11-26-2011

Advance Bandwidth Scheduling in High-speed Dedicated Networks

Yunyue Lin

ADVANCE BANDWIDTH SCHEDULING IN HIGH-SPEED DEDICATED NETWORKS

by

Yunyue Lin

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Major: Computer Science

The University of Memphis

December, 2011

To My Beloved Parents.

Abstract

Lin, Yunyue. Ph.D. The University of Memphis. December 2011. Advance Bandwidth Scheduling in High-speed Dedicated Networks. Major Professor: Dr. Qishi Wu.

An increasing number of high-performance networks provision dedicated connections through circuit switching or MPLS/GMPLS techniques to support large data transfer. The link bandwidths in such networks are typically shared by multiple users through advance reservation, resulting in varying bandwidth availability in future time. We investigate a comprehensive set of advance bandwidth scheduling problems that are categorized into the following four classes.

1) Basic bandwidth scheduling. We formulate four types of problems by exhausting the combinations of different path and bandwidth constraints: fixed/variable path with fixed/variable bandwidth (F/VP-F/VB) with the same objective to minimize the data transfer end time for a given date size. For VPFB and VPVB, we further consider two subcases where the path switching delay is negligible or non-negligible. We propose an optimal algorithm for each of these problems except for FPVB and VPVB with non-negligible path switching delay, which are proved to be NP-complete and non-approximable, and then tackled by heuristics.

2) Bandwidth scheduling in LCC-overlays. We investigate two problems in this class: fixed-bandwidth path (FBP) and varying-bandwidth path (VBP) with the same objective to minimize the data transfer end time for a given data size. We prove both problems to be NP-complete and non-approximable, and propose heuristic algorithms using a gradual relaxation procedure on the maximum number of links from each LCC allowed for path computation.

3) Distributed bandwidth scheduling. We propose distributed algorithms to meet four basic bandwidth scheduling requests: fixed bandwidth in a fixed slot, highest bandwidth in a fixed slot, first slot with fixed bandwidth and duration, and all slots with fixed bandwidth and duration. These algorithms are developed through a rigorous extension of the classical breadth first search and Bellman-Ford algorithms to a complete distributed manner.

4) Periodical bandwidth scheduling. We consider two problems in this class: multiple data transfer allocation (MDTA) and multiple fixed-slot bandwidth reservation (MFBR), both of which schedule multiple user requests accumulated in a certain time window. For MDTA, we design an optimal algorithm and provide its correctness proof; while for MFBR, we prove it to be NP-complete and propose a heuristic algorithm.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background

A number of large-scale applications in various fields of science, engineering and business are generating colossal amounts of data, on the order of terabytes currently and petabytes or even exabytes in the near future, which must be transferred over wide geographical areas for remote operations. Typical examples include next-generation computational science applications where large simulation data sets produced on supercomputers are shared by a distributed team of collaborative scientists [1–3], or a large chain of departmental stores whose transaction records or inventories are synchronized during off-peak hours. Since the data providers and consumers in these distributed applications are generally located at different sites across the nation or around the globe, high-speed connections are needed to support a variety of remote tasks including data mining, consolidation, alignment, storage, visualization, and analysis [35]. Unfortunately, the conventional shared public networks such as the Internet are not adequate to meet the unprecedented data transfer challenge posed by the sheer data volume of such scales.

High-performance networks that provision dedicated links have proved to be very successful in meeting the large data transfer needs in many applications. The high-performance networking requirements in large-scale applications belong to two broad classes: (a) high bandwidths, typically multiples of 10Gbps, to support bulk data transfers, and (b) stable bandwidths, typically at much lower bandwidths such as 100s of Mbps, to support interactive, steering and control operations. The current Internet technologies are severely limited in meeting these demands. First, such bulk bandwidths are available only in the backbone, typically shared among a number of connections that are unaware of the demands of others. Second, due to the shared nature of packet-switched networks, Internet connections often exhibit complicated dynamics, thereby lacking the stability needed for steering and control operations [33].

In recent years, high-speed dedicated networks have emerged to be a promising solution to support remote tasks in these data- and network-intensive applications and the significance of high-performance networks has been well recognized in the broad science and network research communities [35]. Several projects are currently underway to develop such capabilities, including User Controlled Light Paths (UCLP) [4], UltraScience Net (USN) [34], Circuit-switched High-speed End-to-End Transport ArcHitecture (CHEETAH) [17], Enlightened [13], Dynamic Resource Allocation via GMPLS Optical Networks (DRAGON) [5], Japanese Gigabit Network II [6], Bandwidth on Demand (BoD) on Geant2 network [7], On-demand Secure Circuits and Advance Reservation System (OSCARS) [8] of ESnet, Internet2 ION [9], and Bandwidth Brokers [47]. These dedicated channels are a part of the capabilities envisioned in the Global Environment for Network Innovations (GENI) project [10].

The deployments of high-performance networks are expected to increase significantly and proliferate into both public and dedicated network infrastructure across the globe in the coming years. An evidence of this trend in production networks is reflected by Internet2 and ESnet that offer IP-based Multiple Protocol Label Switching (MPLS) tunnels and layer-2 Virtual Local Area Networks (VLANs) using OSCARS. MPLS improves the forwarding speed of IP routers by adopting a key concept from the world of virtual-circuit networks: a fixed-length label. MPLS is often referred to as layer-2.5, which adds a small MPLS header between the layer-2 header and the layer-3 header in a link-layer frame. Since modern optical networks have reached a very high transfer rate (at 40 Gbit/s and beyond), the true advantages of MPLS do not lie in the potential increase in switching speeds, but rather in the new traffic management capabilities that MPLS enables. OSCARS uses MPLS and Resource Reservation Protocol (RSVP) to create virtual circuits or Label Switched Paths (LSPs), while the management and operation of end-to-end virtual circuits within the network are done at the layer-3 network level. All network service requests to OSCARS requires the type of service (i.e., layer 2 or 3), the source and destination node, the required bandwidth, and the duration of use. OSCARS supports advance reservation, but its underlying path computation limits connections over links returned by traceroute; hence, it does not explore all available bandwidths inside the network.

The network infrastructure including edge devices, core switches, and backbone routers in these high-performance networks is coordinated by a management framework, namely control plane, which is responsible for allocating link bandwidth to users, setting up end-to-end transport paths upon request, and releasing resources when tasks are completed. As the central function unit of a generalized control plane, the bandwidth scheduler computes

appropriate network paths and allocates link bandwidths to meet specific user requests based on network topology and bandwidth availability. Hence, the performance of the bandwidth scheduler has a critical impact on the utilization of network resources and the satisfaction of user requests.

Given sufficient link bandwidths, the end-to-end application-level throughout still needs to be realized by transport methods. To achieve high and stable throughput over dedicated channels, many transport methods have been developed based on TCP enhancements [11, 21,39] or UDP with non-AIMD (Additive Increase Multiplicative Decrease) control [26,42, 48]. These transport methods provide a variety of transport capabilities such as maximizing the link utilization, stabilizing the throughput at a fixed target rate, or aggregating multiple data streams along different network paths.

In dedicated networks that support in-advance bandwidth provisioning, the existing bandwidth allocations on a link in future time slots are typically specified as segmented constant functions. The residual bandwidths on certain links are to be allocated to establish new dedicated connections, which may be composed by concatenating several links and matching their bandwidths in corresponding time slots.

1.2 An Overview of Advance Bandwidth Scheduling

This dissertation focuses on bandwidth scheduling and path computation algorithms that can be applied across connection-oriented networks. The proposed bandwidth scheduling problems are categorized into four classes: basic bandwidth scheduling, bandwidth scheduling in LCC-overlays, distributed bandwidth scheduling and periodical bandwidth scheduling.

1.2.1 Basic Bandwidth Scheduling

In view of disparate transport capabilities and multifarious application requirements, we formulate four types of instant advance bandwidth scheduling problems in high-speed networks: (i) fixed path with fixed bandwidth (FPFB), (ii) fixed path with variable bandwidth (FPVB), (iii) variable path with fixed bandwidth (VPFB), and (iv) variable path with variable bandwidth (VPVB), with the same objective to minimize the data transfer end time for a given transfer request with a pre-specified data size. For VPFB and VPVB that allow path switching during the data transfer to fully utilize network resources, two subcases where the path switching delay is negligible are further considered, referred to as VPFB-0 and VPVB-0, or non-negligible, referred to as VPFB-1 and VPVB-1. Note that minimizing the amount of data transfer time does not necessarily minimize the data transfer end time if the transfer start time is not specified. Note that an *instant scheduling* algorithm is executed immediately upon the arrival of a new data transfer request and the bandwidths are then reserved in advance on relevant links, while a *periodical scheduling* algorithm is launched periodically in a certain time interval to schedule multiple data transfer requests accumulated during that interval. We published this work in [30, 31].

Among the proposed scheduling problems, FPVB and VPVB-1 are proved to be NP-complete and non-approximable, and the rest are P problems. We design an optimal scheduling algorithm for each of the P problems with polynomial time complexity with respect to the network size and the total number of time slots in an aggregated bandwidth reservation table. For each of the NP-complete problems, we propose an optimal algorithm with exponential-time complexity for small-scale networks, and develop a heuristic

approach with polynomial-time complexity for large-scale ones. The performance superiority of these heuristics are illustrated by extensive simulation results in comparison with optimal and greedy strategies.

1.2.2 Bandwidth Scheduling in LCC-overlays

A high-performance network may be an overlay network residing over an IP network. The nodes in these high-performance overlay networks are connected by virtual or logical links, each of which corresponds to a path consisting of a number of physical links in the underlying network. If multiple overlay links share a common underlying link segment, the capacities of these overlay links are correlated and their total bandwidth is constrained by the bandwidth of the shared link segment. For example, in overlay networks built over wavelength division multiplexing (WDM) networks, routers are interconnected through wavelength channels. Since each optical fiber carries multiple wavelength channels, these channels are correlated in capacity. We consider a model of overlay networks with linear capacity constraints (LCC), which is first proposed in [49]. The capacities of overlay links in an LCC-overlay network are represented by variables and the link correlations are formulated as linear constraints of link capacities. The LCC model is a simple way to capture the link correlations in an overlay network, which only requires the addition of a set of linear capacity constraints to the overlay network.

We study two advance bandwidth scheduling problems for a data transfer request in LCC-overlay networks: Fixed-Bandwidth Path (FBP) and Variable-Bandwidth Path (VBP), with the same objective to minimize the data transfer end time for a given data size. These two problems are proved to be NP-complete and non-approximable. We further study a

subproblem of the widest-path with liner capacity constraints (WPC) problem, which is first proposed and proved to be NP-complete in [49]. We provide the upper bound of the approximation ratio of any approximate algorithm for the WPC problem and propose an approximate algorithm that gradually relaxes the maximum number of links from each LCC allowed for path computation. This algorithm serves as the base of the solutions to the FBP and VBP problems with necessary adaptations.

1.2.3 Distributed Bandwidth Scheduling

A large number of scheduling algorithms have been developed for centralized advance bandwidth reservation in high-performance networks [19]. These centralized scheduling schemes imply the use of a centralized control plane. Although feasible in small-scale networks, e.g., UltraScience netowrk [34], such centralized management pose significant reliability and scalability challenges as the network size increases. Hence, it is necessary to develop distributed advance bandwidth reservation solutions for large-scale networks. We formulate four basic advance bandwidth scheduling problems: (i) fixed bandwidth in a fixed slot, (ii) highest bandwidth in a fixed slot, (iii) first slot with fixed bandwidth and duration, and (iv) all slots with fixed bandwidth and duration. We propose distributed path computation and bandwidth scheduling algorithms for these scheduling problems. These algorithms are based on a rigorous extension of the classical breadth first search and Bellman-Ford algorithm to a complete distributed environment, and exhibit several salient features including loop free, fault tolerance, and time efficiency. We published this work in [41].

1.2.4 Periodical Bandwidth Scheduling

We study two periodic bandwidth scheduling problems: multiple data transfer allocation (MDTA) and multiple fixed-slot bandwidth reservation (MFBR), both of which schedule a number of user requests accumulated in a certain period. MDTA is to assign multiple data transfer requests on several pre-specified network paths to minimize the total data transfer end time, while MFBR is to satisfy multiple bandwidth reservation requests, each of which specifies a bandwidth and a time slot. A real-life network example using MDTA is to schedule the transfer of a large number of data sets between two remote sites where core switches are deployed and connected with multiple parallel dedicated OC-192 SONET links as is the case in UltraScience Net [34]. A practical application scenario using MFBR is to establish several control channels between collaborative sites for computational monitoring and steering operations that typically require smooth and stable data flows with constant bandwidths during certain time slots. We published this work in [29].

For MDTA, we design an optimal algorithm and provide its correctness proof, while for MFBR, we prove it to be NP-complete by reducing from the *Disjoint Path Problem with Red and Blue arcs* (DPPRB) [37] and propose a heuristic algorithm, Minimal Bandwidth and Distance Product Algorithm (MBDPA), whose performance is compared with a greedy approach. We also conduct performance comparison between periodic scheduling and instant scheduling and identify suitable operational conditions where periodic scheduling outperforms instant scheduling in terms of minimizing total transfer end time or maximizing the number of satisfied reservations.

1.3 Technical Contributions

The technical contributions of our work in this dissertation are summarized as follows:

1. We proposed a generalized control plane framework and constructed realistic network models for high-speed networks that support advance bandwidth reservations.

2. We considered a comprehensive set of advance bandwidth scheduling problems, including instant scheduling and periodical scheduling, centralized scheduling and distributed scheduling, as well as scheduling in LCC-overlays. For each of these problems, we conducted an in-depth investigation through rigorous complexity analysis and algorithm design. To the best of our knowledge, we are among the first to formulate and study these advance bandwidth scheduling problems.

3. We conducted extensive simulation-based performance comparisons using a large set of simulated networks, which show that the proposed heuristics achieve better performance than existing methods in terms of optimization goal, time complexity, or deployment flexiblity. The proposed heuristics have great potential to improve the utilization of high-speed dedicated networks and the performance of large-scale scientific applications that depend on these networks for data transfer.

The rest of the dissertation is organized as follows. An extensive survey of bandwidth scheduling is conducted in Chapter 2. In Chapter 3, we describe the framework of a generalized control plane and our network model. In Chapter 4, we investigate four types of basic instant scheduling problems. In Chapter 5, we introduce the LCC-overlay network

model and study two advance bandwidth scheduling problems in LCC-overlays. In Chapter 6, we propose distributed algorithms for four advance bandwidth scheduling problems. In Chapter 7, we study two periodic scheduling problems: MDTA and MFBR. We conclude our work in Chapter 8.

Chapter 2

Related Work

As dedicated networks are increasingly deployed under different high-performance networking initiatives, many scheduling algorithms have been designed for advance bandwidth reservation. We provide a broad survey of these efforts below.

Instant scheduling problems have been extensively studied in various network contexts and many scheduling techniques have been proposed. In [35], Rao *et al.* described four basic scheduling problems with different constraints on target bandwidths and time slots, i.e. specified bandwidth in a specified time slot, earliest available time with a specified bandwidth and duration, highest available bandwidth in a specified time slot, and all available time slots with a specified bandwidth and duration. The solutions to the first three problems are straightforward extensions of the classical Dijkstra's algorithm, while the last one is based on an extension of Bellman-Ford algorithm. Similar problems are also discussed in [36] with a detailed description of the solution to each of these problems. Guerin *et al.* investigated these basic scheduling problems with several extensions in [27] with a focus on increasing the flexibility of services. The scheduling algorithm proposed by Cohen *et al.* in [18] considers the flexibility of transfer start time and the capability of path switching

between different paths during a connection to improve network utilization. In [25], Grimmell *et al.* formulated a dynamic quickest path problem, which deals with the transmission of a message from a source to a destination with the minimum end-to-end delay over a network with propagation delays and dynamic link bandwidth constraints. In [40], files are transferred with varying bandwidths in different time slots in a simple case where the path is pre-specified. Ganguly *et al.* generalized the problem of finding an optimal path in a graph with varying bandwidths to minimize the total transfer time and proposed an optimal algorithm with exponential complexity in [22]. They also attempted to find the minimum number of path switchings for a file transfer in a specified number of time slots. Their first problem is similar to FPVB, but the authors did not provide this problem's NP-completeness and non-approximability proofs. In [24], Gorinsky *et al.* proposed a Virtual Finish Time First algorithm to schedule incoming files in a preemptive manner to minimize total transfer end time on a dedicated channel. In [38], Shen *et al.* studied advance lightpath scheduling in WDM optical networks using a two-phase approach, in which they re-provision some already scheduled lightpaths to re-optimize the network performance in the second phase.

To the best of our knowledge, there are very few studies on advance bandwidth reservation in LCC-overlay networks. The model of overlay networks with linear capacity constraints (LCC) is first proposed by Zhu *et al.* in [49], where they studied two network flow problems, widest-path (i.e. maximum-bandwidth single-path unicast) and maximum-flow (i.e. maximum-bandwidth multiple-path unicast), with the addition of LCC. They proved that widest-path with LCC is NP-complete, but did not provide any approximate algorithm.

They formulated the problem of maximum-flow with LCC as a linear programming problem and propose an algorithm for it. The shared risk link group (SRLG) is a network model related to the link LCC in IP over WDM networks [44], where the failure of an optical fiber that carries multiple IP logical links results in the failures of all the logical links that depend on it.

Most of the aforementioned work is primarily focused on centralized scheduling of bandwidths provisioned by dedicated networks. Further more, most existing network systems (e.g. OSCARS in ESnet or UltraScience Net) that support advance bandwidth reservations are managed by a centralized control plane, and a centralized repositories is used to maintain all the bandwidth reservations on all the network links. In [45], Xie *et al.* proposed a distributed link-state routing solution for advance bandwidth reservation.

Although network researchers are increasingly realizing the importance of scheduling multiple bandwidth requests to improve the utilization of expensive network resources, periodic scheduling problems in dedicated networks have not received as much attention as instant scheduling problems. However, there is a great deal of similar work on other networking subjects including optical burst switching and traffic engineering. In optical burst switching, the header of a burst is sent in advance of the data burst to reserve a wavelength channel at each optical switching node along the path. A popular algorithm, named LAUC-VF for scheduling each burst as soon as its head arrives at the node is proposed in [46]. Instead of scheduling each burst immediately upon the arrival of its head, Phung *et al.* defer making the scheduling decision until right before the arrival of the data burst in order to have full knowledge about other bursts, hence reducing unnecessary burst dropping [32]. In some traffic engineering tasks, multiple network paths between a pair of

routers are established, which enables network traffic to be split among these disjoint paths in order to reduce latency and balance traffic loads. Once multiple paths are established, individual packets can be allocated to one of these paths using different policies including Round Robin, hash function applied to the source and destination pair [15], and Opportunistic Multipath Scheduling, which opportunistically favors low-delay high-throughput paths while simultaneously ensuring that the traffic splitting ratios defined by the routing policy are satisfied [16]. The scheduling problems addressed in this dissertation differ from the aforementioned work. In MDTA problem, each file cannot be split among paths during transfer and must be strategically allocated in its entirety to one of the predefined paths to minimize the total waiting time. The MFBR problem is an extension of the fixed-slot bandwidth reservation problem proposed in [36], which schedules multiple fixed-slot bandwidth reservations to maximize the number of successful bandwidth reservations.

Chapter 3

Control Plane Framework and Network Model

3.1 Control Plane Framework

We consider a generalized control plane to support in-advance reservation of dedicated channels over high-speed networks [35]. The control plane framework shown in Figure 3.1 consists of the following components: (a) client interface, (b) server front-end, (c) user management, (d) token management, (e) database management, (f) bandwidth scheduler, and (g) signaling daemon. The interactions between these components take place either over the data plane or control plane to accomplish the tasks of user-specified bandwidth reservation, path computation and network signaling.

Depending on the system configuration, the control plane operations can be coordinated by a central management node or by a set of nodes distributed over a network. A user can remotely interact with the system through a web browser, or a web client, for example, using SOAP (Simple Object Access Protocol)-based XML message exchange. Accordingly, the server front-end could deploy a web server or web service that accepts bandwidth reservation requests from users with valid credentials. The user management module supports a

special group of users with administrative privileges to add, delete or modify user account information.

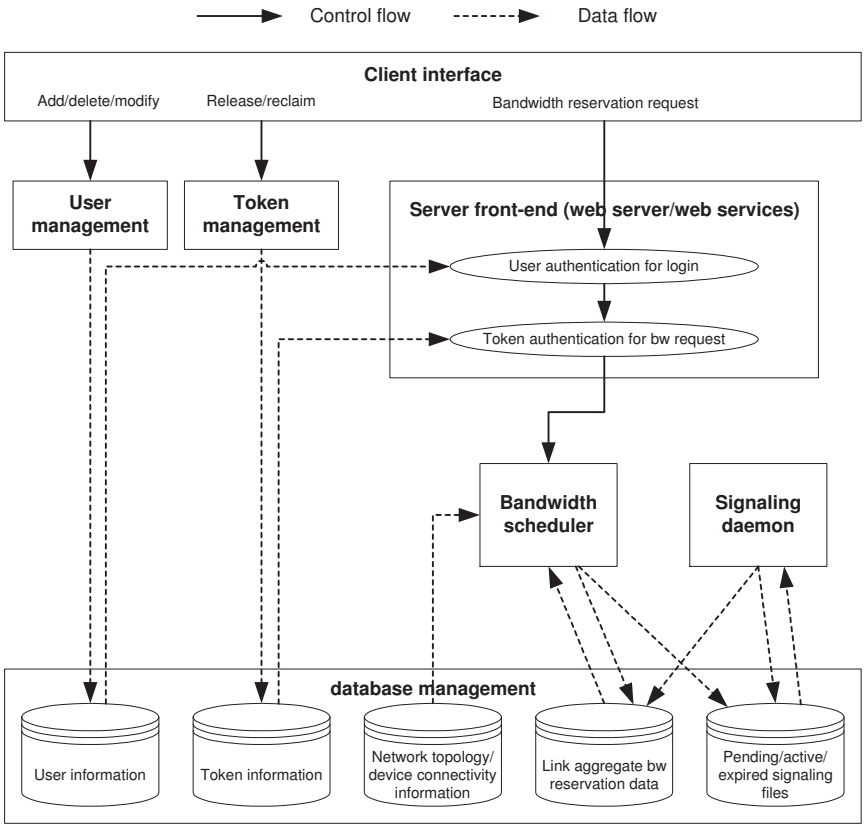

Figure 3.1: Control plane framework : function components, control flow, and data flow.

User sites are connected through their assigned ports on the edge switches in the network infrastructure. In this framework, we use token-based scheme for authorization and coordination of channel setup. Multiple tokens are provided to users for their assigned ports, which they can release to other users and reclaim them as needed. A channel reservation request is honored only if the tokens at both ends are either owned by or released to the user making the request. It is implicitly assumed that users at both ends will work out their connectivity mechanisms and policies before the tokens are released.

As the central component of a control plane, the bandwidth scheduler computes one path or a set of paths and allocates appropriate link bandwidths to satisfy user data transfer requests. Upon the completion of path computation, a signaling record is generated and the bandwidth allocation of each link along that path is updated accordingly. The signaling daemon periodically examines active or expiring signaling records. For each active or expiring signaling record, the daemon invokes appropriate signaling scripts to set up or tear down the connections along the computed or established path, respectively. The aggregate bandwidth reservation data for each component link is updated if the signaling actions are successful. Note that the time interval at which the signaling daemon is periodically activated must be chosen to be compatible with the finest resolution of the bandwidth reservation time.

3.2 Network Model

We represent a dedicated network as a graph $G = (V, E)$ with n nodes and m links, where each link $l \in E$ maintains a list of residual bandwidths specified as segmented constant functions of time. A 3-tuple of time-bandwidth (TB) $(t_l[i], t_l[i+1], b_l[i])$ is used to represent the residual bandwidth $b_l[i]$ of link l at time interval $[t_l[i], t_l[i+1]]$, $i = 0, 1, 2, \ldots, T_l - 1$, where T_l is the total number of time slots of link l. $t_l[0]$ denotes the current time point, and $t_l[i]$ $(i > 1)$ denotes a future time point. $t_l[T_l] = +\infty$, which indicates that there is no bandwidth reservation on link l after $t_l[T_l - 1]$ and therefore $b_l[T_l - 1]$ has the full bandwidth of link l.

A network path is defined as an ordered set of nodes from the source to the destination over one or more links or hops. Before computing paths, the TB lists of all links

are combined to build an Aggregated TB (ATB) list, where the residual bandwidths of all links are stored in each intersected time slot. As shown in Fig. 3.2, a set of new time slots are created by combining the time slots of all links $l \in E$, and the residual bandwidths of each link are mapped to the ATB list in each new time slot. The ABT list is denoted as $(t[0], t[1], b_0[0], b_1[0], ..., b_{m-1}[0]), ..., (t[T-1], t[T], b_0[T-1], b_1[T-1], ..., b_{m-1}[T-1])$, where T is the total number of new time slots after the aggregation of TB lists of m links. The time slot i corresponds to the time interval $[t[i], t[i+1]]$, and $t[T] = +\infty$.

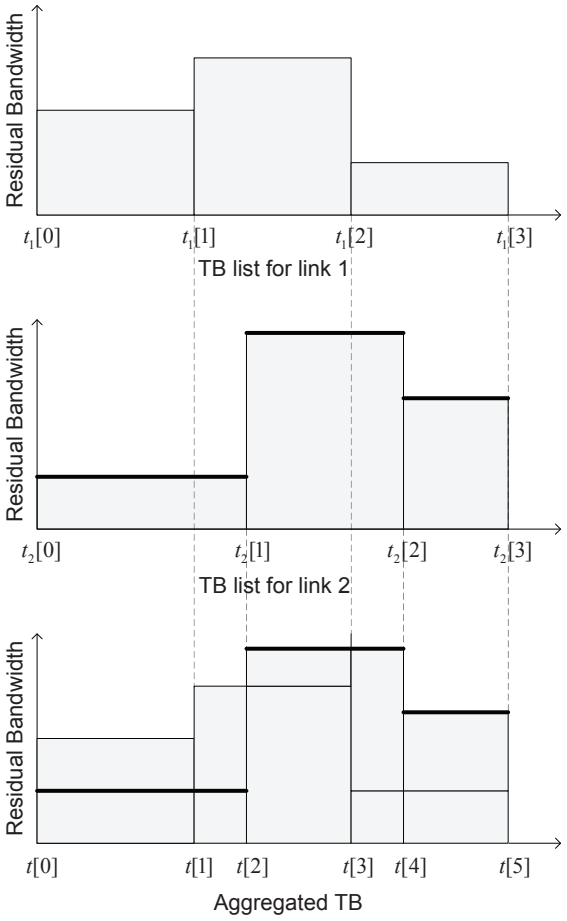

Figure 3.2: An aggregated TB list that combines the individual TB lists of link 1 and link 2, whose bandwidths are marked with different weights.

