Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

AN INTEGRATED TRANSPORT SOLUTION TO BIG DATA MOVEMENT IN HIGH-PERFORMANCE NETWORKS

by
Daqing Yun

Extreme-scale e-Science applications in various domains such as earth science and high energy physics among multiple national institutions within the U.S. are generating colossal amounts of data, now frequently termed as "big data". The big data must be stored, managed and moved to different geographical locations for distributed data processing and analysis. Such big data transfers require stable and high-speed network connections, which are not readily available in traditional shared IP networks such as the Internet. High-performance networking technologies and services featuring high bandwidth and advance reservation are being rapidly developed and deployed across the nation and around the globe to support such scientific applications. However, these networking technologies and services have not been fully utilized, mainly because: i) the use of these technologies and services often requires considerable domain knowledge and many application users are even not aware of their existence; and ii) the end-to-end data transfer performance largely depends on the transport protocol being used on the end hosts. The high-speed network path with reserved bandwidth in High-performance Networks has shifted the data transfer bottleneck from network segments in traditional IP networks to end hosts, which most existing transport protocols are not well suited to handle.

In this dissertation, an integrated transport solution is proposed in support of data- and network-intensive applications in various science domains. This solution integrates three major components, i.e., i) transport-support workflow optimization, ii) transport profile generation, and iii) transport protocol design, into a unified framework. Firstly, a class of transport-support workflow optimization problems are

formulated, where an appropriate set of resources and services are selected to compose the best transport-support workflow to meet user's data transfer request in terms of various performance requirements. Secondly, a transport profiler named Transport Profile Generator (TPG) and its extended and accelerated version named `FastProf` are designed and implemented to characterize and enhance the end-to-end data transfer performance of a selected transport method over an established network path. Finally, several approaches based on rate and error threshold control are proposed to design a suite of data transfer protocols specifically tailored for big data transfer over dedicated connections. The proposed integrated transport solution is implemented and evaluated in: i) a local testbed with a single 10 Gb/s back-to-back connection and dual 10 Gb/s NIC-to-NIC connections; and ii) several wide-area networks with 10 Gb/s long-haul connections at collaborative sites including Oak Ridge National Laboratory, Argonne National Laboratory, and University of Chicago.

AN INTEGRATED TRANSPORT SOLUTION TO BIG DATA MOVEMENT IN HIGH-PERFORMANCE NETWORKS

by
Daqing Yun

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

August 2016

APPROVAL PAGE

AN INTEGRATED TRANSPORT SOLUTION TO BIG DATA MOVEMENT IN HIGH-PERFORMANCE NETWORKS

Daqing Yun

Dr. Chase Qishi Wu, Dissertation Advisor — Date
Associate Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Cristian Borcea, Committee Member — Date
Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Xiaoning Ding, Committee Member — Date
Assistant Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Guiling Wang, Committee Member — Date
Associate Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Yi Chen, Committee Member — Date
Associate Professor of School of Management, New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Daqing Yun

Degree: Doctor of Philosophy

Date: August 2016

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, New Jersey, 2016
- Master of Science in Computer Software and Theory,
  Xidian University, Xi'an, People's Republic of China, 2012
- Bachelor of Science in Software Engineering,
  Xidian University, Xi'an, People's Republic of China, 2009

Major: Computer Science

Publications:

D. Yun, C.Q. Wu, and M.M. Zhu. Transport-Support Workflow Optimization for Big Data Movement in High-performance Networks. Submitted to *the IEEE/ACM Transactions on Networking.*

S. Roy, **D. Yun**, B. Madahian, M.W. Berry, L.Y. Deng, D. Goldowitz, and R. Homayouni. Navigating the Functional Landscape of Transcription Factors via Tensor Analysis of MEDLINE Abstracts. Submitted to *the PLOS Computational Biology.*

Q. Liu, N.S.V. Rao, C.Q. Wu, **D. Yun**, R. Kettimuthu, and I.T. Foster. Measurements-Based Analysis of Performance Profiles and Dynamics of UDP Transport Protocols. In *Proceedings of the 24th IEEE International Conference on Network Protocols (ICNP '16)*, Singapore, November 8–11, 2016.

D. Yun, C.Q. Wu, N.S.V. Rao, Q. Liu, R. Kettimuthu, and E.S. Jung. Profiling Optimization for Big Data Transfer Over Dedicated Channels. In *Proceedings of the 25th International Conference on Computer Communication and Networks (ICCCN '16)*, Waikoloa, Hawaii, August 1–4, 2016.

D. Yun and C.Q. Wu. An Integrated Transport Solution to Big Data Movement in High-performance Networks. In *Proceedings of the 23rd IEEE International Conference on Network Protocols (ICNP '15)*, pp. 460–462, San Francisco, CA, November 10–13, 2015.

D. Yun, C.Q. Wu, and Y. Gu. An Integrated Approach to Workflow Mapping and Task Scheduling for Delay Minimization in Distributed Environments. *Journal of Parallel and Distributed Computing*, vol. 84, pp. 51–64, October 2015.

D. Yun, C.Q. Wu, N.S.V. Rao, B.W. Settlemyer, J. Lothian, R. Kettimuthu, and V. Vishwanath. Profiling Transport Performance for Big Data Transfer over Dedicated Channels. In *Proceedings of the 2015 International Conference on Computing, Networking and Communications (ICNC '15)*, pp. 858–862, Anaheim, CA, February 16–19, 2015.

T. Shu, C.Q. Wu, and **D. Yun**. Advance Bandwidth Reservation for Energy Efficiency in High-performance Networks. In *Proceedings of the 38th Annual IEEE Conference on Local Computer Networks (LCN '13)*, pp. 541–548, Novotel Central Sydney, Sydney, Australia, October 21–24, 2013.

D. Yun, Q. Wu, Y. Gu, and X. Liu. On an Integrated Mapping and Scheduling Solution to Large-scale Scientific Workflows in Resource Sharing Environments. In *Proceedings of the 46th Annual Simulation Symposium (ANSS '13)*, Article 7, 8 pages, San Diego, CA, April 7–10, 2013.

P. Brown, M. Zhu, Q. Wu, **D. Yun**, and J. Zurawski. A Workflow-based Network Advisor for Data Movement with End-to-end Performance Optimization. In *Proceedings of the 7th Workshop on Workflows in Support of Large-Scale Science (WORKS '12)*, pp. 73–81, Salt Lake City, UT, November 10–16, 2012.

P. Brown, M. Zhu, Q. Wu, **D. Yun**, and J. Zurawski. Exploring the Optimal Strategy for Large-scale Data Movement in High-performance Networks. In *Proceedings of the 31st IEEE International Performance Computing and Communications Conference (IPCCC '12)*, pp. 181–182, Austin, Texas, December 1–3, 2012.

D. Yun, Q. Wu, P. Brown, and M. Zhu. Modeling and Optimizing Transport-Support Workflows in High-performance Networks. In *Proceedings of the 37th Annual IEEE Conference on Local Computer Networks (LCN '12)*, pp. 384–391, Clearwater, Florida, October 22–25, 2012.

Q. Wu, **D. Yun**, X. Lin, Y. Gu, W. Lin, and Y. Liu. On Workflow Scheduling for End-to-end Performance Optimization in Distributed Network Environments. In *Proceedings of the 16th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '12)*, *Lecture Notes in Computer Science*, 7698, pp. 76–95, Shanghai, China, May 25, 2012.

To my beloved parents

贠学纯，谢晓梅

ACKNOWLEDGMENT

Professor Chase Wu brought me from Xi'an to Memphis in 2011, and since then I have been academically growing up under his supervision. I sincerely thank him.

I would like to express my gratefulness to my dissertation committee at New Jersey Institute of Technology: Dr. Cristian Borcea, Dr. Xiaoning Ding, Dr. Guiling Wang, and Dr. Yi Chen; and at University of Memphis: Dr. Lan Wang, Dr. Vinhthuy Phan, and Dr. Zhuo Lu. I am truly honored to have them as my committee members.

Dr. Nageswara S.V. Rao from Oak Ridge National Laboratory and Dr. Rajkumar Kettimuthu from Argonne National Laboratory have provided me with generous technical assistance and guidance in conducting some of the experiments presented in this dissertation.

The endless love of my grandparents, my parents, my uncle, my younger brother, and Olivia has been supporting me to conquer every challenge in my life. Any accomplishments of mine would be impossible and meaningless without them.

TABLE OF CONTENTS

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Next-generation scientific applications in various domains such as earth science and high energy physics among multiple national institutions [4] within U.S. are generating colossal amounts of data, now frequently termed as "big data", which must be stored, managed and moved to different geographical locations for distributed data processing and analysis [18, 23]. The successes of these collaborative applications highly depend on stable and high-bandwidth network connections, which, unfortunately, are not readily available in traditional shared IP networks. For example, on the Internet, very little guarantee can be provided on the transport performance and the resource availability is subject to constant changes due to concurrent network traffics.

High-performance Networks (HPNs) such as ESnet [6] and Internet2 [13] featuring dedicated connections with reserved high-bandwidth enabled by the recent development of high-performance networking technologies offer a promising solution to support data- and network-intensive applications. A number of high-performance networking projects are already under way to extend such capabilities to broad science communities. In recent years, significant progress has been made in various aspects [6] including the deployment of 100 Gb/s networks with future 1 Tb/s capacity, the increase in end-host capabilities with multiple cores and buses, the improvement in large-capacity disk arrays, and the use of parallel file systems such as Lustre [16] and GPFS [10]. For example, DOE ESnet and Advanced Networking Initiatives (ANI) network infrastructures [6] have recently been upgraded to 100 Gb/s to meet the long-haul network demands for such big data transfers.

Although having been developed and deployed at various locations, these networking services or resources only have a very limited scope of users at present and are far from being fully utilized. The main reason is that their use typically requires a considerable level of knowledge for network and end host system configurations, which most science domain experts lack. Oftentimes, scientific users are still using old-fashioned tools (e.g., `scp` over a default IP path) that they are familiar with from their empirical studies for their data transfer needs, even not being aware of the existence of such advanced networking resources.

Moreover, even if a dedicated channel is provisioned in high-performance networks and the advanced technologies and services are selected to use, the end-to-end data transfer performance still largely depends on the transport protocols/methods being used on the end hosts. Along with the emergence and proliferation of high-performance networks, various data transfer protocols have been rapidly proposed and developed, including TCP variants and enhancements [30,42,52,54,58,64,66,83,89,90] and UDP-based protocols such as Tsunami [22], RBUDP [53], RAPID/RAPID$^+$ [26, 36], PA-UDP [39], SABUL [47], and UDT [49]. Maximizing their throughput performance over complex high-speed connections is still very challenging: i) their optimal operational zone is affected by the configurations and dynamics of the network, the end hosts, and the protocol itself, ii) their default parameter setting does not always yield the best performance, iii) application users, who are domain experts, typically do not have necessary knowledge to choose which transport protocol to use and which parameter value to set, and iv) the data transfer bottleneck in high-performance networks shifts from network segments as observed on the Internet to the end hosts, which most existing transport protocols are not well suited to handle. Consequently, application users have not seen the corresponding increase in transport performance especially in terms of application-level goodput despite the bandwidth upgrades in the backbones of high-performance networks.

1.2 An Integrated Transport Solution

In view of the above challenges and limitations, we propose an integrated transport solution to big data movement in HPNs, which integrates three major components, i.e., i) transport-support workflow optimization, ii) transport profile generation, and iii) transport protocol design, into a unified framework. The ultimate goal is to provide users an integrated solution for host and network resource discovery, end-to-end path composition and establishment, transport profile generation, and actual data transfer to support large-scale scientific applications in various science domains.

Figure 1.1 shows the framework of the proposed transport solution. Within this framework, the user needs to submit a request that describes the desired data transfer services such as the start and end time, the data source and destination nodes, a desirable or target bandwidth, and possibly a financial cost limit on the deployment and utility expenses. Upon the receipt of such a request, our solution first invokes Network-aware Data Movement Advisor (NADMA) [31, 32] to explore and discover available services and resources, which are modeled as transport-support workflow modules and maintained in a database. Based on these modules, the transport-support workflows are constructed and optimized to compose the best end-to-end data transfer path and select the most appropriate transport methods that perform actual data transfer. The Transport Control System (TCS) takes the established end-to-end path and the selected transport method as input, conducts transport profiling on the selected transport method using Transport Profile Generator (or `FastProf` if necessary) to determine the optimal zone of the control parameters that may have effects on the data transfer performance. The actual data transfer is then conducted using the selected transport method with the tuned parameter values. Optionally, a notification may be sent to user after the data transfer is completed, and meanwhile, corresponding performance measurements are stored in the database for future use.

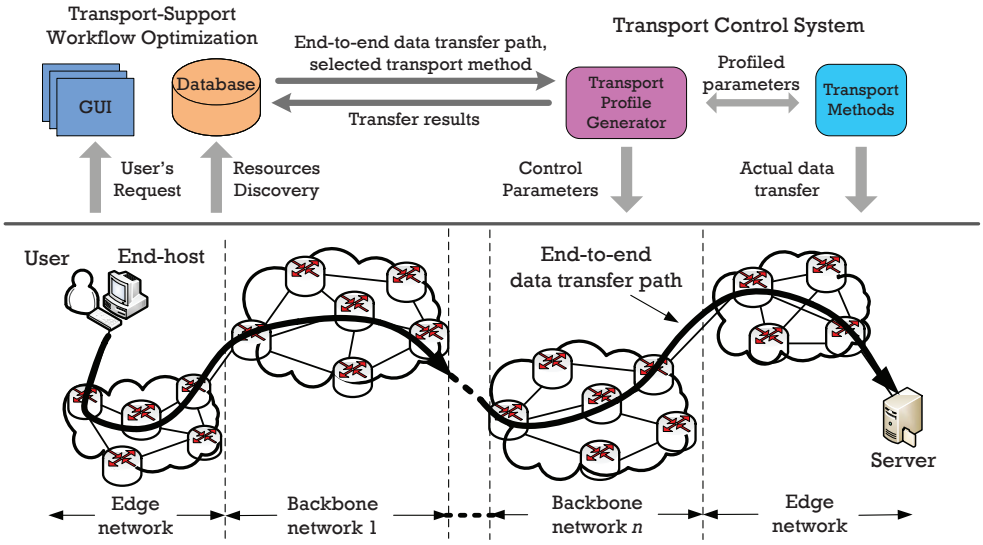


Figure 1.1 Framework of a workflow-based transport solution.

1.3 Main Contributions

We summarize the technical contributions of this dissertation research as follows.

- We model the network resources and services as transport-support workflow modules and formulate a class of transport-support workflow optimization problems for end-to-end data transfer path composition as well as transport method selection. We prove the formulated problem to be NP-complete and design optimal pseudo-polynomial algorithms.

- We design and implement a Transport Profile Generator (TPG) to characterize and enhance the performance of existing transport protocols. TPG is provided to end users as a command-line tool to conduct one-time profiling, which supports both multiple parallel data streams and multiple NIC-to-NIC connections. We further develop `FastProf`, a stochastic approximation-based transport profiler, to accelerate the profiling process for big data transfer in high-performance networks. `FastProf` significantly reduces the profiling overhead while achieving a comparable level of end-to-end throughput performance with the exhaustive search-based approach.

- We model and analyze the performance of the Tsunami UDP protocol over dedicated channels, and then propose several approaches based on rate and error threshold control to design a suite of data transfer protocols specifically tailored for big data transfer in high-performance networks.

CHAPTER 2

TRANSPORT-SUPPORT WORKFLOW OPTIMIZATION

2.1 Introduction

Big data transfer in distributed scientific applications requires high-speed network infrastructure where data packets are transmitted across various network segments such as edge and backbone networks from a source to a destination, as shown in Figure 2.1. Generally, to meet a specific request, we must take multiple steps to acquire and deploy appropriate system hardware/software, select suitable technologies based on available resources, determine the best data transfer path, and perform the actual data transfer, as shown in Figure 2.2. The system and network resources vary significantly in their type, cost, performance, reliability, and security. For example, an end host might be equipped with network interface cards (NICs) of different speed and cost; OSCARS in ESnet [7] and ION in Internet2 [14] provide different levels of provisioning services at different cost and admission rates.

The goal of our work is to develop an integrated solution for resource discovery and path composition to support such big data transfer. In our transport framework, a user only needs to submit a request that describes the data transfer requirements

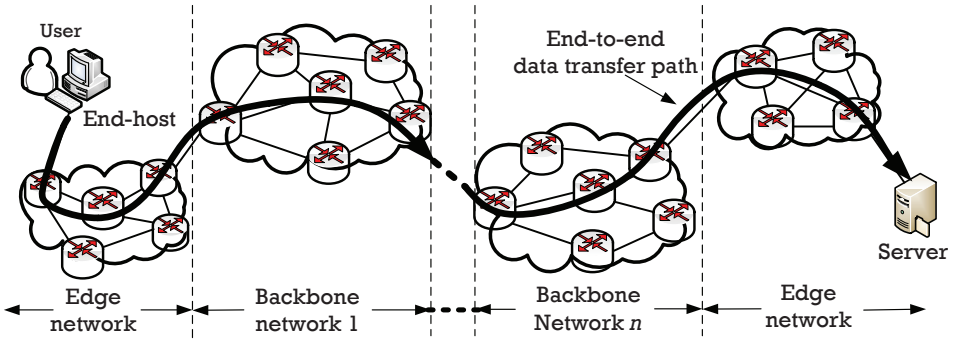


Figure 2.1 Network infrastructure for wide-area data transfer.

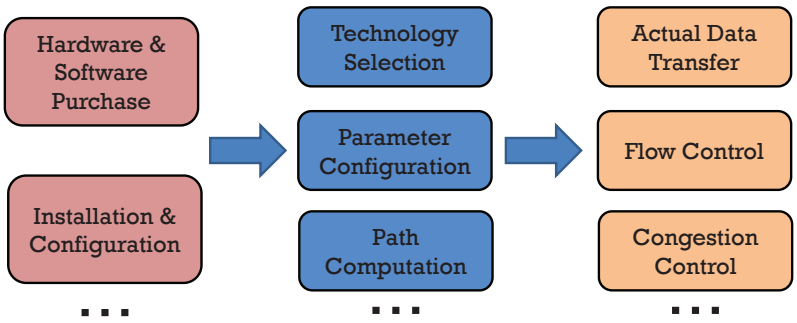


Figure 2.2 General steps in a data transport solution.

such as the service start and end time, the data source and destination nodes, a desirable (target) bandwidth, or possibly a financial cost limit on the deployment and utility expenses[1]. Upon the receipt of such a request, our solution first invokes NADMA [31,32] to explore available services and resources within end systems, edge segments, and backbone networks, and then models and profiles them as transport-support workflow modules with quantified parameters. Based on these modules, we formulate a class of workflow optimization problems with different objectives such as (financial or technical) cost, delay, throughput, bandwidth, reliability, and security requirements. Note that some objectives may be in conflict with each other, which makes the problem nontrivial.

Typically, users want their data to be transferred at a low cost (both financial and technical), at a fast speed, and in a reliable and secure manner; in other words, users are generally greedy, but oftentimes these requirements cannot be met simultaneously. For example, there is an obvious tradeoff between the cost and the speed under a normal circumstance. Furthermore, the modules selected in different segments must match well to achieve an overall good transport performance. In an extreme case, an old end host equipped with a low-speed NIC or CPU

[1]Even though most network services and resources are not free, their financial cost is quite minimal and is often negligible, especially in shared IP networks. Some advanced services such as OSCARS and ION are currently free to authorized users, but it is predictable that some accounting components will be integrated into these services in the future.

may become the bottleneck of the entire transport path and therefore limit the achievable throughput, no matter how much bandwidth is reserved in the backbone or edge networks. The solution to this optimization problem would select a set of appropriate transport-support workflow modules to composite and establish an end-to-end network path together with a suitable transport method selected to perform the actual data transfer.

We prove the formulated transport-support workflow optimization problem to be NP-complete and design optimal pseudo-polynomial algorithms. We evaluate the proposed algorithms using simulations in comparison with a greedy approach, and also conduct proof-of-concept experiments in wide-area networks to validate the cost models and illustrate the efficacy of the proposed workflow-based transport solution.

2.2 A Brief Description of Related Projects

The importance of dedicated channels for big data transfer has been well recognized and several network research projects funded by different agencies are developing such bandwidth provisioning capabilities, as summarized below[2].

UltraScience Net, developed at Oak Ridge National Laboratory, is a wide-area experimental network testbed capable of provisioning dedicated channels through layer-2 switching to support large-scale computational science applications [74].

TeraPaths [29] at Brookhaven National Laboratory (BNL) offers a service to create end-to-end virtual paths together with guaranteed bandwidth for specific data streams. It is a fully-distributed system, dealing with the problem of supporting efficient, reliable, predictable petascale data movement in modern, high-speed networks whose virtual paths prioritize, protect, and throttle network flows in accordance with site agreements and user requests.

[2]More details about these projects can be obtained from [1, 3, 5, 9, 11, 12, 15, 24, 62, 74, 84].

Circuit-switched High-speed End-to-End Transport ArcHitecture (CHEETAH) is an add-on service to the primary Internet connectivity service by providing end hosts with high-speed, end-to-end circuit connectivity based on a call-by-call sharing [93]. The "circuit" includes Ethernet segments at the ends, which are mapped into Ethernet-over-SONET long-distance circuits.

On-Demand Secure Circuits and Advance Reservation System (OSCARS) [7, 51] is a prototype service enabling advance reservation of secure virtual circuit with guaranteed bandwidth within ESnet [6]. The management and operation of virtual circuits within the network are implemented at layer 3 using Multi-Protocol Label Switching (MPLS) [25] and Resources Reservation Protocol (RSVP) to create virtual circuits or Label Switched Paths (LSPs).

Interoperable On-demand Network (ION) provisions dedicated circuits across the Internet2 and other networks [13, 14]. It uses community-developed technologies and protocols to provide on-demand, dedicated paths between end hosts.

B4 [55] is a private WAN that connects Google's data centers around the globe for big data transfer. It adopts a software-defined networking architecture for the data center interconnect, and uses OpenFlow [31, 67] to manage switches and realize centralized traffic control.

2.3 Resource Discovery

We use Network-Aware Data Movement Advisor (NADMA) [31, 32] to explore and discover available services and resources, which are modeled as transport-support workflow modules and maintained in a database. NADMA is a client-end program that interacts with existing data and storage management systems, discovers network and system resources, and advises application users of efficient strategies for successful and high-speed data transfer. Based on discovered resources, NADMA composes a

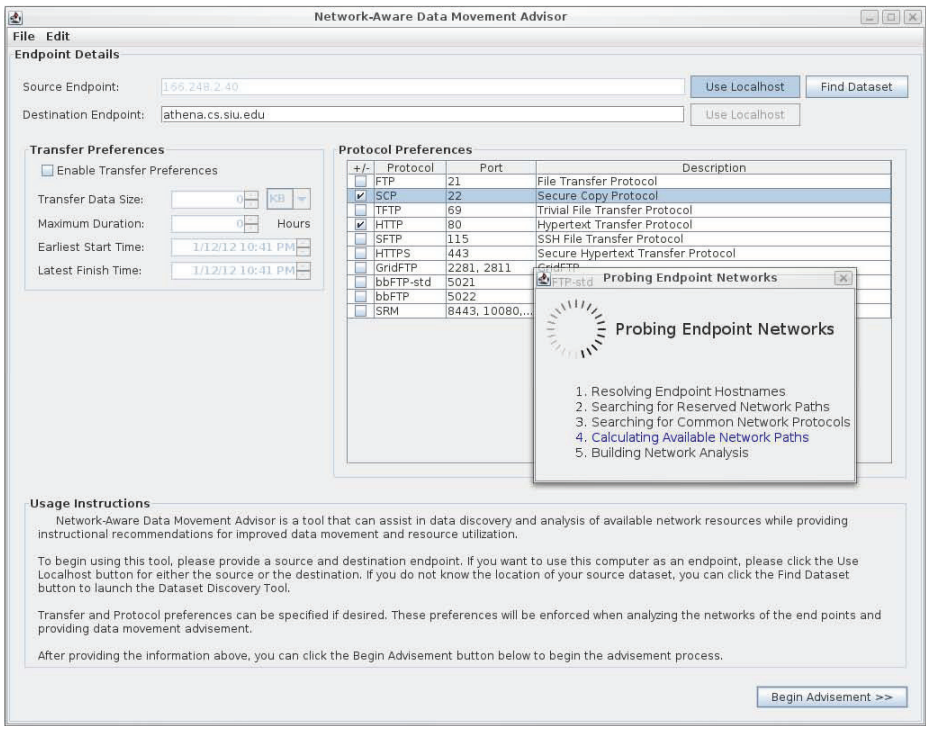


Figure 2.3 Graphical user interface of NADMA [31].

series of feasible route options with performance estimations and provides detailed steps for users to authorize and execute data transfer.

NADMA provides a Graphic User Interface (GUI) for users to submit data transfer requests based on various transport methods such as SRM, GSIFTP, HTTP, HTTPS, BBFTP, SCP, and SFTP. The interface also displays the detailed information of discovered end hosts and network segments as well as the resultant guidelines for data movement. As shown in Figure 2.3, the user specifies the source and destination nodes, between which the data transfer should be performed. This is the minimal information NADMA requires to perform its initial network and storage resource discovery process. Except the source and destination hosts, others such as authentication and protocol-specific information are not required and additional information about the characteristics of the data to be transferred may be optionally provided by the end user. Within NADMA, a user-desired data transfer service is

determined by specifying the following: i) the size of data that needs to be transferred; ii) the maximum desired duration of the transfer; iii) the (expected) earliest start time of the transfer; and iv) the (expected) latest time of the data transfer must complete.

When searching for possible data movement strategies, NADMA takes these parameters into consideration and provides the user with prioritized well-structured data movement options. NADMA probes the availability of popular protocols at predefined ports. In the current version, the path composition is done manually, and the user has to select the best possible route among all feasible route options. As domain experts, science users may not have sufficient knowledge in networking or system domains to understand the performance or capability of each workflow module to make an informative choice. Our work is to automate this path composition process by modeling workflow modules with quantified parameters and presenting to the user the appropriate transport path based on the given performance requirements.

2.4 A Simple Example of Transport-Support Workflow

This transport solution involves three tasks in this part: i) workflow module modeling, ii) transport-support workflow construction, and iii) performance optimization. Note that the workflow in this context is essentially a step-by-step guide for users to compose an end-to-end network path for data transfer. Since different network segments (different steps of the procedure) use different services with different costs and performance metrics, we must create appropriate cost models that reflect the characteristics of such resources and services. In order to create a transport-support workflow, we divide the entire data transfer process into K *zones*, and categorize those workflow modules (resources modules and services modules) into one of them, as shown in Figure 2.4. Each module represents a certain type of task that needs to

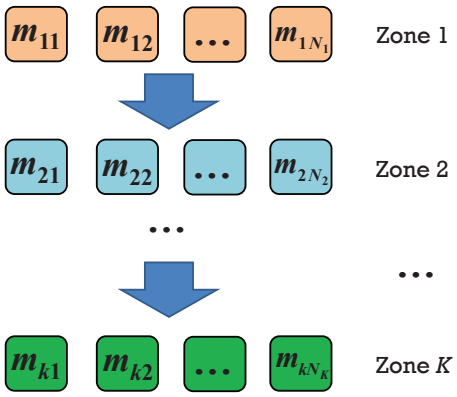


Figure 2.4 Zone-based transport-support workflow structure.

be performed to meet the user's data transfer request, and there might exist execution dependency between some of these modules.

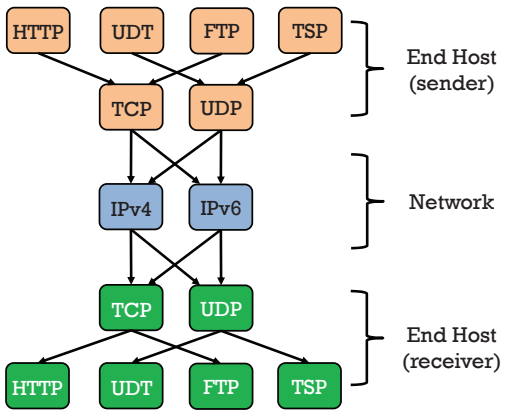


Figure 2.5 A simple example of transport-support workflow.

We shall use an extremely simplified transport-support workflow example to illustrate our approach. Let us consider a user request for reliable and TCP-friendly data transfer from source host "`sender`" to destination host "`receiver`", both of which are connected to the Internet. Given the protocols detected on the hosts and the resources available in the network, we could categorize them into $K = 3$ zones, as shown in Figure 2.5. In this simple case, one can easily construct a transport-support workflow, i.e., HTTP $\rightarrow$ TCP $\rightarrow$ IPv4 $\rightarrow$ TCP $\rightarrow$ HTTP, which is the default IP

path on the Internet, and the selected TCP protocol ensures the fairness in resource sharing with concurrent traffic. However, since there may exist multiple services of various types in high-performance network environments, it is challenging to select appropriate modules to compose a satisfactory workflow, especially when some conflictive performance requirements are specified simultaneously in the user request. In fact, in most high-performance networks, data packets are not carried by a single TCP stream over the default IP path, but oftentimes by multiple concurrent TCP or UDP streams over dedicated channels established by certain bandwidth provisioning services, which introduce inter-stream competition that may lead to complex transfer dynamics even over dedicated connections.

2.5 Cost Models

In this section, we model various services and resources in different segments of the network and end host as transport-support workflow modules.

2.5.1 End Host Modules

Figure 2.6 shows a general structure of end host modules, which are divided into 3 zones, namely, system resources, transport methods, and user applications.

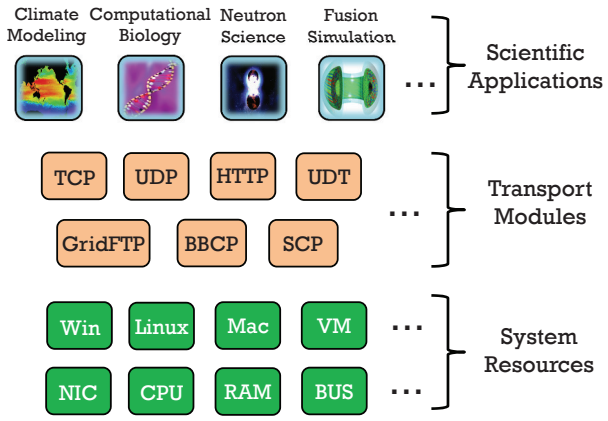


Figure 2.6 General structure of end host modules.

The modules in the system resource zone include both hardware such as CPU, network interface card (NIC), and random access memory (RAM), and system software such as operating systems. The modules in the transport modules zone include application-layer transport protocols, kernel-level transport protocols, and other network services and resources. We model the application-layer protocol HTTP as a module that runs over the kernel-level transport protocol TCP, which is also modeled as a module in transport modules zone. We place them in the same zone as both of them are transport protocols providing services to user applications.

Table 2.1 Parameters of End Host Modules

Parameters	**Remarks**
Financial cost	Purchase/install of hardware/software
CPU cores	Affinities
CPU cycles	Time-varying
Memory space	Time-varying
NIC	Speed, ring buffer, IRQ coalesce
BUS	Speed, connectivity
Stability	Data transfer protocols/applications
Reliability	Data transfer protocols/applications
Security	`ssh` library
Packet loss rate	Measured at end hosts
Delay	Round-trip time, one-way delay
Jitter	Measured at end hosts
Max packet size	MTU, MSS, `UDT_MSS`, etc.

We would like to point out that this zone-based structure is flexible in that we can further divide the modules in each zone into more sub-zones as in the case of TCP/IP stack as shown in Figure 2.6. The modules in the user application zone include all user applications that require data transfer services. These applications come from a wide range of disciplines spanning from climate research, nanoscience, astronomy, neutron sciences, high energy physics, computational materials, fusion simulation, to computational biology.

The packet receiving at the end host generally involves three steps across all the three zones: i) a data packet arrives at the NIC and generates an interrupt, ii) the kernel traps the interrupt and reads the packet from the NIC's buffer to the transport protocol buffer, and iii) the transport protocol processes and forwards the packet to the target user application. The parameters we may consider for end host modules are tabulated in Table 2.1.

2.5.2 Networking Service Modules

In our model, a networking service is a technology, mechanism, hardware, or software system, which takes the user's request as input, performs certain predefined routines, and sends back to the user the resources and/or other relevant results under request.

Table 2.2 Networking Services and Resources

Modules	**Remarks**
USN	DOE/DOD Ultra-Science Net
OSCARS	Bandwidth reservation in ESnet
ION	Bandwidth reservation on Internet2
DYNES	Edge network bandwidth reservation for Internet2
DRAGON	Resource allocation via GMPLS optical networks
CHEETAH	Circuit-switched optical network infrastructure
TeraPaths	End-to-end virtual path with bandwidth guarantees
ESCPS	Dynamic provisioning of inter-domain circuits
UCLP	Network resources treated as software objects
JGN (2/2plus)	Fully-fledged next-generation testbed
Geant2	NRENs and EC network testbed for research
GENI	Virtual laboratory for exploring future Internet
B4	Google's globally-deployed software-defined WAN

We list in Table 2.2 the commonly existing networking service modules, each of which could provide users either a default IP or a network provisioning service with guaranteed bandwidth. These modules utilize graph-based algorithms to compute a path for a reservation request. A user request typically includes several parameters

such as the start time and end time of the demanded service, the source host address and destination host address of the data transfer, the required bandwidth, and in some cases, loss rate, transfer reliability, and transfer security. To compute the data transfer path, in addition to the user request, these networking service modules also take as input the network topology with capacity information and the current resource reservation status, as shown in Figure 2.7. The parameters of networking modules we may consider are listed in Table 2.3.

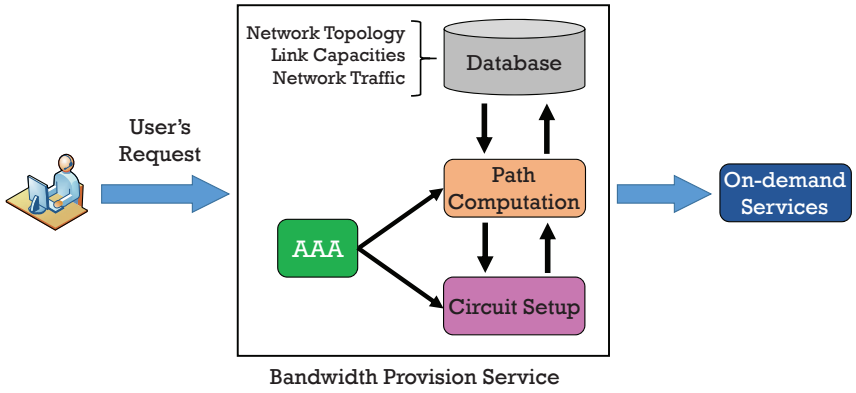


Figure 2.7 Networking service module.

Table 2.3 Parameters of Networking Service Modules

Parameters	Remarks
Start time	Of the required service
End time	Of the required service
Source	IP address or hostname
Destination	IP address or hostname
Data size	Size of data to be transferred
Bandwidth	Desired or target bandwidth/speed
Cost	To use services such as OSCARS
Network speed	Decided by the bottleneck link
Stability	Network connections/links
Reliability	Network connections/links
Security	Transfer security level
Delay	Typically from 0 ms to 380 ms
Jitter	Measured within networks
Topology	Used for path computation
Link capacity	Connection capacity
Reserved bandwidth	Allocated in advance, target rate

2.6 Technical Approach

In this section, we formally define the Transport-Support Workflow Optimization Problem (TSWOP), prove it to be NP-complete, and design optimal pseudo-polynomial algorithms to select a subset of transport-support workflow modules for the composition of an end-to-end data transfer path. Please refer to Table 2.4 for the notations used in the definition of TSWOP.

