Bandwidth Scheduling for Big Data Transfer with Two Variable Node-Disjoint Paths

Aiqin Hou, Chase Qishi Wu, Liudong Zuo, Xiaoyang Zhang, Tao Wang, and Dingyi Fang

Abstract: **Many large-scale applications in broad science, engineering, and business domains are generating big data, which must be transferred to remote sites for various storage and analysis purposes. Bandwidth reservation services that discover feasible routing options over dedicated paths in high-performance networks have proved to be effective for such big data transfer. In this paper, we formulate a generic problem of bandwidth scheduling with two variable node-disjoint paths (BS-2VNDP) by exploring the flexibility and capacity of multiple data transfer paths. We further consider two variable paths of either fixed or variable bandwidth with negligible or non-negligible path switching delay, referred to as 2VPFB-0/1 and 2VPVB-0/1, respectively. We prove that all of these four scheduling problems are NP-complete, and then propose a heuristic algorithm for each. For performance comparison, we also design several other heuristic algorithms based on a greedy strategy. These scheduling algorithms are implemented and tested in both simulated and real-life networks, and extensive results show that the proposed heuristic algorithms significantly outperform other algorithms in comparison.**

Index Terms: **Bandwidth scheduling, high-performance networks, node-disjoint paths, switching delay, variable paths.**

I. INTRODUCTION

MANY large-scale applications in various science, engineering, and business domains require fast and reliable transfer of big data over long distances for remote operations. High-performance networks (HPNs), which feature high link bandwidths and are capable of performing advance bandwidth reservation, have emerged as a promising solution for the unprecedented requirement of big data transfer. Such networks provision dedicated channels with reserved bandwidth using circuit-switching infrastructures as exemplified by UltraScience Net [1] and CHEETAH [2], or IP-based tunneling techniques as exemplified by OSCARS [3] of ESnet and AL2S [4] of Internet2. Typically, the bandwidth scheduler in HPNs is responsible for computing an appropriate network path and allocating link bandwidths to meet a user's data transfer request based on the network topology and bandwidth availability [5], [11], [13], [18].

Considering the sheer volume of data transfer, an increasing number of real-life bandwidth reservation systems adopt multi-path routing instead of single-path routing to improve data transfer throughput. However, multi-path routing also introduces extra overhead to both the control plane and the data plane of a network [19]. Multipath routing could be either link or node disjoint, with varying complexity in different circumstances. Particularly, node-disjoint paths are able to establish multiple completely independent data channels between source and destination, and hence can effectively increase transmission bandwidth and reliability [17]

Many problems on single-path bandwidth scheduling have been proved to be NP-complete [31], which sheds light on the difficulty of multi-path routing. For example, several studies have shown that the problems with multiple constrained paths (MCP) are generally NP-complete [20], [21]. Furthermore, finding disjoint paths with a single constraint is also an NP-hard problem [22]–[26]. The two-path routing problem with reliability consideration is NP-hard in the strong sense, as opposed to the ordinary NP-completeness of the single-path problem [27].

Multipath routing improves throughput in general, but also introduces non-negligible overhead to both the control plane and the data plane of an HPN. Especially for variable paths, it frequently requires path switching between adjacent time slots, and performing an excessive number of path switchings not only degrades transport performance but also increases implementation complexity. Furthermore, if one user request takes up too many paths (resources), it may cause a serious fairness issue to others. Therefore, to find a good tradeoff between high throughput, system overhead, transport robustness, and ease of implementation, we consider two node-disjoint paths in this work.

We formulate a generic problem of bandwidth scheduling with two variable node-disjoint paths (BS-2VNDP) to support big data transfer in HPNs. In BS-2VNDP, we consider two cases with two variable paths of either fixed or variable bandwidth, referred to as 2VPFB and 2VPVB, or 2VPFB/VB for brevity. Note that using variable paths during a data transfer session requires some support from the network infrastructure to perform path switching, which may incur a certain switching delay τ. Therefore, we further divide 2VPFB/VB into two cases where the path switching delay is negligible (i.e., $\tau = 0$) and non-negligible (i.e., $\tau \neq 0$), referred to as 2VPFB/VB-0 and 2VPFB/VB-1, respectively. We prove that all of these four problems with different combinations of bandwidth variability and switching delay negligibility are NP-complete, and design a heuristic approach for each. For performance comparison, we

Manuscript received July 17, 2018; Revised April 20, 2019; approved for publication by Vangelis Angelakis, Division III, February 17, 2020.

This research is sponsored by National Nature Science Foundation of China under Grant No. U1609202, Key Research and Development Plan of Shaanxi Province, China under Grant No. 2018GY-011, and Xián Science and Technology Plan under Grant No. GXYD18.2 with Northwest University, China.

A. Hou, X. Zhang, T. Wang, and D. Fang are with the School of Information Science and Technology, Northwest University, China, email: {houaiqin, dyf}@nwu.edu.cn,{shinezxy, wangt}@stumail.nwu.edu.cn.

C. Q. Wu is with New Jersey Institute of Technology, email: chase.wu@njit.edu.

L. Zuo is with California State University, Dominguez Hills, email: lzuo@csudh.edu.

C. Q. Wu is the corresponding author.

Digital Object Identifier: 10.1109/JCN.2020.000004

1229-2370/19/$10.00 © 2020 KICS

also design several other heuristic algorithms based on a greedy strategy. We implement and test these bandwidth scheduling algorithms in both simulated and real-life networks, and extensive results show that the proposed Imp2VPFB-0, Imp2VPFB-1, and Imp2VPVB-1 algorithms achieve about 10–20%, 12–35%, and 5% performance improvement on average over Greedy2VPFB-0, Greedy2VPFB-1, and Greedy2VPVB-1 in comparison, respectively.

The rest of this paper is organized as follows. We provide a survey of related work on multi-path bandwidth scheduling in Section II. We formulate the BS-2VNDP problem variants and conduct complexity analysis in Section III. In Section IV, we present the algorithm design with detailed explanations. Extensive simulations are conducted and described in Section V. We conclude our work in Section VI.