Chapter 4

Basic Bandwidth Scheduling

4.1 Problem Formulation

In view of different transport constraints and application requirements, we formulate four types of advance bandwidth scheduling problems for minimal data transfer end time as follows: Given a graph $G = (E, V)$ with an ATB list, source v_s and destination v_d, and data size δ,

- FPFB: compute a fixed path from v_s to v_d with a constant (fixed) bandwidth;
- FPVB: compute a fixed path from v_s to v_d with varying bandwidths across multiple time slots;
- VPFB: compute a set of paths from v_s to v_d with the same (fixed) bandwidth at different time slots;
- VPVB: compute a set of paths from v_s to v_d with varying bandwidths at different time slots

with the common goal to minimize the data transfer end time. In the above scheduling problems, the service is provisioned for a certain advance bandwidth reservation request

defined by a set of parameters including source v_s, destination v_d, path property (either fixed or variable), bandwidth constraint (either fixed or variable), and data size δ. A user request is classified into one of four types based on the path property (fixed/variable) and bandwidth constraint (fixed/variable) in the user input.

In both VPFB and VPVB problems, multiple paths are used in a sequential order and a path switching may be needed between two different paths computed in two adjacent time slots to fully utilize network resources. During path switching, the signaling daemon in the control plane need to send and invoke certain signaling scripts to tear down the existing path and set up a new path to continue the data transfer. The path switching generally incurs a certain amount of overhead ranging from milliseconds to seconds, depending on the signaling message delay and the number of affected switches. If the overhead is relatively small compared to the rate adjustment interval of a higher-layer transport protocol, the path switching can be done transparently. However, it may not be always beneficial to do path switching in some scenarios, especially when the path switching delay τ is comparable to the transfer time in a time slot. Here, the switching delay τ is assumed to be a constant. If the path switching delay is negligible (i.e. $\tau = 0$), these two problems are referred as VPFB-0 and VPVB-0; otherwise, (i.e. $\tau > 0$), they are referred as VPFB-1 and VPVB-1. In the extreme case where the path switching delay is so large compared to the time slot that performing any path switching would negatively affect the data transfer end time, VPFB-1 reduces to FPFB and VPVB-1 reduces to FPVB. In FPVB, the source node of the computed path needs to adjust the sending rate according to the path bandwidth in each time slot. Such rate adjustment would also incur a delay, which is relatively small and therefore is ignored.

For illustration purposes, we use a numerical example to explain the above problems. As shown in Fig. 4.1, network G has 4 nodes with designated source v_s and destination v_d, and 5 links, each of which has 2 time slots of unit size. The residual bandwidths in these 2 time slots are labeled on each link. The data size $\delta = 8$ units, the path switching delay $\tau = 0.1$ unit of time and the current time point is 0.

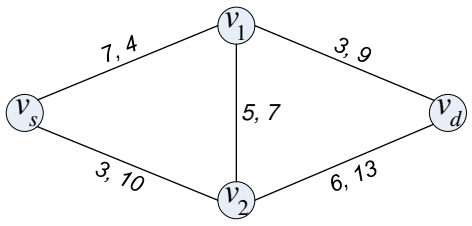

Figure 4.1: A network with two time slots.

The optimal solutions to these six bandwidth scheduling problems in the above example are shown in Fig. 4.2. For FPFB, the optimal path is $v_s - v_2 - v_d$ with the minimal data transfer end time $1 + \frac{8}{10} = 1.8$. Note that the transfer should start at time point 1 instead of 0 because the available bandwidth 10 in the second time slot is much larger than that of the first time slot. For FPVB, the optimal path is $v_s - v_2 - v_d$ with bandwidth 3 in the first time slot and bandwidth 10 in the second time slot, which results in the minimal data transfer end time $\frac{3}{3} + \frac{5}{10} = 1.5$. For VPFB-0, the widest path in the first time slot is $v_s - v_1 - v_2 - v_d$ with bandwidth 5 and the widest path in the second time slot is $v_s - v_2 - v_d$ with bandwidth 10, the maximum available fixed bandwidth spanning both time slots is 5, which results in the minimal data transfer end time $\frac{8}{5} = 1.6$ with the transfer start time at 0. For VPFB-1, the problem reduces to FPFB if no path switching is performed, and the data transfer end time is 1.8; if the paths are switched between these two time slots, the data transfer end

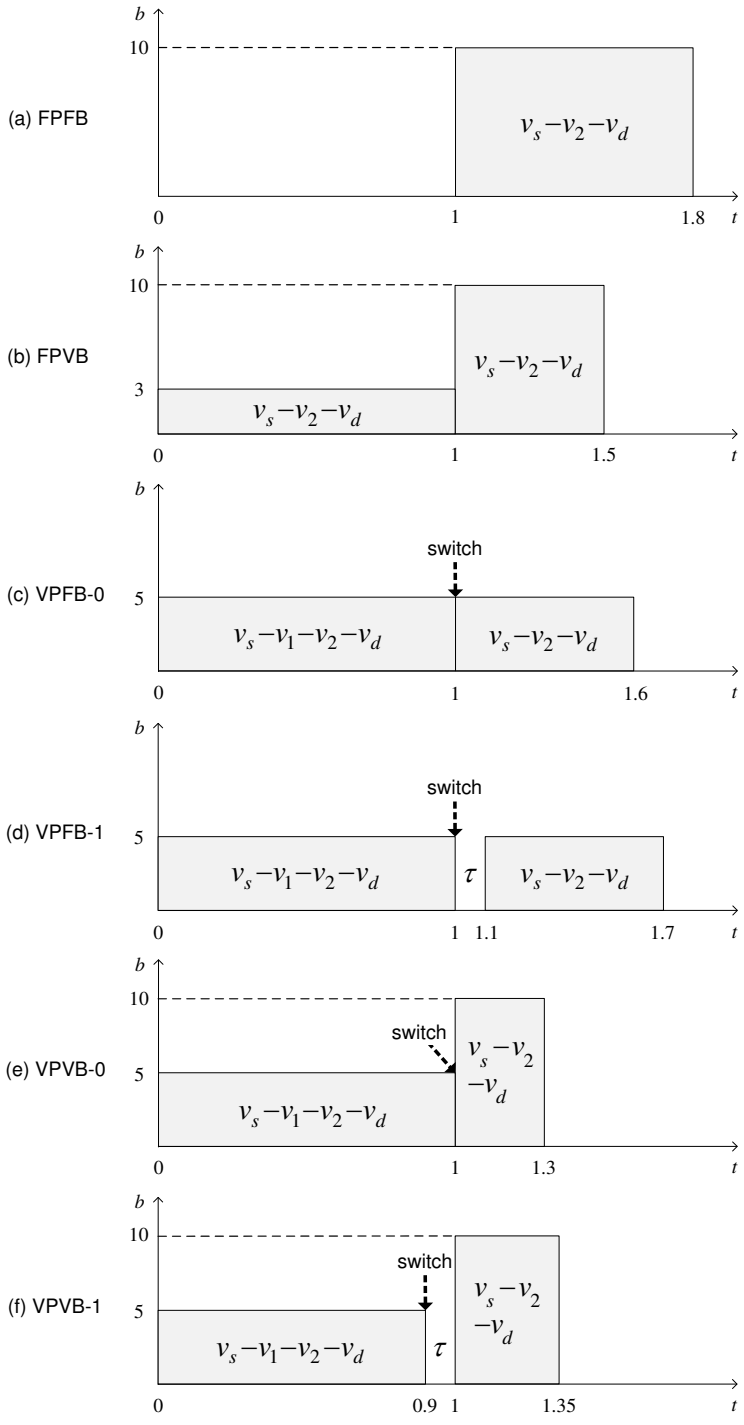

Figure 4.2: The optimal solutions to six bandwidth scheduling problems.

time of VPFB-1 is equal to the data transfer end time of VPFB-0 plus the path switching delay, i.e. 1.6 + τ = 1.7. Therefore, the path switching is profitable in this case. For VPVB-0, using the widest paths calculated for VPFB-0, it has the minimal data transfer end time $\frac{5}{5} + \frac{3}{10} = 1.3$. For VPVB-1, since the bandwidth of the widest path in the first time slot is smaller than that in the second time slot, the path switching is performed at the end of the first time slot (time interval [0.9,1]), and its data transfer end time is $1 + \frac{3.5}{10} = 1.35$.

Among these problems, FPFB represents the most stringent transport conditions by fixing both path and bandwidth, while VPVB is the most flexible transport scheme where the network resources can be fully utilized to achieve the minimum data transfer end time. Since FPFB and VPFB restrict the bandwidth to a fixed value during data transfer, it may not be always optimal to start data transfer immediately at the first possible time slot. These two schemes are mostly suited for transport methods that can stabilize at a fixed target rate such as FRTP [48], Tsunami [12], Hurricane [43], and PA-UDP [20]. FPVB and VPVB use variable bandwidths during transfer and therefore the transfer should always start immediately. These two schemes are particularly suited for transport methods that can dynamically adapt their source rates to different levels such as RAPID [14], SABUL [26], and RUNAT [42].

4.2 Complexity Analysis and Algorithm Design

Table 4.1 summarizes the computational complexities and optimal algorithms for the proposed six bandwidth scheduling problems. Since FPVB and VPVB-1 are NP-complete, their optimal algorithms are of exponential complexity and are only meant for small-scale networks. Furthermore, since FPVB and VPVB-1 can not be approximated unless P = NP,

we design heuristic algorithms with polynomial-time complexity for them in large-scale networks.

Table 4.1: Problem Complexities and Algorithms.

Problem	Complexity	Algorithm
FPFB	P	$OptFPFB$
FPVB	NP-complete	$OptFPVB$, $MinFPVB$
VPFB-0	P	$OptVPFB-0$
VPFB-1	P	$OptVPFB-1$
VPVB-0	P	$OptVPVB-0$
VPVB-1	NP-complete	$OptVPVB-1$, $MinVPVB-1$

A feasible solution to a bandwidth scheduling problem consists a number of scheduling components. Each scheduling component is a 4-tuple that consists a path from source to destination, the bandwidth along the path, the data transfer start time and end time. A feasible solution to FPFB has one scheduling component, while others may have more. For easy explanation, each scheduling algorithm described below only returns the data transfer end time, which is the optimization objective of these scheduling problems.

4.2.1 Optimal Scheduling Algorithm for FPFB

FPFB takes as input a graph $G = (V, E)$ with an ABT list for all links $l \in E$, source v_s and destination v_d, and data size δ, and computes a fixed path with a constant bandwidth from source to destination. We propose an optimal Complete Start Time Search algorithm for FPFB, referred to as $OptFPFB$, whose pseudocode is provided in Algorithm 1. The output of the algorithm is the minimal data transfer end time t_{end}. Since it may not be always optimal to start data transfer immediately, the algorithm varies the transfer start time slot p from 0 to q for a given data transfer end time slot q, and checks whether there exists any

feasible p such that the data of size δ can be transferred during the time slot range $[p,q]$ (i.e. time interval $[t[p],t[q+1]]$). If there does not exist any feasible path, the algorithm repeatedly increases q by 1; otherwise, the algorithm computes the optimal start time slot p and data transfer end time t_{end} by considering all possible p values and terminates. Here, the bandwidth β of the widest path can be calculated by an extended Dijkstra's algorithm. The algorithm guarantees that the returned data transfer end time is minimized since all possible transfer time slot ranges are examined. The time complexity of this algorithm is $O(T^2 \cdot m \cdot \lg n + T^3 \cdot m)$.

Algorithm 1 $OptFPFB(G, ATB, v_s, v_d, \delta)$

```
t_end = ∞;
for q = 0 to T − 1 do
  for p = 0 to q do
    for all l ∈ E do
      b_l = min_{p≤i≤q} (b_l[i]);
    end for
    β = bandwidth of the widest path from v_s to v_d during time slot range [p,q] based
    on b_l, ∀l ∈ E;
    if β · (t[q+1] − t[p]) ≥ δ, and t_end > t[p] + δ/β then
      t_end = t[p] + δ/β;
    end if
  end for
  if t_end < ∞ then
    break;
  end if
end for
return t_end.
```

An example for the Complete Start Time Search algorithm is shown in Fig. 4.3. First, we construct a complete start time slots table, where the cell in the first row only contains the first time slot. If the data cannot be completely transferred during the time interval $[t[0],t[1]]$, we increase data transfer end time slot q by 1 and advance to the second row,

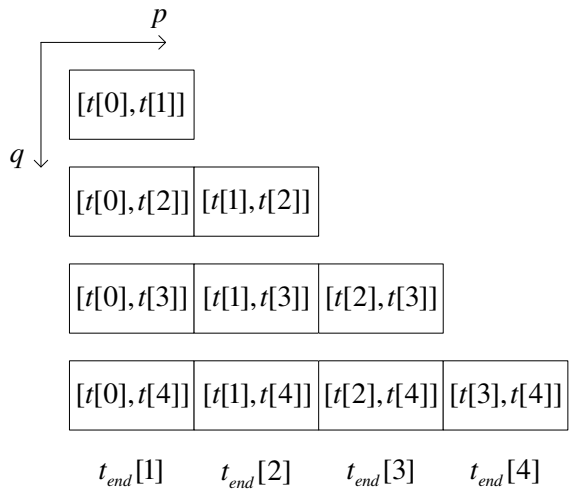

Figure 4.3: An example for the Complete Start Time Search algorithm.

where each cell considers the second time slot as the data transfer end time slot. If the data cannot be completely transferred before $t[2]$, we again increase q by 1 and advance to the third row. This search process continues until we reach a row where the data transfer request can be satisfied (i.e. $t[4]$). We try all possible start time points $t[0], t[1], t[2]$, and $t[3]$, and compute the corresponding data transfer end time $t_{end}[1], t_{end}[2], t_{end}[3]$ and $t_{end}[4]$, based on a comparison of which, we obtain the optimal solution with the minimal data transfer end time.

4.2.2 Optimal and Heuristic Scheduling Algorithms for FPVB

We first prove that FPVB is NP-complete and non-approximable, which indicates that there does not exist any polynomial-time optimal algorithm or approximate algorithm unless P = NP. We then propose an optimal algorithm of exponential complexity for small-scale networks and a heuristic algorithm for large-scale ones.

FPVB is NP-complete

We prove that FPVB is NP-complete by reducing from the *0-1 Total Bandwidth (0-1 TB)* problem, whose NP-completeness is shown in [27]. The decision version of FPVB is as follows: Given a graph $G = (E,V)$ with an ATB list for all links $l \in E$, source v_s and destination v_d, data size δ, does there exist a fixed path from v_s to v_d with varying bandwidths across multiple time slots such that the data can be completely transferred along the path during the time interval $[0,t_{end}]$? Without loss of generality, we suppose that the data transfer request is made at time point $t[0] = 0$.

Theorem 1. *FPVB is NP-complete.*

Proof. We first show that FPVB $\in$ NP. Given a solution (a path from v_s to v_d with a constant bandwidth) to FPVB, one can verify in polynomial time the validity of the solution by checking whether or not the data size transferred on the path during the time interval $[0,t_{end}]$ is greater than or equal to δ. This check obviously can be done in polynomial time.

We now reduce the 0-1 TB problem [27] to FPVB. The decision version of the 0-1 TB problem is defined as follows: Given a graph $G = (E,V)$ with an ATB list of either 0 or 1 available bandwidth for all links $l \in E$ at each time slot of unit length, source v_s and destination v_d, does there exist a path from v_s to v_d such that during the time interval $[0,t_{end}]$, the number of time slots in which all path links have a bandwidth value of 1 is at least β?

Let $(G,ATB,v_s,v_d,t_{end},\beta)$ be an arbitrary instance of 0-1 TB. We construct an instance $(G',ATB',v'_s,v'_d,t'_{end},\delta)$ of FPVB from the instance $(G,ATB,v_s,v_d,t_{end},\beta)$ in polynomial time such that the data of size δ can be completely transferred along a path from v'_s to v'_d

during the time interval $[0, t'_{end}]$, if and only if there exists a path from v_s to v_d in G that the number of time slots in which all path links have a bandwidth value of 1 is at least β during the time interval $[0, t_{end}]$. We set $G' = G$, $TB' = TB$, $v'_s = v_s$, $v'_d = v_d$, $t'_{end} = t_{end}$, and $\delta = \beta$. Clearly, this instance construction can be done in polynomial time.

Suppose that there exists a path P from v_s to v_d in G that the number of time slots in which all path links have a bandwidth of 1 is at least β during interval $[0, t_{end}]$. We can find a corresponding path P' in G', where $P' = P$. Since one can transfer 1 unit of data in each unit time slot with a bandwidth of 1, the accumulated data size transferred along P' during the entire time interval is at least β. Therefore, the data of size $\delta = \beta$ can be completely transferred during the time interval $[0, t'_{end}]$ along P'. Hence, P' composes a solution to FPVB.

Conversely, let P' be the path from v'_s to v'_d and the data of size δ can be completely transferred along the path during the time interval $[0, t'_{end}]$. We can find a corresponding path P in G, where $P = P'$. Obviously, the number of time slots in which all path links have a bandwidth value of 1 is at least $\beta = \delta$ during the time interval $[0, t_{end}]$, since transferring 1 unit of data requires one unit time slot in which all path links have a bandwidth value of 1. Hence, P composes a solution to 0-1 TB. This concludes the proof.

FPVB is Non-approximable

Theorem 2. *For any polynomial-time computable function $f(n,m)$, FPVB cannot be approximated within an approximation ratio of $f(n,m)$, unless P = NP.*

Proof. Assume that there exists an approximate algorithm with an approximation ratio of $f(n,m)$ for FPVB. We show that this assumption implies a polynomial-time optimal algorithm for the 0-1 TB problem [27].

Let $(G, ATB, v_s, v_d, t_{end}, \beta)$ be an arbitrary instance of the 0-1 TB decision problem. We construct a corresponding instance $(G', ATB', v'_s, v'_d, \delta)$ of FPVB in polynomial time by setting $G' = G$, $v'_s = v_s$, $v'_d = v_d$ and $\delta = \beta$. ATB' consists of $t_{end} \cdot (f(n,m)+1)$ time slots, which is divided into three segments as shown in Fig. 4.4. The available bandwidth for all links in ATB' is the same as that in ATB during time interval $[0, t_{end}]$, while the available bandwidth for all links in ATB' is 0 during time interval $[t_{end}, t_{end} \cdot f(n,m)]$ and is 1 during time interval $[t_{end} \cdot f(n,m), t_{end} \cdot (f(n,m)+1)]$.

We apply the approximate algorithm to the FPVB instance $(G', ATB', v'_s, v'_d, \delta)$. Obviously, the approximate algorithm can find a path from v'_s to v'_d such that the data of size δ can be completely transferred along the path during time interval $[0, t_{end} \cdot f(n,m)]$ if and only if there exists a solution to the 0-1 TB instance, since the size of data transferred during time interval $[t_{end}, t_{end} \cdot f(n,m)]$ is 0 in the FPVB instance. Therefore, an $f(n,m)$ approximate algorithm finds a solution to the 0-1 TB problem whenever one exists. This conflicts with the NP-completeness of the 0-1 TB problem. Proof ends.

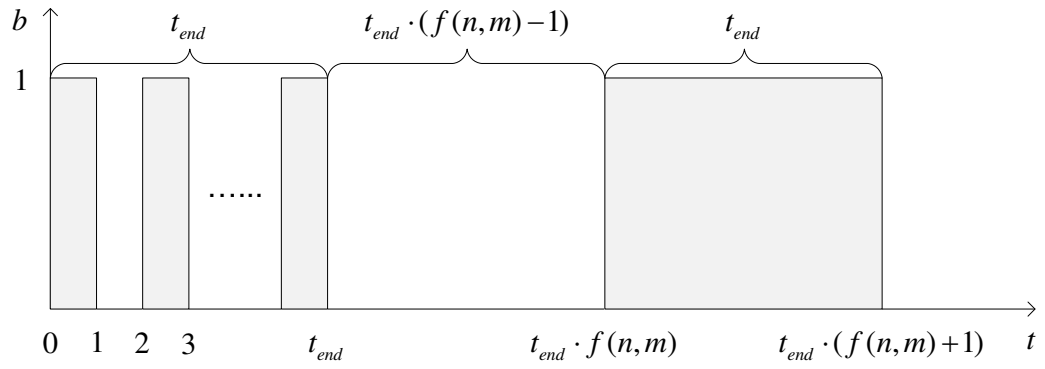

Figure 4.4: ATB' construction for an FPVB instance.

Optimal Scheduling Algorithm for FPVB

With the same input of FPFB, we propose an optimal algorithm for FPVB, referred to as $OptFPVB$, which guarantees the global minimization of the data transfer end time. Ganguly *et al.* proposed an optimal routing algorithm for data transfer in time-varying networks [22] with time slots of equal size, and their objective is to minimize the number of time slots required for data transfer. We first develop an extension of this algorithm, named Maximum Permutation Algorithm (*MPA*), which serves as the base function of $OptFPVB$ for FPVB. Taking as input a graph $G = (V,E)$ with an ATB list for all links $l \in E$, source v_s and destination v_d, data size δ, and a time slot range $[p,q]$, *MPA* determines if there is a path from v_s to v_d such that data of size δ can be transferred within the time slot range $[p,q]$.

To facilitate the explanation of *MPA*, we define several notations and operations as follows:

$E(p,\beta)$**:** a subset of E consisting of links whose residual bandwidths in time slot p are less than β.

$G' = G - E(p,\beta)$**:** the operation of removing the links in $E(p,\beta)$ from G and producing a new graph G'.

Q_p**:** a queue storing the bandwidths of links $l \in E$ sorted in a decreasing order for time slot p.

The pseudocode of *MPA* is shown in Algorithm 2, whose output is the minimum data transfer end time t_{end}. If t_{end} is equal to the initial value ∞, it implies that no path exists that

Algorithm 2 $MPA(G, ATB, v_s, v_d, \delta, p, q)$

```
t_end = ∞;
while Q_p ≠ ∅ do
  β = dequeue(Q_p);
  G' = G − E(p, β);
  if ∃ path P from v_s to v_d in G' then
    if β · (t[p+1] − t[p]) ≥ δ then
      t_end = t[p] + δ/β;
      break;
    else if p ≡ q then
      break;
    else
      δ' = δ − β · (t[p+1] − t[p]);
      p' = p + 1;
      t'_end = MPA(G', ATB, v_s, v_d, δ', p', q);
      if t_end > t'_end then
        t_end = t'_end;
      end if
    end if
  end if
end while
return t_end.
```

can transfer data of size δ in the time slot range of $[p,q]$. The algorithm starts from time slot p and recursively calls itself by modifying the network G, advancing to the next time slot and adjusting the residual data size. Once the links with residual bandwidths less than β are removed from G, and there exists a path P from v_s to v_d in the remaining network G', the bandwidth of path P is at least β. For the current time step p, we consider three cases: (i) if the bandwidth of path P is greater than or equal to the residual data size, the algorithm computes t_{end} and finishes successfully; (ii) otherwise if $p \equiv q$, the algorithm fails to find a feasible path; (iii) otherwise, the algorithm calls itself after increasing p by 1 and updating the residual data size δ. The algorithm examines all possible permutations of bandwidths at different time slots to obtain the minimum data transfer end time. In the worst case, *MPA* makes m^{q-p} recursive calls, during each of which, the runtime is dominated by the

operation $G' = G - E(p, \beta)$ and the computation to decide whether there exists a path from v_s to v_d in G' via breadth first search. The runtime of each recursion is $O(m)$, and the total computational complexity of *MPA* is $O(m \cdot m^{q-p})$, or $O(m^{q-p+1})$.

Algorithm 3 $OptFPVB(G, ATB, v_s, v_d, \delta)$

$t_{min} = OptVPVB-0(G, ATB, v_s, v_d, \delta)$;
$t_{max} = MinFPVB(G, ATB, v_s, v_d, \delta)$;
Compute the corresponding time slots of t_{min} and t_{max}, which are referred as q_{min} and q_{max}, respectively;
while $q_{min} \leq q_{max}$ **do**
  $q_{med} = \lfloor \frac{q_{min}+q_{max}}{2} \rfloor$;
  $t_{end} = MPA(G, ATB, v_s, v_d, \delta, 0, q_{med})$;
  if $t_{end} < \infty$ **then**
    $q_{max} = q_{med}$;
  else
    $q_{min} = q_{med} + 1$;
  end if
end while
return t_{end}.

The pseudocode of the optimal algorithm *OptFPVB* that calls *MPA* to compute the minimum data transfer end time is shown in Algorithm 3. We first define the possible data transfer end time interval as $[t_{min}, t_{max}]$, where t_{min} and t_{max} denote the minimum and maximum data transfer end time required to transfer data of size δ from v_s to v_d, respectively. Note that the lower bound t_{min} can be computed by calling $OptVPVB-0$ we propose for VPVB-0 in Algorithm 8 in Subsection 4.2.5, because VPVB-0 represents the most flexible scheduling scenario and its minimal data transfer end time must be less than or equal to that of FPVB; while the upper bound t_{max} can be simply computed by calling *MinFPVB*, the heuristic algorithm we propose for FPVB in Algorithm 5. We then compute the corresponding time slots of t_{min} and t_{max} according to the *ATB* list, and denote them as q_{min} and q_{max}, respectively. Thus, the minimum end time slot required for data transfer along the

optimal path must fall within the time slot range $[q_{min}, q_{max}]$. We finally conduct a binary search in this time slot range by calling *MPA* to find the minimum data transfer end time.

Since *OptFPVB* calls *MPA* at most $O(\lg T)$ times, the total complexity of *OptFPVB* is $O(m^T \cdot \lg T)$. Note that this optimal algorithm may result in prohibitively long runtime when T is large. For large T but small n, we may employ a brute-force approach by exhausting all permutations and combinations of $n-2$ nodes between v_s and v_d, whose complexity is $O(T \cdot (n-1)!)$. When both T and n are relatively large, heuristic algorithms are needed for practical use.

Heuristic Scheduling Algorithm for FPVB

We propose an efficient heuristic algorithm with polynomial-time complexity for FPVB, and also design a naive greedy algorithm for performance comparison. Again, we define several notations and operations to facilitate our explanation:

$t_{end}[v]$**:** transfer end time for data of size δ from v_s to v along the computed path.

Q**:** a queue of nodes sorted by their data transfer end time $t_{end}[v]$ in an increasing order.

$b_{(v_s,v)}[i]$**:** the bandwidth of the computed path from v_s to v in the i-th time slot, which is the bottleneck bandwidth of all component links on the path.

We propose a greedy approach based on Dijkstra's algorithm, *GreedyFPVB*, whose pseudocode is shown in Algorithm 4. The algorithm first computes the data transfer end time for data of size δ over every single link, and assigns the data transfer end time as a weight to each link. It then computes the narrowest path in the updated graph by extending Dijkstra's algorithm. Here, the narrowest path is defined as the path from the source node

to the destination node on which the maximum weight among all component links is minimized. The total computational complexity of this greedy approach is $O(m \cdot (T + \lg n))$.

