite.

In summary, pushing the XCP control algorithm from routers to endpoints

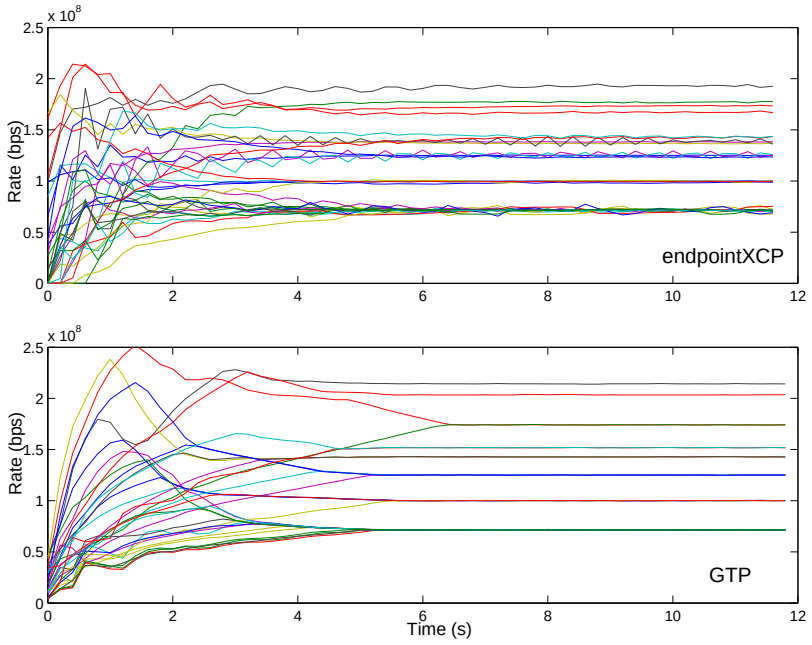


Figure VIII.13: The trajectories of 32 sessions of endpointXCP and GTP in a randomly generated asynchronous 16-node network case. The RTT between each source and sink is randomly distributed between 1ms and 100ms.

Table VIII.3: Comparison of XCP and GTP: Four Highest Sessions Rates

Session	rate of XCP (Mbps)	rate of GTP (Mbps)
1	192.6	214.2
2	177.7	203.5
3	173.9	174.0
4	166.9	174.0

cannot achieve the high efficiency and fairness of GTP. In the multipoint-to-point scenario, we have shown that endpointXCP's efficiency is affected by setting session desired rates. We further showed that even without the limit of session desired rates, endpointXCP is unable to reach the max-min fair rate allocation under a randomly generated network topology and traffic pattern. We have also shown that because XCP uses a queue-based control scheme, it is affected by any packet loss in the network. This becomes a concern if the random link loss ratio is high due to background traffic or random physical media loss. We also want to point out that there are problems in implementing XCP's control scheme at end points. The measured length of the queue in real time may be affected by other processes that are running in the system and require kernel-level implementation. In contrast, GTP, which is based on our distributed end-node bandwidth-sharing algorithm, is able to deliver higher efficiency and better fairness.

VIII.3 Summary

We have compared the behavior of GTP and TCP under different network topologies and traffic patterns. Our results show that GTP performs better in terms of convergence and fairness across sessions. We have also studied the problem of GTP and TCP coexistence and proposed possible solutions. We compared the convergence and fairness properties of GTP with those of high-speed TCP variants and endpointXCP under multipoint-to-point and multipoint-to-multipoint scenarios, showing that GTP is the only protocol that can deliver both high efficiency and fairness in all cases.

Because GTP is mainly used for bandwidth sharing among long-lived sessions, we have compared it against only a small set of representative transport protocols that serve general Internet traffic. The results are encouraging and have two important implications. First, experimental results show that because different protocols target different usage scenarios, protocols for the general Internet are

not able to solve the fair bandwidth-sharing problem that we study in this thesis. Second, we are not able to apply router-assisted approaches directly to the lambda networks we study, because endpointXCP behavior becomes unpredictable and the performance becomes sub-optimal and unfair. In the coming sections, we further prove that GTP is a feasible solution that meets our design goals and the transport requirements for e-science applications in lambda networks.

IX

Prototype Design and Empirical Studies

In this chapter, we describe the prototype implementation of our proposed bandwidth-sharing algorithm. Building upon the simulation results described in Chapters VII and VIII, we focus on addressing the performance issues and then present the performance measurements obtained in a LAN environment with an Dummynet-emulated [71] delay and in a WAN environment on the OptIPuter networks. We show that measurements of the prototype's performance conform with our theoretical analysis and simulated results. This suggests that the proposed algorithm is feasible and that the theoretical analysis and simulations accurately model the behavior of the algorithm.

IX.1 Prototype Design

Any prototype design must meet infrastructure requirements. Our prototype had to be deployed easily across the OptIPuter networks for testing and demonstration purposes. Because the OptIPuter clusters are managed in different administrative domains at multiple organizations, the prototype should be installed without root privilege. Therefore, a user-level design solution is necessary.

The prototype design also needs to consider algorithm-specific functionalities. As our distributed algorithm at each nodes needs to monitor and coordinate among active sessions. General per-flow-based protocol frameworks (e.g. DCCP [61]) can not be directly adopted.

The prototype design also needs to meet performance requirements. The performance impact of the end systems should be limited from a design perspective. When packets are sent at a very high speed, the control logic can be affected easily by a context switch between processes or other "normal" system routines. As a result, overhead in the protocol stack can become an issue of concern and it becomes critical to reduce the overhead associated with the protocol framework as much as possible without losing basic functionalities. Because GTP is a bandwidth-sharing scheme that targets only long-lived flows, our prototype design should optimize GTP performance and coincide with the simulated results as much as possible.

The following list presents several key design issues.

- The type of transfer model. TCP is a streaming-based protocol and UDP is a datagram-based one. One of our design decisions is to choose the right type of transfer model for GTP.

- The implementation of the control algorithm. In running an end-node control algorithm at the user level, we need to find an efficient way to implement the control algorithm so that it oversees the transfer progress of all active sessions and runs the rate allocation and control algorithm during each control interval.

- Loss retransmission at high speed. Loss retransmission is not the major focus of our studies of the distributed end-node bandwidth-sharing algorithm, but a real prototype should be able to efficiently conduct loss retransmission to support high-speed transfers.

- Supporting a large number of connections. Unlike simulations, the prototype implementation needs to consider how to handle of a large number of active session.

- End-system optimization. We must consider how to optimize the implementation and avoid protocol-handling overhead in order to make the prototype's performance coincide with the simulated results as much as possible.

The following sub-sections describe these prototype-design issues in greater detail.

IX.1.A UDP-Based Request-Response Transfer Model

Most unicast Internet traffic uses congestion-controlled TCP. UDP has been used primarily for short request-response transfers. Because one of our design requirements is easy prototype distribution, we adopt a user-level implementation solution based on UDP. UDP is lightweight, offers reliable transmission and stateful connections, and provides only basic check-sum functions without using TCP's three-way handshake. Just as many other protocol frameworks cite DCCP and UDT, we use UDP as our starting point.

GTP has two types of packets: data packets and control packets. These packets can be differentiated by the flag bit, which is the first bit in each packet. UDP is used for bulk data transfers and for the exchange of control information.

The prototype adopts a receiver-driven, response-request model to support bulk data transfers. Using this model, after the connection is established, the receiver (client) sends a data request to the sender (server) with offset and length information. The sink's expected rate is either fed back to the source poetically or pigged back with an acknowledgement(ack) or loss list (nak) packet.

IX.1.B Control Daemon and Protocol Framework

To support multi-flow management and enable efficient and fair utilization of the source and sink capacity, GTP runs a control daemon on each host to implement the distributed allocation and control algorithms. During each control interval, the daemon actively measures the progress of each associated session, estimates the

actual capacity for each flow, and then fairly allocates the available capacity across flows.

The daemon is a stand-alone, long-lived process. Communication between the daemon and each session is facilitated through shared memory. A memory space with a flow table is pre-allocated when the daemon begins. Each session writes its statistics, including status and the number of bytes read or written, to the flow table. Each session also reads the calculated expected rate from the shared memory space. The sink feeds the rate information back to the source, which adjusts its own rate based on its demand, the expected rate from the sink, and its own expected rate.

Thus far, we have described the two levels of control (per-flow and per-host) at each end-node. The prototype architecture is shown in Figure IX.1

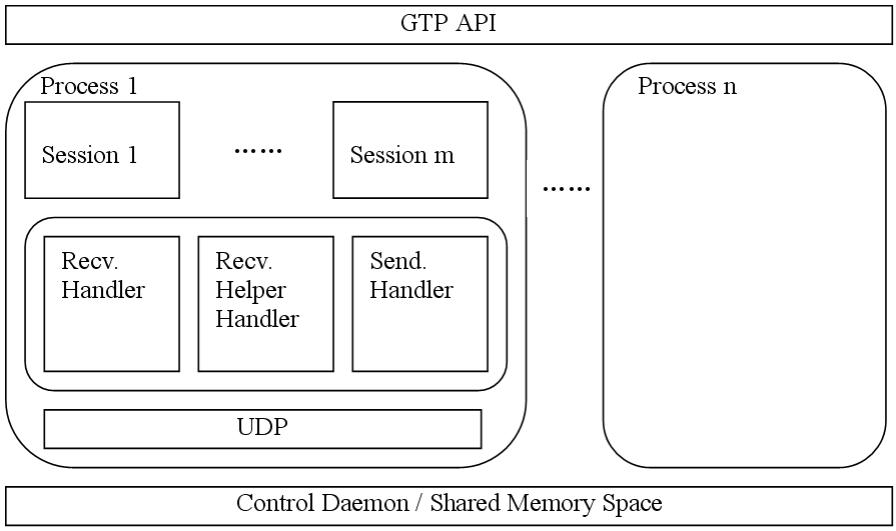


Figure IX.1: GTP prototype architecture

IX.1.C Loss Information Management

We add loss-retransmission functionality to our bandwidth-sharing scheme so that it can support real applications. Unlike TCP and its variants, which fre-

quently send out ack and nack packets, our scheme sends a loss list only periodically, because we assume that the packet loss is rare in most cases.