2.6.1 User Request

A user request r specifies the desired data transfer service such as transfer start time t_s, transfer finish time t_e, source host address h_s, destination host address h_r, and transfer data size DS as well as some data transfer constraints and objectives such as the required or target bandwidth B and upper bound of the (financial) cost $\mathcal{C}_{max}$ for the service/resource use. Although a user request may have its specific objectives and constraints, there exist some general ones, as listed in Table 2.5. We model a generic user request as an n-tuple, i.e.,

$$r = (r_1, r_2, ..., r_n), \tag{2.1}$$

where r_k $(1 \leq k \leq n)$ are the user-specified parameters of the desired services or constraints such as $\mathcal{C}_{max}$, t_s, t_e, h_s, h_r, DS, and B, as detailed in Table 2.1 and Table 2.3. Note that a specific user request may involve only a subset of parameters, in which case, we assign 0 or `null` to other parameters that are not under consideration. Figure 2.8 shows a user request r $(n = 10)$ that asks for the following data transfer service: reliably (see the 8th parameter in Figure 2.8) and securely (see the 9th parameter in Figure 2.8) move 1.0 TB data of file `/dir/to/srcfile` on sender `tubes` to the folder `/dir/to/dstfile` on receiver `midway` at a target rate of 5.0 Gb/s during a time window from "00:00:00" to "00:30:00".

Table 2.4 Notations Used in the Formulation of TSWOP

Notations	Definitions
t_s	Start time of data transfer
t_e	End time of data transfer
h_s	Sender host
h_r	Receiver host
r	User request
n	Number of elements in a user request
α	Weight vector corresponding to r
DS	Data size to be transferred
B	Target (expected) bandwidth
$G(M, E)$	A DAG-structured transport-support workflow
K	Number of zones in $G(M, E)$
M	The set of modules in G
$m_{i,j}$	The ith module at the jth zone
$p_{i,j}$	Profit vector corresponds to module $m_{i,j}$
$\mathcal{P}_{i,j}$	Profit of module $m_{i,j}$
$\mathcal{C}_{i,j}$	Cost of module $m_{i,j}$
d	Edge density of G
E	The set of dependencies in G
$e_{(i_1,j_1),(i_2,j_2)}$	The dependency between m_{i_1,j_1} and m_{i_2,j_2}
$\mathcal{C}_{max}$	Cost constraint
$\mathcal{C}_{RGMS}$	The cost of the path calculated by RGMS
$\mathcal{C}_{G_{max}}$	The cost of the longest cost path in G
$\mathcal{C}_{G_{min}}$	The cost of the shortest cost path in G
$\mathcal{D}(\cdot)$	Intermediate maximal achievable profit
$x_{i,j}$	Binary variable of module selection
K'	Number of disjoint classes of items in MCKP
$m'_{i,j}$	The ith item of the jth class in MCKP
$\mathcal{P}'_{i,j}$	Value of item $m'_{i,j}$ in MCKP
$\mathcal{C}'_{i,j}$	Weight of item $m'_{i,j}$ in MCKP
$\mathcal{C}'_{max}$	Capacity of knapsack in MCKP
$x'_{i,j}$	Binary variable of item selection in MCKP

Table 2.5 Data Transfer Objectives and Constraints

Objectives	**Remark**
Loss rate	Low
Failure rate	Low
Throughput	High
Energy-efficiency	High
Transfer delay	Low
Objectives	**Remark**
Bandwidth	As required
Cost	Upper bound
Reliability	As required
Security level	As required
Transfer stability	Required with a target rate

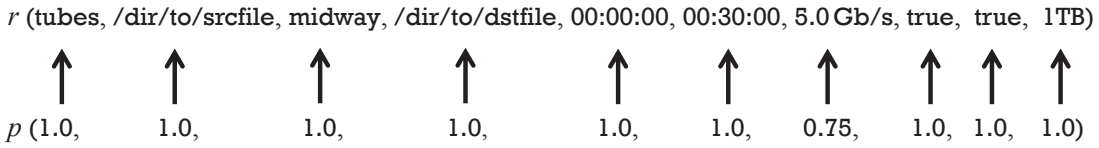


Figure 2.8 User request and profit vector.

2.6.2 Profit Vector

As illustrated in Figure 2.4, we categorize the tasks involved in the entire data transfer process into K zones (or layers[3]), in each of which, there may exist multiple modules that can perform the corresponding functions at a certain level. To meet a given data transfer request, we need to select appropriate modules from each zone to accomplish the task involved in each data transfer step. Depending on the module's properties, the parameters r_i in the user request may be fulfilled partially or completely by the selected module, which reflects the degree of satisfaction for the data transfer request.

Consider K zones with N_j modules in the jth zone ($1 \leq j \leq K$) as shown in Figure 2.4. Our goal is to select a subset of modules across all K zones to maximize the satisfaction of a given user request. For a request r, we define a profit vector $p_{i,j}$

[3]The terms "zones" and "layers" are used interchangeably in this context.

corresponds to module $m_{i,j}$ (i.e., the ith module in the jth zone) as

$$p_{i,j} = (p_{i,j,1},\ p_{i,j,2},\ ...,\ p_{i,j,n}), \tag{2.2}$$

where $p_{i,j,k} \in [0, 1]$ ($k = 1, 2, \cdots, n$) is i) 1 when $m_{i,j}$ is selected from its zone and satisfies the user request parameter r_k completely[4]; ii) 0 when $m_{i,j}$ is selected but it does not fulfill r_k at all[5]; and iii) between 0 and 1 when $m_{i,j}$ satisfies r_k partially[6]. Obviously, the profit vector is specific to a given data transfer scenario including various factors such as host and system configurations and network connection properties. In practice, we use a profiling approach to obtain such profit vectors, as further discussed in Section 2.6.3. Figure 2.8 shows an example of profit vector p, which represents a module that completely satisfies all the requirements in request r except r_7, which is set to be 0.75 because the selected protocol is only able to achieve an average throughput of 3.75 Gb/s from `tubes` to `midway` for a given target rate of 5.0 Gb/s, based on the historical profiling data.

Ideally, we wish to select modules that completely satisfy a user request for all of its objectives and constraints. However, due to network resource limitations and potentially conflictive parameters, it is generally infeasible to select such modules since application users are not expected to always provide reasonable or realistic requests (in many cases, users tend to be greedy).

Given a user request $r = (r_1, r_2, \cdots, r_n)$, we calculate the profit $\mathcal{P}_{i,j}$ of a selected module $m_{i,j}$ as

$$\mathcal{P}_{i,j} = \sum_{k=1}^{n} \alpha_k \cdot p_{i,j,k}, \tag{2.3}$$

[4]For example, when a user sets the target rate to be 5.0 Gb/s, if a protocol is selected and is able to transfer at a speed no less than 5.0 Gb/s, then the selected protocol completely satisfies the target rate requirement.

[5]For example, UDP cannot fulfill the reliability requirement of a data transfer.

[6]For example, for a target rate of 5.0 Gb/s, if the selected tool is only able to transfer data at an average speed of 2.5 Gb/s, then the corresponding $p_{i,j,k}$ is set to be $\frac{2.5}{5} = 0.5$.

where α_k is the weight for each component service parameter r_k that indicates the importance of r_k and could be either manually specified by the user or automatically assigned by our model. Intuitively, $\mathcal{P}_{i,j}$ reflects how well module $m_{i,j}$ satisfies user request r if selected, and a larger value of $\mathcal{P}_{i,j}$ indicates a better satisfaction.

2.6.3 Discussion on Profit Vector Estimation

Given a workflow module, it is critical to determine the parameter values of p since it affects module selection and eventually transport performance. Due to the complex properties of networking services and dynamic requirements of various applications, it is difficult to determine the parameters through a uniform approach.

Some of these parameters may be straightforward to determine while others may not. For example, it is relatively easier to determine if a module is able to provide a reliable data transfer service than to ensure the satisfaction of a certain failure rate or security level requirement. For example, TCP-based transport methods provide reliable data transfer, while UDP-based transport methods generally do not unless an application-level retransmission mechanism is implemented, as in UDP-based Data Transfer (UDT) protocol [48].

For applications that do not require reliable data transfer, the requirement on packet loss is not very critical and is oftentimes application-dependent, e.g., for a real-time video steam, a 5% loss rate might be tolerable; while for applications that require reliable data transfer, a loss rate of 5% makes it almost impossible to yield a satisfactory performance using reliable transport protocols such as TCP, especially when data transfer is conducted over long-haul high-speed connections. In such cases, loss rate clearly has a higher importance value (i.e., the value of α) and should be considered as a more critical performance metric.

It is not straightforward to determine the parameters of p for hardware and system resource modules on end hosts due to time-varying workloads and system

dynamics, which must be explicitly accounted for. Some other requirements are even more difficult to determine. For example, OSCARS [6, 7] uses a topology-based graph algorithm to determine the path for a circuit reservation request, and the topology and capacity information is obtained from network devices every hour and then imported into the OSCARS topology database. When a user request is received, OSCARS generates a base topology graph from the database taking into account any existing reservation whose time ranges conflict with the new one. Subsequently, path computation is performed on the base topology graph considering the parameters and constraints specified in the reservation request such as source and destination hosts, required bandwidth or VLAN tagging. Since OSCARS replies to the user with a notification of success or failure, it is straightforward in such cases to determine the elements in p corresponding to the required bandwidth and dedicated channels. If the user submits a request with certain requirements on failure rate or loss rate, it would be difficult for OSCARS to determine if such requirements can be met as they also depend on other components along the data transfer path as well as the current status of the networks. A feasible approach is to use historical and profiling data to estimate and predict the performance of a service, which can be further used to estimate and determine the corresponding parameters in p.

2.6.4 Module Dependencies

As exemplified in Figure 2.5, among the transport-support workflow modules that are categorized into K disjoint zones, there exist certain dependencies between the modules in adjacent zones. We model such dependencies as directed edges associated with binary variables $e_{(i_1,j_1),(i_2,j_2)}$, whose value is 1 if and only if there is a dependency from module m_{i_1,j_1} to module m_{i_2,j_2}. Such dependencies represent the precedences among the available resources and services.

2.6.5 Transport-Support Workflow Optimization Problem

We formally define the Transport-Support Workflow Optimization Problem (TSWOP) as follows.

Definition 1. *Given*

- *a user data transfer request* $r = (r_1, r_2, \cdots, r_n)$ *that contains* n *elements with an associated weight vector* $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_n)$,
- *a set* $M = \{m_{i,j}\}$ *of transport-support workflow modules that are categorized into* K *zones, where zone* j *contains* N_j *modules* ($i = 1, 2, \cdots, N_j$; *and* $j = 1, 2, \cdots, K$),
- *a set* $E = \{e_{(i_1,j_1),(i_2,j_2)}\}$ *of dependencies from adjacent modules* m_{i_1,j_1} *to* m_{i_2,j_2},
- *a profit vector* $p_{i,j} = (p_{i,j,1}, p_{i,j,2}, \cdots, p_{i,j,n})$ *and a (financial) cost* $\mathcal{C}_{i,j}$ *for each module* $m_{i,j}$ *(the* i*th module in the* j*th zone), and*
- *a cost constraint* $\mathcal{C}_{max}$,

we wish to choose a subset of modules across all K *zones to compose an end-to-end data transfer path to meet the user request* r *with the maximum profit:*

$$\max \sum_{j=1}^{K} \sum_{i=1}^{N_j} \mathcal{P}_{i,j} \cdot x_{i,j} = \max \sum_{j=1}^{K} \sum_{i=1}^{N_j} \sum_{k=1}^{n} \alpha_k \cdot p_{i,j,k} \cdot x_{i,j}, \tag{2.4}$$

subject to

$$\sum_{j=1}^{K} \sum_{i=1}^{N_j} \mathcal{C}_{i,j} \cdot x_{i,j} \leq \mathcal{C}_{max}, \tag{2.5}$$

$$\sum_{i=1}^{N_j} x_{i,j} = 1, 1 \leq j \leq K, \tag{2.6}$$

$$x_{i,j} \in \{0, 1\}, 1 \leq i \leq N_j, 1 \leq j \leq K, \tag{2.7}$$

$$\prod_{j=1}^{K-1} e_{(i_j^*, j),(i_{j+1}^*, j+1)} = 1, x_{i_j^*, j} = 1, 1 \leq j \leq K-1. \tag{2.8}$$

In the above definition, we use a binary variable $x_{i,j}$ to denote the module selection, which is 1 if the ith module is selected from the jth zone, and is 0, otherwise (see Equation 2.7).

The constraints in Equation 2.6 and Equation 2.8 ensure that the module selection always results in a path from zone 1 to zone K in the workflow. Equation 2.6 ensures that one and only one module is selected from each zone. In Equation 2.8, we use i_j^* to denote the index of the module selected from the jth zone, i.e., $x_{i_j^*,j} = 1$ indicates $m_{i_j^*,j}$ is selected from zone j. Thus Equation 2.8 guarantees that if two selected modules belong to adjacent zones, there must be a directed edge between them. This constraint applies to any two modules in adjacent zones (i.e., zone j and zone $j+1$ $(1 \leq j \leq K-1)$) and ensures the selected modules to form a path from zone 1 to zone K. In Equation 2.5, the constraint $\mathcal{C}_{\max}$, which is specified in r, could be the financial cost for deploying network devices or using network resources. The lower part of Figure 2.9 shows an example problem instance of TSWOP with 14 modules that are categorized into 5 layers and the modules in adjacent zones are fully-connected. Without loss of generality, we add a virtual start module $m_{1,0}$ and a virtual end module $m_{1,6}$ in the workflow with zero cost and zero profit.

2.6.6 Complexity Analysis

As shown in Figure 2.4, each zone stands for a segment in the entire process of data transfer. Since there exist dependencies between adjacent modules, the zone-based structure of a transport-support workflow can be represented by a topologically-sorted Directed Acyclic Graph (DAG).

We analyze the computational complexity of TSWOP in two cases: i) when modules are free of cost, i.e., $\mathcal{C}_{max} = \infty$, and ii) when modules incur certain costs, i.e., $\mathcal{C}_{max} < \infty$.

If services are free, i.e., $\mathcal{C}_{max} = \infty$, TSWOP is polynomially solvable by searching for a longest profit path in the given DAG-structured workflow. In Algorithm 1, we design a Critical Path-based Module Selection (CPMS) algorithm for TSWOP under this condition.

If services are not free, i.e., $\mathcal{C}_{max} < \infty$, TSWOP becomes weakly NP-complete and can be solved in pseudo-polynomial time. We first prove the NP-completeness of TSWOP by reducing from the Multiple-Choice Knapsack Problem (MCKP) [43] and then design a Dynamic Programming-based Module Selection (DPMS) algorithm as shown in Algorithm 2.

Theorem 1. *The Transport-Support Workflow Optimization Problem is NP-complete.*

Proof. Without loss of generality, we consider the profit and cost values of each module to be integers within independent ranges.

We define a decision version of TSWOP by introducing an integer bound I on the sum of profits: is there a module selection S with a total profit larger than I and a total cost under $\mathcal{C}_{max}$?

Consider a solution S to TSWOP, which is a subset of modules of the given workflow. It takes polynomial time ($O(K)$, where K is the number zones in the workflow) to determine whether the subset of modules has a sum of profits larger than I and has a sum of cost less than $\mathcal{C}_{max}$, as we only need to traverse each module and add up their profit values and cost values, respectively. Hence, TSWOP is NP.

We reduce from the well-known NP-complete Multiple-Choice Knapsack Problem (MCKP) [57], which is defined as: Given K' mutually disjoint classes of items to be packed into a knapsack of capacity $\mathcal{C}'_{max}$, where class $j\,(1 \leq j \leq K')$ contains N'_j items and the ith item in the jth class, denoted as $m'_{i,j}$, has a value $\mathcal{P}'_{i,j}$ and a weight $\mathcal{C}'_{i,j}$, we want to choose exactly one item from each class such that the total profit is maximized without exceeding the capacity $\mathcal{C}'_{max}$. If we use a binary variable $x'_{i,j}$ to denote whether or not item $m'_{i,j}$ is selected from class N'_j, the objective

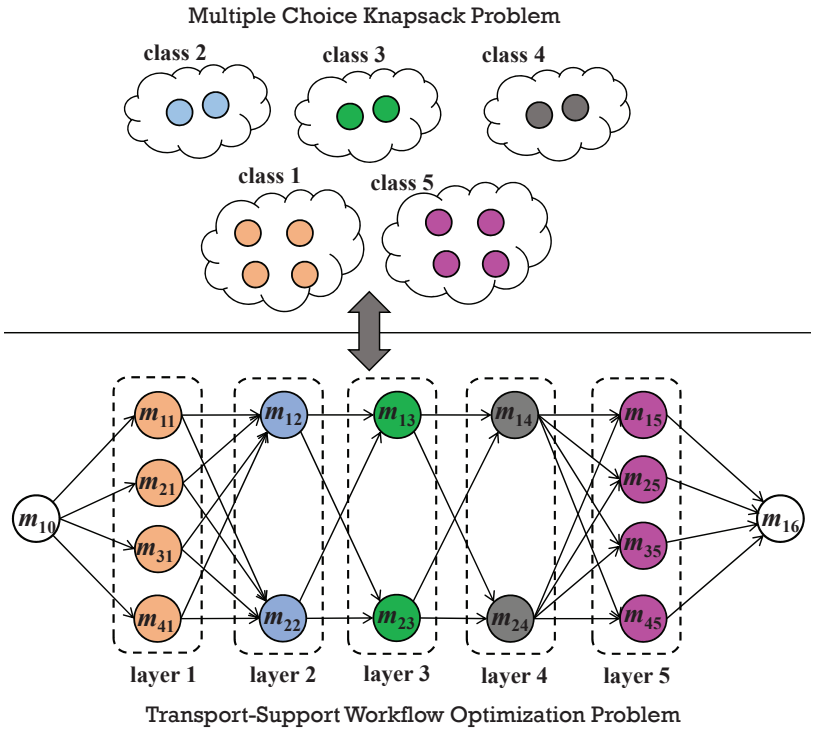


Figure 2.9 NP-completeness proof of TSWOP.

function of MCKP is given as

$$\max \sum_{j=1}^{K'} \sum_{i=1}^{N'_j} \mathcal{P}'_{i,j} \cdot x'_{i,j}, \tag{2.9}$$

subject to

$$\sum_{i=1}^{K'} \sum_{i=1}^{N'_j} \mathcal{C}'_{i,j} \cdot x'_{i,j} \leq \mathcal{C}'_{max}, \tag{2.10}$$

$$\sum_{i=1}^{N'_j} x'_{i,j} = 1, 1 \leq j \leq K', \tag{2.11}$$

$$x'_{i,j} \in \{0,1\}, 1 \leq i \leq N'_j, 1 \leq j \leq K'. \tag{2.12}$$

Given an instance of the MCKP problem, we construct an instance of a special case of TSWOP where adjacent layers (zones) are fully connected. As shown in

Figure 2.9, we take the following steps to construct an instance of TSWOP: i) we treat the capacity $\mathcal{C}'_{max}$ in MCKP as the cost constraint $\mathcal{C}_{max}$ in TSWOP, and then without loss of generality, construct each of the disjoint classes in MCKP as a layer (zone) in TSWOP, in which the items are treated as the modules in the zone with its value $\mathcal{P}'_{i,j}$ as the profit $\mathcal{P}_{i,j}$ and its weight $\mathcal{C}'_{i,j}$ as the cost $\mathcal{C}_{i,j}$, ii) we fully connect the modules in adjacent layers (zones), and iii) we add a virtual start module and a virtual end module, as shown in Figure 2.9. Since the adjacent zones are fully-connected, the module selection in a specific zone does not prevent any modules in its adjacent zones from being selected. Therefore, if we find an optimal module selection across each layer in TSWOP under cost constraint $\mathcal{C}_{max}$, then this selection would result in an optimal solution to the instance of MCKP under capacity constraint $\mathcal{C}'_{max}$, and vice versa. Note that the instances of TSWOP constructed from the instances of MCKP have fully-connected adjacent zones, and hence are a subset of general TSWOP problem instances with arbitrary edges between adjacent zones.

Since MCKP is well-known to be NP-complete [43], so is TSWOP. Proof ends.

□

2.6.7 Algorithm Design

We design two algorithms for TSWOP under constraints $\mathcal{C}_{max} = \infty$ and $\mathcal{C}_{max} < \infty$, respectively.

Without Cost Constraint Since some advanced services such as OSCARS [7] in ESnet [6] and ION [14] in Internet2 [13] are currently free to authorized users, from a practical point of view, it is worthy to explicitly study the complexity of TSWOP when $\mathcal{C}_{max} = \infty$. Given the profit vector p of each module, if we do not consider the cost constraint in Equation 2.5 or consider it as unlimited, the problem could be described as a longest profit path problem in a topologically-sorted DAG, which is polynomially solvable. We design the Critical Path-based Module

Selection (CPMS) algorithm that employs a dynamic programming approach to solve the transport-support workflow optimization problem under $\mathcal{C}_{max} = \infty$ constraint, as shown Algorithm 1. As previously mentioned, without loss of generality, we add a virtual start module $m_{1,0}$ (i.e., the 1st and the only module at zone 0) and a virtual end module $m_{1,K+1}$ (i.e., the 1st and the only module at zone $K+1$) with profit and cost of each module are both zero, and then the problem is equivalent to find the longest profit path from module $m_{1,0}$ to module $m_{1,K+1}$. Detailed CPMS algorithm is given in Algorithm 1, which runs in $O(|E|)$ time, where $|E|$ is the total number of edges (dependencies) between the modules in adjacent zones.

Algorithm 1 Critical Path-based Module Selection (CPMS)

Input: A DAG-structured transport-support workflow $G(M, E, K, N_j, \{\mathcal{P}_{i,j}\}, \{\mathcal{C}_{i,j}\})$.

Output: A longest profit path formed by the modules selected from G.

1: Add virtual modules $m_{1,0}$ and $m_{1,K+1}$ and let $\mathcal{P}_{1,0} = \mathcal{P}_{1,K+1} = \mathcal{C}_{1,0} = \mathcal{C}_{1,K+1} = 0$;
2: Let $\mathcal{D}(i,j)$ be the maximal achievable profit on the path from $m_{1,0}$ to $m_{i,j}$;
3: Let $\mathcal{L}(i,j)$ be the longest path corresponds to $\mathcal{D}(i,j)$;
4: $\mathcal{D}(1,0) = 0$, $\mathcal{L}(1,0) = \{m_{1,0}\}$;
5: **for** $j = 1$ **to** $K+1$ **do**
6: **for** $i = 1$ **to** N_j **do**
7: $\mathcal{D}(i,j) = \max\left\{\mathcal{D}(i',j-1)\middle| e_{(i',j-1),(i,j)} = 1\right\} + \mathcal{P}_{i,j}$;
8: $\mathcal{L}(i,j) = \max\left\{\mathcal{L}(i',j-1)\middle| e_{(i',j-1),(i,j)} = 1\right\} \cup \{m_{i,j}\}$;
9: **return** $\mathcal{D}(1,K+1)$ and $\mathcal{L}(1,K+1)$;

With Cost Constraint We design the Dynamic Programming-based Module Selection (DPMS) algorithm shown in Algorithm 2 for TSWOP when $\mathcal{C}_{max} < \infty$, which also employs a dynamic programming-based approach.

A unique property of TSWOP's given is that the DAG-structured workflow is already topologically-sorted, and moreover, there only exist dependencies between

Algorithm 2 Dynamic Programming-based Module Selection (DPMS)

Input: A DAG-structured transport-support workflow $G(M, E, K, N_j, \{\mathcal{P}_{i,j}\}, \{\mathcal{C}_{i,j}\})$ and a cost constraint $\mathcal{C}_{max} < \infty$.

Output: A longest profit path formed by the modules selected from G with their total cost under $\mathcal{C}_{max}$.

Add virtual modules $m_{1,0}$ and $m_{1,K+1}$ and let $\mathcal{P}_{1,0} = \mathcal{P}_{1,K+1} = \mathcal{C}_{1,0} = \mathcal{C}_{1,K+1} = 0$;
Let $\mathcal{D}(\mathcal{C}, i, j)$ be the maximal achievable profit under cost constraint $\mathcal{C}$ using modules from zone 0 to zone j and module $m_{i,j}$ is selected;
Let $\mathcal{L}(\mathcal{C}, i, j)$ be the longest path corresponds to $\mathcal{D}(\mathcal{C}, i, j)$;
for $1 \leq i \leq N_0$ **do**
  for $0 \leq \mathcal{C} \leq \mathcal{C}_{max}$ **do**
    $\mathcal{D}(\mathcal{C}, i, 0) = 0$;
    $\mathcal{L}(\mathcal{C}, i, 0) = \{m_{i,0}\}$;
for $1 \leq j \leq K+1$ **do**
  for $1 \leq i \leq N_j$ **do**
    for $0 \leq \mathcal{C} \leq \mathcal{C}_{\max}$ **do**
      if $\mathcal{C}_{i,j} > \mathcal{C}$ **then**
        $\mathcal{D}(\mathcal{C}, i, j) = -1$;
      else if $\left|\left\{m_{i',j} \middle| e_{(i',j-1),(i,j)} = 1 \;\&\&\; \mathcal{D}(\mathcal{C} - \mathcal{C}_{i,j}, i', j-1) \geq 0\right\}\right| \neq \emptyset$ **then**
        $\mathcal{D}(\mathcal{C}, i, j) = \max\left\{\mathcal{D}(\mathcal{C} - \mathcal{C}_{i,j}, i', j-1) \;\middle|\; e_{(i',j-1),(i,j)} = 1 \;\&\&\; \mathcal{D}(\mathcal{C} - \mathcal{C}_{i,j}, i', j-1) \geq 0\right\} + \mathcal{P}_{i,j}$;
        $\mathcal{L}(\mathcal{C}, i, j) = \max\left\{\mathcal{L}(\mathcal{C} - \mathcal{C}_{i,j}, i', j-1) \;\middle|\; e_{(i',j-1),(i,j)} = 1 \;\&\&\; \mathcal{D}(\mathcal{C} - \mathcal{C}_{i,j}, i', j-1) \geq 0\right\} \cup \{m_{i,j}\}$;
      else
        $\mathcal{D}(\mathcal{C}, i, j) = -1$;
return $\mathcal{D}(\mathcal{C}_{max}, 1, K+1)$ and $\mathcal{L}(\mathcal{C}_{max}, 1, K+1)$;

adjacent zones, which allows us to design our algorithm using a layer-based approach without considering more complicated connectivity between non-adjacent zones. Since one and only one module must be selected from any zone, i.e., the path that maximizes the profit under the cost constraint must go through a module in each layer, if we let $\mathcal{D}(\mathcal{C}, i, j)$ be the maximal achievable profit using modules selected from zone 0 to zone j when module $m_{i,j}$ is known to be selected at zone j, then the desired final solution would be $\mathcal{D}(\mathcal{C}_{max}, 1, K+1)$. At any given layer j $(1 \leq j \leq K)$, since one and only one module in layer j must be on the path, there are at most N_j possibilities for the path with maximal profit under cost constraint. Suppose that the "optimal" path exists under cost constraint $\mathcal{C}_{max}$ and module $m_{i,j}$ is on this "optimal" path, the path with maximal profit using modules from layer 0 to layer j with a certain cost constraint $\mathcal{C}$ must go through one of $m_{i,j}$'s preceeding modules $m_{i',j-1}$ at layer $j-1$. Moreover, since $m_{i,j}$ takes a cost of $\mathcal{C}_{i,j}$, module $m_{i',j-1}$ must be on the longest path from the modules in layer 0 to the modules in layer $j-1$ that have succeeding module $m_{i,j}$ with a cost under $\mathcal{C}_{max} - \mathcal{C}_{i,j}$, i.e., we have the optimal sub-structure of TSWOP as

$$\mathcal{D}(\mathcal{C}, i, j) = \mathcal{P}_{i,j} + \max \left\{ \mathcal{D}(\mathcal{C} - \mathcal{C}_{i,j}, i', j-1) \;\middle|\; \begin{array}{l} e_{(i',j-1),(i,j)} = 1, \\ \\ \mathcal{D}(\mathcal{C} - \mathcal{C}_{i,j}, i', j-1) \geq 0 \end{array} \right\}. \tag{2.13}$$

Based on Equation 2.13, we get the optimal result as $\mathcal{D}(\mathcal{C}_{max}, 1, K+1)$ through filling a sparse 3-dimensional matrix. In Algorithm 2, we also first add a virtual start module and a virtual end module and make them with zero profit and zero cost (Line 1), and set up the basics from Line 4 to Line 7, where $\mathcal{D}(\mathcal{C}, i, 0) = 0$ indicates that module $m_{i,0}$ (i.e., the only module in layer 0) can be (and should be) on the

"optimal" path formed by the selected modules. Next, we traverse each module in every layer (from Line 8 to Line 9), and for each module, we calculate $\mathcal{D}(\mathcal{C}, i, j)$ for each integer value of the parameter $\mathcal{C}$ $(0 \leq \mathcal{C} \leq \mathcal{C}_{max})$. If the cost of the current module $m_{i,j}$ is less than the current cost constraint $\mathcal{C}$, there are not any possible paths exist and we set $\mathcal{D}(\mathcal{C}, i, 0) = -1$ to indicate such cases (from Line 11 to Line 12). Otherwise, we check module $m_{i,j}$'s preceding modules that has feasible paths traverse through (Line 13), select the one that has maximal achievable profit, and then based on which calculate the value of $\mathcal{D}(\mathcal{C}, i, j)$ for the module being considered (Line 14) and keep track of the path in $\mathcal{L}(\mathcal{C}, i, j)$ (Line 15). If none of $m_{i,j}$'s preceding modules contain feasible paths, $m_{i,j}$'s $\mathcal{D}(\mathcal{C}, i, j)$ value is also –1 (Line 17). After traversing each module in each layer and computing its corresponding value of $\mathcal{D}(\mathcal{C}, i, j)$, the desired final result is returned as $\mathcal{D}(\mathcal{C}_{max}, 1, K + 1)$ (Line 18) and the corresponding module selection (i.e., the longest path under cost constraint) can be interpreted from $\mathcal{L}(\mathcal{C}_{max}, 1, K+1)$. The DPMS algorithm runs in pseudo-polynomial time $O(|E| \cdot \mathcal{C}_{max})$, where $|E|$ the number of edges (dependencies) in the given workflow.

2.7 Simulation-based Performance Evaluation

We conduct simulation-based performance evaluations to show the effectiveness and efficiency of proposed CPMS and DPMS algorithms.

2.7.1 Simulation Setup and Performance Criterions

The test datasets are generated using the following strategies. The problem size is determined by a 3-tuple $\langle |M|, K, d \rangle$, where $|M|$ is the number of modules, $|K|$ is the number of zones, and d is the "density" of the edges that is defined as the ratio of the average edge number of each module over the number of modules in its succeeding zone. The density d in turn determines the total number of edges $|E|$ in the workflow.

We first vary the number of modules $|M|$ from 1,000 up to 10,000, and randomly pick up the number of modules in each layer from the range $\left[1, \frac{|M|}{K} \cdot \xi\right]$, where $\xi = 1.5$ decides the variance of the number of modules in each zone. We then vary the number of zones from 10 to 100, where small values (e.g., 10–30) represent the scale of domestic data movement, medium values (e.g., 40–70) represent the scale of inter-continental data movement, and large values (e.g., 80–100) represent the scale of global data movement. The edge density $d\,(0.0 < d \leq 1.0)$ represents the number of edges on each module that is decided as follows. For module $m_{i,j}$ in zone j, we randomly select $d \cdot N_{j+1}$ modules from zone $j+1$ as $m_{i,j}$'s succeeding modules and to which add directed edges from $m_{i,j}$. We check for each module to make sure that there is at least one incoming edge, and if there is not, we randomly choose one from its preceding adjacent zone. We choose the value of d from three categories: i) fully connected adjacent zones, where $d = 1.00$; ii) moderately connected adjacent zones, where $d \in \{0.75, 0.50\}$; and iii) sparsely connected adjacent zones, where $d = 0.25$.

Given the value of profit within the range of $0 \leq p_{i,j,k} \leq 1$ and the value of cost as an arbitrary positive number, we could always scale them into a range of positive integers, in our simulations, the profit and cost of each module are both randomly picked up from a pre-defined range $[1, 100]$.

We use a naive greedy approach named Ratio-based Greedy Module Selection (RGMS) algorithm as the comparison base, in which each zone is went through and the available module with the best (largest) ratio of $\frac{\mathcal{P}_{i,j}}{\mathcal{C}_{i,j}}$ is selected. In each simulation test, we measure the following criterions: i) the profit of the path calculated by CPMS, which is the best possible path of the workflow since CPMS does not consider any cost constraint; ii) the profit of the path calculated by RGMS, which is the most *cost-efficient* path; and iii) the profit of the path calculated by DPMS under the cost constraint $\mathcal{C}_{max}$, which is the best path as defined in TSWOP.

2.7.2 Simulation Results

The total cost of the path resulted from RGMS (denoted as $\mathcal{C}_{RGMS}$) is used in DPMS as the cost constraint. In such a way, we could measure the improvement of DPMS over RGMS under the same condition. We first run RGMS on a randomly generated workflow and calculate the resulted cost $\mathcal{C}_{RGMS}$, and then use $\mathcal{C}_{RGMS}$ in DPMS, i.e., $\mathcal{C}_{max} = \mathcal{C}_{RGMS}$, on the same workflow. We compare the profit achieved by RGMS, DPMS, and CPMS, and plot representative results in Figure 2.10, where $K = 100$ and $d \in \{0.25, 0.50, 0.75, 1.00\}$. As $|M|$ increases, the profit achieved by DPMS is consistently and significantly higher than RGMS given the same values of K and $\mathcal{C}_{max}$. Note that we also plot the profit achieved by CPMS where $\mathcal{C}_{max} = \infty$ to show the upper bound of the maximal achievable profit.

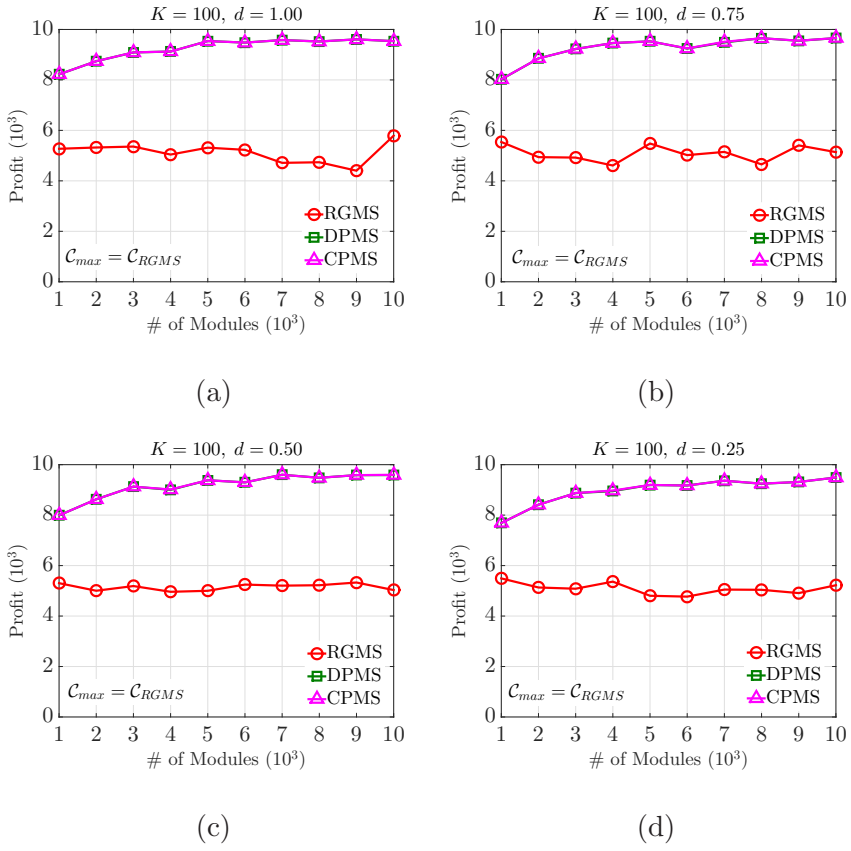


Figure 2.10 Comparison between RGMS, DPMS, and CPMS corresponds to the number of modules ($\mathcal{C}_{max} = \mathcal{C}_{RGMS}$).