Algorithm 4 *GreedyFPVB*$(G, ATB, v_s, v_d, \delta)$

```
for all (u,v) ∈ E do
   t_end[(u,v)] = transfer end time for data of size δ from u to v (or v to u) along link (u,v)
   with varying bandwidth;
end for
for all v ∈ V do
   t_end[v] = ∞;
end for
t_end[v_s] = 0;
Q = V;
while Q ≠ ∅ do
   u = dequeue(Q);
   if u ≡ v_d then
      break;
   end if
   for all v ∈ Q, (u,v) ∈ E do
      if t_end[v] > max(t_end[u], t_end[(u,v)]) then
         t_end[v] = max(t_end[u], t_end[(u,v)]);
      end if
   end for
end while
return t_end[v_d].
```

This greedy approach does not consider coordinating the component links on the computed path. Note that the available bandwidth of the path is determined by the bottleneck bandwidth of all component links at each time slot. We further propose another heuristic algorithm called *MinFPVB* to address this issue by keeping track of the bottleneck bandwidth of the path, as shown in Algorithm 5. At each relaxation step, the bottleneck bandwidth from v_s to the current node $b_{(v_s,v)}[i]$ is updated. The total computational complexity of *MinFPVB* is $O(m \cdot (T + \lg n))$.

Algorithm 5 $MinFPVB(G, ATB, v_s, v_d, \delta)$

for all $v \in V$ **do**
  $t_{end}[v] = \infty$;
end for
$t_{end}[v_s] = 0$;
$Q = V$;
while $Q \neq \emptyset$ **do**
  $u = dequeue(Q)$;
  if $u \equiv v_d$ **then**
    break;
  end if
  for all $v \in Q, (u,v) \in E$ **do**
    if $u \equiv v_s$ **then**
      $b'_{(v_s,v)}[i] = b_{(u,v)}[i], \forall i \in [0, T-1]$;
    else
      $b'_{(v_s,v)}[i] = min(b_{(v_s,u)}[i], b_{(u,v)}[i]), \forall i \in [0, T-1]$;
    end if
    Compute the data transfer end time t'_{end} of data of size δ from v_s to v with available bandwidth $b'_{(v_s,v)}$;
    if $t_{end}[v] > t'_{end}$ **then**
      $t_{end}[v] = t'_{end}$;
      $b_{(v_s,v)}[i] = b'_{(v_s,v)}[i], \forall i \in [0, T-1]$;
    end if
  end for
end while
return $t_{end}[v_d]$.

4.2.3 Optimal Scheduling Algorithm for VPFB-0

VPFB-0 is to compute a set of paths from source to destination at different time slots with a fixed bandwidth and the path switching delay between two adjacent time slots is negligible ($\tau = 0$). We propose an optimal algorithm for VPFB-0, referred to as $OptVPFB-0$, which is also based on the Complete Start Time Search. Note that the path bandwidth are not specified by the user, and it may not be always optimal to start the transfer immediately.

The pseudocode of $OptVPFB-0$ is shown in Algorithm 6. For a give data transfer end time slot q starting from 0, the algorithm varies the transfer start time slot p from 0

Algorithm 6 $OptVPFB-0(G,ATB,v_s,v_d,\delta)$

$t_{end}=\infty$;
for $q=0$ to $T-1$ **do**
  for $p=0$ to q **do**
    β_i = bandwidth of the widest path from v_s to v_d in time slot i, $i\in[p,q]$;
    $\beta=\min_{p\le i\le q}(\beta_i)$;
    if $\beta\cdot(t[q+1]-t[p])\ge\delta$, and $t_{end}>t[p]+\delta/\beta$ **then**
      $t_{end}=t[p]+\delta/\beta$;
    end if
  end for
  if $t_{end}<\infty$ **then**
    break;
  end if
end for
return t_{end}.

to q and computes the bandwidth of the widest path from v_s to v_d in each time slot within the time slot range $[p,q]$. It then computes the minimal bandwidth, which is the bottleneck bandwidth across these time slots and considered as the fixed bandwidth for data transfer. The algorithm repeatedly increases q by 1 until the amount of data transferred up to time slot q is greater than the data size δ, and computes the minimal data transfer end time by considering all possible p values. The widest path calculation in each time slot takes $O(T\cdot m\cdot\lg n)$, which is performed in advance. The total time complexity of this algorithm is $O(T\cdot m\cdot\lg n+T^3)$.

4.2.4 Optimal Scheduling Algorithm for VPFB-1

VPFB-1 considers a constant non-negligible positive path switching delay $\tau>0$. Since data transfer is suspended during the period of path switching, it may not be always beneficial to perform path switching between two adjacent time slots. In the extreme case where τ is sufficiently large, any path switching causes a negative impact on the performance, and

therefore VPFB-1 reduces to FPFB. We propose an optimal algorithm using dynamic programming for VPFB-1, referred as $OptVPFB-1$. Without loss of generality, we assume that τ is smaller than the length of any time slot in the ATB list.

To employ a dynamic programming procedure, we need to characterize the structure of an optimal solution. Let us consider whether the data of size δ can be completely transferred during the time slot range $[p,q]$ with k path switchings. Let $\beta[p,q,k]$ be the maximum available bandwidth during the time slot range $[p,q]$ with k path switchings. The maximum amount of transferred data during the time slot range $[p,q]$ with k path switchings is:

$$\beta[p,q,k] \cdot (t[q+1] - t[p] - \tau \cdot k). \tag{4.2.1}$$

A path switching can be scheduled either at the end of a time slot or at the beginning of the next time slot. If the above maximum amount of transferred data is greater than or equal to the data size δ, we obtain the data transfer end time:

$$t_{end} = t[p] + \delta/\beta[p,q,k] + \tau \cdot k. \tag{4.2.2}$$

Obviously, VPFB-1 boils down to the problem of computing $\beta[p,q,k]$, i.e. how to distribute k path switchings during the time slot range $[p,q]$ such that the constant available bandwidth during this time slot range is maximized? One may exhaust all possible path switching distributions to obtain $\beta[p,q,k]$, but the number of such distributions is C^k_{q-p}, which is exponential.

Following the concept of dynamic programming, we now define the recursive form of the optimal solutions to subproblems. If we divide the problem of computing $\beta[p,q,k]$ by the time slot after which the first path switching is scheduled, $\beta[p,q,k]$ can be computed

from the optimal solutions to its $q-p-k+1$ subproblems:

$$\beta[p,q,k] = \max_{p \le i \le q-k} \{min(\beta[p,i,0], \beta[i+1,q,k-1])\}, \tag{4.2.3}$$

where $\beta[p,i,0]$ is the maximum bandwidth during the time slot range $[p,i]$ without any path switching, which can be computed by the modified Dijkstra's algorithm, and $\beta[i+1,q,k-1]$ is the maximum bandwidth during the time slot range $[i+1,q]$ with $k-1$ path switchings, which is the optimal solution to a subproblem.

Algorithm 7 $OptVPFB-1(G, ATB, v_s, v_d, \delta, \tau)$

```
t_end = ∞;
for q = 0 to T − 1 do
  for k = 0 to q do
    for p = 0 to q − k do
      if k ≡ 0 then
        for all l ∈ E do
          b_l = min_{p≤i≤q} (b_l[i]);
        end for
        β[p,q,0] = bandwidth of the widest path from v_s to v_d during time slot range
        [p,q] based on link bandwidth b_l, ∀l ∈ E;
      else
        β[p,q,k] = max_{p≤i≤q−k} {min(β[p,i,0], β[i+1,q,k−1])};
      end if
      if β[p,q,k] · (t[q+1] − t[p] − τ · k) ≥ δ and t_end > t[p] + δ/β[p,q,k] + τ · k then
        t_end = t[p] + δ/β[p,q,k] + τ · k;
      end if
    end for
  end for
  if t_end < ∞ then
    break;
  end if
end for
return t_end;
```

The pseudocode of $OptVPFB-1$ based on dynamic programming is shown in Algorithm 7. The data transfer end time slot q starts from 0, and for a given q, the algorithm computes $\beta[p,q,0]$ as base conditions when $k=0$ and recursively computes $\beta[p,q,k]$ based

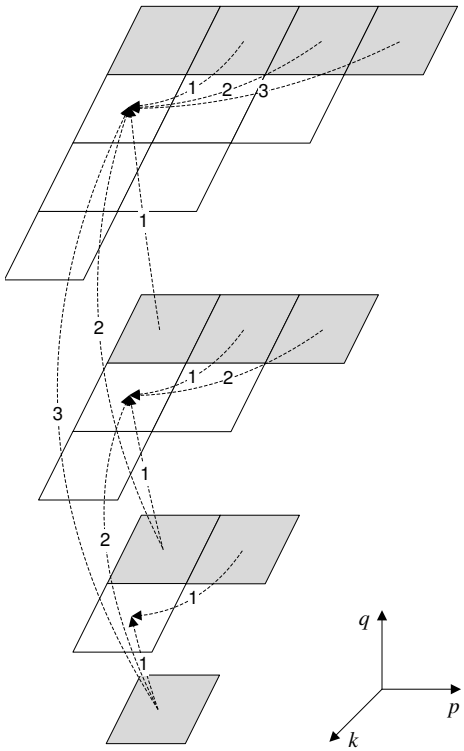

Figure 4.5: A dynamic programming procedure that computes $\beta[p,q,k]$ using a tabular, bottom-up approach.

on Eq. 4.2.3 when $k > 0$. $\beta[p,q,k]$ is computed using a tabular, bottom-up approach, as shown in Fig. 4.5, where the values of $\beta[p,q,k]$ are stored in a 3-dimensional table, and the shadowed entries in the table represent the optimal solutions to base conditions when $k = 0$. The curved lines indicate which entries in the table are used for computing $\beta[p,q,k]$. Two start entries of two curved lines with the same indicators (e.g., '1', '2', '3') ending at the same end entry are a pair of $\beta[p,i,0]$ and $\beta[i+1,q,k-1]$ optimal solutions used to compute the end entry of the two curved lines. The dynamic programming procedure repeatedly increases q by 1 and considers all possible p and k values to compute the amount of data transfer defined in Eq. 4.2.1 until it is greater than or equal to the data size δ, in which case the minimal t_{end} is updated. The time for computing the optimal solutions to

all base entries of the table (when $k = 0$) is $O(T^2 \cdot m \cdot \lg n)$, and the time for computing the other entries of the table (when $k > 0$) is $O(T^4)$. Therefore, the total time of the algorithm is $O(T^2 \cdot (m \cdot \lg n + T^2))$ in the worst case.

Note that if τ is not always smaller than the length of any time slot in the ATB list, we only need to change Eq. 4.2.3 to:

$$\beta[p,q,k] = \max_{p \le i \le q-k} \{min(\beta[p,i,0], \beta[j,q,k-1]) \mid j \le q\}, \tag{4.2.4}$$

where j is the time slot where the time point $t[i] + \tau$ is located.

4.2.5 Optimal Scheduling Algorithm for VPVB-0

VPVB-0 is to compute a set of paths from v_s to v_d in different time slots with varying bandwidths across all time slots to minimize the data transfer end time. Similar to VPFB-0, the path switching delay is negligible ($\tau = 0$). We propose an optimal algorithm for VPVB-0 based on an extension of Dijkstra's algorithm, referred to as $OptVPVB-0$.

As shown in Algorithm 8, $OptVPVB-0$ repeatedly calls the extended Dijkstra's algorithm to compute the bandwidth of the widest path in each time slot and adjusts the residual data size. The terminating condition of the **while** loop is met when the residual data size is less than the amount of data that can be transferred over the widest path in the current time slot. Assuming that the data are completely transferred in k time slots, the maximum number of path switchings needed is $k-1$, in which case the widest paths in any two adjacent time slots are different. The runtime of the extended Dijkstra's algorithm is $O(m \cdot \lg n)$, and the runtime of $OptVPVB-0$ is $O(T \cdot m \cdot \lg n)$ in the worst case.

Algorithm 8 $OptVPVB-0(G,ATB,v_s,v_d,\delta)$

$i = 0$;
while $\delta > 0$ **do**
  β = bandwidth of the widest path from v_s to v_d in time slot i;
  if $\delta \leq \beta \cdot (t[i+1] - t[i]$ **then**
    $t_{end} = t[i] + \delta/\beta$;
  else
    $i = i + 1$;
  end if
  $\delta = \delta - \beta \cdot (t[i+1] - t[i])$;
end while
return t_{end}.

4.2.6 Optimal and Heuristic Scheduling Algorithms for VPVB-1

VPVB-1 considers a constant positive path switching delay $\tau > 0$. When τ is so large that any path switching causes a negative impact on the performance, VPVB-1 reduces to FPVB. We first prove that VPVB-1 is NP-complete and non-approximable, and then propose an optimal algorithm with exponential complexity for small-scale networks and a heuristic algorithm for large-scale ones.

VPVB-1 is NP-complete

We prove that VPVB-1 is NP-complete by showing that FPVB is a special case of VPVB-1.

Theorem 3. *VPVB-1 is NP-complete.*

Proof. We restrict VPVB-1 to FPVB by only allowing those instances in which $\tau \geq \delta/min(b_l[i])$, where $l \in E$, $i \in [0, T-1]$. Since the path switching delay is sufficiently large compared to the length of time slots, performing any path switching would result in a worse data transfer end time. The validity of NP-completeness proof by restriction is established in [23], where "restriction" constrains the given, not the question of a problem. Since FPVB is NP-complete, so is VPVB-1. Proof ends.

VPVB-1 is Non-approximable

Theorem 4. *For any polynomial-time computable function $f(n,m)$, VPVB-1 cannot be approximated within an approximation ratio of $f(n,m)$, unless $P = NP$.*

Proof. Assume that there exists an approximate algorithm with an approximation ratio of $f(n,m)$ for VPVB-1. We show that this assumption implies a polynomial-time optimal algorithm for FPVB.

Let $(G, ATB, v_s, v_d, t_{end}, \delta)$ be an arbitrary instance of the FPVB decision problem. We construct an instance $(G', ATB', v'_s, v'_d, \delta', \tau)$ of the VPVB-1 optimization problem from the instance $(G, ATB, v_s, v_d, t_{end}, \delta)$ in polynomial time by setting $G' = G$, $v'_s = v_s$, $v'_d = v_d$, $\delta' = \delta$, $\tau = t_{end} \cdot f(n,m)$. The construction of ATB' is the same as that described in the proof for Theorem 2.

We apply the approximate algorithm to the VPVB-1 instance $(G', ATB', v'_s, v'_d, \delta', \tau)$. Obviously, the approximate algorithm can find a path without any path switching from v'_s to v'_d such that the data of size δ' can be completely transferred along the path during the time interval $[0, t_{end} \cdot f(n,m)]$ if and only if there exists a solution to the FPVB instance, since the size of data transferred during the time interval $[t_{end}, t_{end} \cdot f(n,m)]$ is 0, and the path switching delay is sufficiently large such that no path switching is performed in the VPVB-1 instance. Therefore, a $f(n,m)$ approximation algorithm finds a solution to FPVB whenever one exists. This conflicts with the NP-completeness of FPVB. Proof ends.

Optimal Scheduling Algorithm for VPVB-1

Similar to the optimal algorithm $OptVPFB-1$ to VPFB-1, we propose an optimal algorithm by combining dynamic programming and $OptFPVB$ for VPVB-1, referred to as $OptVPVB-1$.

A path switching can be performed either at the end of a time slot or at the beginning of the next time slot. Different time points for path switching may lead to different amounts of data transfer because the path bandwidths may be different in two adjacent time slots. Let $g_1[p,q,k]$, $g_2[p,q,k]$, $g_3[p,q,k]$ and $g_4[p,q,k]$ be the maximum amount of data transferred along the optimal FPVB paths with k path switchings in the time interval $[t[p]+\tau, t[q+1]-\tau]$, $[t[p]+\tau, t[q+1]]$, $[t[p], t[q+1]-\tau]$ and $[t[p], t[q+1]]$, respectively. An example of the data transfer in these four path switching schemes is shown in Fig. 4.6, where each time slot is of unit length and the bandwidth of the widest path varies across the three time slots. We compute $g_i[1,1,0]$ $(1 \leq i \leq 4)$, which is the amount of data transfer in the time slot 1 (dark grey rectangle) in four different time intervals due to the different path switchings schemes. In Fig. 4.6 (a), the path bandwidth in time slot 1 is less than that in time slot 0 and 2, therefore, both path switchings are performed in time slot 1. The path switchings in Fig. 4.6 (b) (c) (d) are performed in a similar way.

Consider the case where the data of size δ can be completely transferred during the time slot range $[0,q]$ with k path switchings. If the amount of data $g_4[0,q,k]$ transferred during the time interval $[t[0], t[q+1]]$ is greater than or equal to the data size δ, we can compute the corresponding data transfer end time t_{end}, which is in time slot q. Hence, VPVB-1 boils down to the problem of computing $g_4[0,q,k]$, i.e. how to distribute k path switchings during the time slot range $[0,q]$ such that the amount of data transferred during this time slot range is maximized?

Dividing the problem of computing $g_i[p,q,k]$ by the time slot after which the first path switching is performed, we compute $g_i[p,q,k]$ from the optimal solutions to its $q-p-k+1$ subproblems:

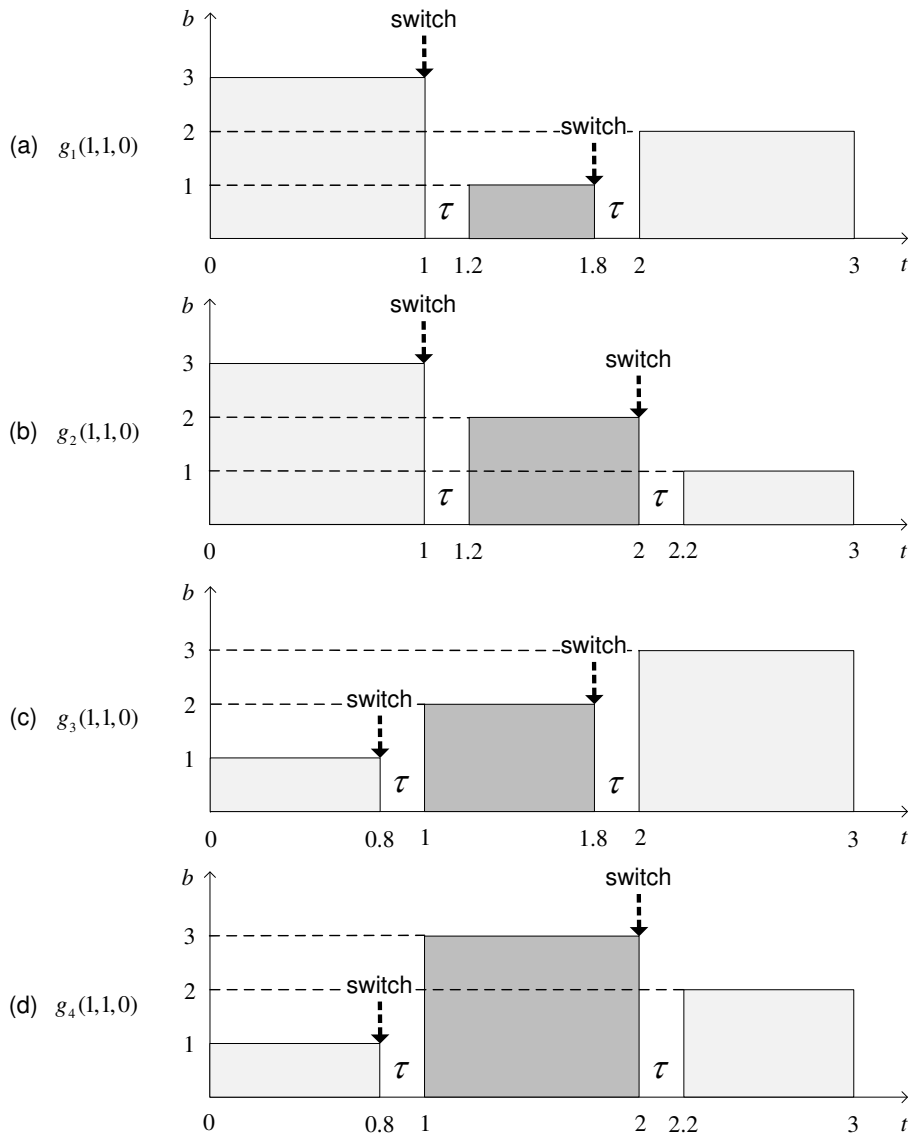

Figure 4.6: The data transfer in different path switching schemes.

$$
\begin{aligned}
g_1[p,q,k] &= \max_{p\le i\le q-k}\{max(g_1[p,i,0]+g_3[i+1,q,k-1],\\
&\qquad g_2[p,i,0]+g_1[i+1,q,k-1])\}\\
g_2[p,q,k] &= \max_{p\le i\le q-k}\{max(g_2[p,i,0]+g_2[i+1,q,k-1],\\
&\qquad g_1[p,i,0]+g_4[i+1,q,k-1])\}\\
g_3[p,q,k] &= \max_{p\le i\le q-k}\{max(g_3[p,i,0]+g_3[i+1,q,k-1],\\
&\qquad g_4[p,i,0]+g_1[i+1,q,k-1])\}\\
g_4[p,q,k] &= \max_{p\le i\le q-k}\{max(g_4[p,i,0]+g_2[i+1,q,k-1],\\
&\qquad g_3[p,i,0]+g_4[i+1,q,k-1])\}
\end{aligned}
\tag{4.2.5}
$$

In the first recursive equation in Eq. 4.2.5, $g_1[p,i,0]$ and $g_2[p,i,0]$ denote the amount of data transferred during the time interval $[t[p]+\tau,t[i+1]-\tau]$ and $[t[p]+\tau,t[i+1]]$ without any path switchings, respectively, which can be computed by the *MPA* algorithm designed for the FPVB problem by changing the output of *MPA* to the maximal amount of transferred data instead of the minimal transfer end time. Similarly, $g_3[i+1,q,k-1]$ and $g_1[i+1,q,k-1]$ denote the amount of the data transferred during the time interval $[t[i+1],t[q+1]-\tau]$ and $[t[i+1]+\tau,t[q+1]-\tau]$ with $k-1$ path switchings, respectively, which are the optimal solutions to subproblems. The first path switching is performed at the end of time slot i when $g_1[p,i,0]+g_3[i+1,q,k-1]$ is larger than $g_2[p,i,0]+g_1[i+1,q,k-1]$; otherwise, it is performed at the beginning of time slot $i+1$. The other recursive equations can be constructed in a similar way.

The pseudocode of $OptVPVB-1$ is shown in Algorithm 9. The data transfer end time slot q starts from 0, and for a given q, the algorithm computes $g_i[p,q,0]$ as base

Algorithm 9 $OptVPVB-1(G,ATB,v_s,v_d,\delta,\tau)$

$t_{end}=\infty$;
for $q=0$ to $T-1$ **do**
  for $k=0$ to q **do**
    for $p=0$ to $q-k$ **do**
      if $k\equiv 0$ **then**
        Compute $g_1[p,q,0]$, $g_2[p,q,0]$, $g_3[p,q,0]$, $g_4[p,q,0]$, which are the maximum amount of data transferred along the optimal FPVB paths without any path switching in the time interval $[t[p]+\tau,t[q+1]-\tau]$, $[t[p]+\tau,t[q+1]]$, $[t[p],t[q+1]-\tau]$ and $[t[p],t[q+1]]$, respectively;
      else
        $g_1[p,q,k]=\max\limits_{p\le i\le q-k}\{max(g_1[p,i,0]+g_3[i+1,q,k-1],g_2[p,i,0]+g_1[i+1,q,k-1])\}$;
        $g_2[p,q,k]=\max\limits_{p\le i\le q-k}\{max(g_2[p,i,0]+g_2[i+1,q,k-1],g_1[p,i,0]+g_4[i+1,q,k-1])\}$;
        $g_3[p,q,k]=\max\limits_{p\le i\le q-k}\{max(g_3[p,i,0]+g_3[i+1,q,k-1],g_4[p,i,0]+g_1[i+1,q,k-1])\}$;
        $g_4[p,q,k]=\max\limits_{p\le i\le q-k}\{max(g_4[p,i,0]+g_2[i+1,q,k-1],g_3[p,i,0]+g_4[i+1,q,k-1])\}$;
      end if
    end for
    if $g_4[0,q,k]\ge\delta$ **then**
      Compute the data transfer end time t'_{end} for data of size δ using $g_4[0,q,k]$ scheduling scheme;
      if $t_{end}>t'_{end}$ **then**
        $t_{end}=t'_{end}$
      end if
    end if
  end for
  if $t_{end}<\infty$ **then**
    break;
  end if
end for
return t_{end}.

conditions when $k = 0$ and recursively computes $g_i[p,q,k]$ based on Eq. 4.2.5 when $k > 0$. The algorithm repeatedly increases q by 1 and considers all possible k values to compute the amount of data transfer defined in Eq. 4.2.5 until $g_4[0,q,k] \geq \delta$, in which case the minimal t_{end} is updated. Since the time complexity of *MPA* is m^{q-p+1}, the time for computing the optimal solutions to all base entries of the table (when $k = 0$) is $O(m^T)$. The time for computing the other entries of the table (when $k > 0$) is $O(T^4)$. Therefore, the total time of the algorithm is $O(m^T + T^4)$ in the worst case. Note that the algorithm has a high time complexity with large T, in which case heuristic algorithms are needed for practical use.

Heuristic Scheduling Algorithm for VPVB-1

We propose a heuristic algorithm with polynomial-time complexity for VPVB-1, referred to as $MinVPVB-1$. Similar to $OptVPVB-1$ shown in Algorithm 9, $MinVPVB-1$ also employs dynamic programming and the same recursions in Eq. 4.2.5 to compute $g_i[p,q,k]$. For the base conditions (when $k = 0$), $MinVPVB-1$ computes $g_i[p,q,0]$ by using the *MinFPVB* heuristic instead of the *MPA* optimal algorithm, by changing the input of *MinFPVB* to (G, ATB, v_s, v_d, p, q) and the output of *MinFPVB* to the maximal amount of data transfer during time slot range $[p,q]$. *MinFPVB* in Algorithm 5 is to compute a path from v_s to v_d to minimize the transfer end time for a given data size, which is essentially the same as computing a path from v_s to v_d to maximize the transferred data size within a given time range. Therefore, the total time of the algorithm is polynomial, which is $O(T^2 \cdot m \cdot (T + \lg n) + T^4)$ in the worst case.

We design a greedy algorithm named $GreedyVPVB-1$ for performance comparison. Again, we define several notations and operations to facilitate our explanation:

$s(i)$**:** the time slot of the most recent path switching considered in time slot i.