To support loss information management, we assign a sequence number to each data packet. We use a bitmap with a starting sequence number and an offset parameter to represent the loss information. Each bit in the bitmap represents one physical packet. Therefore, a 1,000-byte bitmap can cover 12 MB of data. Because packet loss is infrequent, the bitmap always begins with the byte that contains the first packet loss and can effectively describe any burst loss in a 0.1-second period when the session is transferring at the rate of 1 Gbps.

As long as the starting location is provided, the assembly and disassembly of the bitmap is performed easily by directly copying the entire data chunk. To identify the next packet loss, we assume that bit "0" represents a lost packet. To identify the relative position of the first lost packet, we search for the first byte or integer that is not equal to 0xFF or 0xFFFF, and then run a filter.

The assembly and disassembly of the bitmap may be expensive in certain cases. To further improve the performance, loss information-gathering from the bitmap can be distributed evenly to reduce the overhead of each generation process, but at the expense of higher aggregated handling time.

IX.1.D Packet Demultiplexing With Per-Process Helper Thread

Context-switching overhead is expensive for multiple, concurrent connections. Even with only a single session, there are still opportunities for concurrency. For example, if the single session sleeps to wait for incoming packet or wait for the next packet sending time, system kernel may switch CPU process other I/O or system requests. The process-scheduling granularity is either 10 ms or 1 ms in the Linux kernel. To avoid frequent context-switching among the connections within a single process, we rely on user-level packet demultiplexing. For each process, three threads handle all GTP packets: a sending handler, a receiving handler, and a receiving helper. Each session has a unique session identification number. The

sending or receiving handles have control over all ports and handle packets accordingly. More specifically, the receiving handler listens to all ports for active sessions and, whenever a data packet arrives, the handler copies it to the appropriate user buffer, sets packet-loss bits if necessary, and notifies the application thread when all the requested data is available. When a control packet arrives, the receiving handler sends a pointer to the receiving-helper thread through an FIFO pipe. This ensures that the receiving speed is not affected by handling incoming loss lists or other control packets.

IX.1.E End-System Optimization

In this section we discuss several ways in which we optimize the prototype implementation.

Memory Copy Avoidance

For high-performance transport protocols and applications, copying data or buffers consumes a tremendous number of CPU cycles and thus is a significant component of performance overhead. When a normal application reads data from a file and sends it to the network through a user-level protocol, the data is copied at least five times, as follws.

- Data is copied from the disk to the file-cache buffer
- Data is copied from the file-cache buffer to the application buffer
- Data is copied from the application buffer to the protocol buffer
- Data is copied from the protocol buffer to the kernel-socket buffer
- Data is copied from the socket buffer to the network-adaptor buffer

Several techniques can reduce this copying activity. For example, one can use copy-on-write to eliminate the copy from the disk to the file-cache buffer. In

addition, remote, direct memory access (RDMA) can reduce memory overheads due to copying. This type of copy avoidance has been considered in the context of kernel-level, protocol-stack optimization (e.g., zero-copy TCP [33], fast sockets [73], and TCP optimization), but the issue is of much greater concern for user-level communication libraries in which buffer copying may take place not only within the protocol stack, but also at the boundaries between the protocol and the kernel, and between the application and the protocol.

Our protocol framework uses several techniques to avoid data copying. First, copying is avoided by passing only the pointers to data buffers. Second, we implement an IO-vector-based, packet-packing function to avoid data copying when multiple message attributes occupy a single physical packet. Although the data within a physical packet is still copied to a kernel buffer by the kernel UDP driver, all other copies on the sending side are eliminated. To reduce the number of copies on the more complex receiving side, we predict the offset of the next incoming packet by assuming that most packets arrive in order. If this prediction is correct, we can copy the incoming data in its proper position in the user-application buffer only when the new packet is a loss-transmission packet, or when packet loss has occurred in the previous transmission.

Busy Waiting with a High-Resolution Timer

When packets are transferred at a speed of 1 Gbps, a 1.5-KB packet needs to be sent every 12 microseconds. This means that the sending handle can rest for only 12 microseconds before waking up to send another packet. In this case, the normal system call, *usleep*, is not useful, because the Linux-kernel scheduling granularity is 1 or 10 ms. When the CPU is switched to another process, it requires at least 1 ms to wake up.

Using busy looping while continuing to check against the *gettimeofday* system call can avoid this problem. We use a high-resolution timer to query CPU clock sticks while performing busy waiting. This requires fewer CPU cycles than does

the *gettimeofday* system call and is known as calling *rdtsc*. In the x86 assembly language, the RDTSC instruction is a mnemonic for "read-time stamp counter". The instruction returns a 64-bit value in registers EDX:EAX that represents the tick-count from the processor reset. However, the RDTSC instruction cannot be used on machines with SMP architecture, because the process may be scheduled on any of the available CPUs at each scheduling interval, and the inconsistency in the CPU sticks reported by separate CPUs confuse the calling function.

Finding a means for user-level packet pacing without high CPU costs is an ongoing challenge. One possible improvement would be to increase the CPU scheduling frequency, which is 10 ms under Linux kernel version 2.4 and 1 ms for Linux kernel version 2.6. Another possible improvement would be to increase the packet burst size, perhaps by increasing the packet size, which is set by default at 1.5 KB. Increasing the packet size by a factor of n stretches the inter-packet delay by a factor of n . However, two factors limit the packet size. First, when the packet size is larger than the default IP-packet size, fragmentation in the kernel IP stack increases the per-packet handling overhead at the kernel level. Second, if any segment is missing, the entire packet is dropped at the UDP stack at the receiver. When there are concurrent sessions, the IP-packet sequence may be interrupted by packets from other sources. In this case, packet loss is reported, although the missing segments still arrive by following segments from other sources. Another improvement in user-level packet pacing would be to increase the sending burst size by permitting multiple packets to be sent simultaneously without waiting or sleeping, but this could lead to traffic burst, possibly causing congestion or inaccurate transmission-rate reports on both sides.

IX.1.F Application Programming Interface (API)

The application programming interface (API) is an important factor to consider when in prototype design. Normally programmers want to comply with the socket semantics to allow the application developer to migrate the programs

easily. Many novel, user-level transport protocols [?] have been proposed recently for grid applications. Despite their shared goal of supporting efficient data transfer, these protocols present diverse APIs and developers are often forced to implement applications with protocol-specific interfaces. As a component of the Distributed Virtual Computer (DVC) framework, GTP addresses this issue by using a unified Unix-socket-like API.

To distinguish this approach from the original socket-API, GTP adds a prefix *g_* to each socket call. For example, the socket creation call

```
int socket(int domain, int type, int protocol)
```

now becomes

```
int g_socket(int domain, int type, int protocol).
```

There are two methods for using GTP APIs. First, applications can link directly with GTP libraries and call GTP APIs with the prefix *g_* attached to each socket call. Second, Grid applications can use DVC or other socket wrappers to intercept socket calls and redirect them to call corresponding GTP socket calls.

GTP also can be integrated with the Globus-XIO [9] framework. The Globus-XIO framework integrates a variety of transport protocols and provides a set of uniform interfaces to the socket wrapper to perform the I/O and communication operations. For compatibility with Globus XIO, transport protocols are typically implemented with Globus-supplied auxiliary modules (e.g., hostname resolving, timer, thread management). This is useful for Grid applications running on top of Globus middleware, but it may not be the best solution for non-Globus applications.

IX.2 Experiments

In the following sections, we present some measured results based on the GTP implementation on various testbeds, including a Dummynet-emulated environ-

ment and the OptIPuter testbed. The results validate the convergence and fairness properties of our bandwidth-sharing scheme.

IX.3 Experiment Setup

To create an environment with a mixture of short and long links with a local cluster, we use Dummynet routers to emulate various RTT end-to-end delays. Dummynet is a FreeBSD-based system facility that controls traffic passing through the various network interfaces by applying bandwidth and queue-size limitations and simulating delays and losses. Figure IX.2 shows the configuration of a simple Dummynet environment in which two end nodes are in separate subnets (netmask 255.255.255.0) and are connected with a Dummynet router, where we can vary the link's delay, bandwidth, and random-loss properties. Dummynet allows us to emulate long-delay links on under local cluster settings. The major limitations of Dummynet is it's limited data forwarding speed and higher random loss ratio than real network links.

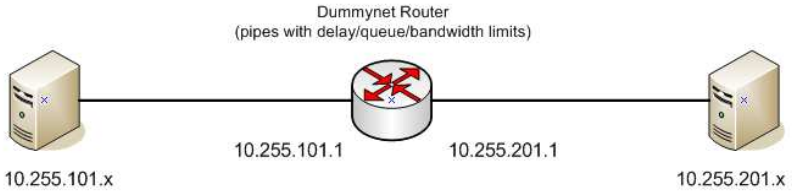


Figure IX.2: Dummynet Setup

We have four Dummynet nodes with dual Intel Xeon CPU at 2.4Ghz and with 2GB of memory. Each node has two Intel Gigabit ethernet cards. The operating system is FreeBSD 6.1. Table IX.1 shows the loss ratio measured from fixed-rate UDP traffic (generated by Iperf tool) under different rates.