We also compare in Figure 2.11 the profits achieved by RGMS, DPMS, and CPMS correspond to K, where $|M| = 10,000$ and $d \in \{0.25, 0.50, 0.75, 1.00\}$. Since each module's profit is randomly generated within the range $[1, 100]$, the actual value of profit increases as K increase and DPMS achieves much higher profit than RGMS. In addition, the improvement becomes larger as K increases.

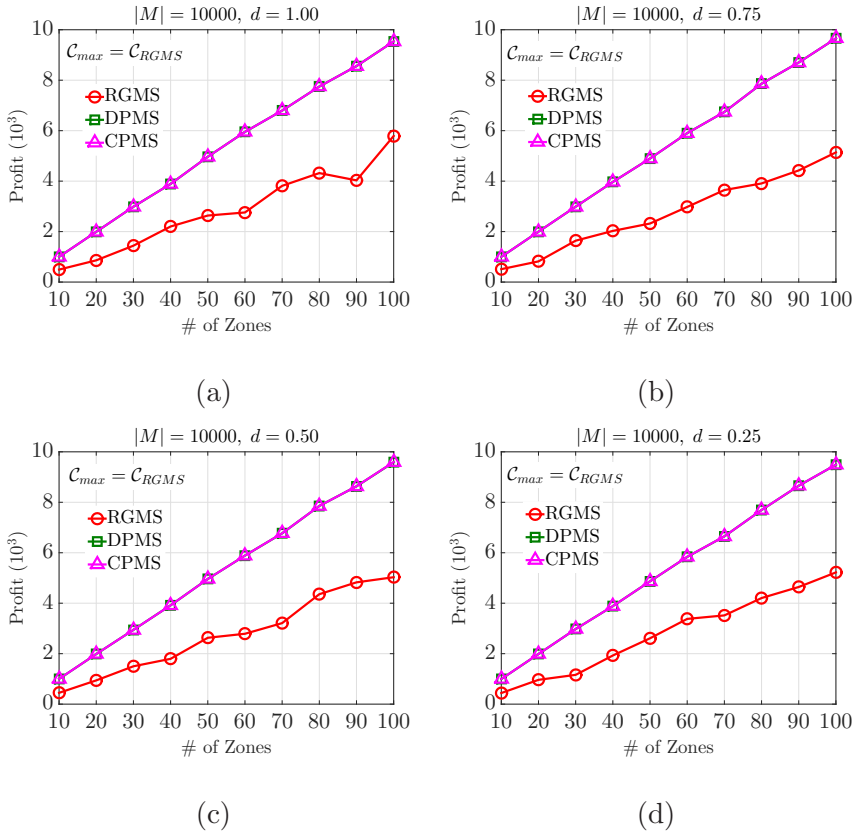


Figure 2.11 Comparison between RGMS, DPMS, and CPMS corresponds to the number of zones ($\mathcal{C}_{max} = \mathcal{C}_{RGMS}$).

Figures 2.10 and 2.11 show the superiority of DPMS over naive greedy RGMS. We plot in Figure 2.12 more complete results to show the performance improvement of DPMS over RGMS, where $|M|$ is from 1,000 to 10,000 with an interval of 1,000 and K is from 100 to 1,000 with an interval of 10. We also pick up value of d from $\{0.25, 0.50, 0.75, 1.00\}$. Figure 2.12 shows that the improvement of DPMS over RGMS

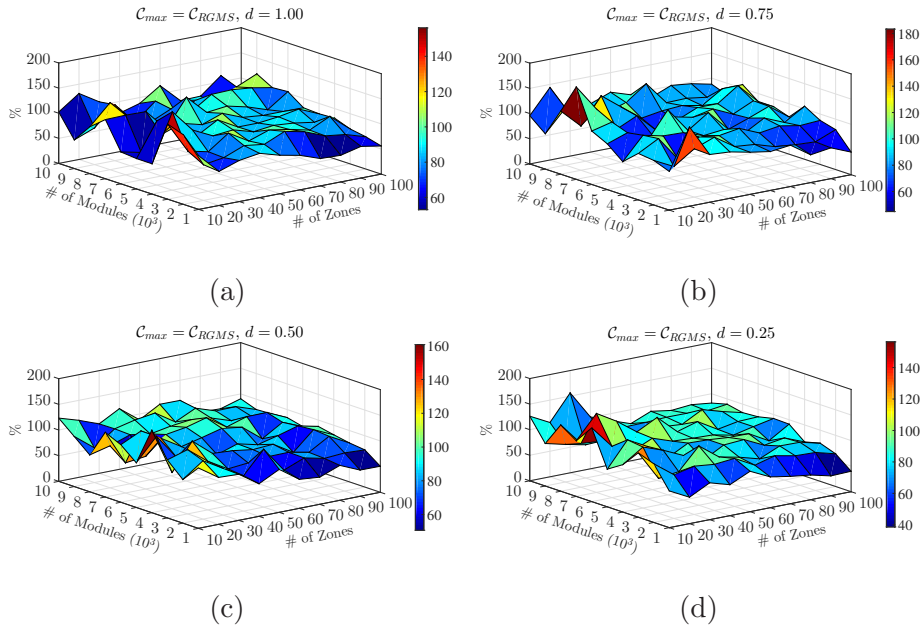


Figure 2.12 Improvement of DPMS over RGMS ($\mathcal{C}_{max} = \mathcal{C}_{RGMS}$).

is consistently higher than 40% up to more than 180% under the same conditions with the same cost constraints.

We next use an even smaller value of cost constraint $\mathcal{C}_{max}$ to test DPMS since the cost consumed by a greedy-based RGMS might be too large to show the significant superiority of DPMS over others. Given a workflow G, we first calculate its longest cost path $\mathcal{C}_{G_{max}}$ and shortest cost path $\mathcal{C}_{G_{min}}$, respectively, i.e., the longest and shortest cost paths calculated by considering the cost $\mathcal{C}_{i,j}$ of each module rather than the profit $\mathcal{P}_{i,j}$ of each module as the criterion. We then set the cost constraint using Equation 2.14, i.e., we set the cost constraint as a quarter of the maximal cost of all possible paths in the workflow if it is feasible, otherwise we simply use the minimal cost of all possible paths,

$$\mathcal{C}_{max} = \max\{\mathcal{C}_{G_{min}}, \frac{1}{4} \cdot \mathcal{C}_{G_{max}}\}. \tag{2.14}$$

We also test and compare DPMS, RGMS, CPMS to show the performance superiority of DPMS over the others under such a smaller cost constraint. Note that

in these simulations, we do not set the cost constraint for RGMS (i.e., $\mathcal{C}_{RGMS} = \infty$) to ensure the randomly generated test cases to be feasible, which provides RGMS with an extremely relaxed cost constraint comparing with the cost constraint for DPMS.

We set the number of zones as 100 and pick up the value of edge density d from $\{0.25, 0.50, 0.75, 1.00\}$, as shown in Figure 2.13, with a smaller cost constraint, DPMS achieves slightly smaller profit than CPMS since CPMS is the best possible one without constraint. DPMS achieves much higher profit than RGMS even RGMS is without any cost constraint. As the number of modules increases from 1,000 to 10,000, the differences between DPMS and CPMS become smaller and the results are quite similar across different values of the edge density d.

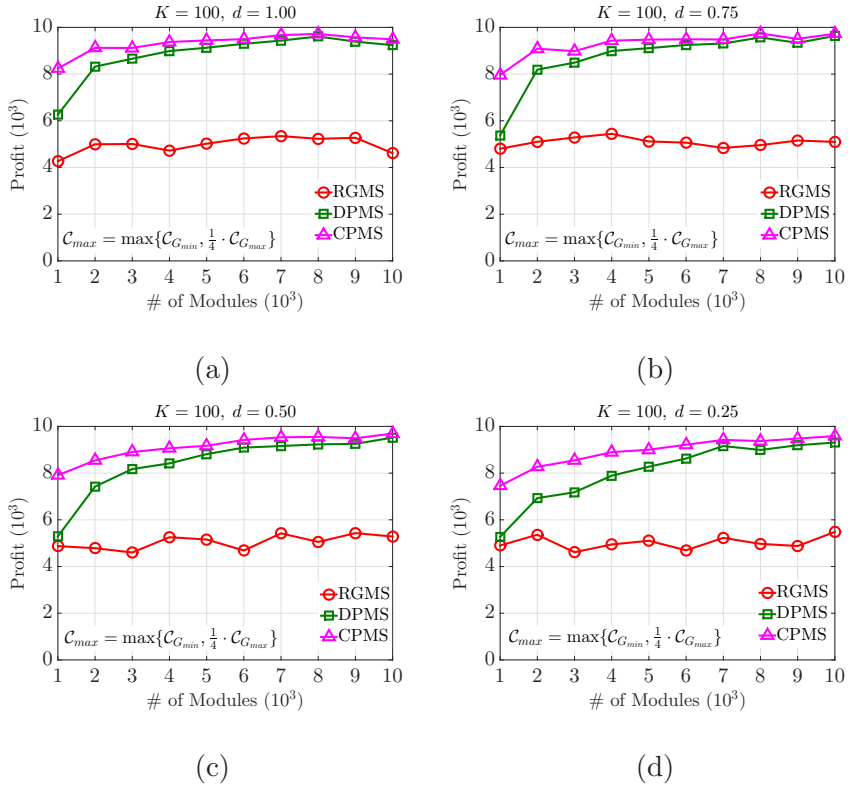


Figure 2.13 Comparison between RGMS, DPMS, and CPMS corresponds to the number of modules ($\mathcal{C}_{max} = \max\{\mathcal{C}_{G_{min}}, \frac{1}{4} \cdot \mathcal{C}_{G_{max}}\}$).

In Figure 2.14, we also compare the profits achieved by RGMS, DPMS, and CPMS under a smaller cost constraint corresponds to the number of zones, where we set the number of modules as 10,000 and the value of d is also selected from $\{0.25, 0.50, 0.75, 1.00\}$. The results are similar to that in Figure 2.11, i.e., i) the actual value of profit increases as number of zones increase; ii) DPMS with a smaller cost constraint achieves much higher profit than RGMS without cost constraint; and iii) the improvement also becomes larger as the number of zones increases.

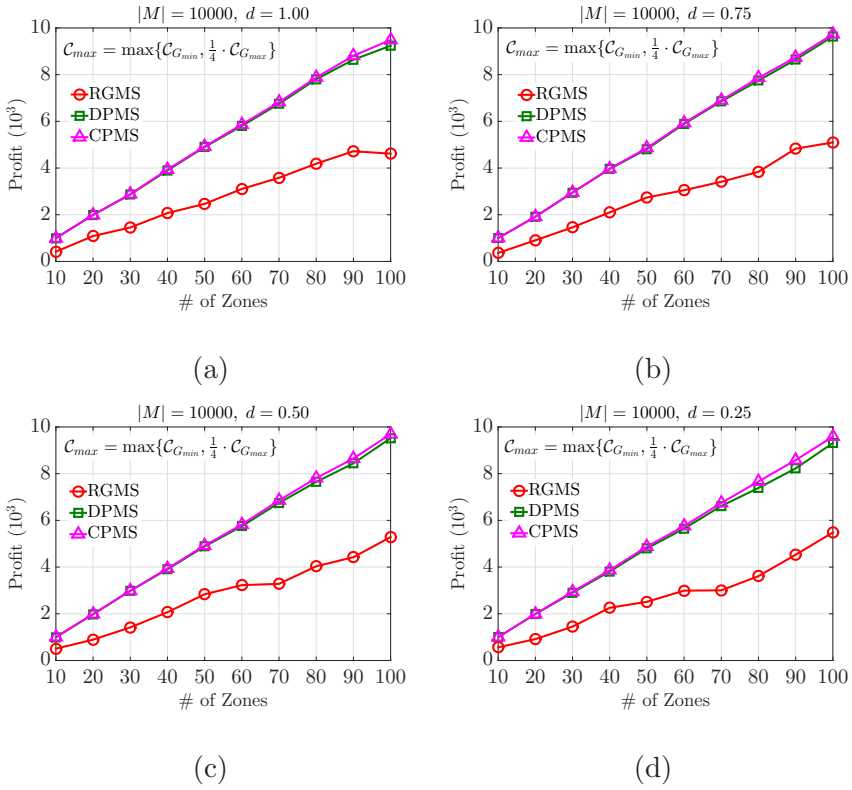


Figure 2.14 Comparison between RGMS, DPMS, and CPMS corresponds to the number of zones ($\mathcal{C}_{max} = \max\{\mathcal{C}_{G_{min}}, \frac{1}{4} \cdot \mathcal{C}_{G_{max}}\}$).

The complete results on the performance improvement of DPMS over RGMS are plotted in Figure 2.15, where the number of modules is from 1,000 to 10,000 with an interval of 1,000 and the number of zones is from 100 to 1,000 with an interval of 10. We also pick up the value of d from $\{0.25, 0.50, 0.75, 1.00\}$. Figure 2.15 shows

that the improvement of DPMS with a smaller cost constraint over RGMS without cost constraint is consistently higher than 20% up to 180%.

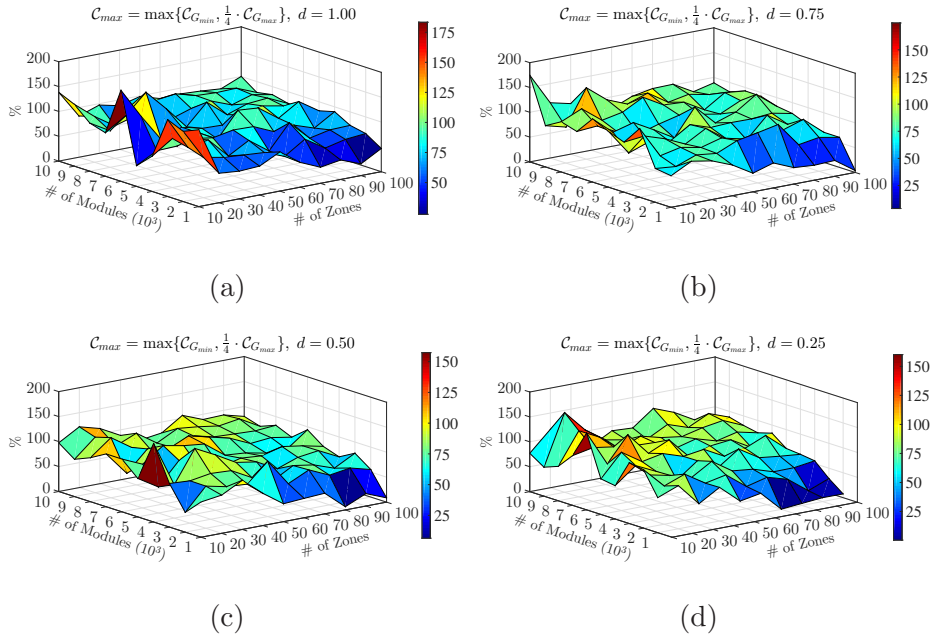


Figure 2.15 Improvement of DPMS over RGMS ($\mathcal{C}_{max} = \max\{\mathcal{C}_{G_{min}}, \frac{1}{4} \cdot \mathcal{C}_{G_{max}}\}$).

2.7.3 How Weak is the NP-completeness of TSWOP?

The MCKP problem has been widely studied in the literature [57] and quite a few heuristics could possibly be adopted for solving TSWOP. The DPMS algorithm is optimal, but runs in pseudo-polynomial time. From a practical point of view, we are more concerned with the optimal module selection rather than the running time of the computation. If the running time of DPMS is tolerable for the use of the solution in reality, by no means we should use heuristics without performance guarantees. We implement the DPMS algorithm in C/C++ and measure the execution time of DPMS algorithm on a RedHat Linux Workstation equipped with Intel(R) Xeon(R) CPU E5-2620 v3 CPU of 2.40 GHz and 16.00 GB memory. Figure 2.16 shows the average execution time of the DPMS algorithm. When the number of modules is

up to 10,000, the number of zones is up to 100, the cost constraint is up to the same level of $\mathcal{C}_{RGMS}$, and the modules in adjacent zones are fully-connected (i.e., the edge density $d = 1.00$), the DPMS algorithm on average finishes less than one minute. As the value of d decreases (i.e., the edges of the workflow becomes more sparse), the execution time of DPMS further decreases to be less than 20 seconds. The results in Figure 2.16 show that DPMS makes the "on-line" path composition feasible, considering its optimality, we are more in favor of DPMS rather than any other heuristics that take shorter time but without any performance guarantee.

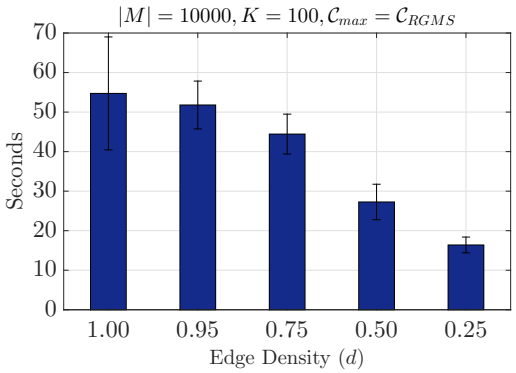


Figure 2.16 Running time of DPMS with 10,000 modules in 100 zones and a cost limit $\mathcal{C}_{max} = \mathcal{C}_{RGMS}$.

2.8 Experiment-based Case Study

In this section, we evaluate our workflow-based transport solution using real-life network experiments based on the services and resources discovered by NADMA [32].

2.8.1 Case Study 1: Data Transfer from Oak Ridge National Laboratory to Lawrence Berkeley National Laboratory

In this experiment, we consider a data transfer request from Oak Ridge National Laboratory (ORNL) to Lawrence Berkeley National Laboratory (LBNL) within the DOE network.

Table 2.6 User Request of Data Transfer from ORNL to LBNL

Notation	Value
Star time (t_s)	Not specified, starts as soon as possible
End time (t_e)	Not specified, finishes as soon as possible
Sender (h_s)	`dtn01.css.ornl.gov` (`128.219.168.100`)
Receiver (h_r)	`datagrid.lbl.gov` (`128.3.41.146`)
Cost ($\mathcal{C}_{max}$)	Ignored (considered as unlimited)

User Request In this experiment, the user requests to transfer data from the source host h_s at ORNL to the destination host h_r at LBNL. Since ORNL and LBNL are both connected to ESnet, OSCARS can be used to set up a dedicated channel in the backbone network free of charge (at least at present to authorized users). Moreover, since the networking devices and end host hardware and software systems have already been deployed, we assume that the financial cost at the end host and on the networking services and resources are negligible. We list the parameters of the user request in Table 2.6.

Table 2.7 Discovered Resources on End Hosts at ORNL and LBNL

Protocol	Port#	Description
FTP	21	File Transfer Protocol
SCP	22	Secure Copy Protocol
TFTP	69	Trivial File Transfer Protocol
HTTP	80	HyperText Transfer Protocol
SFTP	115	SSH File Transfer Protocol
HTTPS	443	Secure HyperText Transfer Protocol
GridFTP	2281, 2811	GridFTP
bbFTP-std	5021	bbFTP-std
bbFTP	5022	bbFTP-ssh
SRM	8433,10080	Storage Resource Manager

Resource Discovery Based on the source and destination hosts, NADMA discovers the services and resources that are available to the user. At the end host, several transfer protocols are detected as listed in Table 2.7. The network segments between

ORNL and LBNL are visualized in Figure 2.17, where ESCPS is available in ORNL's edge network, and OSCARS is available in the ESnet backbone. Note that the default IP path is also available between these two hosts.

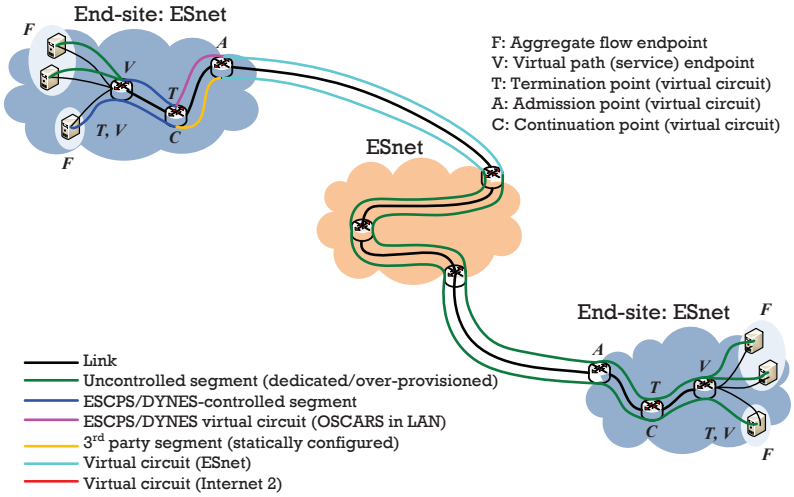


Figure 2.17 Network segments from ORNL to LBNL.

Source (A): ornl.gov	**Destination (F):** lbl.gov
Network: OSCARS	**Max. Backbone Capacity:** 5536 Mb/s
Available Bandwidth:	
A→B: 10000 Mb/s (10GE-Link)	
B→C: 8897 Mb/s	**C→D:** 5536 Mb/s
D→E: 9067 Mb/s	**E→F:** 10000 Mb/s (2*MAN 10G RING)
Path:	
A: ORNL (ornl.gov) → ORNL (site)	
B: (NASH) → NASH	**C: (STAR) → STAR**
D: (PNWG) → PNWG	**E: (SUNN) → SUNN**
F: LBNL (lbl.gov) → LBNL	

Figure 2.18 Details of the data transfer path from ORNL to LBNL.

Solution and Result In this experiment, we have 4 zones, namely, end host (both sender and receiver), edge network, and backbone network. In the backbone network, our method selects the dedicated channel with bandwidth guarantee within ESnet using OSCARS over the default IP path. At the end host, our method selects Globus

Toolkit over GridFTP. In the edge network, we choose ESCPS to set up the data transfer path. The resultant transport-support workflow we construct in this case is GridFTP → ESCPS → OSCARS → GridFTP. The reserved bandwidths and physical paths using the workflow selected by our method are shown in Figure 2.18.

2.8.2 Case Study 2: Data Transfer from Lawrence Livermore National Laboratory to University of Chicago

In this experiment, we consider a data transfer request from Lawrence Livermore National Laboratory (LLNL) to University of Chicago (UChicago).

User Request The source and destination hosts are located at LLNL, CA and University of Chicago, IL, respectively. Other requirements are the same with the Case Study 1 in Section 2.8.1.

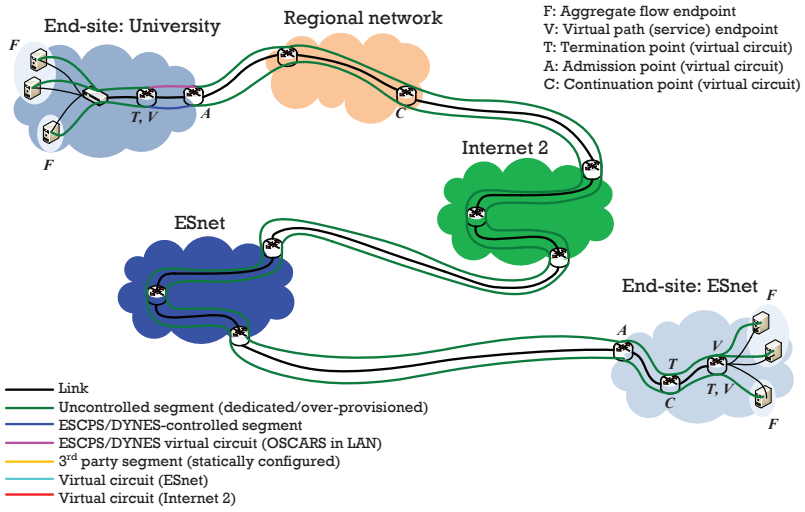


Figure 2.19 Network segments from LLNL to University of Chicago.

Resource Discovery Based on the locations of the hosts, the network segments between them are visualized in Figure 2.19. The available data transfer protocols are the same with those in Section 2.8.1, as listed in Table 2.7. LLNL is connected to the

ESnet with OSCARS service and UChicago is connected to the Internet2 with ION service. ESCPS and DYNES are available in the regional/edge network, respectively.

Solution and Result In this case, our model considers 6 zones, namely, two end hosts (both sender and receiver), two regional or edge networks, Internet2 backbone, and ESnet backbone. Our model selects corresponding services to establish dedicated paths in each network segment to achieve a reliable data transfer path with guaranteed bandwidth. Within each zone, NADMA discovers all available services from the database. To gain successful data transfer and better performance, our model only considers those modules the user has access to with needed credentials.

Module selection highly depends on the estimation of profit vector p, some of which are empirically obtained through historical log data, and a larger collection of the log/profile data could help refine the model. In this case, the resultant transport-support workflow we construct is GridFTP $\rightarrow$ ESCPS $\rightarrow$ OSCARS $\rightarrow$ ION $\rightarrow$ DYNES $\rightarrow$ GridFTP, as shown in Figure 2.20, which exhibits the best performance among all possible route combinations.

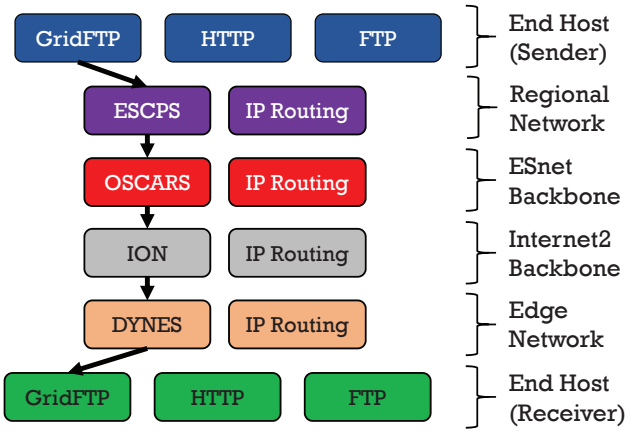


Figure 2.20 Transport-support workflow for the data transfer request from LLNL to University of Chicago.

CHAPTER 3

TRANSPORT PROFILE GENERATION

3.1 Introduction

High-performance Networks (HPNs) featuring high bandwidth and advance reservation as exemplified by ESnet [6], Internet2 [13], and Google's B4 [55] have emerged to be a promising solution to support large-scale data- and network-intensive applications. However, even if a dedicated channel is provisioned, the end-to-end data transfer performance still largely depends on the transport protocol being used at the end hosts. Along with the emergence and proliferation of HPNs, high-performance data transfer methods are being rapidly developed and deployed, but maximizing their application-level throughput over complex high-speed connections is still challenging: i) their optimal operational zone is affected by many factors including complex configurations and dynamics of network segments, end hosts, and protocol itself; ii) different parameter settings may lead to very different performances and oftentimes the default parameter setting does not yield the best performance; iii) application users, who are domain experts, typically do not have the necessary knowledge to choose which transport protocol to use and decide which parameter value to set; iv) due to the lack of accurate performance models for high-performance data transfer protocols such as UDT [48] and the complex dynamics of network environments, it is generally difficult to derive the optimal operational zone using an analytical approach. Consequently, application users have not seen the corresponding increase in transport performance at application level despite the bandwidth upgrades in the backbones of HPNs. Choosing an appropriate set of parameter values for a given data transfer protocol in many cases would result in a significant performance improvement over default settings. As a motivating example, we vary the block size of UDT [48]

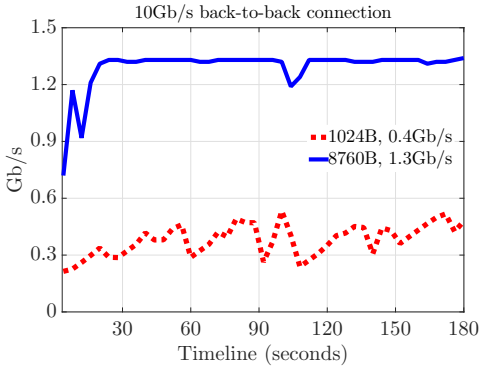


Figure 3.1 Instantaneous performance measurements of UDT over a 10 Gb/s back-to-back connection with different block sizes.

running over a local 10 Gb/s back-to-back connection, and plot the corresponding instantaneous throughput performance measurements in Figure 3.1, which shows more than three times improvement on average due to a simple change of block size. More performance improvements are expected if other parameters such as buffer size and block size are properly tuned as well.

Transport profiling, which sweeps through the combinations of parameter settings such as socket options, application-specific parameters, and protocol-specific configurations, enables users to determine the "best" set of parameter values for the optimal data transfer performance. Bandwidth estimation tools such as `iperf3` [8] could be utilized to conduct such transport profiling. `iperf3` uses continuous data transfers to estimate the performance along an end-to-end path and provides users with various functions and options for tuning TCP, UDP and SCTP, but it does not incorporates UDT [48], a widely adopted data transfer protocol in the HPN community, and does not provides an option to run parallel data streams over multiple NIC-to-NIC connections. A survey of bandwidth estimation tools can be found in [72].

We study the profiling approach to characterize and enhance the end-to-end performance of transport protocols in support of big data transfer over dedicated channels. We design and implement a Transport Profile Generator (TPG) toolkit to provide users with a light-weight and easy-to-use toolkit for conducting "one-time

profiling" to tune various control parameters for optimal performance. TPG supports profiling over multiple parallel data streams and multiple NIC-to-NIC connections. To instantiate the design of TPG, we use UDT protocol [48] as an example in the implementation and conduct extensive data transfer experiments over local- and wide-area network connections to illustrate how existing transport protocols benefit from TPG in optimizing their end-to-end performance.

We present extensive experimental results to show the properties of big data transfer over various high-speed network connections, including: i) a local 10 Gb/s back-to-back connection at University of Memphis (Section 3.4); ii) dual 10 Gb/s NIC-to-NIC connections at New Jersey Institute of Technology (Section 3.5); iii) 10 Gb/s emulated connections with various RTT delays ranging from 0 ms to 366 ms at Oak Ridge National Laboratory (Section 3.6); and iv) 10 Gb/s long-haul (380 ms) physical connection from Argonne National Laboratory to University of Chicago (Section 3.7). The extensive experimental results show that TPG-tuned UDT is able to outperform not only the default UDT but also TCP and its variants over high-speed long-haul dedicated connections with a certain delay.

3.2 Transport Profiling

3.2.1 Performance-related Components

The end-to-end data transfer is a complex process that involves both network and end-host components. Table 3.1 lists various software/hardware components together with their parameters that may affect the end-to-end transport performance in a typical data transfer process using protocols such as SABUL/UDT [47, 48]. Any of these components could become the bottleneck and hence limit the throughput performance, some of which can be accessed and controlled by the application, such

Table 3.1 Factors and Components Related to Data Transfer Performance

System	Parameters	Connection
CPU frequency	Packet size	RTT
Internal conn.	Payload size	Link bandwidth
IRQ balance	Block size	Path MTU
IRQ coalescence	Number of streams	Loss rate
CPU affinity	UDT send buffer size	...
Memory size	UDT recv buffer size	
Disk r/w speed	UDP send buffer size	
Bus speed	UDP recv buffer size	
NIC speed	Other protocol options	
Ring buffer size	...	
OS proc. sched.		
...		

as packet size, block size[1], buffer size, frame size, and number of streams; while others are mainly determined by hardware configurations and network infrastructures, such as CPU frequency, memory size, memory bandwidth, bus speed, disk I/O speed, path MTU size, round trip time, and connection bandwidth and loss rate.

3.2.2 Transport Profile

A transport profile $TP_t(\langle h_s, h_r \rangle, e, \theta)$ is a control-response plot illustrating how a set θ of control parameters affect the transport performance (mainly throughput/goodput) of a transport protocol t over a network connection e between a sender host h_s and a receiver host h_r. Such profiles indicate the qualitative behavior of each component involved in the data transfer process and provide useful information for maximizing the overall transport performance. The transport profile of a given protocol t is obtained by varying $\langle h_s, h_r \rangle$, e, and θ to exhaust the combination of parameter values over different network connections and collecting the corresponding *average throughput* measurements denoted by $G(\theta)$. We use a 2–tuple $e = \langle B, Q \rangle$ to represent

[1]In our design, TPG calls its send and receive functions to transfer a data block, which may in turn call the underlying transport protocol APIs several times to completely deliver an entire block. We use the term "block size" to denote the size specified in TPG's send and receive functions, and use "packet size" to denote the size of a transfer unit in the protocol.

a network connection e including the properties of its bandwidth (B, in Gb/s) and round trip delay (Q, in milliseconds), and use a generic 5–tuple $\theta = \langle m, l, f, p, d \rangle$ to represent the control parameter set including packet size (m, in bytes), block size (l, in bytes), buffer size (f, in megabytes), number of data streams (p, an integer), and data transfer time (d, in seconds). In a specific profiling where t, $\langle h_s, h_r \rangle$, and e are given, we vary the values of parameters in θ within certain ranges and collect the corresponding performance measurements to build a transport profile $TP_t(\langle h_s, h_r \rangle, e, \theta)$. While profiling, we calculate the average throughput performance $\overline{u}_i$ of data stream i during time interval $[0, \Delta T]$ as

$$\overline{u}_i(\theta) = \frac{\int_0^{\Delta T} S_i(x, \theta)\, dx}{\Delta T}, \tag{3.1}$$

where $S_i(x, \theta)$ is the sending rate of data stream i at time point x with parameter setting θ. The corresponding aggregate average throughput $G(\theta)$ is defined as

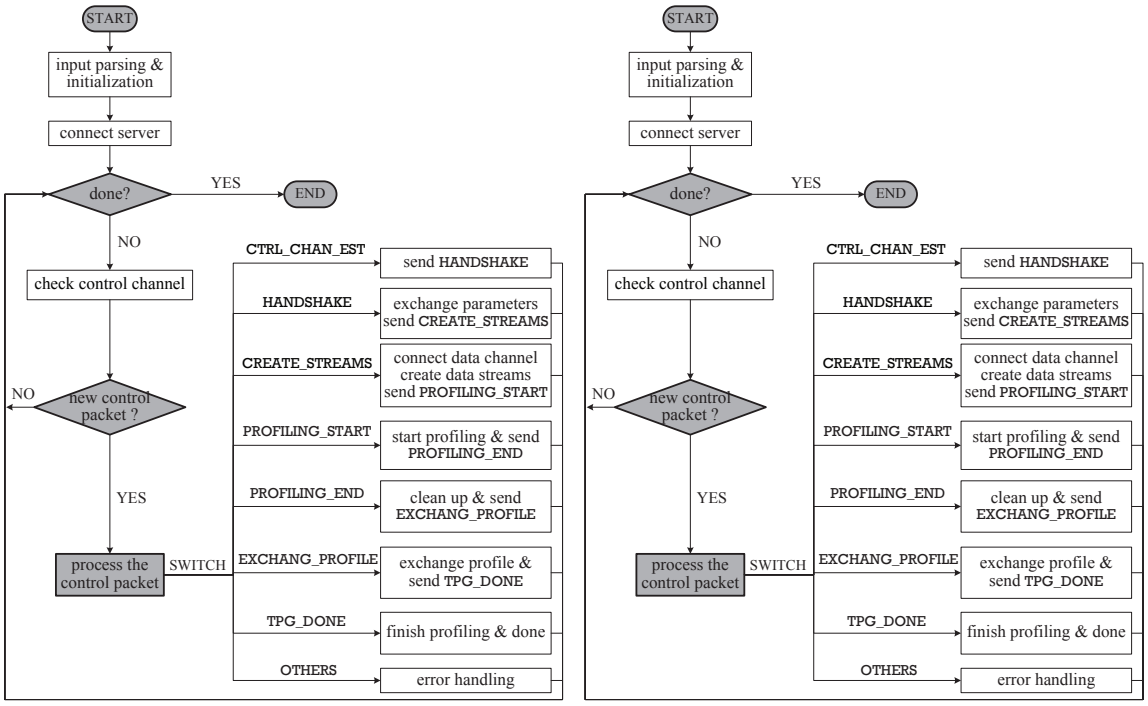
$$G(\theta) = \sum_{i=1}^{p} \overline{u}_i(\theta), \tag{3.2}$$

where p is the number of data streams. We calculate $G(\theta)$ in unit of Gb/s in this dissertation unless indicated otherwise.

