11-28-2011

Performance Optimization and Dynamics Control for Large-scale Data Transfer in Wide-area Networks

Xukang Lu

PERFORMANCE OPTIMIZATION AND DYNAMICS CONTROL FOR LARGE-SCALE DATA TRANSFER IN WIDE-AREA NETWORKS

by

Xukang Lu

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

December 2011

This dissertation is dedicated to my beloved parents.

Acknowledgements

This dissertation arose in part out of years of research that has been done since I came to Dr. Wu's group. By that time, I have worked with a great number of people who have contributed to the final production of this dissertation in assorted ways. I would like to express my gratitude to these individuals for their support and assistance.

First and foremost I want to express my deepest gratitude to my advisor, Prof. Qishi Wu for his excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research. Above all and the most needed, he provided me unflinching encouragement and support in various ways. His time, effort, and resources made him a constant oasis of ideas and passions in research, which exceptionally inspired and enriched my growth as a student. Prof. Wu has been also very supportive and considerate in many non-academic aspects, which have had significant impacts on my life as an international student.

I am equally thankful to the members of my dissertation committee, Prof. Lih-Yuan Deng, Prof. King-Ip Lin, and Prof. Vinhthuy Phan, for their constructive comments and insightful suggestions on my dissertation. Their kind support and guidance have been of great value throughout this study. My sincere thanks also go to all other faculty and staff members in our department and university for their help during my study. I had the pleasure of collaborating with Dr. Runzhi Li of Henan Education and Research Network Center, China, who offered great help with some parts of the performance evaluation. I would also like to pay tribute to my collaborators, Dr. Nageswara Rao at Oak Ridge National Laboratory, and Dr. Dantong Yu at Brookhaven National Laboratory, for sharing with me their expert and domain knowledge. I must acknowledge as well all my friends and lab mates who assisted, and supported my research and writing efforts over the years. Many thanks go in particular to Mr. Patrick Brown for his great contributions to the development of the NADMA system.

Words fail me to express my appreciation to my parents and two brothers for their love and encouragement. They were always supporting me in all my pursuits. Without their unconditional, unchanging, and unending love and care, this work would never have come into existence.

Table of Contents

List of Figures

Abstract

Xukang Lu, Ph.D. The University of Memphis. December 2011. Performance Optimization and Dynamics Control for Large-scale Data Transfer in Wide-area Network.
Major Professor: Prof. Qishi Wu

Transport control plays an important role in the performance of large-scale scientific and media streaming applications involving transfer of large data sets, media streaming, online computational steering, interactive visualization, and remote instrument control. In general, these applications have two distinctive classes of transport requirements: large-scale scientific applications require high bandwidths to move bulk data across wide-area networks, while media streaming applications require stable bandwidths to ensure smooth media playback. Unfortunately, the widely deployed Transmission Control Protocol is inadequate for such tasks due to its performance limitations.

The purpose of this dissertation is to conduct rigorous analytical study of the design and performance of transport solutions, and develop an integrated transport solution in a systematical way to overcome the limitations of current transport methods. One of the primary challenges is to explore and compose a set of feasible route options with multiple constraints. Another challenge essentially arises from the randomness inherent in wide-area networks, particularly the Internet. This randomness must be explicitly accounted for to achieve both goodput maximization and stabilization over the constructed routes by suitably adjusting the source rate in response to both network and host dynamics.

The superior and robust performance of the proposed transport solution is extensively evaluated in a simulated environment and further verified through real-life implementations and deployments over both Internet and dedicated connections under disparate network conditions in comparison with existing transport methods.

Chapter 1

Introduction

1.1 Overview

Transport control is an important factor in the performance of large-scale scientific and media streaming applications involving transfer of large data sets, media streaming, online computational steering, interactive visualization, and remote instrument control. In general, these applications have two distinctive classes of transport requirements: large-scale scientific applications require high bandwidths to move bulk data across wide-area networks, while media streaming applications require stable bandwidths to ensure smooth media playback.

Scientific applications: High-performance computing and networking technologies have enabled large-scale scientific collaborations in various domains such as earth science, climate, and high energy physics among multiple national laboratories within U.S. Department of Energy and other research institutes across the nation. These collaborative scientific applications [1–4, 59, 60, 121] typically generate colossal amounts of simulation or experimental data, on the order of terabytes at present and exabytes in the near future, which must be stored, managed, and transferred to different geographical locations for distributed data processing and analysis. Typical examples include: large simulation datasets produced by an eScience application on a supercomputer that need to be moved to a remote storage site or to another site where the analysis takes place, and computing workflows involving

remote visualization of large datasets.

The success of these collaborative scientific applications critically depends on adequate network accesses to the data generators, i.e., computing or experimental facilities. The efforts to support these large-scale applications on shared IP networks have not been very successful since very little guarantees can be provided on the throughput or dynamics and the available bandwidth varies depending on concurrent network traffic. In view of the limitations of existing methods, dedicated connections offer a promising solution to effectively support these network-intensive applications because they provide large capacity for massive data transfer. In fact, the importance of dedicated connections has been well recognized, and several network research projects are currently underway to develop such capabilities [5–12, 35, 51, 107, 137, 139]. Given dedicated channels, transport protocols are the key to delivering the provisioned bandwidths to the applications. Moreover, when dealing with large amounts of data and storage, scientists often need to interact with multiple heterogeneous storage and file systems, each with different interfaces and security mechanisms, and to pre-allocate storage to ensure unimpeded data generation and analysis. The success of these scientific applications also depend on bulk data management systems, which provide common access interfaces to storage resources, as well as advanced functionality such as dynamic space allocation and file management on shared storage systems.

Media streaming applications: Live media streaming applications often require the collective use of massively distributed network resources and therefore are not adequately supported by the traditional client-server architecture on the Internet. Peer-to-peer (P2P) overlay networks enable efficient resource sharing in distributed environments and provide a highly effective and scalable solution to this problem [13, 14, 135]. The performance metrics of streaming applications mainly concern throughput, jitter, and latency, and to a large degree, these performance metrics rely on the overlay network topology, upon which the streaming application is built. Therefore, constructing an efficient overlay network

topology has become a fundamental task in streaming applications.

Given a well-structured overlay network topology, transport protocols are again the key to achieving and sustaining an acceptable level of quality of service (QoS). Streaming applications often require streaming media be sent with predictable delays, which are in stark contrast with the delays experienced over the Internet, particularly by the messages sent using Transmission Control Protocol (TCP) [74]. In such applications, a stable transport channel serves as a "carrier" for streaming media. Since the flow is stable, the delays of the packets that constitute the flow have low levels of jitter that can be filtered out at the destination. Consequently, the streaming media carried by these packets have stable end-to-end delays suitable for sustaining a streaming level that ensures smooth media playback and continuous media supply. Without such transport stability, the stable throughput needed in these applications cannot be sustained over wide-area networks.

1.2 Problem Statements

In this section, we describe the problems associated with large-scale scientific and media streaming applications, and the problems we aim to solve in this dissertation.

1.2.1 Scientific Applications

We tackle two main problems in scientific applications: discovery of data transfer paths and transport control for goodput maximization, which are briefly described as follows.

1.2.1.1 Resource Provisioning

Several network and storage research projects are currently underway to meet the aforementioned networking and storage requirements of large-scale scientific applications. However, the existing tools, systems, or services have a very limited user scope thus far mainly because their deployment requires a certain level of network/host reconfigurations and most

science users are even not aware of their existence inside their own networks. As new computing and networking technologies rapidly emerge, enabling functionalities are progressing at an ever-increasing pace, unfortunately, so are the dynamics, scale, heterogeneity, and complexity of the networked computing environments. In many cases, application users, who are primarily domain experts, need to manually configure and execute their routine data-centric tasks over networks using software tools they are familiar with based on their own empirical studies, oftentimes resulting in unsatisfactory performance in such diverse and dynamic network environments. The challenge of utilizing these services is overwhelming when the user is unfamiliar with the systems and resources available to them. This challenge is exacerbated by the difficulty of using unfamiliar networking technologies while overcoming an often steep learning curve towards their adoption.

Apparently, the discovery of available networking and storage technologies is a critical step towards their wide adoption. In other words, science users must be made aware of these existing technologies with consideration of the data movement they intend to execute in their target network environments. However, users are often unwilling to explore alternative data transfer options due to the burdensome discovery and initiation process of advanced network protocols and the difficulty of constructing and testing various network paths. While a better data movement strategy may exist, users always tend to use slower but more familiar alternatives.

1.2.1.2 Transport Control for Goodput Maximization

In the goodput[1] maximization problem, we aim to explicitly account for the dynamics of network environments to maximize application goodput over dedicated connections.

Even with a high-bandwidth dedicated channel, an inappropriate transport protocol may lead to a low bandwidth utilization, resulting in a slow data transfer. The design of transport

[1]Goodput only counts the user payload and is equivalent in value to throughput if packet duplicates and protocol headers are negligible.

protocols is critical to achieve high link utilization and satisfy applications' networking requirements. From a transport protocol's perspective, dedicated channels obviate the need for explicit congestion and fairness control. However, the application throughput is still critically affected by a number of parameters that require careful selection and tuning. As indicated by the measurements over dedicated channels [108], the packet loss at high sending rates is often non-zero and the delay variations contain non-trivial random components. Due to the lack of a system-wide advance reservation scheme, the data receiver running in a shared computing environment with other resource-demanding workloads such as visualization and data analysis tools, oftentimes could not obtain sufficient system resources to process packets arriving from high-speed links, therefore leading to significant packet drops at the end system. Many transport protocols send a negative acknowledgment for a dropped data packet to ensure transmission reliability. At high data rates, generating and sending acknowledgments at the receiver consumes CPU time and may interfere with the host's receiving process, which probably leads to even more packet drops. Consequently, simply *a priori* fixing the source sending rate right at the connection capacity is unlikely to maximize the throughput at the destination since it typically causes losses at rates that depend on technologies used to provision the connection and dynamics of the running environment. Instead, the source rate must be continuously adjusted to match suitable rates, which yield the maximum goodput at the destination, which in turn involves accounting at some level for connection effects as well as host effects due to components such as Network Interface Card (NIC), CPU, memory and file systems.

The widely deployed TCP, which has been proved to be remarkably successful on the Internet, is not adequate to make full use of the high link capacity in wide-area dedicated networks because its Additive Increase Multiplicative Decrease (AIMD)-based congestion control algorithm does not perform well with links of high Bandwidth Delay Product (BDP). UDP-based transport protocols typically employ different threads to receive, acknowledge, and store packets, respectively, in order to achieve an overall high transport

performance. However, due to the lack of a parallel packet processing scheme in the kernel and the application, only one thread is actually running to consume all packets arriving from high-speed links, therefore leading to an inefficient resource utilization on the end system. The multi-core processors that are widely deployed and rapidly evolving make it now possible to improve application throughput by implementing a parallel receiving strategy in these UDP-based transport protocols. In order to support parallel data streams for a single data transfer, multiple data receiving processes must be executed on different CPUs.

1.2.2 Media Streaming Applications

We tackle two main problems in media streaming applications: topology construction and transport control for goodput stabilization, which are briefly described as follows.

1.2.2.1 Topology Construction

Due to the high resource demand, live media streaming applications are not adequately supported by the traditional client-server architecture on the Internet. Peer-to-peer (P2P) overlay networks enable efficient resource sharing in distributed environments and provide a highly effective and scalable solution to this problem [13, 14, 135]. There are three types of overlay architectures widely adopted in P2P-based live streaming systems: tree-push mechanism, mesh-pull mechanism, and hybrid mechanism [53] [122] [42] [133] [102] [82] [50] [134] [90].

Tree-push mechanism requires live streaming system to maintain the topology of the peer tree. It is the most natural architecture for overlay multicast, but is vulnerable in response to changes in the tree structure caused by peers' churn. Mesh-pull mechanism reduces the cost of maintaining tree structure. Nevertheless, it increases the redundancy traffic among peers requesting for Buffer-Mapping. Recent comparative study [99] of mesh-pull and tree-push in static and dynamic scenarios shows that mesh-pull exhibits a superior performance over tree-push, which statically maps contents to a particular overlay tree or diversely places peers in different overlay trees. However, another measurement study [73]

with a goal of exploring the global characteristics of a mesh-pull PPLive system shows that a mesh-pull live streaming architecture still incurs long start-up delays and playback lags, ranging from several seconds to a couple of minutes, due to inefficient peering strategies and video chuck scheduling schemes. In the hybrid mechanism, peers are divided into Super Peers (SPs) and Normal Peers (NPs) based on their service capacity such as upload bandwidth, online status, and CPU speed. Since SPs are generally more stable with higher upload bandwidths than NPs, tree-push is usually applied to SPs and mesh-pull is employed among NPs to handle peers' frequent joining and leaving requests. This mechanism enables SPs to accommodate peers' churn and fully utilize the network resources of NPs. The performance metrics of these streaming applications mainly concern throughput, jitter, and latency, and to a large degree, these performance metrics rely on the overlay network topology, upon which the streaming application is built. Therefore, constructing an efficient overlay network topology has become a fundamental task in streaming applications.

1.2.2.2 Transport Control for Goodput Stabilization

In the goodput stabilization problem, our research objective is to dynamically control the source rate such that the goodput is stabilized at a desired level.

In a well-structured overlay network topology, we still rely on a good transport protocol to achieve and sustain an acceptable level of quality of service (QoS). In general, the requirement of a stable flow at a target throughput level is fundamentally at odds with the traditional notions of fair bandwidth sharing in the Internet environments. Note that regular TCP-based Internet traffic competes with concurrent traffic for a fair share of the available bandwidth while the applications requiring stable channels aim to achieve a smooth data flow at a fixed rate. In practice, TCP and some other UDP-like transport methods have been most often used to transmit streaming media in the current (partially successful) implementations of the above applications, but their performance has not been very satisfactory. TCP

"shares" bandwidth with all concurrent sessions through the Additive Increase Multiplicative Decrease (AIMD) congestion control algorithm. But due to the specific non-linear AIMD dynamics, it lacks the ability to maintain a stable and smooth flow to the destination.

In lightly loaded networks with large bandwidth links, TCP provides a higher goodput since it occupies the "available" bandwidth. It is possible to restrict TCP flows to a certain level by throttling the flow window or explicitly adopting a low priority mechanism such as TCP-LP [88]. But in such methods even very small traffic bursts will result in an "underflow"; in other words, they will not be able to sustain the flow levels. In heavily loaded networks, TCP underflows more frequently since multiple losses drastically reduce congestion window size, possibly below the desired goodput level at the destination. More generally, the sawtoothed pattern in the variation of congestion window size controlled by the AIMD algorithm exhibits non-smooth dynamics that make it challenging for TCP to provide stable goodput.

1.3 Main Approaches

We propose a unified transport framework with two components: data route planner and transport control. Data route planner helps applications explore and compose a set of feasible route options and transport control suitably adjusts the source rate in response to both network and host dynamics for goodput maximization and stabilization on the constructed routes. These two components, each of which takes a different approach for a different application type, interact with each other to accomplish the tasks of data transfer.

1.3.1 Data Transfer Route Planner

1.3.1.1 Large-scale Scientific Applications

We design and develop a Network-aware Data Movement Advisor (NADMA) utility to enable automated discovery of network and system resources and advise the user of efficient

strategies for fast and successful data transfer. NADMA interacts with existing data/space management and discovery services such as Storage Resource Management [15], transport methods such as GridFTP [16], and network resource provisioning systems such as TeraPaths [17] and OSCARS [9]. NADMA acts as a route planer to explore and compose a set of feasible route options and provide them to the user along with performance estimations as well as specific steps and commands to authorize and execute data transfer. The efficacy of NADMA is demonstrated in several use cases based on its implementation and deployment in wide-area networks.

1.3.1.2 Media Streaming Applications

The purpose of overlay network topology construction is also to explore and establish data transfer paths for media data delivery. We formulate and investigate a specific type of problem to maximize the minimum node throughput in Tree Construction (max-minTC), which aims at optimizing the system's stream rate by constructing an efficient spanning tree among SPs. We consider two scenarios: (i) When the overlay network is incomplete, we prove max-minTC to be NP-complete by reducing from the Degree Constrained Spanning Tree problem and propose an efficient heuristic algorithm. (ii) When the overlay network is complete, we rigorously prove that the same heuristic algorithm yields an optimal solution to the max-minTC problem.

1.3.2 Transport Control

1.3.2.1 Large-scale Scientific Applications

The selected route by NADMA might be an Internet path or a dedicated path over network like UltrascienceNet. A major challenge for goodput maximization over wide-area dedicated connections is to compute and adapt various data rates and transport parameters automatically, whose estimates are subject to the variations due to connection and host effects as well as the finite window effects. In particular, these estimates contain seemingly stochastic components that are connection- and host-specific, and must be explicitly

accounted for to achieve high link utilization. Based on a mathematical analysis of the impact of system factors on the performance of transport protocols, we propose *Peak Link Utilization Transport* (PLUT) that incorporates a performance-adaptive flow control mechanism to regulate the activities of both the sender and receiver in response to system dynamics and automates the rate stabilization for throughput maximization using stochastic approximation methods, as opposed to the manual parameter tuning in many other UDP-based transport protocols in high-performance dedicated networks. We also explore the use of multiple parallel connections to improve the performance of a TCP-based transport method. To the best of our knowledge, a very limited number of efforts have been made in employing parallel UDP connections for data transfer in dedicated networks. We propose Parallel PLUT (Para-PLUT) to further improve the performance of PLUT by using parallel UDP connections to take advantage of the full power of multi-core processors. We conduct theoretical analysis to investigate the impact of multi-core processors on the performance of transport protocols and design a rigorous approach to adaptively determine the number of parallel UDP connections for high transport performance. We conduct an extensive set of experiments on simulated and real-life networks, and both simulation and experimental results illustrate the performance superiority of our proposed algorithms over existing ones.

1.3.2.2 Media Streaming Applications

Given an efficient overlay network topology, the design of transport protocols is important to achieve and sustain an acceptable level of quality of service (QoS). The difficulty of goodput stabilization over wide-area networks essentially arises from the randomness inherent in the end-to-end delays observed at the application level, particularly over the Internet, where routing changes also significantly affect packet delays and losses other than queueing delays [136]. This randomness must be explicitly accounted for in order to stabilize the goodput at destination by suitably throttling the source rate in response to network dynamics and statistics. We propose Transport Stabilization Protocol (TSP) to

provide reliable transport with stable goodput at a given target rate. TSP features a rate- or window-based flow control method using User Datagram Protocol (UDP) packet streams to transport both data and acknowledgements. The source-sending rate is dynamically controlled based on the estimates of destination goodput and loss rate at the source. Roughly speaking, the source rate is continuously adjusted to be approximately equal to the sum of goodput and loss rate estimates. We conduct an extensive set of experiments on simulated and real-life networks, and both simulation and experimental results illustrate the performance superiority of our proposed algorithms over existing ones.

1.4 Dissertation Organization

The rest of the dissertation is organized as follows:

- In Chapter 2, we provide general research background information and conduct a comprehensive survey of related work;
- In Chapter 3, we introduce the framework and components of the proposed integrated transport solution;
- In Chapter 4, we propose and develop a route planer to explore and compose a set of feasible route options, and design novel transport protocols to maximize transport performance over dedicated connections;
- In Chapter 5, we investigate the complexity of topology construction problems and tackle the problem of stabilizing transport dynamics in overlay networks;
- In Chapter 6, we present the network environmental settings and implementation details, and conduct an extensive set of performance evaluations and comparisons of the proposed transport protocols for both goodput stabilization and maximization using theoretical calculations, simulations, and real network deployments.
- In Chapter 7, we conclude our work and discuss future research directions.

1.5 Main Contributions

This dissertation has made the following contributions to the field of transport control:

1. We propose a route planer to explore and compose a set of feasible route options and provide them to the user along with performance estimations as well as specific steps and commands to authorize and execute data transfer;

2. We conduct a mathematical analysis to investigate the impact of system factors on the performance of transport protocols;

3. We propose a performance-adaptive flow control mechanism to regulate the activities of both the sender and receiver in response to system dynamics to achieve the maximum attainable goodput;

4. We propose a rate control mechanism to adjust the sending rate for goodput stabilization at the estimated maximum attainable goodput;

5. We conduct theoretical analysis to investigate the impact of multi-core processors on the performance of transport protocols, and design a rigorous approach to adaptively determine the number of parallel UDP connections for high transport performance;

6. We investigate the complexity of topology construction problems with rigorous NP-completeness proofs, and propose efficient heuristic algorithms;

7. We propose a new class of protocols capable of stabilizing a transport channel at a specified throughput level in the presence of random network dynamics;

8. We conduct an extensive set of data transfer experiments in both simulated and wide-area networks for design validation and performance valuation of transport protocols.

Chapter 2

Background Survey and Related Work

2.1 Resource Provisioning

2.1.1 Network Resource Provisioning

The importance of dedicated connections has been well recognized, and several network research projects [5–8, 10–12, 35, 51, 137, 139] are currently underway to develop such capabilities. Such dedicated channels are a part of the capabilities envisioned for the Global Environment for Network Innovations (GENI) project [18]. Furthermore, such deployments are expected to increase significantly and proliferate into both public and dedicated network infrastructures across the globe in the coming years. Evidence of this trend in production networks is reflected by Internet2 creating Multiple Protocol Label Switching (MPLS) overlays using ION [19], and ESnet offering both MPLS tunnels and dedicated Virtual Local Area Networks (VLAN) using OSCARS. Here, we provide a brief overview for several of these network research projects.

2.1.1.1 UltraScience Net (USN)

UltraScience Net is a wide-area experimental network testbed to support the development of networking capabilities needed for next-generation computational science applications [107]. USN provides dedicated high-bandwidth channels for large data transfers,

and also high-resolution, high-precision channels for fine control operations. The data plane of USN consists of four thousand miles of dual OC192 connections spanning Oak Ridge, Atlanta, Chicago, Seattle and Sunnyvale shown. These connections are switched in the core at SONET level using Ciena CD-CI switches and are provisioned at the edges using Force10 E300 switches at the Ethernet level. Dedicated channels of maximum 10 Gbps capacity can be provisioned on USN at layer-1 and layer-2 at 150 Mbps resolution. Layer-1 or SONET connections are provisioned exclusively between the CD-CI switches with capacities ranging from OC3 to OC192. Layer-2 channels are provisioned between E300 switches via SONET paths between core switches with capacities ranging from 150 Mbps through 10 Gbps. Also, layer-2 channels can be provisioned between core CD-CI switches but at lower capacities, ranging from 150 Mbps through 1 Gbps.

USN utilizes the OC192 links between Ciena SONET switches at ORNL and Chicago to realize OC21C connections of lengths 700, 1400, ..., 6300 miles by suitably switching them. At the end points, USN maps 1 Gbps Ethernet onto OC21C, thereby realizing 1 GigE connections of various lengths. On ESnet, 1 Gbps VLAN-tagged MPLS tunnel is set up between Chicago and Sunnyvale via Cisco and Juniper routers, which is about 3600 miles long. USN peers with ESnet in Chicago. and 1 GigE USN and ESnet connections are cross-connected using Force10 Ethernet switch. Together, this configuration provides hybrid dedicated channels of varying lengths, namely 4300, 5700, ... ,9900 miles, composed of Ethernet-mapped layer 1 and layer 3 connections.

2.1.1.2 On-demand Secure Circuits and Advance Reservation System (OSCARS)

Energy Sciences Network (ESnet) [20] provides DOE scientists with comprehensive connectivity to the global Internet and high-bandwidth access to DOE sites and primary science collaborators. ESnet shares its optical network with Internet2 [21], a U.S. national Research and Education network, on a dedicated national fiber infrastructure. The ESnet

network is comprised of two distinct core networks, an IP network and a Science Data Network (SDN). ESnet's IP network functions as a Tier 1 internet service provider to ESnet sites with direct connections with all major commercial network providers. ESnet's SDN is designed to address the growing need for guaranteed bandwidth by large-scale collaborations such as the LHC. The SDN is composed of more than 60 10 Gbps optical circuits and provides the means to dynamically provision guaranteed, high-capacity bandwidth between science facilities for DOE researchers to access time-sensitive applications and exchange large datasets.

The OSCARS [9] enables on-demand provisioning of guaranteed bandwidth virtual circuits (VCs) at layer 2 (Ethernet VLANs), and layer 3 (IP) within ESnet. OSCARS utilizes a graph-based algorithm to determine the path for a circuit reservation request. Topology and capacity information is harvested from the network devices once an hour and is then imported into the OSCARS topology database. When a new circuit reservation request is received, a base topology graph is generated from the database taking into account any existing reservations whose time ranges overlap with the new reservation. Path computation is subsequently performed on the base topology graph taking into account the parameters and constraints specified in the VC reservation, such as source and destination endpoints, bandwidth, or VLAN tagging. OSPF-TE [78], Multi-Protocol Label Switching (MPLS) [112], and Resource Reservation Protocol (RSVP) [39] are used to manage MPLS-LSPs on which the dynamic layer 2 and layer 3 VC services are built. Additionally, Label Switched Path (LDP) [38] is used to support layer 2 VCs as defined in the EoMPLS [89] protocol. The management of bandwidth usage and congestion control on each IP core and SDN link is accomplished by VC admission control and by QoS parameters.

2.1.1.3 TeraPaths

The TeraPaths [17] project at Brookhaven National Laboratory (BNL) investigates the combination of DiffServ-based LAN QoS [101] with WAN MPLS tunnels in creating end-to-end (host-to-host) virtual paths with bandwidth guarantees. These virtual paths prioritize, protect, and throttle network flows in accordance with site agreements and user requests, and prevent the disruptive effects that conventional network flows can cause in one another. TeraPaths deals with the problem of supporting efficient, reliable, predictable peta-scale data movement in modern, high-speed networks. Since the network is a shared medium, its default transport behavior is so-called ”best effort,” which essentially treats all data flows as with the same level of priority, urgency, and equal sharing of bandwidth. In a networked environment, it is not always the case that all data flows should be treated equally. Some tasks can consume more bandwidth than others, such as video or real-time device controller data, and some data must be delivered by a specific deadline. Yet in a typical environment, an email transfer containing a large file attachment could disrupt an important, multi-day data transfer. Also, it is impossible to maintain fair sharing of network resources among user groups sharing a connection (the group with more flows receives more bandwidth). It is therefore important to be able to prioritize and protect specific data flows, and to schedule network bandwidth and usage.

The service offered by TeraPaths creates end-to-end (source host computer to destination host computer) virtual paths with guaranteed bandwidth for specific data flows. This setup is realized by coordinating the configuration of the source and destination sites, which are controlled by TeraPaths instances, and by automatically interfacing with WAN provider systems to reserve bandwidth for these flows. This virtual path is tied to a specific data flow, or group of flows, and is active only while the flows are present; as such, there is no impact on other network users when the authorized data flow is inactive. TeraPaths is a fully-distributed system: each site instance needs to know only about itself and how to connect

to other TeraPaths and WAN nodes. The system is implemented entirely in Java, for portability, as a set of web service layers. Access is achieved by means of a web interface for manual reservations and an API for direct invocation from within other applications.

2.1.2 Storage Resource Provisioning

When dealing with large amounts data and storage, the scientists need to interact with multiple heterogeneous storage and file systems, each with different interfaces and security mechanisms, and to pre-allocate storage to ensure data generation and analysis tasks can take place successfully. SRMs [22] are Grid storage services providing common access interfaces to storage resources, as well as advanced functionality such as dynamic space allocation and file management on shared storage systems. SRMs is a standard specification against which multiple implementations can be developed. This approach proved to be a remarkable and unique achievement, in that now there are multiple SRMs developed in various institutions around the world that inter-operate.

SRM research has developed into an internationally coordinated effort between several DOE laboratories including Lawrence Berkeley National Laboratory (LBNL), Fermi National Accelerator Laboratory (FNAL) and Thomas Jefferson National Accelerator Facility (TJNAF), as well as several European institutions. This coordinated effort lead to the development of multiple SRMs at various institutions around the world, including BeStMan [23], CASTOR [24], dCache [25], DPM [26], and StoRM [27]. The most recent version of an SRM developed at LBNL, is called the Berkeley Storage Manager, or BeStMan [23]. BeStMan is designed in a modular fashion, so that it can be adapted easily to different storage systems (such as disk-based systems, mass storage systems, such as HPSS, and parallel file systems, such as Lustre) as well as using different transfer protocols (including GSIFTP, FTP, BBFTP, HTTP, HTTPS). BeStMan is implemented in Java in order to be highly portable. It provides all the functions related to space reservations, dynamic space allocation, directory management, and pinning of files in space for a specified

lifetime. It manages queues of multiple requests to get or put files into spaces it manages, where each request can be for multiple files or entire directories. When managing multiple files, BeStMan can take advantage of the available network bandwidth by scheduling multiple concurrent file transfers.

2.2 Topology Construction

There exist a number of tree construction algorithms in the literature with a different focus on delay, scalability, tree depth, or reliability. The tree-based overlay network topology construction algorithm is referred to as a parent selection strategy in tree management, which provides a parent the privilege to select a peer to transmit stream data. In [119], Sripanidkulchai *et al.* proposed an efficient random algorithm that requires no global topological information. Guo *et al.* [66] proposed a high-bandwidth-first algorithm, which lays out peers according to their outbound bandwidth capacities. The tree-based topology construction problem could be also considered as an optimal path selection problem for each newly joining peer with the requirement that the maximal reservable bandwidth of a feasible path is not less than the requested bandwidth. Dijkstra's or Bellman-Ford shortest path algorithms are often used for this purpose. The widest shortest path algorithm [65], the shortest widest path algorithm [125], and a utilization-based shortest path algorithm [98] were proposed to solve route computation. They assign different weight factors to limit the hop count and balance the network load. Jarvis *et al.* [75] proposed a heap algorithm to construct overlay topology by moving high-bandwidth and long-lived peers upward in the logical tree to provide better service quality.

Degree constraints are considered extensively in multicast tree construction to reduce the overall cost or latency. ZIGZAG [122] distributes media contents to many clients by organizing them into an appropriate delivery tree rooted at the server and including all and only the receivers. The join, departure, and optimization policies of the delivery tree

must follow a set of rules proposed to bound the tree height and node degree. These rules reduce the end-to-end delay from the source to a receiver, while there is no guarantee in receiver's throughput. Yang *et al.* [132] modeled the overlay multicast tree construction as an NP-complete degree-constrained minimum spanning tree problem. They used a heuristic based on Prim's algorithm to construct the initial overlay multicast tree and also proposed a proactive approach to restore overlay multicast trees in order to minimize the disruption of services due to node departures. ALMI [102] organized all peers as a minimum spanning tree where the cost of each link is an application-specific metric, which was implemented as the application-level delay between peers. SCATTERCAST [133] constructed the overlay topology based on a randomized three-step protocol, which runs a routing protocol on top of the mesh to build source-rooted distribution trees. The routing protocol used the unicast latency as its distance metric to ensure that paths in the overlay topology reflect the underlying IP topology by favoring nodes that are closer in the IP topology over more distant nodes. Merz *et al.* [100] modeled topology construction as a combinatorial optimization problem and computed spanning trees for P2P overlay networks. They proposed a distributed algorithm (TreeOpt) for spanning tree optimization to reduce the overall communication time between any pair of nodes in the graph. Small *et al.* [116] formulated the optimal multicast tree problem as a minimal delay multicast (MDM) problem and the network topology optimization problem as a minimization problem of server bandwidth cost. They proved MDM to be NP-complete and proposed several solutions to it.

Some efforts in overlay network tree optimization have a specific goal to maximize throughput. Cui *et al.* [55] investigated the problem of achieving the max-min rate allocation for all clients, which maximally utilizes the network resource, while maintaining max-min fairness. They proposed a distributed algorithm to compute the max-min rate allocation for any overlay multicast tree and proved that finding the optimal tree whose max-min rate allocation is optimal among all trees is NP-complete when the network is modeled as a complete graph. They also proposed a heuristic algorithm for overlay multicast tree

construction with an approximation ratio of 1/2. Wang *et al.* [124] studied the multi-path routing problem and proved it to be NP-complete. Zhu *et al.* [140] [141] introduced overlay networks with linear capacity constraints (LCC) and investigated two problems for widest path and maximum flow. Kim *et al.* [80] proposed an algorithm to find an optimal tree with maximum average incoming rate under two network model assumptions: (i) access links that connect hosts or LANs are bottlenecks causing congestion while backbone links are loss-free; and (ii) access links have incoming and outgoing bandwidths that do not affect each other.