II. RELATED WORK

Different problems regarding multiple disjoint paths have been extensively studied for decades in various contexts. We provide below a survey of such efforts that are closely related to our work.

Several researchers studied the problem of finding maximum combined bandwidth in node-disjoint paths. In [25], Dahshan proved that such problems with node-disjoint paths in communication networks are NP-complete, and proposed a solution using a maximum-cost variant of Dijkstra's algorithm and a virtual-node representation to obtain maximum-bandwidth node-disjoint paths. In [24], Shen *et al.* discussed the problem of finding a pair of edge- or node-disjoint paths with maximum combined bandwidth, including widest pair of disjoint paths coupled (WPDPC) and widest pair of disjoint paths decoupled (WPDPD). They proved that both versions of the problem are NP-complete, and provided exact solutions using ILP and two approximate solutions. In [30], a link-disjoint or node-disjoint multi-path routing strategy is developed using two colored trees, red and blue, rooted at a designated node called the drain. The paths from a given source to the drain on these two trees are link-disjoint or node-disjoint. This approach requires every node to maintain only two preferred neighbors for each destination, one on each tree. In [29], a distributed distance-vector algorithm is used to find multiple node-disjoint paths including the shortest path in a computer network.

Most of the aforementioned work considers static networks for path computing, and the routes computed in such static networks correspond to those computed in a single time slot in dynamic networks with time-varying link bandwidths across multiple time slots. There also exist several efforts on bandwidth scheduling in dynamic HPN networks, [31], [32], [36], [38]. In [35], Zuo *et al.* investigated the problem of scheduling as many concurrent bandwidth reservation requests as possible over different paths in an HPN while achieving the average earliest completion time (ECT) and the average shortest duration (SD) of scheduled BRRs. These two problems were proved to be NP-complete, and heuristic algorithms were proposed. In [37], Zuo *et al.* considered two generic types of bandwidth reservation requests concerning data transfer reliability: (i) To achieve the highest data transfer reliability under a given data transfer deadline, and (ii) to achieve the earliest data transfer completion time while satisfying a given data transfer reliability requirement. Optimal scheduling algorithms with optimality proofs are proposed.

In [31], [32], Lin and Wu investigated single-path bandwidth scheduling with an exhaustive combination of different path and bandwidth constraints: 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). These four problems have the same objective to minimize the data transfer end time for a given transfer request with a pre-specified data size. To support big data transfer, our work extends single-path routing to multi-path routing to minimize the data transfer end time and achieve a higher quality of service (QoS). Note that multi-path routing could be leveraged from software-defined networking (SDN) technologies to realize complex network operations and control [12], [18]. For example, in [12], Aktas *et al.* used SDN to provide data transport service control and resource provisioning to meet different QoS requirements from multiple coupled workflows sharing the same service medium. They presented a flexible control and a disciplined resource scheduling approach for data transfer. In [16], Hou *et al.* studied bandwidth scheduling with two fixed node-disjoint paths for concurrent data transfer. In this work, we consider variable paths to achieve a higher resource utilization than fixed paths.

III. PROBLEM FORMULATION

In this section, we firstly show the scheduling network model, followed by the problem definition and complexity analysis.

A. Network Model

The topology of an HPN can be represented as a graph $G(V,E)$ with $|V|$ nodes and $|E|$ links, where each link $l \in E$ maintains a list of residual bandwidths specified as a segmented constant function of time. We use a 3-tuple of time-bandwidth (TB) $(t_l[i], t_l[i+1], b_l[i])$ to denote the residual bandwidth $b_l[i]$ of link l during the i-th time-slot (i.e., the time interval $[t_l[i], t_l[i+1]]$), $i = 0, 1, 2, \cdots, T_l - 1$, where T_l is the total number of time-slots on link l.

Before path computing, we combine the TB lists of all links to build an aggregated TB (ATB) list, where we store the residual bandwidths of all links in each intersected time-slot. As shown in Fig. 1, we create a set of new time slots by combining the time slots of all links, and then map the residual bandwidths of each link to the ATB list in each new time slot. We denote the ATB list as $(t[0], t[1], b_0[0], b_1[0], \cdots, b_{|E|-1}[0]), \cdots, (t[T-1], t[T], b_0[T-1], b_1[T-1], \cdots, b_{|E|-1}[T-1])$, where T is the total number of new time-slots after the aggregation of TB lists of $|E|$ links. Without loss of generality, we set the smallest time-slot to be 1 time unit.

B. Problem Definition

We define a generic problem of BS-2VNDP as follows [5].

Definition 1: BS-2VNDP: Given a graph $G(V,E)$ of an HPN with an ATB list for all links, and a user request that specifies source v_s, destination v_d, and data size δ, we wish to find

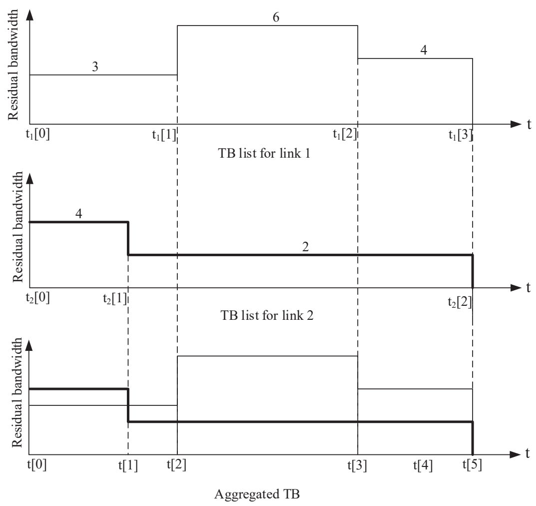


Fig. 1. An aggregated TB list by combining two individual TB lists.

two variable node-disjoint paths for the transfer of data size δ from v_s to v_d such that the data transfer end time is minimized, or equivalently, sum of the bandwidths of these two paths is maximized.