3.3 Design and Implementation of Transport Profile Generator

3.3.1 Design Overview

Transport Profile Generator (TPG) consists of a pair of sender and receiver. The sender (client or source node) generates and delivers a certain amount of test data to the receiver (server or destination node) via a specific data transfer protocol being profiled. The sender also informs the receiver the initialization and termination of a data transfer process (one-time profiling) through an independent TCP-based control



(a) Client (b) Server

Figure 3.2 Control flow charts of Transport Profile Generator.

channel. The client drives the entire profiling process and terminates after a one-time profiling is completed, while the server is always reset for the next cycle of profiling. In such a way, user-specific profiling strategies (e.g., the stochastic approximation-based transport profiler `FastProf` as detailed in [91] and Chapter 4) can be automatically applied by repeatedly running the client with different parameter settings.

The flowcharts of TPG client and server are shown in Figure 3.2(a) and Figure 3.2(b), respectively. A typical TPG profiling carries out the following steps: 1) the server listens on the control channel; 2) the client parses the user input (if any), initializes, and then connects to the server through the control channel; 3) the server accepts the connection request, and then sends back an acknowledgement to the client; 4) the client and server exchange control parameters; 5) the server listens on protocol-specific data channels and then informs the client; 6) the client connects

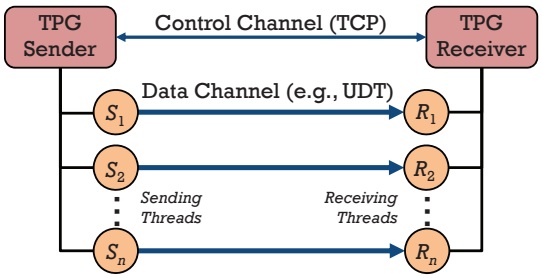


Figure 3.3 Control channel and data channel of TPG.

to the server on the data channel(s); 7) the server accepts the connection(s) on data channel(s); 8) the client and server start transferring data blocks; 9) the client and server exchange profiling results once the profiling is completed; 10) the client exits, and the server cleans up and waits for next profiling. During these steps, if an error or a failure occurs, the client or server sends an error message to the other through control channel before it exits or aborts.

3.3.2 Support of Multiple Data Streams and Multiple NIC-to-NIC Connections

In TPG, a TCP-based control channel is created for exchanging control message and a protocol-oriented data channel is created for the actual profiling. The main thread of TPG entity creates the control channel at initialization stage and then keeps polling it to see if there are newly arrived control packets. Most of the control packets include just one-byte data to inform the other end the state change. As shown in Figure 3.3, when multiple streams are specified by user, TPG creates an independent pair of sending and receiving threads to conduct the profiling task for each data stream.

We define several terms used in the transport profiling on multiple NIC-to-NIC connections: i) a *NIC-to-NIC connection* is identified by a source-and-destination IP pair; ii) a *UDT flow* is a logical channel between two UDP entities (IP and port) [46]; iii) a *UDT connection* is a distinct transfer entity between a pair of UDT sockets [46]; iv) a *TPG data stream* is defined based on a socket-oriented connection.

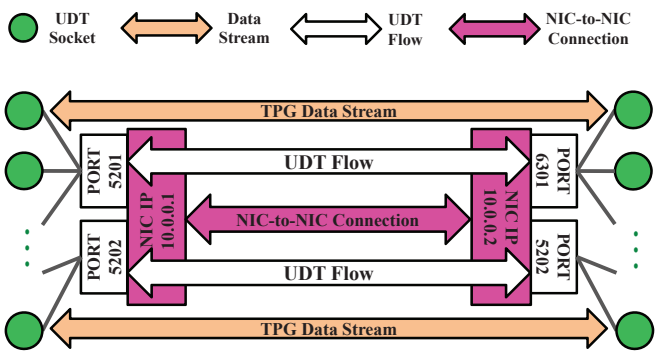


Figure 3.4 Multiple data streams and multiple NIC-to-NIC connections.

As shown in Figure 3.4, a TPG data stream is associated with a socket-oriented connection. As for the UDT case, it is created by assigning connection-related information of UDT to the TPG data stream after a UDT connection is established [46]. Meanwhile, TPG also maintains other information such as instantaneous performance measurements and statistics for each stream.

Multiple data streams may be created over one or multiple NIC-to-NIC connections based on the hardware configurations and user input specifications. In the multiple NIC-to-NIC connections case, a configuration file needs to be provided by user, and TPG includes a simple text parser to extract source and destination IP addresses from the configuration file. Multiple data streams may also be created over one or multiple UDT flows. A UDT flow is differentiated by a 5–tuple, i.e., ⟨source IP, source UDP port, destination IP, destination UDP port, congestion control algorithm⟩, and the UDT congestion control algorithm is applied to a distinct UDT flow, which is transparent to applications [46]. Since TPG data streams are created based on the socket-oriented connections, different TPG data streams may or may not share the same instance of UDT's DAIMD [50] congestion control algorithm. If they do not share the same 5–tuple mentioned above, different data streams stay in different UDT flows and each of them is controlled by an independent instance of UDT congestion control algorithm. Otherwise, the packets transferred in different

streams (i.e., UDT connections) are uniformly handled by the same instance of UDT congestion control algorithm and distinguished by UDT sockets.

TPG creates an independent working thread at both sender and receiver for each data stream without considering if the streams are over the same or different NIC-to-NIC connections, or the same or different UDT flows. Each working thread takes a stream as input and blindly sends/receives data blocks over the socket-oriented connection in its own independent space without interferences between each other. The sending thread keeps sending data blocks for a time duration or a fixed data size in a blocking mode, and then cleans up its stream and returns to the main thread. The main thread of the client waits for all sending threads to finish and then informs the server. The receiving thread at the server side works in a non-blocking manner and keeps checking if there are newly arrived data blocks in the data stream and then returns to the main thread of the server when: i) there are no more data blocks arriving and a timeout happens; ii) the final data block sent from the client is received; or iii) an interrupt signal is caught.

3.3.3 Support of Other Data Transfer Protocols

TPG features a flexible structure for an easy extension of other protocols, where a protocol is defined by its callback functions with a set of tunable control parameters. UDT and TCP are both supported in current version. For different protocols, TPG invokes the same procedure to control the profiling process as shown in Figures 3.2(a) and 3.2(b). To set up a data channel in TPG, the following callback functions need to be called accordingly: i) `tpg_init`, initializes a data channel; ii) `tpg_listen`, the server listens on the channel; iii) `tpg_connect`, the client connects to the server on the channel; iv) `tpg_accept`, the server accepts the connection request; v) `tpg_send`, the client sends data; vi) `tpg_recv`, the server receives data; and vii) `tpg_close`, both the server and the client close and clean up. TPG defines the prototype functions and

other related parameters in the structure `tpg_protocol`, which is loaded at runtime based on the user-specified transport protocol (e.g., `-t` option for UDT protocol, by default TCP is specified). To extend TPG with a protocol, one needs to: i) implement the corresponding protocol-specific callback functions; and ii) optionally, add an option parameter for the protocol. As for a specific profiling, the user can explicitly specify a protocol either with a command-line option or in a profiling function.

Table 3.2 Command Line Options of TPG and `FastProf`

Options	Comments
`-s`	Run as a server
`-c`	Run as a client
`-t`	Select UDT for profiling (default is TCP)
`-B`	Set maximal bandwidth a UDT connection can use
`-M`	Set UDT packet size
`-l`	Set data block size
`-w`	Set TCP socket buffer size
`-f`	Set UDT send buffer size
`-F`	Set UDP send buffer size
`-r`	Set UDT receive buffer size
`-R`	Set UDP receive buffer size
`-P`	Set number of parallel data streams
`-d`	Set profiling time duration
`-p`	Set port number for control channel
`-i`	Set time interval of performance report
`-b`	Bind server with an IP address (port)
`-j`	Enable interval performance report
`-q`	Enable server performance report
`-m`	Enable multiple NIC-to-NIC profiling
`-a`	Enable load balancing
`-x`	Trigger `FastProf` to do fast profiling
`-y`	Set bandwidth for `FastProf`
`-Q`	Set RTT delay
`-C`	Set performance gain ratio
`-L`	Set limit of consecutive iterations without improvement
`-N`	Set limit of total number of iterations

3.3.4 Implementation of TPG

TPG is implemented in C/C++ on Linux platform and is publicly available at [21]. The command-line options included in the current version is listed in Table 3.2.

3.4 Profiling Over a Local Back-to-back Connection

We present profiling results of UDT collected over a local 10 Gb/s back-to-back connection between two regular Linux workstations at University of Memphis. Please refer to Section 3.2.2 and Table 3.3 for parameter notations, some of which are used in the captions of the performance figures in the sections hereafter.

Table 3.3 Notations Used in the Design and Evaluation of TPG and `FastProf`

Notations	Definitions
h_s	Sender host
h_r	Receiver host
e	Network connection
θ	Control parameter set
B	Connection bandwidth
Q	Round Trip Time (RTT)
m	Packet size
l	Block size
l'	Iterative block size
f	Buffer size
f'	Iterative buffer size
f_{ts}	UDT send buffer size
f_{tr}	UDT receive buffer size
f_{ps}	UDP send buffer size
f_{pr}	UDP receive buffer size
p	Number of parallel streams
d	Data transfer duration
$\overline{u}_i$	Average throughput of stream i
G	Aggregate average throughput
$\mathcal{R}$	Limit of the number of consecutive iterations without performance improvement
$\mathcal{M}$	Limit of the total number of one-time profilings
$\mathcal{C}$	Performance gain ratio
$\mathcal{A}$	Bandwidth delay product rule

3.4.1 Testbed Configuration

We set up a local network testbed by back-to-back connecting two Dell workstations. The average round trip time (RTT) between these two hosts through the direct 10 Gb/s link is around 0.04 milliseconds, resulting in a bandwidth delay product (BDP) of 50 KB. The Internet connection between them, which is used for remote

control, has an RTT of 0.25 milliseconds and a bandwidth of 95 Mb/s, resulting in a BDP of around 3 KB. Both of the client (i.e., the sender, `dragon.cs.memphis.edu`) and the server (i.e., the receiver, `rabbit.cs.memphis.edu`) are equipped with a 2.93 GHz Intel Core(TM) 2 Duo E7500 CPU, 2.9 GB RAM, and Fedora 17 Linux Operating System updated with `3.9.10-100` kernel. The default (i.e., the values of `net.core.rmem_default` and `net.core.wmem_default`) and maximum (i.e., the values of `net.core.rmem_max` and `net.core.wmem_max`) memory space allowed for UDP socket buffer size is configured to be 32 MB and 64 MB, respectively.

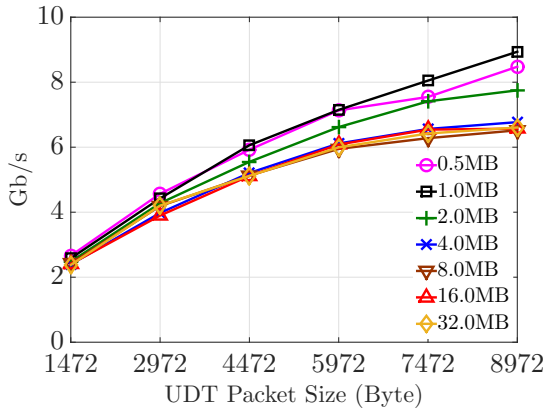
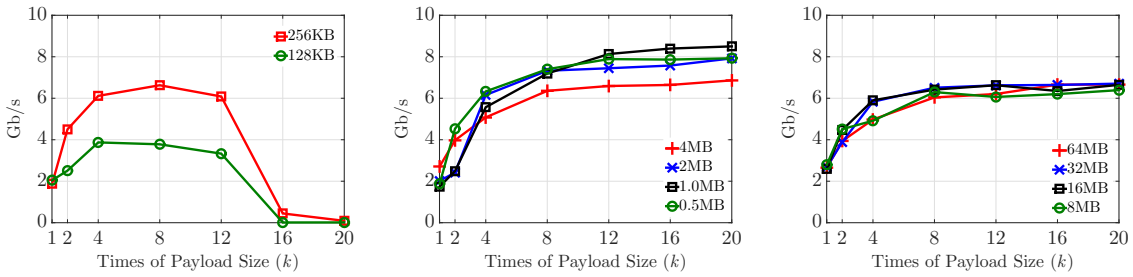


Figure 3.5 UDT profiling on packet size over a 10 Gb/s back-to-back connection. $B = 10$, $Q = 0.04$, $d = 120$, $l = 10 \cdot (m - 16) - 1$, $f \in \{0.5, 1, 2, 4, 8, 16, 32\}$ and $p = 1$.

3.4.2 UDT Profiling on Packet Size

Generally, the throughput can be improved by using a larger packet size to reduce per-packet overhead. Many high-speed ethernet NICs support "jumbo" frame with a packet size up to 9000 bytes and beyond. In the protocol stack of modern OS, the largest MTU supported along the network connection is automatically discovered and used [33, 69]. UDT provides a UDT socket option `UDT_MSS` to configure its packet size. The profiling results across different packet sizes are plotted in Figure 3.5, which shows that the UDT throughput performance is improved by using larger packet sizes and setting the UDT option `UDT_MSS` to be the maximal allowable MTU size along

the path significantly improves the end-to-end data transfer performance. Note that in this experiment, both UDT and UDP are configured with sufficient socket buffer space to maintain the link speed.



(a) $f \in \{0.125, 0.25\}$ (b) $f \in \{0.5, 1, 2, 4\}$ (c) $f \in \{8, 16, 32, 64\}$

Figure 3.6 UDT profiling on block size over a 10 Gb/s back-to-back connection. $B = 10$, $Q = 0.04$, $d = 120$, $m = 8,972$, $l = k \cdot (m - 16) - 1$, and $p = 1$.

3.4.3 UDT Profiling on Block Size

We plot the profiling results on the block size in Figure 3.6, where the x-axis uses a multiple (k) of the payload size[2] (i.e., $m - 16$) to represent the block size. We observe that when the buffer size is limited, increasing the block size does not improve throughput too much, especially when the block size is comparable with the buffer size; when there is sufficient buffer space, increasing the data block size significantly improves the UDT throughput performance, but the improvement becomes less obvious as the data block size increases.

Particularly, in Figure 3.6(a), when the buffer size is set to be 128 KB, which is larger than $B \cdot Q$, the peak throughput we observe is less 7 Gb/s over the 10 Gb/s link; when the block size is further increased from 107,471 bytes to 179,119 bytes (i.e., k from 12 to 20, since $\lceil \frac{107,471}{8,956} \rceil = 12$ and $\lceil \frac{179,119}{8,956} \rceil = 20$), the performance drastically decreases. If we increase the buffer size to 0.5 MB or 1.0 MB, then UDT

[2]UDT payload size is equal to the UDT packet size minus UDT packet header length (16 bytes). If `UDT_MSS` is 9,000, then UDT payload size is $9,000 - 20 - 8 - 16 = 8,956$ bytes.

achieves the peak throughput around 8.5 Gb/s. If we further increase the buffer size to 2 MB or 4 MB, the throughput performance decreases slightly, as shown in Figure 3.6(b). When we continue to increase the buffer size from 8 MB to 64 MB, the peak throughput further decreases, as shown in Figure 3.6(c). Our profiling results on the data block size show that a larger block size generally leads to a better performance, and an appropriately sufficient buffer is also necessary to ensure a satisfactory throughput performance. In this test case, a buffer size of 0.5 MB or 1.0 MB seems to be appropriate.

We also observe that if the block size is exactly an integer multiple of the UDT payload size, UDT exhibits very poor performance. As shown in Table 3.4, when the link layer MTU is 9000 bytes, if we set the block size to be an integer times of the payload size (i.e., $k \times 8956$), the observed average throughput is just around 0.08 Gb/s. However, if we set the block size to be slightly different values, e.g., $l = k \times \text{payload size} + \Delta_l$, as shown in Table 3.4, where k is from 1 up to 24 and $\Delta_l \in \{-1, +1, 0\}$, we observe much better performance up to 7.0+ Gb/s. A conjecture about the reason causes this phenomena is as follows: UDT employs timer-based selective acknowledgment and generates an acknowledgment at a fixed interval (0.01 second). On the receiver side, an irregular sized packet indicates the end of a message (block) and triggers an acknowledgement immediately. If the block size is exactly an integer multiple of the payload size, UDT completely depends on the timer-based acknowledging. Since the time interval is a fixed 10 milliseconds, there may not be enough ACK packets for sender to adjust its sending rate accordingly in a timely manner when the RTT is quite short (e.g., 0.04 milliseconds in our case); if the block size is not exactly an integer multiple of the payload size, more ACKs are generated and delivered (triggered by the last irregular packet in each block), in which case the sender receives more frequent ACKs and has more information to synchronize that in turn results in better throughput performance. We use Equation 3.3 to calculate

Table 3.4 A Special Case of UDT Profiling on Block Size Over a 10 Gb/s Back-to-back Connection ($B = 10$, $Q = 0.04$, $m = 8,972$, $l = k \times (m - 16) + \Delta_l$, $f = 1.0$, $p = 1$, and $d = 300$)

Δ_l	Times of Payload Size (k)						
	1	4	8	12	16	20	24
–1	2.785	5.547	6.398	6.842	6.649	7.371	7.400
+1	1.994	5.911	6.680	7.027	7.268	7.368	7.380
0	0.080	0.080	0.080	0.080	0.080	0.080	0.080

an appropriate block size, avoid wasting the space in a transmission unit, and trigger more ACKs for sender's responsiveness and potentially for better performance,

$$l = k \times (m - 16) \pm \Delta_l, \tag{3.3}$$

where m is the packet size and $k \in \{1, 2, \cdots\}$ is a positive integer, and Δ_l is also a positive integer value within the range $[1, m - 16]$.

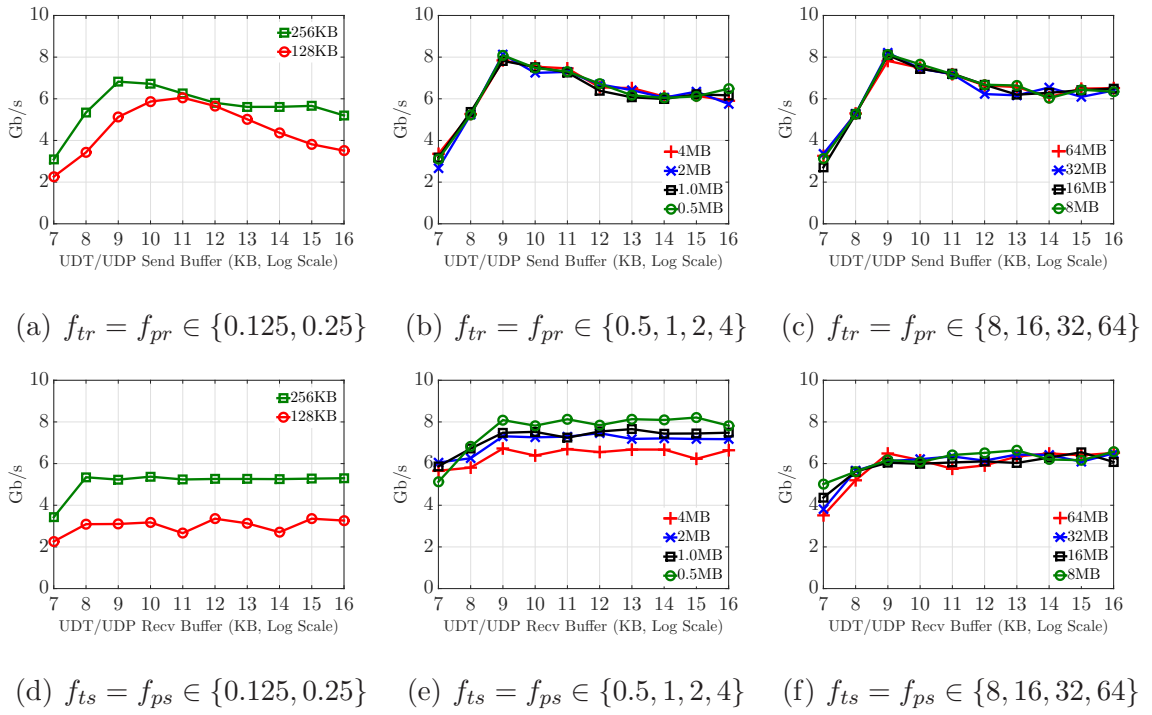
3.4.4 UDT Profiling on Buffer Size

We plot the performance measurements in response to various send/receive buffer sizes[3] in Figure 3.7, where the x-axis takes the logarithm of the actual send/receive buffer size (e.g., $7 = \log_2 128$ represents 128 KB). A rule of thumb for obtaining good transport performance is that both the send buffer and the receive buffer should be no less than the BDP, which is also true in our experiments with UDT.

In Figures 3.7(a), 3.7(b), and 3.7(c), as the send buffer size increases from 128 KB (2^7 KB) to 1024 KB (2^{10} KB), we observe a significant throughput improvement. In the case of a small receive buffer size, e.g., 128 KB or 256 KB in Figure 3.7(a), increasing the send buffer from 1.0 MB (2^{10} KB) to 64 MB (2^{16} KB) drastically decreases the throughput. It is probably due to the fact that a larger send

[3]We use f to denote UDT/UDP send/receive buffer sizes in general, and use subscripts to differentiate between different protocols, e.g., UDT send/receive buffer size (f_{ts} and f_{tr}) and UDP send/receive buffer size (f_{ps} and f_{pr}), see Table 3.3 for references.

buffer results in a longer RTT and in turn a larger BDP [48], which requires an even larger receive buffer to maintain the transfer speed; in the case of a large buffer size, e.g., from 1.0 MB (2^{10} KB) to 64 MB (2^{16} KB), we observe that the throughput first decreases, and then stabilizes around 6 Gb/s.



(a) $f_{tr} = f_{pr} \in \{0.125, 0.25\}$ (b) $f_{tr} = f_{pr} \in \{0.5, 1, 2, 4\}$ (c) $f_{tr} = f_{pr} \in \{8, 16, 32, 64\}$

(d) $f_{ts} = f_{ps} \in \{0.125, 0.25\}$ (e) $f_{ts} = f_{ps} \in \{0.5, 1, 2, 4\}$ (f) $f_{ts} = f_{ps} \in \{8, 16, 32, 64\}$

Figure 3.7 UDT profiling on buffer size over a 10 Gb/s back-to-back connection. $B = 10$, $Q = 0.04$, $d = 120$, $m = 8,972$, $l = 89,559$, and $p = 1$. **(a)**–**(c)** different curves correspond to different UDT/UDP receive buffer sizes; **(d)**–**(f)** different curves correspond to different UDT/UDP send buffer sizes.

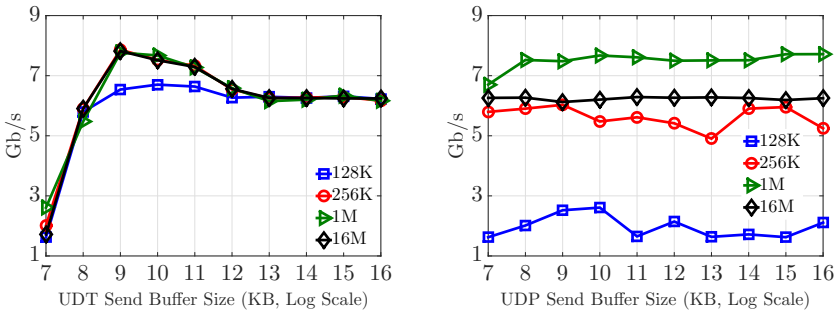
As shown in Figures 3.7(d), 3.7(e), and 3.7(f), a larger receiver buffer also generally leads to a better performance, but the improvement becomes less obvious as the receive buffer increases. In the case of a small send buffer (128 KB and 256 KB), increasing the receive buffer does not have an obvious positive effect as shown in Figure 3.7(d); and in the case of a large send buffer size, increasing the receive buffer greatly improves the performance, as shown in Figures 3.7(e) and 3.7(f). The profiling results on the buffer size show that to maintain a high transfer speed, a large receive

buffer is needed, and an appropriate send buffer is also necessary. A larger send buffer may incur a longer RTT and may not yield the best performance, in this test case, a send buffer of 0.5 MB or 1.0 MB turns out to be appropriate, which is consistent with the results in Section 3.4.3.

UDT Buffer and UDP Buffer Since UDT is implemented on top of UDP, both UDT and UDP buffer sizes may affect the end-to-end throughput performance. To send data packets, the `UDT::send()` function retrieves data from the application buffer, puts them in the UDT send buffer, and then returns (if in non-blocking mode). The data packets are sent to the receiver through the UDP channel by a data sending thread. Similarly, on the receive side, data packets are received through the UDP channel, and stored in the UDT receive buffer temporarily. When the `UDT::recv()` function is called at receiver site, it pulls data packets from the UDT receive buffer and delivers them to applications.

We next study the effects of different UDT buffer sizes and the UDP buffer sizes on the end-to-end throughput performance. We first investigate the relationship between the UDT send buffer (f_{ts}) and the UDP send buffer (f_{tr}), and their effects on the performance. Note that in this experiment both UDT/UDP receive buffer sizes are configured to be 16 MB to match the link speed. With a fixed UDP send socket buffer, we vary the UDT send buffer from 128 KB to 16 MB, collect throughput measurements, and plot them in Figure 3.8(a). Then, with a fixed UDT send buffer, we vary the UDP send buffer from 128 KB to 16 MB and plot the corresponding results in Figure 3.8(b).

The profiling results in Figure 3.8(a) show that when the receive buffer is sufficiently large, the UDT send buffer plays a critical role on the throughput performance. All the curves follow a similar pattern as the UDT buffer increases, which is insensitive to the UDP send buffer as long as it is set to be a reasonably



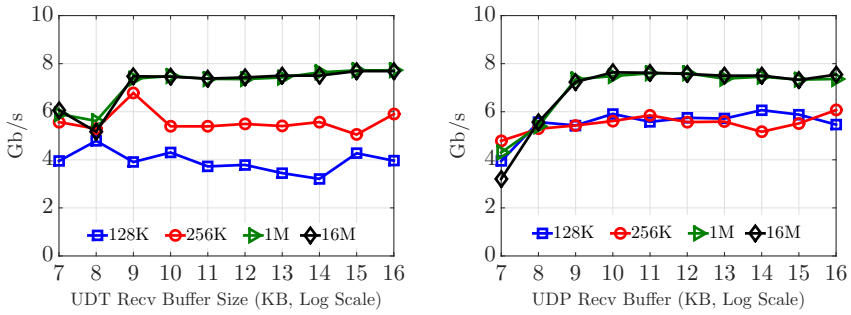
(a) $f_{ps} \in \{0.125, 0.25, 1, 16\}$ (b) $f_{ts} \in \{0.125, 0.25, 1, 16\}$

Figure 3.8 UDT profiling on UDT send buffer size and UDP send buffer size over a 10 Gb/s back-to-back connection. $B = 10$, $Q = 0.04$, $d = 180$, $m = 8,972$, $l = 89,559$, $f_{tr} = f_{pr} = 16$, and $p = 1$. **(a)** different curves correspond to different UDP send buffer sizes; **(b)** different curves correspond to different UDT send buffer sizes.

large value that does not limit the sending rate, which may depend on the network environment, e.g., larger than 256 KB in our case. We obtain the peak throughput when the UDT send buffer is 512 KB or 1 MB, and afterwards the throughput slightly decreases, which is consistent with the results shown in Figure 3.7.

As shown in Figure 3.8(b), with a fixed UDT send buffer size, varying the UDP send buffer does not affect the throughput too much, but different UDT send buffer sizes result in significantly different average throughput, i.e., when it is 512 KB or 1 MB, the average throughput is near 8 Gb/s; when it is increased further from 1 MB to 16 MB or even larger, the average throughput decreases to around 6 Gb/s.

Similarly, we also investigate the relationship between the UDT receive buffer (f_{ps}) and UDP receive buffer (f_{pr}) and their effects on the throughput performance. We set both the UDT/UDP send socket buffer sizes to be 1 MB based on the previous profiling results. With a fixed UDP receive buffer size, we vary the UDT receive buffer size and collect the corresponding throughput measurements shown in Figure 3.9(a). Also, with a fixed UDT receive buffer size, we vary the UDP receive buffer size and collect the corresponding throughput measurements shown in Figure 3.9(b).



(a) $f_{pr} \in \{0.125, 0.25, 1, 16\}$ (b) $f_{tr} \in \{0.125, 0.25, 1, 16\}$

Figure 3.9 UDT profiling on UDT receive buffer size and UDP receive buffer size over a 10 Gb/s back-to-back connection. $B = 10$, $Q = 0.04$, $d = 180$, $m = 8,972$, $l = 89,559$, $f_{ts} = f_{ps} = 1$, and $p = 1$. **(a)** different curves correspond to different UDP receive buffer sizes; **(b)** different curves correspond to different UDT receive buffer sizes.

Figure 3.9(a) show that a small UDP receive buffer limits the throughput even when the UDT receive buffer is large; when the UDP receive buffer is sufficiently large, the performance improves first but then reaches a plateau as the UDT receive buffer increases. As shown in Figure 3.9(b), with a small UDT receive buffer, increasing the UDP receive buffer does not improve the performance; but with a large UDT buffer, increasing the UDP receive buffer improves the performance, which is still limited by the UDT receive buffer though.

3.4.5 UDT Profiling on Parallel Streams

We vary the number of parallel streams and plot the corresponding aggregate throughput performance in Figure 3.10. We observe that with two parallel data streams, we achieve a throughput of 8 Gb/s, and a larger number of parallel streams may not necessarily lead to a better performance as shown in Figure 3.10, which is mainly due to the significant overhead incurred by memory copying, context switching, and multi-threaded implementation. On the hosts with sufficient computing resources, running multiple parallel data streams generally improves the

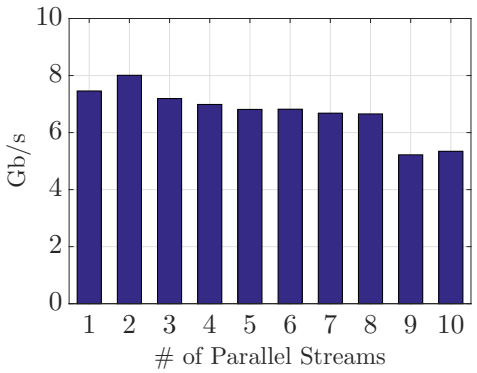


Figure 3.10 UDT profiling on parallel stream number over a 10 Gb/s back-to-back connection. $B = 10$, $Q = 0.04$, $d = 120$, $m = 8,972$, $l = 89,559$, and $f = 1.0$.

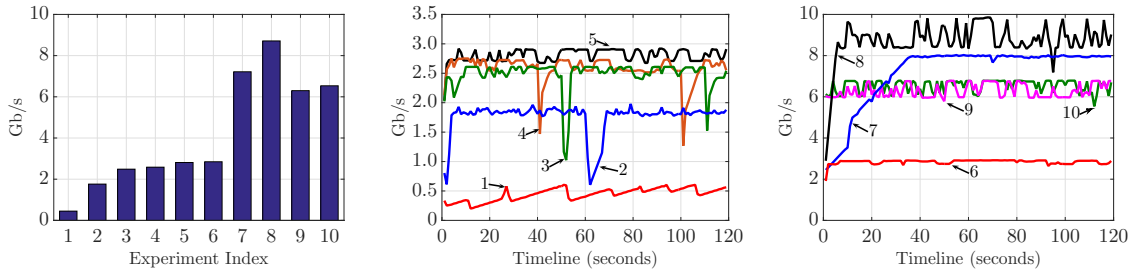
Table 3.5 Performance Comparison of Default UDT and TPG-tuned UDT Over a 10 Gb/s Back-to-back Connection ($B = 10$, $Q = 0.04$, $d = 120$, and $p = 1$)

Index	m	l	f_{ts}	f_{ps}	f_{tr}	f_{pr}	Gb/s
1	***1472***	1455	***10***	***1***	***10***	***1***	0.450
2	1472	8955	***10***	***1***	***10***	***1***	1.762
3	1472	65536	***10***	***1***	***10***	***1***	2.487
4	1472	89559	***10***	***1***	***10***	***1***	2.584
5	1472	89559	1	1	1	1	2.810
6	1472	179119	1	1	1	1	2.847
7	8972	89559	1	1	1	1	7.216
8	8972	179119	1	1	1	1	8.713
9	8972	179119	16	16	16	16	6.300
10	8972	179119	32	32	32	32	6.536

throughput performance, although the protocol itself may not be able to fully utilize the link bandwidth. Determining the optimal number of parallel streams is not straightforward as it highly depends on the configurations of end hosts and networks.

3.4.6 Comparison of Default UDT and TPG-tuned UDT

To illustrate how TPG improves the performance of UDT, we run 10 sets of data transfer experiments using default UDT and TPG-tuned UDT. The performance results are tabulated in Table 3.5 and further plotted in Figure 3.11 for a visual comparison. Note that the *italic* and **bold** numbers in Table 3.5 indicate that they



(a) (b) (c)

Figure 3.11 Performance comparison between default UDT and TPG-tuned UDT over a 10 Gb/s back-to-back connection. $B = 10$, $Q = 0.04$, $d = 120$, $p = 1$, and 1.0 second sampling interval. **(a)** average throughput; **(b)** instantaneous measurements in experiments 1 to 5; **(c)** instantaneous measurements in experiments 6 to 10.

are default values in UDT. We observe that the TPG-tuned UDT in experiment 8 achieves a significant performance improvement over any other parameter settings.

3.5 Profiling Over Dual NIC-to-NIC Connections

We present UDT profiling results in a local testbed with dual 10 Gb/s NIC-to-NIC connections at New Jersey Institute of Technology. Please also refer to Section 3.2.2 and Table 3.3 for notations.

3.5.1 Testbed Configuration

The testbed is established by connecting two high end servers `tiger.arcs.njit.edu` and `rabbit.arcs.njit.edu` that are both equipped with two 10 Gb/s NICs. The average round-trip delays of the two direct 10 Gb/s links between these two servers are both around 0.10 milliseconds, resulting in a BDP of 125 KB. The client and server hosts are both equipped with a 12 Cores Intel Xeon 2.40 GHz CPU, 16 GB RAM, and Red Hat 7 Linux Operating System updated with `3.10.0` kernel. The system's default and maximum memory space allowed for UDP socket buffer is configured to be 32 MB and 64 MB, respectively.

3.5.2 Profiling Results

We use TPG to create two UDT socket-based connections and bind each of them on one of the 10 Gb/s NIC-to-NIC connections to transfer data blocks for 180 seconds. The throughput performance over each of the two connections as well as their aggregations are measured and plotted in Figure 3.12.

As shown in Figure 3.12(a), corresponding to the packet size, the behaviors of UDT on both connections are quite similar. The throughput performance linearly increases as the UDT packet size increases linearly when two UDT connections are transferring data blocks simultaneously and independently.

We plot the throughput performance measurements correspond to various block sizes in Figure 3.12(b), which shows that larger block size brings higher performance and the improvement becomes less obvious as the block size keeps increasing. As the block size is linearly increased from 1 up to 24 times of the UDT payload size, the performance first linearly and significantly increases from 2.5 Gb/s to 8.0+ Gb/s and then stabilizes around 8.0 Gb/s for each of the NIC-to-NIC connections. The results shown in Figure 3.12(b) are different from the profiling results over the 10 Gb/s connection between two regular workstations shown in Figure 3.6, where the performance keeps increasing slightly when the block size approaches to 20 times of payload size; while in Figure 3.12(b), the throughput reaches the peak when block size is 10 times of payload size and the stabilizes around there as block size keeps increasing. The reason could be that a more powerful machine is more responsive and can handle more data blocks in the same period of time, so data blocks are transferred without staying in the buffer for longer time during which they may get lost. Therefore, the data transfer speed quickly reaches the peak performance given a relatively large block size (e.g., 10 times of payload size in this test case), and after which the overhead of the context switch between the protocol itself and the data transfer application (i.e., TPG) become less significant corresponding to the

capability of the end host, which makes the effects of further increased block size become marginal.