$P^f_{[s(i),i]}$**:** the FPVB path from v_s to v_d that maximizes the data transfer within time slot range $[s(i), i]$.

$g(P_i, i)$**:** the amount of data transferred on path P_i in time slot i.

$g(P^f_{[s(i-1),i-1]}, [s(i-1), i-1])$**:** the amount of data transferred on path $P^f_{[s(i-1),i-1]}$ within time slot range $[s(i-1), i-1]$.

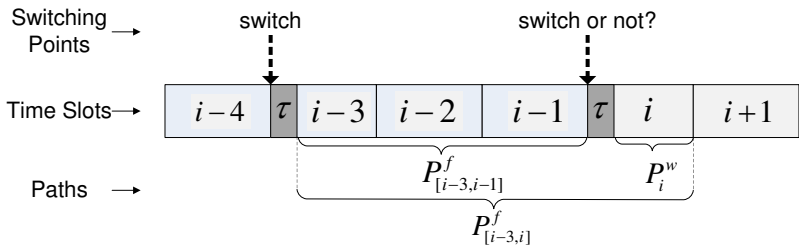

Figure 4.7: Illustration of the $GreedyVPVB-1$ algorithm.

$GreedyVPVB-1$ also calls the $MinFPVB$ heuristic to compute the amount of data transferred along the FPVB path during a given time range. Due to the varying available bandwidths in the future time slots, path switchings allow data to be transferred along the widest path in each time slot to improve the utilization of bandwidths. On the other hand, path switchings incur additional time overhead. We must take both the positive and negative effects of path switchings into consideration, and make a decision on whether a path switching between two adjacent time slots is worthwhile. The pseudocode of the algorithm is shown in Algorithm 10. We assume that the data can not be completely transferred in the first time slot. When moving to time slot i, the algorithm makes a decision on whether

to switch to the widest path in time slot i or stick to the FPVB path computed for time slots from $s(i-1)$ to i. An example of $GreedyVPVB-1$ is shown in Fig. 4.7, where the most recent path switching occurs in time slot $i-3$. When a path switching is performed at the beginning of time slot i, the transferred data size on path $P^f_{[i-3,i-1]}$ within the time slot range $[i-3,i-1]$ and the transferred data size on path P^w_i within time interval $[t[i]+\tau,t[i+1]]$ are $g(P^f_{[i-3,i-1]},[i-3,i-1])$ and $\frac{t[i+1]-t[i]-\tau}{t[i+1]-t[i]}\cdot g(P^w_i,i)$, respectively. When no path switching is performed at the beginning of time slot i, we compute the FPVB path $P^f_{[i-3,i]}$ from time slots $i-3$ to i, in which the transferred data size within the time slot range $[i-3,i]$ is $g(P^f_{[i-3,i]},[i-3,i])$. If the path switching yields a larger data transfer, $s(i)$ is updated to i, which means that the most recent path switching occurs at the time slot i. Meanwhile, the data size δ is updated by subtracting the size of data transferred between time slots $s(i-1)$ and $i-1$. t_{end} is computed when $\frac{t[i+1]-t[i]-\tau}{t[i+1]-t[i]}\cdot g(P^w_i,i)$ is greater than or equal to the remaining data size. During the time interval $[t[i],t[i]+\tau]$, the network is performing path switching and there is no data transfer during this time, so $t[i]$ is updated to $t[i]+\tau$. On the other hand, if the FPVB path in time slot $[s(i-1),i]$ provides a better solution, we set $s(i)$ to be $s(i-1)$, and compute t_{end} when $g(P^f_{[s(i),i]},[s(i),i])$ is greater than or equal to the remaining data size. In Algorithm 10, we assume that τ is smaller than the length of any time slot in the ATB list. In the case where τ is larger than the length of the current time slot i, no path switching is performed at time slot i as $g(P^f_{[s(i),i]},[s(i),i])$ yields a larger amount of data transfer. The total computational complexity of the $GreedyVPVB-1$ algorithm is $O(T\cdot m\cdot(T+\lg n))$.

Algorithm 10 $GreedyVPVB-1(G, ATB, \tau, v_s, v_d, \delta)$

Compute the widest path in time slot 0 as P_0^w;
$s(0) = 0$, $P^f_{[s(0),0]} = P_0^w$;
for $i = 1$ to $T-1$ **do**
  Compute the widest path in the time slot i as P_i^w;
  Compute the FPVB path during time slot range $[s[i-1], i]$ as $P^f_{[s[i-1],i]}$ by using $MinFPVB$ algorithm;
  if $g(P^f_{[s(i-1),i-1]}, [s(i-1), i-1]) + \frac{t[i+1]-t[i]-\tau}{t[i+1]-t[i]} \cdot g(P_i^w, i) > g(P^f_{[s(i-1),i]}, [s(i-1), i])$ **then**
    $s(i) = i$;
    $\delta = \delta - g(P^f_{[s(i-1),i-1]}, [s(i-1), i-1])$;
    if $\frac{t[i+1]-t[i]-\tau}{t[i+1]-t[i]} \cdot g(P_i^w, i) \geq \delta$ **then**
      $t_{end} = t[i] + \tau + \frac{\delta}{g(P_i^w, i)} \cdot (t[i+1] - t[i])$;
      break;
    end if
    $t[i] = t[i] + \tau$;
  else
    $s(i) = s(i-1)$;
    if $g(P^f_{[s(i),i]}, [s(i), i]) \geq \delta$ **then**
      $t_{end} = t[i] + \frac{\delta - g(P^f_{[s(i),i]}, [s(i), i-1])}{g(P^f_{[s(i),i]}, i)} \cdot (t[i+1] - t[i])$;
      break;
    end if
  end if
end for
return t_{end}.

4.3 Performance Evaluation

This section presents simulation-based performance comparisons between the heuristics designed for both FPVB and VPVB-1, which are NP-complete. The proposed heuristics are compared with optimal algorithms and greedy ones in small- and large-scale networks.

4.3.1 Simulation Setup

In the simulation, we first generate a set of networks of random topology with a different number of nodes and links under the following assumptions on the link TB lists: (a) the

initial bandwidth of each link is 10 Gbps; (b) the number of user requests within a certain time period follows a Poisson distribution; (c) the specified bandwidth and duration follow a normal distribution: (d) the available bandwidth on a link varies over time after the initial user reservation. For each simulated network, in each time period of one minute, we generate a number of user requests according to the Poisson distribution, each of which requests a path from a source node v_s to a destination node v_d with specified bandwidth b in a specified time slot $[t_s, t_e]$. For each request, the source node v_s and the destination node v_d are randomly selected, and the start time t_s is also randomly selected within the current one-minute time period. Both the specified bandwidth b and duration $d = t_e - t_s$ follow a normal distribution as follows:

$$\begin{aligned} b &= b_{max} \cdot e^{-\frac{1}{2}(3x)^2}, \\ d &= d_{max} \cdot e^{-\frac{1}{2}(3x)^2}, \end{aligned} \tag{4.3.1}$$

where b_{max} is set to 3 Gbps, d_{max} is set to one minute in large-scale networks and 10 seconds in small-scale networks, and x is a random variable within the range of [0,1]. The generated user requests are injected one by one into a simulated network and the corresponding bandwidths are reserved if there exists a feasible path for the request. In this simulation setting, the reservation load is determined by the number of user requests within a time period of one minute and the adjacent links present a certain degree of link load correlation.

4.3.2 Comparison of Algorithms for FPVB

We compare the heuristic algorithm *MinFPVB* with *GreedyFPVB* and the optimal algorithm *OptFPVB* for FPVB using various simulated networks, reservation loads and data sizes. Since the computational complexity of *OptFPVB* is exponential, the performance of *MinFPVB* is compared with that of *OptFPVB* in 200 small-scale networks with 8 nodes

and 12 links and the give data size is set to 3 GBytes. $MinFPVB$ achieves the optimal result when the available bandwidth of each link is constant. Therefore, we set maximum duration of a request to a smaller value ($d_{max} = 10$ seconds) to make the link bandwidth varying over time, and evaluate the performance of $MinFPVB$ in the harsh cases. For each simulated network with random topology, we run $MinFPVB$ and $OptFPVB$ algorithms under different reservation loads ranging from 200 to 400 at an interval of 20, and measure the data transfer end time of two resultant scheduling schemes. The percentage of networks in which $MinFPVB$ achieves the same data transfer end time as that of $OptFPVB$ under each reservation load is plotted in Fig. 4.8. Since a higher reservation load induces a larger data transfer end time, $MinFPVB$ has a lower probability to achieve the optimality under the higher reservation loads. The figure shows that $MinFPVB$ achieves the optimal performance in more than 95% cases, which indicates that $MinFPVB$ approaches the optimality with a high probability in small-scale networks.

We then compare $MinFPVB$ with $GreedyFPVB$ in 50 large-scale networks with 50 nodes and 200 links for different reservation loads and data sizes. For each simulated network with random topology, we first run these two algorithms under different reservation loads ranging from 1000 to 2000 at an interval of 100, with data size equal to 50 GBytes; then run these two algorithms with data size ranging from 30 to 80 GBytes at an interval of 5 GBytes, under the reservation load of 1500. We plot the means and standard deviations of the data transfer end time measurements among the 50 network instances for each reservation load and given data size in Fig. 4.9 and Fig. 4.10, respectively. The performance superiority of $MinFPVB$ becomes more obvious when the reservation load increases, and $MinFPVB$ is able to achieve about 2 times speedup of transfer over $GreedyFPVB$.

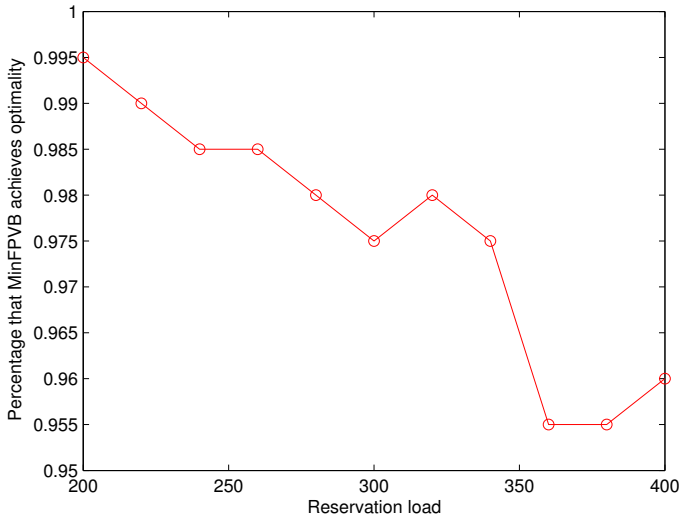

Figure 4.8: The percentage of networks in which *MinFPVB* achieves the optimality in a series of 200 simulated networks of 8 nodes and 12 links under varying reservation loads.

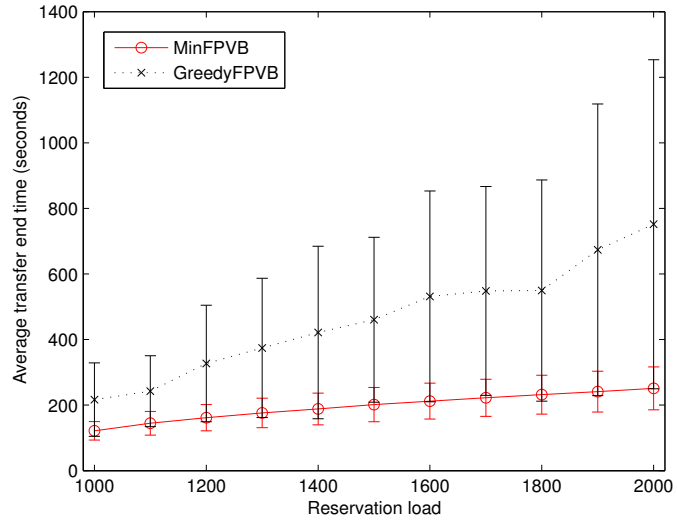

Figure 4.9: Comparison of data transfer end time (mean and standard deviation) between *MinFPVB* and *GreedyFPVB* in 50 simulated networks of 50 nodes and 200 links under varying reservation loads.

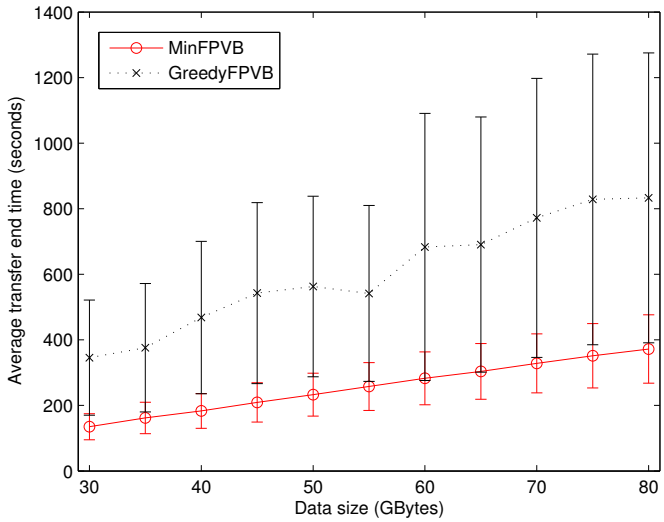

Figure 4.10: Comparison of data transfer end time (mean and standard deviation) between $MinFPVB$ and $GreedyFPVB$ in 50 simulated networks of 50 nodes and 200 links with varying data sizes.

4.3.3 Comparison of Algorithms for VPVB-1

We compare heuristic algorithms $MinVPVB-1$, $GreedyVPVB-1$ and the optimal algorithm $OptVPVB-1$ for VPVB-1 using various simulated networks, reservation loads and data sizes with the same settings as described in the previous subsection. Since the computational complexity of $OptVPVB-1$ is exponential, we first compare the performance of $MinVPVB-1$ with that of $OptVPVB-1$ in small-scale networks under varying reservation loads and the path switching delay is set to 0.5 second. The percentage of networks in which $MinVPVB-1$ achieves the same data transfer end time as that of $OptVPVB-1$ under each reservation load is plotted in Fig. 4.11. We observe that $MinVPVB-1$ almost achieves the optimal performance. VPVB-1 allows path switching and the number of time slots between tow adjacent path switchings is relatively smaller than the total transfer end

time as $MinVPVB-1$ applies $MinFPVB$ to compute the maximum amount of data transferred during the time slots among two path switchings. Since $MinFPVB$ has a higher probability to achieve the optimality in a smaller number of time slots, $MinVPVB-1$ has a higher probability to achieve the optimality in the same network settings. The measurements show that $MinVPVB-1$ has a probability of more than 98% to achieve the global optimality in a statistical sense in small-scale networks.

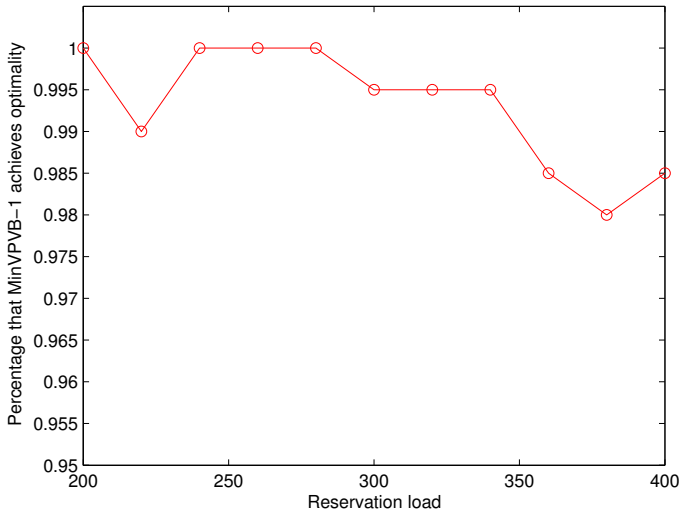

Figure 4.11: The percentage of networks in which $MinVPVB-1$ achieves the optimality in a series of 200 simulated networks of 8 nodes and 12 links under varying reservation loads.

We then compare $MinVPVB-1$ with $GreedyVPVB-1$ in 50 large-scale networks for different reservation loads and data sizes, and the path switching delay is set to 0.5 second. We plot the means and standard deviations of the data transfer end time measurements among 50 network instances for each reservation load and given data size in Fig. 4.12 and Fig. 4.13, respectively. We observe that the average data transfer end time obtained by either $MinVPVB-1$ or $GreedyVPVB-1$ almost linearly increases as the reservation

load or data size increases, and $MinVPVB-1$ always outperforms $GreedyVPVB-1$. The comparison between Fig. 4.12 and Fig. 4.9 and the comparison between Fig. 4.13 and Fig. 4.10 show that VPVB has a better performance than FPVB since the flexibility for path switching improves the utilization of bandwidth resources.

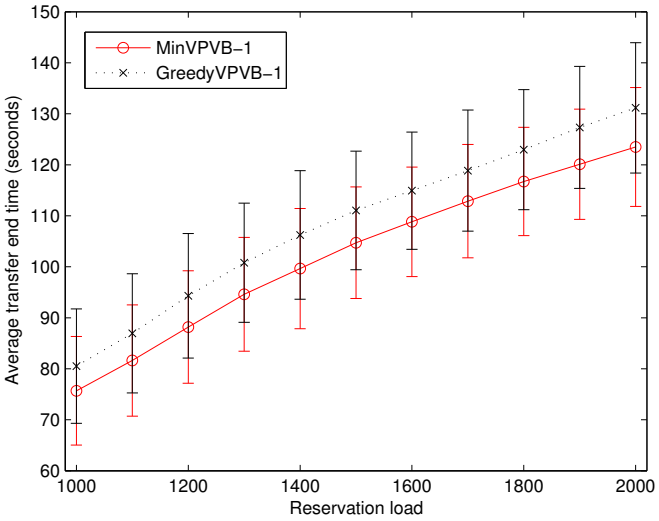

Figure 4.12: Comparison of data transfer end time (mean and standard deviation) between $MinVPVB-1$ and $GreedyVPVB-1$ in 50 simulated networks of 50 nodes and 200 links under varying reservation loads.

We also compare $MinVPVB-1$ with $GreedyVPVB-1$ in 50 large-scale networks for different path switching delays ranging from 0.1 to 1 second at an interval of 0.1 second, and study the effect of the path switching delay on the data transfer end time. The reservation load is set to 1500 and the data size is set to 50 GBytes. We plot the mean and standard deviation of the data transfer end time measurements among the 50 network instances for each path switching delay in Fig. 4.14. We observe that $MinVPVB-1$ consistently outperforms $GreedyVPVB-1$ with different path switching delays and the average data transfer end time linearly increases as the path switching delay increases. When the path switching

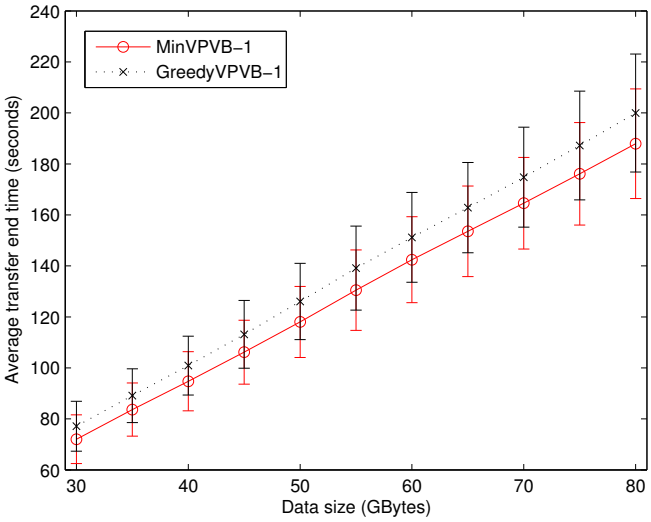

Figure 4.13: Comparison of data transfer end time (mean and standard deviation) between $MinVPVB-1$ and $GreedyVPVB-1$ in 50 simulated networks of 50 nodes and 200 links with varying data sizes.

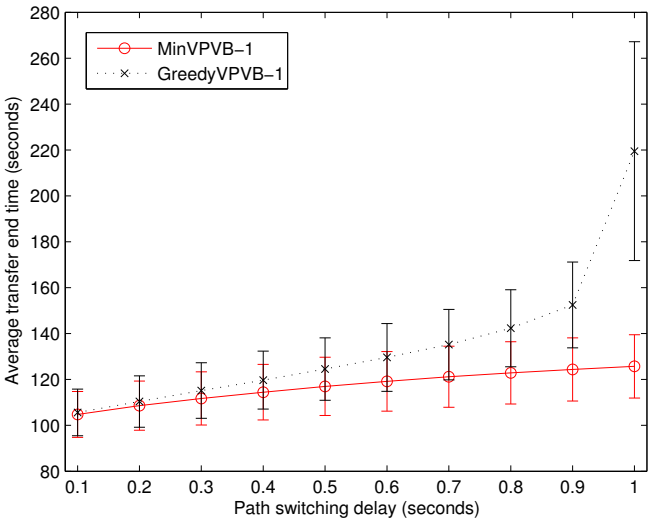

Figure 4.14: Comparison of data transfer end time (mean and standard deviation) between $MinVPVB-1$ and $GreedyVPVB-1$ in 50 simulated networks of 50 nodes and 200 links with varying path switching delays.

delay is set to 1 second, i.e. the length of each time slot, no path switching is performed in $GreedyVPVB-1$, which leads to an unsatisfactory performance. $MinVPVB-1$ employs dynamic algorithm to compute the appropriate time points for path switching although the path switching lasts for an entire time slot, and achieves a better transfer performance in comparison with $GreedyVPVB-1$.

Chapter 5

Bandwidth Scheduling in LCC-overlays

5.1 LCC Model and Problem Formulation

In an overlay network built on top of another network, overlay links are virtual links that correspond to underlying paths in the lower-layer network. Overlay links are correlated if the underlying paths they are mapped to share one or more lower-layer link segments. In this section, we first introduce the LCC model proposed in [49], where the two-layer hierarchy of an overlay network is modeled as:

- A lower-layer (e.g. IP) graph $\hat{G} = (\hat{V}, \hat{E})$; each low-layer link $\hat{l} \in \hat{E}$ has a bandwidth of $c_{\hat{l}}$.
- A higher-layer (i.e. overlay) graph $G = (V, E)$, where $V \subset \hat{V}$;
- A mapping of each overlay link $(v_1, v_2) \in E$ to a low-layer path $P(v_1, v_2) \subset \hat{G}$ from v_1 to v_2.

Let n and m represent the number of nodes and the number of links in overlay graph G, respectively. The LCC-overlay network is defined as follows.

Definition 1. *The LCC-overlay network is a triplet* (G, M, b) *where*

- *the bandwidth of each link* $l \in E$ *is a non-negative variable* x_l.

- (M,b) *represent a set of z linear capacity constraints* $Mx \leq b$*:*
  - *M is a 0-1 coefficient matrix of size* $z \times m$*;*
  - *x is an* $m \times 1$ *vector of link bandwidth variables;*
  - $b \in R^z$ *is the bandwidth vector.*

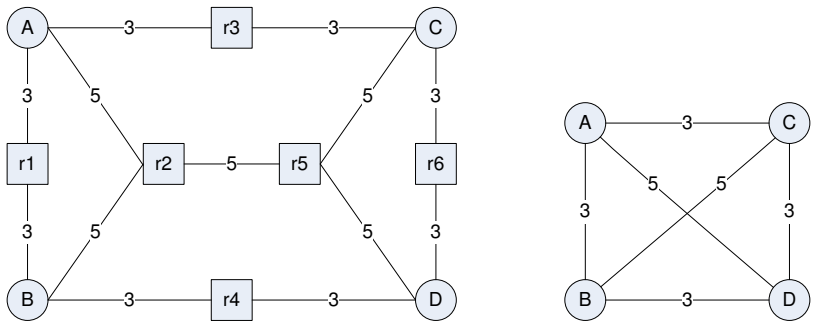

Figure 5.1: An example of the LCC-overlay network of a two-layer hierarchy.

The maximum value of variable x_l for overlay link l is equal to the bandwidth of the corresponding path in the lower-layer network. Each row i in (M,b) is a constraint in the form of $\sum_{l:C[i][l]=1} x_l \leq b[i]$. The number of elements with the value of '1' in each row of M is at least one. For illustration purposes, a numerical example is provided to explain the above two-layer overlay network model, as shown in Fig. 5.1, which consists of four overlay nodes and 6 routers, and the number on a link represents the link bandwidth. Fig. 5.1 (a) is the physical (lower-layer) network graph, and Fig. 5.1 (b) is the overlay network built on top of it. In this example, two overlay links (A,D) and (B,C) are correlated, and hence the sum of their bandwidths is constrained by the capacity of the shared physical link (r_2,r_5), i.e. $x_{(A,D)} + x_{(B,C)} \leq 5$. The rest links in Fig. 5.1 (b) are independent of each other. The LCC for the overlay graph is given below in the form of $Mx \leq b$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{(A,B)} \\ x_{(A,C)} \\ x_{(A,D)} \\ x_{(B,C)} \\ x_{(B,D)} \\ x_{(C,D)} \end{pmatrix} \leq \begin{pmatrix} 3 \\ 3 \\ 5 \\ 3 \\ 3 \end{pmatrix}. \tag{5.1.1}$$

From the constraint $x_{(A,D)} + x_{(B,C)} \leq 5$, we have $x_{(A,D)} \leq 5$ and $x_{(B,C)} \leq 5$. If the bandwidth of the physical link (r_2, r_5) is 6 instead of 5 in Fig. 5.1 (a), besides the constraint of $x_{(A,D)} + x_{(B,C)} \leq 6$, we need to add another two constraints, $x_{(A,D)} \leq 5$ and $x_{(B,C)} \leq 5$, into Eq. 5.1.1.

In an LCC-overlay network that supports advance bandwidth reservation, the available bandwidth on each overlay link is not a single value. Instead, each link $l \in E$ maintains a list of residual bandwidths specified as segmented constant functions of time. Since the residual link bandwidth varies over different time slots, bandwidth vector b is different in the LCC for each time slot. When a user makes a bandwidth reservation on link $l \in E$, considering the LCC in G, not only the time-bandwidth list for link l is updated, but also the time-bandwidth lists for other links that are correlated to link l are updated. Based on the bandwidth reservation on the overlay links and the topology of the lower-layer network, a bandwidth vector b is computed to represent the linear link capacity constraints in each time slot. The coefficient matrix M is fixed across different time slots as the link correlations do not change. Then a bandwidth matrix B of size $z \times T$ is constructed, where the i-th column of B is the bandwidth vector of LCC in the i-th time slot. An LCC-overlay network that supports advance bandwidth reservation can be represented by a triplet of (G, M, B).