In BS-2VNDP, we consider two cases based on the bandwidth variability of each path, as defined below [5].

Definition 2: 2VPFB: Given the above network model and user request, the goal is to find two variable node-disjoint paths from v_s to v_d, each of which has a fixed bandwidth across different time-slots, such that the data transfer end time is minimized.

Definition 3: 2VPVB: Given the above network model and user request, the goal is to find two variable node-disjoint paths from v_s to v_d, each of which could have variable bandwidths across different time-slots, such that the data transfer end time is minimized.

In 2VPFB/2VPVB, considering the path switching delay τ, we further consider two different types of service models: i) The path switching delay is negligible (i.e., $\tau = 0$), referred to as 2VPFB-0/2VPVB-0, and ii) the path switching delay is not negligible (i.e., $\tau > 0$), referred to as 2VPFB-1/2VPVB-1. Note that path switching may happen between two adjacent time slots, either at the end of one time-slot or at the beginning of its succeeding time-slot. Generally, it should be performed at the time slot with a lower bandwidth because the data transfer process is suspended during the period of path switching.

For illustration, we provide an example network in Fig. 2, which has seven nodes, a pair of which are designated as source and destination, and eleven links, each of which has residual bandwidths across two time-slots starting from time point 0 as labeled on the link. The other parameters include data size $\delta = 19$ units and path switching delay $\tau = 0.1$ unit of time.

In 2VPFB-0, the optimal solution is shown in Fig. 3(a). In the first time-slot, we find two node-disjoint paths: $p_1[0] : v_s - v_2 - v_3 - v_d$ (with maximum bandwidth of 6 in time-slot 0), and $p_2[0] : v_s - v_1 - v_4 - v_d$ (with bandwidth of 4, which is also the maximum in the current network except

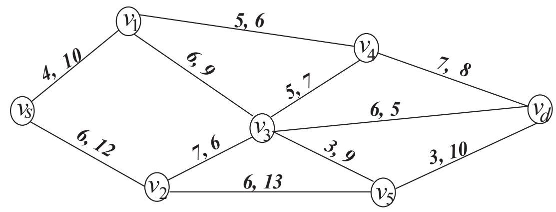


Fig. 2. An example HPN with link bandwidths.

the nodes in use). In the second time-slot, we also find two node-disjoint paths with the maximum bandwidth in the current network: $p_1[1] : v_s - v_2 - v_5 - v_d$ (with maximum bandwidth of 10), and $p_2[1] : v_s - v_1 - v_3 - v_4 - v_d$ (with bandwidth of 7). Obviously, the transfer of data $\delta = 19$ cannot be completed in time-slot 0, so data transfer will continue to time-slot 1. Since we consider fixed bandwidth (i.e., the bandwidth remains constant during the entire data transfer period), the available bandwidth of $p_1[1]$ and $p_2[1]$ is the same as that of $p_1[0]$ and $p_2[0]$, respectively. Therefore, sum of bandwidths in both time-slots is the same as $\beta = 6 + 4 = 10$. For data size $\delta = 19$, the total data transfer end time is $19/10 = 1.9$.

In 2VPFB-1, the optimal solution is shown in Fig. 3(b), where the two node-disjoint paths and the sum of their respective bandwidths are the same as in 2VPFB-0. Since both of the time-slots have the same fixed bandwidth on each path, the path switching could be performed either at the end of the first time-slot (i.e., time interval [0.9, 1]), or at the beginning of the second time-slot (i.e., time interval [1, 1.1]). For data size $\delta = 19$, the data transfer end time is calculated as $9/10 + 0.1 + 10/10 = 2$.

In 2VPVB-0, the optimal solution is shown in Fig. 3(c). In the first time-slot, we find two node-disjoint paths: $p_1[0]$: $v_s - v_2 - v_3 - v_d$ (with maximum bandwidth of 6), and $p_2[0]$: $v_s - v_1 - v_4 - v_d$ (with bandwidth of 4, which is also the maximum in the current network except the nodes in use). The sum of bandwidths in the first time-slot is $\beta[0] = 6 + 4 = 10$. In the second time-slot, we also find two node-disjoint paths: $p_1[1] : v_s - v_2 - v_5 - v_d$ (with maximum bandwidth of 10), and $p_2[1] : v_s - v_1 - v_3 - v_4 - v_d$ (with bandwidth of 7, which is also the maximum in the current network except the nodes in use). Sum of the bandwidths in the second time-slot is $\beta[1] = 10 + 7 = 17$. For data size $\delta = 19$, the data transfer end time is $10/10 + 9/17 = 1.53$.

In 2VPVB-1, the optimal solution is shown in Fig. 3(d), where the two node-disjoint paths and sum of their respective bandwidths are the same as in 2VPVB-0. Since sum of path bandwidths in the first time-slot is smaller than that in the second time-slot, the path switching of both paths should be performed at the end of the first time-slot (i.e., time interval [0.9, 1]). For data size $\delta = 19$, the data transfer end time is $9/10 + 0.1 + 10/17 = 1.59$.

C. Problem Complexity Analysis

The 2VPFB/VB-0 and 2VPFB/VB-1 problems as formulated are more general than their counterparts in static networks with constant link bandwidths [22]–[25], [27].