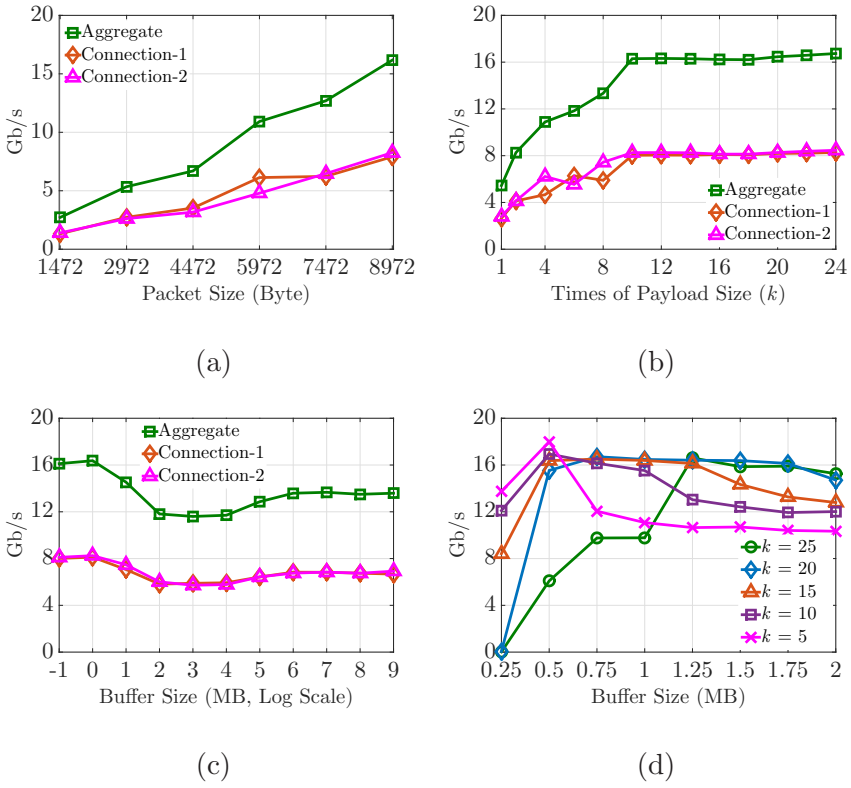


Figure 3.12 UDT profiling over dual 10 Gb/s back-to-back connections. $B = 20$, $Q = 0.1$, $d = 180$, and $p = 2$. **(a)** profiling on packet size, $l = 10 \times (m - 16) - 1$, and $f = 1$; **(b)** profiling on block size, $m = 8,972$, $l = k \times (m - 16) - 1$, and $f = 1$; **(c)** profiling on buffer size, $m = 8,972$, and $l = 179,119$; **(d)** profiling on buffer size (fine-grained), $m = 8,972$, and $l \in \{44779, 89559, 134339, 179119, 223399\}$.

In Figure 3.12(c), we set the block size to be a fixed 179119 bytes, vary the buffer size from 0.5 MB (2^{-1} MB) to 512 MB (2^{10} MB), and measure the throughput performance. Over a back-to-back connection with a 0.1 millisecond of RTT delay, increasing buffer may hurt the performance, and in our test case, a 0.5 MB buffer or a 1 MB buffer produces the best performance. In Figure 3.12(c), we get the optimal performance with a 1 MB (2^0 MB) buffer size and a given block size of 179,119 bytes.

In Figure 3.12(d) we stick around the buffer size at which we get the best performance in Figure 3.12(c) and conduct more tests with more fine-grained buffer size values, i.e., from 0.25 MB to 1 MB with a 0.25 MB interval. When the block size is relatively smaller, e.g., 44,779 bytes, a 0.5 MB of buffer is better than a smaller one (i.e., 0.25 MB), if the buffer size keeps increaseing, the aggregate performance decreases and stabilizes around 10 Gb/s from 18 Gb/s; when the block size is a moderately large value, e.g., 89,559 bytes, 134,339 bytes, or 179,119 bytes, the performance curves follow similar patterns, i.e., they significantly increase first, then slightly decrease after reaching the peak with a buffer size of 0.5 MB, and then stabilize as the buffer size increases. Larger buffer size makes the performance stabilize at a slightly higher performance (i.e., 12 Gb/s, 13 Gb/s, and 15 Gb/s, respectively); when the block size is aggressively large in comparison with buffer size, e.g., 223,899 bytes in this test case, UDT needs a relatively larger buffer (i.e., 1.25 MB) to achieve its peak performance (i.e., 16 Gb/s) that is slightly lower than the overall peak performance, 18 Gb/s, which is achieved when block size is 44,779 bytes and buffer size is 0.5 MB.

To sum up, the profiling results over the dual 10 Gb/s back-to-back NIC-to-NIC connections imply that UDT could behave well simultaneously and independently over different physical connections as long as the end hosts are powerful enough. The aggregate performance of UDT over dual physical NIC-to-NIC connections is simply the sum of the throughput over each connection without major interferences.

3.6 Profiling Over Long-haul Emulated Connections

We present UDT profiling results over emulated connections with different round trip times ranging from 0 ms to 366 ms at Oak Ridge National Laboratory (ORNL).

3.6.1 Emulated Testbed

For memory-to-memory data transfers, throughput measurements are collected between multi-core Linux host systems over a suite of emulated dedicated 10 Gb/s connections. The testbed consists of multiple Linux hosts of two types, 32-core and 48-core HP ProLiant servers, each with Broadcom 10GigE NICs, running Linux 2.6 kernel (CentOS release 6.6). It also consists of ANUE OC192 and IXIA 10GigE hardware connection emulators, and 10 Gb/s Force10 E300 WAN-LAN switch. These hardware connection emulators transport the physical packets between hosts, delaying them during the transit by the specified amount. We utilize these emulators to collect throughput measurements for a suite of dedicated connections with RTT $Q \in \{0, 11.8, 22.6.45.6, 91.6, 183, 366\}$ milliseconds. The lower RTTs match the physical back-to-back connections, the ones in the mid range represent U.S. cross-country connections, e.g., ones between DOE sites provisioned by OSCARS [7], and the higher RTTs represent trans-continental connections.

3.6.2 Profiling Results

We plot the UDT profiling results on packet size, block size, buffer size, and number of data streams over emulated connections with different RTT delays in Figure 3.13.

As shown in Figure 3.13(a), increasing packet size results in an almost linear increase on throughput performance across different RTT delays. With a sufficiently large buffer (e.g., larger than 256 MB in our test cases), UDT achieves similar performance with different delays. The longest delay of 366 ms results in the lowest performance comparing with other shorter delays.

The profiling results on block size in Figure 3.13(b) are consistent with our observations in Section 3.4.3, i.e., a larger block size generally leads to a higher performance across different RTT delays given sufficiently large buffer. Similarly, the connection with the longest delay of 366 ms has a lower performance than others.

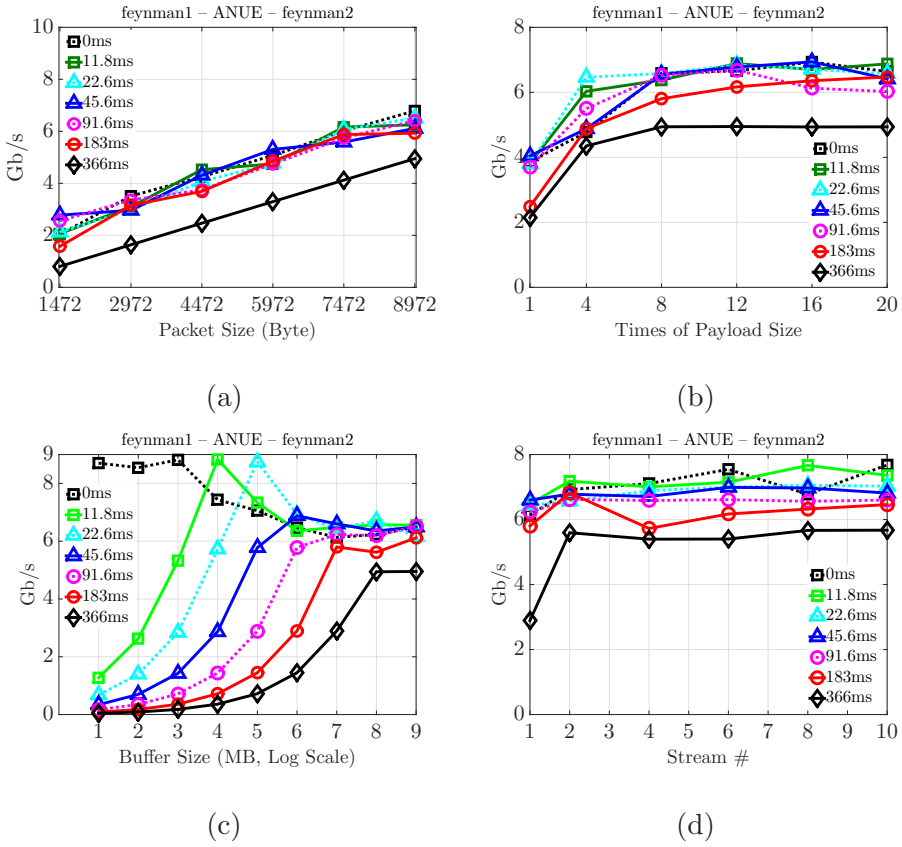


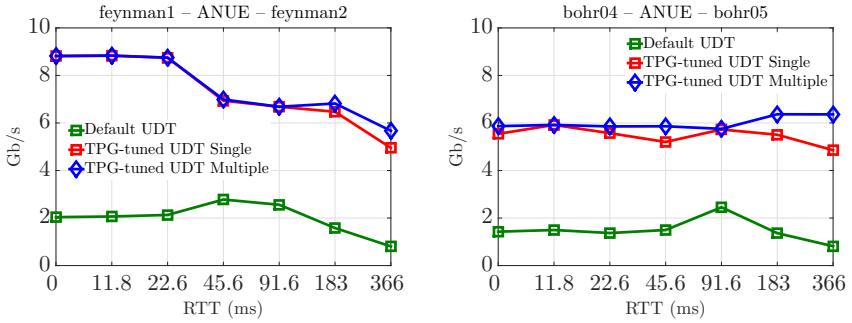
Figure 3.13 UDT profiling over 10 Gb/s emulated connections with different delays at ORNL. **(a)** $l = (m - 16) \times 10 - 1$, $f = 256$, $p = 1$, and $d = 600$; **(b)** $m = 8,972$, $f = 256$, $p = 1$, and $d = 600$; **(c)** $m = 8,972$, $l = 89,559$, $p = 1$, and $d = 600$; **(d)** $m = 8,972$, $l = 89,559$, $f = 128$, and $d = 600$.

Figure 3.13(c) also shows consistent profiling results on buffer size with the observations in Section 3.4.4 in terms of curve patterns. In view of the measurements across different RTTs, an appropriately sufficient buffer is necessary and larger ones may lead to lower performance, but determining an appropriate buffer size is not straightforward for connections with different delays. As shown in Figure 3.13(c), the buffer space required to achieve the peak throughput moves rightwards (increases) as the delay increases. Specifically, we have the following observations: i) on the back-to-back connection[4], the throughput decreases as buffer size increases from 2 MB and the

[4]The delay of the back-to-back connection is less than 1 ms, we simply denote it as 0 ms.

turning point is around 8 MB; ii) for delays of 11.8 ms and 22.6 ms, the performance significantly increases and reaches the peak as the buffer size increases from 2 MB, and then decreases and stabilizes at around 6.5 Gb/s to 7 Gb/s. A longer delay (22.6 ms) requires a larger buffer size (32 MB) to reach the peak throughput than a shorter one (11.8 ms), which requires a relatively smaller buffer size (16 MB); iii) for further longer delays, i.e., 45.6 ms, 91.6 ms, 183 ms, and 366 ms, their performance curves follow a similar pattern, i.e., first increase and reach the peak, and then stabilize; and iv) generally, connection with longer delay needs larger buffer to reach the peak throughput, and both the peak and stabilized throughput decrease as RTT increases.

It is confirmed again in Figure 3.13(d) that the optimal number of data streams is not straightforward to determine. UDT does not seem to be best-suited for multiple parallel data streams. There are not any clear patterns about the performance change as the number of parallel stream changes over connections with different delays.



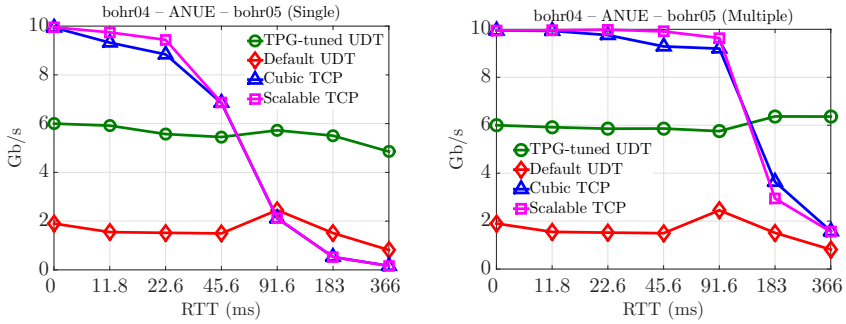
(a) (b)

Figure 3.14 Maximal observed performance comparison between default UDT and TPG-tuned UDT without packet loss over ORNL 10 Gb/s emulated connections with various delays.

3.6.3 Comparison with Default UDT

We plot in Figure 3.14 the performance comparison between default UDT and TPG-tuned UDT for both single and multiple streams over connections with different RTT delay values. It shows that the tuned parameter settings using the profiling approach

enabled by TPG greatly improve the UDT performance in comparison with its default settings. It is also confirmed again that using multiple UDT data streams does not make significant performance improvement on the average throughput in comparison with single UDT data stream.



(a) Single stream (b) Multiple streams

Figure 3.15 Maximal observed performance comparison between Cubic TCP, Scalable TCP, default UDT, and TPG-tuned UDT over ORNL 10 Gb/s emulated connections with various delays and zero packet loss.

3.6.4 Comparison with TCP

We compare the the maximum throughput achieved by TCP variants (Cubic TCP [52] and Scalable TCP [58]), default UDT, and TPG-tuned UDT using both single and multiple data streams over the emulated connections. Figure 3.15 shows the performance comparison when the emulated connection generates zero packet loss, and Figure 3.16 shows the performance comparison when the emulated connection generates 0.1% packet loss that follows four different distributions including gaussian, periodic, poisson, and uniform.

Both Figure 3.15 and Figure 3.16 show that TPG-tuned UDT produces more stable performance over connections with various RTT delays. UDT has the capabilities (after tuned by TPG) to outperform both itself with the default settings and various TCP variants on the connections over a certain delay. We also notice that in Figure 3.16 with such a level (0.1%) of loss rate, multiple data streams greatly help

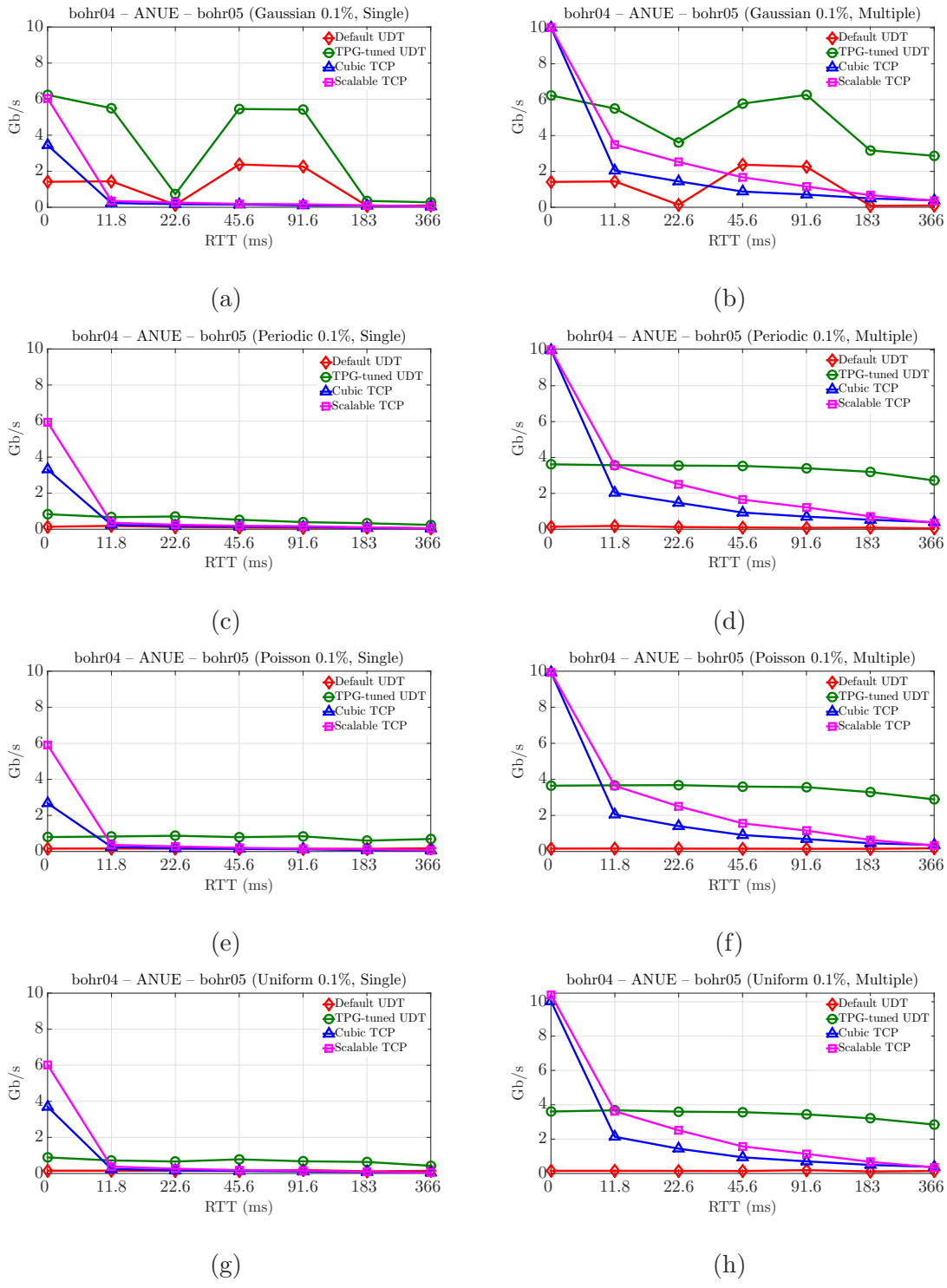


Figure 3.16 Maximal observed performance comparison between Cubic TCP, Scalable TCP, default UDT, and TPG-tuned UDT over ORNL 10 Gb/s emulated connections with various delays and 0.1% packet loss.

improving the end-to-end aggregate throughput of both TCP and UDT for all four types of loss distributions. TCP outperforms UDT for short delays, but UDT is not as sensitive to delays as TCP, which indicates that UDT seems to be more suitable for big data transfer over long-hual high-speed connections. TPG-tuned UDT is able to outperform TCP with a certain delay for both single and multiple data streams.

3.7 Profiling Over Long-haul Physical Connections

We conduct extremely extensive (it includes 12,825 one-time profilings and totally takes around 18 days to finish) UDT data transfer experiments using TPG over the 10 Gb/s physical connection between a sender (`tubes.ftm.alcf.anl.gov`) at Argonne National Laboratory (ANL) and a receiver (`midway.rcc.uchicago.edu`) at University of Chicago (UChicago). We present the complete profile in this section.

3.7.1 Profiling Results

This long-hual (380 ms) physical connection is engineered to have such a long RTT delay through layer 2 circuit within ESnet that from ANL hits the west coast and back before it gets to University of Chicago.

We conduct UDT-based data transfer tests using TPG and build a complete profile (a 2D table, see Chapter 4) that will be used in Chapter 4 by exhausting 12,825 combinations of the values of buffer size and block size. The profiling resolutions of the block size and the buffer size are set to be one payload size and 2 MB, respectively. Every "one-time profiling" is set to take 120 seconds[5].

Figure 3.17 shows the complete UDT transport profile over the 380 ms physical connection corresponds to block size and block size, where the UDT packet size is

[5]Longer time for a "one-time profiling" may improve the profiling accuracy. A two minutes profiling time for each test seems to be a good trade-off between accuracy and efficiency based on our experiences of conducting transport profiling over this physical connection.

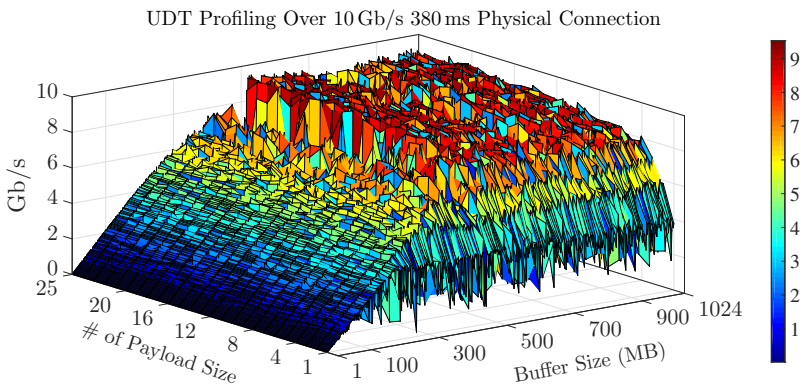


Figure 3.17 Complete UDT transport profile over the 10 Gb/s 380 ms physical connection from Argonne National Laboratory to University of Chicago.

8,972 bytes, the block size changes from 1 UDT payload (i.e., $8,972 - 16 = 8,956$ bytes) to 25 UDT payloads, the buffer size changes from 1 MB to 1 GB with a 2 MB interval, and the stream number is set to be one ($p = 1$). Over such a 380 ms 10 Gb/s connection, as shown in Figure 3.17, both block size and buffer size may limit the end-to-end performance. As for the buffer size, the BDP rule is still valid: if the buffer size is less than 500 MB, the maximal achieved throughput is less than 6 Gb/s (the connection has a BDP of 475 MB); if the buffer size is larger than 500 MB and up to 1 GB, UDT has a chance to obtain performance comparable to the connection speed, but occasionally, the achieved throughput is still only 6 Gb/s. As for the block size, larger block size seems to be necessary to achieve optimal performance because when the block size is small (e.g., 1 to 4 times of UDT payload), the achieved throughput is mostly less than 7 Gb/s, and when the block size is large (e.g., more than 5 times of UDT payload), UDT could achieve higher throughput and reach the connection speed given sufficient buffer. Figure 3.17 implies that both large block sizes and large buffer sizes are necessary but not sufficient conditions for UDT to achieve optimal performance over high-speed long-haul connections. It is not straightforward to derive the best parameter values (at least for block size and buffer size) using an analytical approach especially when the network environment is subject to frequent changes.

CHAPTER 4

TRANSPORT PROFILING OPTIMIZATION

4.1 Introduction

The exhaustive search-based profiling approach as detailed in Chapter 3 is prohibitively time consuming when there exists a large parameter space, which is almost always the case in most transport scenarios. In general, users may not be in favor of performing transport profiling if the profiling overhead is comparable with the time needed for their actual data transfer.

To improve the efficiency of transport profiling, we propose a stochastic approximation-based profiling method, referred to as `FastProf`, to quickly determine the optimal operational zone of a given data transfer method in high-performance network environments. `FastProf` employs the Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm [77] to accelerate the exploration of the control parameter space.

We implement the proposed method by leveraging the existing Transport Profile Generator (TPG) [92], and test it using both emulations with real-life performance measurements and experiments over physical connections with short (2 ms) and long (380 ms) delays. Both the emulation and experimental results show that `FastProf` significantly reduces the profiling overhead while achieving a comparable level of end-to-end throughput performance with the exhaustive search-based approach. `FastProf` makes it possible to conduct "on-line" profiling to support time-critical data transfer, and provides an additional level of intelligence to existing profiling-oriented toolkits such as `iperf3` [8] and `xddprof` [20].

4.2 Profiling Overhead

The goal of transport profiling is to find the parameter values θ^*, at which the throughput $G(\theta^*)$ reaches its global maximum. An exhaustive search-based transport profiling is able to find the optima, but is very time consuming, and therefore is particularly unsuitable for network environments that are subject to frequent changes (e.g., configurations of end hosts, connection delay, connection bandwidth, etc.).

As a numerical example, in the Transport Profile Generator (see Chapter 3), the selected UDT [48] transport method includes several commonly accessible parameters including packet size ($m \in \{m_1, m_2, \cdots, m_{N_m}\}$), block size ($l \in \{l_1, l_2, \cdots, l_{N_l}\}$), buffer size ($f \in \{f_1, f_2, \cdots, f_{N_f}\}$), and number of parallel data streams ($p \in \{p_1, p_2, \cdots, p_{N_p}\}$). If a one-time profiling takes Δt (typically on the order of several minutes) to finish, it takes a total of $\Delta t \cdot N_m \cdot N_l \cdot N_f \cdot N_p$ to generate a complete profile before the actual data transfer. In the emulations in Section 4.5, we fix the packet size m (i.e., $N_m = 1$) and the number p of parallel data streams (i.e., $N_p = 1$), and vary the block size from 1 to 25 times of the UDT payload size (i.e., $N_l = 25$) and the buffer size from 1.0 MB to 1.0 GB with a 2.0 MB step (i.e., $N_f = 513$). If a one-time profiling takes $\Delta t = 2$ minutes, the exhaustive search would take 25,650 minutes (around 18 days) in total, and hence is impractical in real-life applications. As both the number of control parameters and the profiling resolution increase, the time to obtain a complete profile rapidly increases, making the exhaustive search-based approach practically infeasible.

4.3 Fast Profiling Based on Stochastic Approximation

To obviate the need of conducting an exhaustive profiling [92], we propose a fast profiling method based on the Simultaneous Perturbation Stochastic Approximation

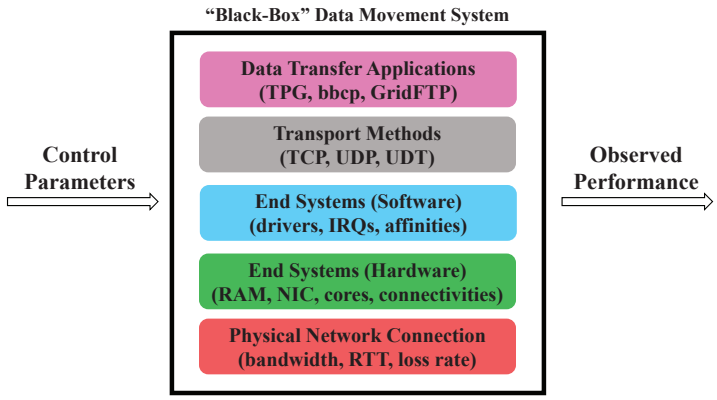


Figure 4.1 The "black-box" data movement system.

algorithm [77], referred to as `FastProf`, to quickly determine the "best" parameter values prior to actual data transfer.

4.3.1 Rationale on the Use of Stochastic Approximation Methods

Figure 4.1 shows a typical data transfer scenario where a user request, which specifies a sender host and a receiver host, is processed for data transfer in a certain network environment. Since we mainly focus on transport profiling at the application layer rather than system tuning at lower layers, the whole data transfer process could be treated as a "black box" system, where the input is the set of control parameter values θ and the output is the corresponding throughput measurement $G(\theta)$. Based on this model, the SPSA algorithm is appropriate to be used for quickly determining the optimal parameter values because: i) it does not require an explicit formula of $G(\theta)$ but only its "noise corrupted" measurements $y = G(\theta) + \xi$, which can be obtained by running a "one-time profiling" using existing tools such as `iperf3` [8] and TPG [92] with a set of specified parameter values; ii) it does not require any additional information about system dynamics or input distribution. These are highly desirable features as they account for the dynamics and randomness in: i) data transfer process; ii) end host and network environments, and iii) performance measurements.

For a given data transfer, if the jumbo frame is supported along the path, it is desirable to enable it to minimize per-packet overhead [33]. Thus, the packet size m can be decided by exploring the Path MTU (PMTU) without profiling. Although multiple parallel data streams may improve performance, they typically introduce inter-stream competition that may lead to complex transfer dynamics even over dedicated connections. Furthermore, since many high-performance transport methods including UDT are not best suited for environments with a high level of concurrency [49], we focus our study on one single data stream (i.e., $p = 1$). Similar to TPG (see Chapter 3 and [92]), where UDT is used in the implementation as a use case, in this Chapter, we also use UDT as an example transport method and consider block size l and buffer size f as its control parameters (or the input of the black box system in Figure 4.1), i.e., $\theta = [l, f]^T$.

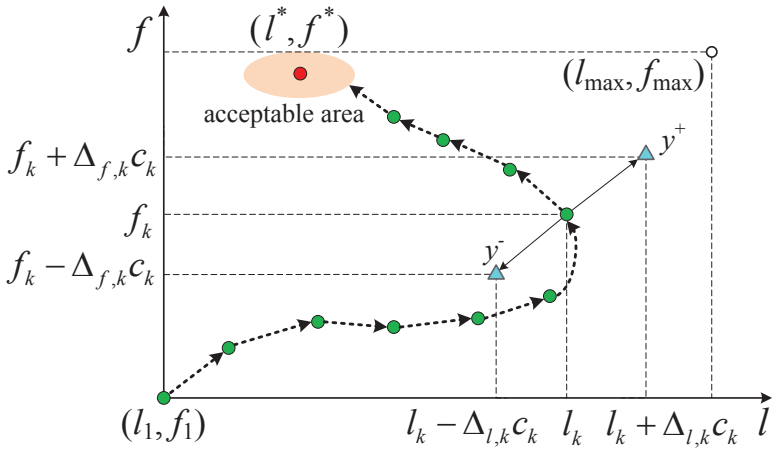


Figure 4.2 Visualized profiling process of `FastProf`.

In the exhaustive search, we need to construct a complete 2D table of performance measurements by running $(N_l \cdot N_f)$ times of one-time profiling, as visualized in Figure 4.2, where the l-axis and f-axis represent block size and buffer size, respectively. Note that each data point in Figure 4.2 is actually a 3-tuple $(l, f, G(l, f))$, where $G(l, f)$ is the corresponding observed throughput. The proposed

`FastProf` method attempts to explore a path of profiling data points in this 2D table to reach the global optimum within an "acceptable area".

4.3.2 Stochastic Approximation Methods

Based on the model shown in Figure 4.1, we assume that the average throughput performance is a function $G(\theta)$ of control parameters θ. The goal is to find the control parameters θ^* that maximize $G(\theta)$ within the feasible space Θ, i.e., $\max_{\theta \in \Theta} G(\theta)$. Following the standard Kiefer-Wolfowitz Stochastic Algorithm (KWSA) [59], we have

$$\hat{\theta}_{k+1} = \hat{\theta}_k + a_k \cdot \hat{g}_k(\hat{\theta}_k), \tag{4.1}$$

where $a_k > 0$ is a scalar gain coefficient, $g(\theta) \equiv \dfrac{\partial G(\theta)}{\partial \theta}$ is the gradient of $G(\theta)$, and $\hat{g}(\hat{\theta}_k)$ is an approximation of $g(\theta_k)$.

We assume that the "noise corrupted" observation, denoted as $y(\theta)$, is available at any value of $\theta \in \Theta$, given by

$$y(\theta) = G(\theta) + \xi, \tag{4.2}$$

where ξ is the noise incurred by the randomness in the network connection and end-host system dynamics. In fact, $y(\theta)$ is the observed average throughput performance of a one-time profiling with a specific set of parameter values θ during a specific time duration $[0, \Delta T]$. Given $\theta = [l, f]^T$, the solution based on the classical KWSA method is a multi-variable recursive optimization procedure, defined as

$$\begin{bmatrix} l_{k+1} \\ f_{k+1} \end{bmatrix} = \begin{bmatrix} l_k \\ f_k \end{bmatrix} + a_k \begin{bmatrix} \hat{g}_{l_k}\left(\begin{bmatrix} l_k \\ f_k \end{bmatrix}\right) \\ \hat{g}_{f_k}\left(\begin{bmatrix} l_k \\ f_k \end{bmatrix}\right) \end{bmatrix}. \tag{4.3}$$

The gradient $g(\theta)$ of the function $G(\theta)$ is approximated by a "two-sided" finite difference given by

$$\begin{cases} \hat{g}_{l_k}\left(\begin{bmatrix} l_k \\ f_k \end{bmatrix}\right) = \dfrac{\hat{y}\left(\begin{bmatrix} l_k + c_k \\ f_k \end{bmatrix}\right) - \hat{y}\left(\begin{bmatrix} l_k - c_k \\ f_k \end{bmatrix}\right)}{2c_k} \\ \\ \hat{g}_{f_k}\left(\begin{bmatrix} l_k \\ f_k \end{bmatrix}\right) = \dfrac{\hat{y}\left(\begin{bmatrix} l_k \\ f_k + c_k \end{bmatrix}\right) - \hat{y}\left(\begin{bmatrix} l_k \\ f_k - c_k \end{bmatrix}\right)}{2c_k} \end{cases}, \tag{4.4}$$

where c_k is a small positive number. The coefficients a_k and c_k in the above equations satisfy the following conditions to guarantee the convergence

$$\lim_{k\to\infty} a_k = 0,\ \lim_{k\to\infty} c_k = 0,\ \sum_{k=1}^{\infty} a_k = \infty,\ \sum_{k=1}^{\infty} (\frac{a_k}{c_k})^2 < \infty. \tag{4.5}$$

We further explore the Simultaneous Perturbation Stochastic Approximation (SPSA) [77, 78] algorithm to further reduce the profiling overhead. Instead of collecting observations along all dimensions of the gradient, SPSA randomly perturbs the control parameter set in two separate directions and collect two corresponding measurements. The gradient approximation of the throughput function based on SPSA is given by

$$\begin{cases} y^+ = y(\theta + \Delta_k c_k) = y\left(\begin{bmatrix} l_k + \Delta_{l,k} c_k \\ f_k + \Delta_{f,k} c_k \end{bmatrix}\right) \\ \\ y^- = y(\theta - \Delta_k c_k) = y\left(\begin{bmatrix} l_k - \Delta_{l,k} c_k \\ f_k - \Delta_{f,k} c_k \end{bmatrix}\right) \end{cases}, \tag{4.6}$$

$$\begin{cases} \hat{g}_{l_k}(\theta_k) = \hat{g}_{l_k}\left(\begin{bmatrix} l_k \\ f_k \end{bmatrix}\right) = \dfrac{y^+ - y^-}{2\Delta_{l,k}c_k} \\ \\ \hat{g}_{f_k}(\theta_k) = \hat{g}_{f_k}\left(\begin{bmatrix} l_k \\ f_k \end{bmatrix}\right) = \dfrac{y^+ - y^-}{2\Delta_{f,k}c_k} \end{cases}, \tag{4.7}$$

where the coefficient sequence $\{\Delta_{i,k}\}$ ($i = 1, \cdots, d$ for d dimensional vector, and in this work $d = 2$) are independent and symmetrically distributed around 0 with finite inverse $E|\Delta_{i,k}^{-1}|$ over all parameter components i and time steps k. A simple and effective way to decide each component of $\{\Delta_{i,k}\}$ is to use symmetric Bernoulli ± 1 distribution with a probability of 0.5 for each outcome of either +1 or –1 [78].

4.3.3 Convergence of SPSA-based `FastProf`

The convergence of SPSA-based `FastProf` is important as it affects the quality of the profiling results as well as the efficiency of the profiler. To explore the applicability of SPSA in the profiling optimization problem and investigate its convergence property, we justify the conditions that lead to the convergence in the context of `FastProf`.

As pointed out by Spall in [79] (pp. 161), the conditions for convergence can hardly be all checked and verified in practice due to the lack of knowledge on $G(\theta)$. We provide some intuitive arguments based on the problem nature, and some empirical verifications based on the extensive experiments to justify the appropriateness of SPSA in the profiling optimization problem. According to the Theorem 7.1 in [79] (pp. 186), if Conditions B.1″ – B.6″ hold and θ^* is a unique maximum of $G(\theta)$, then for SPSA, $\hat{\theta}_k$ almost surely converges to θ^* as $k \to \infty$.

The coefficient sequences a_k and c_k we choose and the distribution we follow to generate the simultaneous perturbations $\{\Delta_{i,k}\}$ easily validate Conditions B.1″ and B.6″ (see Section 4.4.1 and [78] for more details).