For our experiment, we also use the OptIPuter Lambda-network testbed, which consists of distributed storage clusters with 10 Gbps connections at the Jacob School of Engineering, Computer Science Engineering and SIO at UC San Diego,

Table IX.1: Packet Loss Ratio Measured on a Single Link through Dummynet with 50ms RTT

Bandwidth	Measurement 1	Measurement 2	Measurement 3
900 Mbps	0.025%	0.022%	0.01%
850 Mbps	0.027%	0.00085%	0.001%
800 Mbps	0 0.001%	0.0052%	0

UIC/Chicago, NCSA, and UvA/Amsterdam. Figure IX.3 shows the network topology of the OptIPuter testbed. The UCSD OptIPuter network is controlled by a configurable MEMS-based, optical, cross-connect (OXC) switch, which is capable of dynamically switching connections among sites when needed. The RTT between UIC and UCSD or NCSA and UCSD is around 60 ms; the RTT between Amsterdam and UCSD is around 140 ms. Figure IX.4 shows an emulated testbed topology for the above network and end nodes.

In most of the experiments, we use the CSE machines at the UCSD site as receivers. Because the connection between Amsterdam and UCSD is not always operational, we use Dummynet routers to emulate the delays to make our experimental simulation as similar as possible to the real testbed.

Altogether, five Dummynet routers are deployed at CSE. Because there are some performance shortcomings with Broadcom NICs, we cap the bandwidth provided by the dummynet router at 700 Mbps for the experiments in which we validate the fairness and convergence properties of our bandwidth-sharing approach. We also provide some performance measurements obtained with full bandwidth on a real testbed without emulated delay.

IX.4 Measurements

IX.4.A One-to-One Transfer

We first conduct one-to-one fixed-rate transfers to validate whether the basic behavior of the GTP implementation conforms with that of previous simulated

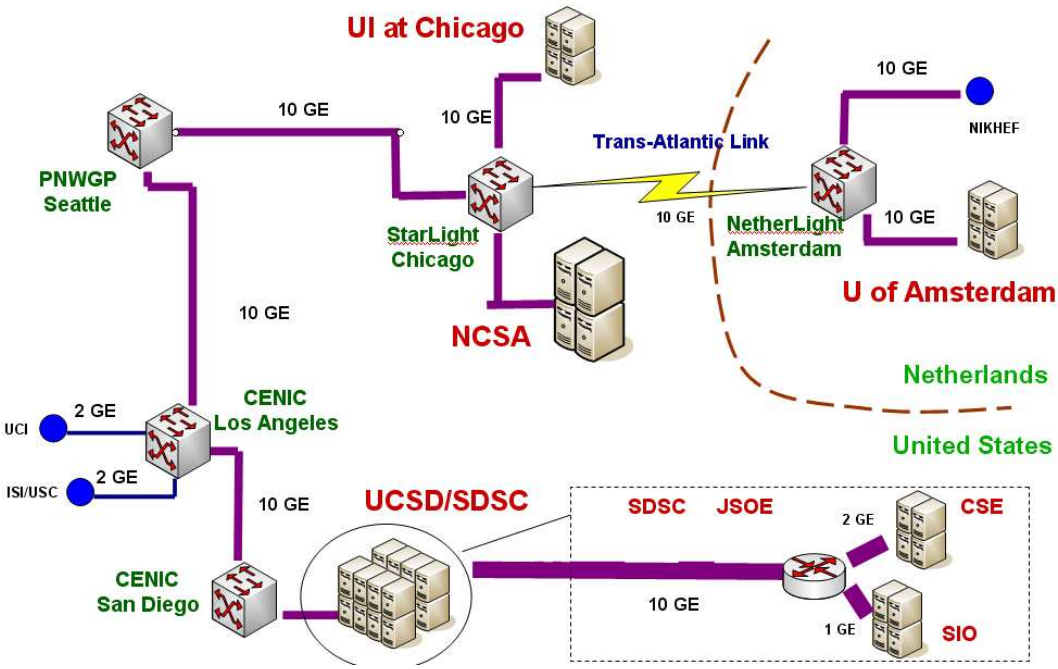


Figure IX.3: The OptIPuter testbed topology

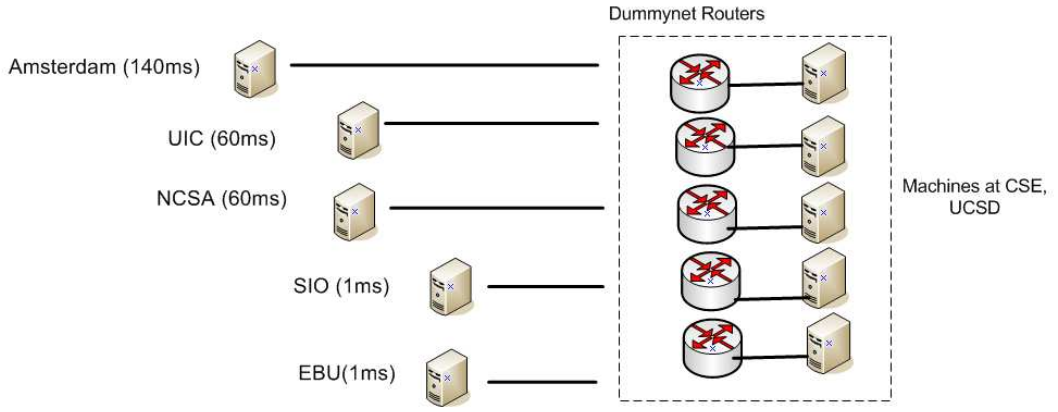


Figure IX.4: The emulated testbed topology

results.

Be default the emulated RTT is 60ms, $\alpha = 0.1$ and $\beta = 0.2$. The central control interval is set to $20ms$. Figure IX.5 depicts the trajectories of a single GTP session from prototype implementation and simulations. We observe that two

trajectories conform with each other, but there is a slight difference in their convergence rate: the simulated trajectory from begins more slowly but later increases faster than does the prototype-implementation trajectory. We believe this is due to the randomness of scheduling associated with the implementation.

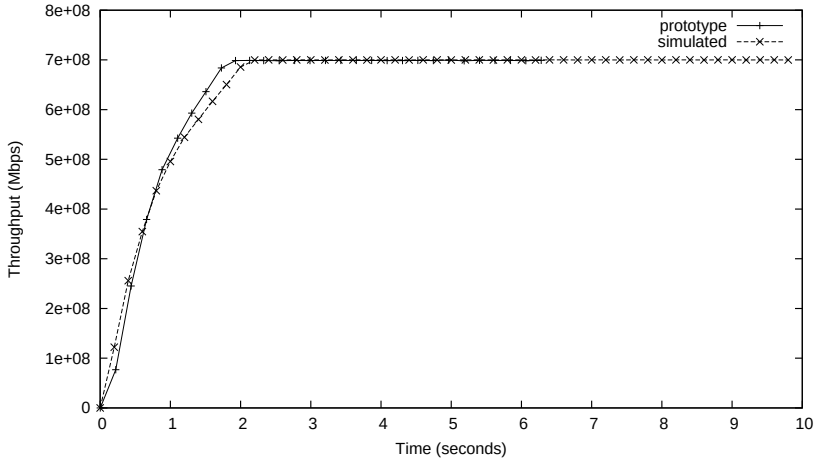


Figure IX.5: Comparison between simulation and prototype behaviors: the case of one-to-one transfer with 60 ms delay and 700 Mbps bandwidth

We show in Figure IX.6 the rate trajectories of GTP with different values of α. This confirms with previous simulation results and the smaller the α, the longer it takes to reach full capacity. Figure IX.7 shows the rate trajectories for different values of round-trip time. Several factors may cause the short-term randomness and oscillations shown in these figures. First, if the sending/receiving process is scheduled to be operational during less than 100% of a control interval, then the sending/receiving process may not send/receive as fast as expected and the number reported for that control interval may not conform with the number obtained from the simulations. Second, if the control interval is too small, it may not be scheduled as planned because it is competing for CPU scheduling cycles with the sending/receiving process. Third, any noises from *nak* packet handling, other I/O operations, or thread scheduling may influence the performance. However, we expect that these effects are relevant only in the short run and that the

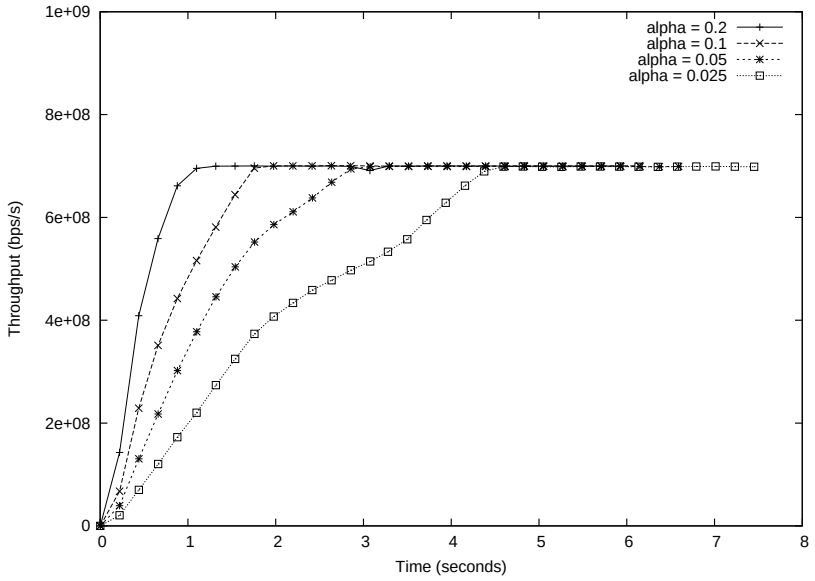


Figure IX.6: Trajectories of single GTP session with various values of α : the case of one-to-one transfer with 60 ms delay and 700 Mbps bandwidth