The max-minTC problem in our study differs from the aforementioned ones in that we consider a set of super peers with an arbitrary topology and the optimization objective is to maximize the minimum node throughput in the tree for achieving the highest system throughput.

2.3 Transport Protocol Optimization

The network protocol is a standard procedure for facilitating data transmission between nodes geographically distributed in networks [84]. The International Organization for Standardization (ISO) has created a Reference Model of Open System Interconnection (OSI), which consists of seven layers: Application, Presentation, Session, Transport, Network, Data Link, and Physical. TCP/IP Reference Model is the most widely employed model in all computer networks, from the ARPANET to the worldwide Internet, which consists of five layers: Application, Transport, Network, Data Link, and Physical. The base function of the transport layer in the network protocol stack is to provide data transferring service to the user applications. The OSI and TCP/IP layering model defines two protocols at the transport layer: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

2.3.1 Protocols in Transport Layer

TCP is a reliable, connection-oriented, byte-stream-based protocol. TCP uses a variety of mechanisms to guarantee reliable transmission and attempts to achieve fair sharing of network resources among users, such as byte sequencing, positive acknowledgement, lost packet retransmission, congestion avoidance/recovery, and sliding-window based flow control. The TCP congestion control based on Additive Increase and Multiplicative Decrease (AIMD) algorithm unfavorably imposes a major methodological limit on transmission throughput and results in complicated end-to-end dynamics with links of high Bandwidth Delay Product (BDP). TCP increases the congestion window *cwin* by one Maximum Segment Size (MSS) every Round Trip Time (RTT), and halve its size in the presence of detected packet loss. TCP's additive increase policy is explicitly in favor of low BDP connections and TCP's throughput is inversely proportional with RTT. Furthermore, TCP detects packet loss either by timeout of an unacknowledged segment or several duplicated acknowledgements. If packet loss is caused by network congestion, TCP is able to achieve a reasonable link utilization. However, many observations have shown that packet loss is a poor indicator of network congestion, especially in high speed dedicated networks where congestion has actually been pushed to the end system. This congestion shift deludes TCP into increasing *cwin* conservatively and decreasing *cwin* aggressively, both to an excessive degree, resulting in very low application goodput.

UDP is a simpler protocol providing unreliable, connectionless, datagram-based delivery. The unit of data processed by UDP is usually called datagram. UDP offers a direct way to send and receive datagrams over networks, which makes it much faster than TCP in terms of transmission rate, but its performance suffers from transmission errors and unreliability in either one of these situations: packet lost, packet partially damaged, packet delivered out of order, and packet duplicated.

2.3.2 TCP Enhancements

In recent years, many changes to TCP have been introduced to improve its performance for high-speed networks [63]. Efforts by Kelly have resulted in a TCP variant called Scalable TCP [79]. High-Speed TCP Low Priority (HSTCP-LP) is a TCP-LP version with aggressive window increase policy targeted toward high-bandwidth and long-distance networks [87]. In some cases when interacting with certain types of traffic, it has been observed and proved that TCP could exhibit even chaotic dynamics [104, 123]. An empirical study also shows that TCP cannot adapt well to online game traffic [52], which carries similar control packets but with less transmission reliability requirements. A number of new TCP variants such as TCP Vegas [48, 49, 93, 94], Fast Active-Queue-Management Scalable (FAST) TCP [76], TCP BIC [131], and TCP CUBIC [109] have been recently proposed to reduce the AIMD dynamics by employing a smoother (linear or cubic) rate control function based on queuing delay, loss rate, or event time instead of duplicate ACKs. TCP Vegas measures timeouts were set and round-trip delays for every packet in the transmit buffer and uses additive increases in the congestion window. The FAST is based on a modification of TCP Vegas. The difference between TCP Vegas and FAST TCP lies in the way in which the rate is adjusted when the number of packets stored is too small or large. TCP Vegas makes fixed size adjustments to the rate, independent of how far the current rate is from the target rate. FAST TCP makes larger steps when the system is further from equilibrium and smaller steps near equilibrium. This improves the speed of convergence and the stability. TCP BIC is an implementation of TCP with an optimized congestion control algorithm for high speed networks with high latency. TCP BIC is used by default in Linux kernels 2.6.8 through 2.6.18. TCP CUBIC is a less aggressive and more systematic derivative of BIC, in which the window is a cubic function of time since the last congestion event, with the inflection point set to the window prior to the event. CUBIC is used by default in Linux kernels since version 2.6.19.

The Explicit Control Protocol (XCP) has a congestion control mechanism designed for

networks with a high bandwidth-delay-product (BDP) [77]. Datagram Congestion Control Protocol (DCCP) is a newly emerged UDP-like protocol that provides a congestion-controlled flow of unreliable datagrams. This protocol is particularly useful for multi-media applications that prefer timeliness to reliability, but is also not meant for stabilizing the data stream at a target rate [81]. The Stream Control Transmission Protocol (SCTP) is a new standard for robust Internet data transport proposed by the Internet Engineering Task Force [120]. Other efforts in this area are devoted to TCP buffer tuning, which retains the core algorithms of TCP but adjusts the send or receive buffer sizes to enforce supplementary rate control [61, 103, 113]. However the congestion and fairness control of these TCP enhancements still hinders the performance these protocols can achieve for high-speed dedicated networks.

Since multiple TCP connections are simply faster than a single one, the simplest way for achieving a faster transfer, without introducing a new congestion control algorithm, would be a flow that sends data over multiple connections, e.g., GridFTP [16] [36] and bbcp [28]. This approach is effective, yet suffers from several problems such as unfairness to other TCP streams, complex and inefficient multiplexing and demultiplexing data, difficulty to determine the optimal number of streams [67] [68] [95]. The MulTCP [54] protocol imitates the behavior of n standard TCP flows by having an n times higher increase factor than the standard TCP protocol (n/cwnd) and decreasing the window size by (n - 0.5)/n in case of congestion. MulTCP has an aggressive sending rate since it experiences a smaller loss rate. Probe-Aided MulTCP (PA-MulTCP) [83] is based on MulTCP, and it tries to overcome the problems of MulTCP. PA-MulTCP employs probes to determine a loss rate, which is closer to the loss rate experienced by a real TCP than the loss rate of the original MulTCP. Stochastic TCP collapses a set of parallel TCP streams into a single TCP stream by partitioning a single TCP congestion window into a set of virtual TCP streams that are stochastically managed to emulate the behavior of a collection of parallel TCP streams. MPAT [114] is a scalable algorithm for fairly providing differential services to TCP flows

that share a bottleneck link. MPAT preserves the cumulative fair share of the aggregated flows even where the number of flows in the aggregate is large. The use of multiple parallel connections to improve the performance of a TCP-based transport method has been studied in [37] [70] [69] [115].

2.3.3 UDP-based Protocols

Several UDP-based high-performance transport protocols have been developed to overcome TCP's throughput limitations for high bandwidth connections, although not necessarily optimized for dedicated connections. Such research efforts include Hurricane [126], UDT [64], FRTP [138], PAUDP [62], Tsunami [29], RBUDP/LambdaStream [71, 130], RAPID/RAPID+ [40,57], PAPTC [97], PAT [96], and many others (see [72] for overviews).

SABUL/UDT features with high performance, fairness, friendly and reliability. SABUL uses UDP to transfer data and TCP to transfer control information. UDT is an improvement over SABUL, which uses UDP only for both data and control information. SABUL uses an MIMD rate-based congestion control algorithm. UDT uses DAIMD (AIMD with decreasing increases) as its rate control algorithm, associated with a bandwidth estimation technique to determine the increase parameter, even it has the similar efficiency with MIMD, but is superior with regard to fairness. FRTP is designed for wide area circuit-switched networks. Because circuit-switched networks reserve resources in a dedicated manner, congestion control is not required during the data transfer. FRTP focuses on fixing the sending rate at an value carefully chosen in order to get the acceptable circuit utilization and low receiver buffer overflow. FRTP separates data channel and control channel through separate NICs, which means that data is transferred on the circuit while control messages are transferred over the Internet. RBUDP targets to keep link utilization as full as possible during data transfer. It aggregates acknowledgements and delivers them from receiver to sender at the end of the data transfer.

The aforementioned transport issue in shared network environments has been studied

by a number of works. LambdaStream [130] uses receiving interval as the primary metric to control the sending rate. However the interval measurement probably yields very transient values, which are inefficient in networks with high latency. In [40] [58], Banerjee *et al.* and Datta *et al.* studied feedback-based network-scheduling approaches that allow the receiver to deliver feedback to the sender. However, they cannot accurately estimate the context-switch intervals, as shown in [57], and can also lead to poor link utilization, due to their "stop-and-go" approach. Furthermore, Datta *et al.* [57] focused on dynamically monitoring the packet loss at the receiving end host, where a large number of packets may be dropped, to adapt the sending rate. But at high data rates, generating and sending packets retransmission requests at the receiver consume CPU time and may significantly interfere with the data receiving process. Banerjee *et al.* [41] investigated an approach to estimate the end system network I/O bottleneck rate by employing an off-line process (System Evaluation Process) to generate the effective bottleneck rate tables. However, the off-line generation of an accurate bottleneck rate table reflecting all host dynamics is a very time-consuming process, and the subsequent parameter selection requires active human involvement. Compared to PAPTC [97], PAT [96] further improved the overall goodput over dedicated links by employing a more aggressive flow control mechanism, which incorporates the dynamics of running processes into the calculation of maximum attainable goodput. But PAT requires extra efforts followed by trial-and-error parameter tuning to find out the optimal values of the rate increase and decrease factors for different hardware configurations. These protocols require finer manual tuning of parameters to achieve high throughput. This tuning required an intricate knowledge of implementation details, and is typically very labor-intensive and somewhat unstructured. The underlying optimizations are ad-hoc and must be repeated for each different connection, thereby incurring the efforts all over again.

Chapter 3

An Integrated High-Performance Transport Solution

In this chapter, we first present the design of an integrated transport solution, and then introduce its main functional components.

3.1 Integrated Transport Solution Design

The proposed integrated transport consists of two functional components: data route planner and transport control. These two components interact with each other to accomplish the tasks of data transfer. Data route planner helps applications explore and compose a set of feasible route options and transport control suitably adjusts the source rate in response to both network and host dynamics for goodput maximization and stabilization on the constructed routes. For large-scale scientific applications, the selected route might be an Internet path or a dedicated path over network like UltrascienceNet. Transport control maximizes bulk data transfer performance on the selected route. For media streaming applications, the route planner explores and composes a set of feasible routing paths for streaming media delivery, which could be also considered as topology construction problem. Transport control stabilizes media streaming at a desired level to achieve and sustain

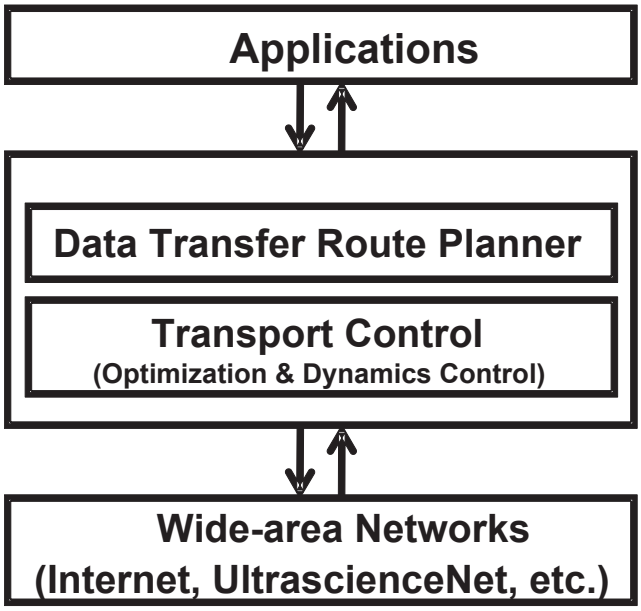


Figure 3.1: An Integrated Transport Solution.

an acceptable level of quality of service over the constructed topology. This unique transport solution has great potential to impact the science and network community by augmenting the traditional way of data transfer. Fig. 3.1 shows the framework of the proposed transport solution.

3.2 Functional Components

3.2.1 Data Transfer Route Planner

3.2.1.1 Large-scale Scientific Applications

In many cases, large-scale scientific application users, who are primarily domain experts, need to manually configure and execute their routine data-centric tasks over networks using software tools they are familiar with based on their own empirical studies, oftentimes resulting in unsatisfactory performance in the diverse and dynamic network environments. For these scientific application users, the route planner enables automated discovery of network and system resources and advises the user of efficient strategies for fast and successful data

transfer. The route planner allows the user to specify the source data by using metadata or logical names and interacts with the remote metadata services and replica location services (RLS) to identify the physical locations of these data items and stores the information in a cache for quick access. It is designed to support generic data discovery mechanisms based on web services. The route planner automatically discovers network resource provisioning systems such as TeraPaths [17] and OSCARS [9] as well as host capabilities with Storage Resource Management [15], and transport methods such as GridFTP [16]. It estimates the network performance of a transfer strategy using both an automated ICMP-based network performance measurement as well as a manual protocol-based performance measurement. At the end, the router planner explores and composes a set of feasible route options and provides them to the user along with performance estimations as well as specific steps and commands to authorize and execute data transfer.

3.2.1.2 Media Streaming Applications

Live media streaming applications often require the collective use of massively distributed network resources and therefore are not adequately supported by the traditional client-server architecture in the Internet. Peer-to-peer (P2P) overlay networks enable efficient resource sharing in distributed environments and provide a highly effective and scalable solution to this problem [13, 14, 135]. For these applications, the route planner explores and composes a set of feasible routing paths for streaming media delivery, which could be also considered as topology construction problem. The route planner finds a spanning tree whose minimum node throughput is maximized among all possible spanning trees of the given overlay networks. By maximizing the minimum node throughput in tree construction, the route planner improves the level of streaming quality for all the users, which can not be achieved by increasing the average throughput.

3.2.2 Transport Control

Transport control is an important factor in the performance of the large-scale scientific and media streaming applications involving transfer of large data sets, and streaming media, computational steering, interactive visualization, and instrument control. In general, these applications have two broad classes of networking requirements. Large-scale scientific applications need high bandwidth to move bulk data across the wide-area networks. The media streaming applications require stable bandwidth to ensure good media playback and continuous supply of streaming media. The widely deployed Transmission Control Protocol is inadequate for these tasks due to its performance drawbacks.

A main challenge in the goodput maximization over wide-area dedicated connections is to compute and adapt various data rates and transport parameters automatically, whose estimates are subject to the variations due to connection and host effects as well as the finite window effects. The transport control component regulates the activities of both the sender and receiver in response to system dynamics and automates the rate stabilization for throughput maximization using stochastic approximation methods, as opposed to the manual parameter tuning in many other UDP-based transport protocols in high-performance dedicated networks.

The difficulty of goodput stabilization over wide-area networks essentially arises from the randomness inherent in the end-to-end delays observed at the application level, particularly over the Internet, where routing changes also significantly affect packet delays and losses other than queueing delays [136]. The transport control component explicitly accounts for this randomness to stabilize the goodput at destination by suitably throttling the source rate in response to network dynamics and statistics.

Chapter 4

Maximizing Transport Performance over Dedicated Connections

In this chapter, we present how the integrated transport solution works for large-scale scientific applications. We first propose and develop a data transfer router to explore and compose a set of feasible route options and provide them to the user along with performance estimations as well as specific steps and commands to authorize and execute data transfer in Section 4.1. In Section 4.2, we propose *Peak Link Utilization Transport* (PLUT) to maximize data transfer performance over the discovered dedicated path. We will also introduce Parallel PLUT (Para-PLUT) in Section 4.3, which further improves the performance of PLUT by using multiple parallel UDP connections to take advantage of the full power of multi-core processors.

4.1 Bulk Data Transfer Route Planner

We design and develop a Network-aware Data Movement Advisor (NADMA) utility to enable automated discovery of network and system resources and advise the user of efficient strategies for fast and successful data transfer. NADMA is primarily a client-end program that interacts with existing data/space management and discovery services such as Storage Resource Management [15], transport methods such as GridFTP [16], and network resource provisioning systems such as TeraPaths [17] and OSCARS [9]. NADMA acts as

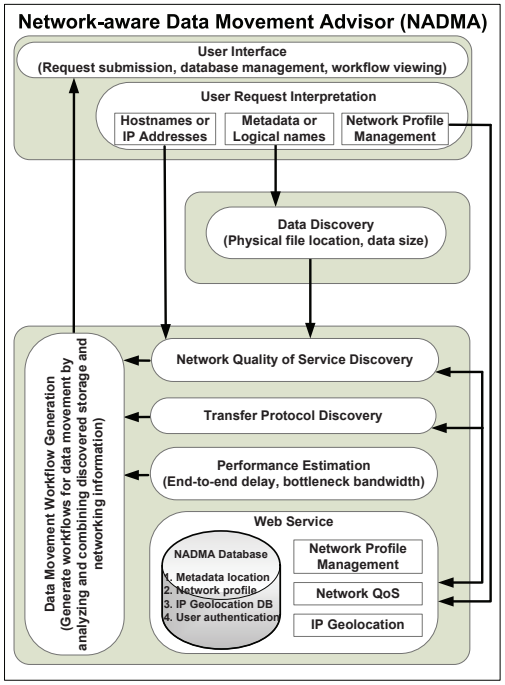


Figure 4.1: NADMA framework: functional components and control flow.

a route planer to explore and compose a set of feasible route options and provide them to the user along with performance estimations as well as specific steps and commands to authorize and execute data transfer. The efficacy of NADMA is demonstrated in several use cases based on its implementation and deployment in wide-area networks.

4.1.1 NADMA Architecture and Functional Components

As shown in Fig. 4.1, the NADMA framework consists of a User Interface with a User Request Interpretation component that interacts with other components for Data Discovery, Network Quality of Service Discovery, Transfer Protocol Discovery, and Performance Estimation to identify and develop data movement strategies through a Data Movement Workflow Generation component.

The client end of NADMA is a downloadable software program with a user interface that communicates with a lightweight web service. The NADMA client interacts with

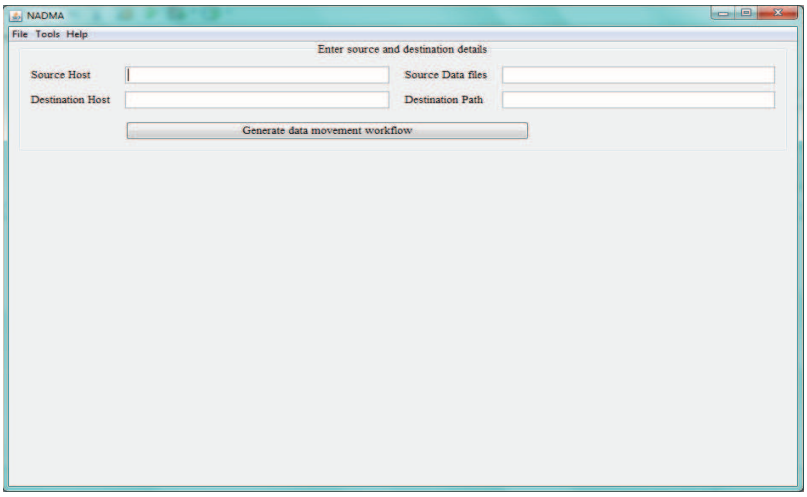


Figure 4.2: Graphical User Interface.

existing data management, discovery, and movement tools and services such as SRM and GridFTP, as well as existing network resource provisioning systems such as TeraPaths, OSCARS, and ESCPS to generate workflow-based solutions to data movement. The server end of NADMA is based on a web service and provides a client administration interface to manage a database of discovered storage and network resource provision systems.

4.1.1.1 User Interface and Request Interpretation

The User Interface provides a general means for users to submit data movement requests based on SRM, gsiftp, http, https, bbftp, scp, sftp, and so on. The interface also displays the discovered host and network information as well as the resultant data movement suggestions.

As shown in Fig. 4.2, in the main interface of NADMA, the user inputs the source and destination hosts, the data files to be transferred, and the path to which the transferred files should be saved. Both source and destination fields could be either a URL or an IP address. Note that this graphical user interface might be removed in the final version for integration with other actual data transfer tools such as GridFTP.

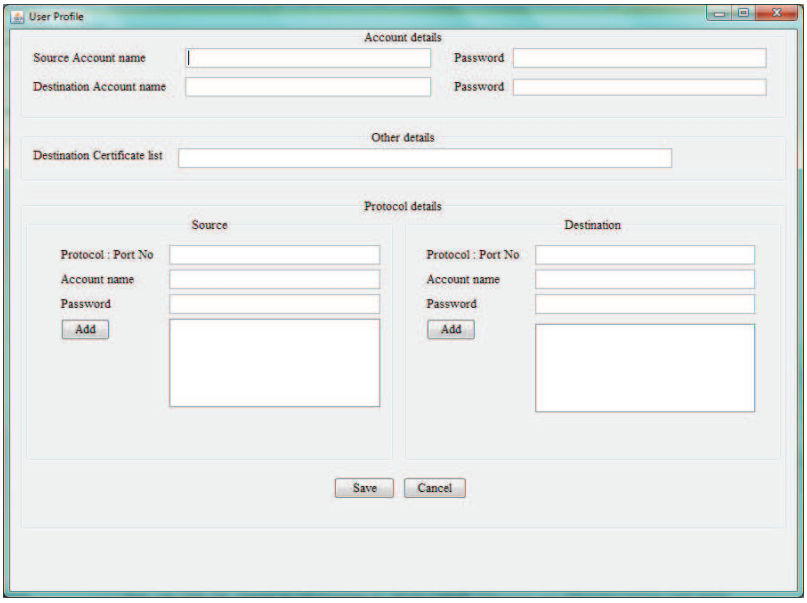


Figure 4.3: The user profile page.

To discover the system resources of the source and destinations hosts, the user may provide access information such as an account the user has on those machines, the protocols the user wishes to use for data transfer, and the user account details for each of these protocols through the user profile page as shown in Fig. 4.3. On this page, the user may also upload a list of user certificates to be used for accessing the remote host.

For each proposed data transfer solution, the corresponding network path is visually displayed on a dynamically generated map. The route information determined by NADMA is used to produce the path layout by first matching the intermediate nodes of each route against a Geolocation IP database using the NADMA web service and then placing them onto the map using Google Static Maps API.

The User Request Interpretation component is responsible for interpreting the request and invoking the corresponding functions.

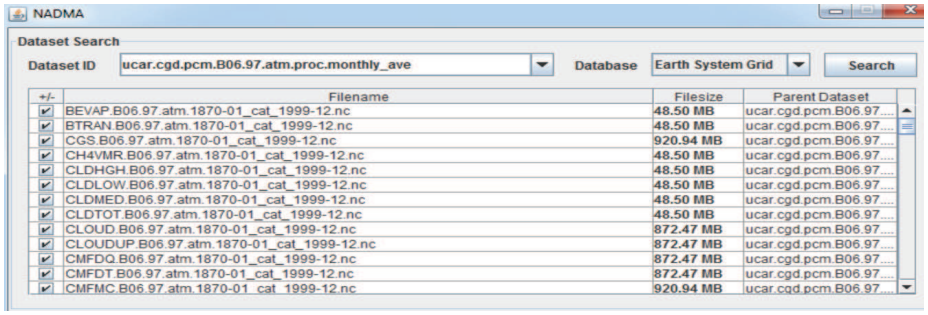


Figure 4.4: Data discovery interface for Earth System Grid.

4.1.1.2 Data Discovery

In addition to specifying a particular dataset, the user is also provided the capability of discovering the dataset of interest using the built-in data discovery component. The dataset properties including source location and dataset size are automatically determined when the user selects a target dataset from the dataset discovery component.

NADMA allows the user to specify the source data by using metadata or logical names. The data discovery component interacts with the remote metadata services and replica location services (RLS) [30] to identify the physical locations of these data items and stores the information in a cache for quick access. NADMA is designed to support generic data discovery mechanisms based on web services, and we use the Earth System Grid (ESG) data discovery web service as an example for testing purposes. The user can search for data by only providing a dataset ID, which is submitted by NADMA in a query to multiple data repositories using the available data services. The search results from those geographically located ESG gateways are assembled and presented to the user. As shown in Fig. 4.4, the location and size of each data file with the matched ID are displayed in the user interface. Other information such as physical access points is cached and may be used to determine and compose the fastest transfer option.

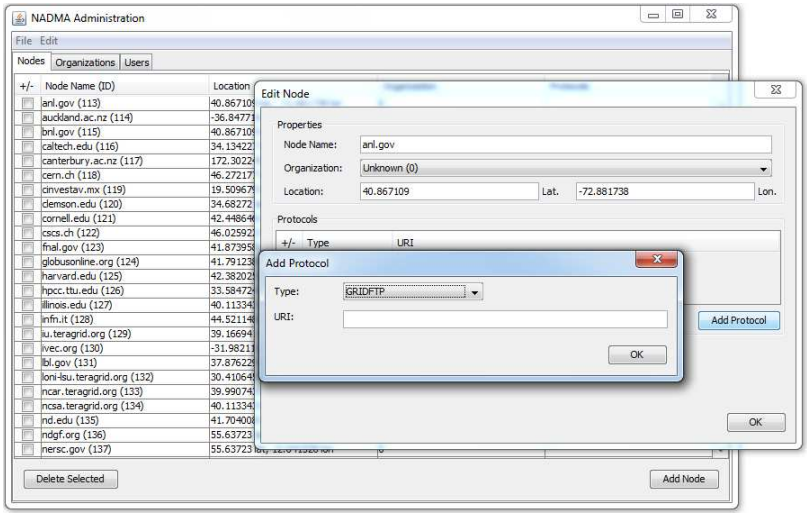


Figure 4.5: A screenshot of the administrative view of the web-service driven database.

4.1.1.3 Network Profile Management Using Web Services

To provide accurate advising for data movement in dynamic network environments, it is critical to collect and store status and resource information including network topology, provisioning services, data management and movement protocols, which must be updated in a timely manner. For this purpose, we create and maintain a database, which is accessible by regular users for information lookup and by the system administrator for information updates through web service calls. The user interface interacts with the remote centralized database to retrieve network profile information. Both client requests and server responses are encoded using Hessian binary web service protocol.

Since the network profile changes as new services are added or existing services are updated or removed, NADMA provides a set of system administration functions to add, remove, and edit these catalog entries through an administration interface. Fig. 4.5 shows an example of displaying all entries from a node table and editing a node's fields such as organization and protocols supported via a web service System Administration window. User can also choose to delete selected nodes or add additional nodes in that same interface.

4.1.1.4 Network Quality of Service Discovery

Different from best-effort IP networks, high-performance networks are capable of provisioning dedicated bandwidths. The Network Quality of Service Discovery component is designed to automatically discover network resource provisioning systems as well as host capabilities with SRM and GridFTP services. This component interacts with the NADMA web service to determine if an end host has access to the high-performance network infrastructure and necessary provisioning services. The NADMA web service queries a predefined centralized network profile database of known subnets and domains to determine the resource availability without the need to query the services directly. If the source and destination hosts do not support some advanced networking services such as SRM and GridFTP, NADMA tries to compose a data transfer path using known intermediate hosts in the same domain or subnet in order to achieve the best possible performance over the WAN.

In addition to a possible high-performance network path with reserved bandwidth, NADMA also explores a default Internet path between the source and destination nodes. If the source or destination node is a local host, NADMA identifies the path between the local host and the remote host using the existing traceroute or tracepath utility. If the system natively supports the tracepath command using UDP, it is preferred to the standard ICMP-based traceroute to minimize the use of superuser privileges.

4.1.1.5 Transfer Protocol Discovery

The Transfer Protocol Discovery component discovers transfer protocols that may be used to support the desired data transfer. NADMA scans for open ports of popular transfer protocols such as GSIFTP, HTTP, HTTPS, BBFTP, SCP, SFTP, and FTP. Special consideration is taken when initiating a third-party transfer from a remote source host to a remote destination host with respect to these protocols.

An open protocol port detected by the port scanner is considered available only if it is supported in the transfer scenario. There are three types of transfer options:

- *Local Host to Remote Host:* The protocol is considered available if the destination host port for the protocol is open.

- *Remote Host to Local Host:* The protocol is considered available if the source host port for the protocol is open and the protocol supports pull requests.

- *Remote Host to Remote Host:* The protocol is considered available if both the source host and the destination host have open ports for the protocol and the protocol allows third-party transfers.

Available transfer protocols are ordered by priority numbers according to the known performance and general security of the protocol with preference given to security-conscience massive data transfer protocols.

4.1.1.6 Bandwidth Estimation and Measurement

NADMA estimates the network performance of a transfer strategy using both an automated ICMP-based network performance measurement as well as a manual protocol-based performance measurement.

Bandwidth Estimation Using ICMP. Bing [31] is a standalone network performance measurement program that computes point-to-point throughput using two ICMP *ECHO_REQUEST* packets of different sizes between a pair of nodes. We convert Bing to a library that works on both Windows and Linux systems. The Performance Estimation component invokes the Bing library to estimate the end-to-end delay and the bottleneck bandwidth of the network path.

The bandwidth is estimated for both reserved and Internet network paths. However, there are two distinct processes used to estimate the bottleneck bandwidth depending on the type of network.

- *Reserved Network Bandwidth Estimation:* When a bandwidth reservation service such as OSCARS is available between two endpoints, a default shortest path is computed based on the ESnet topology. The minimum link capacity along the path is reported to the user together with the bandwidth estimation acquired from the Bing-based estimation from the local host to its respective OSCARS ingress endpoint, namely the intranet performance of the LAN. These two measurements are intended for the user to make an informative bandwidth reservation request to OSCARS in the future.

- *Internet Network Bandwidth Estimation:* The bottleneck bandwidth of the default Internet path between the local host and the remote host is directly measured using the Bing library. This bottleneck bandwidth provides a snapshot of the current Internet traffic condition and can also be considered as a projection on the expected performance in the near future.

Both bandwidth estimations require either the source or the destination host to be a local host due to the limitation of ICMP packets. NADMA does not perform ICMP-based bandwidth estimation for third-party transfer.

Bandwidth Estimation using Transfer Protocol. When ICMP-based bandwidth estimation is not available, NADMA offers a protocol-specific bandwidth estimation method, which can be used to select the most suitable transfer protocol. The users can initiate bandwidth measurement within NADMA to estimate bandwidth between two hosts, including two remote hosts. The bandwidth measurement based on a specific transfer protocol such as SCP or FTP involves sending a sequence of data packets of different sizes using that protocol, measuring the corresponding data transfer times, and performing a linear regression whose slope approximates the path bandwidth and intercept approximates the minimum path delay. A linear regression line that computes the bandwidth and delay of an Internet

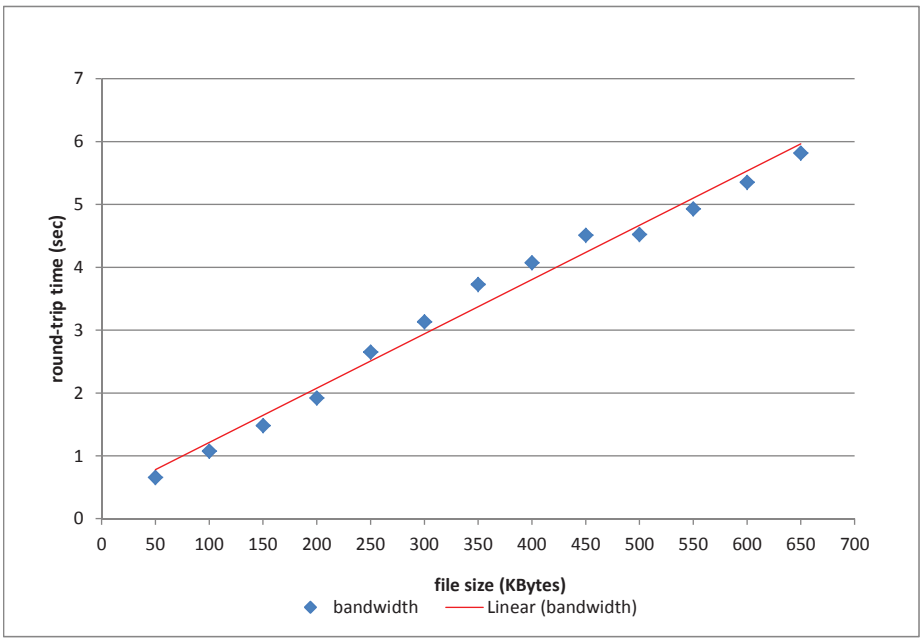


Figure 4.6: A linear regression line to estimate the path bandwidth and latency between a local SIUC node and a remote node in Connecticut: the x-axis represents the file sizes in KB, and the y-axis represents the round-trip time in seconds.

path between a local host at Southern Illinois University, Carbondale (SIUC) and a remote host in Connecticut is illustrated in Fig. 4.6 .