The main concern of Conditions B.2″ and B.3″ is to ensure that $\hat{\theta}_k$ is close enough to θ^* such that $\hat{\theta}_k$ has a tendency to converge to θ^*. These two conditions are valid in our problem scenario because: i) the requirement for Condition B.3″, i.e., $\sup_{k\geq 0}\left\|\hat{\theta}_k\right\| < \infty$, can be verified since the control parameter values of block size and buffer size are both finite positive numbers; ii) since the feasible regions of the control parameters of `FastProf` are finite and mapped to a limited range (e.g., $[1.0, 25.0]$ in our test cases in Section 4.5 and Section 4.6) of the iterative variables, $\hat{\theta}_k$ (including the starting point) is sufficiently close to θ^*; iii) θ^* is not a single point but an "acceptable area" including a set of adjacent points (see Figure 4.2); iv) from a practical point of view, `FastProf` attempts to move $\hat{\theta}_k$ to the nearest point within the feasible space if it deviates from the feasible space, and then adjust the step size accordingly to avoid such situations; and v) our extensive emulations and experiments in Section 4.5 and Section 4.6 confirm the closeness and natural tendency even if the starting point is randomly selected.

The way we generate the simultaneous perturbations $\{\Delta_{i,k}\}$ (see Section 4.4.1) ensures that $\{\Delta_{i,k}\}$ is a mutually independent sequence, which is independent of $\hat{\theta}_0, \hat{\theta}_1, \cdots, \hat{\theta}_k$. The observed noise during data transfer is mainly caused by the dynamics of end-hosts and network segments. Since we measure the *average* throughput in time duration $[0, \Delta T]$ (see Equation 3.1), which captures both the positive noise and negative noise, the long-term conditional expectation of the observed noise is considered to be zero, i.e., $E[(\xi^{(+)} - \xi^{(-)})|\{\hat{\theta}_0, \hat{\theta}_1, \cdots, \hat{\theta}_k\}, \Delta_k] = 0$. In addition, since $\{\Delta_{i,k}\}$ is generated following the symmetric Bernoulli ± 1 distribution with a probability of 0.5 for each outcome of either +1 or –1, $E|\Delta_{i,k}^{-1}|$ is uniformly bounded. The observations $y(\hat{\theta}_k \pm c_k\Delta_k)$ are also bounded by the link capacity, so the ratio of measurement to perturbation $E\left[\left(\frac{y(\hat{\theta}_k \pm c_k\Delta_k)}{\Delta_{i,k}}\right)^2\right]$ is uniformly bounded over i and k. Hence, Condition B.4″ holds.

As for Condition B.5″, it is theoretically unverifiable whether $G(\theta)$ is three-times continuously differentiable and bounded since $G(\theta)$ is practically unknown. However, the smoothness of $G(\theta)$ can be intuitively verified based on the nature of the problem being studied since the throughput $G(\theta)$ as well as the gradient $g(\theta) \equiv \frac{\partial G(\theta)}{\partial \theta}$ are at least bounded by the connection capacity and the finite feasible region of θ.

In addition to the above justification, we also would like to point out that although the throughput performance should have a unique theoretical peak over the feasible control parameter space given a specific snapshot of the status of end-hosts and network environments, it has been observed in our experiments that different runs with identical parameter values may yield different throughputs, which makes the uniqueness of θ^* unverifiable. But the observed performance $y = G(\theta) + \xi$ indeed shows a peak property over the feasible parameter region. As shown in Figure 3.13(b), increasing the block size improves the performance if the buffer is sufficient; otherwise, increasing the block size decreases the performance especially when the block size is comparable with the buffer size [92]. In Figure 3.13(c), it is more clearly shown that larger buffer sizes may not always lead to better performance given a fixed block size, and this trend is consistent for different RTTs only with different turning points.

4.4 Implementation of An SPSA-based Transport Profiler

4.4.1 An SPSA-based Profiling Process

We present our SPSA-based profiling process as follows.

1. Select an either fixed or random starting point within the feasible space of block size (l) and buffer size (f).
2. Check the termination conditions (see Section 4.4.3 for details) to continue or terminate the profiling process.
3. Calculate $a_k = \frac{a}{(A+k+1)^\alpha}$ and $c_k = \frac{c}{(k+1)^\gamma}$, where we set $\alpha = 0.602$, $\gamma = 0.101$, and $A = 0.0$ (or other values much less than the expected/allowed

number of iterations). The step sizes a and c are determined empirically based on the size of the entire search space.

4. Generate a pair $\Delta_{\{l,\,f\}} \in \{+1,\,-1\}$ of perturbations following the symmetric Bernoulli ± 1 distribution with a probability of 0.5 for each outcome of either +1 or –1.

5. Perform one-time profiling twice to collect two corresponding throughput observations $y^{\pm}$, see Equation 4.6.

6. Generate the simultaneous perturbation approximation to the unknown gradient $g(\theta_k)$ using Equation 4.7.

7. Apply the standard stochastic approximation form (Equation 4.1) to update θ_k to a new value θ_{k+1}, increase k by 1 (i.e., $k = k + 1$), and go back to Step 2.

4.4.2 Profiling Precision

We set the two elements of the control parameter set used in the stochastic approximation model, denoted as $\theta' = [l', f']^T$, as positive numbers within a reasonably selected range to ensure a comparable magnitude of each parameter. We perform a rounding operation in calculating the actual values of the control parameters l and f in the case of fractional results.

The profiling unit of block size, denoted by μ_l, is defined as one payload size, and the block size l ($l \geq 1$) is defined as a multiple of the payload size. The profiling process transfers a data block of $\lambda_l(l') \cdot \mu_l$ bytes each time by calling the `appsend()/apprecv()` functions, which may in turn call the `send()/recv()` API functions of the underlying transport protocol multiple times to completely deliver an entire data block. If a UDP-based protocol such as SABUL [47] or UDT [48] is used, it is recommended to set the block size to be a multiple of the protocol's payload size if possible to avoid UDP automatic segmentation. For example, UDT's payload size is UDT data packet size (m) minus UDT header length [47], i.e., the profiling unit of block size μ_l is given by

$$\mu_l = m - 16 = (\text{MTU} - 28) - 16 = \text{MTU} - 44, \tag{4.8}$$

where UDT header is of 16 bytes, and IP and UDP headers are of 20 and 8 bytes, respectively (hence 28 bytes in total). For example, on our testbed in Section 4.5.1, the jumbo frame is enabled and the MTU is 9,000 bytes, then μ_l is 8,956 bytes.

The profiling unit of buffer size, denoted by μ_f, is decided by the specific profiling precision chosen by the end user in any unit within a feasible profiling range, e.g., 1 Byte, 512 KB, 1.0 MB, 2.0 MB, or others.

Based on the above profiling units, we can calculate the actual values of block size (l) and buffer size (f) for performance observations (i.e., the calculations of y^+ and y^- through one-time profilings) given by

$$\begin{cases} l = round\big(\lambda_l(l') \cdot \mu_l\big) \\ f = round\big(\lambda_f(f') \cdot \mu_f\big) \end{cases}, \tag{4.9}$$

where λ_l and λ_f are scaling functions that may take different profiling patterns. For example, with a function $\lambda_f(f') = 2^{f'}$, the buffer size would exponentially increase as the iterative value of buffer size f' increases.

4.4.3 Termination Conditions

Many efforts (e.g., [40, 82]) have been devoted to the termination conditions of SA methods since the KWSA algorithm was first proposed [59]. In `FastProf`, we consider the following three simple and practical conditions to guarantee the performance and the termination of a profiling process.

Early Termination – the $\mathcal{C}$ rule The best throughput performance y^* of a given data transfer method over a given network connection is unknown until a complete profile is obtained. We define the *performance gain ratio* of a one-time profiling as

$$\rho = \frac{y}{y^*}, \tag{4.10}$$

where y is the observed throughput of a one-time profiling. Over a dedicated connection, we consider bandwidth B as a known constant since it is reserved in advance through services such as OSCARS [7] and set $y^* = B$. When `FastProf` reaches an operational zone that produces a throughput y with a performance gain ratio no less than $\mathcal{C}$, i.e.,

$$\rho = \frac{y}{y^*} = \frac{y}{B} \geq \mathcal{C}, \tag{4.11}$$

`FastProf` terminates. Note that this condition may or may not be satisfied in a certain one-time profiling.

Upper Bound – the $\mathcal{M}$ rule `FastProf` terminates when the number of one-time profilings exceeds a threshold $\mathcal{M}$, which is typically set as $\mathcal{M} \ll (N_l \cdot N_f)$. If $\mathcal{M} = (N_l \cdot N_f)$, `FastProf` rolls back to the exhaustive search as in TPG [92].

Impeded Progress – the $\mathcal{R}$ rule If the number of consecutive iterations that do not produce any performance improvement compared with the best one observed so far exceeds an upper bound ($\mathcal{R}$), `FastProf` terminates.

4.5 Emulation-based Performance Evaluation

We conduct profiling emulations using the profiling data collected on a real-life testbed to gain insights into the behaviors of `FastProf` and also compare `FastProf` with other search algorithms including random walk [61] and Tabu search [44].

4.5.1 Data Collection

We build a complete profile (the 2D table in Figure 4.2) by exhausting $N_l \cdot N_f = 12{,}825$ combinations of block size and buffer size, whose profiling resolutions are set to be one payload size and 2 MB, respectively. These results are collected by running TPG

tests over a 10 Gb/s 380 ms connection between a sender host at Argonne National Laboratory (ANL) and a receiver host at University of Chicago (UChicago). We conduct profiling emulations based on this complete profile.

4.5.2 Scaling Functions and Parameter Settings

We use Equation 4.12 to calculate the actual parameter values ($\theta = [l,\ f]^T$) based on the iterative parameter values ($\theta' = [l',\ f']^T$),

$$\begin{cases} l = round\Big(\lambda_l(l') \cdot \mu_l\Big) = \\ round\left\{\left((l' - l'_{\min}) \cdot \dfrac{(l_{\max} - l_{\min})}{(l'_{\max} - l'_{\min})} + l_{\min}\right) \cdot \mu_l\right\} \\ \\ f = round\Big(\lambda_f(f') \cdot \mu_f\Big) = \\ round\left\{\left((f' - f'_{\min}) \cdot \dfrac{(f_{\max} - f_{\min})}{(f'_{\max} - f'_{\min})} + f_{\min}\right) \cdot \mu_f\right\} \end{cases} . \tag{4.12}$$

In particular, the profiling range for block size is from 1 payload to 25 payloads (i.e., $l_{\min} = 1$, $l_{\max} = 25$, and $\mu_l = 8{,}956$ bytes), and the profiling range for buffer size is from 1 MB to 1 GB (i.e., $f_{\min} = 1$, $f_{\max} = 1{,}024$, and $\mu_f = 1{,}048{,}576$ bytes). We set the iterative variables for both block size and buffer size to be from 1.0 to 25.0 (i.e., $l'_{\min} = 1.0$, $l'_{\max} = 25.0$, $f'_{\min} = 1.0$, and $f'_{\max} = 25.0$) to ensure that they are of comparable and consistent magnitudes.

4.5.3 Performance Measurements

We consider several different parameters in the profiling emulations: i) the starting point (SP) (either fixed or randomly selected); ii) the $\mathcal{A}$ rule (either disabled or enabled)[1]; iii) the value of $\mathcal{M}$ (either ∞ or $2\mathcal{R}$); iv) the value of $\mathcal{C}$ (selected from $\{0.95, 0.90, 0.80\}$); and v) the value of $\mathcal{R}$ (integers from 1 to 50). We measure four types of performance metrics in 5,000 runs: i) the average throughput; ii) the

[1]If $\mathcal{A}$ is enabled, `FastProf` only searches in the feasible space where the buffer size is larger than BDP, i.e., $f \geq$ BDP; otherwise, it searches the entire feasible space.

percentage that leads to a desired performance; iii) the average profiling time as indicated by the average number of one-time profilings; and iv) the longest profiling time as indicated by the maximum number of one-time profilings. The subfigures in each figure in this section correspond to these four types of performance metrics labeled by $\mathcal{A}$, $\mathcal{M}$, $\mathcal{C}$, and starting point (SP), respectively.

Overall Performance We randomly select a starting point within the feasible region of control parameters, disable the $\mathcal{A}$ rule, and plot the results in Figure 4.3, which shows that `FastProf` is able to find a set of control parameter values that result in satisfactory performance in a short time.

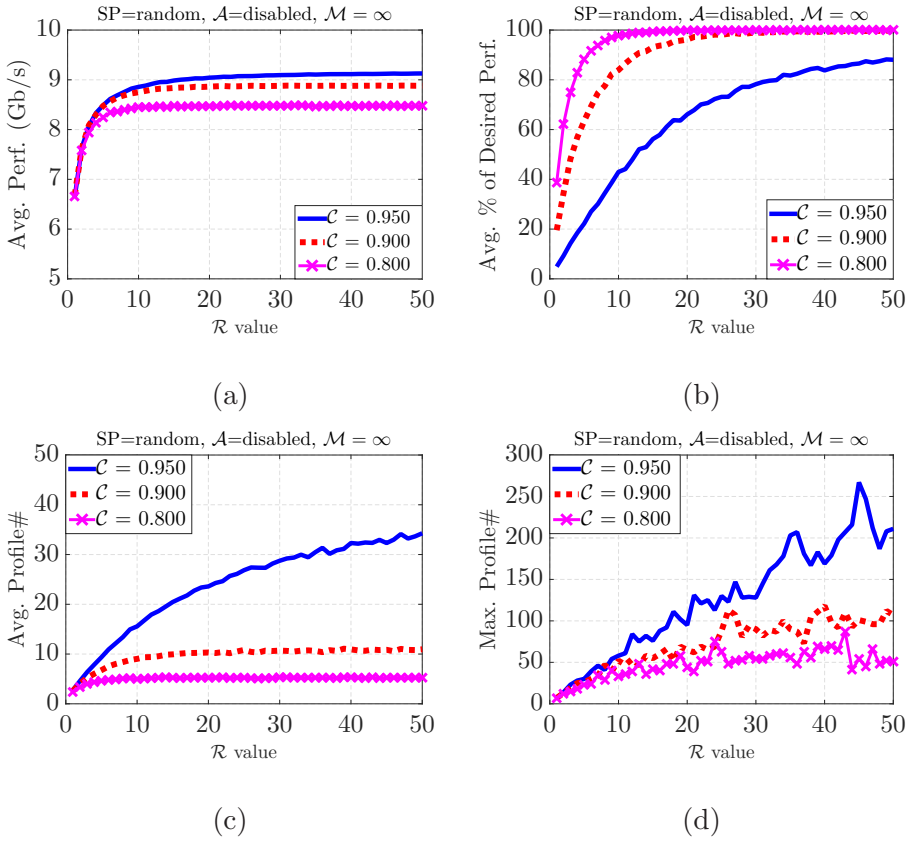


Figure 4.3 Overall performance of `FastProf`.

The actual throughput obtained by `FastProf` is considered as the most important performance metric. Figure 4.3(a) shows that a satisfactory throughput

performance can be achieved as long as a reasonably large $\mathcal{R}$ is specified (e.g., >10). As $\mathcal{R}$ increases, the average throughput performance explored by `FastProf` first increases and then stabilizes at the desired level as specified by the value of $\mathcal{C}$.

We measure and plot the percentage of cases that yield a user-desired performance among all 5,000 runs in Figure 4.3(b). As $\mathcal{R}$ increases, the percentage significantly increases up to 100% for $\mathcal{C} \in \{0.80, 0.90\}$. Although the percentage does not reach 100% as $\mathcal{R}$ approaches 50 when $\mathcal{C} = 0.95$, we still achieve slightly higher performance near or above 9.0 Gb/s as $\mathcal{R}$ increases, as shown in Figure 4.3(a).

We measure the profiling speed by calculating the average number of one-time profilings conducted by the profiler among all 5,000 runs. As expected, this average number generally increases as $\mathcal{R}$ increases, as shown in Figure 4.3(c). For a relatively smaller $\mathcal{C}$ value (e.g., 0.80 and 0.90 in our test cases), it does not always increase as $\mathcal{R}$ increases since the percentage of yielding a desired performance reaches 100% quickly and `FastProf` terminates the profiling process without consuming more profiling time; while for a larger $\mathcal{C}$ value (e.g., 0.95 in our test cases), it takes longer to obtain a higher percentage for a desired performance. As $\mathcal{R}$ increases, there is a higher probability to obtain a desired performance by conducting more one-time profilings, but the actual average throughput performance does not increase as significantly as the percentage of obtaining desired performance does. This observation implies that an appropriate $\mathcal{R}$ is needed: a larger value may lead to a longer profiling process without perceivable performance improvement.

We measure and plot the maximum number of one-time profilings among all 5,000 runs in Figure 4.3(d), which reflects the longest profiling time that `FastProf` may take. Since the number of one-time profilings is not limited ($\mathcal{M} = \infty$), the profiling process terminates only when either a desired performance is achieved or the number of consecutive iterations without performance improvement reaches $\mathcal{R}$. As $\mathcal{R}$ increases, there is more space to be explored by `FastProf` for better performance,

and the longest profiling time among all runs may either stop increasing (for smaller $\mathcal{C}$) or keep increasing (for larger $\mathcal{C}$).

Effects of $\mathcal{M}$ As shown in Figure 4.3(d), as $\mathcal{R}$ increases, the maximum number of one-time profilings resulted from a larger $\mathcal{C} = 0.95$ is up to 250. Even if a one-time profiling takes only 2 minutes, the profiler would take at most 10 hours to obtain a desired performance with a relatively high probability. In practice, due to the complex system and network dynamics, the profiler may run much longer without getting a desired performance. We avoid this situation by setting the upper bound of the total number of one-time profilings ($\mathcal{M}$) to be a finite value.

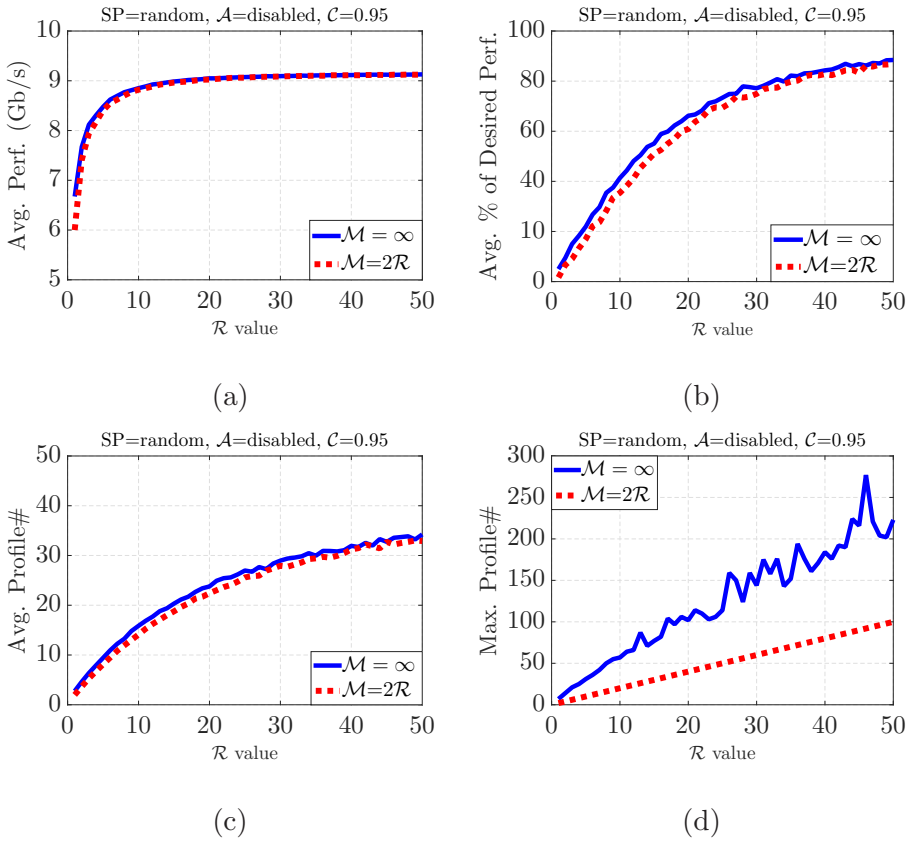


Figure 4.4 Effects of $\mathcal{M}$ on profiling performance.

The results in Figure 4.4 are based on $\mathcal{C} = 0.95$, the upper bound $\mathcal{M} = 2\mathcal{R}$, and the same other parameters as those in Figure 4.3. Setting such an upper bound

provides a guarantee on the profiling time without significantly affecting the profiling performance. Figure 4.4(c) shows that the average number of profilings is slightly reduced with $\mathcal{M} = 2\mathcal{R}$, and Figure 4.4(d) shows the maximal number of one-time profilings among all 5,000 runs is limited by the finite $\mathcal{M}$. Restricting the total number of one-time profilings does not perceivably affect the average profiling performance, as shown in Figures 4.4(a) and 4.4(b). As $\mathcal{R}$ increases, the actual average performance increases and stabilizes between 8.0 Gb/s and 9.5 Gb/s, which is considered to be a satisfactory performance over a 10 Gb/s connection. This observation implies that for one specific run of `FastProf`, if a desired performance could not be achieved in a reasonable amount of time, simply extending the profiling process may not improve the performance. Therefore, this upper bound $\mathcal{M}$ should be set for `FastProf` based on the expectation of the tolerable amount of profiling time. In the experiments in Section 4.6, we set this value to be $\mathcal{M} = 2\mathcal{R}$. Note that since the user-desired performance tends to be achieved before the number of profilings reaches or exceeds $\mathcal{M}$ when $\mathcal{C}$ is relatively small (e.g., 0.80), we choose a relatively larger $\mathcal{C} = 0.95$ and plot in Figure 4.4 the effects of $\mathcal{M}$ on the profiling performance. The measurements in Figure 4.4 suggest that setting $\mathcal{M}$ to be a finite value is more useful and even critical for an aggressive $\mathcal{C}$.

Effects of $\mathcal{A}$ We plot in Figure 4.5 the comparisons between the cases where $\mathcal{A}$ is disabled and enabled. When $\mathcal{A}$ is enabled in Figure 4.5, the profiling range for buffer size is from $f_{\min} = 475$ MB to $f_{\max} = 1,024$ MB since a delay of 380 ms over a connection of 10 Gb/s bandwidth yields a BDP of 475 MB. Since the ranges of iterative variables l' and f' are both from 1.0 to 25.0, the buffer size range mapped to the range of iterative variable f' is actually $[475, 1024]$, i.e., $[475, 1024]$ is linearly mapped to $[1.0, 25.0]$. We plot the effects of cutting the search space into nearly half with larger buffer sizes in Figure 4.5. Firstly, for $\mathcal{R} < 10$, the average performance

when $\mathcal{A}$ is enabled is slightly higher than that when $\mathcal{A}$ is disabled, see Figure 4.5(a). This is because the number of profilings is mainly limited by the value of $\mathcal{R}$ when it is relatively small. The profiling process terminates before `FastProf` further explores the space for improvement, and thus the performance mainly depends on the initial starting point. As $\mathcal{R}$ increases, the differences made by $\mathcal{A}$ become less obvious, and the average performance converges and stabilizes around 9 Gb/s. Secondly, when $\mathcal{C}$ is larger and hard to achieve (e.g., $\mathcal{C} = 0.95$), enabling $\mathcal{A}$ results in a slightly lower percentage of obtaining desired performance and a slightly longer profiling time in comparison with the case where $\mathcal{A}$ is disabled, see Figures 4.5(b) and 4.5(c). Figure 4.5(d) shows that enabling $\mathcal{A}$ does not make significant difference on the longest profiling time. Note that in Figure 4.5(d) we set $\mathcal{M} = \infty$ rather than $\mathcal{M} = 2\mathcal{R}$ to eliminate the effects of $\mathcal{M}$ and show only the effects of $\mathcal{A}$ on the profiling time.

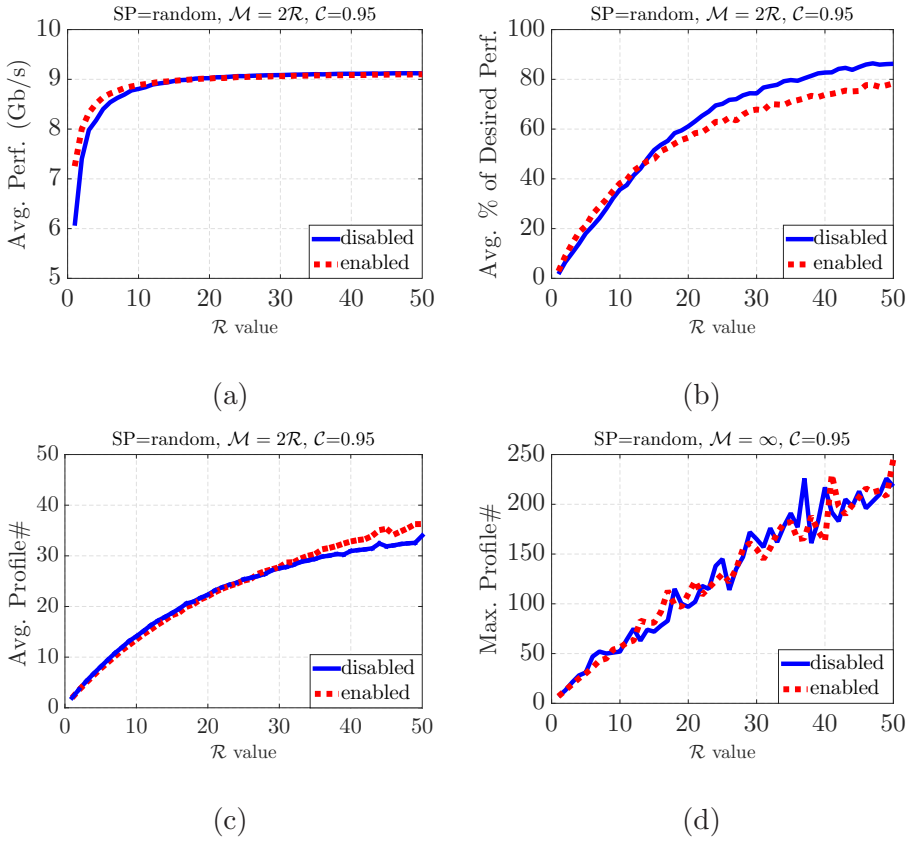


Figure 4.5 Effects of $\mathcal{A}$ on profiling performance.

Effects of Starting Point (SP) To study the effects of the SP, we fix the starting point of the profiling at the "left-bottom" $(l_1, f_1) = (1, 1)$, i.e., one payload size for block size and 1 MB for buffer size (see Equation 4.12 and Figure 4.2) and re-run the emulations using the same other parameter settings as those in Figures 4.3, 4.4, and 4.5, and compare the results with those with randomly selected SPs in Figure 4.6. We observe that randomly selected SP produces: i) either slightly higher (when $\mathcal{R}$ is small) or the same (when $\mathcal{R}$ is large) average throughput performance, as shown in Figure 4.6(a); ii) a consistently higher average probability (percentage) of obtaining a user-desired performance, as shown in Figure 4.6(b); iii) a consistently shorter average profiling time, as shown in Figure 4.6(c); and iv) a more stable status in the worst case, as shown in Figure 4.6(d).

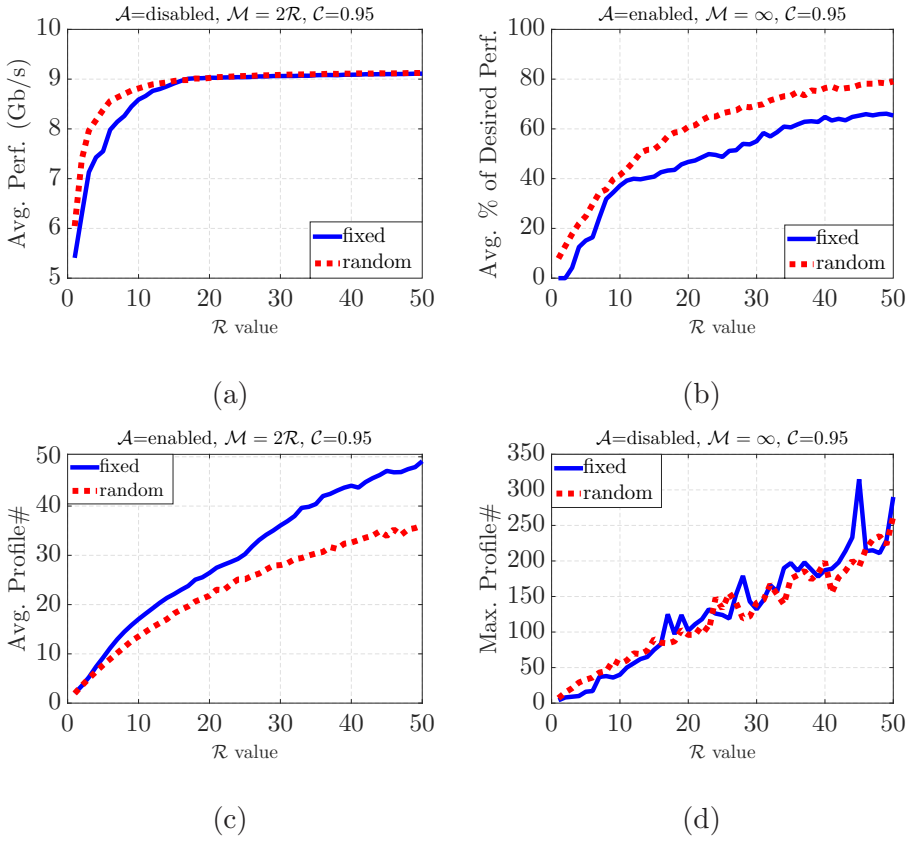


Figure 4.6 Effects of starting point (SP) on profiling performance.

4.5.4 Trace of the Profiling Process

To show the detailed profiling process of `FastProf`, we set $\mathcal{R} = 35$ and $\mathcal{C} = 0.95$, disable $\mathcal{A}$, and then keep track of each pair of parameter values (l, f) as the profiling process progresses. We plot a trace of control parameter values profiled by `FastProf` in Figure 4.7, where we use (green) circles to indicate the values of $\hat{\theta}_k$, use triangles to indicate the perturbations $\hat{\theta}_k \pm \Delta_k$, i.e., the values used by `FastProf` to measure y^+ and y^-, and use red circles (i.e., the last one on the path) to indicate the control parameter values that produce a desired performance, i.e., the ones in the "acceptable area" (user-desired) as shown in Figure 4.2. In this tracing experiment, we set $a = 25$ and $c = 9.5$, and fix the SP at $(1, 1)$. Figure 4.7 shows that `FastProf` is able to explore a path in the search space from the fixed starting point to an acceptable area.

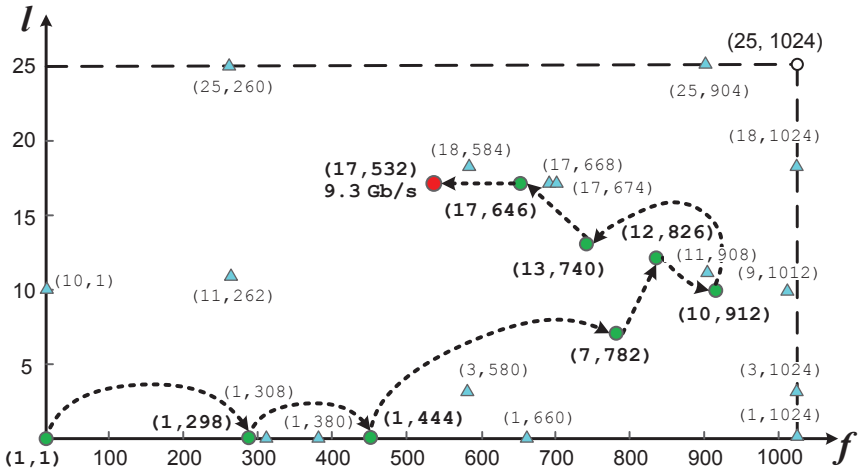


Figure 4.7 Profiling trace with $\mathcal{R} = 35$, $\mathcal{C} = 0.95$, and disabled $\mathcal{A}$.

4.5.5 Comparison with Other Search Algorithms

We conduct profiling emulations using two existing heuristics, random walk [61] and Tabu search [44], and compare their performances with `FastProf`. We measure the same four performance metrics as in Section 4.5.3 and plot the results in Figure 4.8. In comparison with random walk and Tabu search, `FastProf` consistently

produces significantly better performance (Figure 4.8(a)), has a higher probability of obtaining a user-desired performance (Figure 4.8(b)), takes much less profiling time (Figure 4.8(c)), and has much better worst cases (Figure 4.8(d)) among all 5,000 runs.

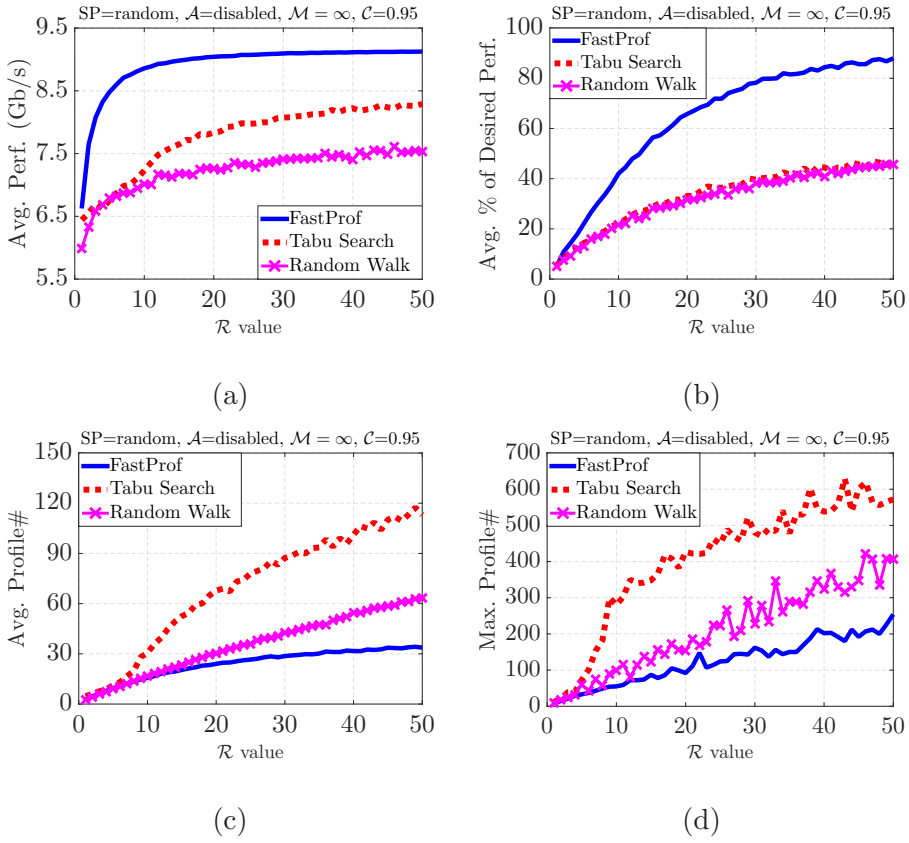


Figure 4.8 Profiling comparison among `FastProf`, random walk, and Tabu search.

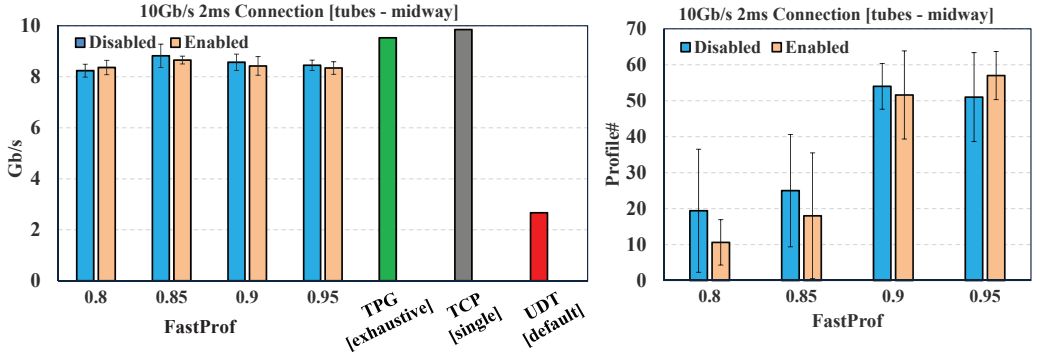
4.6 Experiment-based Performance Evaluation

We implement `FastProf` based on TPG and conduct experiments over two physical connections with 2 ms and 380 ms RTTs in real-life network environments.