Based on different data transport constraints and application requirements, we formulate two advance bandwidth scheduling problems with the same objective to minimize the data transfer end time as follows: Given an LCC-overlay network (G,M,B) that supports advance bandwidth reservation, source v_s and destination v_d, data size δ,

- Fixed-Bandwidth Path (FBP): compute a path from v_s to v_d with a fixed bandwidth;
- Varying-Bandwidth Path (VBP): compute a path from v_s to v_d with varying bandwidths across multiple time slots.

5.2 Complexity Analysis

FBP and VBP problems are corresponding to FPFB and FPVB problems presented in the previous chapter, respectively, where FBP is optimally solved in polynomial time and VBP is proved to be NP-complete. Obviously, VBP is still NP-complete in LCC-overlay networks, since a network without LCC is a special case of networks with LCC. FBP computes a path to minimize the data transfer end time for a given data size, which is equivalent to the widest-path problem. Since the widest-path problem with liner capacity constraints (WPC) is proved to be NP-complete in [49], so is FBP.

We first prove the lower bound of the approximation ratio for WPC, and then prove all of these bandwidth scheduling problems are non-approximable in LCC-overlay networks. The decision version of WPC is defined as follows: Given an LCC-overlay network graph (G,M,b), source v_s and destination v_d, a positive value $\beta \leq max\{b[i]\}$, does there exist a path from v_s to v_d such that the path bandwidth is no less than β? Since the link bandwidth in WPC is a single value that does not change over time, b is a vector.

5.2.1 WPC is Approximable

This subsection proves that there does not exist an approximate algorithm with an approximation ratio less than or equal to $(\gamma-1)/\lambda$ for WPC, where γ denotes the maximum number of elements with the value of 1 in a row of coefficient matrix M, which is equal to the maximum number of overlay links that are correlated, and λ denotes the number of links from an LCC that are bottleneck links in the optimal path. An approximate algorithm with an approximation ratio of γ/λ for this problem is proposed in Section 7.2. To prove the lower bound of the approximation ratio for WPC, we first define the Path with Forbidden Tuples (PFT) problem: Given a directed graph $G=(V,E)$, source v_s and destination v_d, a collection $F=\{(v_1^1,...,v_k^1),...,(v_1^p,...,v_k^p)\}$ of k-tuples ($k\geq 2$) of nodes from V, does there exist a direct path from v_s to v_d that contains at most $k-1$ nodes from each k-tuple in F?

Theorem 5. *PFT is NP-complete.*

Proof. Given a solution (a path from v_s to v_d) to PFT, one can verify in polynomial time the validity of the solution by checking whether or not the path contains at most $k-1$ nodes from each k-tuple in F. Hence, PFT $\in$ NP.

Now we reduce the Path with Forbidden Pairs (PFP) problem in [23] to PFT. PFP is defined as follows: Given a directed graph $G=(V,E)$, source v_s and destination v_d, a collection $F=\{(v_1^1,v_2^1),...,(v_1^p,v_2^p)\}$ of pairs of nodes from V (all the pairs in F are disjoint), does there exist a path from v_s to v_d that contains at most one node from each pair in F?

Let (G,v_s,v_d,F) be an arbitrary instance of PFP. An instance (G',v_s',v_d',F',k') of PFT is constructed from the PFP instance in polynomial time such that G' has a direct path from

v'_s to v'_d that contains at most $k'-1$ nodes from each k'-tuple in F', if and only if G has a direct path from v_s to v_d that contains at most one node from each pair in F. k' is set to be an integer that is greater than 2. Any node that is in V but not in any pair of F is added to V'. Any link $l \in E$ that is not incident to any node in F is added to E'. For each pair (v_1, v_2) in F, v_2 is replaced with $k'-1$ nodes $v'_1, ..., v'_{k'-1}$ and $k'-2$ directed links (v'_1, v'_2), (v'_2, v'_3), ..., $(v'_{k'-2}, v'_{k'-1})$, which are referred to as v_2's replacement links. For each directed link $l = (u, v_2) \in E$ that ends at v_2, a directed link $l' = (u, v'_1)$ is added to E'; similarly, for each directed link $l = (v_2, u) \in E$ that starts from v_2, a directed link $l' = (v'_{k'-1}, u)$ is added to E'. Let $v'_s = v_s$ when there does not exist a pair (u, v_s), $u \in V$ in F; otherwise, v_s is replaced with $k'-1$ nodes in V'. v'_s is set to be the first node of these $k'-1$ nodes, and v'_d is set in a similar way. F' is a simple copy of F by replacing each pair with a corresponding k'-tuple.

Suppose that G has a direct path P from v_s to v_d that contains at most one node from each pair in F. A corresponding path P' can be found in G' by simply substituting each P's constituent node that appears in the latter position of a pair in F by its $k'-2$ replacement links in G'. Clearly, P' contains at most $k'-1$ nodes from each k'-tuple in F'. Hence, P' composes a solution to PFT.

Conversely, let P' be a direct path in G' from v'_s to v'_d that contains at most $k'-1$ nodes from each k'-tuple in F'. P' is collapsed to a path P in G by shrinking the replacement links into their corresponding nodes. Obviously, P is the solution to PFP. This concludes the proof.

Theorem 6. *For a given WPC problem, let γ be the maximum number of overlay links that are correlated and λ be the number of links from an LCC that are bottleneck links in the optimal path. WPC cannot be approximated within an approximation ratio of $(\gamma-1)/\lambda$, unless P = NP.*

Proof. Assume that there exists an approximate algorithm with an approximation ratio of $(\gamma-1)/\lambda$ for WPC. This assumption is shown to imply a polynomial-time optimal algorithm for the PFT problem.

Let (G, v_s, v_d, F, k) be an arbitrary instance of PFT. An instance (G', M', b', v'_s, v'_d) of WPC optimization problem is constructed from the PFT instance in polynomial time. The WPC optimization problem computes a path from v'_s to v'_d to maximize the path bandwidth. Any node that is in V but not in any k-tuple of F is added to V', and any link $l \in E$ that is not incident to any node in F is added to E'. Each node v in F is replaced with two nodes v'_1, v'_2 and a directed link from v'_1 to v'_2, which is referred to as v's replacement link. For each directed link $l = (u, v) \in E$ that ends at v, a directed link $l' = (u, v'_1)$ is added to E'; similarly, for each directed link $l = (v, u) \in E$ that starts from v, a directed link $l' = (v'_2, u)$ is added to E'. Let $v'_s = v_s$ when v_s does not appear in F; otherwise, v_s is replaced with two nodes in V'. Let v'_s to be the first node of these two nodes, and set v'_d in a similar way. Then the linear capacity constraints are constructed. Each link $l \in E'$ except those with replaced nodes in F translates to a one-variable constraint $x_l \leq 1$. For each k-tuple of nodes $(v_1, ..., v_k)$ in F with k replacement links $l_1, ..., l_k$ in G', respectively, a k-variable constraint $x_{l_1} + ... + x_{l_k} \leq 1$ is constructed.

In the WPC instance (G', M', b', v'_s, v'_d), since the maximum number of links that are correlated is k, $\gamma = k$. The approximate algorithm with an approximation ratio of $(\gamma-1)/\lambda = (k-1)/\lambda$ is applied to the WPC instance. Since the approximation ratio $(k-1)/\lambda$ must be greater than or equal to 1, the approximation ratio of the approximate algorithm is 1 instead of $(k-1)/k$ when $\lambda = k$. Obviously, the approximation algorithm can find a direct

path from v'_s to v'_d with path bandwidth at least $1/(k-1)$ if and only if there exists a direct path from v_s to v_d that contains at most $k-1$ nodes from each k-tuple in F, because the bandwidth of a replacement link is at least $1/(k-1)$ if and only if at most $k-1$ replacement links from a k-variable constraint are used. Therefore, a $(\gamma-1)/\lambda$ approximate algorithm to WPC finds a solution to PFT whenever one exists. This conflicts with the NP-completeness of PFT. Proof ends.

5.2.2 FBP and VBP are Non-approximable

FBP and VBP are not only NP-complete, but also non-approximable.

Theorem 7. *For any polynomial-time computable function $f(n,m)$, FBP and VBP cannot be approximated within an approximation ratio of $f(n,m)$, unless P = NP.*

Proof. Firstly, we prove FBP to be non-approximable. Assume that there exists an approximate algorithm with an approximation ratio of $f(n,m)$ for FBP. We will show that this assumption implies a polynomial-time optimal algorithm for WPC.

Let (G,M,b,v_s,v_d,β) be an arbitrary instance of the decision version of WPC. An instance $(G',M',B',v'_s,v'_d,\delta')$ of FBP optimization problem is constructed from the WPC instance in polynomial time. Let $G'=G$, $M'=M$, $v'_s=v_s$, $v'_d=v_d$, and $\delta'=\beta$. The aggregated time-bandwidth list ATB' is constructed for all links $l'\in G'$, which consists of 3 time slots as shown in Fig. 5.2. The available bandwidth of all links $l'\in G'$ in the first time slot (i.e. time interval $[0,1]$) and the third time slot (i.e. time interval $[f(n,m),+\infty]$) is the same as the available bandwidth of all links $l\in G$, while the available bandwidth of all links $l'\in G$ is 0 in the second time slot (i.e. time interval $[1,f(n,m)]$). Correspondingly, B' is a matrix of three columns: the first and third column of B' are the copies of bandwidth vector b, and all elements in the second column of B' have the value of 0.

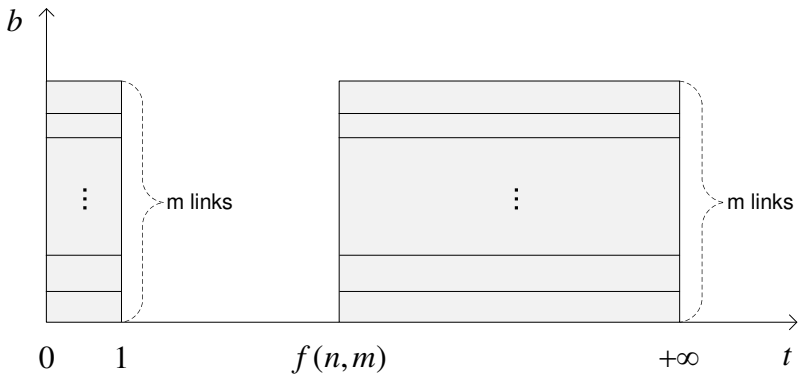

Figure 5.2: Construction of ATB' in an FBP instance.

The approximate algorithm is applied to the FBP instance $(G', M', B', v'_s, v'_d, \delta')$. Apparently, the approximate algorithm can find a path from v'_s to v'_d with fixed bandwidth such that the data of size δ' can be completely transferred along the path during time interval $[0, f(n,m)]$ if and only if there exists a path from v_s to v_d in G such that the path bandwidth is no less than β, since the size of data transferred during time interval $[1, f(n,m)]$ is 0 for the FBP instance. Therefore, an $f(n,m)$ approximate algorithm to FBP finds a solution to WPC whenever one exists. This conflicts with the NP-completeness of WPC.

Similarly, VBP can be proved to be non-approximable by constructing its problem instance from the 0-1 TB problem instance [27]. Proof ends.

FBP and VBP are non-approximable with respect of the function of network size $f(n,m)$. If the time slots in the aggregated time-bandwidth list is taken into consideration, FBP and VBP are approximable.

5.3 Algorithm Design

The non-approximability of these scheduling problems in LCC-overlay networks indicates that there does not exist any polynomial-time optimal algorithms or approximate algorithms for these problems unless P = NP. Therefore, we propose a heuristic algorithm for each of these problems.

5.3.1 Heuristic Algorithm for FBP

Algorithm 11 $MinFBP(G,M,B,v_s,v_d,\delta)$

1: $t_{end}^{min} = \infty$;
2: **for** $q = 0$ to $T-1$ **do**
3: **for** $p = 0$ to q **do**
4: **for all** $i = 1$ to z **do**
5: $b[i] = \min_{p \le j \le q} (B[i,j])$;
6: **end for**
7: $\beta = MaxBW(G,M,b,v_s,v_d)$;
8: **if** $\beta \cdot (t[q+1] - t[p]) \ge \delta$, and $t_{end}^{min} > t[p] + \delta/\beta$ **then**
9: $t_{end}^{min} = t[p] + \delta/\beta$;
10: **end if**
11: **end for**
12: **if** $t_{end}^{min} < \infty$ **then**
13: **break**;
14: **end if**
15: **end for**
16: **return** t_{end}^{min}.

FBP takes as input an LCC-overlay network (G,M,B), source v_s and destination v_d, and data size δ, and computes a path with a fixed bandwidth to minimize the data transfer end time. Note that the data transfer start time and path bandwidth are not specified by the user. A heuristic algorithm referred to as *MinFBP* is proposed for FBP, whose pseudocode is provided in Algorithm 11. The output of the algorithm is the minimal data transfer end time t_{end}^{min}. Since it may not be always optimal to start data transfer immediately, the

algorithm varies the transfer start time slot p from 0 to q for a given data transfer end time slot q, and checks whether there exists any feasible time slot p such that the data of size δ can be transferred during the time slot range $[p,q]$ (i.e. time interval $[t[p],t[q+1]]$). If there does not exist any feasible path, the algorithm repeatedly increases q by 1; otherwise, the algorithm computes the best start time slot p and the corresponding minimal data transfer end time t_{end}^{min} by considering all possible p values and terminates. In line 7, the bandwidth β of the widest path is calculated by *MaxBW* algorithm, which is the solution to the WPC optimization problem.

To facilitate the explanation of *MaxBW*, the following notations are defined:

$\sum M, \sum M[j,*]$**:** compute the sum of all the elements in M and the sum of the elements in the j-th row of M, respectively.

$c[j]$**:** the maximum number of links from the j-th LCC that are used for the widest path computation.

$b_{(v_s,v)}$**:** the bandwidth of the computed path from v_s to v, which is the bottleneck bandwidth of all component links on the path.

Q**:** a queue of nodes sorted by their bandwidth $b_{(v_s,v)}$ in an decreasing order.

MaxBW takes as input an LCC-overlay network (G,M,b), source v_s and destination v_d, and computes the bandwidth of the widest path from v_s to v_d. The pseudocode of *MaxBW* is shown in Algorithm 12. In a network without LCC, the widest path can be computed by a modified Dijkstra's algorithm since the link bandwidth is determined before path computation. However, in an LCC-overlay network, the bandwidth of the links within each LCC

Algorithm 12 $MaxBW(G, M, b, v_s, v_d)$

1: $\beta_{max} = 0$;
2: **for** $i = 1$ to $\sum M - z$ **do**
3: **if** $i \equiv 1$ **then**
4: $c[j] = 1, \forall j \in [1, z]$;
5: **else**
6: $k = \underset{j \in [1,z], c[j] < \sum M[j,*]}{\operatorname{argmax}} \{\frac{b[j]}{c[j]+1}\}$;
7: $c[k] = c[k] + 1$;
8: **end if**
9: **for all** $l \in E$ **do**
10: $b_l = \min_{j \in [1,z], M[j,l] \equiv 1} (b[j]/c[j])$;
11: **end for**
12: **for all** $v \in V, v \neq v_s$ **do**
13: $b_{(v_s,v)} = 0$;
14: **end for**
15: $b_{(v_s,v_s)} = \infty$;
16: $Q = V$;
17: **while** $Q \neq \emptyset$ **do**
18: $u = dequeue(Q)$;
19: **if** $u \equiv v_d$ **then**
20: **break**;
21: **end if**
22: **for all** $v \in Q, (u,v) \in E$ and the computed path from v_s to u uses at most $c[j] - 1$ links from the j-th LCC that contains (u,v), $\forall j \in [1, z]$ **do**
23: **if** $b_{(v_s,v)} < min(b_{(v_s,u)}, b_{(u,v)})$ **then**
24: $b_{(v_s,v)} = min(b_{(v_s,u)}, b_{(u,v)})$;
25: **end if**
26: **end for**
27: **end while**
28: **if** $\beta_{max} < b_{(v_s,v_d)}$ **then**
29: $\beta_{max} = b_{(v_s,v_d)}$;
30: **end if**
31: **end for**
32: **return** β_{max}.

is not determined until the actual number of links from each LCC that are used to constitute the widest path is known. Lines 3-8 in Algorithm 12 restrict the maximum number of links from each LCC that are used for the widest path computation, and incrementally relax the restriction in each **for** loop (line 2), which is also shown in Table 5.1. This table contains z rows, where z is the total number of LCCs in the given network, and $\sum M - z$ columns (excluding the first column). In the base case (the second column of Table 5.1), $c[i], i \in [1,z]$ is initialized to be 1, which means that at most one link from each LCC is used for the widest path computation. In each successive column, one of the LCCs is chosen to relax the restriction on the number of links based on line 6 in Algorithm 12, while the restrictions on the number of links for other LCCs remain the same. The k-th LCC is chosen such that $b[k]/(c[k]+1)$ is maximized, e.g., $b[k]/(c[k]+1)$ is no less than $b[j]/(c[j]+1)$, $j \in [1,z]$, to increase $c[k]$ by 1 and balance the link bandwidth in the network. Then the link bandwidth is computed based on $c[j]$ and the given bandwidth vector b in lines 9-11. A link may appear in multiple LCCs, and the link bandwidth is determined by the minimal $b[j]/c[j]$. Note that fair sharing of the bandwidth among the links within one LCC is used.

Table 5.1: Increment in the number of links from each LCC.

$c[1]$	1	1	1	...	$\sum M[1,*]$	$\sum M[1,*]$
$c[2]$	1	2	2	...	$\sum M[2,*]$	$\sum M[2,*]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$c[j]$	1	1	2	...	$\sum M[j,*]$	$\sum M[j,*]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$c[z]$	1	1	1	...	$\sum M[z,*]-1$	$\sum M[z,*]$

Once the bandwidths of all links are determined, the widest path is computed by using a modified Dijkstra's algorithm in lines 12-30. During the bandwidth relaxation of the

current node u to one of its neighbor nodes v, line 22 guarantees that the computed path from v_s to v satisfies $c[j]$ restriction. Each node maintains a vector of z size to store the number of links from each LCC that are already used in the computed path from v_s to itself. If the computed path from v_s to u already consists of $c[j]$ links from the j-th LCC and the link (u,v) is also in the j-th LCC, then u does not relax its bandwidth to v. There are total $\sum M - z$ loops in the algorithm, and the algorithm updates the maximum available bandwidth β_{max} at the end of each loop.

Theorem 8. *For a given WPC problem, let γ be the maximum number of overlay links that are correlated and λ be the number of links from an LCC that are bottleneck links in the optimal path. The approximation ratio of MaxBW algorithm to WPC is γ/λ.*

Proof. The case of the last column of Table 5.1 is considered, where $c[j]$ is set to be $\sum M[j,*]$ and all the links from each LCC can be used for widest path computation. $\gamma \geq \sum M[j,*]$, and the bandwidths of all links in the j-th LCC are at least $b[j]/\gamma, \forall j \in [1,z]$. If the bottleneck links in the optimal path are from the k-th LCC, the bandwidth of the optimal path is $b[k]/\lambda$. The LCCs that contain the links constituting the optimal path are denoted as optimal LCCs. *MaxBW* can find a path with bandwidth of at least $b[k]/\gamma$ by using only the links from the optimal LCCs. Therefore, the approximation ratio of *MaxBW* is $\frac{b[k]/\lambda}{b[k]/\gamma} = \gamma/\lambda$. Proof ends.

The time complexity of *MaxBW* is $O((\sum M - z) \cdot (n^2 + z \cdot m))$, or $O(z \cdot m \cdot (n^2 + z \cdot m))$ in the worst case, and the time complexity of *MinFBP* is $O(T^2 \cdot z \cdot m \cdot (n^2 + z \cdot m) + T^3 \cdot z)$.

5.3.2 Heuristic Algorithm for VBP

With the same input of FBP, VBP computes a path with variable bandwidth during file transfer to better utilize network resources. We propose a heuristic algorithm for VBP,

referred to as *MinVBP*. Again, several notations or functions are defined to facilitate our explanation:

$\Psi(\frac{B[j,]}{c[j]+1}, \delta)$**:** a function that computes the transfer end time of data of size δ using $\frac{B[j,]}{c[j]+1}$ time-bandwidth list, where $B[j,]$ denotes the j-th row of bandwidth matrix B.

$t_{end}[v]$**:** the transfer end time for data of size δ from v_s to v along the computed path.

Q**:** a queue of nodes sorted by their data transfer end time $t_{end}[v]$ in an increasing order.

$b_{(v_s,v)}[j]$**:** the bandwidth of the computed path from v_s to v in the j-th time slot.

The pseudocode of *MinVBP* algorithm is shown in Algorithm 13. Similar to *MaxBW*, lines 3-8 in Algorithm 13 restrict the maximum number of links from each LCC that are used for widest path computation. In each **for** loop, line 6 chooses one LCC to relax such that the data transfer end time along the shared physical link is minimized. The link bandwidth is computed at different time slots based on $c[j]$ and the given bandwidth matrix B in lines 9-11. Then the data transfer end time $t_{end}[v_d]$ is computed by using a modified Dijkstra's algorithm in lines 12-31. Similarly, line 23 guarantees that the computed path from v_s to v satisfies $c[j]$ restriction. Note that the available bandwidth of the path is determined by the bottleneck bandwidth of all component links in each time slot. *MinVBP* addresses this issue by keeping track of the bottleneck bandwidth of the computed path. At each relaxation step, the bottleneck bandwidth $b'_{(v_s,v)}[j]$ from v_s to node v is computed. The algorithm computes the corresponding data transfer end time of data of size δ from v_s to v using bandwidth $b'_{(v_s,v)}[j]$, and updates the minimal data transfer end time t_{end}^{min} at the end of each loop. The total time complexity of *MinVBP* is $O(z \cdot m \cdot (n^2 + m \cdot (z+T)))$.

Algorithm 13 $MinVBP(G, M, B, v_s, v_d, \delta)$

1: $t_{end}^{min} = \infty$;
2: **for** $i = 1$ to $\sum M - z$ **do**
3: **if** $i \equiv 1$ **then**
4: $c[j] = 1, \forall j \in [1, z]$;
5: **else**
6: $k = \underset{j \in [1,z], c[j] < \sum M[j,*]}{\operatorname{argmin}} \{\Psi(\frac{B[j,]}{c[j]+1}, \delta)\}$;
7: $c[k] = c[k] + 1$;
8: **end if**
9: **for all** $l \in E$ **do**
10: $b_l[j] = \min_{k \in [1,z], M[k,l] \equiv 1} (\frac{B[k,j]}{c[k]}), \forall j \in [0, T-1]$;
11: **end for**
12: **for all** $v \in V$ **do**
13: $t_{end}[v] = \infty$;
14: **end for**
15: $t_{end}[v_s] = 0$;
16: $b_{(v_s,v_s)}[j] = \infty, \forall j \in [0, T-1]$;
17: $Q = V$;
18: **while** $Q \neq \emptyset$ **do**
19: $u = dequeue(Q)$;
20: **if** $u \equiv v_d$ **then**
21: **break**;
22: **end if**
23: **for all** $v \in Q, (u, v) \in E$ and the computed path from v_s to u uses at most $c[j] - 1$ links from the j-th LCC that contains (u, v), $\forall j \in [1, z]$ **do**
24: $b'_{(v_s,v)}[k] = min(b_{(v_s,u)}[k], b_{(u,v)}[k]), \forall k \in [0, T-1]$;
25: Compute the data transfer end time t'_{end} of data of size δ from v_s to v using bandwidth $b'_{(v_s,v)}$;
26: **if** $t_{end}[v] > t'_{end}$ **then**
27: $t_{end}[v] = t'_{end}$;
28: $b_{(v_s,v)}[k] = b'_{(v_s,v)}[k], \forall k \in [0, T-1]$;
29: **end if**
30: **end for**
31: **end while**
32: **if** $t_{end}^{min} > t_{end}[v_d]$ **then**
33: $t_{end}^{min} = t_{end}[v_d]$;
34: **end if**
35: **end for**
36: **return** t_{end}^{min}.

5.4 Performance Evaluation

We conduct simulation-based performance comparisons between the heuristics for FBP and VBP. Realistic overlay networks of a two-layer hierarchy with various sizes and topologies are generated in the simulation. Each simulated lower-layer network has a randomly generated network topology for a given number of nodes and links, and the TB list of each link is also randomly generated with residual bandwidth ranging from 0.2 Gbps to 10 Gbps at each time slot with the same length of 1 second. The residual bandwidths follow a normal distribution according to the following function:

$$b_l[i] = 0.2 + 10 \cdot (1 - e^{-\frac{1}{2}(3x)^2}), \tag{5.4.1}$$

where x is randomly selected within the range of [0,1]. A number of nodes from the lower-layer network are selected to serve as overlay nodes. An overlay link that connects two overlay nodes are computed by Breadth First Search, which corresponds to a path between these two nodes in the lower-layer network. Then the LCC model is constructed based on the correlations among these overlay links.

5.4.1 Comparison of Algorithms for FBP

We evaluate the performance of the approximate algorithm *MaxBW* in LCC-overlay networks. *MaxBW* is compared with the optimal algorithm and modified Dijkstra's algorithm using various simulated networks. Similar to *MaxBW*, the optimal algorithm gradually increases the number of links from each LCC and tries all possible combinations of $C[i]$ links from the i-th LCC and the combinations of links from different LCCs. The modified Dijkstra's algorithm first computes the widest path without considering the link LCCs, and

then computes the path bandwidth by considering the LCCs of the links on the computed path.

Since the computational complexity of the optimal algorithm is exponential, the performance of *MaxBW* is compared with that of the optimal algorithm in 200 small-scale overlay networks with 8 overlay nodes, which are built on top of lower-layer networks with 15 nodes and 30 links. *MaxBW* and the optimal algorithm are run in each of these 200 networks, and the path bandwidth of two resultant scheduling schemes is measured. Fig. 5.3 shows a histogram-like performance comparison, where the x-axis represents the ratio of path bandwidth obtained by *MaxBW* over the optimal bandwidth, and the y-axis represents the percentage of sample networks fall in each ratio range. The figure shows that *MaxBW* achieves the optimal performance in 98% of all the test cases. Since the problem search space is relatively small when there are only 8 overlay nodes, *MaxBW* approaches the optimality with a high probability in such small-scale networks.