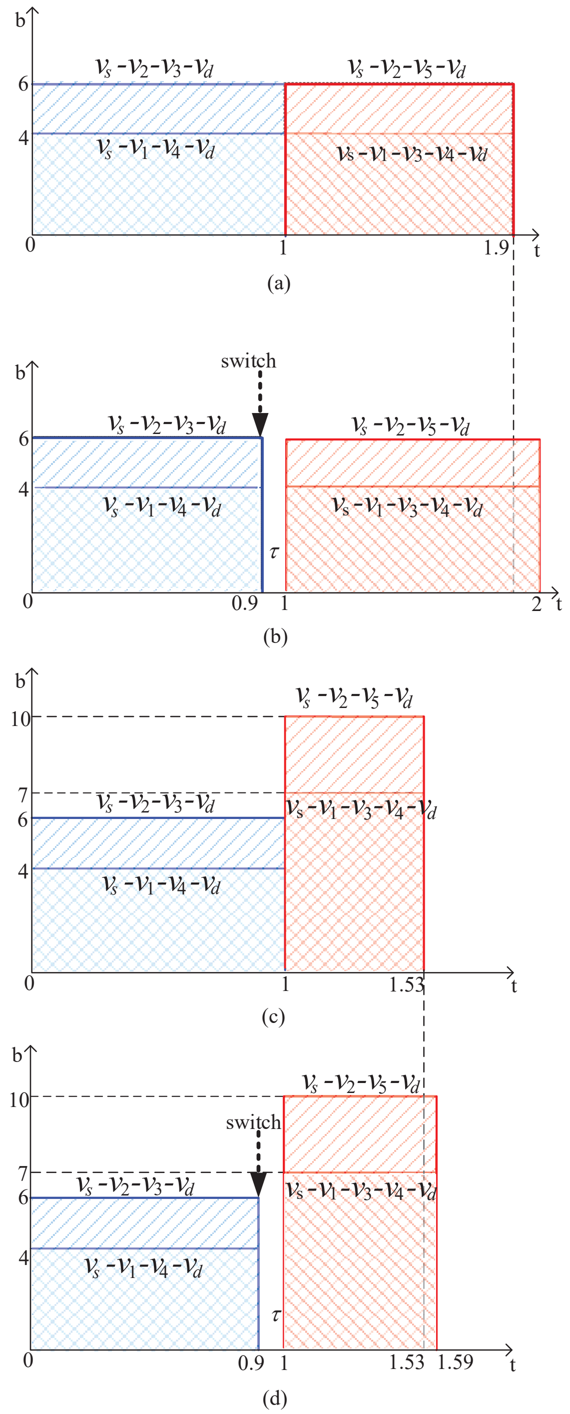


Fig. 3. Illustration of 2VPFB-0/1 and 2VPVB-0/1: (a) 2VPFB-0, (b) 2VPFB-1, (c) 2VPVB-0, and (d) 2VPVB-1.

To analyze the computational complexity of these problems, we first introduce the WPDPC problem, as defined and proved to be NP-complete in [24].

Definition 4: WPDPC: given a network with a fixed bandwidth for each link, does there exist two disjoint paths P_1 and P_2 from v_s to v_d, such that the sum of the bandwidths of these two paths is greater than or equal to X?

WPDPC is to compute two edge-/node-disjoint paths with the largest sum of bandwidths in a single time slot, which is a special case of our problems using two disjoint variable paths across multiple time slots, as detailed below.

C.1 Complexity of 2VPFB-0

We have the following theorem for the complexity of 2VPFB-0.

Theorem 1: 2VPFB-0 is NP-complete.

Proof: We first show that 2VPFB-0 $\in NP$. The decision version of 2VPFB-0 is as follows: Given the network model and user request, are there two variable node-disjoint paths from v_s to v_d, each of which has a fixed bandwidth across different time-slots, such that the data transfer end time is no larger than a given bound T without considering the path switching delay? Given two variable node-disjoint paths from v_s to v_d, we can identify the available fixed bandwidth of these two paths and further check whether the data transfer end time is no larger than T. Obviously, the above process can be done in polynomial time, so 2VPFB-0 $\in NP$.

The NP-hardness of 2VPFB-0 could be established through proof-by-restriction. We consider a special case of 2VPFB-0 where the bandwidth of each link keeps the same across all time-slots. Obviously, any instance of WPDPC could be mapped to the above special case of 2VPFB-0, and such mapping could be done in polynomial time. If we could find two paths satisfying WPDPC, these two paths could also satisfy 2VPFB-0 ($t = \delta/X$), and vice versa ($X = \delta/t$). Therefore, 2VPFB-0 is at least as hard as WPDPC.

Since a special case of 2VPFB-0 is NP-complete, so is the general 2VPFB-0 problem with time-varying link bandwidths. Proof ends.

□

C.2 Complexity of 2VPFB-1

We have the following lemma for the complexity of 2VPFP-1.

Lemma 1: 2VPFB-1 is NP-complete.

Proof: We prove the NP-completeness of 2VPFB-1 by showing that 2VPFB-0 is a special case of 2VPFB-1. This is straightforward as we can restrict 2VPFB-1 to 2VPFB-0 by only considering those problem instances where the path switching delay is negligible (i.e., $\tau = 0$). Since 2VPFB-0 is NP-complete, so is 2VPFB-1.

□

C.3 Complexity of 2VPVB-0

We have the following lemma for the complexity of 2VPVB-0.

Lemma 2: 2VPVB-0 is NP-complete.

Proof: Similar to the proof of Theorem 1, we can prove the NP-completeness of 2VPVB-0 by showing that the WPDPC problem [24] is a special case of 2VPVB-0. We restrict 2VPVB-0 to WPDPC by only considering those problem instances where the bandwidth of each link remains constant across all time-slots. In other words, there is no need to switch paths between any adjacent time-slots. Hence, WPDPC is a special case of

2VPVB-0 when the network is static with constant link bandwidths. Since WPDPC is NP-complete [24], so is 2VPVB-0. □

C.4 Complexity of 2VPVB-1

We have the following lemma for the complexity of 2VPVB-1.

Lemma 3: 2VPVB-1 is NP-complete.

Proof: The NP-completeness of VPVB-1 has been established in [31] by showing that FPVB, which is NP-complete, is a special case of VPVB-1. Obviously, 2VPVB-1 is a more general version of VPVB-1 that computes multiple concurrent VPVB-1 paths. □

IV. DESIGN OF SCHEDULING ALGORITHMS

The NP-completeness of 2VPFB/VB-0/1 indicates that there does not exist any polynomial-time optimal algorithm unless $P = NP$. Therefore, we focus on the design of heuristic algorithms for 2VPFB/VB-0/1.