4.6.1 Experimental Results on ANL Testbed

We run `FastProf` over 10 Gb/s physical connections from ANL to University of Chicago with 2 ms and 380 ms delays. In these experiments, we set y^* to be the link

capacity (i.e., $y^* = 10\,\text{Gb/s}$), $a = 30.0$, $c = 9.5$, $\mathcal{R} = 30$, and $\mathcal{M} = 60$, and collect experimental results in response to various profiling precision restrictions on buffer size including 2.0 MB, 1.0 MB, 512 KB, and 1 Byte (i.e., without precision restriction)[2].

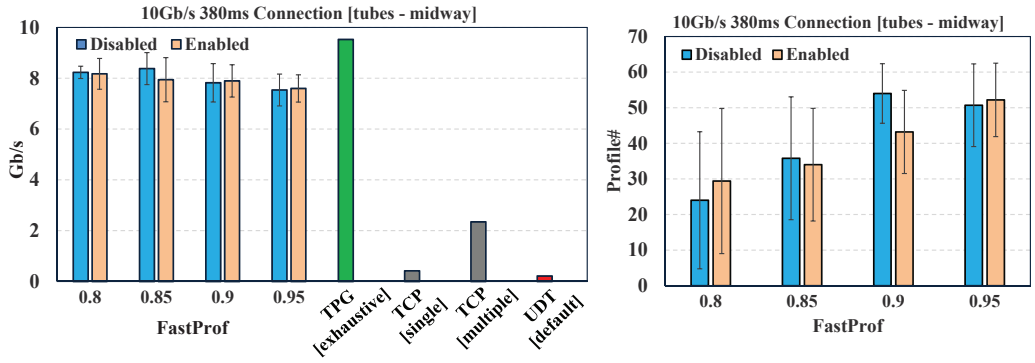


(a) (b)

Figure 4.9 Experimental results of `FastProf` on a 10 Gb/s 2 ms physical connection with 1 Byte buffer resolution. **(a)** performance comparison of `FastProf`-tubed UDT, TPG-tubed UDT, single/multiple stream(s) TCP, and default UDT; **(b)** profiling time of `FastProf`.

We run each test for 10 times, measure the average performance and average profiling time (as indicated by the number of profilings) together with their standard deviations, and plot the experimental results with different profiling precisions on buffer size in Figure 4.9 (2 ms, 1 Byte), Figure 4.10 (380 ms, 2.0 MB), Figure 4.11 (380 ms, 1.0 MB), Figure 4.12 (380 ms, 512 KB), and Figure 4.13 (380 ms, 1 Byte). In each comparison, in addition to the average throughput achieved by `FastProf`-tuned UDT, we also include: i) the maximal throughput achieved by single stream TCP; ii) the maximal throughput achieved by multiple streams TCP; iii) the maximal throughput achieved by TPG-tuned UDT using exhaustive search; and iv) the performance achieved by default UDT. The results show that `FastProf` is able

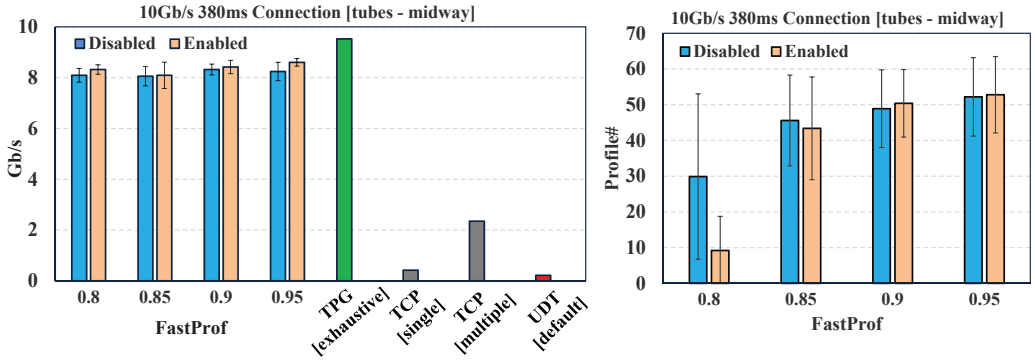
[2]We perform rounding operations to ensure that an integer number of bytes are set for both the block size and the buffer size.



(a) (b)

Figure 4.11 Experimental results of `FastProf` on a 10 Gb/s 380 ms physical connection with 1.0 MB buffer resolution. **(a)** performance comparison of `FastProf`-tuned UDT, TPG-tuned UDT, single/multiple stream(s) TCP, and default UDT; **(b)** profiling time of `FastProf`.

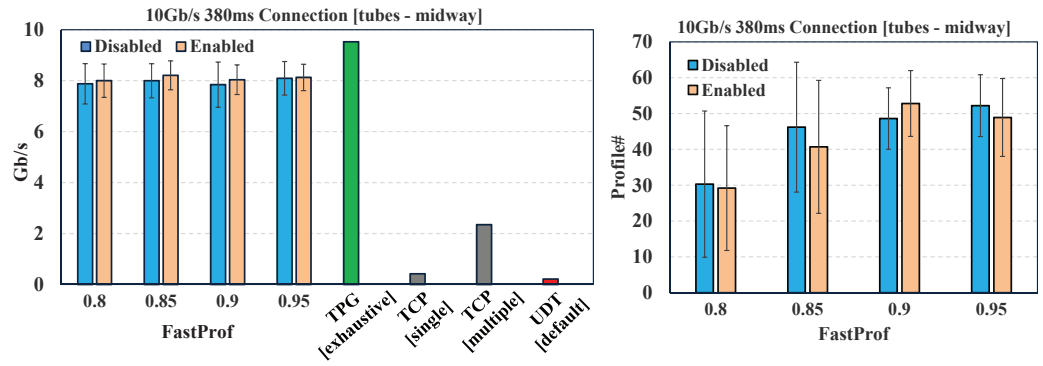
to consistently find a set of control parameter values that produce a satisfactory throughput in a short period for both short (2 ms) and long (380 ms) RTT delays.



(a) (b)

Figure 4.10 Experimental results of `FastProf` on a 10 Gb/s 380 ms physical connection with 2.0 MB buffer resolution. **(a)** performance comparison of `FastProf`-tubed UDT, TPG-tuned UDT, single/multiple stream(s) TCP, and default UDT; **(b)** profiling time of `FastProf`.

For a RTT of 2 ms, TCP is better as it can achieve the link speed performance using a single stream. Tuned by TPG using exhaustive search, UDT produces a

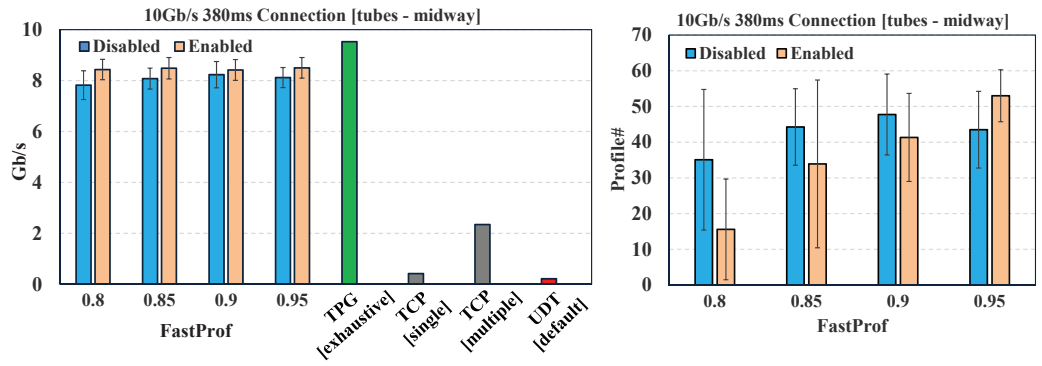


(a) (b)

Figure 4.12 Experimental results of `FastProf` on a 10 Gb/s 380 ms physical connection with 512 KB buffer resolution. **(a)** performance comparison of `FastProf`-tuned UDT, TPG-tuned UDT, single/multiple stream(s) TCP, and default UDT; **(b)** profiling time of `FastProf`.

slightly lower performance than TCP, despite of which Figure 4.9 still shows the effectiveness of `FastProf` in reducing the profiling overhead as well as achieving a comparable level of performance. The performance explored by `FastProf` is quite stable (> 8.0 Gb/s): when $\mathcal{C} \in \{0.80, 0.85\}$, `FastProf` achieves a user-desired performance in all 10 runs with less than 30 one-time profilings on average; when $\mathcal{C} \in \{0.90, 0.95\}$, although the desired performance can only be occasionally achieved, the actual performance produced by `FastProf`-tuned UDT is consistently higher than 8.0 Gb/s, see Figure 4.9(a).

For a long RTT of 380 ms, carefully tuned UDT is obvious a better choice to conduct the data transfer. TPG-tuned (using exhaustive search) UDT gives us the highest performance one can possibly expects, thus we use it as the comparison base. As shown in Figures 4.10, 4.11, 4.12, and 4.13, no matter with or without profiling precision restrictions on buffer size, `FastProf` is able to discover an appropriate set of values for block size and buffer size that result in an average performance between 7.5 Gb/s and 8.5 Gb/s. The performance achieved by `FastProf` is comparable with the results achieved by the exhaustive search-based approach that we manually



(a) (b)

Figure 4.13 Experimental results of `FastProf` on a 10 Gb/s 380 ms physical connection with 1 Byte buffer resolution. **(a)** performance comparison of `FastProf`-tuned UDT, TPG-tuned UDT, single/multiple stream(s) TCP, and default UDT; **(b)** profiling time of `FastProf`.

conducted for emulations in Section 4.5, while at the same time the profiling time is significantly reduced from 18 days to several hours at most. On the other hand, the performance achieved by single stream TCP, multiple streams TCP, and default UDT are all far from satisfactory.

In above experiments, we observe that enabling the $\mathcal{A}$ rule does not necessarily improve the performance, which implies that `FastProf` is not sensitive to the size of the parameter search space. In particular, when the buffer size precision restriction is removed, for a longer RTT, enabling $\mathcal{A}$ improves the performance in terms of both average performance (Figure 4.13(a)) and average profiling time (Figure 4.13(b)) when $\mathcal{C}$ is set to be a relatively conservative value (e.g. 0.80, 0.85, and 0.90); when $\mathcal{C}$ is set to be an aggressive value (e.g. 0.95), enabling $\mathcal{A}$ improves the average performance but meanwhile consuming more profiling time (Figure 4.13(b)).

CHAPTER 5

TRANSPORT PROTOCOL DESIGN

5.1 Introduction

Sustaining a high end-to-end data transfer performance over dedicated channels in High-performance Networks (HPNs) by using TCP-like transport protocols requires physical media with an extremely low loss rate (e.g., 10^{-11}) that is not available in today's hardware [41,42,81]. Due to the performance limitation of the standard TCP, many TCP enhancements have been proposed (see Section 5.6), which employ various methods for the control of parameter increasing and deceasing in TCP's Additive Increase Multiplicative Decrease (AIMD) algorithm (e.g., Scalable TCP [58]). UDP-based protocols also have been proposed and implemented typically with non-AIMD algorithms. Some of them such as UDT [49] consider fairness as an important issue in their design, which is not desirable for transport control over dedicated channels; and others such as RBUDP [53] make an assumption that the receiver end host is not the bottleneck, which oftentimes is not the case over high-speed dedicated connections.

In this Chapter, through analysis of the data packet receiving process on Linux platform and performance modeling of the Tsunami UDP protocol [22,68], we propose to use an approach based on adaptive rate and error threshold control to improve the performance of big data transfer over high-speed dedicated connections.

5.2 Problem Statement

The transport control in HPNs is quite different from that in traditional shared-IP networks such as Internet. Typically, the processing speed of end host cannot keep up with the connection bandwidth reserved in advance in HPNs, and the bottleneck of

data transfer as observed in the traditional Internet is shifted from network segments to the end-hosts. In addition, even if the processing power of the end host is enough to keep up with the connection speed, the end-to-end data transfer performance is still limited by the specific transport protocol that is being used at end hosts. Transport protocols that work well in traditional shared-IP networks such as TCP may not be appropriate to perform big data transfer tasks over long-hual dedicated connections in HPNs due to its conservative AIMD congestion control algorithm. For example, Floyd shows in [41] that over a 10 Gb/s connection with 100 ms RTT delay, even if there is no packet lost or corrupted, the "standard" TCP (i.e., TCP Reno) needs around one hour to fully utilize the connection capacity. To maintain the connection speed, it requires an extremely low loss rate (10^{-11}) in the network segment, such physical media is currently not available.

A nationwide or worldwide dedicated connection typically could be reserved and established in advance by certain agents in HPNs, e.g., OSCARS [7] in ESnet [6] and ION [14] in Internet2 [13]. Different from the shared network environment, once a dedicated connection is established in HPNs, it is exclusively allocated to the stakeholder end users. The ultimate goal of the end users is then to move their data as quickly as possible without considering fairness and friendliness. Under such conditions, we have the following given and assumptions for the problem of big data transfer over dedicated connections in HPNs:

- A dedicated and "perfectly" reliable connection that typically has a fairly long delay, (e.g., 300 ms or longer) and a quite high bandwidth reserved in advance (e.g., 10 Gb/s or higher). The loss rate of the connection's physical media is quite small and thus could be ignored, and the available bandwidth of the connection is known to be a constant;
- The bottleneck of data transfer exists at the receiver site, where the maximal receiving rate is typically less or equal to the reserved bandwidth and is also time-varying due to the dynamics and background load of the end system. Therefore, to optimize the utilization of the reserved high-speed network connection, the sending rate should neither overwhelm the time-varying capacity of the receiver nor be too conservative;

- The inter- and intra-protocol fairness and TCP friendliness are not concerns since there are not any other data flows competing for the available bandwidth.

Based on such given and assumptions, our goal is to develop a data transfer protocol that has the following features:

- The protocol should maximize the resource (mainly bandwidth) utilization;
- The protocol should be light-weight and cannot bring too much computational control overhead to the end hosts;
- The protocol should not require any modifications of the network or the end host system, and thus can be deployed and used easily.

5.3 Data Packet Receiving Process

We take Linux as an example operating system and review its data packet receiving process. The end-to-end data transfer is a complex process that involves various components. Table 3.1 shows the software and hardware entities together with their parameters that may affect the performance in a typical data transfer using UDP-based protocols such as UDT [49]. Other protocols such as TCP have similar processes and related factors. Any of these components could become the bottleneck and hence limit the end-to-end data transfer performance.

We briefly review the data packet receiving process in each layer followed by an experimental study on the effects of resource utilization at receiver site. Wu *et al.* provide a detailed analysis of packet receiving process of TCP in [88], and here we focus on the packet receiving process of UDP or UDP-based protocols. The main steps of packet receiving process is shown in Figure 5.1 and the arriving packets could be dropped at any step as described in the following.

5.3.1 Link Layer

As shown in Figure 5.1, when the data packets arrive at the receiver site, they are transformed from raw bit signal into datalink frames by Network Interface Card (NIC)

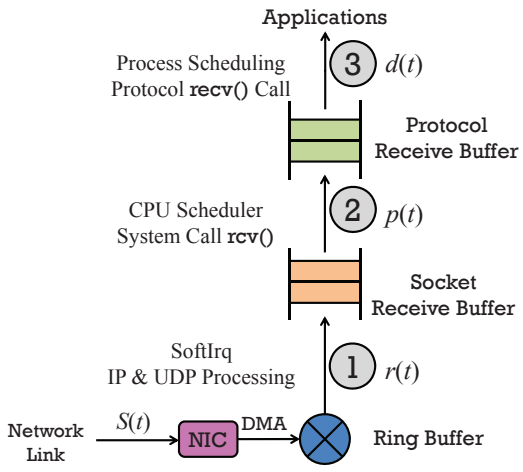


Figure 5.1 Packet receiving process at receiver end host.

and stored in ring buffer by DMA [35]. The ring buffer is maintained and managed by the device driver and comprised of a "ring" of packet descriptors of socket kernel buffers (`sk_buff`), and each of which holds a single data packet with size up to the Maximum Transmission Unit (MTU). After a data packet is written into a `sk_buff`, its packet descriptor will be marked as "used" that needs to be refilled to hold other further incoming packet. A data packet will be discarded if there is no "ready" packet descriptor available when it arrives. Suppose the incoming packet rate is $S(t)$, the ring buffer size is R_{NIC}. To avoid packet loss at the link layer, the ring buffer should be refilled (i.e., `sk_buff` is marked as "ready") as soon as possible to make sure there is `sk_buff` available when a data packet arrives. Thus we have

$$A_{NIC}(t_0) + \int_{t_0}^{t_0+\Delta t} \big(S(t) - f(t)\big)\, dt \leq R_{NIC}, \text{ when } S(t) \geq f(t), \tag{5.1}$$

where $A_{NIC}(t_0)$ is the available NIC buffer size at time point t_0 and $f(t)$ is the refill rate of packet descriptor of the ring buffer. Two major factors that affect the refill speed $f(t)$ are: i) the `sk_buff` consuming rate, and ii) the system memory allocation status [88]. The `sk_buff` consuming rate is actually the transport protocol service rate $r(t)$, or named as receiving rate at transport protocol layer (see Figure 5.1). The

$r(t)$ is directly affected by CPU occupancy status and the corresponding scheduling algorithm. When the receiving process runs out its CPU time slice and new packets arrive when there are no `sk_buff` available; or when the rate that CPU polls data packets from the ring buffer is smaller than the packet arriving rate, i.e., $r(t) < S(t)$, the arrived packets will be dropped unless $S(t)$ is adjusted appropriately. In addition, when the system is in high memory pressure status, memory allocation for new packets tends to fail, which also eventually limits the refill rate $f(t)$.

5.3.2 IP Layer

After a packet is transferred into a `sk_buff` of the ring buffer, it becomes accessible to the Linux kernel. It is the NIC's responsibility to generate an interrupt to let CPU know that the data packet is ready for upper layer processing. CPU handles the interrupt by calling the *interrupt handler* of the device driver and scheduling the corresponding *softirq*. The interrupt handler places a reference to the *device* in the *poll queue* of the interrupted CPU. Afterwards, when CPU serves the *softirq*, it first checks its own *poll queue*, polls each *device* in the queue, and then calls the *poll* method of the device driver to get the received packets from the ring buffer. After a received packet is dequeued from its receive ring buffer for further upper layer processing, its corresponding packet descriptor in the receive ring buffer would be re-initialized and refilled. The IP protocol processing function is called within the service of the *softirq*, which verifies the integrity of the packet, applies the firewall rules, and delivers the packet for forwarding or local delivery to a higher layer protocol.

5.3.3 Transport Layer

When a packet is passed upwards for transport layer processing, a specific handler function will be called, e.g., `tcp_v4_rcv()` for TCP, and `udp_rcv()` for UDP. Since our protocol is designed and developed based on UDP, we will not cover the processing

details of TCP, and detailed modeling and analysis of TCP processing can be found in [88] and other numerous literatures such as [70]. UDP provides the simplest transport services: i) process-to-process communication channel, and ii) per-segment error control. The `udp_rcv()` verifies the integrity of the UDP packet and queues one or more copies for further delivery. When the *receive queue* of the corresponding socket queues the received packets, if there is not enough buffer space in the UDP receive socket buffer, the packets are dropped. The packets stored in the UDP socket receive buffer are ready for delivery to the user space where our protocol is developed.

5.3.4 Effects of Resource Utilization

Based on the above discussion, when the memory is not in high allocation pressure status, the potential bottleneck at the receiver site is essentially caused by the shortage of CPU cycles. Although there are methods and models to predict and estimate the available CPU cycles such as [28], it is difficult to make such estimation and prediction accurate enough for the transport control in high-speed networks, which typically is on the order of milliseconds or microseconds. When the receiver system is heavily loaded, a single misleading transport control decision may cause severe packet loss and bad performance. To show how resource utilization at the receiver affects the data transfer performance, we use SABUL[1] [47] to transfer 100 GB data over a local 10 Gb/s back-to-back connection with different number of background competing processes and measure the corresponding goodput performance.

The main scheduling process of this experiment first runs the data transfer program using SABUL, and once the data transfer starts, it simultaneously starts to run several background competing processes at the receiver site to compete for CPU time. An infinite `for` loop is executed in the competing process until the data receiving process is finished. In each `for` loop, the program first performs some

[1]SABUL is the early version of UDT [45] that has simpler implementation than later ones, which also handle other UDT socket-related issues besides data transfer performance.

CPU-intensive and memory-limited computation, e.g., division, and then sleeps for several milliseconds. We use shared memory for interprocess communication: when the data receiving process is finished, the scheduling process modifies the value of corresponding shared memory. By checking the value of the share memory, the competing process decides to either keep running or terminate. When the data transfer is finished, the competing process calculates the CPU time used by itself, terminates its `for` loop, and then writes the value of consumed CPU time into the corresponding shared memory. This information is read by the main scheduling process for performance measurements.

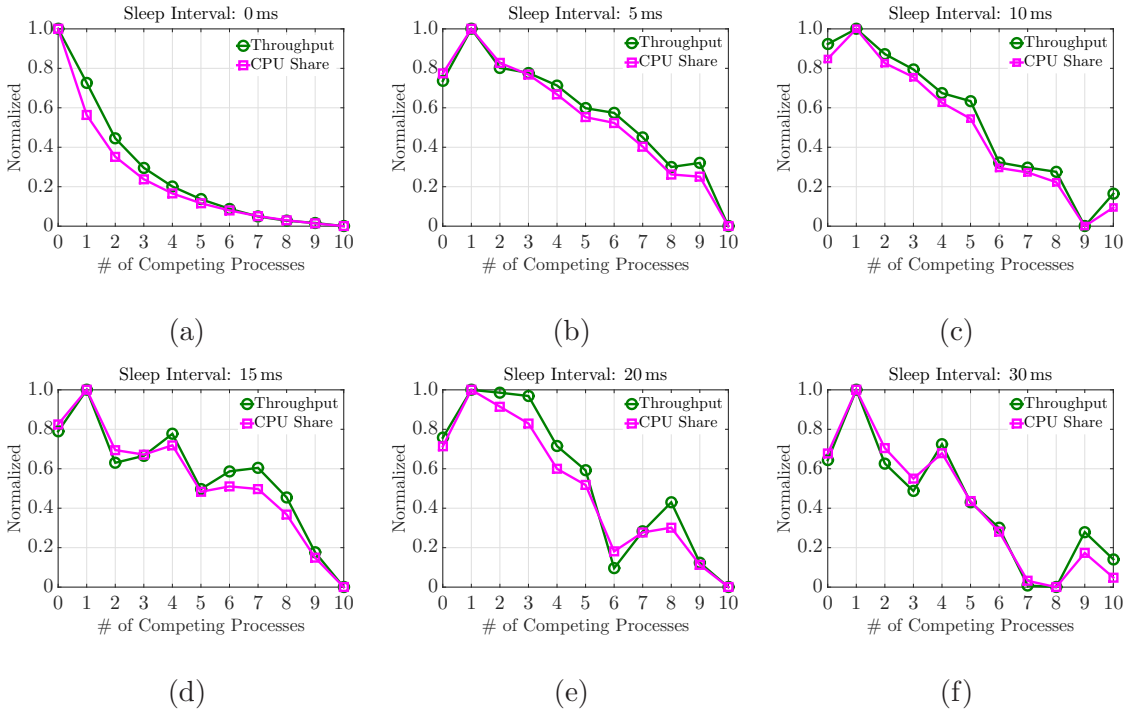


Figure 5.2 Effects of background competing processes.

In this experimental study, we bind all processes and threads on the same CPU core through setting CPU affinities. By varying the number of competing processes and the sleep interval, we measure the goodput performance and the CPU time consumed by the data receiving process, and plot the normalized comparisons between

goodput performance and CPU share percentage in Figure 5.2. The results show that as the number of competing processes increases, the transfer performance and the CPU cycles spent on data receiving process both gradually and consistently decrease. The CPU share percentage (denoted as q) of the data receiving process during time duration Δt is defined as Equation 5.2

$$q = \frac{T_{recv}}{T_{wall}}, \tag{5.2}$$

where T_{recv} is the CPU time spent on the data receiving process during Δt and T_{wall} is the wall-clock time, i.e., $T_{wall} = \Delta t$.

5.4 Technical Approach

In this section, we present out technical approach to improve the data transfer performance over dedicated connections based on the Tsunami UDP protocol [22]. Please refer to Table 5.1 for notations used in the performance modeling and analysis.

Table 5.1 Notations Used in the Protocol Design

Notations	Definitions
B	Bandwidth of the dedicated connection
ϵ_i	Error rate of the ith interval
S_i	Sending rate in the ith interval
$S(t)$	Sending rate at time point t
S_{max}	Target sending rate
ρ	Percentage of historical data used in error rate calculation
L_i	Number of blocks retransmitted in the ith interval
N_i	Number of original blocks transmitted in the ith interval
θ_i	Retransmission percentage of the ith interval
R_{used}	Number of used data slots in the ring buffer
R	Capacity of the ring buffer
λ_i	Occupancy rate of the ring buffer in the ith interval
ξ_i	Error rate threshold in the ith interval

5.4.1 Overview

Two major components are necessary for a data transfer protocol over dedicated connections: i) transmission rate control, which intends to avoid either overwhelming the available capability of network and end host or wasting them; ii) acknowledgement control, which is mainly concerned with the reliability of the transmission and is also potentially concerned with more accurate and effective rate control.

The *de facto* standard transport protocol TCP and its variants such as Scalable TCP [58] employ window-based Additive Increase Multiplicative Decrease (AIMD) approaches to control the packet sending speed, which has been shown to be quite successful within the shared Internet where friendliness and fairness are both critical criterions. Such AIMD-based rate control treats a single packet loss as the direct indicator of congestion along the entire transfer path, and conducts significant back off to avoid further loss and to maintain fairness, which is too conservative to effectively utilize the resources of dedicated connections that have been reserved in advance. On one hand, to optimize the resource utilization of dedicated connections, a data transfer method should be as aggressive as possible for increasing the sending speed to occupy the bandwidth as much as possible, since fairness and friendliness are not the concerns; on the other hand, the sending rate cannot be so aggressive to overwhelm the end host (especially the receiver) and waste the resources, where many data packets have arrived at the receiver got lost due to their overwhelmed arriving speed. As verified by the transport profiles in [85], the optimal transport performance is typically obtained when a small packet loss does exist.

As pointed out by [37], the decision making of rate control in the Internet is difficult since the environments are complex. This is also true for the case of data transfer over dedicated connections, because although the bandwidth is exclusively allocated from the user's perspective, the underlying physical links still essentially operate on the hardware with diverse configurations. In addition, the reserved

bandwidth might be realized based on various technologies that may introduce significantly different levels of dynamics, randomness, and jitters. Assuming that the receiver system is the source of data transfer bottleneck, the sending rate should be adjusted based on the information collected at the receiver site and delivered to the sender site. This decision making mechanism in TCP is based on the occurrence of the packet loss event without considering where it happens. UDP-based protocols such as UDT [49] use a similar control structure with TCP but employs a Decreasing AIMD (DAIMD) algorithm [50] to increase the sending rate more aggressively based on the positive acknowledgements and decrease the sending rate more conservatively based on negative acknowledgements, both by tuning the inter-packet delay (IPD).

Over high-speed dedicated connections, the packet loss typically happens at the end hosts rather than the network segments and thus is not a good indicator of congestion. A packet loss event itself is not enough for the sender to cut its sending rate by half or other significant amount to avoid further congestion and to gain long-term good performance. A single packet loss is not enough for the adjustment calculation of sending rate either, because it could be caused by various components along the entire data transfer path and not all of them are correlated to the severe congestions, which truly require for backing off. For example,

- The sending rate may be too high and makes the packets arrive at a speed that the receiver can not handle due to its computing power limitation. In such case, the sender should back off and reduce its sending rate;
- The packet loss may be caused by the specific CPU scheduling algorithm at receiver site, i.e., when the data receiving process runs out its CPU time, the data packets statistically happen to overflow the NIC or kernel buffer that are not polled or drained out in time. In such case, it is enough for the sender to back off just a little bit;
- The packet loss may be caused by the limitation of the physical connection or the end host hardware error. Since this is a non-congestion packet loss, the sender should not back off but just simply retransmit the lost packet and maintain or even increase its sending rate in order to gain high utilization.

There exists two similar UDP-based file transfer protocols, RBUDP [53] and Tsunami [22, 68], which have shown promising performance over dedicated connections. Both of them use a similar architecture that blasts or dumps data blocks containing user payloads through UDP channel at a fixed target rate and retransmits the lost ones after receiving error reports (acknowledgements) through TCP channel, and then repeats this process until the entire file is delivered.

We first model the data transfer process of Tsunami and analyze its goodput performance, and then following a similar strategy with Tsunami, we propose to use the following approaches to improve the data transfer performance over dedicated connections: i) similar to Tsunami, we directly control the sending rate by tuning the inter-packet delay (IPD) rather than adjusting the window size, ii) unlike Tsunami, we simply set the target rate to be the reserved bandwidth and let the protocol itself figure out the optimal sending rate; iii) similar to Tsunami, we adjust the sending rate according to the error rate that is calculated based on loss rate and buffer occupancy; and iv) unlike Tsunami, we adaptively change the error rate threshold for the sending rate control to eliminate unnecessary packet loss. We conduct data transfer experiments over 10 Gb/s connections with various RTT delays emulated by `netem` [17] and present preliminary results in comparison with UDT.

5.4.2 Architecture of the Tsunami Protocol

As shown in Figure 5.3, both sender and receiver of Tsunami maintain a ring buffer to hold sent or received data blocks that are un-acknowledged. Datagrams (or data blocks as named in Tsunami [68]) are sent and received through a UDP data channel. The control messages such as control parameter values, notifications of start and stop, are all sent and received through a TCP channel. An independent thread is in charge of reading/writing data blocks from/into the disk. We modify the source code of

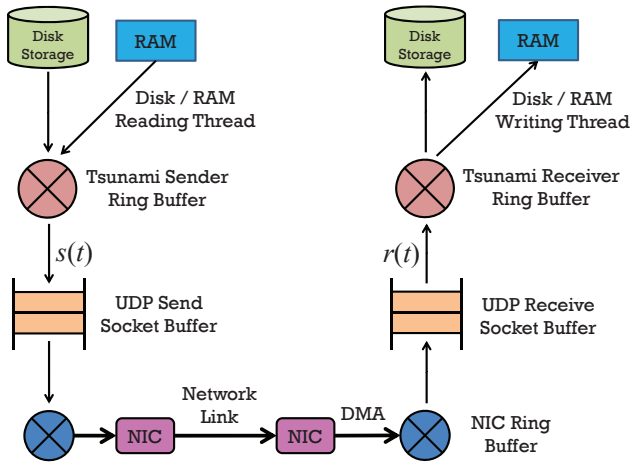


Figure 5.3 Data transfer process of the Tsunami protocol.

Tsunami and let the thread read/write data blocks from/into memory rather than disk to test the memory-to-memory transfer performance of the protocol.

5.4.3 Rate Control and Flow Control of the Tsunami Protocol

In Tsunami, the sender sends data blocks (i.e., UDP datagrams) at a user-specified target rate, and the receiver sends an acknowledgement back to the sender when every 50 data blocks are received. An acknowledgement includes a list of sequence numbers of lost blocks and an error rate of the current iteration. The error rate in the ith interval is calculated by

$$\epsilon_i = \rho \cdot \epsilon_{i-1} + (1 - \rho) \cdot (\theta_i + \lambda_i), \tag{5.3}$$

where θ_i and λ_i are the retransmission rate of data blocks and the occupancy of the ring buffer in the ith iteration, respectively. The retransmission rate is given by

$$\theta_i = \frac{L_i}{L_i + K}, \tag{5.4}$$

where L_i is the number of lost data blocks in the ith interval, which could be identified based on the block sequence numbers; and K is the number of received data blocks

when an error report is triggered, which is user-specified. The occupancy rate of the ring buffer is calculated by

$$\lambda_i = \frac{\mathcal{R}_{used}}{\mathcal{R}}, \tag{5.5}$$

where $\mathcal{R}_{used}$ is the number of used data slots in the ring buffer when λ_i is being checked; and $\mathcal{R}$ is the maximum number of blocks in ring buffer (i.e., the capacity of the ring buffer) and its default value specified by Tsunami is 4096, which can be changed by users. In each interval, if the number of lost blocks exceeds the half of the ring buffer capacity, Tsunami restarts from the first missing block. From a user perspective, it is tolerable to allow the protocol to use a relatively large memory space to achieve the optimal data transfer performance, and to simplify our analysis, we assume that the ring buffer is large enough to prevent any restarts.

Tsunami starts sending data blocks at an initial target rate S_1 and adjusts its fixed sending rate for the next interval when an error report (i.e., an acknowledgement) is received. If the error rate ϵ_{i-1} is above the threshold ξ, Tsunami decreases its fixed sending rate using factor β, otherwise Tsunami increases its sending rate using factor α unless the target rate S_{max} is already reached, i.e.,

$$S_i = \begin{cases} min\left\{\alpha \cdot S_{i-1},\, S_{max}\right\}, & \epsilon_{i-1} \leq \xi \\ \beta \cdot S_{i-1}, & \epsilon_{i-1} > \xi \end{cases} \tag{5.6}$$

As indicated by Equation 5.6, the sending rate control in Tsunami and the corresponding resulted goodput performance highly depend on the error rate ϵ and threshold ξ. The error rate is mainly decided by the loss rate and the buffer occupancy, and the threshold is user-specified and directly affects the adjustments of sending rate.

5.4.4 Performance Analysis of the Tsunami Protocol

In Tsunami, the target sending rate S_{max} is user-specified and serves as an upper bound of the sending rate adjustment, and the acknowledgement (i.e., the error

report) is interval-based and sent by the Tsunami receiver every time when it receives a certain number of data blocks.

Interval	Send Rate	# of Received Block	# of Lost Blocks
1	S_1	K	L_1
2	S_2	K	L_2
3	S_3	K	L_3
⋮	⋮	⋮	⋮
n	S_n	K	L_n
n+1	S_{n+1}	L_n	0

Figure 5.4 Statistics of data transfer using the Tsunami protocol.

As illustrated by Figure 5.4, suppose there are K data blocks received before the first error report is received, and during which time duration T_1 (i.e., in the first interval) there are L_1 blocks get lost, we can estimate T_1 as

$$T_1 = \frac{K + L_1}{S_1}. \tag{5.7}$$

Similarly, suppose there are K blocks received at a fixed rate S_i and L_i blocks get lost in the ith interval. If the $(n+1)$th interval is the last one, then there are total $L_n \leq K$ blocks received (i.e., the lost ones in the previous interval) at a fixed rate S_{n+1} and there are not data blocks get lost (otherwise it would not be the last interval). We then have the total transmitted number of data blocks

$$n \cdot K + L_n = F + \sum_{i=1}^{n} L_i, \tag{5.8}$$

where F is the total number of data blocks of the user payload.

This entire process to delivery the user payload F totally takes time

$$T = \sum_{i=1}^{n+1} T_i = \sum_{i=1}^{n} \frac{K + L_i}{S_i} + \frac{L_n}{S_{n+1}}, \tag{5.9}$$

and results in a goodput performance

$$G = \frac{F}{T} = \frac{n \cdot K - \sum_{i=1}^{n-1} L_i}{\sum_{i=1}^{n} \frac{K + L_i}{S_i} + \frac{L_n}{S_{n+1}}}. \tag{5.10}$$

Suppose that the sending rate at the equilibrium status is S^*, the corresponding number of lost blocks is L^* and the corresponding loss rate is θ^*, Equation 5.10 could be simplified and approximated as follows

$$G = \frac{n \cdot K - n \cdot L^* + L^*}{n \cdot \left(\frac{K + L^*}{S^*}\right) + \frac{L_n}{S_{n+1}}} \approx S^* \cdot \frac{K - L^*}{K + L^*}, \tag{5.11}$$

in which the approximation is valid given the fact that data size of user payload F is "big", i.e., n is "big", and thus we could eliminate the performance statistics in the last interval since $L^* \ll F \approx n \cdot K - n \cdot L^*$ and $\frac{L_n}{S_{n+1}} \ll n \cdot \left(\frac{K + L^*}{S^*}\right)$.