MaxBW is further compared with a modified Dijkstra's algorithm in a series of 200 large-scale networks. The number of nodes and links in the lower-layer networks is fixed to be 100 and 300, respectively, and the number of overlay nodes is varied from 10 to 90 at an interval of 10. For each given number of overlay nodes, 200 overlay network instances are randomly generated, and these two algorithms are run in each of these network instances. The percentage of network instances in which *MaxBW* outperforms the modified Dijkstra's algorithm versus the ratio of the overlay network size over the lower-layer network size is plotted in Fig. 5.4, while in the rest network instances, these two algorithms achieve the same performance. The figure shows that the modified Dijkstra's algorithm is able to achieve the actual bandwidth that is close to the one computed by *MaxBW*, although

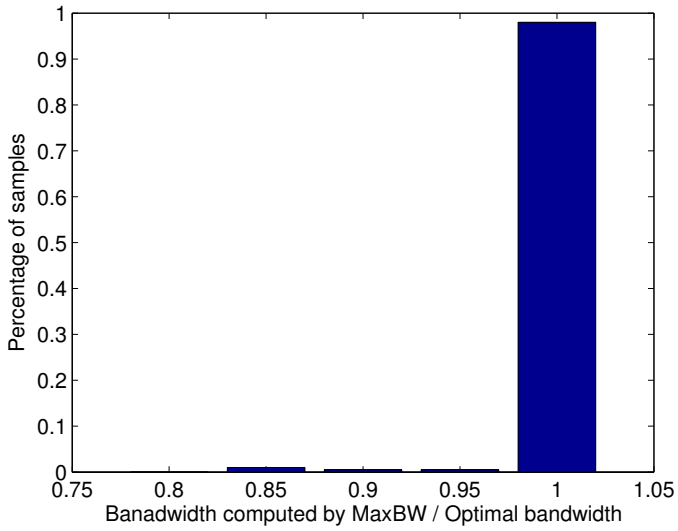

Figure 5.3: Performance comparison of *MaxBW* and the optimal algorithm in 200 sample networks with 8 overlay nodes.

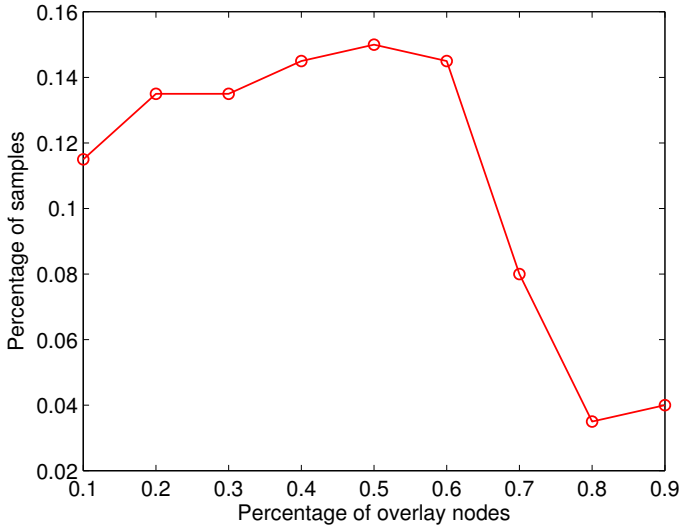

Figure 5.4: Percentage of network instances in which *MaxBW* outperforms the modified Dijkstra's algorithm versus the ratio of the overlay network size over the lower-layer network size.

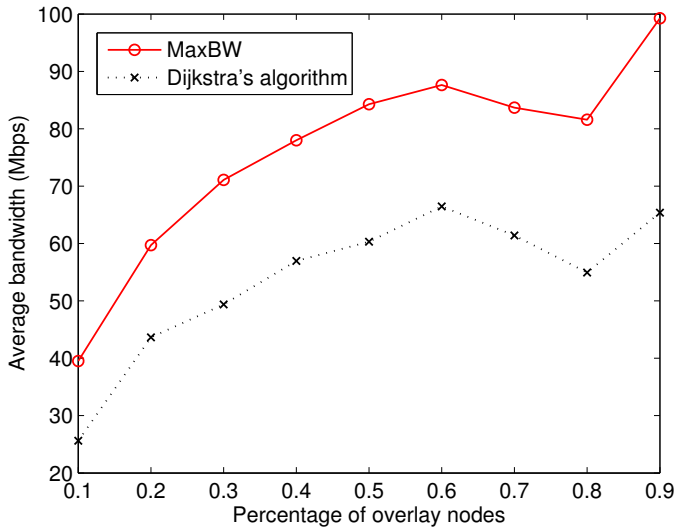

Figure 5.5: Average bandwidth versus the ratio of the overlay network size over the lower-layer network size.

the modified Dijkstra's algorithm does not consider link LCCs in the path computation. The main reason is that the links on a single path are not very likely correlated, which means that an overlay path rarely traverses a physical link segment more than once. When the number of overlay nodes increases, both of these algorithms tend to achieve the optimal performance, resulting in insignificant performance differences. For the network instances where *MaxBW* outperforms the modified Dijkstra's algorithm, the average bandwidths computed by these two algorithms are plotted in Fig. 5.5, which indicates that *MaxBW* significantly improve the performance when the links on the widest path are correlated, where the widest paths are computed without considering LCCs.

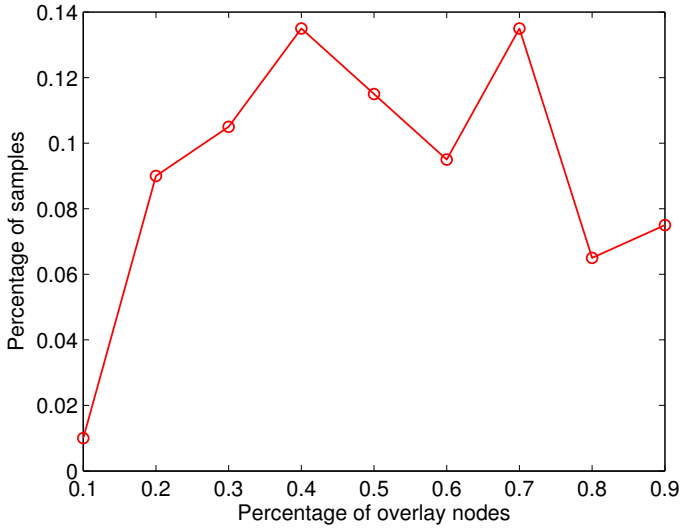

Figure 5.6: Percentage of network instances in which *MinVBP* outperforms the modified Dijkstra's algorithm versus the ratio of the overlay network size over the lower-layer network size.

5.4.2 Comparison of Algorithms for VBP

A modified Dijkstra's algorithm for VBP (or FPVB) in networks without LCC was proposed in the previous chapter. This algorithm is adopted for VBP in LCC-overlay network for comparison. Both *MinVBP* and the modified Dijkstra's algorithm are executed in various simulated large-scale overlay networks with the same settings as described in the previous subsection. The data size is set to 3 GBytes. The percentage of the network instances in which *MinVBP* outperforms the modified Dijkstra's algorithm against the ratio of the overlay network size over the lower-layer network size is plotted in Fig. 5.6, while in the rest network instances, these two algorithms achieve the same performance. For the network instances in which *MinVBP* outperforms the modified Dijkstra's algorithm, the average data transfer end time computed by these two algorithms is plotted in Fig. 5.7.

The performance of *MinVBP* illustrated in Fig. 5.6 and Fig. 5.7 is very similar to the performance of *MaxBW*. When the number of overlay nodes increases, the number of link correlations decreases. Hence, the computed path has a relatively higher bandwidth and the transfer end time decreases accordingly.

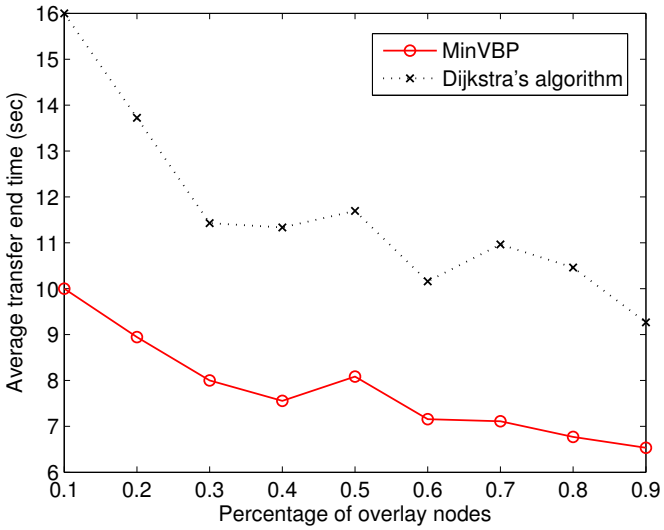

Figure 5.7: Average data transfer end time versus the ratio of the overlay network size over the lower-layer network size.

Chapter 6

Distributed Bandwidth Scheduling

6.1 Problem Formulation

Most existing scheduling algorithms are focused on centralized advance bandwidth reservation in high-performance networks [19] using a centralized control plane. We formulate four basic advance bandwidth scheduling problems and then develop distributed solutions: Given a graph $G = (E, V)$ with a time-bandwidth list TB for each link $l \in E$, source v_s and destination v_d,

- Fixed-Bandwidth: Compute a path from v_s to v_d with a fixed bandwidth β in time slot $[t_s, t_e]$.
- Highest-Bandwidth: Compute a path from v_s to v_d with the highest available bandwidth in time slot $[t_s, t_e]$.
- First-Slot: Compute the earliest start time of a path from v_s to v_d with a fixed bandwidth β for duration t_d.
- All-Slots: Compute all start time slots of all paths from v_s to v_d with a fixed bandwidth β for duration t_d.

Note that the solution to the First-Slot problem is the earliest start time, while the solution to the All-Slots problem is a union of all feasible start times. If t is a feasible start time in the solution for all-slots, the computed path has bandwidth β from t to $t + t_d$.

6.2 Distributed Routing and Bandwidth Scheduling Algorithms

We propose an optimal bandwidth scheduling algorithm in a distributed manner for each of these problem. The proposed algorithms are based on the BFS and Bellman-Ford algorithms and are different from the existing link-state and distance-vector routing protocols: a node makes a routing decision based on its local TB lists and connectivity information, and only broadcasts its own information to its neighbor nodes. Although link-state routing protocols are easy to implement, the periodical broadcasting of node connectivity and link TB lists incurs a significant amount of overhead. Furthermore, if the changes in node connectivity and link bandwidth are not promptly updated, the network may operate with inaccurate information.

We use two types of routing messages between nodes for distributed path exploration for advance bandwidth scheduling: (i) bandwidth reservation message, and (ii) acknowledgment (ACK) message. The bandwidth scheduler running on every node incorporates the scheduling algorithms and handles the routing messages during path exploration. Node and link states can be updated by simple periodic message exchanges between neighbor nodes. Every node broadcasts a HELLO message to its neighbor nodes. Upon the receival of a HELLO message, a node simply replies with an ACK message to let the neighbor node know that the link between them is active.

6.2.1 Fixed-Bandwidth

Given a fixed-bandwidth (FB) reservation request r_i^{FB}, the fixed-bandwidth scheduling problem is to compute a dedicated channel from source node v_s to destination node v_d with specified bandwidth β in time slot $[t_s, t_e]$. The source node v_s receives r_i^{FB} from an end user, and initiates path exploration by broadcasting r_i^{FB} to its neighbor nodes. When an intermediate node receives r_i^{FB} from one of its neighbor nodes, it checks the TB lists of its outgoing links and determines whether r_i^{FB} can be scheduled on these links. An example of bandwidth reservation process is shown in Fig. 6.1. After receiving r_i^{FB} from node v_1, node v_2 checks the TB lists of three outgoing links $(v_2, v_3), (v_2, v_4)$ and (v_2, v_5). If r_i^{FB} is feasible only on links (v_2, v_3) and (v_2, v_5), node v_2 sends r_i^{FB} to nodes v_3 and v_5. Once r_i^{FB} reaches the destination node v_d, node v_d replies with a positive acknowledgment message, which is echoed all the way back to the source node.

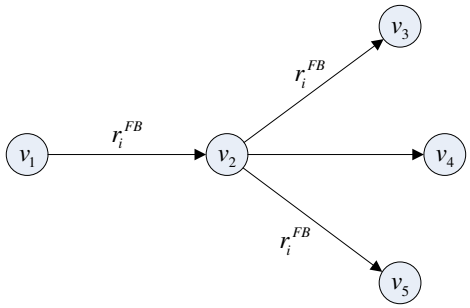

Figure 6.1: An example of bandwidth reservation message processing for fixed-bandwidth problem.

The algorithm details for the fixed-bandwidth scheduling problem are provided in Algorithm 14. Each node maintains a job queue Q that stores bandwidth reservation requests. When a bandwidth reservation request arrives, Q is dynamically updated and the request

Algorithm 14 Routing algorithm for fixed-bandwidth problem

1: A job queue Q is allocated for storing all requests.
2: Listening to routing messages.
3: **if** receive a fixed-bandwidth reservation request r_i^{FB} from its neighbor node u **then**
4: Add u to V_i^{pre}.
5: **if** r_i^{FB} is not in Q **then**
6: Add r_i^{FB} to Q and mark r_i^{FB} as "pending".
7: **if** I am the destination node of r_i^{FB} **then**
8: Send a positive acknowledgment of r_i^{FB} to u.
9: **else**
10: Compute the neighbor node set S_i (excludes u) such that r_i^{FB} can be scheduled on each link between current node and any neighbor node in S_i. If $S_i \neq \emptyset$, broadcast r_i^{FB} to all nodes in S_i; otherwise, send a negative acknowledgment of r_i^{FB} to u and mark r_i^{FB} as "failed". Initialize $n_i = |S_i|$.
11: **end if**
12: **else if** r_i^{FB} is marked as "failed" in Q **then**
13: Send a negative acknowledgment of r_i^{FB} to u.
14: **end if**
15: Return to line 2.
16: **end if**
17: **if** receive an acknowledgment of request r_i^{FB} from its neighbor node u and u is in S_i **then**
18: Remove u from S_i.
19: **if** the acknowledgment is positive **then**
20: Allocate the bandwidth on the link between the current node and node u for r_i^{FB}. Mark r_i^{FB} as "successful". Send a positive acknowledgment of r_i^{FB} to the first node that is added to V_i^{pre}.
21: **else**
22: $n_i = n_i - 1$.
23: **if** $n_i \leq 0$ **then**
24: Mark r_i^{FB} as "failed". Send a negative acknowledgment of r_i^{FB} to all nodes in V_i^{pre}.
25: **end if**
26: **end if**
27: Return to line 2.
28: **end if**

state is changed. A bandwidth reservation request in Q is in one of three states: "pending", "successful" and "failed". The scheduling daemon waits for control messages and processes bandwidth reservation messages in lines 3-16 and acknowledgment messages in lines 17-28. (i) When the current node receives a fixed-bandwidth reservation request r_i^{FB} from its neighbor node u, the algorithm first adds u to a node set V_i^{pre} that stores the all the previous node from which r_i^{FB} is received and is used for sending back the acknowledgment. The algorithm then checks whether r_i^{FB} is in Q. If r_i^{FB} is not in Q, the algorithm adds r_i^{FB} to Q, marks r_i^{FB} as "pending", and replies with a positive acknowledgment if r_i^{FB} is destined to itself; otherwise, the algorithm computes the potential qualified neighbor node set S_i. A neighbor node is qualified if r_i^{FB} can be successfully scheduled on the link between the current node and that neighbor node based on its time-bandwidth list. If S_i is not empty, the current node broadcasts r_i^{FB} to all nodes in S_i, which is an expansion performed in BFS. If S_i is an empty set, which means that there does not exist any qualified neighbor node, the current node replies with a negative acknowledgment to u where r_i^{FB} received from and mark r_i^{FB} in Q as "failed". In the case that r_i^{FB} is in Q but r_i^{FB} is already marked as "failed", the current node sends a negative acknowledgment of r_i^{FB} to u since there does not exist any feasible path that passes the current node to satisfy r_i^{FB}. (ii) When the current node receives an acknowledgment of r_i^{FB} from its neighbor u and u is in S_i, the algorithm removes u from S_i to avoid receiving duplicate acknowledgments from the same neighbor node and checks whether the acknowledgment is positive or negative. If the acknowledgment is positive, the algorithm allocates the bandwidth on the link between current node and u for r_i^{FB} and sends a positive acknowledgment of r_i^{FB} to the first node that is added to V_i^{pre}. Otherwise, the algorithm decreases n_i by 1. Note that n_i is initialized to be $|S_i|$ in line 10 and is used to

count the number of received negative acknowledgments from neighbor nodes in S_i. If n_i reaches 0, which indicates that the current node receives negative acknowledgments from all nodes in S_i and there does not exist a qualified link of the current node, the algorithm marks r_i^{FB} as "failed" and sends a negative acknowledgment of r_i^{FB} to all nodes in V_i^{pre}. An example of the algorithm dealing with acknowledgment messages is shown in Fig. 6.2, where the solid line represents the fixed-bandwidth reservation request and the dashed line represents the acknowledgment. The current node is v_3 that receives r_i^{FB} from both v_1 and v_2, and broadcasts it to v_4 and v_5. In this example, $V_i^{pre} = \{v_1, v_2\}$, $S_i = \{v_4, v_5\}$. The positive acknowledgment process is shown in Fig. 6.2 (a): once v_3 receives a positive acknowledgment of r_i^{FB} from one node in S_i (v_5), it sends the acknowledgment to the node that is firstly added to V_i^{pre} (v_1). The negative acknowledgment process is shown in Fig. 6.2 (b): only when v_3 receives negative acknowledgment of r_i^{FB} from all nodes in S_i, v_3 broadcasts the negative acknowledgment to all nodes in V_i^{pre}. The routing exploration for r_i^{FB} is terminated when the source node of r_i^{FB} receives an acknowledgment.

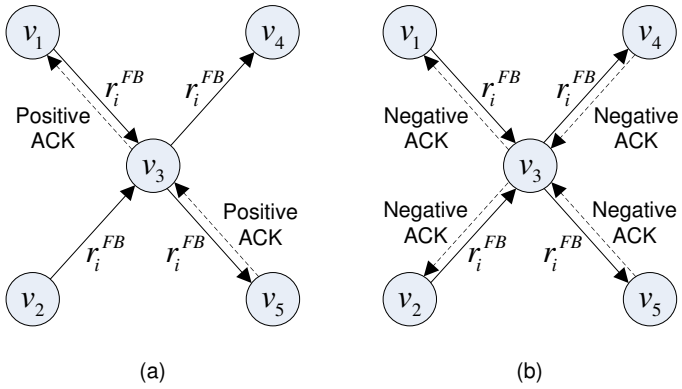

Figure 6.2: An example of acknowledgment message processing for fixed-bandwidth problem.

Performance Tuning

Algorithm 14 is simple and scalable, but some extra work is needed to improve its performance. A deadlock may occur during the acknowledgment message processing, as shown in Fig. 6.3, where there is a cycle of r_i^{FB} among v_2, v_3 and v_4, but v_2 only sends r_i^{FB} to v_3 once. With the qualified neighbor node set $S_i = \{v_2, v_5\}$ for r_i^{FB}, v_4 receives a negative acknowledgment from v_5 and is waiting for the acknowledgment from v_2 before sending any acknowledgments to v_3. However, v_2 is waiting for the acknowledgement from v_3 and v_3 is waiting for the acknowledgment from v_4. Therefore, there is a deadlock among v_2, v_3 and v_4. To address this problem, a set of nodes that a bandwidth reservation request have traversed can be encoded. When the algorithm computes S_i for r_i^{FB}, S_i only includes the qualified neighbor nodes that are not in the set of nodes that r_i^{FB} have traversed. Therefore, there is no bandwidth reservation request from v_4 to v_2 in the above example and the deadlock is avoided.

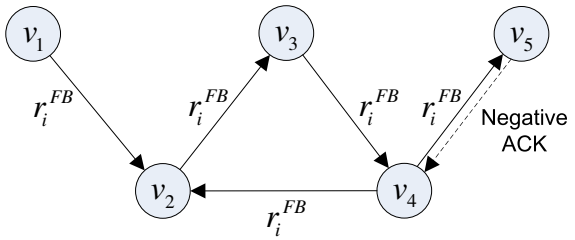

Figure 6.3: An example of deadlock in Algorithm 14.

In a special case that a neighbor node u in S_i breaks down right after the current node broadcasting r_i^{FB}, and the current node receives negative acknowledgments from all the nodes in S_i except u, n_i never reaches 0 in line 23 of Algorithm 14. If there does not exist a feasible path to satisfy r_i^{FB}, the source node of r_i^{FB} may never receive a negative

acknowledgment. The solution is that detecting the breakdown of a neighbor node u in S_i should be equivalent to receiving a negative acknowledgment from u for the pending bandwidth reservation request r_i^{FB}. Also, all the scheduled bandwidth reservations that using u must be canceled. The current node sends a CANCEL message along the path for each scheduled bandwidth reservation request using the current node and u, and the reserved bandwidth on the corresponding links will be released. Once the source node of a bandwidth reservation request receives a CANCEL message, the source node initiates another routing exploration to find a new path. The handling of node failures can be applied to the rest algorithms.

Algorithm Analysis

Algorithm 14 exhibits several salient features.

(i) Loop free: The job queue that maintains all incoming bandwidth reservation requests and the verification condition in line 5 ensure that each node broadcasts a bandwidth reservation request at most once. Hence, there is no loop for a bandwidth reservation request. Further more, the condition of whether u is in S_i in line 17 and update of S_i in line 18 ensure that each node receives at most one acknowledgment from a neighbor node in S_i. Hence, there is no loop for an acknowledgment.

(ii) Fault tolerant: Any node and link failure can be detected by the periodical HELLO messages exchanged between nodes. Hence, any node failure does not affect the routing exploration process if there still exists a feasible path.

(iii) Time efficient: The running time complexity of this algorithm is $O(T \cdot m)$ in the worst case. Unlike most distributed routing algorithms where each node must wait for a

constant time period to collects all messages from its neighbor nodes, the algorithm processes each incoming routing message immediately to speed up path exploration. In the worst case, the algorithm involves $O(m)$ message communications in the entire network.

6.2.2 Highest-Bandwidth

Given a highest-bandwidth reservation quest r_i^{HB}, the highest-bandwidth scheduling problem is to compute a dedicated channel from source node v_s to destination node v_d with the highest available bandwidth during time slot $[t_s, t_e]$. This problem can be solved by extending Dijkstra's shortest path algorithm in the centralized scheme. We propose a distributed solution based on Bellman-Ford algorithm to this problem. The source node v_s receives r_i^{HB} from an end user, initializes the highest bandwidth of r_i^{HB} to be infinity, and initiates the path exploration process by broadcasting r_i^{HB} to its neighbor nodes. The neighbor nodes computes their highest bandwidth according to the incoming r_i^{HB}, and broadcast their results only if the value is increased. Note that the highest available bandwidth of the entire path is determined by the bottleneck bandwidth of all component links in the specified time slot. Hence, the highest bandwidth of r_i^{HB} on each node is dynamically updated during the path exploration process.

The algorithm details for the highest-bandwidth scheduling problem are provided in Algorithm 15. Let $BW_i(v_s, v_{cur})$ denote the highest bandwidth of the path found so far from the source node v_s to the current node v_{cur} for r_i^{HB} in Q, and $BW_i'(v_s, v_{cur})$ denote that for the incoming r_i^{HB}. The algorithm waits for control messages, and processes bandwidth reservation messages in lines 3-22 and acknowledgment messages in lines 23-26. When the current node receives a highest-bandwidth reservation request r_i^{HB} from its neighbor node u,

the algorithm checks whether r_i^{HB} is in Q. If r_i^{HB} is not in Q, the algorithm adds r_i^{HB} to Q. If the highest bandwidth of the incoming r_i^{HB} is larger than that of r_i^{HB} in Q ($BW_i'(v_s, v_{cur}) > BW_i(v_s, v_{cur})$), the algorithm updates the highest bandwidth of r_i^{HB} in Q. Then the algorithm computes the highest bandwidth of the path found so far from the source node v_s to every neighbor node v by calculating $BW_i(v_s, v) = min\{BW_i(v_s, v_{cur}), BW_i(v_{cur}, v)\}$, where $BW_i(v_{cur}, v)$ is the highest bandwidth of the link (v_{cur}, v) during the time slot specified in r_i^{HB}. The algorithm encodes $BW_i(v_s, v)$ in r_i^{HB} and sends r_i^{HB} to v. If the highest bandwidth of r_i^{HB} in Q does not increase ($BW_i'(v_s, v_{cur}) \leq BW_i(v_s, v_{cur})$), the algorithm returns to line 2 directly to avoid message broadcasting. If the current node is the destination node of r_i^{HB}, the algorithm restarts a timer for r_i^{HB}. This timer is used by the destination node to acknowledge the granting of the request since the destination node does not know when the path exploration process reaches an equilibrium. If the destination node does not receive any updated r_i^{HB} from its neighbor nodes for a period of time, it is very likely that the path exploration process for r_i^{HB} has reached an equilibrium. Once the timer for r_i^{HB} expires, the destination node determines the highest bandwidth of the entire path and sends an acknowledgment of r_i^{HB} that carries the highest bandwidth to v_i^{pre}, which is the best neighbor node on the widest path from the source node to the current node. When the current node receives an acknowledgment of r_i^{HB} from its neighbor u, it allocates the bandwidth on the link between the current code and node u, and forwards the acknowledgment of r_i^{HB} to node v_i^{pre}. This backtracking process continues until the source node of r_i^{HB} receives the acknowledgment.

Algorithm 15 Routing algorithm for highest-bandwidth problem

1: A job queue Q is allocated for storing all requests.
2: Listening to routing messages.
3: **if** receive a highest-bandwidth reservation request r_i^{HB} from its neighbor node u **then**
4: **if** I am the destination node of r_i^{HB} **then**
5: Restart a timer for r_i^{HB}.
6: **end if**
7: **if** r_i^{HB} is not in Q **then**
8: Add r_i^{HB} to Q.
9: **else if** $BW_i'(v_s, v_{cur}) > BW_i(v_s, v_{cur})$ **then**
10: $BW_i(v_s, v_{cur}) = BW_i'(v_s, v_{cur})$
11: **else**
12: Return to line 2.
13: **end if**
14: Set $v_i^{pre} = u$.
15: **if** I am not the destination node of r_i^{HB} **then**
16: Compute the neighbor node set S_i (exclude u).
17: **for all** $v \in S_i$ **do**
18: $BW_i(v_s, v) = min\{BW_i(v_s, v_{cur}), BW_i(v_{cur}, v)\}$. Encode $BW_i(v_s, v)$ to r_i^{HB} and send r_i^{HB} to v.
19: **end for**
20: **end if**
21: Return to line 2.
22: **end if**
23: **if** receive an acknowledgment of request r_i^{HB} from its neighbor node u **then**
24: Allocate the bandwidth on the link between the current node and node u for r_i^{HB}. Forward acknowledgment of r_i^{HB} to node v_i^{pre}.
25: Return to line 2.
26: **end if**

Performance Tuning

The time cost of the path exploration process is affected by the timer on the destination node, which needs to be carefully decided according to the network size and link delay. Let *DELAY* denote the average delay of a message communication between two adjacent nodes, which includes the processing delay on two end nodes and the link delay between them. A node can estimate *DELAY* by measuring the round trip time of a message between itself and its neighbor nodes. Since a bandwidth reservation message traverses at least 1

hop and at most $n-1$ hops from v_s to v_d, the difference between the arrival time of any two request messages is at most $(n-2) \cdot DELAY$, which could be used to set the timer on v_d.