Since no existing algorithm can be used directly to solve these four scheduling problems, we design a heuristic algorithm for each in Sections IV.A, IV.B, IV.C, and IV.D, respectively. Meanwhile, we design heuristic algorithms based on a greedy strategy for performance comparison.

A. Heuristic Scheduling Algorithms for 2VPFB-0

In 2VPFB-0, we compute two node-disjoint paths in time-slot $[0, i]$, $i \leq T-1$, each of which is allowed to use different routes with zero path switching delay across different time-slots, while the bandwidth must be fixed.

A.1 Greedy Algorithm for 2VPFB-0

We design a polynomial-time greedy algorithm, Greedy2VPFB-0, whose pseudocode is provided in Algorithm 1. The algorithm computes two variable node-disjoint path sets (i.e., p_1 and p_2) with fixed bandwidth, on which data of size δ has the earliest transfer end time (i.e., maximum sum of fixed bandwidths). Each path set is composed of paths $p_1[i]$ and $p_2[i]$ from v_s to v_d with maximum bandwidth $\beta_1[i]$ and $\beta_2[i]$ using Dijkstra's algorithm at different time-slots. The bandwidths of p_1 and p_2 are $\beta_1 = \min(\beta_1[0], \cdots, \beta_1[i])$ and $\beta_2 = \min(\beta_2[0], \cdots, \beta_2[i])$, respectively. The algorithm checks if the data of size δ can be concurrently transferred by the path-pair during the time-slot range $[0, i]$ (i.e., time interval $[t[0], t[i+1]]$). If paths in these two path sets could not finish the data transfer, we move onto the next time slot; otherwise, we compute the transfer end time t_{end}. At the end, t_{end} is returned. If $t_{\text{end}} = \infty$, it means that we could not finish the data transfer of size δ within T time slots. Note that the data size to be transferred by each path set is proportional to its bandwidth during the data transfer period.

Since the time complexity of Dijkstra's algorithm is $O(|V|^2)$, the time complexity of Greedy2VPFB-0 is $O(T \cdot |V|^2 + T)$ in the worst case, where T is the total number of new time slots in the ATB list.

A.2 Improved Algorithm for 2VPFB-0

In Greedy2VPFB-0, for a given data size δ, the bandwidths of path sets p_1 and p_2 are determined by the bandwidths of the bottleneck paths, namely the paths with the least available bandwidths among all paths in p_1 and p_2, respectively. Although the bandwidth of each path is maximized in each time-slot, there is no guarantee that the concurrent transfer end time by the path-pair for data size δ is minimized from a global perspective. Since the bandwidth of each path during transfer period $[0, i]$ is fixed, it may not be always optimal to start data transfer immediately, for example, when the path bandwidths in the preceding time-slots are much smaller than those in the succeeding time-slots during the transfer period. In this case, we may start the data transfer at the beginning of some time-slot after time-slot 0 to improve Greedy2VPFB-0, referred to as Imp2VPFB-0.

Algorithm 1 Greedy2VPFB-0

Input: an HPN graph $G(V, E)$ with an ATB list, source v_s, destination v_d, and data size δ

Output: the earliest transfer end time t_{end}

$t_{\text{end}} = \infty$, $\beta_1 = \infty$, and $\beta_2 = \infty$;
for $0 \leq i \leq T-1$ **do**
  $\beta_1[i]$ = bandwidth of the widest path $p_1[i]$ from v_s to v_d in time slot i;
  Remove the nodes and links of p_1 from G to create a new graph G';
  $\beta_2[i]$ = bandwidth of the widest path $p_2[i]$ from v_s to v_d in time slot i;
  $\beta_1 = \min(\beta_1[i], \beta_1)$;
  $\beta_2 = \min(\beta_2[i], \beta_2)$;
  $\beta = \beta_1 + \beta_2$;
  if $\beta \cdot (t[i+1] - t[0]) \geq \delta$ **then**
    $t_{\text{end}} = t[0] + \delta/\beta$;
  if $t_{\text{end}} < \infty$ **then**
    Break;
return t_{end}.

The pseudocode of Imp2VPFB-0 is provided in Algorithm 2. For a given time-slot i, Imp2VPFB-0 computes the widest path-pair from v_s to v_d in the time-slot i, and then repeatedly checks if the given data δ can be transferred during time-slot $[j, i]$, $i \geq j \geq 0$. If there exists certain j such that the data of size δ can be transferred during time-slot $[j, i]$ (i.e., time interval $[t[j], t[i+1]]$), the data transfer start time is $t[j]$ and the data transfer end time is computed as shown in Line 12 of Algorithm 2; otherwise, Imp2VPFB-0 increases i by 1. The earliest data transfer end time t_{end} is returned at the end.

The same as in Greedy2VPFB-0, the data size to be transferred by each path set in Imp2VPFB-0 is also proportional to its bandwidth during the data transfer period. Since the time complexity of Dijkstra's algorithm is $O(|V|^2)$, the complexity of Imp2VPFB-0 is $O(T \cdot |V|^2 + T^2)$ in the worst case.

B. Heuristic Scheduling Algorithms for 2VPFB-1

For 2VPFB-1, the path-switching delay is not negligible. Since each path set (such as p_1 and p_2) has a fixed bandwidth, a path switching with a delay $\tau > 0$ could occur either at the end of one time-slot or at the beginning of its succeeding time-slot. 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

Algorithm 2 Imp2VPFB-0

Input: an HPN graph $G(V, E)$ with an ATB list, source v_s, destination v_d, and data size δ
Output: the earliest transfer end time t_{end}

$t_{\text{end}} = \infty$;
for $0 \le i \le T-1$ **do**
  $\beta_1[i]$ = bandwidth of the widest path $p_1[i]$ from v_s to v_d in time slot i;
  Remove the nodes and links of p_1 from G to create a new graph G';
  $\beta_2[i]$ = bandwidth of the widest path $p_2[i]$ from v_s to v_d in time slot i;
  $\beta_1 = \infty$ and $\beta_2 = \infty$;
  for $i \ge j \ge 0$ **do**
    $\beta_1 = \min(\beta_1[j], \beta_1)$;
    $\beta_2 = \min(\beta_2[j], \beta_2)$;
    $\beta[j] = \beta_1 + \beta_2$;
    if $\beta[j] \cdot (t[i+1] - t[j]) \ge \delta$ and $(t[j] + \delta/\beta[j]) < t_{\text{end}}$ **then**
      $t_{\text{end}} = t[j] + \delta/\beta[j]$;
  if $t_{\text{end}} < \infty$ **then**
    Break;
return t_{end}.

where τ is sufficiently large, any path switching would cause a negative impact on the performance, and therefore 2VPFB-1 reduces to 2FPFB. In this paper, we assume that τ is a constant and smaller than the length of any time-slot on the ATB list.