For instance, the FTP bandwidth measurement tool in NADMA measures the bandwidth between two FTP servers. This tool uses File eXchange Protocol (FXP) that is built into the FTP standard. It allows file copy from one FTP-server to another using an FXP-client without going through the local machine. To use the FXP operation, both servers must support and enable FXP. However, many servers do not have it enabled by default due to its potential security risks and the overhead of allowing passive mode transfers, which typically require a block of ports to be open in the system firewall.

4.1.1.7 Data Movement Workflow Generation

The Data Movement Workflow Generation component generates workflows for data movement by analyzing and combining discovered storage, host, and network resource information to build feasible data movement strategies. The estimated performance of each transfer strategy including network bandwidth and transfer duration is used to prioritize transfer workflows and make specific recommendations to the user. Each workflow-based data transfer strategy contains performance estimation, network route as well as security and transfer protocol information, and presents several specific instructional steps using third-party tools to realize the recommended transfer solution.

4.1.2 System Implementation and Operation Procedure

NADMA is implemented in Java, consisting of a client front-end and a web service back-end. The web service initiates a Hessian binary web service on top of Apache Tomcat to provide a simple and secure SSL-enabled web service. The back-end uses MySQL to store and query against a database of known advanced network resource storage and provisioning systems as well as find the geolocation of IP addresses. The client generates data movement strategies in a three-step process involving Endpoint Specification, Network Discovery, and Workflow Generation and Display.

4.1.2.1 Endpoint Specification

NADMA requires the entry of the source endpoint and destination endpoint, one of which may be a local host, for the desired data transfer. The two endpoints are sufficient for NADMA to provide a detailed analysis, but the user may optionally specify dataset characteristics and transfer constraints such as dataset size, transfer times, and preferred transfer protocols. Additional information provided by the user assists in determining routes more suitable to the request of the user.

If an endpoint is unknown, the dataset discovery tool may be used to locate the desired dataset. The dataset discovery tool interacts with metadata services to locate and retrieve information about the desired data. The user may query the target metadata service for a dataset of interest by keywords. The user selects from a list of matched datasets and NADMA populates the appropriate endpoint and dataset information automatically.

4.1.2.2 Network Discovery

The source and destination endpoints provided by the user are scanned to determine supported transfer protocols. The client application attempts to connect to the common ports of various transfer protocols to determine if the protocol is available. If the source or destination host is a local host, an Internet trace route is executed to determine an Internet path and an ICMP bandwidth test is performed.

The client queries the NADMA web service with the IP addresses and hostnames of the endpoints to determine if the endpoints have access to advanced network provisioning systems, including bandwidth reservation networks such as OSCARS as well as known SRM systems. The web service locates systems that are associated with either the endpoint hostname or a subnet of the endpoint IP address. Specific information about these systems is retrieved from the database and sent to the client application.

Geographic locations of all endpoints and intermediate nodes are determined by querying the IP address geolocation database using the web service. The client application determines the geographic location of the IP address of each endpoint and intermediate node. The city, state or province, country, postal code, and longitude and latitude are sent to the client. The client uses this information to construct a map of the topology of potential routes using the Google Static Maps API.

4.1.2.3 Workflow Generation and Display

After the network, path, and location information have been discovered, the client application generates multiple data movement workflows. The workflows are limited by the dataset characteristics and transfer preferences specified in the first step and organized by priority of transfer protocols and adherence to dataset characteristics and transfer preferences.

The client application presents a topological overview and basic network summary, including discovered route information for a high-performance network path with bandwidth reservation as well as an Internet path with basic bandwidth information. A summary of the available transfer protocols are presented with links to launch protocol-specific bandwidth tests. The tests allow for additional authentication input from the user and perform an automated bandwidth test using the given protocol.

A detailed analysis containing multiple data movement strategies is also available. The user may cycle through different workflows to view estimated performance, network path and topology, and specific steps to realize each workflow for data transfer. Each step contains general information about its function as well as specific commands that can be executed using third-party tools.

4.1.3 Case Studies

4.1.3.1 Case Study I: from University of Memphis (UM) to UM

Experimental Setup. The data source (dragon.cs.memphis.edu) and destination (mouse.cs.memphis.edu) in this use case are both located in the campus network at UM. The data source is an Intel Xeon Linux box with kernel 2.6.30, equipped with one 1 GigE NIC, Intel(R) Core(TM)2 Duo CPU, 3 GBytes of RAM, and 250 GBytes of SCSI hard drive. The data destination is a Dell Precision 490 Linux box with kernel 2.6.20, equipped with a 1 Gigabit NIC, 8 Pentium 4 processors, 4 GBytes of RAM, and 800 GBytes of SCSI

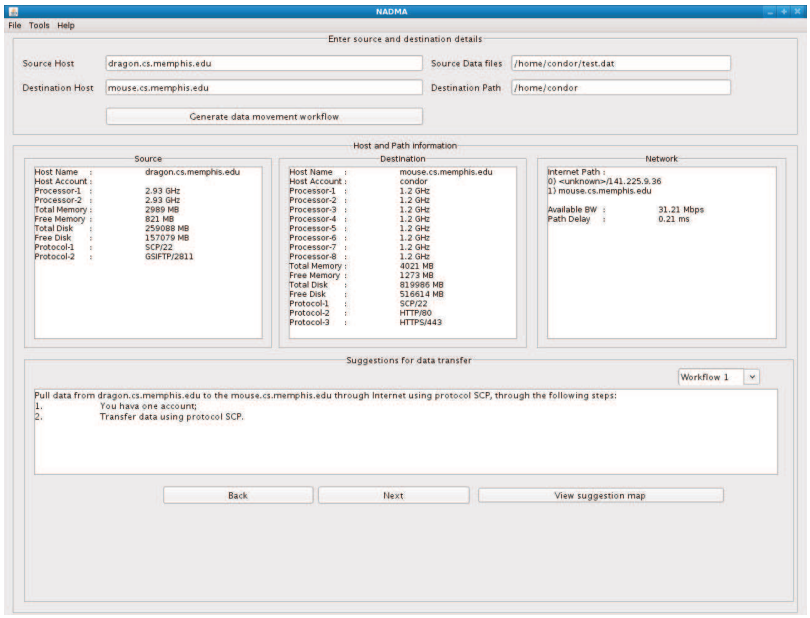


Figure 4.7: The output in the use case from UM to UM.

hard drive. The available transport protocols include SCP, HTTP and HTTPS. We have an account and password on the destination host.

Data Movement Advising Output. As shown in Fig. 4.7, the discovered host and network information is displayed in the source, destination, and network panels. The source and destination panels display the host profiles such as the host name, account, total memory, free memory, total disk, free disk, and available protocols. The OSCARS dedicated channel service is not available between these two end hosts. The network panel displays the network profile such as the Internet path, bottleneck bandwidth in Mbps, path delay in ms. In this use case, we estimate that the bottleneck bandwidth is about 32 Mbps and the path delay is around 0.2 ms. The workflow panel displays all workflow-based data transfer options, and the user has the flexibility to explore the details (e.g, path map) on each of them.

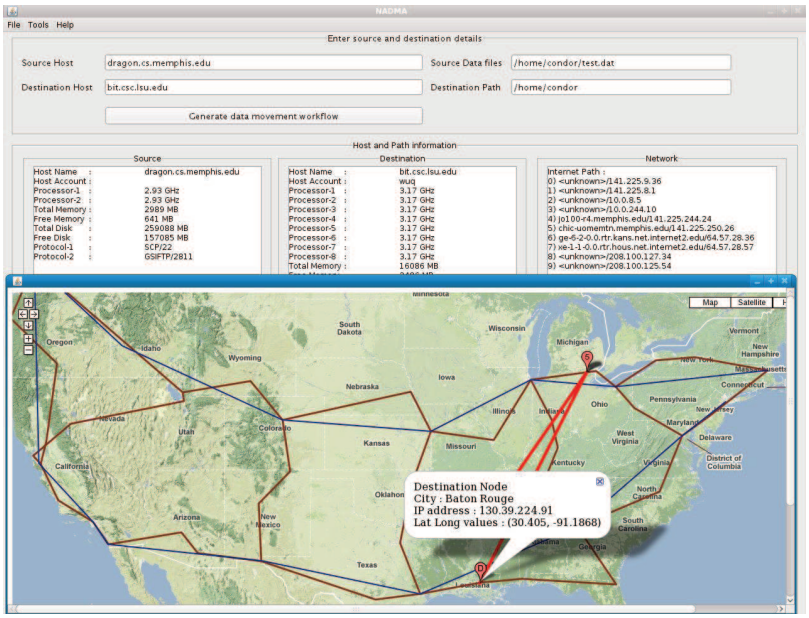


Figure 4.8: The network path in the use case from UM to LSU.

4.1.3.2 Case Study II: from UM to Louisiana State University (LSU)

Experimental Setup. In this use case, the data source (dragon.cs.memphis.edu) is the same as in Case I and the data destination (bit.csc.lsu.edu) is located in the LSU campus network. The data destination is a Linux box with kernel 2.6.20, equipped with 8 3.2 GHz processors and 16 GBytes of RAM. The available protocols include SCP and HTTP. We have an account and password on the data destination.

Data Movement Advising Output. As shown in Fig. 4.8, the discovered host and network information is displayed in the source, destination, and network panels. We estimate that the bottleneck bandwidth is about 18 Mbps and the path delay is around 28 ms. The network path corresponding to each workflow-based data transfer option is displayed on a Google map.

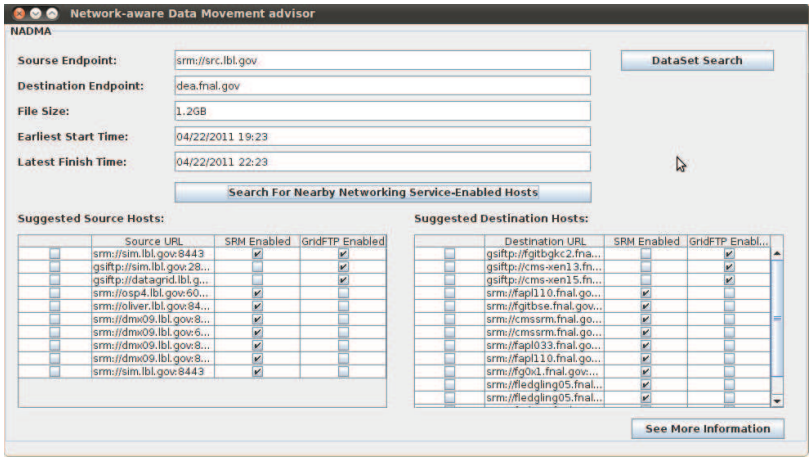


Figure 4.9: A screenshot of the list of suggested nearby hosts with advanced networking services in Case Study III.

4.1.3.3 Case Study III: from Lawrence Berkeley National Laboratory (LBL) to Fermi National Laboratory (FNL)

Experimental Setup. This use case handles an OSCARS-based high-performance data transfer request. The source host (src.lbl.gov) is a node located at LBL, Berkeley, California. The destination host (dea.fnal.gov) is located at FNL, Batavia, Illinois near Chicago.

Data Movement Advising Output. After receiving the user inputs, the NADMA system looks up the networking service profiles stored in multiple databases and generates a list of preferred source and destination URIs, which have advanced networking services installed such as SRM and GridFTP, and are close to the given endpoints. This list is sorted in a certain priority order based on the capability of performing high-performance data transfer. The given endpoints are always placed on the top of the list if they have those advanced networking services installed. Fig. 4.9 shows an example of the suggested intermediate sources and destinations close to the endpoints.

Among these suggestions, the user may choose one source and one destination based on his or her preference and accessibility. Suppose that the user has chosen (srm://sim.lbl.gov:8443)

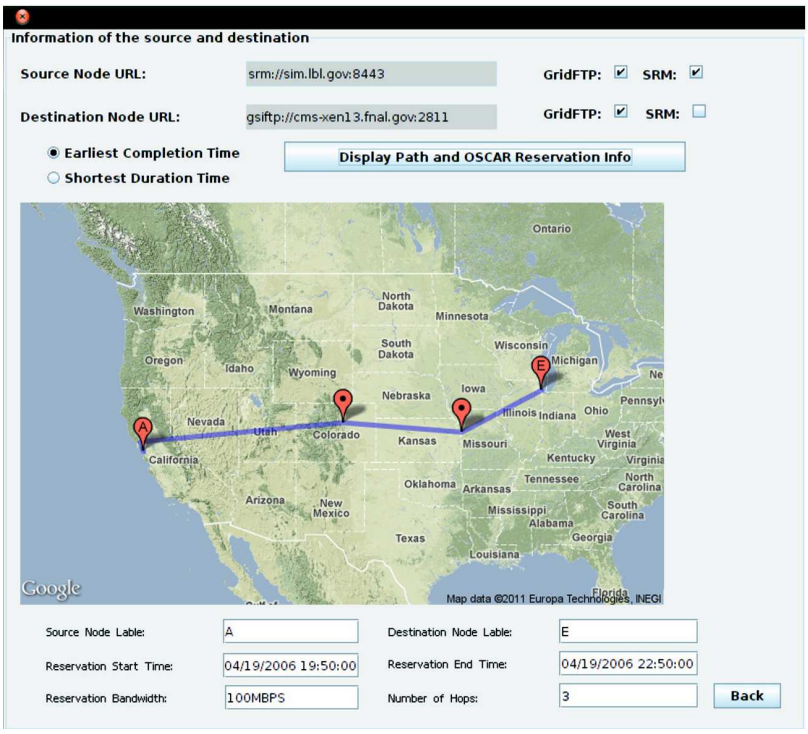


Figure 4.10: A screenshot of the map generated using Google Static Maps API to display the suggested routes and estimated performance in Case Study III.

as the source and (gsiftp://cms-xen13.fnal.gov:2811) as the destination from the suggested list. The automatic port scanner also reports that both hosts have GridFTP service running at port 2811 and SRM running at port 8443. Note that the port scanning results can be used to update the database. For any pair of selected source and destination hosts, the user can further examine the transfer route information and predict the transfer performance by interacting with OSCARS API. Based on the query results from OSCARS, the user is able to explore the best bandwidth reservation solution on this dedicated channel, either for the shortest duration or for the earliest finishing time. For illustration purposes, the map in Fig. 4.10 displays a dedicated high-performance path with intermediate routers at Kansas City and Denver, reserved bandwidth, number of hops, start time, and end time.

4.2 Peak Link Utilization Transport

Given the high bandwidth dedicated channel, inefficient transport protocols will ultimately lead to underutilized reserved bandwidth networks and significantly slower data transfer. So the design of Transport protocols are very important to achieve high link utilization and satisfy large-scale scientific applications networking requirements. In this section, we first provide a mathematical analysis to investigate the impact of system factors on the performance of transport protocols. Then we propose *Peak Link Utilization Transport* (PLUT) that incorporates a performance-adaptive flow control mechanism to regulate the activities of both the sender and receiver in response to system dynamics and automates the rate stabilization for throughput maximization using stochastic approximation methods, as opposed to the manual parameter tuning in many other UDP-based transport protocols in high-performance dedicated networks.

4.2.1 Performance Analysis of Single Packet Processing

We conduct an analytical study to investigate the impact of system factors on the performance of transport protocols. To instantiate our analysis, we consider the commonly used Linux kernel.

4.2.1.1 Packet Processing Issues

When a new packet arrives, the NIC generates an interrupt and the packet is put into the kernel buffer by the card DMA engine. In general, heavily engaging the CPU in other compute-bound tasks during an interrupt may severely hinder a running process. To avoid flooding the host system with too many interrupts, the interrupt coalescence scheme collects multiple packets and generates one single interrupt for them, therefore reducing the amount of time that the CPU would otherwise have to spend on context switching to serve multiple interrupts. The Linux kernel uses *sk_buff* structure to hold any single packet. The

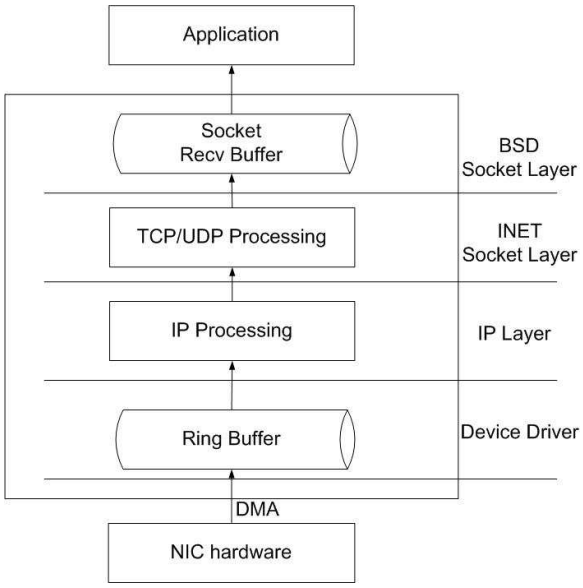


Figure 4.11: Packet processing flow in Linux.

pointers of *sk_buff* are held in a ring buffer in the kernel memory and manipulated through the network stack. If there are no free pointers in the ring buffer, incoming packets will be dropped by the kernel silently. From the ring buffer, the packets are delivered to the corresponding receiving function of the IP layer, which examines the packets for errors and then forwards them up to the INET Socket layer (such as TCP or UDP), which in turn checks for errors and copies the packets into the socket receive buffer. Then, the waiting application wakes up and returns from a corresponding receive system call that copies the data from the kernel into the application buffer. For convenience, we plot in Fig. 4.11 an overview of Linux packet processing that involves the NIC hardware, device drive, kernel protocol stack, and application [110] [111].

The Linux packet processing flow shows that packet drops by the kernel could happen in either the ring buffer, or the socket receive buffer, or both. Since the data receiving process has a lower priority than the packet processing by the kernel and the Interrupt Service Routine (ISR), packets are more likely to drop in the socket receive buffer. Although UDP is buffered on both the sender and receiver sides, we focus on the receiver side since the

receiver is under considerably more system strain than the sender. The flow control mechanism of TCP is implemented to avoid packet drops in the receive buffer. However, the UDP receive buffer might be overflowed if the packet receiving process can not acquire enough CPU cycles to consume the data in the buffer due to CPU contention. In this case, all incoming packets are discarded, hence wasting the protocol processing resources and impairing the application performance. Therefore, it is critical to employ a flow control mechanism in UDP-based transport protocols to avoid having the sender send data too fast for the receiver to receive and process.

4.2.1.2 Single Connection With a Single Data Receiving Process

We assume that the packets are of a fixed size up to the Maximum Transfer Unit (MTU) and are placed in the receive socket buffer by the kernel protocol stack at a mean Poisson rate of λ. We further assume that the effective service time is exponentially distributed and its mean is $\frac{1}{\mu}$ for packet processing carried out by the data receiving process. Therefore, the time that the data receiving process spends in processing the incoming packet and writing it to the disk is $\frac{1}{\mu}$. We denote the quantity $\frac{\lambda}{\mu}$ by α, i.e., $\alpha = \frac{\lambda}{\mu}$. Let T be the time in seconds, M be the UDP buffer size in bytes, and B be the maximum number of packets in the UDP buffer. Thus, we have:

$$B = \lceil \frac{M}{MTU} \rceil. \tag{4.2.1}$$

Since UDP was designed without transmission reliability guarantee, the default UDP receive buffer size on most operating systems is set very small. The kernel variable that defines the maximum buffer size has different values in different kernels. For example, in Linux, the variable named *net.core.rmem_max* has a default value of 131071 Bytes. Since the MTU is of 1500 Bytes in the Internet, given this default UDP receive buffer size, we have B that is equal to 88.

Based on the above assumptions, the packet receiving process that starts from the socket

receive buffer to the user application can be modeled as a Markovian birth-and-death process where the state-space is finitely limited by the buffer size [46] [47] [56], as shown in Fig. 4.12.

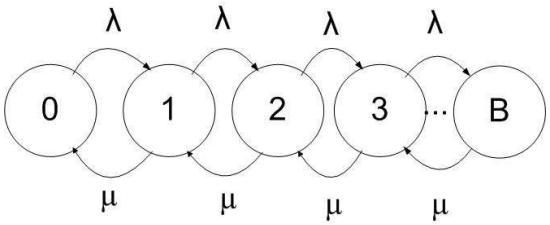


Figure 4.12: Markov state transition diagram for modeling packet processing.

In this model, the states of the process are represented by the number n of packets in the socket receive buffer. State *0* represents the state where there are no packets in the socket receive buffer, and state n, $1 \leq n \leq B$, represents the state where there are n packets in the buffer. The steady-state probability P_n of this process being in state n is given by:

$$P_n = \frac{\lambda^n}{\mu^n} \cdot P_0, \; n = 1, 2, ..., B. \tag{4.2.2}$$

Using the boundary condition:

$$P_0 + \Sigma_{n=1}^{B} P_n = 1, \tag{4.2.3}$$

we obtain

$$P_0 = \frac{1}{1 + \Sigma_{n=1}^{B} \frac{\lambda^n}{\mu^n}} = \frac{1}{1 + \alpha + \alpha^2 + ... + \alpha^B} = \frac{1-\alpha}{1-\alpha^{B+1}}, \quad \alpha \neq 1.$$

Using Eqs. (3.12) and (3.11) in [47], we obtain the steady-state probability of n packets being in the socket receive buffer:

$$P_n = \begin{cases} \frac{(1-\alpha)\cdot\alpha^n}{1-\alpha^{B+1}} & 0 \leq n \leq B, \\ 0 & n > B. \end{cases} \tag{4.2.4}$$

We plot in Fig. 4.13 this steady-state probability for $\alpha = \frac{4}{5}$. From the figure, we observe that this probability decreases over the increasing number of packets in the socket receive buffer.

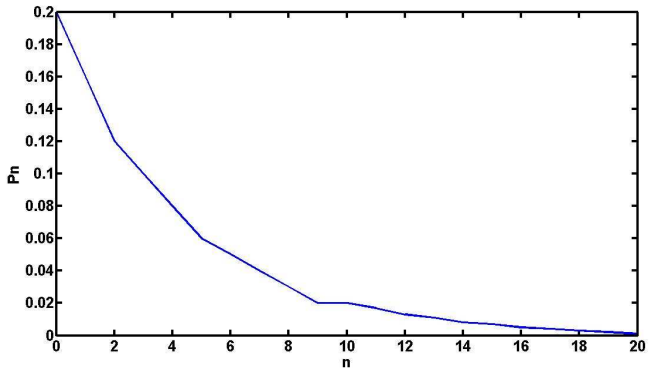


Figure 4.13: The probability of n packets in a M/M/1/B queue.

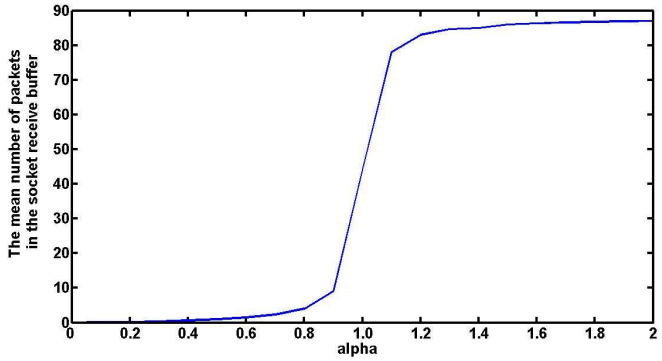


Figure 4.14: The mean number of packets in a M/M/1/B queue.

Since there are at most B packets in the socket receive buffer, this queueing system is stable for all values of λ and μ. If $\lambda = \mu$, then $\alpha = 1$ and

$$P_0 = \frac{1}{B+1} = P_n, \quad n = 1, 2, ..., B. \tag{4.2.5}$$

The mean number of packets in the socket receive buffer is given by:

$$E[n] = \begin{cases} \frac{\alpha}{1-\alpha} - \frac{B+1}{1-\alpha^{B+1}} \cdot \alpha^{B+1}, & \alpha \neq 1, \\ \frac{B}{2}, & \alpha = 1. \end{cases} \tag{4.2.6}$$

We plot in Fig. 4.14 the mean number of packets in the socket receive buffer as a function of α, which clearly illustrates the effect of the increasing number of packets due

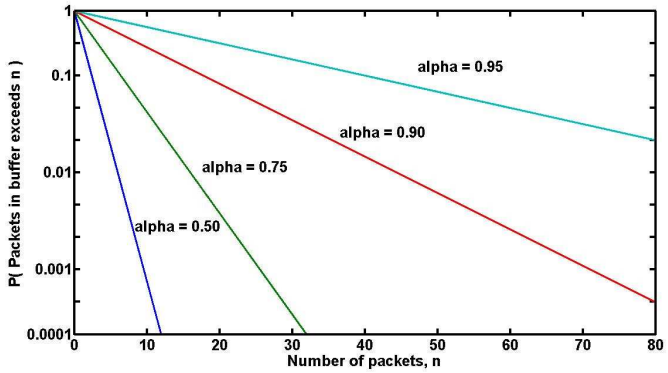


Figure 4.15: Survivor function for the number of packets for several values of alpha.

to the increase of α.

The probability of having n or more packets in the socket receive buffer is given by:

$$\begin{aligned} P(\geq n \text{ packets in buffer}) &= \Sigma_{j=n}^{B} P_j \\ &= \Sigma_{j=n}^{B} \frac{(1-\alpha)\cdot\alpha^j}{1-\alpha^{B+1}} \\ &= \frac{\alpha^n - \alpha^{B+1}}{1-\alpha^{B+1}}, \quad \alpha \neq 1. \end{aligned} \tag{4.2.7}$$

The quantity $P(\geq n \text{ packets in buffer})$ is called the survivor function for the number of packets in the socket receive buffer. We plot in Fig. 4.15 this survivor function for several different values of α. We observe that as α approaches unity, the probability that the number of packets in the receive buffer exceeds a given value increases substantially due to the small changes in α.

The utilization ρ of this queueing system is defined as:

$$\begin{aligned} \rho &= 1 - P_0 \\ &= 1 - \frac{1-\alpha}{1-\alpha^{B+1}} \\ &= \frac{\alpha - \alpha^{B+1}}{1-\alpha^{B+1}}, \quad \alpha \neq 1. \end{aligned} \tag{4.2.8}$$

By Little's law, the mean response time of this queueing system is:

$$
\begin{aligned}
R &= \frac{E[n]}{\tilde{\lambda}} \\
&= \frac{1}{\mu}[\frac{1}{1-\alpha} - \frac{B\cdot\alpha^B}{1-\alpha^B}], \quad \alpha \neq 1.
\end{aligned} \tag{4.2.9}
$$

The throughput G of this queueing system is defined as:

$$
G = \lambda \cdot (1 - P_B) = \lambda \cdot \frac{1-\alpha^B}{1-\alpha^{B+1}}. \tag{4.2.10}
$$

The time to deplete the buffer size M when the packet receiving process runs out of its time slice is given by:

$$
T = \frac{M}{\lambda}. \tag{4.2.11}
$$

On the other hand, the time to deplete the buffer size M when the CPU time is available to process the arriving packets is given by:

$$
T = \frac{M}{\lambda - \mu}. \tag{4.2.12}
$$

At time T, the UDP receive buffer is not able to accept any new packets and thus will have to drop them. The depleted UDP buffer results in the drop of UDP datagrams received by the kernel.

Eq. 4.2.10 has some important implications. (i) If $\alpha > 1$, the socket receive buffer will become full after time t, as shown in Eqs. 4.2.11 and 4.2.12. In this situation, the data receiving process is not able to consume all the packets arriving from the network, which ultimately results in packet loss in the UDP receive buffer. At high data rates, generating and sending packet retransmission requests at the receiver consume CPU time and may significantly affect the data receiving process. (ii) If $\alpha < 1$, the data receiving process has enough CPU cycles to consume packets but no enough packets are placed in the UDP receive buffer. In this situation, the UDP receive buffer could become empty and there are still idle CPU cycles, either of which is a waste of system resources.

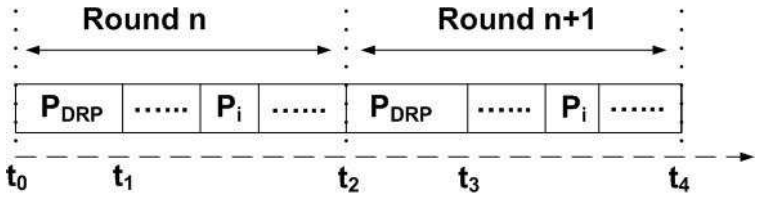


Figure 4.16: Data receiving process running model (P_{DRP} representing the data receiving process).

4.2.1.3 Mathematical Model for Data Receiving Process

Linux 2.6 is a preemptive multi-processing kernel whose scheduling policy is priority-based and is explicitly in favor of I/O bound processes in order to provide a fast process response time (interactive processes are I/O bound). Processes are initially assigned with static priorities, which can be modified dynamically by the scheduler to fulfill scheduling objectives. The Linux scheduler calculates a dynamic priority through the static priority and interactivity of the process. A process with a higher interactivity is assigned with a higher dynamic priority and hence runs more frequently. On the contrary, CPU bound processes receive a lower dynamic priority. The timeslice of a process is determined by its dynamic priority per round of execution. Thus, important processes are assigned a longer timeslice that enables these processes to run longer. The old Linux CPU scheduler recalculates each task's timeslice using an $O(n)$ algorithm implemented as a loop over each task; while the newer Linux scheduler maintains two priority arrays, an active array and an expired array, with $O(1)$ complexity for priority updating. Processes move from the active array to the expired array when they exhaust their timeslices. Recalculating all timeslices is just to switch the active and expired arrays [92] [91].

Based on the analysis of scheduling policy for Linux 2.6, the runtime behavior of the data receiving process is shown in Fig. 4.16. Let t_{DRP} and t_{EXP} be the CPU time and the expired time assigned to the data receiving process, respectively, and t_{TOT} be the total CPU

time assigned to all the running processes. We have:

$$t_{DRP} = \text{timeslice}(P_{DRP}). \tag{4.2.13}$$

$$t_{TOT} = \text{timeslice}(P_{DRP}) + \sum_{i=1}^{n} \text{timeslice}(P_i), \quad P_i \neq P_{DRP}. \tag{4.2.14}$$

The expired time for the data receiving process is:

$$t_{EXP} = t_{TOT} - t_{DRP}. \tag{4.2.15}$$

From Eqs. 4.2.13, 4.2.14 and 4.2.15, we know that the running time of the data receiving process is contingent on its own priority and the system load, which includes all interrupt-related processing and handling as well as the load of concurrent processes. Note that interrupt handling has the highest priority and is always scheduled to run before other tasks. Hence, a system with a high interrupt rate is not able to respond to the data receiving process immediately, resulting in a decreased data receiving rate. In an extreme case where the system is completely occupied for handling interrupts, the data receiving process could be temporarily suspended, resulting in significant packet losses in the socket receive buffer. Similarly, a system heavily loaded with concurrent processes could not guarantee enough CPU cycles for the data receiving process because processes with higher priorities may starve the data receiving process. To increase the data receiving rate, one needs to either increase the data receiving process' priority or reduce the system load. However, reducing the system load does not seem to be a viable solution since the data receiving process typically runs with other concurrent resource-intensive workloads in a shared computing environment.

In order to show the effects of concurrent background workloads on the performance of UDP-based transport protocols, we run iperf UDP on a local dedicated connection, which is provisioned by a back-to-back link between two Dell Precision 490 Linux boxes with kernel 2.6.20, each equipped with a 1 Gigabit NIC, dual Pentium 4 processors, 3 GBytes of RAM, and 1 TBytes of SCSI hard drive. A CPU-bound program named Burncpu is

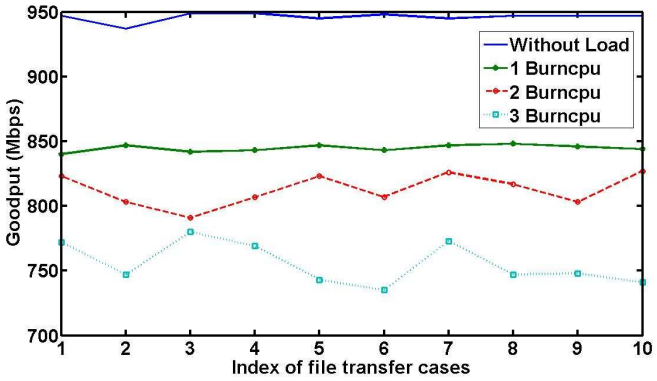


Figure 4.17: Goodput performance comparison with and without concurrent loads.