By solving the loss rate equation in Equation 5.4, we have

$$\theta^* = \frac{L^*}{K + L^*} \Rightarrow L^* = \frac{K \cdot \theta^*}{1 - \theta^*}, \tag{5.12}$$

and then the goodput G in Equation 5.10 could be further simplified as

$$G = S^* \cdot \frac{K - L^*}{K + L^*} = S^* \cdot \frac{K - \frac{K \cdot \theta^*}{1 - \theta^*}}{K + \frac{K \cdot \theta^*}{1 - \theta^*}} = S^* \cdot (1 - 2\theta^*). \tag{5.13}$$

We consider the following three scenarios of the big data transfer over dedicated connections and discuss their corresponding performance.

Low Target Rate If the target rate is low $S_{max} = S' \ll B$, then packet losses are mainly caused by physical media error and statistically bad luck of the packet being processed at the end host. Such loss rate is quite marginal and could be ignored, i.e., $\theta' \approx 0$. Assuming the ring buffer capacity is large enough and then the error rate

is consistently below threshold and does not trigger any sending rate decreasing, the sending rate would stay near or at the target rate and the transfer performance is almost the same with the sending rate, i.e.,

$$G = S' \cdot (1 - 2\theta') \approx S',\ S_{max} = S' \ll B. \tag{5.14}$$

Moderate Target Rate If the target rate is moderately high, i.e., $S_{max} = S'' < B$ and $S' \leq S''$, then the packet losses may occasionally be caused by the pressure of processing the quickly arriving packets at the receiver site. The loss rate in such case is noticeable and could not be ignored, i.e., $0 \approx \theta' < \theta'' < 1$. If the loss rate together with the ring buffer occupancy do not exceed the threshold ξ, we could analytically derive the transfer performance as

$$G = S'' \cdot (1 - 2\theta''),\ S' < S_{max} = S'' < B. \tag{5.15}$$

By combining the cases in Equation 5.14 and Equation 5.15, we have the following partial guidelines for sending rate adjustment,

$$\begin{cases} \dfrac{S'}{S'' \cdot (1 - 2\theta'')} \geq 1 \Rightarrow \dfrac{S'}{S''} \geq 1 - 2\theta'' \\ \dfrac{S'}{S'' \cdot (1 - 2\theta'')} < 1 \Rightarrow \dfrac{S'}{S''} < 1 - 2\theta'' \end{cases}. \tag{5.16}$$

High Target Rate If the target rate is aggressively high, i.e., $S_{max} = S''' \approx B$ and $S'' < S'''$, the loss rate is significant and may cause rate adjustment since it exceeds the error rate threshold ξ. As shown in Equation 5.13, we use the equilibrium status to approximate the transfer performance as

$$G = S^* \cdot (1 - 2\theta^*).$$

Over a high-speed dedicated connection, the retransmission rate or loss rate θ_i is mainly decided by the sending rate S_i, which in turn decides the error rate ϵ_i.

Note that θ_i is dominating in the calculation of ϵ_i when the buffer occupancy is not a concern (i.e., when the ring buffer is large enough). In such case, we could reasonably assume that $\theta_i \leq \theta_j \Rightarrow \epsilon_i \leq \epsilon_j$ and $\theta_i > \theta_j \Rightarrow \epsilon_i > \epsilon_j$.

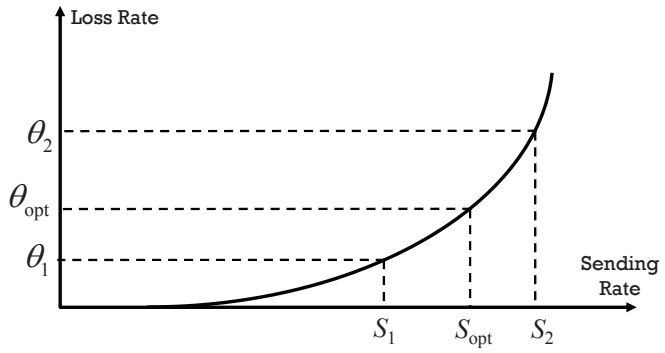


Figure 5.5 Loss rate corresponds to sending rate.

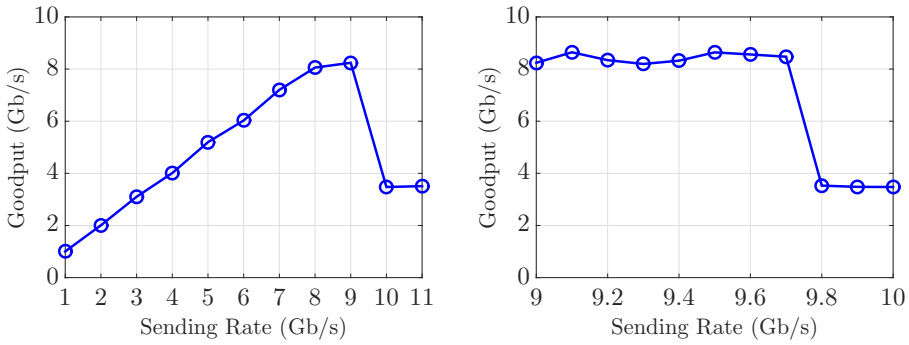
Given the network connection, the sender host, and the receiver host, as shown in Figure 5.5, higher sending rate results in high loss rate. There exists a certain sending rate S_{opt} and corresponding loss rate θ_{opt} that result in the optimal transfer performance $G_{opt} = S_{opt} \cdot (1 - 2\theta_{opt})$. The sending rate S_{opt} has the following properties

$$\begin{cases} S_1 < S_{opt} \Rightarrow \theta_1 < \theta_{opt} \Rightarrow G_1 < G_{opt} \\ S_2 > S_{opt} \Rightarrow \theta_2 > \theta_{opt} \Rightarrow G_2 < G_{opt} \end{cases}, \tag{5.17}$$

i.e., from sending rate S_{opt}, increased sending rate causes increased packet loss more than the increased sent data; and decreased sending rate causes the decreased packet loss less than the decreased sent data, both of which result in lower performance.

In Tsunami, the increasing speed of sending rate is faster than the decreasing speed given that $\alpha = \frac{6}{5}$ and $\beta = \frac{24}{25}$, see Equation 5.6. Therefore, the sending rate would be easily increased up to the upper limit S_{max}. Based on the above modeling and analysis, we know that S_{max} is critical for the transfer performance. Given a specific network environment, we do not have enough knowledge about setting the optimal S_{max} unless we perform a complete transport profiling.

As shown in Figure 5.6(a), if the target rate is "allowable" (e.g., 1 Gb/s < S_{max} < 8 Gb/s in this test case), the target rate limits the sending rate and eventually results in relatively lower performance comparing with the peak we could obtain; while if the target rate is set to be near or higher than the connection bandwidth, the performance dramatically decreases, which indicates that the overwhelmed sending rate causes severe packet losses and significant amount of time is wasted on the retransmission. In Figure 5.6(b), we "zoom in" and figure out the maximum allowable sending rate with smaller sending rate intervals, in this test case, the maximal allowable sending rate given the specific data transfer environment is around 9.7 Gb/s.



(a) (b)

Figure 5.6 Performance corresponds to target sending rate.

5.4.5 Adaptive Rate and Error Threshold Control

Given parameter values of α, β, S_{max}, and ξ, the sending rate S^* at equilibrium status of a data transfer using Tsunami satisfies the following conditions,

$$\begin{cases} S_1 < S^* \Rightarrow \theta_1 < \theta^* \Rightarrow \epsilon_1 < \xi \\ S_2 \geq S^* \Rightarrow \theta_2 \geq \theta^* \Rightarrow \epsilon_2 \geq \xi \end{cases} . \tag{5.18}$$

If the sending rate is decreased from S^* to S_1 and the error rate ϵ_1 is below ξ, Tsunami would increase the sending rate in next interval; if the sending rate is increased from S^* to S_2 and the corresponding error rate is above ξ, Tsunami would

decrease the sending rate in next interval. Based on the performance analysis in Section 5.4.4, to optimize the overall performance, we should stabilize the sending rate at equilibrium status to be the optimal one, i.e., let $S^* = S_{opt}$. Different host configurations and connections may have different optimal target sending rates, and Equation 5.18 tells us that S^* is critically decided by the user-specified error rate threshold ξ. In Tsunami, once the threshold is determined by the user, it is fixed for the entire data transfer, which makes it uncertain to stabilize the sending rate around the optimal value: i) if ξ is set to be too high, when the sending rate is increased up to a level that causes severe packet loss, Tsunami is not able to reduce the sending rate accordingly (due to the high value of ξ) and thus resources would be wasted on retransmitting unnecessarily lost blocks; ii) if ξ is set to be too low that is easily exceeded, the sending rate may keep decreasing and be limited within a small lower range that results in lower loss and lower buffer occupancy. In such case, network and end host power may be left unused, although it could be better utilized by allowing higher error rate thresholds.

We propose an Adaptive Rate and Error Threshold (ARET) control approach to improve the data transfer performance over dedicated connections. Suppose that at the the ith interval, the sending rate is S_i, the corresponding loss rate, error rate, and goodput are θ_i, ϵ_i, and G_i, respectively. We adjust the error threshold ξ at a specific interval based on the observations and analysis of its previous intervals.

Increased Sending Rate If the sending rate is increased from S_{i-1} to S_i, $S_{i-1} < S_i$, we observe G_i, θ_i, and ϵ_i and then adjust ξ in each interval as follows:

- $G_{i-1} < G_i, \theta_{i-1} < \theta_i \Rightarrow \xi \uparrow$
  Increased sending rate results in both higher performance and higher packet loss. This may indicate that higher sending rate that leads to better performance may be allowable. We increase ξ and let the protocol have a better chance to further increase its sending rate for better performance;

- $G_{i-1} < G_i, \theta_{i-1} \geq \theta_i \Rightarrow \xi \rightarrow$
  Increased sending rate results in higher performance but lower packet loss. This indicates that the observations may be statistically inaccurate due to dynamics and randomness. We keep the current ξ and let the protocol accumulate more information for further adjustment;

- $G_{i-1} \geq G_i, \theta_{i-1} < \theta_i \Rightarrow \xi \downarrow$
  Since increasing sending rate when the loss rate is θ_{i-1} causes lower performance. This may indicate that a loss rate of θ_{i-1} is probably the upper limit that the environment can handle. We decrease the current error threshold ξ accordingly;

- $G_{i-1} \geq G_i, \theta_{i-1} \geq \theta_i \Rightarrow \xi \rightarrow$
  Increased sending rate results in both lower performance and lower packet loss. Such observations are in conflict and do not provide any meaningful suggestions for adjustment of ξ. We keep the current ξ.

Decreased Sending Rate If the sending rate is decreased from S_{i-1} to S_i, $S_{i-1} > S_i$, we similarly adjust ξ as follows:

- $G_{i-1} < G_i, \theta_{i-1} < \theta_i \Rightarrow \xi \rightarrow$
  Decreased sending rate results in higher performance and higher packet loss. We keep the current ξ and accumulate more observations for further adjustment;

- $G_{i-1} < G_i, \theta_{i-1} \geq \theta_i \Rightarrow \xi \downarrow$
  Decreased sending rate results in higher performance but lower packet loss. We further decrease ξ to limit the sending rate increase and trigger more decreases;

- $G_{i-1} \geq G_i, \theta_{i-1} < \theta_i \Rightarrow \xi \rightarrow$
  Decreased sending rate results in lower performance and higher packet loss. We keep the current ξ since such observations are in conflict;

- $G_{i-1} \geq G_i, \theta_{i-1} \geq \theta_i \Rightarrow \xi \uparrow$
  Decreases sending rate results in lower performance and lower packet loss. We increase ξ and let the protocol probe for higher performance.

5.5 Performance Evaluation

We conduct data transfer experiments over a local testbed with 10 Gb/s connections with various RTT delays ranging from 0 ms to 300 ms emulated by `netem` [17]. We preliminarily observe in Figure 5.7 that the proposed approach makes the protocol insensitive to the target rates across different RTTs. In addition, the performance

produced by the proposed method seems to be insensitive to the RTT delays (at least not in the same way with AIMD-family protocols) although longer delays do cause higher performance variances.

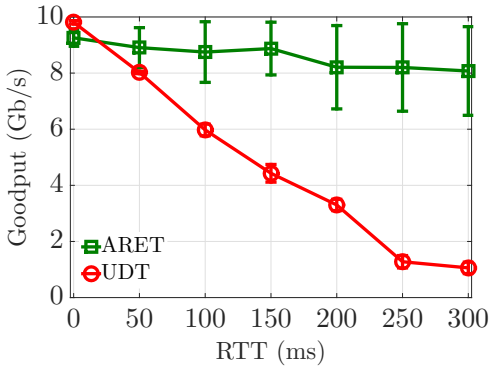


Figure 5.7 Performance comparison over emulated connections with various delays.

5.6 Related Work

Existing work on protocol design to improve the data transfer performance mainly falls into two categories: i) TCP enhancements; and ii) UDP-based transport typically with non-AIMD control. We conduct a brief survey as follows.

5.6.1 TCP Enhancements

Transmission Control Protocol (TCP) [54] has shown its success in the past decades on the Internet. Its AIMD algorithm has been proven to be effective and sufficient for convergence when the network needs to be fairly shared among different users [34]. However, TCP is not well suited for big data transfer over long-haul dedicated connections in HPNs due to its conservative AIMD congestion control. In recent years, many changes to TCP have been introduced to improve its performance in high-speed networks [41]. Scalable TCP [58], HSTCP [42], BIC-TCP [90], CUBIC TCP [52] use packet loss as the only indication of congestion, but with different formulas to adjust

the sending rate. TCP Vegas [30] and FAST TCP [83] use delay as a signal to detect the network congestion. The Fast Active-Queue-Management Scalable TCP (FAST) is based on a modification of TCP Vegas [30, 65]; it continuously measures RTT to estimate the queuing delay for congestion detection and employs a linear segmented congestion control mechanism. Sync-TCP [89] employs a synchronization approach in its delay-based congestion algorithm to facilitate bandwidth-greedy but elastic applications, while at the same time not hurting other competing flows. Compound TCP [81] explores a hybrid approach that takes both delay and loss information into consideration for rapidly increasing and gracefully retreating the sending rate. Similarly, TCP-Illinois [64] uses both packet loss information and queuing delay for rate control: packet loss is used to make a decision on whether the window size needs to be changed or not, and delay information is used to calculate the increment and decrement quantity. High-Speed TCP Low Priority (HSTCP-LP) is a TCP-LP variant with aggressive window increase policy targeting high-bandwidth and long-distance networks [60]. The Explicit Control Protocol (XCP) has a congestion control mechanism designed for networks with a large BDP [56], and requires the changes of routers in networks. The Rate Control Protocol (RCP) [19] adds an end host congestion control layer between IP and TCP/UDP, and similar to XCP, it also requires the participation of routers. The Stream Control Transmission Protocol (SCTP) is a new standard for robust Internet data transport [80], whose congestion control algorithms are derived from TCP with changes to allow for multihoming. Another type of approaches uses multiple TCP streams, including bbFTP [2], GridFTP [12], and MPTCP [27, 73]. Although providing high bandwidth utilization, multiple TCP streams have been observed to be unstable when an excessive number of sockets are used, and it is not straightforward to determine an appropriate number of sockets to use. Other efforts in this area are devoted to end host tuning and optimization, which usually retains the core algorithms of TCP but

adjusts the send or receive buffer sizes to enforce supplementary rate control [38,71,75] or modifies the system configurations specifically for high-speed data transfer [33,63]. Tools such as `iperf3` [8], Transport Profile Generator (TPG) [92], and `FastProf` [91] are available to help tuning the parameters of data transfer protocols.

5.6.2 UDP-based Protocols

Transport protocols based on User Datagram Protocol (UDP) have been developed by using various rate control algorithms. Although the underlying rate control algorithms of these methods can be applied at the transport layer, they are typically implemented over UDP as application-level programs. Such implementations enable easy deployment by avoiding the modifications of operating system kernels, routers, and other network infrastructures. RBUDP [53] uses a UDP blast channel to send data blocks and a TCP channel to deliver control information and acknowledgements. RBUDP asks the user to specify a desired data transfer target rate, and to obtain a good performance, it requires the receiver not to be the bottleneck, which oftentimes is not the case in HPNs. Tsunami [22] explores a similar approach to RBUDP, but additionally adds a sending rate control mechanism based on a periodically calculated error rate. Simple Available Bandwidth Utilization Library (SABUL) [47] is a Multiplicative Increase Multiplicative Decrease (MIMD) rate-based protocol designed for shared networks where the sender senses the available bandwidth and adjusts its sending rate accordingly by tuning the inter-packet delay. SABUL uses UDP to transfer data and TCP to exchange control information. Based on SABUL, the UDP-based Data Transfer Protocol (UDT) [45, 48, 49] removes the TCP control channel in SABUL and is purely built on the top of UDP. UDT incorporates an AIMD with decreasing increases, namely DAIMD algorithm for rate control, and also uses a bandwidth estimation technique to determine the increase parameter for efficiency. RUNAT [86] explores a stochastic approximation method to achieve high

throughput at the application level. It operates around a local maximum of the throughput regression curve by dynamically adjusting the source rate in response to acknowledgements and losses based on the statistical behavior of the network connection. Hurricane [87] strategically selects and tunes hosts parameters to control the sending rate and the retransmission process to achieve high channel utilization. PA-UDP [39] is a file transfer protocol that explores a novel delay-based rate throttling model to dynamically and autonomously maximize performance under different systems without modifications of system kernels. RAPID$^+$ is also an end system aware protocol [36] developed based on RBUDP [53] and RAPID [26], which allows multiple other applications to run simultaneously on the end system by monitoring the performance of the receiver. RAPID$^+$ uses the NIC buffer capacity to calculate the initial sending rate, and the succeeding rate control depends on performance parameters sampled on the receiver such as packet loss and incoming packet rate. The measurement of incoming packet rate in RAPID$^+$ requires kernel modification, which limits its deployment [36].

In addition, a set of utility-based transport control methods such as PCC [37] are also proposed. In these approaches, the sender constantly observes the correlations between its rate control actions and corresponding performance changes, and then conduct the rate control empirically based on historical experienced measurements.

CHAPTER 6

CONCLUSION AND FUTURE WORK

The objective of this dissertation is to provide end users of science domains with an integrated and easy-to-use transport solution for host and network resource discovery and selection, end-to-end data transfer path composition and establishment, transport profile generation, and actual data movement in HPNs. The successful development and deployment of our solution would untangle scientific users from complex data transfer tasks as they only need to provide a data transfer request describing the desired transport service and the corresponding performance requirement, so they could focus on their own science missions.

We proposed a workflow-based transport solution to meet big data transfer requirements of large-scale scientific applications, which integrates three major components, i.e., i) transport-support workflow optimization; ii) transport profile generation; and iii) transport protocol design, into a unified framework. Experimental results showed that the proposed solution achieves a reasonable accuracy in modeling the existing resources/services and improves the end-to-end data transfer performance over dedicated connections in HPNs.

By leveraging the resource discovery capability of NADMA, we constructed cost models for discovered resources and formulated path composition and module selection as an optimization problem. We proved it to be NP-complete and designed optimal pseudo-polynomial (i.e., practically efficient) algorithms. We evaluated the proposed algorithms using simulations in comparison with a greedy approach, and also conducted proof-of-concept experiments in wide-area networks to validate the cost models and illustrate the efficacy of the proposed solution. The current HPN environments only have a limited number of advanced networking services that are

free of charge to authorized users, which makes it possible to employ a linear optimal algorithm to choose transport-support workflow modules and compose the optimal end-to-end network path. With new services and technologies rapidly emerging, more sophisticated approaches are needed to tackle this optimization problem. The framework of the proposed solution has been proven to be flexible to accommodate the increase on the number of modules in the workflow up to tens of thousands with quite high edge densities. We plan to integrate this workflow solution to the NADMA system and further test it extensively in various network environments.

We designed and implemented a Transport Profile Generator (TPG) to characterize and enhance the transport performance of the selected transport method by tuning the control parameters using an exhaustive approach. We used UDT as an example in the implementation and conducted extensive data transfer experiments over local- and wide-area network connections to illustrate how existing transport protocols benefit from TPG in optimizing their performance. It is of our interest to extend TPG with more transport protocols such as UDP and SCTP [80] and conduct more experiments to understand and exploit the properties of big data transfer over high-speed dedicated connections.

We further designed a stochastic approximation-based transport profiler, namely, `FastProf`, to accelerate the profiling process for big data transfer in HPNs. We implemented `FastProf` based on TPG, and conducted both extensive profiling emulations in comparison with other search algorithms and profiling experiments on physical connections with short (2 ms) and long (380 ms) delays. The emulation and experimental results showed that `FastProf` significantly reduces the profiling time while achieving a comparable level of data transfer performance, which makes it feasible to conduct "on-line" profiling. It is worthy to investigate various aspects of the applications of the SA-based method such as gradient approximation averaging, step size adaption, intelligent termination conditions to further improve the profiling

performance and accuracy of `FastProf`. Since the end-to-end data transfer is a complex process that involves both network and end hosts, the parameter selection for each part of the entire process would affect the application level performance observed by end users. Naturally, it is also of our interest to explore the possibility of applying such SA-based approaches to storage profiling on end hosts based on tools such as XDD [76].

Extensive experimental studies have shown the insufficiencies of traditional transport control methods over high-speed dedicated connections. We conducted data transfer experiments using various existing methods, among which the Tsunami protocol has shown promising performance gain over high-speed dedicated connections. We constructed performance model of Tsunami and proposed several approaches to improve the resource utilization of dedicated connections, and the preliminary experimental results showed promising performance along this research direction.

BIBLIOGRAPHY

[1] Advanced Networking Initiative. https://goo.gl/gjBdrj, accessed on July 19, 2016.

[2] bbFTP. http://doc.in2p3.fr/bbftp/, accessed on July 20, 2016.

[3] Berkeley Storage Manager. https://goo.gl/NsOq0Y, accessed on July 19, 2016.

[4] Department of Energy. http://www.energy.gov/, accessed on June 15, 2016.

[5] End Site Control Plane Service. https://goo.gl/F3mXuU, accessed on July 20, 2016.

[6] Energy Sciences Network. http://www.es.net, accessed on June 15, 2016.

[7] ESnet OSCARS. http://www.es.net/oscars, accessed on June 15, 2016.

[8] ESnet iperf3. https://github.com/esnet/iperf, accessed on June 15, 2016.

[9] GÉANT. http://goo.gl/kQ5vil, accessed on July 20, 2016.

[10] General Parallel FileSystem. http://goo.gl/AJFVhZ, accessed on July 19, 2016.

[11] GENI. http://www.geni.net, accessed on June 15, 2016.

[12] GridFTP. http://goo.gl/6Q3pl5, accessed on July 19, 2016.

[13] Internet2. http://www.internet2.edu, accessed on June 15, 2016.

[14] Internet2 ION Service. http://www.internet2.edu/ion, accessed on June 15, 2016.

[15] JGN II. http://www.jgn.nict.go.jp, accessed on June 15, 2016.

[16] Lustre File System. http://lustre.org/, accessed on June 15, 2016.

[17] Network Emulation netem. http://goo.gl/RdKohR, accessed on July 19, 2016.

[18] ORNL Climate Change Science. http://goo.gl/jZ2SfJ, accessed on July 19, 2016.

[19] Rate Control Protocol. http://goo.gl/yUjVWG, accessed on July 19, 2016.

[20] The eXtreme dd toolset. https://github.com/bws/xdd, accessed on June 28, 2016.

[21] Transport Profile Generator. https://goo.gl/dViNuk, accessed on July 19, 2016.

[22] Tsunami UDP Protocol. http://goo.gl/FFW64K, accessed on July 19, 2016.

[23] The Office of Science Data-Management Challenge. Report from the DOE Office of Science Data-Management Workshops, Technical Report SLAC-R-782, SLAC, March–May 2004. Available online: http://goo.gl/cbCkX3, accessed on July 20, 2016.

[24] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus Striped GridFTP Framework and Server. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC '05)*, pages 54–65.

[25] L. Andersson and G. Swallow. The Multiprotocol Label Switching (MPLS) Working Group decision on MPLS signaling protocols. IETF Request for Comments 3468, 2003.

[26] A. Banerjee, W. Feng, B. Mukherjee, and D. Ghosal. RAPID: An End-System Aware Protocol for Intelligent Data Transfer over Lambda Grids. In *Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS '06)*, pages 93–102, 2006.

[27] S. Barré, C. Paasch, and O. Bonaventure. MultiPath TCP: From Theory to Practice. In *Proceedings of the 10th International IFIP TC 6 Networking Conference (NETWORKING '11)*, pages 444–457, 2011.

[28] M. Beltrán, A. Guzmá n, and J.L. Bosque. A New CPU Availability Prediction Model for Time-Shared Systems. *IEEE Transactions on Computers*, 57(7):865–875, 2008.

[29] S. Bradley, F. Burstein, L. Cottrell, B. Gibbard, D. Katramatos, Y. Li, S. McKee, R. Popescu, D. Stampf, and D. Yu. TeraPaths: A QoS-enabled Collaborative Data Sharing Infrastructure for Petascale Computing Research. In *Proceedings of the 12th Conference on Computing in High Energy and Nuclear Physics (CHEP '06)*, 2006.

[30] L.S. Brakmo, S.W. O'Malley, and L.L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *ACM SIGCOMM Computer Communication Review*, 24(4):24–35, 1994.

[31] P. Brown, M. Zhu, Q. Wu, and X. Lu. Network-aware Data Movement Advisor. In *Proceedings of the 1st International Workshop on Network-aware Data Management (NDM '11)*, pages 31–40, 2011.

[32] P. Brown, M. Zhu, Q. Wu, D. Yun, and J. Zurawski. A Workflow-Based Network Advisor for Data Movement with End-to-End Performance Optimization. In *Proceedings of the 7th Workshop on Workflows in Support of Large-Scale Science (WORKS '12)*, pages 73–81, 2012.

[33] J.S. Chase, A.J. Gallatin, and K.G. Yocum. End System Optimizations for High-speed TCP. *IEEE Communications Magazine*, 39(4):68–74, 2001.

[34] D.M. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks*, 17(1):1–14, 1989.

[35] J. Corbet, A. Rubini, and G. Kroah-Hartman. *Linux Device Drivers, Third Edition.* O'Reilly Media, Inc., Boston, MA, 2005.

[36] P. Datta, W. Feng, and S. Sharma. End-system Aware, Rate-adaptive Protocol for Network Transport in LambdaGrid Environments. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing (SC '06)*, Article No. 112, 14 pages.

[37] M. Dong, Q. Li, D. Zarchy, P.B. Godfrey, and M. Schapira. PCC: Re-architecting Congestion Control for Consistent High Performance. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*, pages 395–408, 2015.

[38] T. Dunigan, M. Mathis, and B. Tierney. A TCP Tuning Daemon. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing (SC '02)*, pages 1–16.

[39] B. Eckart, X. He, Q. Wu, and C. Xie. A Dynamic Performance-Based Flow Control Method for High-Speed Data Transfer. *IEEE Transactions on Parallel and Distributed Systems*, 21(1):114–125, 2010.

[40] R.H. Farrell. Bounded Length Confidence Intervals for the Zero of a Regression Function. *Annals of Mathematical Statistics*, 33(1):237–247, 1962.

[41] S. Floyd. HighSpeed TCP for Large Congestion Windows. IETF Request for Comments 3649, 2003.

[42] S. Floyd. Limited Slow-Start for TCP with Large Congestion Windows. IETF Request for Comments 3742, 2004.

[43] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman, San Francisco, CA, 1979.

[44] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[45] Y. Gu. *UDT: A High Performance Data Transport Protocol*. PhD thesis, University of Illinois at Chicago, 2005.

[46] Y. Gu and R. Grossman. UDTv4: Improvements in Performance and Usability. In *Proceedings of the 2nd International Conference on Networks for Grid Applications (GridNets '08)*, pages 9–23, 2008.

[47] Y. Gu and R.L. Grossman. SABUL: A Transport Protocol for Grid Computing. *Journal of Grid Computing*, 1(4):377–386, 2003.

[48] Y. Gu and R.L. Grossman. UDT: UDP-based Data Transfer for High-speed Wide Area Networks. *Computer Networks*, 51(7):1777–1799, 2007.

[49] Y. Gu, X. Hong, and R.L. Grossman. Experiences in Design and Implementation of a High Performance Transport Protocol. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC '04)*, pages 22–35.

[50] Y. Gu, X. Hong, and R.L. Grossman. An Analysis of AIMD Algorithm with Decreasing Increases. In *Proceedings of the 1st International Workshop on Networks for Grid Applications (GridNets '04)*, 2004.

[51] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston. Intra and Interdomain Circuit Provisioning using the OSCARS Reservation System. In *Proceedings of the 3rd International Conference on Broadband Communications, Networks and Systems (BROADNETS '06)*, pages 1–8, 2006.

[52] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-friendly High-speed TCP Variant. *ACM SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[53] E. He, J. Leigh, O. Yu, and T. A. DeFanti. Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. In *Proceedings of the 2002 International Conference on Cluster Computing (CLUSTER '02)*, pages 317–324.

[54] V. Jacobson. Congestion Avoidance and Control. *ACM SIGCOMM Computer Communication Review*, 18(4):314–329, 1988.

[55] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a Globally-Deployed Software Defined WAN. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[56] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. *ACM SIGCOMM Computer Communication Review*, 32(4):89–102, 2002.

[57] H. Kellerer, U. Pferschy, and D. Pisinger. *The Multiple-Choice Knapsack Problem*, chapter 11, pages 317–347. Springer, Berlin Heidelberg, 2004.

[58] T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. *ACM SIGCOMM Computer Communication Review*, 33(2):83–91, 2003.

[59] J. Kiefer and J. Wolfowitz. Stochastic Estimation of the Maximum of A Regression Function. *Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[60] A. Kuzmanovic, E.W. Knightly, and R.L. Cottrell. HSTCP-LP: A Protocol for Low-Priority Bulk Data Transfer in High-Speed High-RTT Networks. In *Proceedings of the 2nd International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet '04)*, 2004.

[61] G.F. Lawler and V. Limic. *Random Walk: A Modern Introduction*. Cambridge University Press, New York, NY, 2010.

[62] T. Lehman, J. Sobieski, and B. Jabbari. DRAGON: A Framework for Service Provisioning in Heterogeneous Grid Networks. *IEEE Communications Magazine*, 44(3):84–90, 2006.

[63] B. H. Leitao. Tuning 10Gb Network Cards on Linux. In *Proceedings of the 2009 Linux Symposium*, pages 169–184.

[64] S. Liu, T. Başar, and R. Srikant. TCP-Illinois: A Loss- and Delay-based Congestion Control Algorithm for High-speed Networks. *Performance Evaluation*, 65(6–7):417–440, 2008.

[65] S.H. Low, L.L. Peterson, and L. Wang. Understanding TCP Vegas: A Duality Model. *Journal of the ACM*, 49(2):207–235, 2002.

[66] J. Martin, A. Nilsson, and I. Rhee. Delay-based Congestion Avoidance for TCP. *IEEE/ACM Transactions on Networking*, 11(3):356–369, 2003.

[67] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[68] M.R. Meiss. Tsunami: A High-Speed Rate-Controlled Protocol for File Transfer. Available online: `http://goo.gl/hGczKp`, accessed on July 19, 2016.

[69] J. Mogul and S. Deering. Path MTU Discovery. IETF Request for Comments 1191, 1990.

[70] J. Padhye, V. Firoiu, D.F. Towsley, and J.F. Kurose. Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, 2000.

[71] R.S. Prasad, M. Jain, and C. Dovrolis. Socket Buffer Auto-Sizing for High-Performance Data Transfers. *Journal of Grid Computing*, 1(4):361–376, 2003.

[72] R.S. Prasad, M. Murray, C. Dovrolis, and K. Claffy. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE Network*, 17(6):27–35, 2003.

[73] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, pages 399–412, 2012.

[74] 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.

[75] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP Buffer Tuning. *ACM SIGCOMM Computer Communication Review*, 28(4):315–323, 1998.

[76] B.W. Settlemyer, J.D. Dobson, S.W. Hodson, J.A. Kuehn, S.W. Poole, and T.M. Ruwart. A Technique for Moving Large Data Sets over High-performance Long Distance Networks. In *Proceedings of the 27th Symposium on Mass Storage Systems and Technologies (MSST '11)*, pages 1–6, 2011.

[77] J.C. Spall. Multivariate Stochastic Approximation Using a Simultaneous Perturbation Gradient Approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[78] J.C. Spall. Implementation of the Simultaneous Perturbation Algorithm for Stochastic Optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):817–823, 1998.

[79] J.C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. John Wiley & Sons, Inc., New York, NY, 2003.

[80] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson. Stream Control Transmission Protocol. IETF Request for Comments 2960, 2000.

[81] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. In *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM '06)*, pages 1–12, 2006.

[82] T. Wada and Y. Fujisaki. A Stopping Rule for Simultaneous Perturbation Stochastic Approximation. In *Proceedings of the 2013 European Control Conference (ECC '13)*, pages 644–649.

[83] D.X. Wei, C. Jin, S.H. Low, and S. Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Transactions on Networking*, 14(6):1246–1259, 2006.

[84] J. Wu, M. Savoie, H. Zhang, and S. Campbell. A User-Controlled Lightpath Provisioning System for Grid Optical Networks. In *Proceedings of the 10th IEEE Singapore International Conference on Communication Systems (ICCS '06)*, pages 1–5, 2006.

[85] Q. Wu, N. S. V. Rao, and X. Lu. On Transport Methods for Peak Utilization of Dedicated Connections. In *Proceedings of the 6th International Conference on Broadband Communication (BROADNETS '09)*, pages 1–8, 2009.

[86] Q. Wu and N.S.V. Rao. A Class of Reliable UDP-based Transport Protocols based on Stochastic Approximation. In *Proceedings of the 24th IEEE International Conference on Computer Communications (INFOCOM '05)*, pages 1013–1024, 2005.

[87] Q. Wu and N.S.V. Rao. Protocol for High-Speed Data Transport over Dedicated Channels. In *Proceedings of the 3rd International Workshop on Protocols for Fast Long-Distance Networks (PFLDNet '05)*, 2005.

[88] W. Wu, M. Crawford, and M. Bowden. The performance analysis of linux networking - Packet receiving. *Computer Communications*, 30(5):1044–1057, 2007.

[89] X. Wu, M.C. Chan, A.L. Ananda, and C. Ganjihal. Sync-TCP: A New Approach to High Speed Congestion Control. In *Proceedings of the 17th International Conference on Network Protocols (ICNP '09)*, pages 181–192, 2009.

[90] L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control (BIC) for Fast Long Distance Networks. In *Proceedings of the 23rd IEEE International Conference on Computer Communications (INFOCOM '04)*, pages 2514–2524, 2004.

[91] D. Yun, C.Q. Wu, N.S.V. Rao, Q. Liu, R. Kettimuthu, and E.S. Jung. Profiling Optimization for Big Data Transfer Over Dedicated Channels. In *Proceedings of the 25th International Conference on Computer Communication and Networks (ICCCN '16)*, 2016.

[92] D. Yun, C.Q. Wu, N.S.V. Rao, B.W. Settlemyer, J. Lothian, R. Kettimuthu, and V. Vishwanath. Profiling Transport Performance for Big Data Transfer over Dedicated Channels. In *Proceedings of the 2015 International Conference on Computing, Networking and Communications (ICNC '15)*, pages 858–862.

[93] 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(8):s11–s17, 2005.