B.1 Greedy Algorithm for 2VPFB-1

We design a polynomial-time greedy algorithm, Greedy2VPFB-1, whose pseudocodes is provided in Algorithm 3. Firstly, it computes the node-disjoint path pair with the maximum bandwidth, i.e., $p_1[i]$ and $p_2[i]$, in every time-slot, $i = 0, 1, \cdots, T-1$. During time slot $[0, i]$, the bandwidths of p_1 and p_2 are $\beta_1 = \min(\beta_1[0], \cdots, \beta_1[i])$ and $\beta_2 = \min(\beta_2[0], \cdots, \beta_2[i])$. Each path may perform a path switching at every adjacent time-slot to use the path pair with the maximum bandwidths. There are i path switchings during the data transfer, and the total switching delay is $\tau \cdot i$. The maximum amount of transferred data by path pair p_1 and p_2 concurrently during the time-slot range $[0, i]$ with i path switchings is $(\beta_1 + \beta_2) \cdot (t[i+1] - t[0] - \tau \cdot i)$. If it is greater than or equal to the data size δ, then the data transfer is completed.

Since the time complexity of Dijkstra's algorithm is $O(|V|^2)$, the complexity of Greedy2VPFB-0 is $O(T \cdot |V|^2)$ in the worst case.

B.2 Improved Algorithm for 2VPFB-1

Greedy2VPFB-1 starts data transfer immediately at time point $t[0]$ and performs path switching in every adjacent time-slot, neither of which may not be always optimal. The improved algorithm for 2VPFB-1, referred to as Imp2VPFB-1, varies the transfer start time and, meanwhile, reduces the number of path switchings between different time-slots to improve the performance. The pseudocode of Imp2VPFB-1 is provided in Algorithm 4.

Algorithm 3 Greedy2VPFB-1

Input: an HPN graph $G(V, E)$ with an ATB list, source v_s, destination v_d, data size δ, and path switching delay τ
Output: the earliest transfer end time t_{end}

$t_{\text{end}} = \infty$, $\beta_1 = \infty$, and $\beta_2 = \infty$;
for $0 \le i \le T-1$ **do**
  Use Dijkstra's algorithm to compute path $p_1[i]$ with the widest bandwidth $\beta_1[i]$ from v_s to v_d in G;
  Remove the nodes and links of $p_1[i]$ from G to create a new graph G';
  Use Dijkstra's algorithm to compute path $p_2[i]$ with the widest bandwidth $\beta_2[i]$ from v_s to v_d in G';
  $\beta_1 = \min(\beta_1[i], \beta_1)$;
  $\beta_2 = \min(\beta_2[i], \beta_2)$;
  $\beta[i] = \beta_1 + \beta_2$;
  if $\beta[i] \cdot (t[i+1] - t[0] - \tau \cdot i) \ge \delta$ **then**
    $t_{\text{end}} = t[0] + \delta/\beta[i] + \tau \cdot i$;
  if $t_{\text{end}} < \infty$ **then**
    Break;
return t_{end}.

In Line 3, it initializes $t_{\text{end}} = \infty$ (transfer end time for δ), $k_1 = 0$ (the path-switching time of p_1) and $k_2 = 0$ (the path-switching time of p_2).

In Lines 6–12, during time slot $[p, q]$, under the condition of fixed bandwidth, it optimizes the first path p_1 so that the number of path-switchings is minimized. For a certain transfer start time-slot p and a certain transfer end time-slot j, we can optimize path p_1 such that it has the minimum number of path switchings during time slot $[p, q]$. In Line 6, we firstly compute the maximum available fixed bandwidth $\beta_1[p, q]$ of p_1 during the time slot range $[p, q]$ with $q - p$ times of path switchings in the worst case, which can be used as a reference value to decide if we should perform a path switching between any two adjacent time-slots. We use $p_1[j]$ to denote path p_1 with the maximum bandwidth in time-slot j, and its bandwidth in the next time-slot $j+1$ is denoted as $\beta_1'[j+1]$. If $\beta_1'[j+1] == \beta_1[j+1]$ or $\beta_1'[j+1] \ge \beta_1[p, q]$ then there is no need to perform path switching between time slot j and $j+1$ (i.e., we keep the $p_1[j]$ in the time-slot $j+1$ and let $p_1[j+1] = p_1[j]$). Otherwise, it switches path $p_1[j]$ to path $p_1[j+1]$ with the widest bandwidth in time-slot $j+1$, and increases the number of switchings k_1 on path p_1 by 1.

In Lines 13–16, in each time-slot j, after deleting $p_1[j]$, it computes a node-disjoint path set $p_2[j]$, which forms p_2 with the maximum bandwidth. Then, we optimize path p_2 by reducing the number of path-switchings k_2 during time slot $[p, q]$ using the method adopted for p_1 previously.

After that, the path-switching counts k_1 and k_2 are minimized for paths p_1 and p_2 during transfer period $[p, q]$. Note that generally, the number of path-switchings and the path-switching time points of two path sets p_1 and p_2 are different.