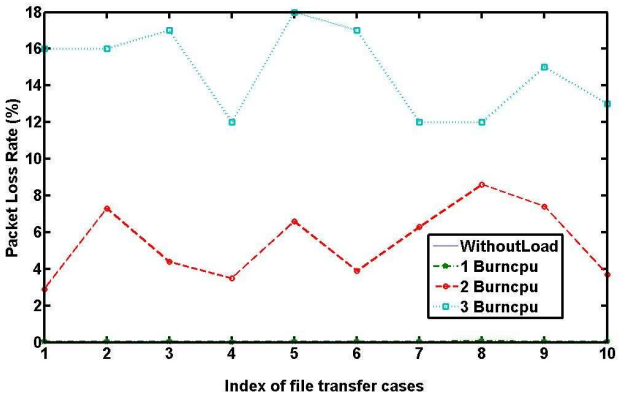


Figure 4.18: Packet loss rate with and without concurrent loads.

specifically designed and executed to emulate concurrent host background workloads. We conduct four sets of transport experiments, in each of which, 10 files are transferred using iperf. In the first set of experiments, no Burncpu process is executed while in the other three sets of experiments, 1, 2 and 3 concurrent Burncpu processes are launched, respectively.

The goodput performance measurements and packet loss rates for iperf are shown in Figs. 4.17 and 4.18. From these measurements, we observe that the amount of concurrent background workloads has a significant impact on the transport performance of iperf. The iperf UDP achieves high and stable goodputs when there are no concurrent background workloads, but yields lower goodputs of varying dynamics with 1, 2, 3 concurrent Burncpu

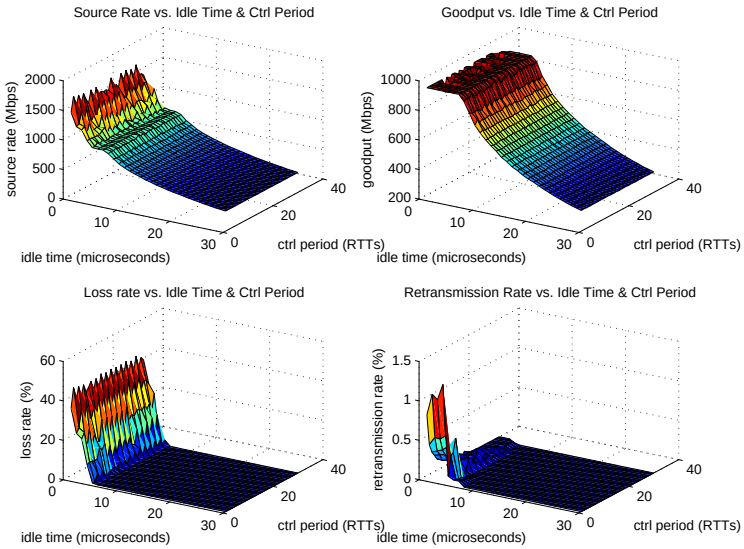


Figure 4.19: Sending, goodput, loss and retransmission profiles over 9900 mile 1 Gbps USN-ESnet hybrid connection.

processes. The iperf UDP peak goodputs without background workloads and with 3 concurrent Burncpu processes differ in about 200 Mbps, which is due to the limited CPU cycles assigned to the iperf UDP when there are competitive background workloads. In addition to having higher goodputs compared to those cases with 2 and 3 concurrent Burncpu processes, from Fig. 4.18, we also observe that the iperf UDP without background workloads and with 1 concurrent Burncpu process has almost near-zero packet loss.

4.2.2 Transport Profiles

We collected throughputs, loss rates and retransmission rates of PLUT over USN-ESnet hybrid channel as shown in Fig. 4.19, where each point in the horizontal plane corresponds to $(T(t), I(t))$. These profiles illustrate how the destination acknowledgment interval together with the source rate affects the transport performances over dedicated channels. On USN we utilize the OC192 links between Ciena SONET switches at ORNL and Chicago to realize OC21C connections of lengths 700, 1400, ..., 6300 miles by suitably switching

them. At the end points, we map 1 Gbps Ethernet onto OC21C, thereby realizing 1 GigE connections of various lengths. On ESnet, 1 Gbps VLAN-tagged MPLS tunnel is set up between Chicago and Sunnyvale via Cisco and Juniper routers, which is about 3600 miles long. USN peers with ESnet in Chicago, and 1 GigE USN and ESnet connections are cross-connected using Force10 Ethernet switch. Together, this configuration provides us hybrid dedicated channels of varying lengths, namely 4300, 5700, ... ,9900 miles, composed of Ethernet-mapped layer 1 and layer 3 connections. We also utilize USN infrastructure to provision OC192 connections of lengths 1400, 6600 and 8600 miles in loopback configurations. In each transport experiment, we employ a fixed pair of $(T(t), I(t))$ such that $T(t)$ controls the source rate and $I(t)$ determines the ACK sending rate. By varying these two parameters, we measure the corresponding source rate, destination goodput, loss rate and retransmission rate as plotted in Fig. 4.19. The ACK interval is measured in the unit of Round Trip Time (RTT) of the connection. These transport profiles provide us a revealing insight into the parameter values to be employed by the transport protocol.

The peak throughput is achieved with low loss and low retransmission rate when $T(.)$ is slightly below 10 microseconds and $I(.)$ is above 20 times the round trip time; if parameters are manually set to values within these ranges, PLUT will achieve the peak throughput of approximately 950 Mbps. These ranges are dependent on the connection as well as host parameters, and can be manually identified if various profiles are *a priori* generated. However, profile generation typically is a very time consuming process (about 8 hours for this case), and the subsequent parameter selection requires active human involvement. In Section 4.2.3, we present stochastic approximation methods to automate the process for parameter selection that bypasses both the profile generation and manual parameter tuning.

If the window size $W(t)$ is fixed to be one datagram, we have the rate-based control such that

$$r_S(t) = \frac{C_{NIC} \times MDS}{C_{NIC} \times T_i(t) + MDS}, \tag{4.2.16}$$

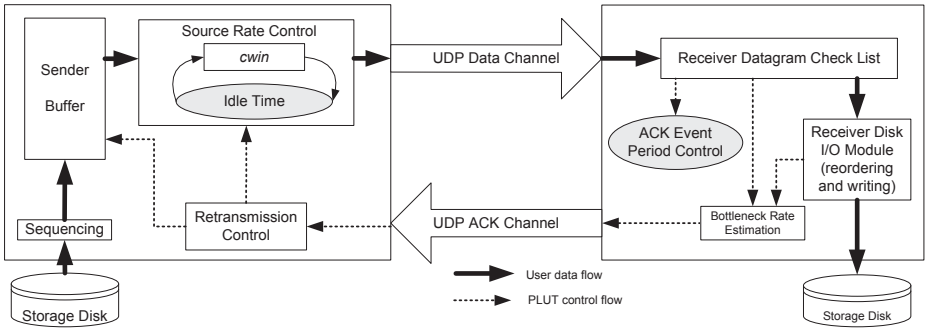


Figure 4.20: PLUT control structure.

where the rate C_{NIC} is determined by the speed of the host NIC in "writing" to connection. The statistical effects of *cwin* or $W(t)$ and idle time on transport performances were discussed in [127]. Note that the source rate calculated by Eq. 4.2.16 might differ significantly from the actual sending rate observed during file transfer. Besides the value set for idle time, other host factors such as disk I/O speed, buffer management process, and CPU scheduling policy may all affect the actual sending rate significantly. This is particularly true when high sending rates require peak processor utilization or the processor is shared with other CPU-bound applications, where the idle time itself may not be precisely enforced. Therefore, the destination goodput back-calculated from a user-specified target rate using Eq. 4.2.16 does not automatically match the target rate. Such host effects combined with variations in connection delays and finite computation periods lead to random components in the computed rates; such randomness must be explicitly accounted for in automatically tuning the parameters. We assume that errors due to these effects are bounded, which is justified by the small variations observed in our measurements.

4.2.3 Design of Peak Link Utilization Transport

4.2.3.1 PLUT Control Structure

PLUT employs a UDP-based transport control structure for disk-to-disk data transfer as shown in Fig. 4.20. The sender (source) reads data sequentially from its local storage device as a set of UDP datagrams of *Maximum Datagram Size* (MDS), each of which is assigned a unique continuous sequence number and loaded into the sender buffer. The receiver (destination) accepts the incoming datagrams in the order of their arrival and keeps track of the datagram sequence numbers in a check list. The received datagrams are immediately forwarded to a disk I/O module that handles datagram reordering if necessary and writes them to the disk in order in the background. Based on the status of the datagram checklist, a list of positive or negative acknowledgments (ACK) of lost datagrams for the interval $I(t)$ are generated and sent periodically to the sender for retransmission. The receiver's maximum attainable goodput is is dynamically estimated and sent back to the sender for source rate control.

As shown in Fig. 4.20, the data flow moves from source to destination along the solid lines and the acknowledgment feedback follows the dotted lines from destination to source. In this transport structure, there are two control operations represented by two shaded elliptic boxes: (a) source rate control through idle time and (b) ACK event interval control. The transport performance over high-speed dedicated channels critically depends on the strategies used in these control operations. In several transport control protocols, a positive acknowledgment is sent for received data packets, which is necessary for shared lossy links in Internet environments. However, dedicated channels usually provide much more reliable connections, where packet loss is much smaller than connection capacities. At high data rates, generating and sending acknowledgments at the receiver consumes CPU time and may interfere with the host receiving process. Similarly, accepting and processing acknowledgments at the sender may also affect the host sending process. To achieve peak

performance over dedicated channels, we employ a mixed acknowledgment mechanism that sends an either positive or negative acknowledgment after a carefully selected period of time. We adaptively determine appropriate delay times of mixed acknowledgments for network connections based on link and host properties.

4.2.3.2 Sender-receiver flow equations

We consider a steady state flow of packets from a sender to a receiver over a dedicated connection as shown in Fig. 4.21, wherein the time window over which various rates are computed are assumed to be sufficiently large to ignore the small window effects.

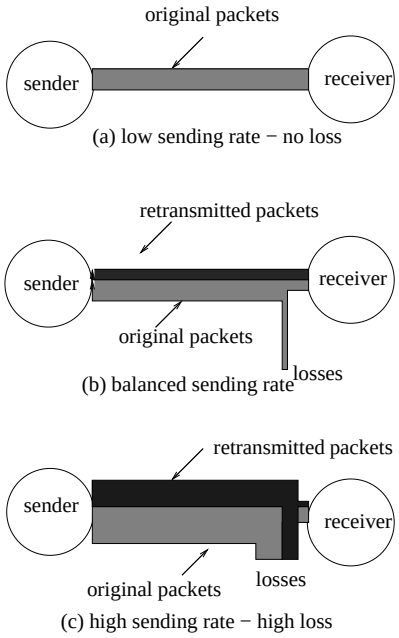


Figure 4.21: Steady-state packet flows over a dedicated connection.

In particular, rate variations due to jitter in packet delays caused by connection and host dynamics are assumed to be negligible, which is verified by our previous transport performance measurements over dedicated connections [106]. Let $r_S(t)$ be the rate at which packets are sent and let $l(t)$ be the fraction of them that are lost before being read by the receiver, and hence have to be retransmitted. Let $x(t)$ be the fraction of $r_S(t)$ that corresponds to retransmitted packets. Thus the flow $r_S(t)$ is composed of two streams of rates $g_S(t)$ and $x(t)r_S(t)$ corresponding to packets sent for the first time and retransmissions, respectively. In general the goodput $g_R(t)$ at the receiver depends on $r_S(t)$, $l(t)$ and $x(t)$, and there are three different regions:

(a) *No loss region:* Under very low sending rate, there are no losses and retransmissions as shown in Fig. 4.21(a) such that $g_R(t) = r_S(t)$, which results in low utilization.

(b) *Low loss region:* Under the peak utilization and low loss rate, the retransmitted packets

are not lost as shown in Fig. 4.21(b). Thus we have

$$\begin{aligned} g_R(t) &= g_S(t)[1-l(t)] + r_S(t)x(t) \\ &= r_S(t)[1-x(t)][1-l(t)] + r_S(t)x(t) \\ &= r_S(t)[1-l(t)+x(t)l(t)]. \end{aligned}$$

When all lost packets are replenished in each window, we have $g_S(t)l(t) = r_S(t)x(t)$ and

$$\begin{aligned} g_R(t) &= r_S(t)\left[1-[1-x(t)]\frac{r_S(t)x(t)}{g_S(t)}\right] \\ &= r_S(t)\left[1-[1-x(t)]\frac{r_S(t)x(t)}{r_S(t)[1-x(t)]}\right] \\ &= r_S(t)[1-x(t)]. \end{aligned}$$

This is an optimal region to stabilize transport since the peak utilization is achieved with a low "wasted" bandwidth due to retransmissions.

(c) *High loss region:* Under the peak utilization and high loss rate, both original and retransmitted packets can be lost as shown in Fig. 4.21(c), and we have

$$\begin{aligned} g_R(t) &= g_S(t)[1-l(t)] + r_S(t)x(t)[1-l(t)] \\ &= r_S(t)[1-x(t)][1-l(t)] + r_S(t)x(t)[1-l(t)] \\ &= r_S(t)[1-l(t)]. \end{aligned}$$

In practice, however, the above rates are computed over finite window sizes, and the packets experience non-constant delays at the destination. Typically, jitter levels are more prominent over MPLS tunnels compared to SONET circuits, and heavily-loaded, shared end hosts lead to higher delay variations compared to dedicated hosts. Consequently, the estimators $\hat{r}_S(.)$, $\hat{g}_R(.)$ and $\hat{x}(.)$ computed at discrete time points are random variables with typically unknown distributions. Their values deviate from their long term averages, and in particular, they do not exactly satisfy the equalities such as $g_R(t) = r_S(t)[1-x(t)]$ due to randomness. The effect of such randomness necessitates the utilization of stochastic approximation methods, which has a non-trivial effect on the underlying transport method: the step sizes used in parameter adaptation must be appropriately varied as per conditions such as in classical Robbins-Monro case [86]. In particular, methods that utilize fixed step sizes are not sufficient to guarantee optimal results in all but very simple cases. To take into account such random effects, we define

goodput-rate regression as $G_R(r) = E\left[\hat{g}_R(t)|r_S(t) = r\right]$, and *goodput-ACK regression* as $G_I(a) = E\left[\hat{g}_R(t)|I(t) = a\right]$. Similarly, we have *loss-fraction* and *retransmission-fraction regressions* defined as $L(r) = E\left[\hat{l}(t)|r_S(t) = r\right]$ and $X(r) = E\left[\hat{x}(t)|r_S(t) = r\right]$.

Let g^* be the maximum attainable goodput at the receiver over a given dedicated connection. The objective of PLUT control is to stabilize both $r(.)$ and $I(.)$ at suitable rates r^* and I^*, respectively, such that:

(a) $G_R(r^*) = g^* = r^*[1 - X(r^*)]$, which ensures that peak throughput is attained at low loss rate, and

(b) $G_I(I^*) = \max_I G_I(I)$, which ensures that I is optimally tuned.

We make the following assumptions about the regression functions and the underlying random process based on the measurements from Section 4.2.2:

(A.1) The goodput-rate regression $G_R(r)$ is continuous and non-decreasing in r in both zero and low loss regions, and both $L(r)$ and $X(r)$ are continuous and non-decreasing in r.

(A.2) The goodput-ACK regression $G_I(a)$ is a unimodal and differentiable function of a.

(A.3) The error magnitudes are bounded for $G_R(.)$, $G_I(.)$, $L(.)$ and $X(.)$. For example, $|\hat{g}_R(r) - G(r)| < \tau_G$ for all r and some τ_G.

(A.4) Error terms for $G_R(.)$, $G_I(.)$, $L(.)$ and $X(.)$ are not temporally correlated in the sense described in the next section using the martingale property.

We define two functions $\alpha(k)$ and $a(k)$ such that $\hat{g}_R(k) = \alpha(k)g^*$ and $\hat{r}_S(k) = a(k)g^*$.

The source rate control of PLUT at the sender is a two-step process corresponding to the above two objectives:

(a) In step one, the maximum attainable goodput is estimated by $\hat{g}^*(k)$ at time step k, and $\hat{r}_S(k)$ is stabilized to achieve this goodput at a low loss rate. This step involves the

adaptation based on monotone $G_R(.)$ and $X(.)$, which makes it suitable for Robbins-Monro [86] type stochastic approximation.

(b) In step two, the source rate is adjusted so that the measured goodput $\hat{g}_R(k)$ at the receiver stabilizes at the maximum achievable level for the given connection. This step involves the adaptation of $\hat{I}(k)$ based on unimodal $G_I(.)$, which makes it suitable for Keifer-Wolfowitz [86] type gradient descent method.

4.2.3.3 Source rate control for peak link utilization

At time step k, for the measured source rate $\hat{r}_S(k)$, measured goodput $\hat{g}_R(k)$, and measured retransmission rate $\hat{x}(k)$, the equation $\hat{r}(k) = \hat{g}(k)/[1-\hat{x}(k)]$ is only approximately satisfied. For $\hat{r}_S(k) = a(k) \cdot g^*(k)$ and $\hat{g}_R(k) = \alpha \cdot g^*(k)$, the coefficient function are typically $a(k) >= 1$ and $\alpha(k) <= 1$. Thus there are two possible estimates of $g^*(k)$ based on $\hat{r}_S(k)$ and $\hat{g}_R(k)$, which yield two different values. We consider the following general form that combines these two estimates:

$$\hat{g}^*(k) = [\hat{r}_S(k)(1-\hat{x}(k))]^{\beta} \hat{g}_R(k)^{1-\beta}, \quad 0 \leq \beta \leq 1, \tag{4.2.17}$$

where β is determined by host and link properties. Typically, $\hat{r}_S(k)$ and $\hat{x}(k)$ are more stable compared to $\hat{g}_R(k)$ since the the former are not subject to connection-level variations. For the specific case where $\alpha(k) = 1/a(k)$, we have $\hat{g}^*(k) = \sqrt{\hat{r}_S(k)\hat{g}_R(k)}$. To account for randomness in measurements and the effects of delay and its variation of sending rate $\hat{r}_S(k)$ on goodput measurement $\hat{g}_R(k)$, we apply a dynamic version of Robbins-Monro method [86,118] to adjust the source rate to achieve the target goodput $g^*(k)$ at the receiver:

$$\hat{r}_S(k+1) = \hat{r}_S(k) - \rho_k[\hat{g}_R(k) - \hat{g}^*(k)], \tag{4.2.18}$$

where the time step adjustment coefficient is given by $\rho_k = b/k^{\gamma}$ for $0.5 < \gamma < 1.0$ and $b > 0$, a suitably chosen constant. The sending rate will increase if the measured goodput $\hat{g}_R(k)$ is less than the estimated maximum attainable goodput $\hat{g}^*(k)$ at low sending rates; while in

the source rate control zone approaching the peak goodput, the goodput measurement may exceed the maximum goodput estimate due to increased retransmission rate, causing the sender to back off.

The step sizes satisfy the Robbins-Monro property namely, $\sum_{k=1}^{\infty} \rho_k = \infty$ and $\sum_{k=1}^{\infty} \rho_k^2 < \infty$. We assume that the errors satisfy the following martingale property for $\hat{r}_S(k) = r$:

$$E\left[\hat{g}(k) - \hat{g}^*(k) | \hat{r}_S(k) = r\right] = G_R(r) - [r(1 - X(r)]^{\beta} G_R(r)^{1-\beta},$$

which essentially assumes that the errors are not correlated across the time steps other than through $\hat{r}(.)$. Then the limit behavior of Eq. 4.2.18 is specified by the Ordinary Differential Equation (ODE) (Chapter 5, [86]):

$$\frac{d\hat{r}}{dt} = E\left[\hat{g}^*(k) - \hat{g}_R(k)\right] = E\left[\hat{g}^*\right] - G_R(\hat{r}).$$

Under low loss condition, we approximate

$$E[\hat{g}^*] = [\hat{r}(1 - X(\hat{r})]^{\beta} G_R(\hat{r})^{1-\beta}.$$

Then under the conditions (A.1), (A.3-4), the solution to ODE is given by the stationary point corresponding to

$$G_R(\hat{r}) \left[1 - \left(\frac{\hat{r}[1 - X(\hat{r})]}{G_R(\hat{r})}\right)^{\beta}\right] = 0,$$

which in turn corresponds to $G_R(\hat{r}) = \hat{r}[1 - X(\hat{x})] = g^*$. Thus the limit behavior of this algorithm is to stabilize at sending rate $\hat{R}_S(k) \to \hat{r}$ such that $\hat{g}_R(k) \to g^*$ as $k \to \infty$. Alternatively, the required stability property can be derived for this algorithm using the monotonic property of $G_R(.)$ and $X(.)$ to show this convergence result as in [44]. Thus, this step ensures that PLUT probabilistically stabilizes at a high utilization rate g^* of the connection while ensuring the low loss rate.

4.2.3.4 Destination ACK interval control for goodput maximization

At the destination, we adaptively adjust the ACK interval $I(k) = I^*$ such that the goodput is maximized, i.e., $G_I(I^*) = \max_I G_I(I)$. The Kiefer-Wolfowitz Stochastic Approximation (KWSA) and Simultaneous Perturbation Stochastic Approximation (SPSA) have

been shown to be very effective in solving such stochastic maximization problems whose gradient information cannot be directly obtained [117].

These two methods require collecting at least two measurements before making an adjustment on control parameters for the next time step. In transport control, however, it might be difficult to do so if the process dynamics change in the course of collecting two measurements for the gradient approximation. We apply a one-measurement form of SPSA to $I(k)$ at the receiver for goodput maximization as follows:

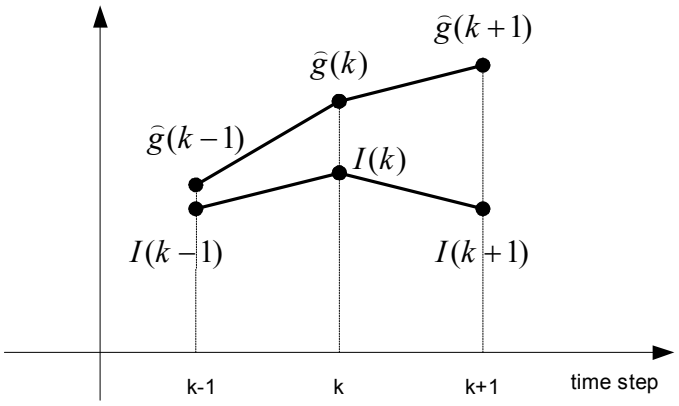


Figure 4.22: Approximation of goodput gradient in the one-measurement SPSA.

$$\hat{I}(k+1) = \hat{I}(k) - c_k \hat{\mathscr{G}}(I(k)), \tag{4.2.19}$$

where $\hat{I}(k)$ denotes the ACK interval at time step k, $\hat{\mathscr{G}}(I(k))$ denotes the SP approximation to the gradient of goodput-ACK regression $G_I(.)$ and c_k is a scalar gain coefficient such that $c_k \to 0$ as $k \to \infty$, $\sum_{k=1}^{\infty} c_k = \infty$ and $\sum_{k=1}^{\infty} c_k^2 < \infty$. As shown in Fig. 4.22, the goodput gradient in the one-measurement form of SPSA is approximated as:

$$\hat{\mathscr{G}}(I(k)) = \frac{\hat{g}_R(k) - \hat{g}_R(k-1)}{d_k \cdot (\hat{I}(k) - \hat{I}(k-1))}, \tag{4.2.20}$$

where d_k is a positive scalar.

We assume that the error process satisfies the following martingale property:

$$E[\hat{\mathscr{G}}(I)|\hat{I}(k) = I] = \hat{\mathscr{G}}(I)$$

, which is similar to the above case in that errors are not correlated in time domain except through $\hat{I}(.)$. Then under the conditions (A.2-4), the ODE specifying the limit behavior of

Eq. 4.2.19 is given by $\frac{d\hat{I}}{dt} = E[\hat{\mathscr{G}}(\hat{I})]$, which corresponds to the maximization of $G_I(.)$ [86]. Thus $\hat{I}(k) \rightarrow I^*$ as $k \rightarrow \infty$ such that $G_I(I^*) = \max_I G_I(I)$. The convergence to optimal I^* is asymptotically and probabilistically guaranteed and does not depend on the knowledge of the underlying probability distributions.

4.2.3.5 Maximum attainable goodput estimation for rate adjustment

In our previous analysis, we ignored the impact of the concurrent background workloads, which vary during the data transfer period. As more background workloads are added, t_{TOT} increases. From Eqs. 4.2.13, 4.2.14, and 4.2.15, we know that this will incur a larger expired time for the data receiving process and vice versa. The probability of processes changes in the running environment might result in a changing maximum attainable goodput $\hat{g}^*(k)$ during the data transfer. In this case, we are more interested in incorporating the dynamics of running processes into the calculation of $\hat{g}^*(k)$. Employing a mechanism in PLUT such that the sender could suitably adjust its sending rate in response to dynamics of the receiver is essential to maximize the overall goodput over dedicated links. The experiences gained and lessons learned from the design of PAPTC and PAT enable us to propose a more stable and efficient flow control scheme by removing the tuning process for PAT's rate increase and decrease factors.

PLUT maintains a table GT at the receiver, which holds the estimated maximum attainable goodputs represented as a function of the number of CPU-bound processes bl_{CPU} and I/O-bound processes bl_{IO}. This table is initiated as empty and dynamically updated during the process of data transfer. The change of the number of running processes at the receiver triggers the goodput estimation process to calculate the new maximum attainable goodput. Then the sending rate is adjusted for goodput stabilization at the estimated maximum attainable goodput using the aforementioned stochastic approximation methods.

Suppose at time step k, the number of running processes is changed. Let t be the time period in seconds between time step k and $k+1$, $fb(k)$ and $fb(k+1)$ be the size of the free

buffer at time step k and $k+1$, and $fb = fb(k+1) - fb(k)$. Then fb is the size of free buffer depleted in the amount of time t after the change of background workloads, which is given by:

$$fb = (\hat{g}^*(k+1) - \hat{g}^*(k)) * t. \tag{4.2.21}$$

Rearranging, we see that the new maximum attainable goodput $\hat{g^*}(k+1)$ can be updated by the following equation:

$$\hat{g}^*(k+1) = \hat{g}^*(k) + \frac{fb}{t}. \tag{4.2.22}$$

$fb > 0$ means the maximum attainable goodput needs increasing; $fb = 0$ means the goodput needs no adaption; and $fb < 0$ means its decrease.

In practice, we can sample the background loads $bl_{IO}(k)/bl_{CPU}(k)$, the goodput $\hat{g}_R(k)$, and the free buffer size $fb(k)$ at an carefully selected interval Δ. We denote such sequences of bl_{IO}, bl_{CPU}, and $\hat{g}_R(k)$ samples as $< bl_{IO}(k) >= ...bl_{IO}(k-1)bl_{IO}(k)bl_{IO}(k+1)...$, $< bl_{CPU}(k) >= ...bl_{CPU}(k-1)bl_{CPU}(k)bl_{CPU}(k+1)...$, $< \hat{g}_R(k) >= ...\hat{g}_R(k-1)\hat{g}_R(k)\hat{g}_R(k+1)...$, and $< fb(k) >= ...fb(k-1)fb(k)fb(k+1)....$ For the k-th control interval, if the value of $bl_{IO}(k)/bl_{CPU}(k)$ is different from the value of $bl_{IO}(k-1)/bl_{CPU}(k-1)$, the maximum attainable goodput needs to be updated and sent back to the sender in the acknowledgement. Let C be the link capacity, which is known in advance with dedicated channels, M be the size of buffer allocated at the destination. The flow control follows these steps:

(1) Initialize parameters:

$\hat{g}^*(0) \leftarrow C$,

$fb(0) \leftarrow M$,

$bl_{IO}(0) \leftarrow 0$,

$bl_{CPU}(0) \leftarrow 0$,

$GT \leftarrow 0$

(2) The sender transmits data with the rate equal to the link capacity C, which is known in advance with dedicated channels;

(3) Compute the difference of background workloads $bl_{IO}(k-1)/bl_{CPU}(k-1)$ and $bl_{IO}(k)/bl_{CPU}(k)$; if there is difference, then proceed to Step (4); otherwise, proceed to Step (6);

(4) Check the maximum attainable goodput table GT, if $GT(bl_{IO}(k), bl_{CPU}(k))$ is not equal to 0, set

$$\hat{g}^*(k) \leftarrow GT(bl_{IO}(k), bl_{CPU}(k)), \tag{4.2.23}$$

and send it back to the sender; otherwise, the maximum attainable goodput $\hat{g}^*(k)$ is adjusted in the proportion of current CPU cycles assigned to the data receiving process by the Eq. 4.2.22, set

$$GT(bl_{IO}(k), bl_{CPU}(k)) \leftarrow \hat{g}^*(k), \tag{4.2.24}$$

and send it back to the sender;

(5) The source rate $\hat{r}_S(k)$ is stabilized to achieve this goodput at a low loss rate by the Robbins-Monro Stochastic Approximation (RMSA) algorithm and the acknowledgment interval $\hat{I}(k)$ is adapted to maximize the application goodput at the receiver based on the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm.

(6) Return to Step (3).

4.3 Parallel Peak Link Utilization Transport

When a large number of packets arrive in the receive buffer, multiple threads or processes can process these packets in parallel to achieve a high aggregate throughput. Therefore,

employing an efficient parallel mechanism in UDP-based transport protocols is critical to improving application goodput and resources utilization. To the best of our knowledge, a very limited amount of efforts have been made in employing parallel UDP connections for data transfer in dedicated networks.

In this section, we first conduct theoretical analysis to investigate the impact of multi-core processors on the performance of transport protocols and design a rigorous approach to adaptively determine the number of parallel UDP connections for high transport performance. Then we propose Parallel PLUT (Para-PLUT) to further improve the performance of PLUT by using multiple parallel UDP connections to take advantage of the full power of multi-core processors. Currently Para-PLUT does not account for the impact of the background workloads, which vary during the data transfer period.

4.3.1 Performance Analysis of Multiple Packet Processing

The UDP-based protocols without a multiple data receiving scheme employ a single thread to process all incoming packets and therefore exhibit a low rate in resource utilization on the end system. These under-utilized system resources could be used to further improve application throughput. We conduct an analytical investigation into the impact of multiple data receiving processes on the performance of transport protocols.

4.3.1.1 Single Connection With Multiple Data Receiving Processes

In this case, one UDP connection is built to do the data transfer just like the case in subsection 4.2.1.2. But there are multiple data receiving processes running in parallel to consume data arriving from high-speed links as shown in Fig. 4.23. In this subsection, we provide an initial analysis of the impact of this model on the data transfer performance.

A data receiving process can only handle one packet at a time and hence, it is either in a ”busy” or an ”idle” state. If all data receiving processes are busy upon the arrival of a packet, the newly arriving packet is buffered, assuming that socket receive buffer space is

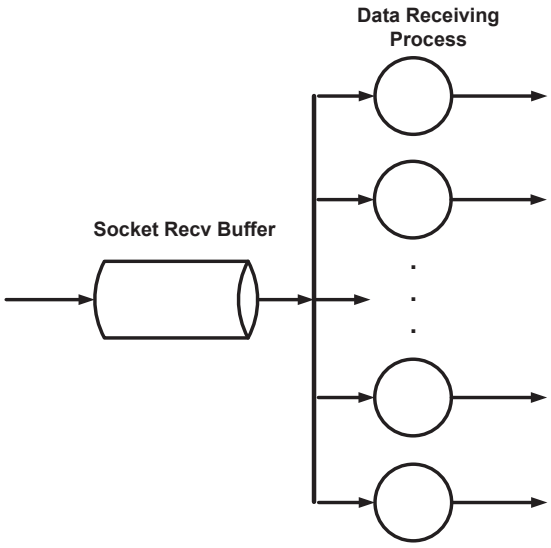


Figure 4.23: Single connection with parallel data receiving.

available. When the packet currently in process is finished, one of the waiting packets is selected for service according to a queueing discipline.