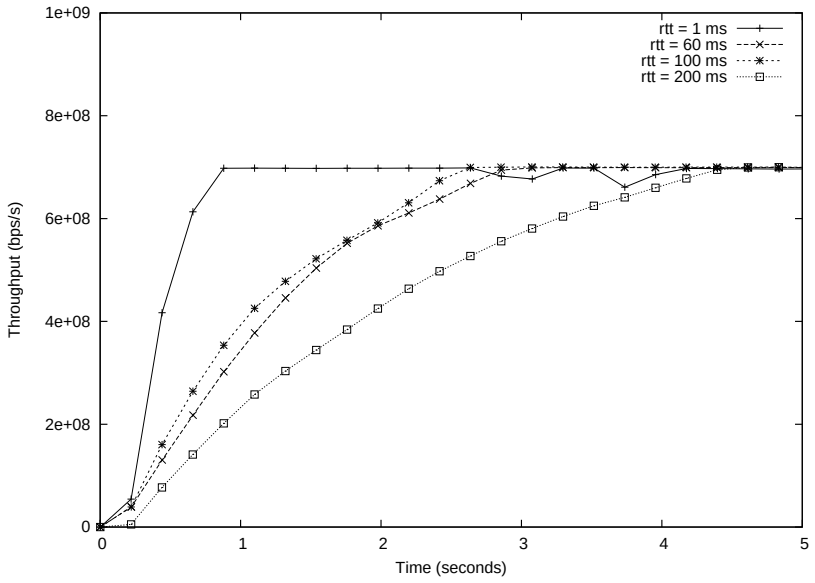


Figure IX.7: Trajectories of single GTP session with various values of RTT: the case of one-to-one transfer with 60 ms delay and 700 Mbps bandwidth

system-convergence trend nevertheless holds.

IX.4.B Many-to-One Transfers

In this section, we present the measured results of a five-to-one transfer, the layout of which mimics the OptIPuter testbed topology. The receiver is located at CSE and senders are from SIO (emulated), JSOE, NCSA, UIC and Amsterdam (emulated) with RTTs of 1 ms, 1 ms, 60 ms, 60 ms, and 140 ms, respectively. The connections from SIO and Amsterdam are emulated with Dummynet.

We first let all sessions begin simultaneously; the resulting trajectories are shown in Figure IX.8. We observe that the sessions with shorter RTTs increase faster, but eventually all sessions converge and receive an equal share of the end-node capacity. Figure IX.9 shows the case in which sessions start one at a time. Every time a new session joins, all active session converge to the same rate share. Figure IX.10 illustrates the case in which five sessions begin in a sequence that is the reverse of the sequence used in Figure IX.9. Finally, in the last experiment of many-to-one transfers, we let each session transfer an equal amount of the data (1.5 GB), but begin at different time. The resulting trajectories of the session rates are seen in Figure IX.11, which shows that when one session finishes, the remaining sessions converge to a new, higher equilibrium rate. Note that the remaining flows increase in rate as desired when several flows complete and exit the system.

IX.4.C Many-to-Many Transfers

In this section, we illustrate the performance of the prototype implementation with a more complicated five-to-five transfer scenario. The five receivers are located at CSE. The five senders are the same as those in our previously described five-to-one scenario: SIO (emulated), JSOE, NCSA, UIC and Amsterdam (emulated), with RTTs of 1 ms, 1 ms, 60 ms, 60 ms, and 140 ms, respectively (see Figure **??**).

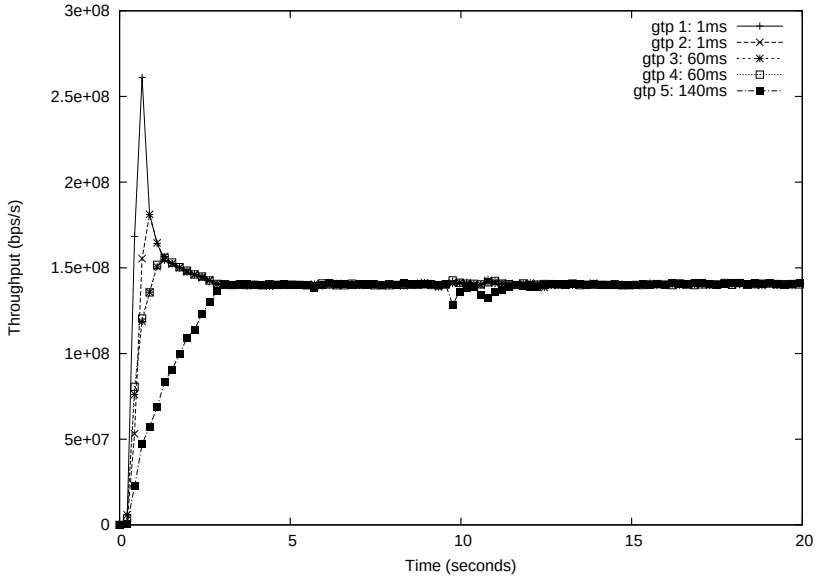


Figure IX.8: Rate trajectories of five-to-one transfer: all sessions start at the same time

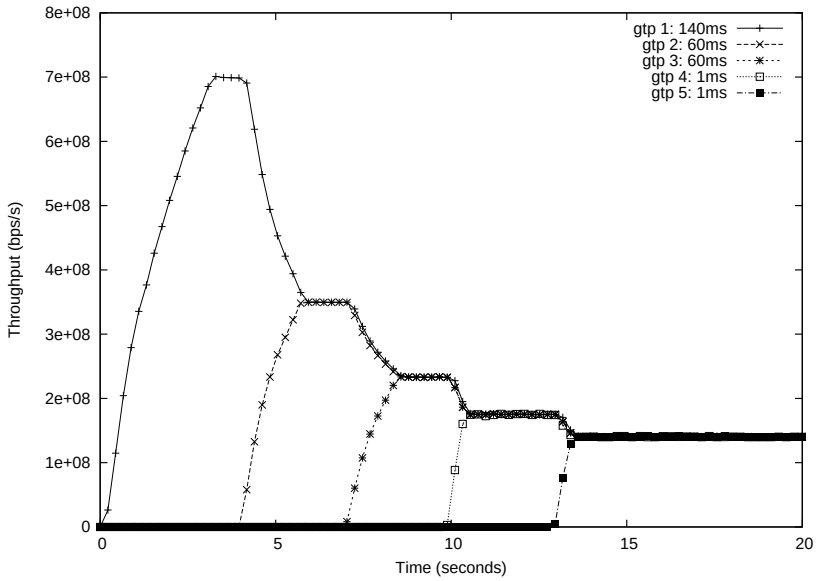


Figure IX.9: Rate trajectories of five-to-one transfer: sessions start at different time

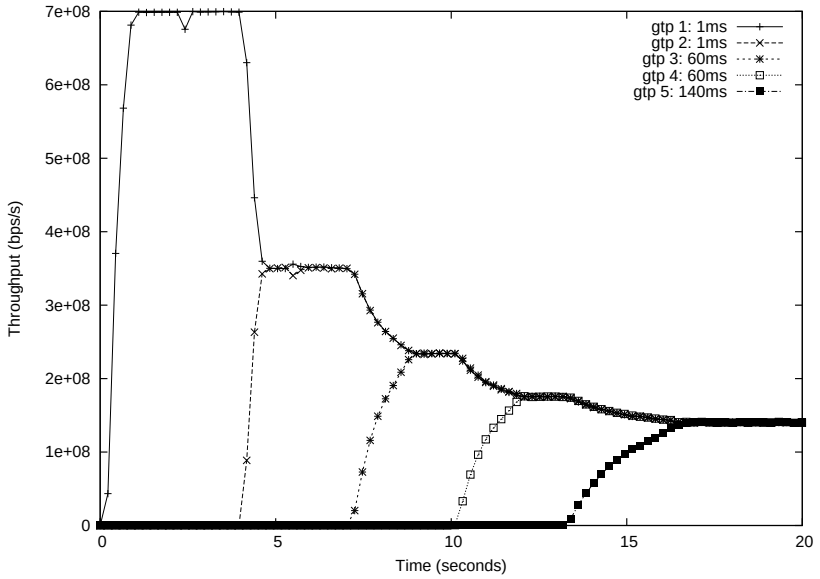


Figure IX.10: Rate trajectories of five-to-one transfer: sessions start at different time, in reversed order