In Lines 17–22, if the maximum amount of concurrently transferred data $(\beta_1[p, q] \cdot ((t[q+1] - t[p]) - \tau \cdot k_1) + \beta_2[p, q] \cdot ((t[q+1] - t[p]) - \tau \cdot k_2)) \ge \delta$, we can obtain the data transfer end time t'_{end}. If $t'_{\text{end}} < t_{\text{end}}$, it updates t_{end} with the smaller t'_{end}. After iterating through all time slots, we have the earliest

Algorithm 4 Imp2VPFB-1

Input: an HPN graph $G(V, E)$ with an ATB list, source v_s, destination v_d, data size δ, and path switching delay τ

Output: the earliest transfer end time t_{end}

1: **for** $0 \le i \le T-1$ **do**
2: Compute path $p_1[i]$ with the widest bandwidth $\beta_1[i]$ from v_s to v_d in time slot i in G;
3: $t_{\text{end}} = \infty, k_1 = 0, k_2 = 0$;
4: **for** $0 \le q \le T-1$ **do**
5: **for** $0 \le p \le q$ **do**
6: Compute the fixed bandwidth $\beta_1[p, q]$ of p_1 among time-slots $[p, q]$;
7: **for** $p \le j \le q-1$ **do**
8: Compute the bandwidth of $p_1[j]$ in next time-slot $j+1$, denoted as $\beta_1'[j+1]$;
9: **if** $\beta_1'[j+1] == \beta_1[j+1] \,||\, \beta_1'[j+1] \ge \beta_1[p, q]$; **then**
10: $p_1[j+1] = p_1[j]$;
11: **else**
12: $k_1 = k_1 + 1$;
13: **for** $p \le j \le q$ **do**
14: Remove the nodes and links of $p_1[j]$ from G to create a new G' in current time-slot j;
15: Use Dijkstra's algorithm to compute the path with the widest bandwidth from v_s to v_d in G' in current time slot j, denote the returned path as $p_2[j]$ and its bandwidth as $\beta_2[j]$;
16: Repeat line 6–12 with the subscript 1 replaced by 2, and return $\beta_2[p, q]$ and k_2;
17: **if** $(\beta_1[p, q] \cdot ((t[q+1] - t[p]) - \tau \cdot k_1) + \beta_2[p, q] \cdot ((t[q+1] - t[p]) - \tau \cdot k_2)) \ge \delta$ **then**
18: $\delta' = \delta - (\beta_1[p, q] \cdot ((t[q] - t[p]) - \tau \cdot k_1) + \beta_2[p, q] \cdot ((t[q] - t[p]) - \tau \cdot k_2))$;
19: $t'_{\text{end}} = t[q] + \delta' / (\beta_1[p, q] + \beta_2[p, q])$;
20: **if** $t'_{\text{end}} < t_{\text{end}}$ **then**
21: $t_{\text{end}} = t'_{\text{end}}$;
22: **return** t_{end}.

transfer end time t_{end}.

Since the time complexity of Dijkstra's algorithm is $O(|V|^2)$, the complexity of Imp2VPFB-1 is $O(T^3 \cdot |V|^2)$ in the worst case.

C. Heuristic Scheduling Algorithm for 2VPVB-0

For 2VPVB-0, the path-switching delay is negligible, so we can perform path switching at any adjacent time slots without affecting the transfer time. We design a greedy approach, Greedy2VPVB-0, whose pseudocode is provided in Algorithm 5. It computes two node-disjoint paths p_1 and p_2 with the shortest transfer time (i.e., the maximum sum of variable bandwidths) for the transfer of data size δ. Each path is composed of a set of paths from source v_s to destination v_d with the maximum bandwidth in respective time-slots. In each time-slot i, it computes two node-disjoint paths $p_1[i]$ and $p_2[i]$ from v_s to v_d, with the maximum bandwidth $\beta_1[i]$ and $\beta_2[i]$ using Dijkstra's algorithm, and checks if the data of size δ can be concurrently transferred during the time-slot range $[0, i]$ (i.e., time interval $[t[0], t[i+1]]$). If it can finish the data transfer, it computes the transfer end time t_{end} by path-pair (p_1 and p_2); otherwise, it repeatedly increases i by 1. If the data can be completely transferred during time-slots $[0, i]$, the maximum number of path switchings needed is i in the worst case, which does not affect the transfer time. The data size to be transferred over each path is also proportional to its bandwidth during the data transfer period.

Algorithm 5 Greedy2VPVB-0

Input: an HPN graph $G(V, E)$ with an ATB list, source v_s, destination v_d, and data size δ

Output: the earliest transfer end time t_{end}

for $0 \le i \le (T-1)$ **do**
$\beta_1[i] =$ bandwidth of the widest path $p_1[i]$ from v_s to v_d in time slot i in G;
Remove the nodes and links of $p_1[i]$ from G to create a new graph G';
$\beta_2[i] =$ bandwidth of the widest path $p_2[i]$ from v_s to v_d in time slot i in G';
$\beta[i] = \beta_1[i] + \beta_2[i]$;
if $\delta \le \beta[i] \cdot (t[i+1] - t[i])$ **then**
$t_{\text{end}} = t[i] + \delta / \beta[i]$;
Break;
else
$\delta = \delta - \beta[i] \cdot (t[i+1] - t[i])$;
$i = i + 1$;
return t_{end}.

Since the time complexity of Dijkstra's algorithm is $O(|V|^2)$, the time complexity of Greedy2VPVB-0 is $O(T \cdot |V|^2)$ in the worst case.

We would like to point out that an improved algorithm for 2VPVB-0 is not designed. For 2VPVB-0, it is obvious that the data transfer should start at $t[0]$ since the path-switching delay is negligible without further delay for changing bandwidths.

D. Heuristic Scheduling Algorithms for 2VPVB-1

For 2VPVB-1, the path-switching delay is non-negligible. Moreover, performing a path switching at different time points (i.e., at the end of one time-slot or at the beginning of its succeeding time-slot) may lead to different performances if the path bandwidths are different across two adjacent time-slots. It is favorable to perform path-switching in the time-slot with a smaller bandwidth between two adjacent time slots, as data transfer is suspended during the period of path switching.