We assume that the packets are of a fixed size up to the Maximum Transfer Unit (MTU) and are placed in the socket receive buffer by the kernel protocol stack at a mean Poisson rate of λ. We further assume that the effective service time is exponentially distributed and its mean is $\frac{1}{\mu}$ for packet processing carried out by each data receiving process. Let m be the number of data receiving process, and M be the UDP buffer size in bytes and B be the maximum number of packets in the UDP buffer, and $B > m$.

We denote the quantity $\frac{\lambda}{(m\mu)}$ by α, i.e., $\alpha = \frac{\lambda}{(m\mu)}$ and the quantity $\frac{\lambda}{(\mu)}$ by ρ, i.e., $\rho = \frac{\lambda}{(\mu)}$. Based on the above assumptions, this packet processing flow that employs multiple data receiving to consume arriving data is a M/M/m/B queuing system which can be modeled as a death birth process [46] [47] [56] [45], as shown in Fig. 4.24.

In this model, the states of the process are represented by the number n of packets in the socket receive buffer. State *0* represents the state where there are no packets in the socket receive buffer, and state n, $1 \leq n \leq B$, represents the state where there are n packets in the buffer. The state-space is finitely limited by the buffer size.

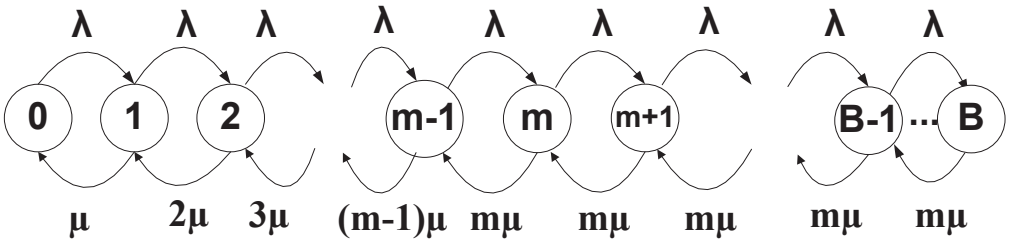


Figure 4.24: M/M/m/B state transition diagram for modeling packet processing.

The steady-state probability P_n of this process being in state n is given by:

$$P_n = \begin{cases} \frac{\lambda^n}{n!\mu^n} \cdot P_0 & 0 \leq n \leq m-1, \\ \frac{\lambda^n}{m!m^{n-m}\mu^n} \cdot P_0 & m \leq n \leq B, \\ 0 & n > B. \end{cases} \tag{4.3.1}$$

And, because $\alpha = \frac{\lambda}{(m\mu)}$,

$$P_n = \begin{cases} \frac{(m\alpha)^n}{n!} \cdot P_0 & 0 \leq n \leq m-1, \\ \frac{\alpha^n m^m}{m!} \cdot P_0 & m \leq n \leq B, \\ 0 & n > B. \end{cases} \tag{4.3.2}$$

Using the boundary condition:

$$P_0 + \Sigma_{n=1}^{B} P_n = 1, \tag{4.3.3}$$

we obtain

$$P_0 = \begin{cases} \frac{1}{1+\frac{(1-\alpha^{B-m+1})(m\alpha)^m}{m!(1-\alpha)}+\sum_{n=1}^{m-1}\frac{(m\alpha)^n}{n!}} & \alpha \neq 1, \\ \frac{1}{1+\frac{m^m}{m!}(B-m+1)+\sum_{n=1}^{m-1}\frac{(m)^n}{n!}} & \alpha = 1. \end{cases} \tag{4.3.4}$$

The number of packets n in the queuing system is defined as the number of packets in queue n_q plus that of in service n_s. The mean number of packets in the queuing system is given by:

$$E[n] = \sum_{n=1}^{B} nP_n. \tag{4.3.5}$$

The mean number of packets in the socket receive buffer is given by:

$$E[n_q] = \sum_{n=m+1}^{B} (n-m)P_n. \tag{4.3.6}$$

The probability of having n or more packets in the socket receive buffer is given by:

$$\begin{aligned} P(\geq n \text{ packets in buffer}) &= \Sigma_{j=n}^{B} P_j \\ &= \Sigma_{j=n}^{B} \frac{(1-\alpha)\cdot\alpha^j}{1-\alpha^{B+1}} \\ &= \frac{\alpha^n - \alpha^{B+1}}{1-\alpha^{B+1}}, \quad \alpha \neq 1 \end{aligned} \tag{4.3.7}$$

Because arrivals of packets are constrained by the size of socket receive buffer, packets can be received only when buffer is available. The effective packet arrival rate $\tilde{\lambda}$ is less than λ:

$$\begin{aligned} \tilde{\lambda} &= \sum_{n=0}^{B-1} \lambda P_n \\ &= \lambda(1-P_B). \end{aligned} \tag{4.3.8}$$

And the difference $\lambda - \tilde{\lambda}$ is the packet loss rate.

The throughput G of this queuing system is defined as the number of packets processed per unit time.

$$\begin{aligned} G &= \mu \sum_{j=1}^{m-1} jP_j + mu \sum_{j=m}^{B} P_j \\ &= \lambda(1-P_B). \end{aligned} \tag{4.3.9}$$

By Little's law, the mean response time of this queuing system is:

$$R = \frac{E[n]}{\tilde{\lambda}}.$$

4.3.1.2 Parallel Connections With Multiple Data Receiving Processes

In this case, multiple UDP connections are built to do the data transfer and each of these UDP connections creates a data receiving process. These multiple data receiving processes

running in parallel to consume data arriving from high-speed network. In this subsection, we provide theoretical analysis on how the use of parallel UDP connections affects the transport performance and bandwidth utilization.

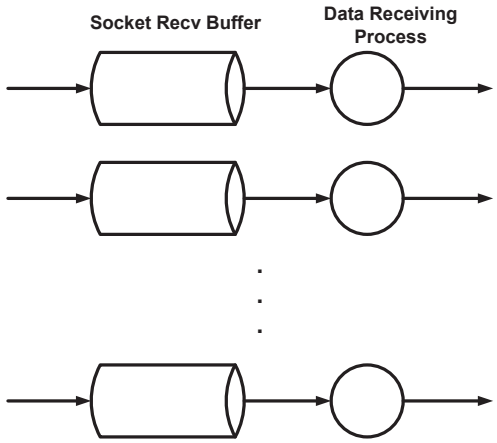


Figure 4.25: Parallel connections each with a single data receiving process.

We assume that m parallel UDP connections are established to transfer data in a single file transfer. Each of these UDP connections creates a data receiving process to consume data arriving from high-speed links at the rate of μ as shown in Fig. 4.25. Based on the above assumptions on packet arrival and consumption, each packet receiving flow starting from the socket receive buffer to the user application is an M/M/1/B queueing system with an arrival rate of λ/m. This aggregated system can be modeled as a special case of the death/birth process, whose Markov state transition diagram is illustrated in Fig.4.26.

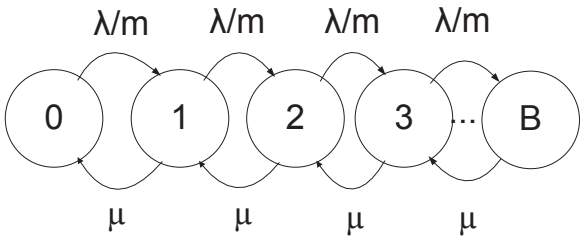


Figure 4.26: Markov state transition diagram for modeling packet processing.

By replacing the arrival rate in the equations derived in subsection 4.2.1.2 with λ/m, we obtain the steady-state probability, mean response time, and waiting time of this aggregated queueing system as follows.

$$P_0 = \frac{1-(\frac{\alpha}{m})}{1-(\frac{\alpha}{m})^{B+1}}, \quad \alpha \neq 1.$$

Using Eqs. (3.12) and (3.11) in [47], we obtain the steady-state probability of n packets being in the socket receive buffer:

$$P_n = \begin{cases} \frac{(1-(\frac{\alpha}{m}))\cdot(\frac{\alpha}{m})^n}{1-(\frac{\alpha}{m})^{B+1}} & 0 \leq n \leq B, \\ 0 & n > B. \end{cases} \tag{4.3.10}$$

The utilization ρ of this queueing system is defined as:

$$\rho = \frac{(\frac{\alpha}{m})-(\frac{\alpha}{m})^{B+1}}{1-(\frac{\alpha}{m})^{B+1}}, \quad \alpha \neq 1.$$

By Little's law, the mean response time of this aggregated queueing system is:

$$R = \frac{1}{\mu}[\frac{1}{1-\frac{\alpha}{m}} - \frac{B\cdot(\frac{\alpha}{m})^B}{1-(\frac{\alpha}{m})^B}], \quad \alpha \neq 1. \tag{4.3.11}$$

The throughput G of this aggregated queueing system is defined as:

$$G = \lambda \cdot \frac{1-(\frac{\alpha}{m})^B}{1-(\frac{\alpha}{m})^{B+1}}, \quad \alpha \neq 1. \tag{4.3.12}$$

4.3.1.3 Comparison of Single and Parallel Connections

We conduct further comparative analysis on single and parallel UDP-based transport protocols to show the difference in their performance such as throughput and mean response time. These analysis results provide us a deep insight into the benefits of using parallel connections and also help us determine the optimal number of parallel connections for a given system.

Since UDP was designed without transmission reliability guarantee, the majority of operating systems have a very small default UDP receive buffer size. For example, in most

Linux implementations, the default UDP receive buffer size is 131071 bytes. Given the MTU of 1500 bytes in the Internet, with this default UDP receive buffer size, we have *B=88*. In general, the packet processing capacity of the kernel protocol stack is much larger than that of the application. Suppose that the largest packet receiving rate μ of each data receiving process is 2 Gpbs and the number *m* of data receiving processes is 3. Based on Eqs. 4.3.10, 4.2.10, 4.3.11, and 4.3.12, we obtain the performance comparison of mean response time and throughput between the single M/M/1/88 queueing system and the combined system with 3 aggregated M/M/1/88 queueing systems in response to increasing packet arrival rate λ, as shown in Figs. 4.27 and 4.28.

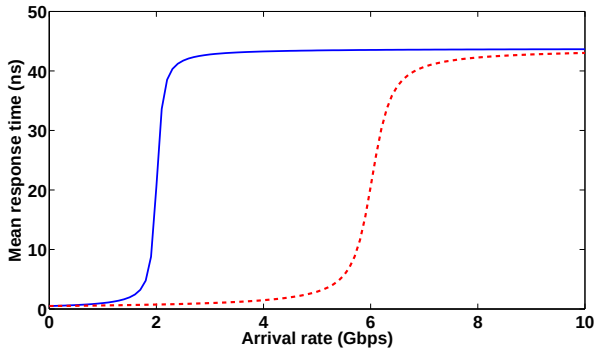


Figure 4.27: Mean response time comparison with a service rate of 4 Gbps.

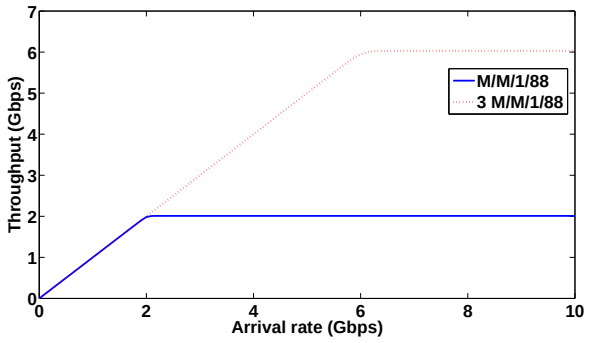


Figure 4.28: Throughput comparison with a service rate of 4 Gbps.

From these performance comparison curves, we observe some important features and

their implications. (i) When $\lambda < \mu$, these two methods have the same throughput and a very small difference in mean response time. The data receiving process in both methods has sufficient CPU cycles to consume packets but no enough packets are placed in the UDP receive buffer. Under this circumstance, the UDP receive buffer could become empty and there are still idle CPU cycles, either of which is a waste of system resources. (ii) When $\mu \leq \lambda < 3\mu$, the single M/M/1/88 queueing system has a much larger mean response time but a much smaller throughput than that of the combined M/M/1/88 queueing system. This is because the packet arrival rate is beyond the processing capability of a single data receiving process even if there is unused computing resources available at the receiver. The combined M/M/1/88 queueing system is able to obtain a higher throughput by aggregating multiple UDP connections to utilize the under-utilized computing resources. This observation strongly indicates that the presence of multi-core processors offers a great opportunity to achieve better transport performance by executing multiple threads at different processing cores. (iii) When $\lambda \geq 3\mu$, the mean response time of the combined M/M/1/88 queueing system increases quickly and approaches that of the single M/M/1/88 queueing system. Meanwhile, there is no more throughput improvement because the packet arrival rate is beyond the system processing capability, which is limited by the total computing power of the receiver host.

4.3.2 Para-PLUT Control Structure

Para-PLUT employs a UDP-based transport control structure for disk-to-disk data transfer as shown in Fig. 4.29. The data flow moves from source to destination along the solid lines and the acknowledgment feedback follows the dotted lines from destination to source. The sender (source) reads data sequentially from its local storage device as a set of UDP datagrams of Maximum Datagram Size (MDS), each of which is assigned a unique continuous sequence number and loaded into the sender buffer. Para-PLUT opens multiple connections between the sender and the receiver, divides the entire set of datagrams into a number

of partitions, and assigns them proportionally to different UDP data channels based on their throughput measurements for parallel transfer. On a computer with multi-core processors, when the number of UDP data channels is larger than the number of CPU processors, Para-PLUT performs parallel data receiving to increase the receiver throughput by making full use of the multi-core processors. Para-PLUT regulates the source sending rate of each UDP data channel by a pair of congestion window $W(t)$ and sleep or idle time (i.e., inter-window delay) $T(t)$ as in PLUT [129].

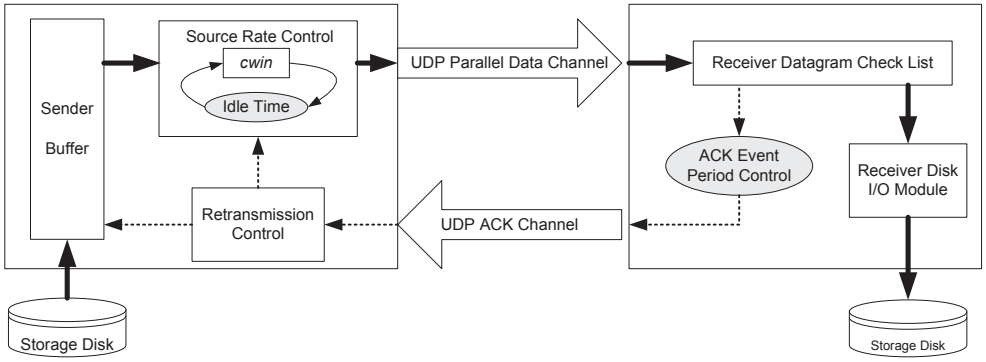


Figure 4.29: Transport control structure for disk-to-disk data transfer.

Each UDP data channel accepts incoming datagrams in the order of their arrival and keeps track of the datagram sequence numbers in a shared checklist. The received datagrams are immediately forwarded to a disk I/O module that handles datagram reordering if necessary and writes them to the disk in order in the background. Para-PLUT opens a UDP control channel for sending datagram acknowledgements from the receiver to the sender. Based on the status of the datagram checklist, Para-PLUT receiver generates and sends a list of positive or negative acknowledgments (ACK) of lost datagrams to the sender for retransmission through the control channel.

4.3.3 Automatic Parallelism Tuning Mechanism

Para-PLUT uses multiple UDP data channels in a single file transfer to improve the system resource utilization and the aggregated throughput performance. However, the aggregated throughput may drop if the number m of parallel UDP connections is too large because of the increased overhead for scheduling (e.g. context switching) and coordinating these concurrent threads on the end host. As in many parallel transport methods, it is important to determine the optimal number of UDP connections based on the end system status.

In [129], PLUT uses stochastic approximation methods to estimate the maximum attainable goodput g^* of the data receiver at the beginning of data transfer. Para-PLUT starts from one UDP connection, and gradually increases the number of parallel UDP connections at the end of the transfer of every data partition until the Para-PLUT receiver sees a decrease in its goodput measurements. Let $G(m)$ be the Para-PLUT goodput measured at a certain interval, and m_{-1} be the number of parallel UDP connections used for the previous data partition transfer, R be the sending rate for the newly established UDP connection, and C be the link bandwidth. Para-PLUT takes the following steps to adjust the number of parallel UDP connections:

(a) Initialize parameters:

$$m \leftarrow m_0,$$

$$R \leftarrow g^*,$$

$$G(m_{-1}) \leftarrow 0,$$

where m_0 is set to 1 as the initial number of parallel UDP connections.

(b) Transfer partitions of data through each UDP channel and measure the aggregated Para-PLUT goodput $G(m)$.

(c) Terminate the algorithm if the following inequality is satisfied:

$$G(m) < G(m_{-1});$$

otherwise, set

$$R \leftarrow \min(C - G(m),\ \ g^*).$$

and proceed to Step (d).

(d) Increase the number of parallel UDP connections by 1, set the sending rate of the newly added UDP channel to be *R*, and return to Step (b).

Chapter 5

Stabilizing Transport Dynamics in Overlay Networks

In this chapter, we present how the integrated transport solution works for media streaming applications in P2P overlay networks. In section 5.1, we first formulate and investigate a specific type of problem to maximize the minimum node throughput in Tree Construction (max-minTC), and investigate the complexity of max-minTC problem. Overlay network topology construction could be also considered as a path routing problem subject to multiple constraints. So max-minTC works as a route planner to explore and compose a set of feasible routing paths for streaming media delivery. Then in section 5.2, we propose *Transport Stabilization Protocol* (TSP) to stabilize transport channels over the composed routing paths at a specified throughput level in the presence of random network dynamics. TSP dynamically adjusts the window size or sleep time at the source to achieve stable throughput at the destination.

5.1 Route Planner for Media Streaming Applications

5.1.1 Problem Formulation

An overlay network is often modeled as a complete graph under the assumption that there is an overlay path between any two peers, which, however, may not be always true. For

example, when some selfish peers are reluctant to contribute bandwidth to other peers, the overlay network can not be simply modeled as a complete graph. In most local- and wide-area networks, the computer nodes are of disparate system resources and the network links (IP paths) are of different transport properties. Depending on the underlying network infrastructure, the topology of an overlay network may be complete as in the case of the Internet based on layer-3 IP routing, or not as in the case of most dedicated research testbed networks using layer-1 or 2 circuit/lambda switching or MPLS/GMPLS techniques. Even in Internet environments, the overlay network topology may not be always complete because the network connectivity and resource accessibility or availability could be largely affected by system dynamics and firewall settings on either routers or end hosts.

We model an overlay network as a directed graph *G(V, E)*, where V is the set of nodes, and E is the set of overlay links between all pairs of nodes in V, $|V| = n$. Each node $v \in V$ has an incoming (downloading) bandwidth, an outgoing (uploading) bandwidth, a splitting outgoing bandwidth, and a throughput, which are defined as follows:

Definition 1. ***incoming bandwidth*** *is the maximum downloading speed of a node. As evidenced in many commercial production networks, a node's incoming bandwidth is typically much higher than its outgoing bandwidth and the data rate, and hence is assumed to be unconstrained in our problem.*

Definition 2. ***outgoing bandwidth*** $b[v_i] \in Z^+$ *is the maximum uploading speed of node* v_i *denoted by a finite constant value, which is much smaller than the incoming bandwidth*[1] *and usually causes a bottleneck of data transfer.*

Definition 3. ***splitting outgoing bandwidth*** $s[v_i] \in Z^+$ *is the bandwidth share over an outgoing overlay link of a node. In a tree-structured graph, if the outgoing bandwidth of a node* v_i *is* $b[v_i]$ *and it has* p *child nodes, each of which is connected via one overlay link, then* $s[v_i] = b[v_i]/p$ *for each overlay link, assuming that the outgoing bandwidth of* v_i *is*

[1] *In the current Internet access market, most Internet Service Providers offer very limited uploading speed compared to downloading speed.*

fairly shared by its p child nodes. This cost model is based on the fact that the transport bottleneck is often located on or close to the end node (last mile) in the underlying physical network and the fair share of bandwidth is well supported by the wide deployment and use of TCP or TCP-friendly protocols on the Internet.

Definition 4. ***node throughput*** $r[v_i] = \min(r[v_p], s[v_p])$ *is the data receiving rate of node* v_i *assuming a sufficiently large source data rate*[2]*, where* v_p *is* v_i*'s parent node.*

Any node must respect the following constraints: (i) flow constraint, i.e., its throughput does not exceed the throughput of its parent node (namely, a node cannot deliver more than received), and (ii) capacity constraint, i.e., the sum of the throughput of all its children nodes does not exceed its own outgoing bandwidth. These two constraints are defined as:

$$\sum_{v_i \in child(v_k)} r[v_i] \quad \leq \quad b[v_k], \;\; k = 0, 1, 2, ...n-1$$

$$r[v_i] - r[v_j] \quad \leq \quad 0, \quad v_j = parent(v_i), \;\; i = 1, 2, ...n-1$$

Based on the above network models, we formulate the maxminTC problem as follows: given a directed weighted graph $G = (V, E)$, outgoing bandwidth $b[v]$ for each node $v \in V$, and a specified root node $v_0 \in V$, find a spanning tree T rooted at node v_0 in G whose minimum node throughput is maximized among all possible spanning trees of G. The corresponding decision problem is: given *G(V, E)*, other related information, and a positive integer $\beta \leq r[v_0]$, does G have a spanning tree in which any node has throughput at least β?

In max-minTC, we assume that the overlay link capacity and the node incoming bandwidth are much larger than the node outgoing bandwidth and the root node has the largest outgoing bandwidth. Therefore, the bandwidth of a path from the root node to any other

[2] *If the bottleneck is limited by the source data rate, the tree construction problem becomes trivial.*

node is determined by the minimal splitting outgoing bandwidth of all the nodes on that path. We maximize the minimum node throughput in tree construction to improve the level of streaming quality for all the users, which can not be achieved by increasing the average throughput.

5.1.2 Complexity Analysis of max-minTC

We proved max-minTC to be NP-complete by reducing from DCST (Degree-Constrained Spanning Tree) whose NP-completeness is well known in the literature.

Theorem 1: The max-minTC problem is NP-complete.

Proof. To show that max-minTC $\in$ NP, for a given graph *G(V, E)* and an integer β, we use the spanning tree T as a certificate for G. Obviously, checking if the throughput of all nodes in T is larger than β can be done in polynomial time. We prove its NP-hardness by showing DCST $\leq_P$ max-minTC.

Let $< G,k >$ be an instance I_{DCST} of DCST where $G = (V,E)$. We construct an instance $I_{max-minTC}$ of max-minTC $< G',k' >$ such that G' has a spanning tree in which no node has throughput less than β, if and only if G has a spanning tree in which no node has a degree greater than k. First, we make a copy of G and denote it as $G' = (V,E)$. Second, we set the specified root node v_0, a source rate $r[v_0]$ and node outgoing bandwidth $b[v] = b, v \in V$ and a constant integer $\beta = b/k$. The instance $I_{max-minTC}$ asks if there exists a spanning tree of G' in which no nodes has throughput less than β. Obviously, the transformation can be done in polynomial time.

Now we prove that there exists a spanning tree in G in which no node has a degree greater than k if and only if there exists a spanning tree of G' in which no nodes has throughput less than β. Given a solution T to I_{DCST}, we can find a spanning tree T' in G' that corresponds to T. It follows that no node has a degree larger than k in T'. Based on the instance construction, we know that every node in G' has the same outgoing bandwidth b. The node that has the largest degree has the smallest node throughput, which is larger

than or equal to $\frac{b}{k}$. Therefore, the throughput of any node in T' is at least $\beta = \frac{b}{k}$, i.e., T' is a solution to $I_{max-minTC}$.

Similarly, given a solution T' to $I_{max-minTC}$, i.e., the throughput of any node in T' is at least β, we can find a spanning tree T in G which is the corresponding spanning tree of T'. The node with the smallest throughput in tree T' has the largest degree, which is less than or equal to k. Therefore, T' is a solution to I_{DCST}. This concludes the proof.

5.1.3 A Heuristic Solution to max-minTC

We design a heuristic solution based on the search strategy of A^* algorithm, which uses a distance-plus-cost heuristic function $f(x)$ to determine the order in which the nodes are visited in the tree. The heuristic function $f(x)$ is a sum of two functions: $g(x)$, i.e., the "path-cost" from the starting node to the current node, and $h(x)$, i.e., an admissive heuristic estimate of the distance to the goal. In our solution, $g(x)$ and $h(x)$ are defined as functions of the outgoing bandwidth or streaming rate. For different optimization purposes, these functions can represent different quantities such as the available splitting outgoing bandwidth, the smallest node throughput of a path rooted at one node, the smallest node throughput of a spanning tree rooted at one node, or simply the outgoing bandwidth itself. This property makes our solution generic to fit in a wide variety of multicast scenarios with different $g(x)$ and $h(x)$ functions. Here, we propose a specific heuristic algorithm, named Largest Bandwidth Sum First (LBSF), to solve the max-minTC problem. In LBSF, $g(x)$ and $h(x)$ are defined as the available splitting outgoing bandwidth and outgoing bandwidth, respectively. We further define the following notations:

B**:** the list of outgoing bandwidths of all nodes.

$nc(v)$**:** the number of child nodes of v that have been selected for tree construction.

X : the set of nodes already added to the tree.

Y : the set of nodes that are not added to the tree yet.

$G(v)$: the set of undetermined neighbor nodes of v.

$S(G)$: the deterministic state of the links in G for constructing a tree.

$f(v)$: the function that determines the order the nodes are selected from the graph for tree construction.

$g(v)$: the function that is the available splitting outgoing bandwidth of the current node.

$h(v)$: the function that denotes a "heuristic estimate" of the possible minimum node throughput if adding this node into the tree.

max : the maximum value of $f(v)$.

Algorithm 1 Algorithm LBSF(G, B, v_0, r)
Input: network $G(V,E)$, outgoing bandwidth list B, root node v_0 and source data rate r
Output: spanning tree T

1: $X \leftarrow v_0$;
2: $Y \leftarrow V - v_0$;
3: **for all** $(u,v) \in E$ **do**
4: $S((u,v)) = 0$;
5: **for all** $v \in V$ **do**
6: $nc(v) = 0$;
7: **while** $Y \neq \emptyset$ **do**
8: $max = 0$;
9: **for all** $v \in X, G(v) \neq \emptyset$ **do**
10: $g(v) = \frac{b[v]}{nc(v)+1}$;
11: **for all** $u \in G(v), u \in Y$ **do**
12: $h(v) = b[u]$;
13: $f(v) = g(v) + h(v)$;
14: **if** $f(v) > max$ **then**
15: $max = f(v)$;
16: $v_s = v$;
17: $v_t = u$;
18: $S((v_s, v_t)) = 1$;
19: $nc(v_s) = nc(v_s) + 1$;
20: $X \leftarrow v_t$;
21: $Y \leftarrow Y - v_t$;
22: Construct the spanning tree T in G using links $\{l \in E | S(l) \equiv 1\}$;
23: **return** T.

The pseudocode of LBSF is provided in Algorithm 1. LBSF computes a spanning tree rooted at a specified root node to maximize the minimum node throughput. This heuristic algorithm takes a graph G, outgoing bandwidth list B for all nodes, and root node v_0 as input, and computes a spanning tree rooted at v_0. At lines 1-6 in Algorithm 1, we initialize the states of all links and the number of child nodes to be undetermined. Initially, only root node v_0 is in X and all other nodes are in Y. Starting from the root node v_0, at each iteration (lines 7-23), we select the pair of nodes with the largest value of $f(v)$ as the current code and the child node in the spanning tree, which means that this pair of nodes have the largest sum of available splitting outgoing bandwidth and outgoing bandwidth. Once a child node is selected, the state of the link between the current node and the selected child node is marked. This iterative search process terminates when all nodes have been selected, and the final spanning tree is constructed by using only those marked links. Since the algorithm always picks up the node with the largest value of $f(v)$, nodes with larger splitting outgoing bandwidths and its neighbor nodes with larger outgoing bandwidths are generally deployed at higher tiers of the spanning tree. The computational complexity of this algorithm is $O(|V|^3)$ in the worst case.

5.1.4 An Optimal Solution for Complete Networks

When the overlay network is complete, LBSF appears similar to the heuristic algorithm in [55]. However, the network model and optimization objective function in our problem are different from those in [55]. We provide a rigorous proof on the optimality of LBSF for max-minTC in this special case. With a complete network, LBSF always selects the node with the largest outgoing bandwidth as a child of the node with the largest available splitting outgoing bandwidth.

$$v_s = \underset{\forall v \in X, G(v) \neq \emptyset}{\operatorname{argmax}} \{\frac{b[v]}{nc(v)+1}\}, \tag{5.1.1}$$

$$v_t = \underset{\forall v \in G(v_s)}{\operatorname{argmax}} \{b[v]\}. \tag{5.1.2}$$

Without loss of generality, we assume that the source rate is larger than any other node's outgoing bandwidth.

Lemma 1: Child node v_t must be one of the nodes with the minimum node throughput $r[v_t]$ of the current spanning tree, which is equal to $\frac{b[v_s]}{nc(v_s)+1}$.

Proof. We prove this lemma by contradiction. Suppose that child node v_t is not one of the nodes with the minimum node throughput. This means that there is at least another node already in the tree which has smaller node throughput than this child node. Let the node resulting in the minimum throughput be v_{oc}, and its parent node be v_{op}. The claim is $r[v_t] \geq r[v_{oc}]$. Since node v_{oc} results in the minimum throughput, $r[v_{oc}]$ must be equal to $\frac{b[v_{op}]}{nc(v_{op})}$, where $nc(v_{op})$ is the number of children after v_{oc} was added under v_{op}. There are two cases: (i) If v_s was added to the tree before v_{oc}, based on Eq. 5.1.1, we have $\frac{b[v_{op}]}{nc(v_{op})} \geq \frac{b[v_s]}{nc(v_s)+1}$, where $nc(v_s)$ is the number of children before v_t was added under v_s. From the definition, we know $\frac{b[v_s]}{nc(v_s)+1} \geq r[v_t]$. (ii) If v_s was added to the tree after v_{oc}, there must be one ascendant node v_{sp} of v_s which was added before v_{oc} and satisfies $\frac{b[v_{op}]}{nc(v_{op})} \geq \frac{b[v_{sp}]}{nc(v_{sp})+1}$ based on Eq. 5.1.1. From the definition, we know $\frac{b[v_{sp}]}{nc(v_{sp})+1} \geq r[v_t]$. In both cases, we have $\{r[v_{oc}] = \frac{b[v_{op}]}{nc(v_{op})}\} \geq r[v_t]$, which contradicts each other. Therefore, child node v_t is one of the nodes with the minimum node throughput and $r[v_t] = \frac{b[v_s]}{nc(v_s)+1}$.

Lemma 2: The minimum node throughput of the current spanning tree is larger than or equal to the maximum outgoing bandwidth of all leaf nodes including the newly added node.

Lemma 2 is obvious according to Lemma 1.

Theorem 2: LBSF constructs a spanning tree T with the maximized minimum node throughput among all the spanning trees for a given complete graph.