Algorithm Analysis

Algorithm 15 also exhibits similar features as Algorithm 14.

(i) Loop free: A node broadcasts r_i^{HB} to its neighbor nodes only when the highest bandwidth of r_i^{HB} increases. When a node broadcasts r_i^{HB} to its neighbor nodes, the highest bandwidth of r_i^{HB} does not increase during the path exploration process. Hence, there is no loop for a bandwidth reservation request. Since node v_i^{pre} is set only when the highest bandwidth of r_i^{HB} increases and the acknowledgment is sent to v_i^{pre}, there is no loop for an acknowledgment, either.

(ii) Fault tolerant: Since each node makes a local decision and acts as an autonomous system, a node or link failure would not affect the path exploration process. If one node on the computed path for r_i^{HB} breaks down after an equilibrium is achieved but before the acknowledgment of r_i^{HB} is forwarded, the source node of r_i^{HB} will never receive the acknowledgment. This problem can be solved as follows. The failure of a node can be detected by its neighbor nodes by periodical HELLO message exchanges between them. Each neighbor node then sends a negative acknowledgment of r_i^{HB} to its v_i^{pre}. The source node of r_i^{HB} eventually receives the negative acknowledgment of r_i^{HB} and may initiate another path exploration process for r_i^{HB}.

(iii) Time efficient: The running time complexity of this algorithm is $O(m \cdot T)$ in the worst case. Except the destination node, all other nodes process each incoming routing

message immediately. In the worst case, the algorithm requires $O(n^3)$ message broadcasting as the distributed Bellman-Ford algorithm.

6.2.3 First-Slot and All-Slots

Given a first-slot or all-slots bandwidth reservation request, r_i^{FS} or r_i^{AS}, the first-slot or all-slots bandwidth scheduling problem is to compute the time slot with the earliest start time or all possible time slots of a dedicated channel from v_s to v_d with a fixed bandwidth β for duration t_d. Obviously, first-slot is a special case of all-slots, and the solution to all-slots can be applied to first-slot. We propose a distributed routing algorithm based on Bellman-Ford algorithm for these two problems.

A list of start time $[t_i, t_{i+1}]$ is defined for each link $l \in E$, denoted as $ST(l)$. For any time point t during a start time slot $[t_i, t_{i+1}]$, i.e. $t \in [t_i, t_{i+1}]$, link l has available bandwidth of β from time point t to time point $t + t_d$. The time slots on ST are disjoint and arranged in an ascending order. The ST list of a link can be constructed from its TB list in $O(T)$ time, and the ST list of a path can be constructed by combining the ST lists of all component links. Let $ST(v_s, v)$ denote the union of the ST lists of all paths from source node v_s to node v. Hence, $ST(v_s, v_d)$ contains all start time slots of all paths from v_s to v_d with bandwidth β for duration t_d. Let $\bigoplus$ and $\bigotimes$ denote the point-wise merging and intersection operations of the time slots in two ST lists, respectively. $ST(l) \bigoplus \emptyset = ST(l)$, $ST(l) \bigoplus \Re^+ = \Re^+$, $ST(l) \bigotimes \emptyset = \emptyset$, and $ST(l) \bigotimes \Re^+ = ST(l)$, where $\emptyset$ is the empty time slot and $\Re^+$ is the infinite time slot of non-negative real values.

The algorithm details for the all-slots bandwidth scheduling problem are provided in Algorithm 16. The source node v_s receives r_i^{AS} from an end user, initializes the ST list of

Algorithm 16 Routing algorithm for all-slots problem

1: A job queue Q is allocated for storing all requests.
2: Listening to routing messages.
3: **if** receive an all-slots bandwidth reservation request r_i^{AS} from its neighbor node u **then**
4: **if** I am the destination node of r_i^{AS} **then**
5: Restart a timer for r_i^{AS}.
6: **end if**
7: **if** r_i^{AS} is not in Q **then**
8: Add r_i^{AS} to Q. Set $v_i^{pre} = u$.
9: **else if** $ST_i'(v_s, v_{cur}) \subsetneq ST_i(v_s, v_{cur})$ **then**
10: $ST_i(v_s, v_{cur}) = ST_i(v_s, v_{cur}) \bigoplus ST_i'(v_s, v_{cur})$.
11: **else**
12: Return to line 2.
13: **end if**
14: **if** I am not the destination node of r_i^{AS} **then**
15: Compute the neighbor node set S_i (exclude u).
16: **for all** $v \in S_i$ **do**
17: $ST_i(v_s, v) = ST_i(v_s, v_{cur}) \bigotimes ST_i(v_{cur}, v)$. Encode $ST_i(v_s, v)$ to r_i^{AS} and send r_i^{AS} to v.
18: **end for**
19: **end if**
20: Return to line 2.
21: **end if**
22: **if** receive an acknowledgment of request r_i^{AS} from its neighbor node u **then**
23: Forward acknowledgment of r_i^{AS} to node v_i^{pre}.
24: Return to line 2.
25: **end if**

r_i^{AS} to be $ST(v_s, v_s) = \Re^+$ and initiates the path exploration process by broadcasting r_i^{AS} to its neighbor nodes. Let $ST_i(v_s, v_{cur})$ denote the list of start time slots of the paths found so far from the source node v_s to the current node v_{cur} for r_i^{AS} in Q, and $ST_i'(v_s, v_{cur})$ denote that for the incoming r_i^{AS}. The algorithm is modified from Algorithm 15 by replacing the bandwidth operation with the ST list operation. If the ST list of the incoming r_i^{AS} is not a subset of the ST list of r_i^{AS} in Q (i.e. $ST_i'(v_s, v_{cur}) \subsetneq ST_i(v_s, v_{cur})$), the algorithm updates the ST list of r_i^{AS} in Q (i.e. $ST_i(v_s, v_{cur}) = ST_i(v_s, v_{cur}) \bigoplus ST_i'(v_s, v_{cur})$). Here, the relationship $\subsetneq$ of two ST lists holds if at least one time slot in $ST_i'(v_s, v_{cur})$ does not belong to any time

slots on $ST_i(v_s, v_{cur})$. Due to the monotonicity property of $\bigoplus$ operation, once start times are placed on $ST_i(v_s, v_{cur})$, they will not be removed. Then the algorithm computes the start time slots of all paths found so far from the source node v_s to every neighbor node v by calculating $ST_i(v_s, v) = ST_i(v_s, v_{cur}) \bigotimes ST_i(v_{cur}, v)$, where $ST_i(v_{cur}, v)$ is the ST list of link (v_{cur}, v) for r_i^{AS}. The algorithm encodes $ST_i(v_s, v)$ to r_i^{AS} and send r_i^{AS} to v. If the current node is the destination node of r_i^{AS}, the algorithm restarts a timer for r_i^{AS}. Once the timer for r_i^{AS} expires, the destination node sends an acknowledgment of r_i^{AS} that carries all start time slots $ST_i(v_s, v_d)$ to v_i^{pre}.

For first-slot problem, the earliest start time is the lower boundary of the first time slot on the returned ST list. For the all-slots problem, the end user at the source node may choose one or multiple start times from the returned ST list. Once the start time t for a feasible path is decided, Algorithm 14 for the fixed-bandwidth problem can be applied to perform the actual path computation and bandwidth scheduling with $t_s = t$ and $t_e = t + t_d$.

The runtime complexity of Algorithm 16 is $O(m)$ in terms of $\bigoplus$ and $\bigotimes$ operations. Since the complexities of $\bigoplus$ and $\bigotimes$ operations are determined by the length of the ST list, which is at most $m \cdot T$ in the algorithm, the complexities of $\bigoplus$ and $\bigotimes$ operations are of $O(m \cdot T)$. Therefore, the algorithm complexity is $O(m^2 \cdot T)$ in the worst case. Due to the similarity in the algorithm structure, the performance tuning and algorithm analysis for Algorithm 15 are applicable to Algorithm 16.

6.3 Performance Evaluation

Simulation-based evaluations are conducted for the proposed distributed scheduling algorithms. For performance comparison, we also design and implement a simple greedy

algorithm. In the simulations, each simulated network is randomly generated with an arbitrary network topology with 50 nodes and 200 links, and the TB list of each link is also randomly generated with residual bandwidths ranging from 0.2 Gbps to 10 Gbps in each time slot with an identical length of 1 second. The residual bandwidths follow a normal distribution:

$$b_l[i] = 0.2 + 10 \cdot (1 - e^{-\frac{1}{2}(3x)^2}), \tag{6.3.1}$$

where x is a random variable within the range of [0,1]. There are 600 time slots in the time-bandwidth list of each link.

6.3.1 Comparison of Algorithms for Fixed-bandwidth

Performance comparison between Algorithm 14 and the traceroute based algorithm is conducted for the fixed-bandwidth problem using various simulated networks. Note that traceroute is implemented in OSCARS to find the shortest path within ESnet that MPLS LSP traverses [28]. Once the entire path controlled by OSCARS is obtained, each link on the path is then checked for available bandwidth.

Fixed-bandwidth is a decision problem, and the satisfiability of a fixed-bandwidth request is determined by the availability of the network resource. Algorithm 14 is an optimal algorithm that is able to find a feasible solution when there exists one. The simulation randomly generates 200 network instances of different topologies, in each of which, the simulation randomly generate a series of fixed-bandwidth requests with requested bandwidth β ranging from 0.24 Gbps to 2.4 Gbps at an interval of 0.24 Gbps. The duration of a request $t_e - t_s$ is constrained within the the range of [1, 10]. Then Algorithm 14 and

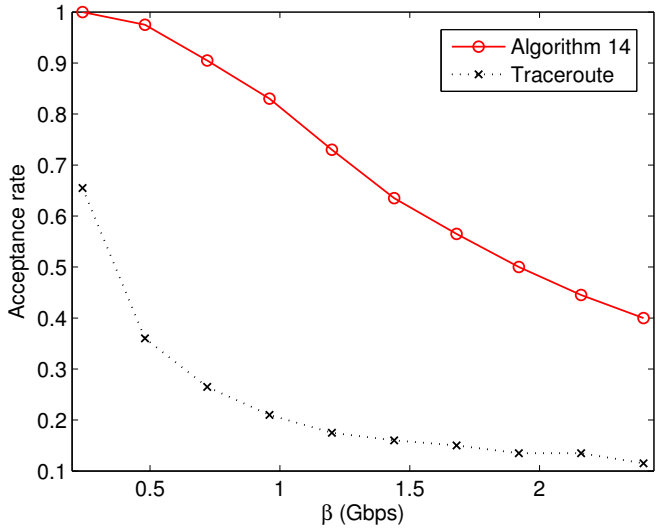

Figure 6.4: Acceptance rate comparison between Algorithm 14 and traceroute for fixed-bandwidth problem.

traceroute are run on these fixed-bandwidth requests and a series of acceptance rates in response to different β values are plotted in Fig. 6.4. The acceptance rate is defined as the ratio of successfully scheduled requests and 200 submitted requests. The figure shows that Algorithm 14 exhibits superior performance over the traceroute-based method. Since the requests with larger β values require more network resources, the acceptance rate decreases as β increases.

6.3.2 Comparison of Algorithms for Highest-bandwidth

The performance of Algorithm 15 is compared with that of a greedy algorithm for the highest-bandwidth problem. In the greedy algorithm, a node always chooses one neighbor node whose link has the highest available bandwidth in the specified time slot. In each of 200 randomly generated network instances, a series of highest-bandwidth requests with duration $t_d = t_e - t_s$ ranging from 1 to 10 seconds at an interval of 1 second are generated.

Algorithm 15 and the greedy algorithm are run on these highest-bandwidth requests and the average and standard deviation of the highest available bandwidth in response to different t_d values are plotted in Fig. 6.5, which shows that Algorithm 15 outperforms the greedy approach in all the cases. The figure also shows that the average highest available bandwidth decreases as t_d increases.

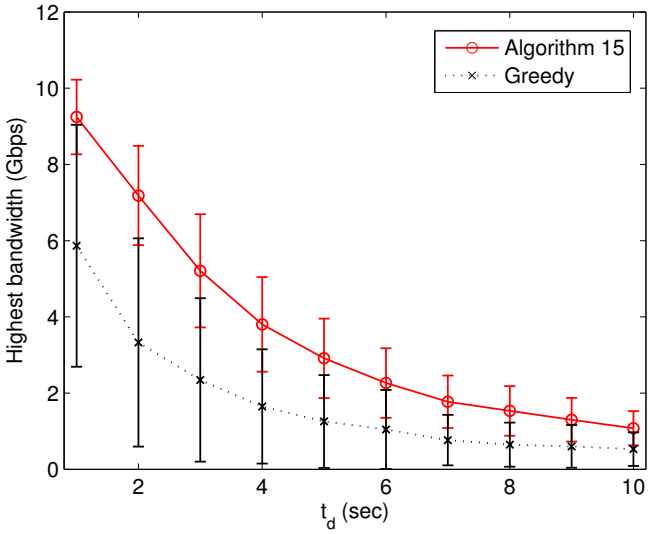

Figure 6.5: Highest bandwidth comparison (mean and standard deviation) between Algorithm 15 and greedy algorithm for highest-bandwidth problem.

6.3.3 Comparison of Algorithms for First-slot and All-slots

We design Algorithm 16 for both first-slot and all-slots problems. The performance of Algorithm 16 is compared with that of a greedy method for the first-slot problem. In the greedy algorithm, a node always chooses one neighbor node such that the earliest start time of the path from the source node to the neighbor node for that request is minimized. A series of first-slot requests with $t_d = 5$ seconds and requested bandwidth β ranging from 0.24 Gbps to 2.4 Gbps at an interval of 0.24 Gbps are randomly generated. The average

and standard deviation of the earliest start time in response to different β values for both algorithms are plotted in Fig. 6.6. In most of the cases, the earliest start time computed by Algorithm 16 is 0 second. The largest earliest start time is 600 seconds since there is no bandwidth reservation on each link after 600 seconds. The figure shows that the average of 200 earliest start time computed by Algorithm 16 is much less than that computed by the greedy method.

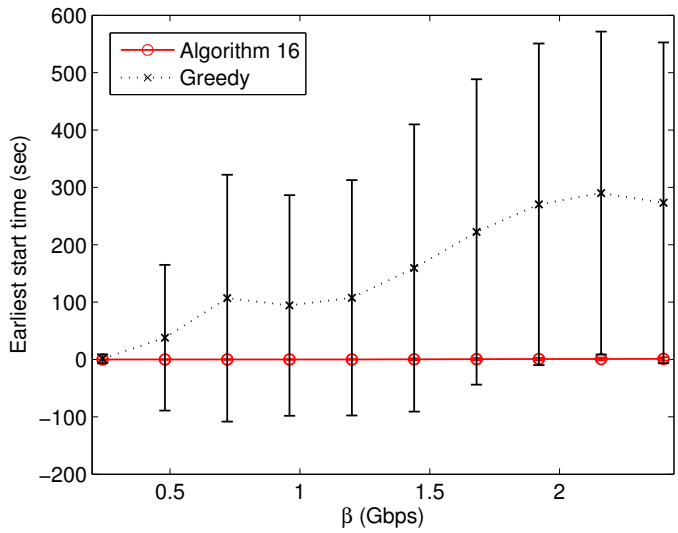

Figure 6.6: Earliest start time comparison (mean and standard deviation) between Algorithm 16 and greedy algorithm for first-slot problem.

The performance of Algorithm 16 is also compared with that of a greedy algorithm for the all-slots problem. The objective function in the all-slots problem is the total length of start times. In the greedy method, a node always chooses one neighbor node such that the total length of start times of the path from the source node to the neighbor node for that request is maximized. The simulation settings for the all-slots problem are the same as those for the first-slot problem. The average and standard deviation of the total length of

start times in response to different β values are plotted in Fig. 6.7, which shows that the performance superiority of Algorithm 16 is dominant in all cases.

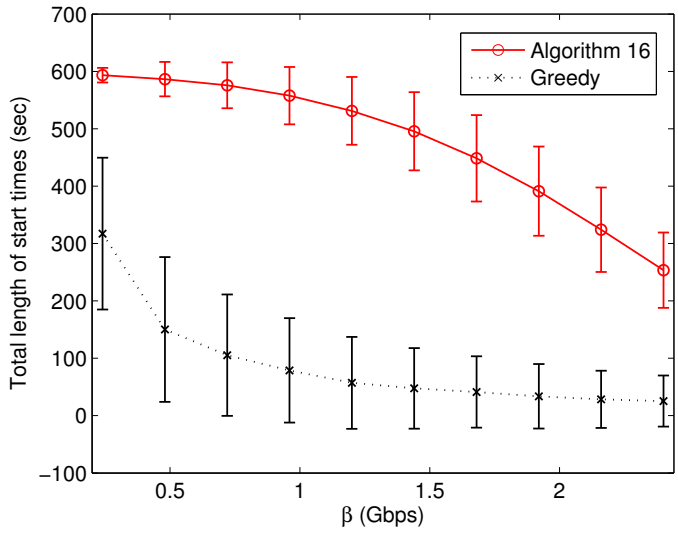

Figure 6.7: Total length of start times comparison (mean and standard deviation) between Algorithm 16 and greedy algorithm for all-slots problem.

Chapter 7

Periodical Bandwidth Scheduling

7.1 Problem Formulation

A periodical scheduling algorithm is launched periodically in a certain time interval to schedule multiple data transfer requests accumulated during that interval. We study two periodic multiple data transfers scheduling problems: (i) multiple data transfer allocation (MDTA) to assign multiple data transfer requests on several pre-specified network paths for the minimum total data transfer end time, and (ii) multiple fixed-slot bandwidth reservation (MFBR) to satisfy multiple fixed time slot bandwidth reservation requests. These two problems are formally defined as follows.

- **Multiple Data Transfer Allocation (MDTA)**

  Given p pre-specified disjoint paths from a source vertex s to a destination vertex d with an identical bandwidth b and k files of sizes $\delta_1, \delta_2, ..., \delta_k$, find a task assignment scheme that allocates k files on p paths to minimize the total file transfer end time.

- **Multiple Fixed-slot Bandwidth Reservation (MFBR)**

  Given a graph $G = (E,V)$ with an aggregated time-bandwidth list TB_l combining the reservation information on all links $l \in E$ and k fixed-slot bandwidth requests,

each of which specifies a source vertex s_i and a destination vertex d_i, a bandwidth β_i, and a future time slot (t_i^s, t_i^e), where $i = 1, 2, ..., k$, find a scheduling scheme that computes network paths and allocates link bandwidths to maximize the number of satisfied reservations.

In the MDTA problem, the user, whose primary interest is in completing the file transfer task as soon as possible, does not need to specify the bandwidth and time slot for each task. To minimize the total transfer end time, it is desired that the residual bandwidths of each path be reserved for one file transfer at a time. In the case where only one path is considered, the well-known Shortest Job First (SJF) algorithm finds the optimal solution. When the number of paths is greater than one, the problem boils down to computing a strategy that optimally distributes all file transfer tasks onto the pre-specified paths.

In the MFBR problem, each request is to find a path from a source vertex s_i to a destination vertex d_i with a residual bandwidth of at least β_i from start time t_i^s to end time t_i^e. Once a satisfied path for a particular request is obtained, the residual bandwidths of the component links along that path are updated by subtracting the bandwidths reserved for the current request. Hence, there is no guarantee that all bandwidth reservation requests could be satisfied. The objective is to maximize the number of successful bandwidth reservations given the network topology, residual bandwidths, and user requests.

7.2 Complexity Analysis and Algorithm Design

In this section, we design an optimal algorithm and provide the correctness proof for MDTA. Also, we prove MFBR to be NP-complete and propose two heuristic algorithms.

7.2.1 Optimal Algorithm for MDTA

We design an optimal algorithm based on an extension of the Shortest Job First algorithm (ESJF) for MDTA, which takes as input p disjoint paths from source s to destination d with identical bandwidth b and k files of sizes $\delta_1, \delta_2, ..., \delta_k$, and targets minimizing the total transfer end time. The pseudo-code for the algorithm $ESJF(p, \{P\}, b, k, \{\delta\})$, where $\{P\}$ denotes the set of p disjoint paths with bandwidth b and $\{\delta\}$ denotes the set of k file sizes, is shown in Algorithm 17. The ESJF algorithm starts with sorting the files in an increasing order by their sizes, and then take turns to assign each file to one of the p paths. This assignment evenly distributes smaller files onto multiple paths, which are transferred before larger files. Apparently, transferring a smaller file before a larger one reduces the total waiting time, resulting in a shorter total transfer end time. Once the assignment is completed, the files on each path are transferred using the SJF algorithm.

Algorithm 17 $ESJF(p, \{P\}, b, k, \{\delta\})$

```
Sort the files in {δ} in an increasing order by sizes;
i = 1; j = 1;
while (i ≤ k) do
   Assign the i-th file in {δ} to the j-th path in {P};
   i = i + 1;
   j = j + 1;
   if j > p then
      j = 1;
   end if
end while
```

Theorem 9. *ESJF produces the minimal total transfer end time for a given set of files.*

Proof. To simplify the proof, we show a special case where $p = 2$, which can be extended to a general case where $p > 2$. We first consider an even number $k = 2n$ of files sorted by their file sizes. The transfer time for each file is $t_i = \delta_i / b$, where $t_1 \leq t_2 ... \leq t_{2n}$. Note that

the transfer end time includes both waiting time and transfer time. After assigning the $2n$ files to two paths using the ESJF algorithm, the total transfer end time on the first path P_1 is:

$$T_{P_1} = t_1 + (t_1 + t_3) + (t_1 + t_3 + t_5) + ... + (t_1 + t_3 + t_5 ... t_{2n-1}), \tag{7.2.1}$$

and the total transfer time on the second path P_2 is

$$T_{P_2} = t_2 + (t_2 + t_4) + (t_2 + t_4 + t_6) + ... + (t_2 + t_4 + t_6 ... t_{2n}). \tag{7.2.2}$$

After adding up Eqs. 7.2.1 and 7.2.2 and rearranging the equation, we obtain the total transfer end time on two paths:

$$\begin{aligned} &T_{total} = T_{P_1} + T_{P_2} = \\ &n \cdot (t_1 + t_2) + (n-1) \cdot (t_3 + t_4) + (n-2) \cdot (t_5 + t_6) \\ &+ ... + (t_{2n-1} + t_{2n}). \end{aligned} \tag{7.2.3}$$

In Eq. 7.2.3, the coefficients of the factors, i.e. $n, (n-1), (n-2), ..., 1$, are in the decreasing order, while the times $(t_1 + t_2), (t_3 + t_4), ..., (t_{2n-1} + t_{2n})$, which could be considered as a set of combined jobs, are in the increasing order. Essentially, Eq. 7.2.3 presents a generalized SJF scheme, which produces the minimal total time among all possible combinations. In other words, if we exchange any t_i with t_j in this equation, the new result will be greater than or equal to Eq. 7.2.3. When there are an odd number $k = 2n - 1$ of files, we could simply add a new file with size 0 to the beginning of the file set, and then the proof remains the same. Note that there are totally 2^n optimal solutions to the MDTA problem when $p = 2$ because there are n combined jobs (t_{2i-1}, t_{2i}), in each of which, either component could be assigned to one of the two paths. Proof ends.

7.2.2 MFBR is NP-Complete

To prove MFBR problem to be NP-Complete, we first define *Widest Pair of Paths*(WPP) problem, which is a simplified version of the MFBR problem, and prove it to be NP-complete. In [37], Shen *et al.* proved that *Widest Pair of Disjoint Paths (Decoupled)* is NP-complete. However, in the WPP problem, the paths do not have to be disjoint.

Widest Pair of Paths (WPP): Given a graph $G = (V, E)$ with available bandwidth b_l on each link $l \in E$, a source vertex s, a destination vertex d, and specified bandwidths β_1 and β_2. QUESTION: Do there exist paths P_1 and P_2 from s to d such that the bandwidth of path P_1, $BW(P_1) \geq \beta_1$ and the bandwidth of path P_2, $BW(P_2) \geq \beta_2$?

Theorem 10. *WPP problem is NP-complete.*

Proof. WPP is obviously in NP because given a solution (a pair of paths) to WPP, one can verify in polynomial time the validity of the solution by checking whether or not the bandwidth of path P_i is greater or equal to β_i, i = 1 or 2. Without loss of generality, we assume that $\beta_1 < \beta_2$.

We now reduce the *Disjoint Path Problem with Red and Blue arcs* (DPPRB) [37] to WPP. The DPPRB problem is defined as follows: Given a graph $G = (V, E)$, where each arc $e \in E$ is colored either red or blue, a source vertex s and and a destination vertex d. QUESTION: Do there exist two disjoint paths from s to d such that at least one of the paths uses the red arcs only?

Let (G, s, d) be an arbitrary instance of DPPRB. We construct an instance $(G', b, s, d, \beta_1, \beta_2)$ of WPP problem from the instance (G, s, d) in polynomial time such that G' has two paths from s to d of bandwidths no less than β_1 and β_2, respectively, if and only if there exist two

disjoint paths from s to d in G, at least one of which uses the red arcs only. We first set $G' = G$ and the available bandwidth b_l of the link $l \in E'$ to 1 if the link l in G is colored blue, otherwise $b_l = 2$. Then we set $\beta_1 = 1$, $\beta_2 = 2$. Clearly, this construction process can be accomplished in polynomial time.

Suppose that there exist two disjoint paths P_1 and P_2 from s to d in G such that at least one of them, say, path P_2, uses the red arcs only. We can find two corresponding paths P_1' and P_2' in G', where $P_1' = P_1$ and $P_2' = P_2$. Since the red arcs composing P_2 have available bandwidth 2 in G', $BW(P_2') = 2$. Also, because the minimal bandwidth of all links is 1, $BW(P_1')$ is at least 1. Hence, P_1' and P_2' compose a solution to WPP.

Conversely, let P_1' and P_2' be a pair of paths from s to d in G' with $BW(P_1') \geq 1$ and $BW(P_2') \geq 2$. It is obvious that P_1' and P_2' are disjoint as the maximum bandwidth of the links is 2. We can find two corresponding paths P_1 and P_2 in G, where $P_1 = P_1'$ and $P_2 = P_2'$. Path P_2 contains red arcs only since blue arcs have bandwidth less than 2. Hence, P_1 and P_2 compose a solution to DPPRB. This concludes the proof.

Theorem 11. *MFBR problem is NP-complete.*

Proof. We could restrict MFBR problem to WPP problem by allowing only instances in which $k = 2$, $s_1 = s_2$, $t_1 = t_2$, and $(t_1^s, t_1^e) = (t_2^s, t_2^e)$. The validity of NP-completeness proof by restriction is established in [23], where "restriction" constrains the given, not the question of a problem. The bandwidth of the link l is defined as the minimal residual bandwidth during the time slot (t_1^s, t_1^e) based on the aggregated time-bandwidth list TB_l. Since WPP is NP-complete, so is MFBR. Proof ends.