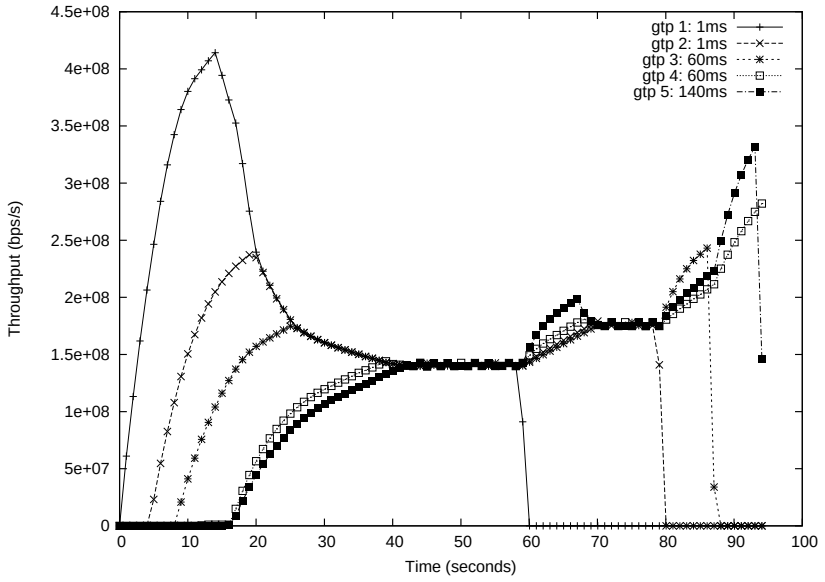


Figure IX.11: Rate trajectories of five-to-one transfer with fixed (1.5 GB) transfer size

We consider the following two scenarios:

- Scenario 1: Sessions start at the same time. We show the trajectories as well as 2-norm distance to validate whether the system converges.
- Scenario 2: Sessions start at different time. In this case, we show the rate trajectories of each session to observe session dynamics when new sessions join or existing sessions exit.

For Scenario 1, we firstly let each source node initiate four sessions with different destination nodes. Each sink node has exactly four associated sessions. The rate trajectories are shown in Figure IX.12; rate trajectories from simulations with the same settings are shown in Figure IX.13 and 2-norm distance trajectory is shown in Figure IX.14. We observe that although sessions may increase their transmission rate at different speed initially, they share the same equilibrium rate, which is one-fourth of the end-node capacity. And 2-norm distance monotonically decreases to zero. Trajectories from prototype measurements and simulations show similar convergence patterns.

We now vary the number of sessions from each source between 1 and 5. The connections are described in Table IX.2. Under this setup, sessions have different rates under equilibrium. The rate trajectories are shown in Figure IX.15 and 2-norm distance trajectory is shown in Figure IX.16. As the 2-norm distance converges to zero, it is proved that the system converges to an equilibrium state, in which sessions have different rates.

For Scenario 2, we vary the number of concurrent sessions of each end node in the following two cases.

(1) Each sender initiates two sessions to *two* different sinks, and each sink has two associated sessions.

(2) Each sender initiate three sessions to *three* different sinks, and each sink has three associated sessions.

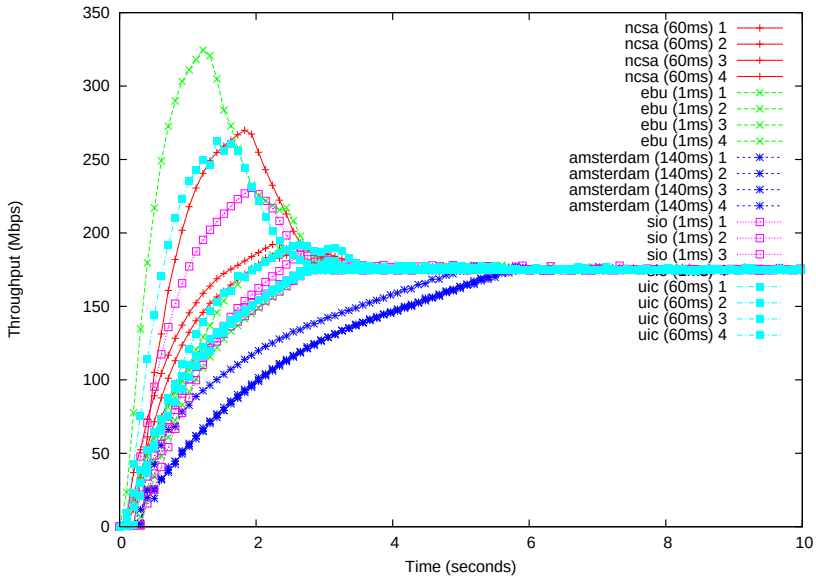


Figure IX.12: Rate trajectories of five-to-five transfer: 20 sessions (prototype)

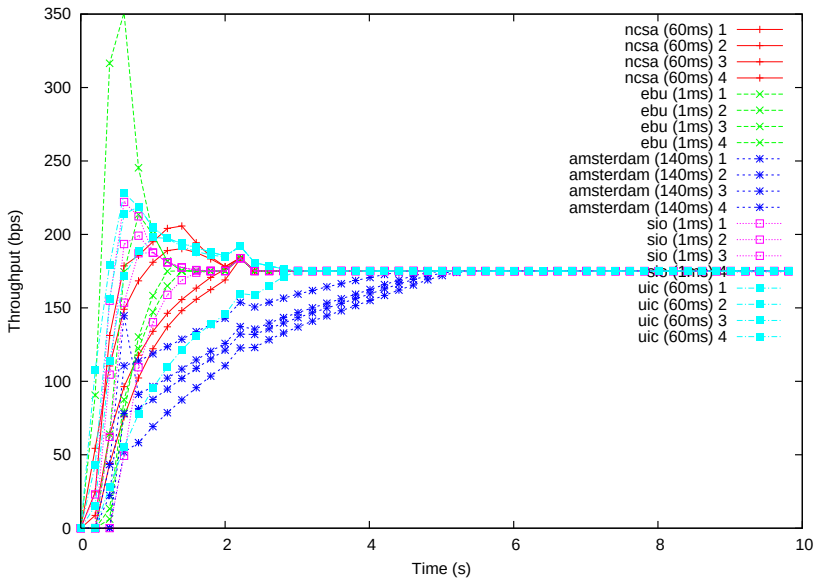


Figure IX.13: Rate trajectories of five-to-five transfer: 20 sessions (simulations)

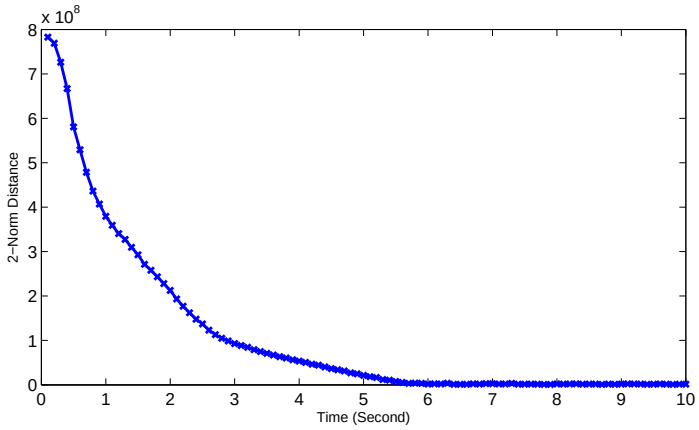


Figure IX.14: 2-Norm distance of five-to-five transfer: 20 sessions

Table IX.2: Connections between the sources and sinks for a five-to-five transfer

Source	Sink
1 (NCSA, 60ms)	1, 2, 3, 4, 5
2 (EBU, 1ms)	1, 2, 3, 4
3 (Amsterdam, 140ms)	1, 2, 3
4 (SIO, 1ms)	1, 2
5 (UIC, 60ms)	1

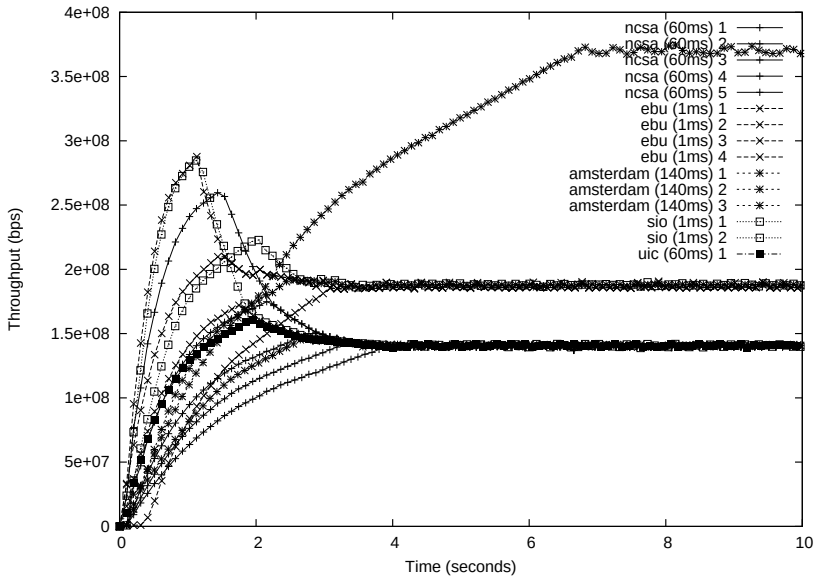


Figure IX.15: Rate trajectories of five-to-five transfer: 15 sessions

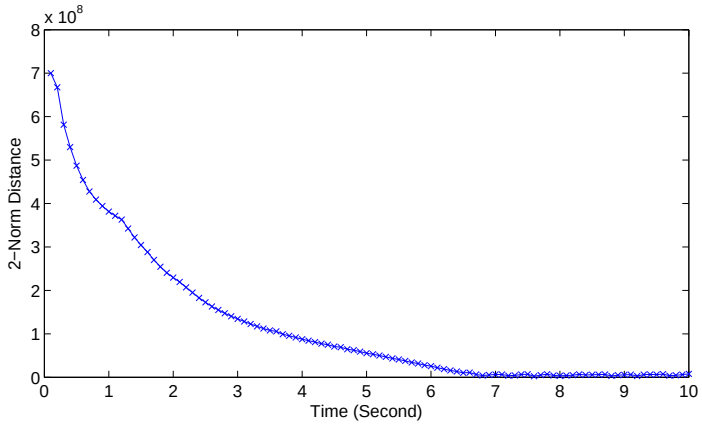


Figure IX.16: 2-Norm distance of five-to-five transfer: 15 sessions

In each of these scenarios, sessions will begin at different times. We validate the convergence properties by determining whether there exists a period in which each session receives an equal share of the bandwidth, which represents half or one-third of the total link capacity.