Proof. Let $R(T)$ be the minimum node throughput of the spanning tree T, $leaf(T)$ be the set of leaf nodes. We prove this theorem by showing that the statement holds for any partial spanning tree $T(k)$, $k \leq |V|$. This can be done as follows. Based on Lemma 1, $R(T_k) = \frac{b[v_s]}{(nc(v_s)+1)}$. Given any other spanning tree T'_k with the number k of nodes. Let

$S = T' - T$ be the set of nodes that are in spanning tree T' but not in T, $S' = T - T'$ be the set of nodes that are in spanning tree T but not in T'. The outgoing bandwidth of any node in S' is larger than the outgoing bandwidth of any node in S because LBSF selects nodes with larger outgoing bandwidths first as shown in Eq. 5.1.2. Let T_k'' be the spanning tree generated by replacing S of T_k' with S'. T_k'' has exactly the same nodes as T_k, and $R(T_k'') \geq R(T_k')$, because increasing the node outgoing bandwidth does not decrease the node throughput. We consider two cases: (i) If any node $u \in leaf(T_k)$ is an internal node in T_k'', then $R(T_k'') \leq b[u]$. Based on Lemma 2, $R(T_k) \geq b[u]$. Then we get $R(T_k) \geq R(T_k'')$. (ii) If all nodes in $leaf(T_k)$ are still leaf nodes in T_k'', then $leaf(T_k) \leq leaf(T_k'')$. When $leaf(T_k) < leaf(T_k'')$, at least one node v' among all internal nodes in T_k'' has a larger degree than its corresponding node v in T_k, which means $nc(v') \geq nc(v) + 1$. Since $\frac{b[v']}{nc(v')} \geq R(T_k'')$, $\frac{b[v]}{nc(v)+1} \geq \frac{b[v']}{nc(v')}$, where $b(v')$ is equal to $b(v)$, and $(R(T_k) = \frac{b[v_s]}{(nc(v_s)+1)}) \geq \frac{b[v]}{nc(v)+1}$, we get $R(T_k) \geq \frac{b[v']}{nc(v')} \geq R(T_k'')$. When $leaf(T_k)$ is equal to $leaf(T_k'')$, let v_s' be the node in T_k'' corresponding to node v_s in T_k. If $nc(v_s') \geq nc(v_s)$, which means $\frac{b[v_s]}{nc(v_s)+1} \geq \frac{b[v_s']}{nc(v_s')}$, where $b[v_s]$ is equal to $b[v_s']$, then we get $R(T_k) \geq \frac{b[v_s']}{nc(v_s')} \geq R(T_k'')$. Otherwise, $nc(v_s') \leq nc(v_s)$, which means that at lease one of the other internal nodes $v'' \in T_k''$ has larger degree than its corresponding node $v \in T_k$. As the above analysis, shows $R(T_k) \geq R(T_k'')$. Therefore, the optimality statement holds for any $k \leq |V|$.

5.2 Transport Stabilization Protocol

Given an efficient overlay network topology, the design of transport protocols is important to achieve and sustain an acceptable level of quality of service (QoS). The media streaming applications often require streaming media be sent with predictable delays, which are in stark contrast with the delays experienced over the Internet, particularly by the messages sent using Transmission Control Protocol (TCP) [74]. In this section, we present a network

transport method that dynamically controls the source rate such that the goodput is stabilized at a desired stream level which ensures good media playback and continuous supply of streaming media.

5.2.1 Transport Control Using a Window Structure

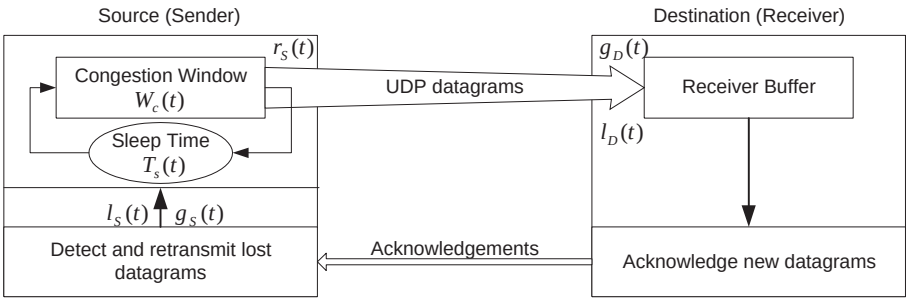


Figure 5.1: Transport control model using two control parameters.

We consider a transport control method using a window structure as illustrated in Fig. 5.1. The sender or source divides the data into parts of size MTU minus UDP and IP header lengths (i.e., MTU – 8 bytes – 20 bytes), and sends them to the receiver or destination as UDP datagrams at the sending rate $r_S(t)$. The receiver reads and acknowledges the incoming datagrams in the order they arrive. Upon the receival of a new acknowledgement, the sender computes the goodput estimate $g_S(t)$ and loss rate estimate $l_S(t)$ as the number of successfully acknowledged datagrams and the number of lost datagrams, respectively, each of which is divided by the time elapsed. All lost datagrams are reloaded for retransmission right after they are concluded to have been lost. The goodput and loss rate at the destination node are denoted by $g_D(t)$ and $l_D(t)$. The goodput measurements at the destination are sent to the source along with the acknowledgements. The measurements collected at the source are used to estimate those at the destination node as explained later.

This transport control scheme maintains two parameters, congestion window and sleep

time, both of which control the source-sending rate, but with different effects on the goodput. The congestion window, denoted by $W_c(t)$, is a counterpart of *cwnd* in TCP. It represents the number of UDP datagrams that can be sent in a burst as fast as the computer and communication hardware resources allow. The sleep time or idle time, denoted by $T_s(t)$, represents the amount of time the sender suspends transmission right after sending a full congestion window of UDP datagrams until next burst transmission. Based on this flow control model, the instantaneous source rate $r_S(t)$ can be computed as:

$$r_S(t) = \frac{W_c(t)}{T_s(t) + T_c(t)} = \frac{W_c(t)}{T_s(t) + \frac{W_c(t)}{BW}} = \frac{1.0}{\frac{T_s(t)}{W_c(t)} + \frac{1.0}{BW}}, \tag{5.2.1}$$

where $T_c(t) = W_c(t)/(BW)$ is the time needed to continuously send a full window of UDP datagrams. It is determined by the congestion window size and communication hardware resources, and mostly by the system bandwidth BW, i.e., the maximum speed at which the sender host can generate the bit signal and put it on wire. The sleep time $T_s(t)$ is akin to RTT of TCP. Generally we have $T_c(t) < T_s(t)$ due to the long delay, widely available high bandwidth, and relatively small packet size in wide-area networks. Note that the inter-*cwin* delay in TCP is approximately set to the value of RTT.

5.2.2 Goodput Stabilization of TSP

We consider the problem of transport control for stabilizing a flow from a source node S to a destination D at a specified goodput level over a wide-area network, typically the Internet, in the presence of dynamically changing background traffic. A message made up of a number of data packets is sent from source node S to destination node D. Both data packets and acknowledgements may be delayed or lost due to a variety of reasons such as buffer occupancy levels at intermediate routers and end hosts. The objective is to achieve a specified constant target goodput g_D^* at the receiver D by dynamically controlling the transmission rate $r_S(t)$ at the sender S through the adjustment made on the congestion window or sleep time. We collect measurements of goodput $g_S(\cdot)$ at source to estimate the

destination goodput $g_D(\cdot)$. The main difficulty is that the relationship between $g_D(\cdot)$ and $r_S(t)$ is very complicated.

Due to the randomness in network traffic, the destination goodput $g_D(\cdot)$ is a random variable even at a fixed source rate $r_S(t) = r$. The presence of router buffers, host buffers, and the interaction between the NIC and host CPU, in general results in a non-linear relationship between $r_S(t)$ and $g_D(\cdot)$. As a result, it is not possible to simply "back calculate" a suitable value of $r_S(t)$ to achieve the desired goodput level. Instead, we rely on the overall statistical properties of these parameters.

We define the destination *goodput response regression* as:

$$M(r) = E[g_D(t)|r_S(t) = r] = \int g_D G(dg_D, r), \tag{5.2.2}$$

where $G(\cdot, r)$ is an *unknown* distribution function of the real-valued random variable g_D at a given constant sending rate r. It is practically infeasible to provide a general analytical form for the range of g_D, which could be observed, though, for a specific sending rate using a large number of measurements in a specific real network environment. We assume that $M(\cdot)$ is *completely unknown*. For the analytical results, we assume it to be constant in the first part of analysis, and then consider it to be time-varying in the second part. Note that even the first condition is quite general in that no constraints are imposed on $G(\cdot, r)$ except its existence; since it can be infinite-dimensional, it is significantly more general than the stationary condition under parametric distributions.

We assume that the destination goodput response regression is *locally monotonic* at the target goodput $g_D(t) = g^*$ such that there exists a target source rate r^* satisfying $M(r^*) = g^*$, and the goodput response regression is *monotonic* in the vicinity of r^*, i.e.,

$$\begin{aligned} M(r) > g^* \quad &\text{if} \quad r_S(t) > r^*, \\ M(r) < g^* \quad &\text{if} \quad r_S(t) < r^*. \end{aligned} \tag{5.2.3}$$

This monotonic assumption has been validated by extensive real network performance measurements conducted in [128] at a target level below the available path bandwidth. Informally, by maintaining $r_S(t) = r^*$, we would achieve an average goodput of g^*, and an

increase (decrease) in r^* results in an increase (decrease) in $M(r^*)$. In general, computing r^* is difficult based on the information about the packet transmissions and acknowledgements alone since $G(\cdot, r)$ (and hence $M(\cdot)$) is completely unknown. Note that based on the transmissions and acknowledgements, we can only infer "noisy" versions of $M(\cdot)$ in the vicinity of actual sending rates.

As defined in Eq. 5.2.1, we may control the source rate $r_S(t)$ by adjusting either congestion window $W_c(t)$ or sleep time $T_s(t)$ individually, or both simultaneously. Generally, many different transport control problems can be solved by computing $W_c(t)$ and/or $T_s(t)$ dynamically to achieve the optimal source sending rates. However, the two-way statistical analysis conducted in [128] indicates that these two parameters strongly interact with each other, which could lead to unstable control if both are adjusted simultaneously. Therefore, we adjust one parameter at a time while fixing the other as described in the following two subsections.

5.2.3 Congestion Window Adjustment

We shall first fix the sleep time and dynamically adjust the congestion window to stabilize the goodput at a desired level given a time-varying goodput response regression. From Eq. 5.2.1 we have:

$$W_c(t) = \frac{T_s(t)}{1.0/r_S(t) - 1.0/BW}. \tag{5.2.4}$$

Consider that the sleep time is fixed at $T_s(t) = T_s$, and the desired goodput level is chosen within the initial monotonically increasing part of the goodput response regression; hence, the sending rate is a small portion of the network bandwidth. We may rewrite this equation in an approximate form:

$$W_c(t) \approx \frac{T_s}{1.0/r_S(t)} = T_s \cdot r_S(t). \tag{5.2.5}$$

It follows that the sending rate can be also approximated as the ratio of the varying congestion window to the fixed sleep time:

$$r_S(t) \approx W_c(t)/T_s. \tag{5.2.6}$$

At time step n+1, we apply a new sending rate based on the new congestion window size $W_{c,\,n+1}$, which is computed as follows:

$$W_{c,\,n+1} = W_{c,\,n} - \frac{a \cdot T_s}{n^\alpha}(g_n - g^*), \tag{5.2.7}$$

where the gain coefficient $a \cdot T_s/n^\alpha$ of the step adjustment size will be chosen as specified later.

5.2.4 Sleep Time Adjustment

Besides the congestion window, the source rate is also controllable through sleep time. From Eq. 5.2.1, we obtain the expression for sleep time $T_s(t)$ in terms of window $W_c(t)$ and sending rate $r_S(t)$ as follows:

$$T_s(t) = W_c(t)(1.0/r_S(t) - 1.0/BW). \tag{5.2.8}$$

Again, since the desired goodput is targeted at a level much lower than the network bandwidth, we may rewrite it in the following approximate form with a fixed window W_c:

$$T_s(t) \approx W_c/r_S(t). \tag{5.2.9}$$

Similarly, the source rate is approximately determined by the ratio of the fixed congestion window to the varying sleep time:

$$r_S(t) \approx W_c/T_s(t). \tag{5.2.10}$$

At time step n+1, the new sleep time is computed as follows to update the sending rate to a new value:

$$T_{s,\,n+1} = \frac{1.0}{\frac{1.0}{T_{s,\,n}} - \frac{a/W_c}{n^\alpha} \cdot (g_n - g^*)}. \tag{5.2.11}$$

After rearranging, we have the following recursive updating procedure for sleep time:

$$\frac{1.0}{T_{s,\,n+1}} = \frac{1.0}{T_{s,\,n}} - \frac{a/W_c}{n^{\alpha}} \cdot (g_n - g^*). \tag{5.2.12}$$

We define a new sleep time variable $T^{'}_{s,\,n} = \frac{1.0}{T_{s,\,n}}$ and obtain the following:

$$T^{'}_{s,\,n+1} = T^{'}_{s,\,n} - \frac{a}{n^{\alpha} \cdot W_c} \cdot (g_n - g^*), \tag{5.2.13}$$

where the gain coefficient $\frac{a}{n^{\alpha} \cdot W_c}$ of the step adjustment size will be chosen as specified in the next subsection.

5.2.5 Performance Analysis

The TSP rate control methods defined above are instantiations of the classical Robbins-Monro Stochastic Approximation (SA) method described in general as follows:

$$\begin{gathered} r_{n+1} = r_n - \varepsilon_n(g_n - g^*) = r_n - \tfrac{a}{n^{\alpha}}(g_n - g*), \\ a > 0,\ \tfrac{1}{2} < \alpha < \min(1,\ \omega) \end{gathered} \tag{5.2.14}$$

where $\varepsilon_n = a/n^{\alpha}$ is the gain coefficient of the step adjustment size and g_n is a random variable denoting the goodput estimate at the source. The classical RM conditions on the gain coefficients or step sizes are more general and are given by:

$$\sum_{i=0}^{+\infty} \varepsilon_i = +\infty, \quad \sum_{i=0}^{+\infty} \varepsilon_i^2 < +\infty, \tag{5.2.15}$$

where $\varepsilon_i \to 0$ when $i \to +\infty$, and $\varepsilon_i > 0$ for all i. Our results are valid under these general conditions, but we use Eq. 5.2.14 for concreteness. Note that the most important consideration for applying the RM stochastic approximation method to source rate control is to explicitly account for the random components and delay effects in performance measurements.

We will present two stability results. First, we consider $M(\cdot)$ to be stationary. By tagging the goodput estimates to the acknowledgements from D, we estimate two loss rates: from S to D, and from D to S. We obtain the estimator g_n by using either one of the following two ways:

1. subtracting from source rate the mean loss rate estimate from S to D;

2. adding to the acknowledgement rate the mean loss estimate from D to S.

In the special case of symmetric loss rate [105], there is no need to tag the acknowledgments since the loss rate can be estimated as the half of difference between the sending rate and received acknowledgement rate. In either case, the estimator g_n is unbiased at fixed sending rate $r(t) = r$. We also clip the sending rate values to be below a fixed bound. Thus at the time step n we have the following two properties:

$$E[g_n | r_i,\, g_i,\, r_n,\, i < n] = M(r_n) \quad \text{for all } n, \tag{5.2.16}$$

$$Var[g_n | r_i,\, g_i,\, r_n,\, i < n] \leq \sigma^2 \quad \text{for all } n. \tag{5.2.17}$$

The recursive procedure in Eq. 5.2.14 can be rewritten as:

$$\begin{aligned} r_{n+1} &= r_n - \tfrac{a}{n^\alpha}(M_n - g^*) - \tfrac{a}{n^\alpha}(g_n - M_n), \\ & a > 0,\ \tfrac{1}{2} < \alpha < 1, \end{aligned} \tag{5.2.18}$$

where the noise terms $(g_n - M_n)$, denoted by δM_n, are the martingale differences [85]. Eq. 5.2.16 describes the martingale property, which together with the properties of the step size adjustment coefficient $\varepsilon_n = \frac{a}{n^\alpha}$ guarantees the convergence $r_n \to r^*$ with probability one.

Chapter 6

Performance Evaluation and Comparison

In this chapter, we conduct an extensive set of performance evaluation and comparison of the proposed transport protocols for both stabilization and maximization using theoretical calculation, simulation, and real-life wide-area networks.

6.1 Performance Evaluation for PLUT

6.1.1 Implementation of PLUT

6.1.1.1 Sender buffer management

We now present the details of Sender Datagram Buffer Management (SDBM) and receiver acknowledgment scheme. The initial PLUT implementation employs a simple static circular buffer management that allocates a buffer of datagrams and iterates that buffer for transmission. This has its merits: indexing with random access, fairly simple implementation, and no expensive allocation/deallocation of heap-dynamic memory. There is, however, one critical flaw in that the static buffer has a chance of "overflowing" in the case of a persistent datagram loss or delayed receiver retransmission request. When the flow window in the circular buffer cannot advance, the sender will not load datagrams until the first outstanding datagram is acknowledged, hence drastically reducing the overall transport throughput.

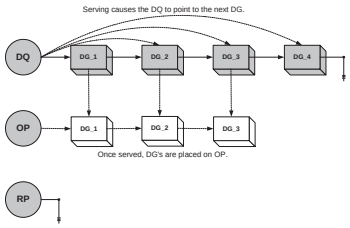


Figure 6.1: Initial buffer states.

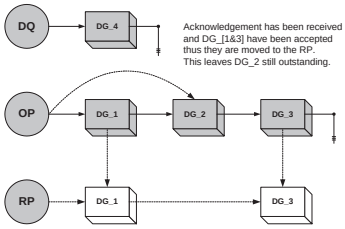


Figure 6.2: Buffer states after receiving acknowledgements.

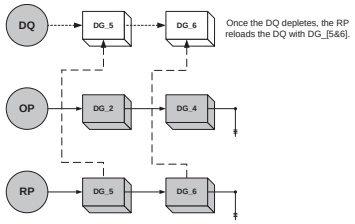


Figure 6.3: Buffer states after reloading.

To solve this SDBM problem, we designed a new data structure — Three Tier Dynamic Queuing Buffer (3TDQB) and a dynamic buffer management scheme to not only fix but provide a failsafe measure for buffer overflow as well as low retransmission frequency. This 3TDQB is composed of: 1st tier – Linked Queue, 2nd tier – Linked List, and 3rd tier – Linked List. These three tiers implement three buffers: Datagram Queuing Buffer (DQ), Outstanding Pointer Buffer (OP), and Reload Pointer Buffer (RP), respectively. Note that the DQ is dynamically allocated at the start of the protocol to alleviate allocating memory from the heap during protocol runtime, while the OP and RP only contain the pointers to the datagram buffer space allocated in DQ. This process is shown in Fig. 6.1 via the DQ's four datagrams ($DG\_[1-4]$). Once allocated, the total number of nodes the DQ initially created will not change except under extreme circumstances. The 3TDQB works in a descending manner. Each datagram will travel through the tiers consecutively before being flushed, reloaded and returned to the DQ. The DQ is a queue in which datagrams reside awaiting to be sent.

As shown in Fig. 6.1, once served, the datagram will be placed (linked through a

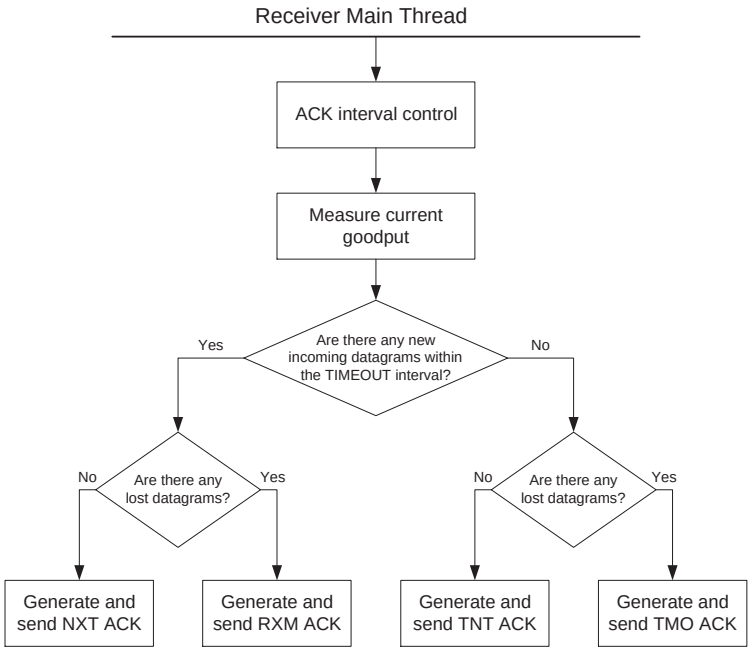


Figure 6.4: Control flow diagram at the PLUT receiver.

pointer) in the OP buffer where it waits for acknowledgment from the receiver. When acknowledged, the datagram is then placed in the RP buffer for flushing of the accepted data and reloading of the new data, as shown in Fig. 6.2. As the DQ depletes of all datagrams, the RP then reloads the DQ with an address change as shown in Fig. 6.3 so that the DQ now points to the flushed/reloaded datagrams on the RP, thus creating a new queue without new memory allocation.

The 3TDQB provides a failsafe feature to overcome the buffer overflow problem in the static circular buffer management scheme. Consider the following case: The DQ has served all remaining datagrams, the RP lies empty, and for some reason the OP has not yet received an acknowledgment. This is the 3TDQB's worst case scenario in that all datagrams reside in the OP buffer. This is similar to the circular buffer implementation, resembling a buffer overflow. To counter the overflow, the 3TDQB allocates datagrams from the heap to continue sending while waiting for acknowledgment from the receiver, thus never completely stopping the stream of data, just reducing the datagram sending rate.

6.1.1.2 Acknowledgment types

As shown in Fig. 6.4, we implement four different types of acknowledgment at the receiver: NXT (Next), RXM (Retransmission), TNT (Timeout Next), and TMO (Timeout Retransmission). For every normal ACK control period, if all datagrams received so far are in continuity, an "NXT" ACK is generated and sent to the sender; otherwise if there are lost datagrams (i.e., "holes" in the datagram checklist), the receiver compiles a list of lost datagram sequence numbers and sends them with a "RXM" ACK. If no datagram is received within a certain period of time, a timeout event is triggered where the receiver sends either a "TNT" ACK if all datagrams received so far are in continuity, or a "TMO" ACK enclosing the lost datagram sequence number list if there are "holes" in the datagram checklist. For all ACK types, the receiver measures the current instant goodput and sends it to the sender as part of the acknowledgment. On the sender side, for each incoming acknowledgment, we apply rate control using the goodput measurements enclosed in the acknowledgment.

6.1.1.3 PLUT Monitor

The PLUT Monitor provides a layer between the operating system state information in Linux environments and the transport protocol that needs the state information for flow control. We obtain the required state information by using LibGTop library [32] to read the */proc* filesystem. The virtual file system */proc* contains a hierarchy of special files which represent the current state of the kernel, and stores the information of the processes currently running. After reading the PID and state (running or sleeping) for each process from this filesystem, PLUT checks the state of these processes to estimate processes changes. A process is considered as CPU-bound if it is always ready to run when its state is checked and as IO-bound if the state of this process is different from "running" [43].

Dedicated channels	Goodput #1 (Gbps)	Goodput #2 (Gbps)	Goodput #3 (Gbps)	Goodput #4 (Gbps)	Goodput #5 (Gbps)	Goodput #6 (Gbps)	Goodput #7 (Gbps)	Goodput #8 (Gbps)	Goodput #9 (Gbps)	Goodput #10(Gbps)	Mean (Gbps)	Standard deviation
ORNL-CHI USN EoS	954.22	954.57	954.78	954.58	954.03	951.70	953.94	954.57	952.64	954.42	953.95	0.996
CHI-SUN ESnet MPLS	952.11	952.13	952.09	948.92	952.41	951.39	952.16	947.27	950.41	951.69	951.06	1.707
ORNL-SUN EoS/MPLS Mixed	952.14	952.03	952.11	951.67	949.29	952.02	950.32	952.85	951.06	946.73	951.02	1.829

Figure 6.5: PLUT performance comparison with iperf.

6.1.2 Experimental-based Performance Evaluation

6.1.2.1 Wide-area dedicated connections

We collect goodput measurements using iperf and PLUT over USN and ESnet. For iperf TCP, the number of streams n is varied between 1 and 10, and for iperf UDP, the target rate is varied as 100, 200, ..., 1000, 1100 Mbps; each set of measurements is repeated 100 times. First, we compare USN and ESnet connections of lengths 3500 and 3600 miles, respectively, and their concatenation. TCP throughput is maximized when n is around 7 or 8 and remained constant around 900, 840 and 840 Mbps for SONET, MPLS and hybrid connections, respectively. For UDP, the peak throughput is 957, 953 and 953 Mbps for SONET, MPLS and hybrid connections, respectively. Thus there is a difference of 60 Mbps and 4 Mbps between the TCP and UDP peak throughputs, respectively, over SONET and MPLS connections. There is a difference in peak throughput achieved by TCP and UDP in all cases, in particular, 57 and 93 Mbps for SONET and MPLS connections, respectively. This difference is in part due to the congestion control of TCP, and the high UDP bandwidth makes it a viable candidate for transport since there is no “congestion” on dedicated channels. We measured file transfer rates over these connections using PLUT, which achieved 954, 951 and 951 over SONET, MPLS and hybrid connections, respectively. The hosts used in these experiments are Intel Xeon Linux workstations, each of which is equipped with 4 GB memory, four 3.2 GHz CPUs, one 1 GigE NIC, and one 10 GigE NIC. The results are summarized in Fig. 6.5. Thus PLUT is able to achieve actual file transfer rate within 3 Mbps of iperf UDP bandwidth estimate in all three types of dedicated

Table 6.1: Input file sizes in file transfer experiments.

Idx. of transfer case	1	2	3	4	5	6	7
File size (MB)	100	300	500	800	1000	1500	2000

connections.

6.1.2.2 Local dedicated connections

For performance comparison, we run PLUT, UDT (version 4.4) and Tsunami (version 1.1) on a local dedicated connection, which is provisioned by a back-to-back link between two Linux boxes with kernel 2.6.30, each equipped with one 1 GigE NIC and one 10 GigE NIC, Intel(R) Core(TM)2 Duo CPU, 3 GBytes of RAM, and 1 TBytes of SCSI hard drive. Physical packet loss rate in optical fiber cables is considered very small (around 10^{-7}) and packet loss could happen at end hosts or switches. As indicated by measurements over dedicated channels [108], the packet loss at high sending rates is not negligible and the delay variations contain non-trivial random components. netem [33] provides network emulation functionality for testing protocols by emulating the delay, packet loss, packet duplication and re-ordering of wide area networks. In our experiment, the packet loss rate and delay are emulated by netem. And the packet loss rate is always set to be 10^{-5}.

Case A: 1 Gbps, disk-to-disk transfer. In this case we generate test data from the disk, the link bandwidth is set to be 1 Gbps, and the link RTT is set to be 125ms. We transfer seven files with different sizes using these three transport methods in comparison. For each file size, we run 10 tests using each transport method and collect corresponding throughput measurements. Tab. 6.1 tabulates the file sizes used in seven file transfer experiments. The mean and standard deviation of the average throughput measurements for seven different file sizes obtained by each of these three protocols are plotted in Fig. 6.6. We observe that PLUT consistently achieves higher throughput than UDT and Tsunami. The maximum possible throughput achieved by PLUT is about 970 Mbps on this link.

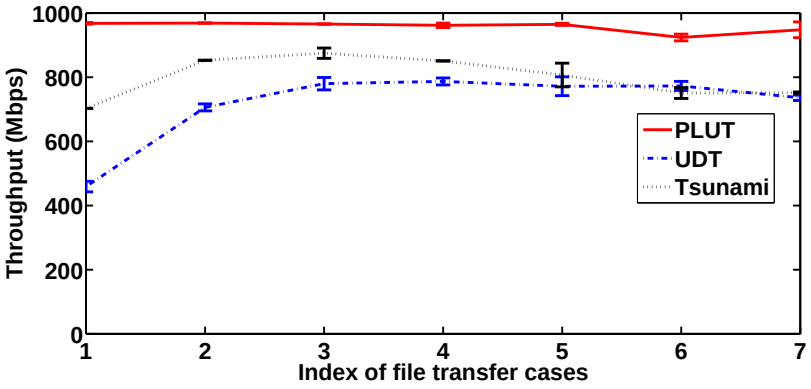


Figure 6.6: Disk to disk performance comparison over 1 Gpbs link.

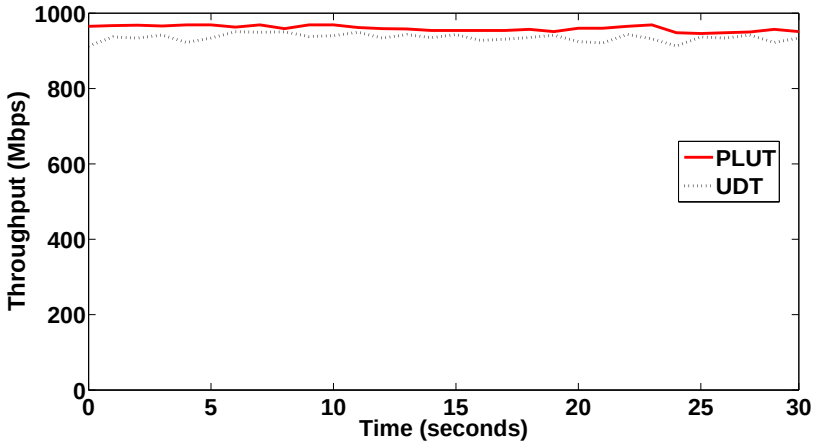


Figure 6.7: Memory to memory performance comparison over 1 Gpbs link.

Case B: 1 Gbps, memory-to-memory transfer. We set up another experiment on the same link as in Case A to test the memory-to-memory transfer performance of PLUT. The throughput measurements are plotted in Fig. 6.7, where we only include the performance comparison between PLUT and UDT because Tsunami (version 1.1) does not support memory-to-memory data transfer. This experiment demonstrates that PLUT without disk I/O activities is able to sustain an average throughput of 980 Mbps over this 1 Gbps link, which is within 4 Mbps of the iperf UDP bandwidth estimate. We observe that UDT also achieves a higher throughput than that achieved in Case A.

Case C: 10 Gbps, memory-to-memory transfer with different link delay. We now

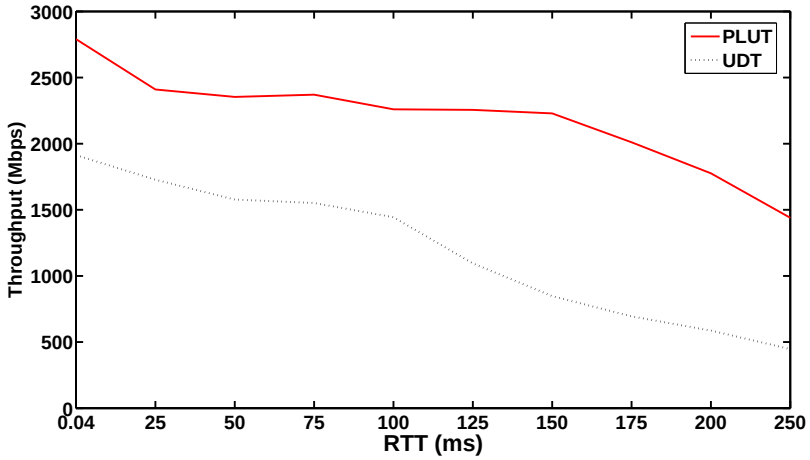


Figure 6.8: Performance comparison over a 10 Gpbs link with different RTT.

use a local dedicated connection of 10 Gbps for further performance comparison on memory-to-memory data transfer. This link bandwidth is provisioned by connecting two 10 GigE NICs back-to-back and all other network and transport control settings remain the same as in Case A. To investigate the impact of link RTT on the PLUT performance, we conduct data transport experiments by varying link RTT from 0.04 to 250 ms. Since the network delay is not constant, each selected RTT in the experiment varies based on a normal distribution. The average throughput performance measurements for PLUT and UDT are plotted in Fig. 6.8, where both protocol's throughput decrease along with the increase of RTT.

Case D: 10 Gbps, disk-to-disk transfer. We set up another experiment on the same 10Gbps link as in Case C to test disk-to-disk data transfer performance. The link RTT is set to be 125ms. We transfer seven files with different sizes using these three transport methods in comparison as in Case A. The corresponding mean and standard deviation of the average throughput measurements of these three protocols are plotted in Fig. 6.9, which again illustrates the performance superiority of PLUT over UDT and Tsunami. From the measurements of iperf UDP, we know that the maximum achievable bandwidth is about 3.1 Gbps on this link. The increase in link bandwidth shifts the bottleneck from the network to the end hosts. As a result, the maximum throughput achieved by PLUT in this case is

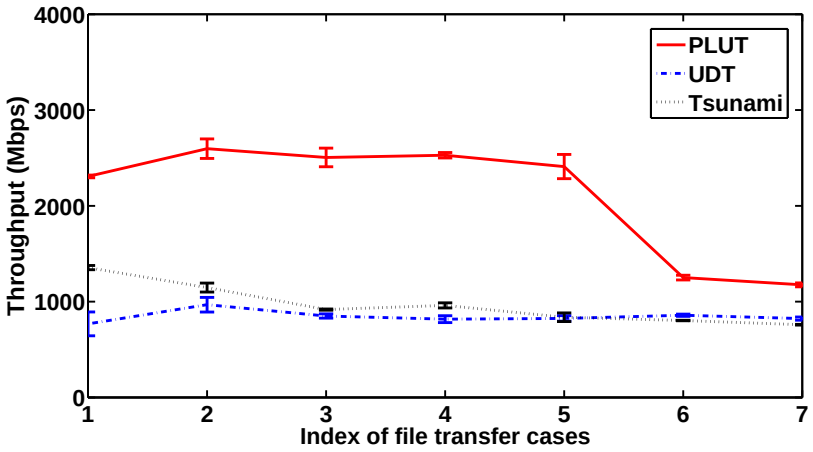


Figure 6.9: Disk to disk performance comparison over 10 Gpbs link.