7.2.3 Heuristic Algorithms for MFBR

We propose two heuristic algorithms for MFBR. Based on the distribution of the pre-specified time slots of bandwidth reservations in MFBR, there are three cases: (i) When all bandwidth reservations have the same pre-specified time slot, it has been proved to be NP-complete above; (ii) When the pre-specified time slots of bandwidth reservations are partially overlapped, it is still NP-complete for the overlapped portion; (iii) When the pre-specified time slots of bandwidth reservations are complete disjoint, it is polynomially solvable using a modified Dijkstra's algorithm. In the algorithm design, we consider the first case with an identical pre-specified time slot. Here, all bandwidth reservations are assumed to have the same priority.

Greedy Algorithm

The objective of MFBR is to maximize the number of satisfied bandwidth reservations, and hence scheduling the reservations that use less network resources first will return a better result. Since all reservations have the same start and end time, the residual bandwidth of link l is the minimal residual bandwidth among all the time slots between the start and end time based on TB_l, $l \in E$, which is a bottleneck. The greedy algorithm takes as input a graph $G = (V,E)$ with bottleneck bandwidth in a given time slot on each link l, $l \in E$ and k fixed-slot bandwidth reservations R. This algorithm is refered as $Greedy(G,R)$.

We first define several notations and operations to facilitate our explanation:

$Dequeue(R)$**:** dequeue the first element in R.

r**:** a fixed-slot bandwidth reservation.

Algorithm 18 *Greedy*(G,R)

```
Sort the elements in R by their reserved bandwidths in increasing order;
while R ≠ ∅ do
  r = Dequeue(R);
  G = G − E(β^r);
  for all v ∈ V do
    b[v] = ∞;
    prev(v) = NULL;
  end for
  b[v_s^r] = 0;
  S = ∅;
  Q = V[G];
  while Q ≠ ∅ do
    u = ExtractMin(Q);
    S = S ∪ {u};
    for all v ∈ Q, (u,v) ∈ E do
      if b[v] > max(b[u], b_(u,v)) then
        b[v] = max(b[u], b_(u,v));
        prev(v) = u;
      end if
    end for
  end while
  if b[v_d^r] = ∞ then
    no path can be found to satisfy r;
  else
    Construct the path from v_s^r to v_d^r, and reserve the bandwidth β^r on links along the
    path for r;
  end if
end while
```

β^r, v_s^r, v_d^r : the specified bandwidth, source vertex, destination vertex of r, respectively.

$E(\beta^r)$**:** a subset of E, consisting of links whose residual bandwidth is less than β^r.

$G - E(\beta^r)$: the operation of removing the links in $E(\beta^r)$ from G.

$b[v]$**:** the maximum bandwidth of the component links on the path from v_s to v.

S**:** a set of vertices whose final narrowest paths from the source have already been determined.

Q**:** a min-priority queue of vertices, keyed by their bandwidth values.

$ExtractMin(Q)$**:** the operation of extracting the vertex with minimal bandwidth in Q.

The pseudo-code for $Greedy(G,R)$ is shown in Algorithm 18, which sorts all tasks by their requested bandwidths in increasing order, and schedules them in the same order. For each task with a specified bandwidth β, the algorithm first removes the links whose residual bandwidths are less than β since these links do not contribute to the current task and subsequent tasks. It then computes the narrowest path in the residual graph by extending the Dijkstra's algorithm. Here, the narrowest path is defined as the path whose maximum residual bandwidth of all component links on the path from the source vertex to the destination vertex is minimized. Since the computational complexity of the extended Dijkstra's algorithm is $O(|E| \cdot \lg|V|)$ and there are k tasks, the total runtime of the greedy algorithm is $O(k \cdot |E| \cdot \lg|V|)$.

Minimal Bandwidth and Distance Product Algorithm (MBDPA)

The scheduling order of the tasks largely determines the efficiency of a scheduling algorithm. MBDPA considers the product of bandwidth and distance from source to destination as the amount of network resources needed for each task. Here, the distance is counted as the number of hops. The input of MBDPA is the same as the greedy algorithm, and we refer to this algorithm as $MBDPA(G,R)$.

Again, we define several notations and operations to facilitate our explanation:

$d_{(v_s^r, v_d^r)}$**:** the number of hops from v_s^r to v_d^r.

α^r**:** the product of the specified bandwidth β^r and $d_{(v_s^r, v_d^r)}$ for a request r.

ExtractMin(R)**:** the operation of extracting the fixed-slot reservation with minimal product α in R.

Algorithm 19 *MBDPA*(G,R)

while $R \neq \emptyset$ **do**
  for all $r \in R$ **do**
    $G' = G - E(\beta^r)$;
    compute $d_{(v_s^r, v_d^r)}$ in G' by breadth-first search, $d_{(v_s^r, v_d^r)} = 0$ if no path is found;
    $\alpha^r = \beta^r \cdot d_{(v_s^r, v_d^r)}$;
  end for
  $r = ExtractMin(R)$;
  for all $v \in V$ **do**
    $b[v] = \infty$;
    $prev(v) = NULL$;
  end for
  $b[v_s^r] = \beta^r$;
  for $(i = 1; i \leq d_{(v_s^r, v_d^r)}; i++)$ **do**
    for all $(u,v) \in E$ **do**
      if $b[v] > \max(b[u], b_{(u,v)})$ and $b_{(u,v)} \geq \beta^r$ **then**
        $b[v] = \max(b[u], b_{(u,v)})$;
        $prev(v) = u$;
      end if
    end for
  end for
  if $b[v_d^r] = \infty$ **then**
    no path can be found to satisfy r;
  else
    Construct the path from v_s^r to v_d^r, and reserve the bandwidth β^r on links along the path for r;
  end if
end while

The pseudo-code for *MBDPA*(G,R) is shown in Algorithm 19. For each fixed-slot reservation r, the algorithm first runs breadth-first search to determine the distance $d_{(v_s^r, v_d^r)}$ from v_s^r to v_d^r in G', which is computed by removing the links with residual bandwidths less than β^r. Here, $d_{(v_s^r, v_d^r)}$ is computed as the number of hops along the found path, and the bandwidth of the path is guaranteed to be at least β^r. Each reservation is then keyed by the

product of its specified bandwidth and computed distance. The reservation with minimal product is extracted for bandwidth scheduling. The narrowest path with bandwidth at least β^r is computed by an extended Bellman-Ford algorithm, which iterates for $d_{(v_s^r, v_d^r)}$ times. The product of each of the remaining tasks is recomputed in each outmost **while** loop because the network G is updated as bandwidth reservation is recorded at the end of each loop. The runtime of MBDPA in worst case is $O(k \cdot |E| \cdot (k + |E|))$.

7.3 Performance Evaluation

We first present the simulation-based performance comparison between Greedy and MBDPA under various network sizes and different numbers of accumulated tasks, and then conduct more simulations to investigate the performance benefits of periodic scheduling in comparison with instant scheduling in both MDTA and MFBR problems.

7.3.1 Comparison of Greedy and MBDPA

We conduct performance evaluation of Greedy and MBDPA heuristic algorithms for MFBR problem using various simulated networks and bandwidth reservation requests. Each simulated network has an arbitrary network topology with a given number of nodes and links. The residual bandwidths of the links are randomly selected within a certain range. A number of fixed-slot reservations are generated, whose source and destination vertices are randomly selected and bandwidths are randomly specified within a certain range.

Table 7.1: Performance comparison between Greedy (G) and MBDPA (M) algorithms.

# of nodes, # of links	10 tasks		20 tasks		50 tasks		100 tasks		200 tasks		300 tasks		400 tasks	
	G	M	G	M	G	M	G	M	G	M	G	M	G	M
10, 40	10	10	19	20	35	46	44	58	57	77	61	84	64	89
20, 80	10	10	20	20	42	45	52	66	64	88	69	105	76	119
50, 200	10	10	20	20	45	50	53	78	76	110	90	134	95	150
100, 400	10	10	20	20	47	50	83	93	97	135	106	177	120	186
200, 800	10	10	20	20	50	50	94	97	122	185	142	227	158	255
500, 2000	10	10	20	20	50	50	99	100	169	199	194	285	197	349

Given the same set of reservations and network configurations, we compare the numbers of satisfied reservations that are achieved by Greedy and MBDPA. Table 7.1 summarizes the performance comparison of Greedy and MBDPA under various network topologies and different numbers of bandwidth reservations. The simulation results show that MBDPA consistently outperforms Greedy in all the cases. Since larger networks have more bandwidth resources and the reservations are randomly distributed over the network, larger network sizes are expected to have more satisfied reservations, which is confirmed by the observation in Table 7.1. For visual comparison purposes, the performance of the two heuristic algorithms under a small network of 10 nodes and 40 links is plotted in Fig. 7.1, where only a small portion of 400 reservations are satisfied. Their performance under a large network of 500 nodes and 2000 links is also plotted in Fig. 7.2 , where MBDPA algorithm satisfies the majority of the reservations while Greedy algorithm satisfies less than half of the reservations.

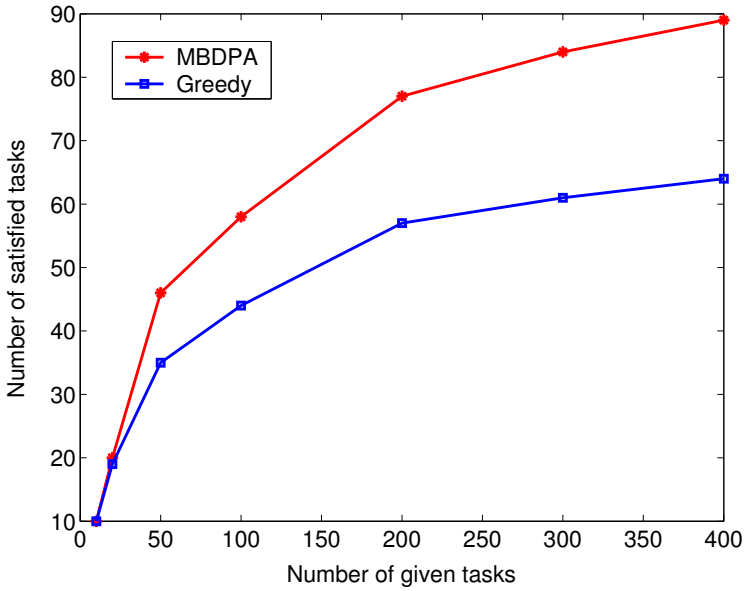

Figure 7.1: Comparison of Greedy and MBDPA under a network with 10 nodes and 40 links.

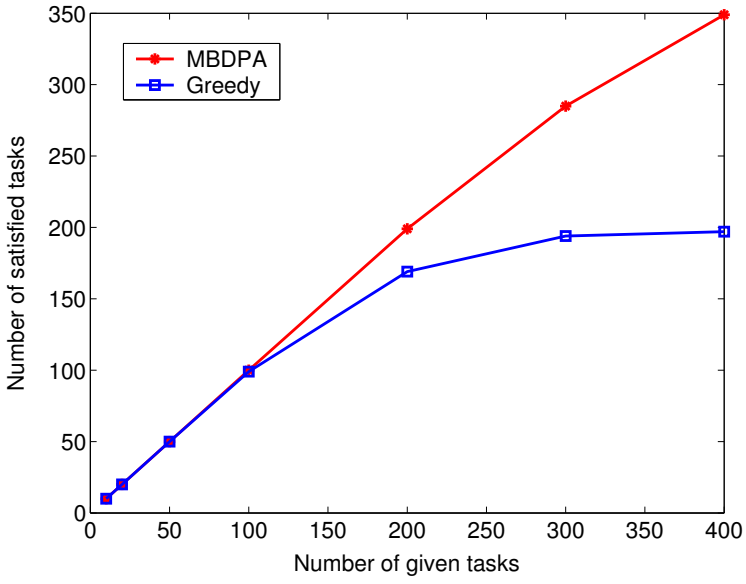

Figure 7.2: Comparison of Greedy and MBDPA under a network with 500 nodes and 2000 links.

7.3.2 Periodic Scheduling vs. Instant Scheduling

We compare the performances of periodic scheduling and instant scheduling in both MDTA and MFBR problems to justify the motivation for developing periodic scheduling algorithms. In instant scheduling, tasks are scheduled at their arrival times in the arriving order; while in periodic scheduling, scheduling algorithms are launched periodically at a certain time interval on a number of tasks accumulated in one period. In fact, instant scheduling is a special case of periodic scheduling when the time interval is set to one time unit. Therefore, the performance of periodic scheduling is at least as good as instant scheduling if an appropriate scheduling interval is applied. Note that to a large degree, the performance of periodic scheduling is determined by scheduling interval, which is confirmed by the simulations described below.

For MDTA problem, based on a number of randomly generated tasks with sizes and arrival times evenly distributed at a given range, instant and periodic scheduling algorithms are run using different values of scheduling interval and their corresponding performance curves are plotted in Fig. 7.3. The figure shows that the periodic scheduling performance curve exhibits an obvious unimodal pattern. Therefore, the best value for scheduling interval, can be empirically determined as the valley point to minimize the total transfer end time. Periodic scheduling reduces to instant scheduling when scheduling interval is set to the smallest time unit. In real network applications, the next optimal scheduling interval is predicted as an exponential average of the previous scheduling intervals. Let t_n be the true value of scheduling interval and τ_n be the predicted optimal at the n-th step. The value for

the next optimal interval τ_{n+1} can be predicted as:

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, \tag{7.3.1}$$

where the parameter α, $0 \leq \alpha \leq 1$, controls the relative weight of the most recent and past history in the prediction.

We conduct simulations to investigate how different properties of the given tasks affect periodic scheduling performance, considering two most import task parameters: the number of given tasks and the variance of data sizes. In the simulations, we vary the number of tasks and the variance of data sizes while fixing other parameters such as the number of paths and their bandwidths, and compute the performance improvement of periodic scheduling using the optimal scheduling interval over instant scheduling in terms of total transfer end time, defined as:

$$T_{improved} = T_{instant} - T_{periodic}. \tag{7.3.2}$$

The simulation results are shown in Fig. 7.4, which illustrates that the total transfer end time of instant scheduling is always greater or equal to that of periodic scheduling. The performance superiority of periodic scheduling becomes more obvious when the values of these two parameters increase, which is due to the fact that at each time interval a larger number of files with smaller sizes are scheduled first, hence reducing the total waiting time.

For MFBR, we investigate how the length of the scheduling time interval affects the performance of periodic scheduling. The simulation results produced by MBDPA algorithm on both periodic and instant scheduling are shown in Fig. 7.5, under the network size of 100 nodes and 400 links, with 300 randomly generated data transfer tasks. The arrival times of these tasks are evenly distributed at time range of $[0, 100]$, and the start

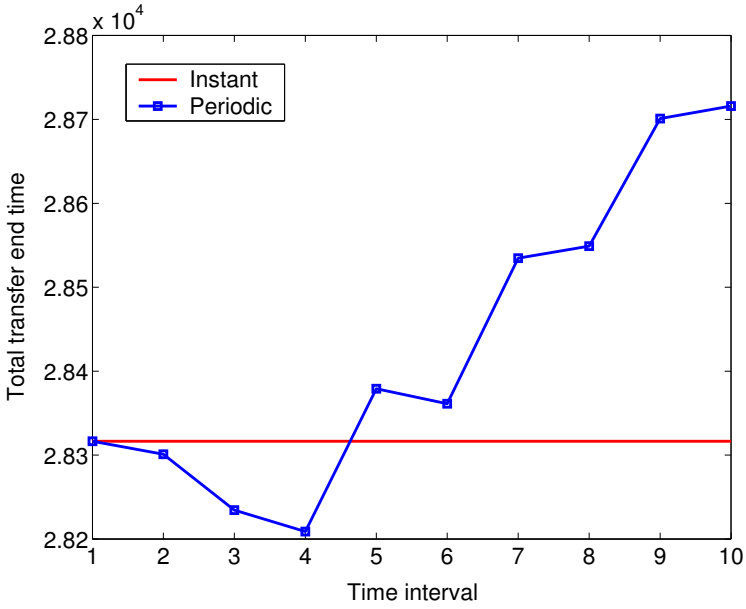

Figure 7.3: Comparison of periodic scheduling and instant scheduling in MDTA problem.

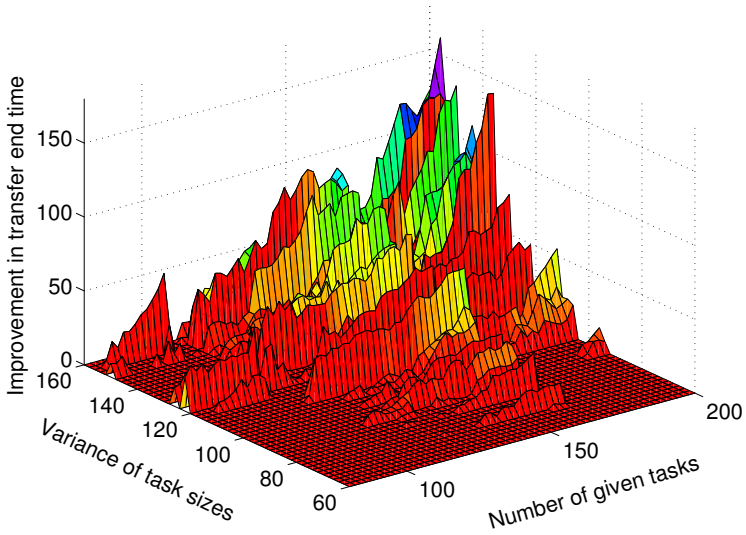

Figure 7.4: Comparison of periodic scheduling and instant scheduling in MDTA problem under different numbers of tasks and different variances of task sizes. Z axis denotes the performance improvement of periodic scheduling over instant scheduling in terms of total transfer end time.

time of these tasks are specified at 100. Fig. 7.5 illustrates that periodic scheduling always achieves a larger number of satisfied tasks compared to instant scheduling, which is not affected by the length of the scheduling time interval. Furthermore, the performance of periodic scheduling improves when the scheduling time interval increases, which is due to the fact that tasks requiring less network resources will be always scheduled first.

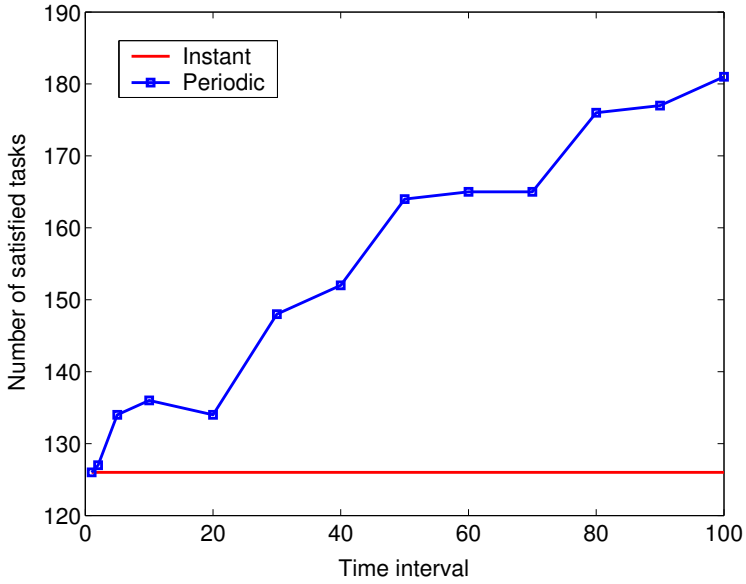

Figure 7.5: Comparison of periodic scheduling and instant scheduling in MFBR problem.

Chapter 8

Conclusion

Within a generalized control plane framework that supports advance bandwidth reservations in high-speed dedicated networks, we constructed realistic network models and studied a comprehensive set of advance bandwidth scheduling problems, including instant scheduling and periodical scheduling, centralized scheduling and distributed scheduling, and scheduling in LCC-overlays.

We conducted an in-depth investigation into these advance bandwidth scheduling problems through rigorous complexity analysis and algorithm design. To the best of our knowledge, we are the first to formulate and study these advance bandwidth scheduling problems. We presented simulation-based performance comparisons between the proposed heuristics with optimal and greedy strategies in a large set of simulated networks.

It is of our future interest to improve the performance of our scheduling algorithms in a more dynamic network environment. For example, once a network condition change is detected, we may re-schedule the reserved bandwidth resource before the start time of the data transfer previously scheduled for an advance bandwidth reservation request.

Bibliography

[1] Terascale Supernova Initiative (TSI). http://www.phy.ornl.gov/tsi.

[2] Terascale High-Fidelity Simulations of Turbulent Combustion with Detailed Chemistry. http://scidac.psc.edu.

[3] National Leadership Computing Facility (NLCF). http://www.ccs.ornl.gov/nlcf.

[4] UCLP: User Controlled LightPath Provisioning. http://www.uclp.ca.

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

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

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

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

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

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

[11] BBCP. http://www.slac.stanford.edu/∼abh/bbcp.

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

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

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

[15] Z. Cao, Z. Wang, and E. Zegura. Performance of hashing-based schemes for internet load balancing. volume 1, pages 332–341, 2000.

[16] C. Cetinkaya and E.W. Knightly. Opportunistic traffic scheduling over multiple network paths. volume 3, pages 1928–1937, Mar. 2004.

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

[18] R. Cohen, N. Fazlollahi, and D. Starobinski. Graded channel reservation with path switching in ultra high capacity networks. In *Proc. of Broadnets*, San Jose, CA, 2006.

[19] R. Cohen, N. Fazlollahi, and D. Starobinski. Path switching and grading algorithms for advance channel reservation architectures. *IEEE/ACM Transactions on Networking*, 17(5):1684–1695, 2009.

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

[21] S. Floyd. HighSpeed TCP for large congestion windows. Internet Draft, Feb. 2003.

[22] S. Ganguly, A. Sen, G. Xue, B. Hao, and B.H. Shen. Optimal routing for fast transfer of bulk data files in time-varying networks. In *Proc. of IEEE Int. Conf. on Communications*, 2004.

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

[24] S. Gorinsky and N.S.V. Rao. Dedicated channels as an optimal network support for effective transfer of massive data. In *INFOCOM 2006 Workshop on High-Speed Networks*, 2006.

[25] W.C. Grimmell and N.S.V. Rao. On source-based route computation for quickest paths under dynamic bandwidth constraints. *Int. J. on Foundations of Computer Science*, 14(3):503–523, 2003.

[26] Y. Gu and R.L. Grossman. SABUL: A transport protocol for grid computing. 1:377–386, 2004. Journal of Grid Computing.

[27] R.A. Guerin and A. Orda. Networks with advance reservations: the routing perspective. In *Proc. of the 19th IEEE INFOCOM*, 2000.

[28] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston. Intra and interdomain circuit provisioning using the oscars reservation system. In *Proc. of the BROADNETS*, pages 1–8, San Jose, CA, Oct. 1-5 2006.

[29] Y. Lin and Q. Wu. On design of bandwidth scheduling algorithms for multiple data transfers in dedicated networks. In *Proc. of the 4th ACM/IEEE Symp. on Arch. for Net. and Comm. Sys.*, pages 151–160, San Jose, CA, USA, Nov. 6-7 2008.

[30] Y. Lin and Q. Wu. Path computation with variable bandwidth for bulk data transfer in high-performance networks. In *Proceedings of INFOCOM HSN Workshop*, Rio de Janeiro, Brazil, April 24 2009.

[31] Y. Lin, Q. Wu, N.S.V. Rao, and M. Zhu. On design of scheduling algorithms for advance bandwidth reservation in dedicated networks. In *The 2008 INFOCOM High-Speed Networks Workshop*, Phoenix, Arizona, Apr. 13 2008.

[32] M.H. Phùng, K.C. Chua, G. Mohan, M. Motani, T.C. Wong, and P.Y. Kong. On ordered scheduling for optical burst switching. *Computer Networks*, 48(6):891–909, 2005.

[33] N. S.V. Rao, J. Gao, and L. O. Chua. *Complex Dynamics in Communication Networks*, chapter On dynamics of transport protocols in wide-area internet connections. 2004.

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

[35] N.S.V. Rao, Q. Wu, S.M. Carter, W.R. Wing, D. Ghosal A. Banerjee, and B. Mukherjee. Control plane for advance bandwidth scheduling in ultra high-speed networks. In *INFOCOM 2006 Workshop on Terabits Networks*, 2006.

[36] S. Sahni, N.S.V. Rao, S. Ranka, Y. Li, E. Jung, and N. Kamath. Bandwidth scheduling and path computation algorithms for connection-oriented networks. In *Proc. of Int. Conf. on Networking*, 2007.

[37] B.H. Shen, B. Hao, and A. Sen. On multipath routing using widest pair of disjoint paths. In *Proc. of Workshop on High Performance Switching and Routing*, pages 134–140, 2004.

[38] L. Shen, X. Yang, A. Todimala, and B. Ramamurthy. A two-phase approach for dynamic lightpath scheduling in wdm optical networks. In *Proc. of IEEE Int. Conf. on Comm.*, pages 2412–2417, Glasgow, Jun. 24-28 2007.

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

[40] M. Veeraraghavan, H. Lee, E.K.P. Chong, and H. Li. A varying-bandwidth list scheduling heuristic for file transfers. In *Proc. of IEEE Int. Conf. on Communications*, 2004.

[41] Q. Wu and Y. Lin. Distributed bandwidth scheduling for advance reservation in high-performance networks. In *Proc. of 7th Int. ICST Conf. on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QShine)*, Houston, Texas, Nov. 17-19 2010.

[42] Q. Wu and N.S.V. Rao. A class of reliable UDP-based transport protocols based on stochastic approximation. In *Proc. of the 24th IEEE INFOCOM*, Miami, Florida, Mar. 13-17 2005.

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

[44] K. Xi, H.J. Chao, and C. Guo. Recovery from shared risk link group failures using ip fast reroute. In *Proc. of 19th Int. Conf. on Comp. Comm. and Net.*, pages 1–7, Zurich, Switzerland, Aug. 2-5 2010.

[45] C. Xie, H. Alazemi, and N. Ghani. Routing and scheduling in distributed advance reservation networks. In *Proc. of IEEE Global Telecommunications Conference*, pages 1–6, Miami, FL, Dec. 6-10 2010.

[46] Y. Xiong, M. Vandenhoute, and H.C. Cankaya. Control architecture in optical burst-switched wdm networks. *IEEE J. on Selected Areas in Communications*, 18(10):1838–1851, Oct. 2000.

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

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

[49] Y. Zhu and B. Li. Overlay network with linear capacity constraints. *IEEE Trans. on Parallel and Distributed Systems*, 19:159–173, Feb. 2008.