The resulting rate trajectories are shown in Figures IX.17 and IX.18. We observe that in both cases, after all the sessions have begun, there is a time segment in which all sessions receive the same bandwidth share. In Figure IX.18 we see that when one session finishes, the two competing sessions at the source or sink are able to increase their bandwidth share. At approximately $t = 35s$, five session have ended and ten sessions remain active. These remaining sessions continue to converge and each receives an equal share of the bandwidth. This is then equivalent to the results shown in Figure IX.18. Together, these two experiments demonstrate GTP's ability to manage across sessions with different RTTs and ensure that each session receives a fair share of the bandwidth.

IX.5 Summary

In this chapter, we described a prototype implementation of our distributed end-node bandwidth sharing algorithm. We validate its performance with point-to-point, multipoint-to-point and multipoint-to-multipoint experiments. By comparing with simulation results, verifying using 2-norm distance and observing the session dynamics, we validate that the prototype holds the same convergence and fairness properties of its underling algorithm. The prototype achieves high efficiency and treat sessions with a wide range of RTTs fairly.

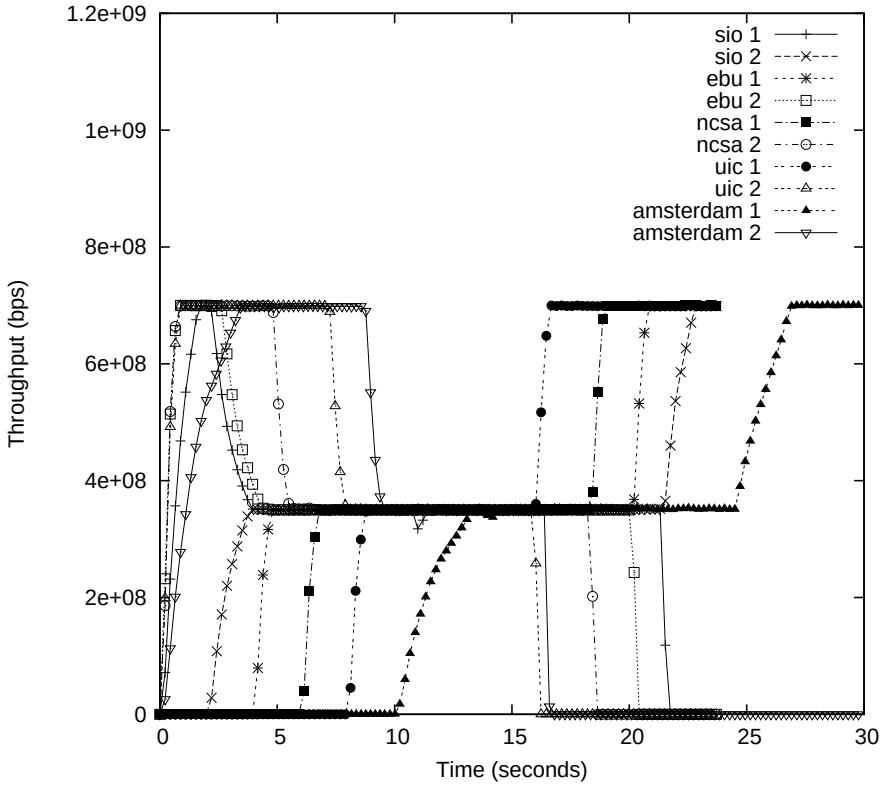


Figure IX.17: Rate trajectories of five-to-five transfer: each source and sink has two associated sessions

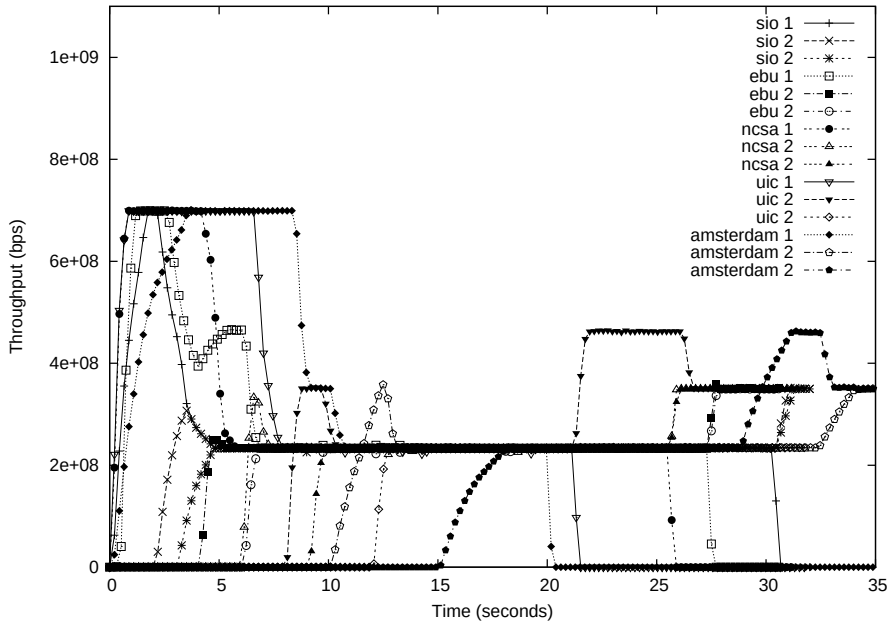


Figure IX.18: Rate trajectories of five-to-five transfer: each source and sink has three associated sessions

X

Conclusion

In this chapter, we summarize the research described in this thesis. In Section X.1 we highlight the key research contributions. In Section X.2 we discuss the implications and impacts of our research. We describe possible future work in Section X.3.

X.1 Dissertation Summary

Recent advances in optical networking provide abundant network bandwidth that exceeds the capacity of end systems. In such a networked environment, known as a lambda network, in which high-speed optical links connect end nodes with different RTTs, the network traffic contention is pushed from the network's core to the end nodes. Providing efficient and fair rate allocation and sharing for active sessions in lambda networks is an important and ongoing research challenge. Traditional transport protocol (TCP) performs poorly on high-speed, long-distance network links. Various high-speed TCP variants have been proposed. However, they target different network environments and have different optimization goals and it is unclear whether those schemes can be directly applied to solve the problem of fair bandwidth sharing in lambda networks. It is also unknown whether we can develop a new distributed bandwidth-sharing scheme based on the unique characteristics of

such lambda network environments.

In this thesis, we have attempted to answer these key research questions by exploring an end-node-based distributed bandwidth-sharing approach. Our approach captures the fact that when end-node capacity becomes the bottleneck in lambda networks, fair rate allocation becomes a question of sharing end node capacities fairly. Our approach also tries to create smooth transitions via rate adaptation while considering each session's desired rate. We study the convergence and stability properties of our approach in order to understand not only how well it performs under different network topologies and design parameters, but also how to implement such a scheme most efficiently.

We have created a mathematical model that captures the characteristics of lambda networks and how they differ from the traditional Internet. Based on these characteristics, the model simplifies the roles of packet switches within the network and contains variables such as end-node capacity, session RTTs and the sessions' desired rates. The model provides a basis for comparative studies of rate allocation problems that are created under different high-speed protocols.

We then identified the key challenges of bandwidth allocation and sharing for lambda networks and to created a formulation of the bandwidth-sharing problem. The formulation of the problem defined the optimization goal that we subsequently established for possible solutions.

As the first step toward a new approach, we provided a global rate-allocation algorithm that calculates the max-min rate allocation given the topology of lambda networks and the constraints of a set of finite desired session rates, providing that such max-min rate allocation is unique. This global algorithm provides a means to validate whether the convergence point of any distributed algorithm is the optimal solution to the problem that we defined.

We then proposed a distributed end-node bandwidth-sharing algorithm, using control and adaptation schemes for rate allocation. The algorithm allocates source and sink capacity among active sessions at each end node. The real session

sending rate is determined by these allocations.

We conducted analytical and simulated studies of the convergence and stability properties of our distributed end-node bandwidth-sharing algorithm. The analytical study proved that the distributed algorithm converges to the unique max-min fair rate allocation, as calculated by the global algorithm, from any initial or transitional state. Simulations confirmed the results of the analytical studies in distributed network environments with various parameters for size, latency, and synchrony. Comparative studies of the distributed rate-allocation algorithm with TCP, its variants, rate-based protocols, and router- or switch-assisted rate-allocation schemes showed that our algorithm performs better than existing systems in terms of both efficiency and fairness.

We also studied the problems inherent in design and implementation and presented a prototype of the distributed end-node bandwidth-sharing algorithm. Measurements taken while using the prototype in emulated environment and in real networks conformed with the results that were obtained previously through simulations and analytical studies.