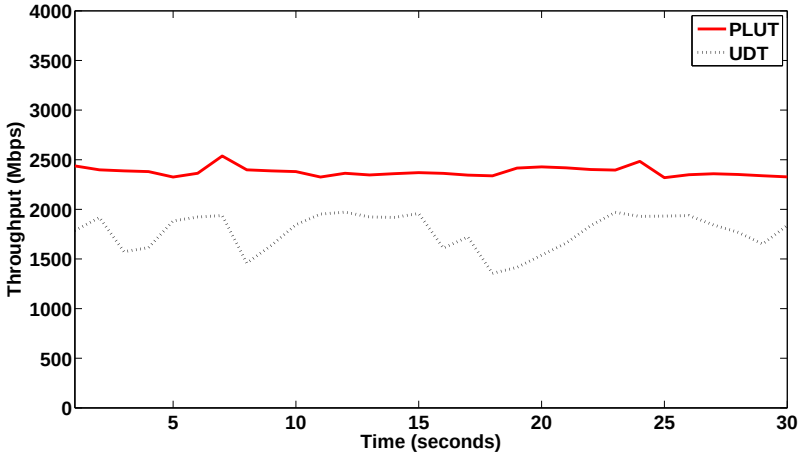


Figure 6.10: Memory to memory performance comparison over 10 Gpbs link.

about 2.4 Gbps.

Case E: 10 Gbps, memory-to-memory transfer. We test memory-to-memory transfer on the same link as in Case D and all other network and transport control settings also remain the same. Fig. 6.10 plots the corresponding PLUT and UDT throughput measurements. The maximum throughput of PLUT reaches 2.5 Gbps and stabilizes at that level.

Case F: 10 Gbps, memory-to-memory transfer with concurrent background workloads. We test memory-to-memory transfer on the same link as in Case E and all other

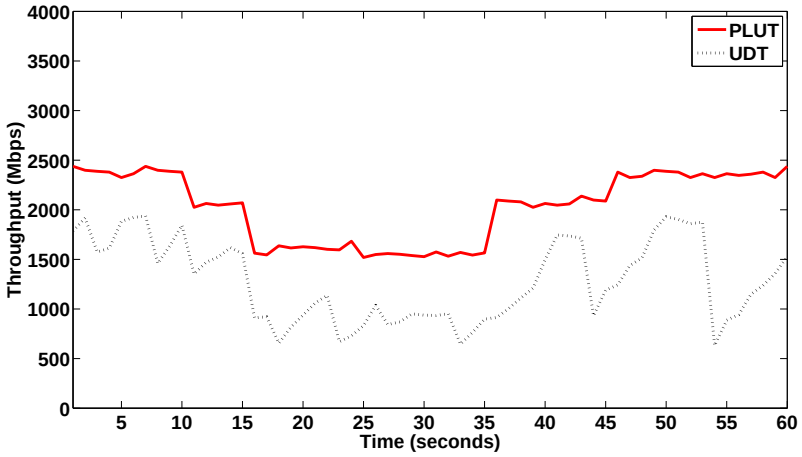


Figure 6.11: Memory to memory performance comparison over 10 Gpbs link with background workloads.

network and transport control settings also remain the same. Burncpu is a CPU-bound program designed and executed to emulate concurrent host background workloads. In this experiment, 10 seconds after the data transfer begins, we execute the first concurrent Burncpu process at the data receiver, which will run 25 seconds. And 15 seconds after the data transfer begins, we execute the second concurrent Burncpu process, which will run 30 seconds.

Fig. 6.11 plots the corresponding PLUT and UDT throughput measurements. From these measurements, we observe that the amount of concurrent background workloads has a significant impact on the performance of each transport method. UDT also adapts to the workload changes but adopts a somewhat more conservative rate control than PLUT. UDT has a relative unstable throughput measurement, because the packet loss rate increases when several CPU-bound processes compete for CPU resources.

Case G: 10 Gbps, memory-to-memory transfer with different link MTU. The previous experiments use the default MTU (1500 Bytes). To investigate the impact of link MTU on the PLUT performance, we conduct data transport experiments by varying link MTU from 3000 to 9000 bytes over the same 10 Gbps connection as in Case E. The throughput performance measurements and standard deviations for PLUT are tabulated in Fig. 6.12, where the maximum throughput is more than 7.3 Gbps.

Link MTU (Byte)	Goodput #1 (Mbps)	Goodput #2 (Mbps)	Goodput #3 (Mbps)	Goodput #4 (Mbps)	Goodput #5 (Mbps)	Goodput #6 (Mbps)	Goodput #7 (Mbps)	Goodput #8 (Mbps)	Goodput #9 (Mbps)	Goodput #10(Mbps)	Mean (Mbps)	Standard deviation
3000	3833.87	3638.49	3837.67	3588.69	3796.16	3524.89	3627.23	3569.66	3757.78	3630.39	3680.5	115.28
4000	4453.02	4745.19	4451.08	4510.52	4740.97	4771.68	4743.89	4491.91	4667.82	4431.95	4600.8	144.31
5000	5172.70	5169.18	5189.87	5167.06	5169.90	5195.95	5482.91	5464.07	5473.70	5211.91	5280.9	145.25
7000	6566.56	6224.71	6568.56	6472.92	6237.34	6276.81	6182.01	6588.95	6252.93	6503.08	6387.4	165.9
9000	7045.62	7032.46	7051.13	7005.08	6990.82	7362.09	7330.65	7025.61	7337.69	7040.57	7122.2	153.97

Figure 6.12: PLUT performance over a 10 Gpbs link with different MTU sizes.

6.2 Performance Evaluation for Para-PLUT

6.2.1 Implementation of Para-PLUT

We implement a proportional datagram allocation scheme for parallel transfer at the sender. Once the largest number of parallel UDP connections is reached, the sender divides the rest datagrams into several partitions, one for each connection. The partition size is proportional to the throughput measurement of the corresponding UDP channel.

6.2.2 Experimental-based Performance Evaluation

For performance comparison, we run Para-PLUT over the same local dedicated connection as in 6.1.2.2. In our experiment, the packet loss rate and delay are emulated by netem. The packet loss rate is set to be 10^{-5} and the link RTT is set to be 125ms.

Case A: 1 Gbps, memory-to-memory transfer. We set up one experiment on the same 1Gbps link as in PLUT to test the memory-to-memory transfer performance of Para-PLUT. The throughput measurements are plotted in Fig. 6.13. This experiment demonstrates that Para-PLUT almost reaches 100% of link utilization by using parallel UDP connections.

Case B: 10 Gbps, memory-to-memory transfer. We test memory-to-memory transfer on the same 10Gbps link as in PLUT. Fig. 6.14 plots the corresponding Para-PLUT throughput measurements. The maximum throughput of Para-PLUT reaches 3.1 Gbps, which is around 20% higher than the maximum throughput reached by PLUT.

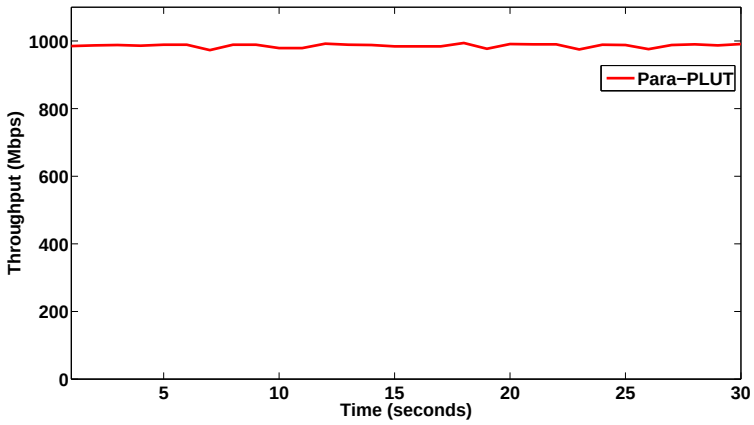


Figure 6.13: Memory to memory performance comparison over 1 Gpbs link.

In both cases we studied, the proposed Para-PLUT method consistently achieves better performance than PLUT and other protocols.

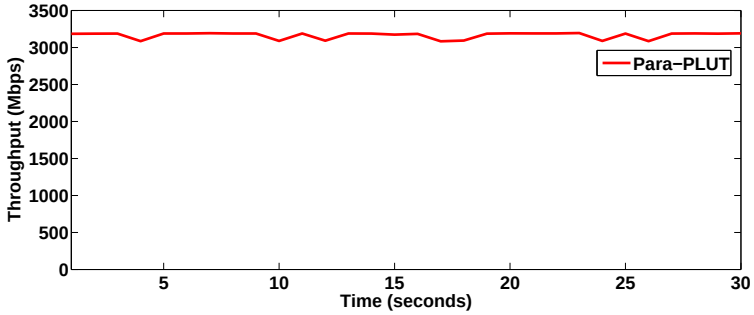


Figure 6.14: Memory to memory performance comparison over 10 Gpbs link.

6.3 Performance Evaluation for Topology Construction

We conduct a set of experiments based on simulated and real networks of various sizes and topologies for performance evaluation. The proposed LBSF algorithm is compared with a Greedy algorithm and a *k* Degree Constrained (*k*-DC) algorithm that were previously implemented in the LStream live streaming system [34]. The Greedy algorithm always selects the neighbor node with the largest outgoing bandwidth of the current code as the child node in the spanning tree at each iteration, starting from the root node, and the next

Table 6.2: Network parameters for performance comparison between *k*-DC, greedy, LBSF, and LABF algorithms

Index of problem size	1	2	3	4	5	6	7	8	9
Number of nodes	4	6	8	10	12	14	16	18	20

Table 6.3: Throughput performance comparison between *k*-DC, Greedy and LBSF algorithms.

	Simulation results / Experimental results								
Alg	1	2	3	4	5	6	7	8	9
k-DC	180 /167	160 /145	100 /91	150 /124	100 /88	113 /76	120 /64	86 /68	70 /23
Greedy	100 /98	160 /143	100 /91	50 /34	140 /76	140 /102	120 /102	90 /68	130 /56
LBSF	180 /167	160 /156	100 /88	200 /189	160 /96	170 /124	120 /102	100 /96	140 /92

iteration of search starts from the selected child node. The *k*-DC algorithm first sorts the nodes by their capacities in a decreasing order, and then adds neighbor nodes to each node (also starting from the root node) until the degree constraint *k* is reached. Both greedy and degree-constrained strategies have been widely used to solve similar multicast tree construction problems, and therefore are suited for performance comparison.

6.3.1 Experimental-based Performance Evaluation

We build a live streaming testbed by deploying 20 computers (peers) in a local area network and run LStream, the P2P live streaming system developed by Henan Education and Research Network [34] on each of these nodes. For a convenient reference, we tabulate these network parameters in Table 6.2. The source video streaming rate is set to be 400 Kbps in our experiments. The outgoing bandwidth of each node is configured to be a random quantity in a range from 200 Kbps to 800 Kbps in the granularity of 20 Kbps. We run three

Table 6.4: Network parameters for performance comparison between *k*-DC, greedy, LBSF, and LABF algorithms

Index of problem size	1	2	3	4	5	6	7	8	9	10
Number of nodes	20	40	60	80	100	120	140	160	180	200
Number of links	60	120	180	240	300	360	420	480	540	600

algorithms in comparison to compute three trees in each of 9 overlay networks with different numbers of nodes varying from 4 to 20 at an interval of 2 nodes. We then construct the tree topology accordingly for streaming experiments and collect their minimum node throughput performance measurements as tabulated in Table 6.3, where the corresponding simulation results are also provided in pair for comparison. We observe that the experimental results match well with the simulation results, which indicates the accuracy of our cost models and also validates our assumptions on the location of the bandwidth bottleneck.

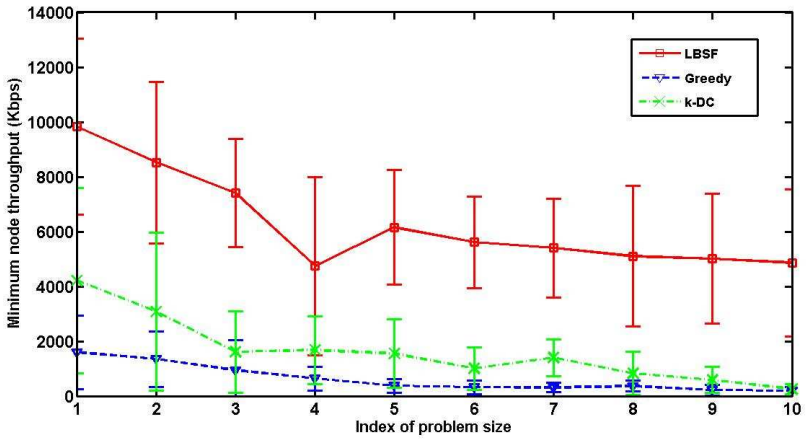
6.3.2 Simulation-based Performance Evaluation

To account for the lack of network and system resources, we further conduct simulation-based performance comparison among these three algorithms using a large set of simulated networks. For a given number of nodes and links, each simulated network is created with a randomly generated network topology, and a random outgoing bandwidth within a range from 0 to 32767 Kbps is assigned to each node.

6.3.2.1 Scalability

In the first set of simulations, we test the scalability of these three algorithms based on a series of 10 simulated networks, indexed from 1 to 10, with a varying number of nodes from 20 to 200 at an interval of 20 nodes, and a varying number of links from 60 to 600 at an interval of 60 links, respectively. For a convenient reference, we tabulate these network parameters in Table 6.4. The capacity or outgoing bandwidth of each node is randomly

assigned. The capacity or outgoing bandwidth of each node is randomly assigned.



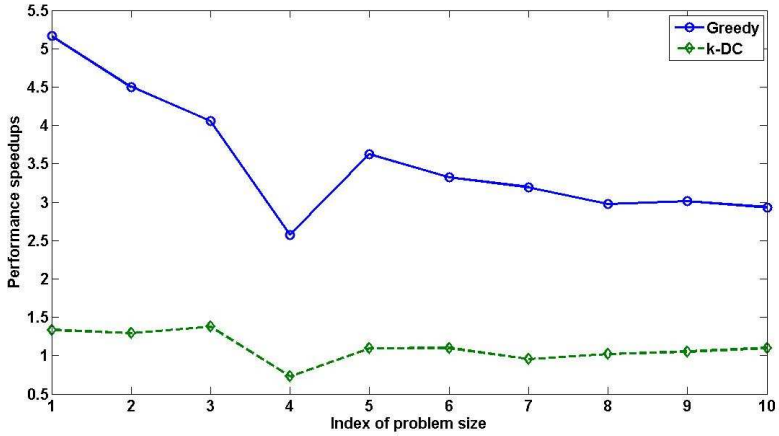
[h]

Figure 6.15: Minimum node throughput performance comparison (mean and standard deviation) among three algorithms based on a series of 10 simulated networks of various sizes ranging from small to large scales.

For each given network size, we create 10 random instances with different network topologies. We run three algorithms in each of these 10 network instances and compute the minimum node throughput of three resultant trees. The mean and standard deviation of the minimum node throughput measurements in 10 instances for each network size are plotted in Fig. 6.15. We also plot in Fig. 6.16 the performance speedups of LBSF over the *k*-DC and Greedy algorithms defined as $\frac{LBSF - kDC \text{ or } Greedy)}{(kDC \text{ or } Greedy)}$, which shows that the LBSF algorithm achieves at least 90% higher minimum node throughput than the *k*-DC algorithm and up to 5 times improvement over the Greedy algorithm.

6.3.2.2 Robustness

We conduct a second set of simulations to study the robustness of these three algorithms based on 10 simulated networks of 400 nodes and a varying number of links from 1000 to 10000 at an interval of 1000 links. Similarly, for each given number of links (note that the number of nodes is fixed to be 400), we create 10 random instances with different



[h]

Figure 6.16: Performance speedups of LBSF over DC and Greedy based on a series of 10 simulated networks of various sizes ranging from small to large scales.

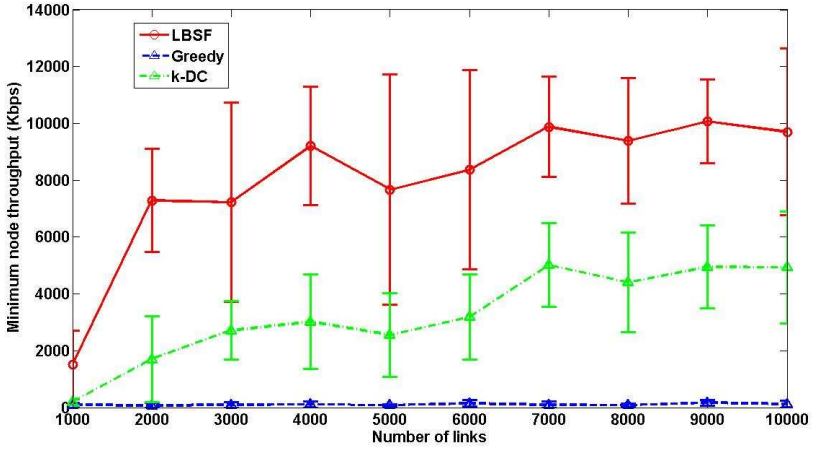


Figure 6.17: SMinimum node throughput performance comparison (mean and standard deviation) among three algorithms based on a series of 10 simulated networks of 400 nodes and a varying number of links from 1000 to 10000 at an interval of 1000 links.

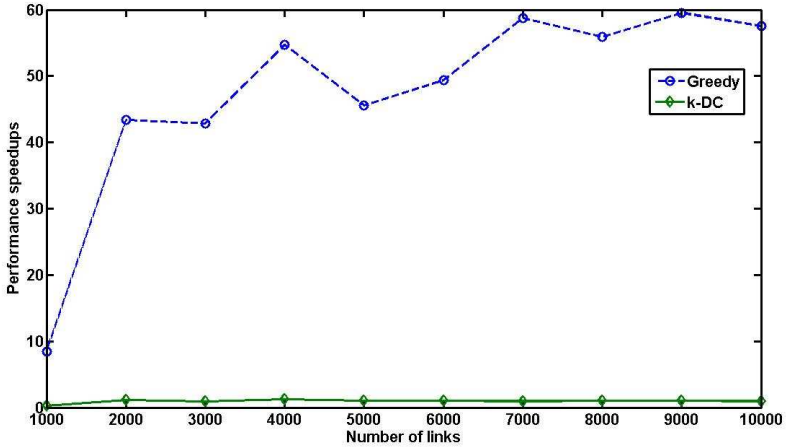


Figure 6.18: Performance speedups of LBSF over *k*-DC and Greedy based on a series of 10 simulated networks of 400 nodes and a varying number of links from 1000 to 10000 at an interval of 1000 links.

network topologies. The mean and standard deviation of the minimum node throughput measurements obtained by each of these three algorithms in 10 instances for each number of links are plotted in Fig. 6.17 and the corresponding performance speedups of LBSF over the other two algorithms are plotted in Fig. 6.18. We also observe that LBSF consistently achieves at least 100% higher minimum node throughput than the *k*-DC algorithm and almost 60 times improvement over the Greedy algorithm because LBSF approaches the optimum as the number of links increases.

Both the Greedy and k-DC algorithm select the pair of current node and child node only based on the nodes outgoing bandwidth without considering the available splitting outgoing bandwidths. Note again that the minimum node throughput on a path from the root node to any other node is determined by the bottleneck splitting outgoing bandwidth. Therefore, a node with a large outgoing bandwidth but a relatively small available splitting outgoing bandwidth degrades the performance of all its downstream nodes. But LBSF always picks up the node with the largest value of $f(v)$ which considers both nodes available splitting outgoing bandwidths and outgoing bandwidths which results in a much higher minimum

node throughput than the other two methods.

6.4 Performance Evaluation for TSP Based on Simulations and Experiments

6.4.1 Simulation-based Performance Evaluation

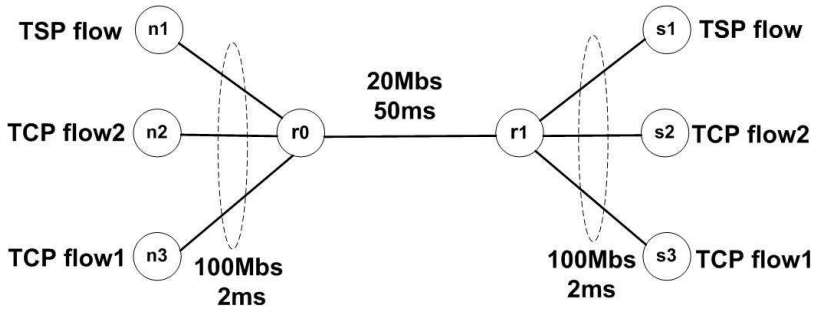


Figure 6.19: Simulation setup for TSP stabilization.

We first evaluate TSP's performance in a controlled simulation environment using the NS3 simulator. As shown in Fig. 6.19, we create a network topology that consists of 4 computer nodes in two separate local-area networks connected through fully duplex network links. All the local network links have a bandwidth of 1 Gbps and a link delay of 6560 ns while the wide-area network link between routers r0 and r1 has a bandwidth of 1 Gbps and a link delay of 200 ms. We deploy a TSP sender on n1, which is connected to a TSP sink on n3.

We conduct two sets of transport experiments in this simulated network environment. In the first case, the sending rate of two TCP flows is set to be 5 Mbps, and hence the peak available bandwidth of the network connection is about 10 Mbps. We set the target goodput of TSP to be 7 Mbps, and set the parameters $a = 0.8$ and $\alpha = 0.8$. In the second case, we increase the link bandwidth between r0 and r1 to 40 Mbps and the sending rate

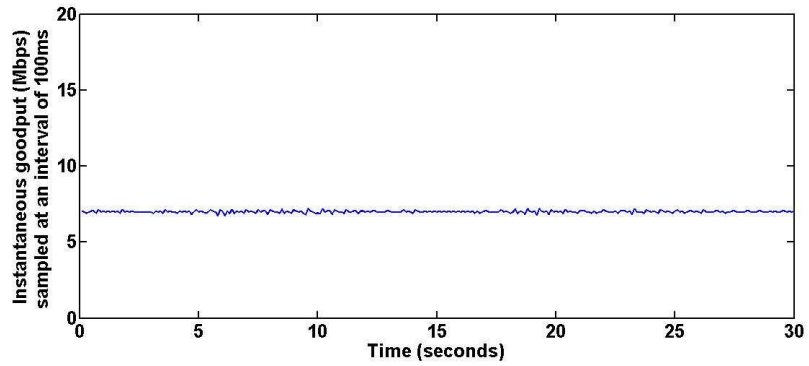


Figure 6.20: Goodput stabilization at a target rate = 7 Mbps with a = 0.8 and $\alpha = 0.8$, adjustment made on sleep time.

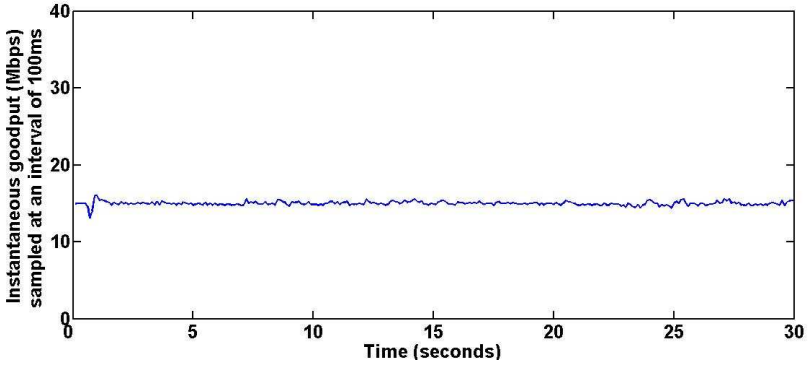


Figure 6.21: Goodput stabilization at a target rate = 15 Mbps with a = 0.8 and $\alpha = 0.8$, adjustment made on sleep time.

of both TCP flows to 10 Mbps, while all other network and transport settings remain the same as in the previous case. The peak available bandwidth of the network connection in this case is about 20 Mbps. We set the desired goodput of TSP to be 15 Mbps. We collect instantaneous goodput measurements at an interval of 100 ms in both cases and plot TSP's goodput stabilization performance in Fig. 6.20 and Fig. 6.21, respectively. We observe that TSP is able to stabilize at the target level with a low level of transport dynamics. The stabilization performance of TSP in other simulation settings is qualitatively similar.

6.4.2 Experiment-based Performance Evaluation

6.4.2.1 Implementation Details and Network Settings

We implemented the TSP transport method in C++ on Linux operating system. The TSP sender consists of two child threads, one for sending the datagrams, and the other for receiving the acknowledgements. The source rate control related activities such as RTT estimation, packet loss detection, goodput and loss rate calculation, are carried out in the receiving thread upon the arrival of each acknowledgement. The source rate is controlled by the transport dynamics defined in Eq. 5.2.7 or 5.2.13 through the adjustment made on either congestion window or sleep time. Besides the thread for receiving datagrams, a separate child thread in the receiver stores or forwards in-order datagrams to user applications.

TSP provides reliable data transfer by assigning a sequence number to each packet and retransmitting those lost ones. We consider four different types of acknowledgment at the receiver: NXT (Next), RXM (Retransmission), TNT (Timeout Next), and TMO (Timeout Retransmission). For every normal ACK control period, if all datagrams received so far are in continuity, an NXT ACK is generated and sent to the sender; otherwise if there are lost datagrams (i.e., holes in the datagram checklist), the receiver compiles a list of lost datagram sequence numbers and sends them with a RXM ACK. If no datagram is received within a certain period of time, a timeout event is triggered where the receiver sends either a TNT ACK if all datagrams received so far are in continuity, or a TMO ACK enclosing the lost datagram sequence number list if there are holes in the datagram checklist.

We tested the TSP transport method over one Internet connection: UM (University of Memphis)-LSU. The Internet connection is of low bandwidth over few thousand network miles with high LAN traffic at both ends. For each set of experiments on the Internet connection, we run concurrent control traffic as well as on-host and LAN background network traffic such as HTTP, FTP, and SSH, during the experiments to test the robustness of our method. The source rate is controlled through either $W_C(\cdot)$ or $T_S(\cdot)$ to stabilize at a given g^*.

Since g^* is lower than the available bandwidth, the target source rate matches the desired goodput level with very little packet loss. Therefore, we use Eq. 5.2.1 to determine the initial values of congestion window and sleep time. We achieved qualitatively similar stabilization between several other nodes connected over wide-area networks, and the above results are typical.

6.4.2.2 Performance Comparison with TCP and DCCP on UM-LSU Connection

We first conduct TSP transport experiments on UM-LSU connection and compare its performance with that of two widely deployed transport protocols, TCP and DCCP [81]. Both end hosts, cow.cs.memphis.edu and robot.rrl.lsu.edu, are regularly configured Linux workstations with 2.6.25 kernel and are connected to their corresponding local campus network through one GigE network interface card (NIC). To closely examine the microscopic behaviors of these three protocols in comparison, we collect instantaneous goodput measurements at an interval of 100 ms, which is about 2-3 times the round trip time (RTT) of this wide-area connection.

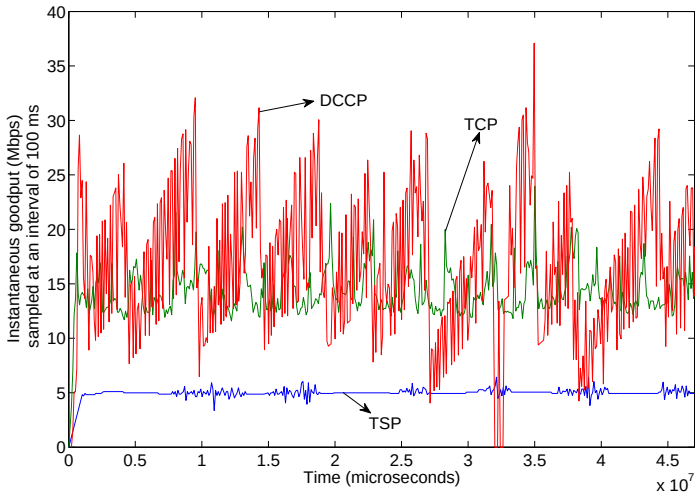


Figure 6.22: Performance comparison of instantaneous goodput sampled at an interval of 100 ms using TCP, DCCP, and TSP targeting 5 Mbps.

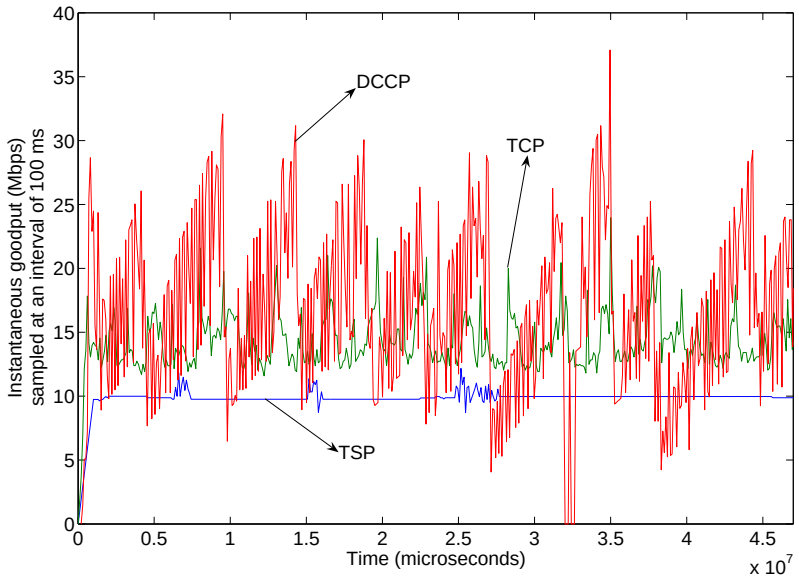


Figure 6.23: Performance comparison of instantaneous goodput sampled at an interval of 100 ms using TCP, DCCP, and TSP targeting 10 Mbps.

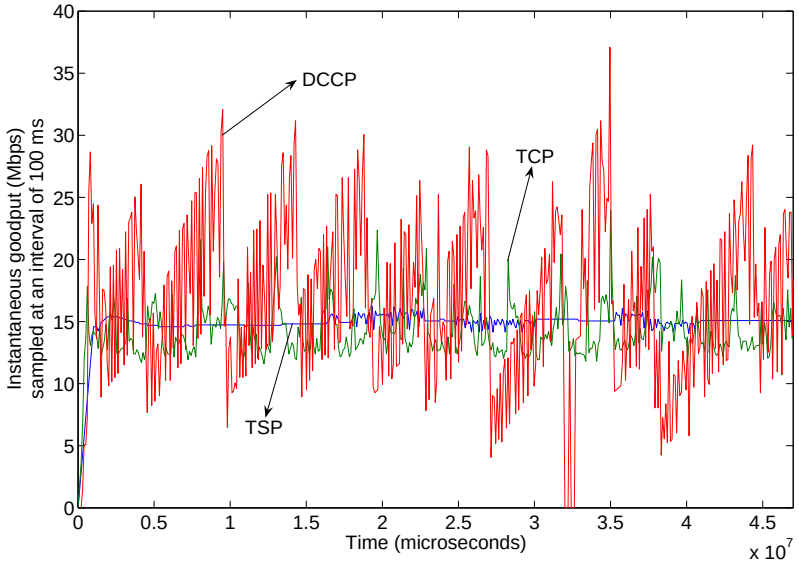


Figure 6.24: Performance comparison of instantaneous goodput sampled at an interval of 100 ms using TCP, DCCP, and TSP targeting 15 Mbps.

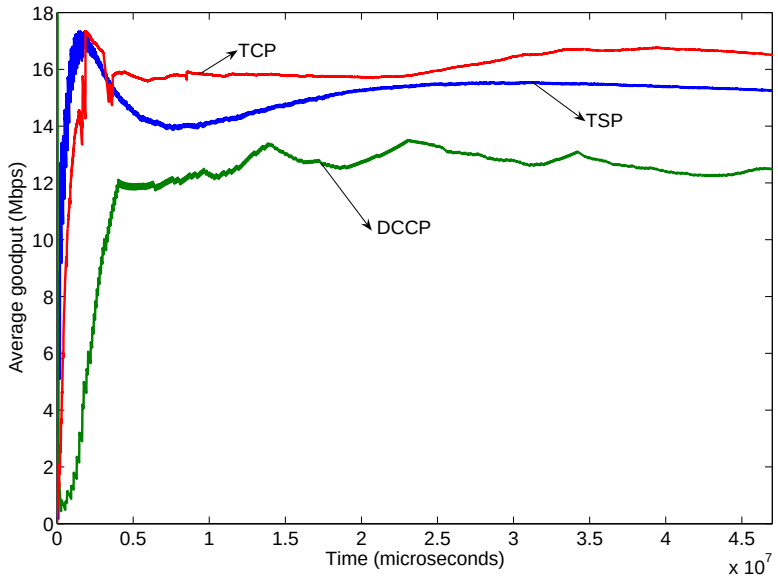


Figure 6.25: Performance comparison of average goodput using TCP, DCCP, and TSP targeting 15 Mbps.

A test message of 100 Mbytes is created on the LSU host and transferred to the UM host. The instantaneous transport performance measurements for TCP, DCCP, and TSP targeting 5, 10, and 15 Mbps are plotted in Figs. 6.22, 6.23, and 6.24, respectively. The source rate is controlled through the adjustment of the congestion window size only. For these three target rates, we select the starting points of TSP at $Wc(0) = 42$, $Wc(0) = 84$, and $Wc(0) = 127$ datagrams, respectively, with fixed sleep time $Ts(t) = 100$ ms, $a = 0.8$, and $\alpha = 0.8$. We observe that TSP experiences significantly less transport dynamics at the specified target rates than the other two protocols shooting for the maximum possible goodput. Note that the peak available bandwidth of this connection is slightly more than 15 Mbps achieved by TCP during the course of the experiments. Even at this peak rate, TSP is able to stabilize without significantly affecting concurrent TCP traffic. We also plot the average goodput measurements for these three protocols in Fig. 6.25, which clearly shows that TSP achieves the smoothest goodput among the three from a global perspective. To produce cleaner performance curves, we use average goodputs in the rest of the TSP

experiments where we focus more on the robustness of TSP and investigate the effects of parameter selection on TSP transport performance.

6.4.2.3 Effects of TSP on Concurrent TCP

Since TCP is the de facto standard for reliable end-to-end data transport and carries the majority of the traffic in today's Internet, any new transport protocols are generally required to be TCP-friendly to be widely deployed in Internet environments. To study the effects of the proposed TSP on regular TCP traffic, we conduct two transport experiments on UM-LSU connection using the same network and control settings as in Subsection 6.4.2.2. In the first experiment, we run a single TCP session, while in the second one, we run one TCP session and one TSP session targeting 5 Mbps concurrently on both end hosts. Three instantaneous goodput performance curves sampled at an interval of 100 ms from these two transport experiments are plotted in Fig. 6.26 for comparison.

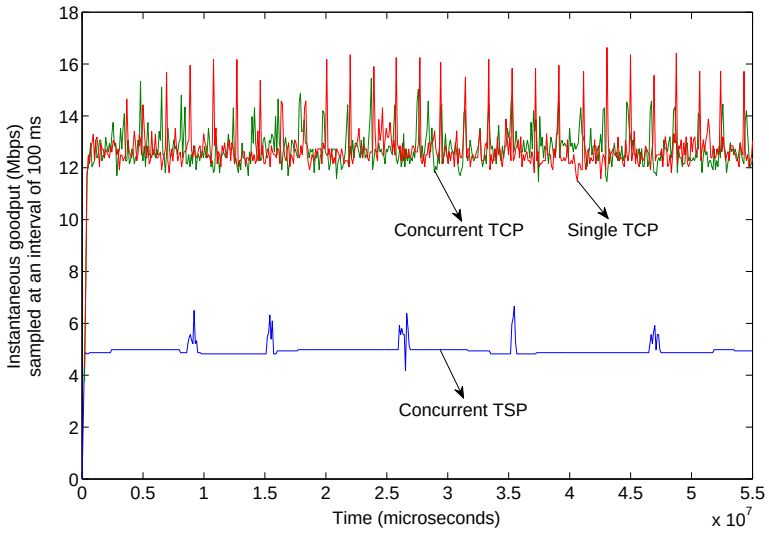


Figure 6.26: Effects of TSP on concurrent TCP traffic.

We observe that the concurrent TCP flow only experiences about 3% drop in average goodput from 13.27 Mbps of the single TCP flow to 12.86 Mbps, which strongly indicates

the TCP-friendliness of TSP. We repeat these experiments on other Internet connections with different target rates, and observe qualitatively very similar effects of TSP on TCP: TSP does not affect the performance of concurrent TCP traffic significantly as long as the target rate is controlled within a fraction of the peak available connection bandwidth, which is in good line with the goodput requirement of control channels. The design goal of TSP is not to aggressively grab as much bandwidth as possible by suppressing other traffic, but to strategically utilize a small portion of the available bandwidth where one can achieve a smooth data flow with predictable delays and dynamics for control command delivery purposes.

6.4.2.4 Concurrent Control over UM-LSU Link

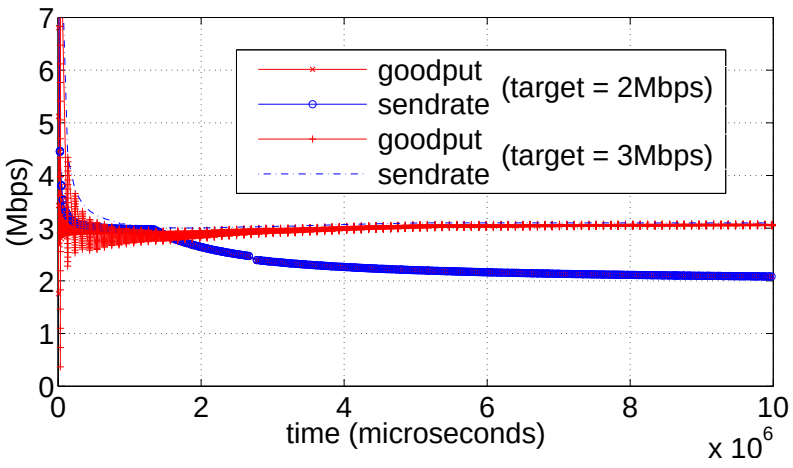


Figure 6.27: UM-LSU link: concurrent control channels at 2.0 Mbps and 3.0 Mbps, respectively, a = 0.8, $\alpha = 0.8$, adjustment made on sleep time.

To investigate the interaction between concurrent TSP flows that employ the same rate control strategy, we conduct experiments where two concurrent TSP control channels targeting 2.0 Mbps and 3.0 Mbps, respectively, are established over the same UM-LSU Internet path. The goodput and sending rate performance measurements are plotted in Fig. 6.27,

which shows that both flows stabilize at their corresponding target levels. Extensive experimental results have shown that as long as the total required throughput of multiple control channels is less than the available path bandwidth, concurrent TSP transport sessions using the proposed stabilization method are able to live “peacefully” without significantly affecting each other’s performance.

In all the above experiments, we observed that the measured TSP goodput converges to the target rate quickly within a few seconds or less. Our experimental results show that the TSP goodput stabilization is robust against the presence of various concurrent TCP traffic such as HTTP, FTP, and SSH.

Chapter 7

Conclusion and Future Work

Our research effort was focused on a rigorous analytical study of the design and performance of transport solutions, and develop an integrated transport solution to overcome the limitations of current methods. We have presented the integrated transport solution, which consists of two functional components: data route planner and transport control. This architecture enables easy implementation of new route planners, as well as easy evaluation of new transport control protocols.

This final chapter begins with a summary of the contributions of this dissertation in section 7.1. Section 7.2 lists some limitations as well as the guidelines for future research work.

7.1 Contribution

Below we highlight the contributions of this dissertation:

7.1.1 Route Planner for Bulk Data Transfer in Scientific Applications

NADMA explores and composes a set of feasible route options and provides them to the user along with performance estimations as well as specific steps and commands to authorize and execute data transfer. Our work systematically investigated the design and implementation issues of a high performance route planer for bulk data transfer. NADMA

identified the main challenges in the wide adoption of new networking and storage technologies and proposed appropriate solutions. As a result, application users can accomplish their data-centric tasks by following specific steps and commands to authorize and execute data transfer.

7.1.2 Peak Link Utilization Transport

PLUT incorporates a performance-adaptive flow control mechanism to regulate the activities of both the sender and receiver in response to system dynamics and automates the rate stabilization for throughput maximization using stochastic approximation methods. Para-PLUT identified the problems associated with PLUT and further improved the performance of PLUT. To the best of our knowledge, Para-PLUT is the first transport protocol that attempts to use parallel UDP connections to take advantage of the full power of multi-core processors in maximizing application throughput.

7.1.3 Route Planner for Streaming Media Delivery

We formulated a novel tree construction problem max-minTC. We proposed an efficient heuristic algorithm named LBSF to solve max-minTC problem. The goal of LBSF is to optimize the system's stream rate by constructing a spanning tree whose minimum node throughput is maximized among all possible spanning trees of the overlay network. LBSF addresses both scalability and robustness.

7.1.4 Transport Stabilization Protocol

We identified the difficulty of goodput stabilization over wide-area networks. TSP explicitly accounts for the randomness inherent in wide-area networks to stabilize the goodput at destination. While TSP is highly efficient, it is not necessarily aggressive. It is friendly to concurrent TCP flows.

The superior and robust performance of the proposed transport solutions is extensively evaluated in a simulated environment and further verified through real-life implementations and deployments over both Internet and dedicated connections under disparate network conditions in comparison with existing transport methods.

7.2 Limitations and Future Work

Although Para-PLUT employs an parallel mechanism to improve the application goodput and resources utilization, it didn't account for the impact of the background workloads, which vary during the data transfer period. The parallelism tuning mechanism employed in Para-PLUT needs to be adjusted to achieve the maximal bottleneck rate by taking the varying background workloads into consideration. Thus, new flow control approaches in the context of multiple parallel connections need to be developed to estimate the best rate at which the end system can consume packets coming from the network. This best rate will be sent back to the sender for source rate control.

The depletion of a fixed buffer results in the drop of the data, even there might be plenty of free memory on the receiving end host. It is necessary to design an automatic buffer adaption mechanism to decide whether the receive buffer needs to be resized and to what extent the receiver needs to be adjusted. Furthermore, as the number of competing workloads increases or decreases, the number of parallel connection should decrease or increase correspondingly. It is critical to design an automatically mechanism to adjust the number of parallel connections in response to the end host dynamics to achieve the maximum goodput. We will design an integrated dynamic control mechanism to automatically adjust buffer size and the number of parallel streams and find the optimal values for those parameters.

Bibliography

[1] NSF Grand Challenges in eScience Workshop, 2001. http://www2.evl.uic.edu/NSF/index.html.

[2] Large Hadron Collider (LHC). http://lhc.web.cern.ch/lhc.

[3] http://www.jamstec.go.jp/esc/.

[4] Spallation Neutron Source. http://www.sns.gov.

[5] User Controlled LightPath Provisioning. http://phi.badlab.crc.ca/uclp.

[6] Dynamic Resource Allocation via GMPLS Optical Networks. http://dragon.maxgigapop.net.

[7] JGN II: Advanced Network Testbed for Research and Development. http://www.jgn.nict.go.jp.

[8] Geant2. http://www.geant2.net.

[9] On-demand Secure Circuits and Advance Reservation System. http://www.es.net/oscars.

[10] https://plone3.fnal.gov/P0/ESCPS/.

[11] HOPI: Hybrid Optical and Packet Infrastructure. http://networks.internet2.edu/hopi.

[12] Advance Network Initiative. http://www.es.net/RandD/advanced-networking-initiative/.

[13] PPLive. http://www.pplive.com.

[14] PPStream. http://www.ppstream.com.

[15] Storage Resource Management (SRM). https://sdm.lbl.gov/srm-wg/.

[16] http://dev.globus.org/wiki/GridFTP.

[17] https://www.racf.bnl.gov/terapaths/.

[18] GENI: Global Environment for Network Innivations. http://www.geni.net.

[19] Internet2 Interoperable On-Demand Network (ION) Service. http://www.internet2.edu/ion.

[20] http://www.es.net/.

[21] Internet2. http://www.internet2.edu.

[22] http://en.wikipedia.org/wiki/Storage_Resource_Manager.

[23] https://sdm.lbl.gov/bestman/.

[24] http://www.gridpp.ac.uk/wiki/RAL_Tier1_CASTOR_SRM.

[25] http://www.dcache.org/.

[26] http://www.gridpp.ac.uk/wiki/Disk_Pool_Manager.

[27] http://storm.forge.cnaf.infn.it/home.

[28] http://www.slac.stanford.edu/ abh/bbcp/.

[29] Tsunami. http://newsinfo.iu.edu/news/page/normal/588.html.

[30] http://www.globus.org/rls/.

[31] http://linux.maruhn.com/sec/bing.html.

[32] Libgtop. ftp://ftp.gnome.org/pub/GNOME/sources/libgtop/2.0.

[33] http://www.linuxfoundation.org/collaborate/workgroups/networking/netem.

[34] HERNET. http://web.ha.edu.cn/jianjiee.aspx.

[35] Enlightened computing: An architecture for co-allocating network, compute, and other grid resources for high-end applications. In *Proc. of IEEE Honet*, Dubai, UAE, Nov. 2007.

[36] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The globus striped GridFTP framework and server. In *Proc. of Supercomputing*, 2005.

[37] E. Altman and D. Barman. Parallel TCP sockets: Simple model, throughput and validation. In *Proc. of IEEE INFOCOM*, 2006.

[38] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas. Rfc3036: Ldp specification. In *IETF RFC*, Jan. 2001.

[39] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. Rfc3209: Rsvp-te: Extensions to rsvp for lsp tunnels. In *IETF RFC*, Dec. 2001.

[40] A. Banerjee, W. Feng, B. Mukherjee, and D. Ghosal. Rapid: An end-system aware protocol for intelligent data transfer over lambda grids. In *Proc. of the 20th IEEE/ACM Int. Parallel and Distributed Processing Symp.*, Rhodes Island, Greece, Apr. 25-29 2006.

[41] A. Banerjee, B. Mukherjee, and D. Ghosal. Modeling and analysis to estimate the end system performance bottleneck rate for high-speed data transfer. In *Proc. of the 5th Int. Workshop on Protocols for FAST Long-Distance Networks*, Los Angeles, CA, Feb. 7-9 2007.

[42] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of the ACM SIGCOMM*, pages 205–217, 2002.

[43] Marta Beltran and Antonio Guzman. A new cpu availability prediction model for time-shared systems. *IEEE Transaction 2009*, 57:865–875, 2009.

[44] A. Benveniste, M. Metivier, and P. Priouret. *Adaptive Algorithms and Stochastic Approximation*. Springer-Verlag, New York, 1990.

[45] U. Bhat. *An Introduction to Queueing Theory*. Birkhauser Boston Publications, 2008.

[46] P.P. Bocharov, C. D'Apice, A.V. Pechinkin, and S. Salerno. *Queueing Theory*. Walter de Gruyter, 2004.

[47] G. Bolch, S. Greiner, H. Meer, and K.S. Trivedi. *Queueing Networks and Markov Chains*. Wiley-Interscience, 2006.

[48] L.S. Brakmo and S.W. O'Malley. Tcp vegas: new techniques for congestion detection and avoidance. In *SIGCOMM '94 Conf. on Comm. Arch. and Proto.*, pages 24–35, London, United Kingdom, Oct. 1994.

[49] L.S. Brakmo, S.W. O'Malley, and L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. In *SIGCOMM'94 Conf. on Communications Architectures and Protocols*, pages 24–35, London, United Kingdom, Oct. 1994.

[50] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proc. of the 19th ACM Symp. on Oper. Sys. Prin.*, pages 298–313, 2003.

[51] CHEETAH: Circuit-switched High-speed End-to-End Transport ArcHitecture, http://www.ece.virginia.edu/cheetah.

[52] K.T. Chen, C.Y. Huang, P. Huang, and C.L. Lei. An empirical evaluation of TCP performance in online games. In *Proc. of ACM SIGCHI ACE06*, Los Angeles, USA, June 2006.

[53] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. *IEEE J. on Selected Areas in Communication*, 20(8):1456–1471, Oct. 2002.

[54] J. Crowcroft and P. Oechslin. Diferentiated end-to-end internet services using a weighted proportional fair sharing tcp. In *SIGCOMM Computer Communication Review*, pages 28(3):53–69, 1998.

[55] Y. Cui, Y. Xue, and K. Nahrstedt. Max-min overlay multicast: Rate allocation and tree construction. In *12th IEEE Int. Workshop on Quality of Service (IwQoS 04)*, pages 7–9, 2004.

[56] J.N. Daigle. *Queueing Theory with Applications to Packet Telecommunication*. Springer, 2005.

[57] P. Datta, W. Feng, and S. Sharma. End-system aware, rate-adaptive protocol for network transport in lambdagrid environments. In *Proc. of the 2006 ACM/IEEE conference on Supercomputing*, Tampa, FL, Nov. 11-17 2006.

[58] P. Datta, S. Sharma, and W. Feng. A feedback mechanism for network scheduling in lambdagrids. In *Proc. of the 6th International Symposiumon on Cluster Computing and the Grid*, Singapore, May 16-19 2006.

[59] High-performance networks for high-impact science, Aug. 13-15 2002. Report of the High-Performance Network Planning Workshop, http://DOECollaboratory.pnl.gov/meetings/hpnpw.

[60] Network provisioning and portocols for DOE large-science applications, Aug. 10-11 2003. Report of DOE Workshop on Ultra High-Speed Transport Protocol and Dynamic Provisioning for Large-Scale Applications, http://www.csm.ornl.gov/ghpn/wk2003.html.

[61] T. Dunigan, M. Mathis, and B. Tierney. A tcp tuning daemon. In *Proc. of Supercomputing: High-Performance Networking and Computing*, Nov. 2002.

[62] B. EcKart, X. He, and Q. Wu. Performance adaptive UDP for high-speed bulk data transfer on dedicated links. In *Proc. of the 22nd IEEE Int. Parallel and Distributed Processing Symp.*, Miami, Florida, Apr. 14-18 2008.

[63] S. Floyd. Highspeed tcp for large congestion windows. Request for Comments: 3649, Dec. 2003.

[64] Y. Gu and R. L. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Computer Networks*, 51:1777–1799, May 2007.

[65] R. Guerin, A. Orda, and D. Williams. QoS routing mechanisms and OSPF extensions. In *Proc. of Global Telecommunications Conf.*, pages 1903–1908, 1996.

[66] M. Guo and M. Ammar. Scalable live video streaming to cooperative clients using time shifting and video patching. In *Proc. of IEEE INFOCOM*, volume 3, pages 1501–1511, March 2004.

[67] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area metwork. In *The 16th IEEECS/ACM International Parallel and Distributed Processing Symposium*, Ft. Lauderdale, FL, Apr. 2002.

[68] T. J. Hacker, B. D. Noble, and B. D. Athey. Adaptive data block scheduling for parallel tcp streams. In *The 14th IEEE International Symposium on High Performance Distributed Computing*, 2005.

[69] T.J. Hacker and B.D. Athey. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. In *Proc. of the IEEE/ACM Int. Parallel and Distributed Processing Symp.*, 2002.

[70] T.J. Hacker, B.D. Noble, and B.D. Athey. Improving throughput and maintaining fairness using parallel TCP. In *Proc. of IEEE INFOCOM*, March 2004.

[71] E. He, J. Leigh, O. Yu, and T.A. DeFanti. Reliable blast UDP: predictable high performance bulk data transfer. In *IEEE Int. Conf. on Cluster Computing*, Chicago, Illinois, Sep. 23-26 2002.

[72] E. He, P. V. Primet, and M. Welzl. A survey of tranport protocols other than "standard" tcp. Technical report, Global Grid Form Report, 2005.

[73] X. Hei, C. Liang, J. Liang, Y. Liu, and K.W. Ross. A measurement study of a large-scale P2P IPTV system. *IEEE Tran. on Multimedia*, 2(8):1283–1292, March 2003.

[74] V. Jacobson. Congestion avoidance and control. In *Proc. of SIGCOMM*, pages 314–329, 1988.

[75] S. Jarvis, G. Tan, D. Spooner, and G. Nudd. Constructing reliable and efficient overlays for P2P live media streaming. *Int. J. of Simulation*, 7(2):54–63, March 2006.

[76] C. Jin, D.X. Wei, and S.H. Low. FAST TCP: motivation, architecture, algorithms, performance. In *Proc. of INFOCOM*, 2004.

[77] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for future high-bandwidth-delay product environments. In *Proceedings of ACM SIGCOMM'02*, Pittsburgh, PA, August 19-21 2002. Also see: www.acm.org/sigcomm/sigcomm2002/papers/xcp.pdf.

[78] D. Katz, K.Kompella, and D. Yeung. Rfc2370: Traffic engineering (te) extensions to ospf version 2. In *IETF RFC*, Sept. 2006.

[79] T. Kelly. Scalable tcp: Improving performance in highspeed wide area networks. In *PFLDnets*, Feb. 2003.

[80] M. Kim, S. Lam, and D. Lee. Optimal distribution tree for internet streaming media. In *Proc. of the 23rd Int. Conf. on Dist. Comp. Sys.*, page 116, 2003.

[81] E. Kohler, M. Handley, and S. Floyd. Designing dccp: Congestion control without reliability. In *Proc. of ACM SIGCOMM*, 2006.

[82] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. of the 19th ACM symposium on Operating Systems Principles*, 2003.

[83] F. Kuo and X. Fu. Probe-aided multcp: an aggregate congestion control mechanism. In *SIGCOMM Computer Communication Review*, pages 38(1):17–28, 2008.

[84] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach, 4/E*. Addison-Wesley, 2008.

[85] H. J. Kushner and D. S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, 1978.

[86] H. J. Kushner and C. G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, 2003. Second Edition.

[87] A. Kuzmanovic, E. Knightly, and R. L. Cottrell. Hstcp-lp: A protocol for low-priority bulk data transfer in high-speed high-rtt networks. In *PFLDnets*, Feb. 2004.

[88] A. Kuzmanovic and E. W. Knightly. Tcp-lp: A distributed algorithm for low priority data transfer. In *Proceedings of IEEE INFOCOM 2003*, San Francisco, CA, April 2003.

[89] M. Lasserre and V. Kompella. Rfc:4762, virtual private lan service (vpls) using label distribution protocol (ldp) signaling. In *IETF RFC*, Jan. 2007.

[90] R. Li, C. Guo, M. Fa, and Z. Wang. AHLSS: A hierarchical, adaptive, extendable P2P live streaming system. In *Int. Symp. on Advances in Computer and Sensor Networks and Systems*, pages 178–184, Apr. 2008.

[91] R. Love. *Linux Kernel Development*, chapter The Linux Process Scheduler. Sams Publishing, 2003.

[92] R. Love. *Linux Kernel Development, 3rd Edition*. Addison-Wesley Professional, 2010.

[93] S.H. Low, L.L. Peterson, and L. Wang. Understanding Vegas: a duality model. *J. of the ACM*, 49(2):207–235, Mar. 2002.

[94] S.H. Low, L.L. Peterson, and L. Wang. Understanding vegas: a duality model. *Journal of the ACM*, 49(2):207–235, March 2002.

[95] D. Lu, Y. Qiao, P. A. Dinda, and F. E. Bustamante. Modeling and taming parallel tcp on the wide area network. In *IPDPS*, 2005.

[96] X. Lu, Q. Wu, N. S. V. Rao, and Z. Wang. On performance-adaptive flow control for large data transfer in high speed networks. In *Proc. of the 28th IEEE Int. Performance Computing and Communications Conf.*, Phoenix, AZ, Dec. 14-16 2009.

[97] X. Lu, Q. Wu, N. S. V. Rao, and Z. Wang. Performance-adaptive prediction-based transport control over dedicated links. In *Proc. of the 6th Int. ICST Conf. on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Spain, Nov. 23-25 2009.

[98] Q. Ma, P. Steenkiste, and H. Zhang. Routing high bandwidth traffic in max min fair share networks. In *Proc. of ACM SIGCOMM*, pages 206–217, 1996.

[99] N. Magharei, R. Rejaie, and Y. Guo. Mesh or multiple-tree: A comparative study of p2p live streaming. In *Proc. of IEEE Infocom*, 2007.

[100] P. Merz and S. Wolf. TreeOpt: Self-organizing, evolving P2P overlay topologies based on spanning trees. In *Proc. of KiVS*, page 12, Feb. 2007.

[101] K. Nichols, S. Blake, F. Baker, and D. Black. Rfc2474: Definition of the differentiated services field. In *IETF RFC*, Dec. 1998.

[102] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. Almi: an application level multicast infrastructure. In *Proc. of the 3rd Conf. on USENIX Symp. on Internet Tech. and Sys. (USITS'01)*, page 5, 2001.

[103] R. Prasad, M. Jain, and C. Dovrolis. Socket buffer auto-sizing for high-performance data transfers. *Journal of Grid Computing*, 1(4):361–376, 2004.

[104] N. S. V. Rao and L. O. Chua. On dynamics of network transport protocols. In *Proc. of Workshop on Signal Processing*, 2002.

[105] N. S. V. Rao, Q. Wu, and S. S. Iyengar. On throughput stabilization of network transport. *IEEE Communications Letters*, 8(1):66–68, 2004.

[106] N.S.V. Rao, W. R. Wing, Q. Wu, N. Ghani, T. Lehman, and E. Dart. Measurements on hybrid dedicated bandwidth connections. In *INFOCOM2007 Workshop on High Speed Networks*, 2007.

[107] N.S.V. Rao, W.R. Wing, , S.M. Carter, and Q. Wu. Ultrascience net: Network testbed for large-scale science applications. *IEEE Communications Magazine*, 43(11):s12–s17, 2005. An expanded version available at www.csm.ornl.gov/ultranet.

[108] N.S.V. Rao, Q. Wu, S.M. Carter, and W.R. Wing. High-speed dedicated channels and experimental results with hurricane protocol. *Annals of Telecommunications*, 61(1-2):21–45, 2006.

[109] I. Rhee and L. Xu. CUBIC: a new TCP-friendly high-speed TCP variant. In *Workshop on Protocols for Fast Long-Distance Networks*, Lyon, France, Feb. 2005.

[110] M. Rio. A map of the networking code in Linux kernel 2.4.20. *Technical Report DataTAG-2004-1*, March 2004.

[111] Todd L.Montgomery Robert A.Van Valzah and Eric Bowden. *Topics In High-Performance Messaging*. 29West, Inc., 2009.

[112] E. Rosen, A. Viswanathan, and R. Callon. Rfc3031: Multiprotocol label switching architecture. In *IETF RFC*, Jan. 2001.

[113] J. Semke, J. Madhavi, and M. Mathis. Automatic tcp buffer tuning. In *Proceedings of ACM SIGCOMM*, August 1998.

[114] M. Singh, P. Pradhan, and P. Francis. Mpat: Aggregate tcp congestion management as a building block for internet qos. In *In Proceedings of the 12th IEEE International Conference on Network Protocols*, pages 129–138, Berlin, Germany, 2004.

[115] H. Sivakumar, S. Bailey, and R. L. Grossman. Psockets: the case for application-level network striping for data intensive applications using high speed wide area networks. In *Proc. of Supercomputing*, 2000.

[116] T. Small, B. Li, and B. Liang. On optimal peer-to-peer topology construction with maximum peer bandwidth contributions. In *Proc. of the 23rd Biennial Symposium on Communications*, pages 157–160, 2006.

[117] James C. Spall. A one-measurement form of simultaneous perturbation stochastic approximation. *Automatica*, 33(1):109–112, 1997.

[118] J.C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley Pub, 2003.

[119] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proc. of the 4th ACM/IEEE Symp. on Architectures for Networking and Communications Systems*, pages 107–120, 2004.

[120] R. Stewart and Q. Xie. Stream control transmission protocol. www.ietf.org/rfc/rfc2960.txt, Oct. 2000. IETF RFC 2960.

[121] NSF Teragrid. http://www.teragrid.org.

[122] D. Tran, K. Hua, and T. Do. Zigzag: an efficient peer-to-peer scheme for media streaming. In *Proc. of IEEE Infocom*, volume 9, pages 1672–1687, Dec. 2007.

[123] A. Veres and M. Boda. The chaotic nature of tcp congestion control. In *Proceedings of IEEE INFOCOM 2002*, 2002.

[124] J. Wang, L. Li, S. Low, and J. Doyle. Can shortest-path routing and tcp maximize utility. In *Proc. of IEEE Infocom*, pages 2049–2056, 2003.

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

[126] Q. Wu and N.S.V. Rao. Protocol for high-speed data transport over dedicated channels. In *Proc. of the 3rd Int. Workshop on Protocols for Fast Long-Distance Networks*, pages 155–162, Feb. 3-4 2005.

[127] Q. Wu, N.S.V. Rao, and S.S. Iyengar. Statistical effects of control parameters on throughput of window-based transport methods. In *Proceedings of the 12th International Conference on Computer Communications and Networks*, pages 587–590, 2003.

[128] Q. Wu, N.S.V. Rao, and S.S. Iyengar. Statistical effects of control parameters on throughput of window-based transport methods. In *Proc. of the 12th Int. Conf. on Computer Communications and Networks*, pages 587–590, 2003.

[129] Q. Wu, N.S.V. Rao, and X. Lu. On transport methods for peak utilization of dedicated connections. In *Proc. of the 6th Int. Conf. on Broadband Communications, Networks, and Systems*, Marid, Spain, Sept. 14-17 2009.

[130] C. Xiong, J. Leigh, E. He, V. Vishwanath, and T. Murata. Lambdastream - a data transport protocol for streaming network-intensive applications over photonic netwotks. In *Proc. of the 3th Int. Workshop on Protocols for FAST Long-Distance Networks*, Lyon, France, Sept. 2005.

[131] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control for fast long-distance networks. In *INFOCOM*, Hongkong, China, March 2004.

[132] M. Yang and Z. Fei. A proactive approach to reconstructing overlay multicast trees. In *Proc. of IEEE Infocom*, 2004.

[133] C. Yatin. Scattercast: an adaptable broadcast distribution framework. *Multimedia Syst.*, 9(1):104–118, 2003.

[134] C. Zhang, H. Jin, D. Deng, S. Yang, Q. Yuan, and Z. Yin. Anysee: Multicast-based peer-to-peer media streaming service system. In *Proc. of Asia-Pacific Conf. on Communications*, pages 274–278, 2005.

[135] X. Zhang, J. Liu, B. Li, and T.S.P. Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *Proc. of IEEE INFOCOM*, volume 3, pages 2102–2110, March 2005.

[136] Y. Zhang, Z.M. Mao, and J. Wang. A framework for measuring and predicting impact of routing changes. In *Proc. of IEEE INFOCOM*, Alaska, USA, 2007.

[137] Z.L. Zhang, Z. Duan, and Y.T. Hou. Decoupling QoS control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In *Proc. of ACM SIGCOMM*, 2000.

[138] X. Zheng, A.P. Mudambi, and M. Veeraraghavan. FRTP: Fixed rate transport protocol – a modified version of SABUL for end-to-end circuits. In *Proc. of Broadnets*, 2004.

[139] X. Zheng, M. Veeraraghavan, N.S.V. Rao, Q. Wu, and M. Zhu. Cheetah: Circuit-switched high-speed end-to-end transport architecture testbed. *IEEE Communications Magazine*, 43(11):s11–s17, 2005.

[140] Y. Zhu and B. Li. Overlay networks with linear capacity constraints. *IEEE Tran. on Parallel and Distributed Systems*, 19(2):159–173, Feb. 2008.

[141] Y. Zhu, B. Li, and K. Pu. Dynamic multicast in overlay networks with linear capacity constraints. *IEEE Tran. on Parallel and Distributed Systems*, 20(7):925–939, July 2009.