X.2 Implications and Impacts

The main implication of our research is that the end-node-based rate allocation and control approach is a fundamentally sound scheme that can improve the efficiency and fairness of bandwidth sharing among active sessions. First, the analytical studies have shown that convergence and fairness can be achieved by letting each end node operate asynchronously under the distributed end-node bandwidth-sharing algorithm. Specifically, the results from our analytical study prove that session rates converge to a unique max-min fair bandwidth allocation from any initial or transitionary state. Furthermore, this end-node-based rate allocation and control approach allows each session to have different desired session rates. Second, the results of the simulations and prototype applications show that our approach

causes session rates to converge under a wide range of network sizes, RTTs, desired rates, and algorithm parameters.

A second implication of our research is that most of the high-speed transport protocols cannot achieve both efficiency and fairness in lambda networks as our distributed end-node bandwidth-sharing algorithm does. This is made obvious by comparing our scheme with other approaches under various network topologies, traffic patterns and sessions' desired rates. Different transport schemes target different types of networks and traffic, and thus have different optimization goals. For example, TCP targets scalable traffic management for the Internet and ensuring fairness for sessions with different RTTs is not one of its optimization goals. In contrast, our algorithm's aim is to share end-node capacity among a limited number of long-lived sessions while prioritizing max-min fairness among sessions with different RTTs.

A third implication of our research is that the design of any aggressive rate allocation and control schemes must address coexistence with TCP. We showed that the implementation of our distributed end-node bandwidth-sharing algorithm (GTP) may occupy most of the end-node capacities, thus behaving unfairly to TCP sessions. We have proposed several possible solutions including limiting total GTP capacity at each end node and dynamically predicting the aggregate TCP rate and adjusting GTP session rates accordingly.

The fourth and final implication of our research is that a lot of end-system optimization is required to build an efficient prototype. Our experience shows that in order to send packets at gigabits per second, the prototype must be carefully designed to minimize data-handling overhead. This suggests that using a general protocol implementation framework may be inadequate, because optimization techniques vary with individual protocols.

X.3 Future Work

The research described in this thesis focused primarily on efficient bandwidth sharing in lambda networks and the convergence, fairness and stability properties of end-node-based bandwidth-sharing algorithms. We have demonstrated that we met this goal through a comprehensive set of analytical, simulation and experimental prototype studies. We believe that additional contributions to this research topic can be made by extending our approach to a wider range of network topologies and granularity, considering the dynamics of end-node capacities, researching the algorithm's coexistence with other protocols, and considering rate control over bursty traffic. The following sections briefly discuss these potential areas for future work.

X.3.A Considering Networks with Larger Scales and Higher Bandwidth

In this thesis, we focused on analytical studies of our distributed end-node bandwidth sharing scheme. The main goal for simulations and prototype experiments is to validate its convergence and fairness properties. As a future work, empirical studies can be extended to networks with larger scales and higher bandwidth. More specifically, we can expand the measurements to include the following scenarios.

- End nodes equipped with 10 Gbps NICs;
- emulated 10 Gbps environment based on time dilution [46];
- extended network topology with the OptIPuter partner networks (e.g. National Lambda Rails [14], StarLight [19]), and
- emulated environment [6] with more end nodes.

X.3.B Considering Networks with In-Network Bottlenecks

For our research, it was sufficient to consider only those traffic bottlenecks located at sources and sinks, but we believe that our end-node bandwidth-sharing algorithm can also be applied to networks in which routers and packet switches are bottlenecks. Only minor changes need to be made to let each router or packet switch run the same bandwidth sharing algorithm that end nodes do and then to send feedback about their desired rates for each session to the traffic sources. In this case, the real packet sending rate is bounded by the expected rates from each router or switch along its path. Formal analytical and simulation studies are needed to prove whether this is a workable solution. The purpose behind this study would be to investigate whether our distributed end-node bandwidth-sharing scheme can be applied to a general Internet environment to control long-lived sessions with the help from routers.

X.3.C Considering Different Control Granularities

In this thesis, we focused on our distributed end-node bandwidth-sharing scheme at the packet level and presented it as an efficient alternative transport protocol in lambda networks. However, we want to point out that our approach can also work at different granularities, including:

- Session level. For example, if we want to deploy a long-lived FTP service to transfer data among different sites with the requirement that session rates are not changed very often, one possible solution is to apply our distributed end-node-based bandwidth-sharing algorithm by setting control intervals with a longer period time (e.g., 1 minute).

- Light path level. When a limited set of light paths are required for multiple applications connecting different end nodes, we can apply our end-node bandwidth-sharing algorithm to dynamically schedule light paths among different end nodes.

Future studies can apply the proposed distributed algorithm to solve the problems presented by different control granularities.

X.3.D Considering End-Node Capacity Dynamics

An important assumption in our research is that we have knowledge about the capacity of each end node. In our prototype, we preferred to set it to no more than 90% of the access bandwidth of each end node, because the total data-handling speed may not be able to reach full capacity. A suggested future area of research is to study how we can dynamically adjust this total capacity based on the packet loss experienced for each session. One possible adjustment could be that if all sessions experience the same level of packet loss, the total capacity should be adjusted downward. However, if packet loss is observed only at some of the sessions, it may not be related to the total bandwidth capacity of the nodes.

X.3.E Considering Bursty Traffic

We have demonstrated that our distributed end-node bandwidth-sharing algorithm is a sound solution for long-lived sessions. A future research topic is to study how it can be extended to handle bursty traffic. Possible questions to ask include:

- Can we use the proposed algorithm in its unaltered form to handle bursty traffic?
- If not, what is the best way to adapt our approach to handle bursty traffic?
- Can the system still converge under background bursty traffic?

References

[1] Biomedical informatics research network (birn). *http://www.nbirn.net*.

[2] Canarie. *http://www.canarie.ca*.

[3] Data reservoir project. *http://data-reservoir.adm.s.u-tokyo.ac.jp/*.

[4] Earthscope. *www.earthscope.org*.

[5] ediamond: Digital mammography. *http://www.ediamond.ox.ac.uk*.

[6] Emulab. *http://www.emulab.net/*.

[7] Fusion collaboratory. *http://www.fusiongrid.org/*.

[8] Glimmerglass: Intelligent optical switching solution. *http://www.glimmerglass.com*.

[9] Globus xio. *http://www-unix.globus.org/developer/xio/*.

[10] Google said to be building massive core network. *http://www.informationweek.com/story/showArticle.jhtml?articleID =171000612*.

[11] Griphyn project. *http://www.griphyn.org*.

[12] International virtual data grid laboratory (ivdgl). *http://www.ivdgl.org*.

[13] Naregi. *http://www.naregi.org*.

[14] National lambda rail. *http://www.nlr.net/*.

[15] Netherlight. *http://www.netherlight.net*.

[16] Ns-2 simulator. *http://www.isi.edu/nsnam/ns*.

[17] Particle physics data grid (ppdg). *http://ppdg.net/*.

[18] Sloan digital sky survey. *http://www.sdss.org*.

[19] starlight. *http://www.startap.net/starlight/.*

[20] Teragrid. *http://www.teragrid.org.*

[21] Allcock, W., Bester, J., Bresnahan, J., Chervenak, A., Liming, L., and Tuecke, S., Gridftp: Protocol extensions to ftp for the grid. *Draft GridFTP Protocol.*

[22] Altman, E., Barman, D., Tuffin, B., and Vojnovi, M., Parallel tcp sockets: Simple model, throughput and validation. *In Proceedings of IEEE INFOCOM 2006.*

[23] Athuraliya, S., Li, V. H., Low, S. H., and Yin, Q., REM: Active queue management. *IEEE Network, Vol 15 No. 3 P48-53.*

[24] Berger, L., Generalized multi-protocol label switching (gmpls) signaling functional description. *IETF, RFC 3471, Jan. 2003.*

[25] Bertsekas, D. P., and Gallager, R., Data networks. *Prentice-Hall, Englewood-Cliffs, New Jersey, 1992.*

[26] Bonomi, F., and Fendick, K., The rate-based flow control framework for the available bit rate atm service. *IEEE Network, March/April 1995, pp. 25-39.*

[27] Border, J., Kojo, M., Griner, J., and Montenegro, G., Performance enhancing proxies. *RFC 3135, Nov 2000.*

[28] Brakmo, L., O'Malley, S., and Peterson, L., Tcp vegas: New techniques for congestion detection and avoidance. *In Proceedings of the SIGCOMM 1994 Symposium.*

[29] Charny, A., Clark, D. D., and Jain, R., Congestion control with explicit rate indication. *Proc. IEEE International Conference on Communications (ICC'95).*

[30] Chase, J., Gallatin, A., and Yocum, K., End system optimizations for high-speed TCP. *IEEE Communications Magazine*, **39**(4), 68–74.

[31] Chien, A. A., Wu, X., Taesombut, N., Weigle, E., Xia, H., and Burke, J., Optiputer system software framework.

[32] Chiu, D. M., and Jain, R., Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems, 17(1):1–14, June 1989.*

[33] Chu, J., Zero-copy tcp in solaris. *In Proc. Usenix 1996, San Diego, CA, Jan 1996.*

[34] Clark, D. D., Lambert, M., and Zhang, L., Netblt: a high throughput transport protocol. *In Proceedings of ACM SIGCOMM 1987.*

[35] DeFanti, T., Laat, C., Mambretti, J., Neggers, K., and Arnaud, B., Translight: a global-scale lambdagrid for e-science. *Communications of the ACM (CACM), 47(11), November 2003.*

[36] et. al., J. L., The global lambda visualization facility: An international ultra-high-definition wide-area visualization collaboratory. *Future Generation Computer Systems, Volume 22, Issue 8, Elsevier, October 2006, pp. 964-971.*

[37] Falk, A., Faber, T., Bannister, J., Chien, A., Grossman, R., and Leigh, J., Transport protocols for high performance. *Communications of the ACM (CACM), 47(11), November 2003.*

[38] Fisk, M., and Feng, W., Dynamic right-sizing in tcp. *In Proc. of the Los Alamos Computer Science Institute Symposium, October 2001.*

[39] Floyd, S., Highspeed tcp for large congestion windows. *Internet draft.*

[40] Floyd, S., Handley, M., Padhye, J., and Widmer, J., Equation-based congestion control for unicast applications. *In Proceedings of ACM SIGCOMM 2000.*

[41] Floyd, S., and Jacobson, V., Random early detection gateways for congestion avoidance. *In IEEE/ACM Transactions on Networking, 1(4):397-413, Aug. 1993.*

[42] Foster, I., and Kesselman, C., The grid: blue print for a new computing infrastructure. *Morgan Kaufmann, 1999.*

[43] Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S., The physiology of the grid: an open grid services architecture for distributed systems integration. *Grid Forum white paper, 2003.*

[44] Greene, T., Dwdm is the right rx for new york presbyterian hospital. *Network World, 05/16/05, available at http://www.networkworld.com/news/2005/051605-presbyterian.html.*

[45] Gu, Y., and Grossman, R., Experiences in design and implementation of a high performance transport protocol. *In Proceedings of Supercomputing 2004.*

[46] Gupta, D., Yocum, K., McNett, M., Snoeren, A. C., Vahdat, A., and Voelker, G. M., To infinity and beyond: Time-warped network emulation. *In Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI 2006). May 2006.*

[47] Hacker, T. J., Noble, B. D., and Athey, B. D., Improving throughput and maintaining fairness using parallel tcp. *In proceedings of IEEE INFOCOM 2004, Hongkong, March 2004.*

[48] He, E., Leigh, J., Yu, O., and DeFanti, T., Reliable blast udp: predictable high performance bulk data transferrbudp. *IEEE Cluster Computing, 2002, p. 317.*

[49] He, E., Wang, X., Vishwanath, V., and Leigh, J., Ar-pin/pdc: Flexible advance reservation of intradomain and interdomain lightpaths. *Proceedings of IEEE GLOBECOM 2006, San Francisco, November 2006.*

[50] Hollot, C., Misra, V., Towsley, D., and Gong, W., On designing improved controllers for aqm routers supporting tcp flows. *In Proceedings of IEEE INFOCOM 2001, April 2001.*

[51] Hou, Y. T., Tzeng, H., Panwar, S. S., and Kumar, V. P., A generalized max-min rate allocation policy and its distributed implementation using the abr flow control mechanism. *In Proceedings of IEEE INFOCOM 1998, pp.1366-1375, San Francisco, CA.*

[52] Jain, R., A delay based approach for congestion avoidance in interconnected heterogeneous computer networks. *Computer Communications Review, ACM SIGCOMM, pp. 56-71.*

[53] Jain, R., The art of computer systems performance analysis: techniques for experimental design, measurement, simulation and modeling. *John Wiley and Sons INC.*

[54] Jin, C., Wei, D., and Low, S., Fast tcp: Motivation, architecture, algorithms, and performance. *In Proceedings of IEEE INFOCOM 2004, Hongkong, March 2004.*

[55] Kalampoukas, L., Varma, A., and Ramakrishnan, K. K., An efficient rate allocation algorithm for ATM networks providing max-min fairness. 143–154.

[56] Karbhari, P., Zegura, E., and Ammar, M., Multipoint-to-point session fairness in the internet. *Proceedings of IEEE INFOCOM 2003.*

[57] Katabi, D., and Blake, C., A note on the stability requirements of adaptive virtual queue, 2002. *MIT Technical Memo.*

[58] Katabi, D., Handley, M., and Rohrs, C., Internet congestion control for high bandwidth delay product network. *In Proceedings of ACM SIGCOMM 2002, Pittsburgh, Aug 2002.*

[59] Kelly, F. P., Maulloo, A. K., and Tan, D. K. H., Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society 49 (1998), 237-252.*

[60] Kelly, T., 2003: Scalable tcp: Improving performance in highspeed wide area networks.

[61] Kohler, E., Handley, M., and Floyd, S., Designing dccp: Congestion control without reliability. *In Proceedings of SIGCOMM 2006.*

[62] Low, S. H., Andrew, L. L. H., and Wydrowski, B. P., Understanding xcp: Equilibrium and fairness. *Proc. IEEE Infocom 2005, Miami, FL, March 2005.*

[63] Low, S. H., Paganini, F., Wang, J., Adlakha, S., and Doyle, J. C., Dynamics of tcp/aqm and a scalable control. *In Proceedings of IEEE INFOCOM, June 2002.*

[64] Martin-Flatin, J., and Ravot, S., Tcp congestion control in fast long-distance networks. *Technical Report CALT-68-2398, CalTech, USA.*

[65] Metz, C., The bright side of dark fiber optics. *PC Magazine, available at http://www.pcmag.com/article2/0,1759,1813381,00.asp.*

[66] Padhye, J., Firoiu, V., Towsley, D., and Kurose, J., Modeling tcp throughput: A simple model and its empiracal validation. *In ACM SIGCOMM 1998 conference on Applications, Technologies, Architectures, and Protocol for Computer Communication, Vancouver, Canada.*

[67] Radunovic, B., and Boudec, J. L., A unified framework for max-min and min-max fairness with applications. *Proceedings of 40th Annual Allerton Conference on Communication, Control, and Computing, Allerton, IL, October 2002.*

[68] Radunovic, B., and Boudec, J. L., Rate performance objectives of multi-hop wireless. *Proc. IEEE Infocom 2005, Miami, FL, March 2005.*

[69] Ramakrishnan, K., and Floyd, S., A proposal to add explicit congestion notification (ecn) to ip. *RFC 2481.*

[70] Rhee, S. H., and Konstantopoulos, T., Achieving max-min fairness by decentralization for the abr traffic control in atm networks. *IEICE Trans. Commun., Vol. E84-B, No.8.*

[71] Rizzo, L., Dummynet: a simple approach to the evaluation of network protocols. *Computer Communication Review, 27, Jan 1997.*

[72] Rizzo, L., Effective erasure codes for reliable computer communication protocols. *Computer Communication Review, vol. 27, no. 2, pp. 24-36, April 1997.*

[73] Rodrigues, S. H., Anderson, T. E., and Culler, D. E., High-performance local area communication with fast sockets. *USENIX'97, Anaheim, California, Jan 6-10, 1997.*

[74] Scarpa, M., Belleman, R., Sloot, P., and de Laat, C., Highly interactive distributed visualization. *iGrid2005 special issue, Future Generation Computer Systems, volume 22 issue 8, pp. 896-900 (2006).*

[75] Sivakumar, H., Bailey, S., and Grossman, R. L., PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks. *In Proceedings of Supercomputing 2000.*

[76] Smarr, L., Chien, A. A., DeFanti, T., Leigh, J., and Papadopoulos, P., The optiputer. *Communications of the ACM (CACM), 47(11), November 2003.*

[77] Swany, M., Improving throughput for grid applications with network logistics. *In Proceedings of Super Computing 2004, Pittsburgh, PA.*

[78] Taesombut, N., and Chien, A. A., Distributed virtual computer(dvc): Simplifying the development of high performance grid applications. *In Proceedings of the Workshop on Grids and Advanced Networks (GAN 04), April 2004, Chicago, Illinois.*

[79] Taesombut, N., Wu, X., Chien, A. A., and et al, Collaborative data visualization for earth sciences with the optiputer. *Journal of Future Generation Computer Systems.*

[80] Veeraraghavan, M., Cheetah: Circuit-switched high-speed end-to-end transport architecture. *Proc. 4th Optical Network and Communication Conference, Oct. 2003.*

[81] Weigle, E., and Chien, A. A., The composite endpoint protocol (cep): Scalable endpoints for terabit flows. *Proceedings of CCGrid 2005).*

[82] Wu, X., and Chien, A., Gtp: Group transport protocol for lambda-grid. *Proceedings of CCGrid 2004.*

[83] Wu, X., and Chien, A. A., Evaluation of end-node based protocols for lambda networks. *In Proceedings of the Fourth International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet2006), Nara, Japan, Feb 2-3, 2006.*

[84] Wu, X., and Chien, A. A., Evaluation of rate based transport protocols for lambda-grids. *In Proceedings of the 12th IEEE International Symposium on High-Performance Distributed Computing (HPDC), Honolulu, Hawaii, June 2004.*

[85] Xia, H., and Chien, A. A., Robustore: Robust performance for distributed storage systems. *Proceedings of the 14th NASA Goddard - 23nd IEEE Conference on Mass Storage Systems and Technologies (MSST2006), May 2006.*

[86] Xu, L., Harfoush, K., and Rhee, I., Binary increase congestion control for fast long-distance networks. *In Proceedings of IEEE INFOCOM 2004, Hongkong, March 2004.*

[87] Yin, N., and Hluchyi, M., On closed-loop rate controls for atm cell relay networks.

[88] Yu, O., Interacrrier interdomain control plane for global optical networks. *Proc. IEEE ICC, June, 2004.*

[89] Zhang, Y., and Dao, S., A measurement of tcp over long-delay network. *The 6th International Converence on Telecommunication Systems, Modeling and Analysis, Nashville, TN, March 1998.*