D.1 Greedy 2VPVB-1 Algorithm

We design a polynomial-time greedy algorithm, Greedy2VPVB-1, whose pseudocode is provided in Algorithm 6. Starting from time-slot 0 to $T-1$, in each time-slot i, it computes two node-disjoint paths $p_1[i]$ and $p_2[i]$ with the maximum bandwidths $\beta_1[i]$ and $\beta_2[i]$ using Dijkstra's algorithm, respectively. Taking path p_1 as an example, path-switching in time-slot i falls in several cases: i) When $\beta_1[i] \ge \beta_1[i-1]$ and $\beta_1[i] \ge \beta_1[i+1]$, it does not perform path-switching and the transferred data size is $\beta_1[i] \cdot (t[i+1] - t[i])$, ii) when $\beta_1[i] < \beta_1[i-1]$ and $\beta_1[i] < \beta_1[i+1]$, there are two path switchings (i.e., both the beginning and the end of time-slot i), and the transferred data

Algorithm 6 Greedy2VPVB-1

Input: an HPN graph $G(V, E)$ with an ATB list, source v_s, destination v_d, data size δ and path switching delay τ
Output: the earliest transfer end time t_{end}
for $0 \leq i \leq T-1$ **do**
$\beta_1[i] =$ the widest bandwidth of path $p_1[i]$ from v_s to v_d in time slot i in G;
Remove the nodes and links of $p_1[i]$ from G to create a new graph G' in time slot i;
$\beta_2[i] =$ the widest bandwidth of path $p_2[i]$ from v_s to v_d in time slot i in G';
$\beta[i] = \beta_1[i] + \beta_2[i]$;
for $0 \leq i \leq T-1$ **do**
if $\delta \leq \beta[i] \cdot (t[i+1] - t[i])$ **then**
$t_{\text{end}} = t[i] + \delta/\beta[i]$;
Break;
else
if $\beta[i] \geq \beta[i+1]$ **then**
$\delta = \delta - \beta[i] \cdot (t[i+1] - t[i])$;
else
if $(\beta[i] < \beta[i-1])$ and $(i \neq 0)$ **then**
$\delta = \delta - \beta[i] \cdot (t[i+1] - t[i] - 2 \cdot \tau)$;
else
$\delta = \delta - \beta[i] \cdot (t[i+1] - t[i] - \tau)$;
return t_{end}.

size is $\beta[i] \cdot (t[i+1] - t[i] - 2 \cdot \tau)$, and iii) when $\beta_1[i] > \beta_1[i-1]$ and $\beta_1[i] < \beta_1[i+1]$, it performs the path switching at the end of time-slot i, and when $\beta_1[i] < \beta_1[i-1]$ and $\beta_1[i] > \beta_1[q+1]$, it performs path switching at the beginning of time-slot i. The transferred data size in both subcases of case iii) above is $\beta[i] \cdot (t[i+1] - t[i] - \tau)$. On path p_2, it performs path switching between two adjacent time-slots in the same way.The data size to be transferred by each path is also proportional to its bandwidth during the data transfer period. The time complexity of Greedy2VPVB-1 is $O(T \cdot (|V|^2 + T))$.

D.2 Improved Algorithm for 2VPVB-1

In Greedy2VPVB-1, we perform path switching in the time-slot with lower bandwidth, but frequent path switchings may cause a considerable overhead. We design an improved algorithm to reduce the number of path switchings, referred to Imp2VPVB-1, whose pseudocode is provided in Algorithm 7.

In Line 3, it initializes the number of path-switchings in each time slot for each path. We use $k_1[i]$ and $k_2[i]$ to denote the number of path switchings on $p_1[i]$ and $p_2[i]$ in time slot i, respectively. We also use δ' to denote the remaining data size at time point $t[i]$ (i.e., the beginning of time-slot i), and use $\delta'[j]$ to denote the size of data transferred in time-slot j.

In Lines 4–32, it iterates through each time slot and check if the data transfer can be finished.

While in Lines 5–19, for each i, it checks if the remainder data size can be transferred in this time slot. If data transfer can be completed in time-slot i, it computes the transfer end time according to different path-switching scenarios: i) When the path-switching time of two paths in time-slot i is the same (i.e., $k_1[i] = k_2[i]$), the transfer time is updated to $tt = t[i] + k_1[i] \cdot \tau + \delta'/(\beta_1[i] + \beta_2[i])$, ii) when the number of path-switchings of $p_1[i]$ is larger than the number of path-switchings of $p_2[i]$ (i.e., $k_1[i] > k_2[i]$), the transfer time is updated to $tt = t[i] + k_1[i] \cdot \tau + \delta'/(\beta_1[i] + \beta_2[i])$, and iii) when the number of path-switchings of $p_1[i]$ is smaller than the number of path-switchings of $p_2[i]$ (i.e.,$k_1[i] < k_2[i]$), the transfer time is updated to $tt = t[i] + k_2[i] \cdot \tau + \delta'/(\beta_1[i] + \beta_2[i])$. Since path switching may be performed at the end of one time-slot or at the beginning of its preceding time-slot, the value of $k_1[i]$ and $k_2[i]$ could be $0, 1, 2$ (the value could be 0 and 1 if i is the first or the last transfer time-slot). In Fig. 4, we illustrate different path switching cases in time-slot 1 for p_1, where the number of path switchings $k_1[1]$ is 0, 1, 1, and 2 in Figs. 4(a), 4(b), 4(c), and 4(d), respectively. For convenience, we set the time-slot to be 1 time unit, and path-switching delay to be 0.1 time unit.

If the data movement can be completed before time point

Algorithm 7 Imp2VPVB-1

Input: an HPN graph $G(V, E)$ with an ATB list, source v_s, destination v_d, data size δ, and path switching delay τ
Output: the earliest transfer end time t_{end}
1: **for** $0 \leq i \leq T-1$ **do**
2: Find a path-pair $p_1[i]$ and $p_2[i]$ with the the widest bandwidth $\beta_1[i]$ and $\beta_2[i]$, respectively, from v_s to v_d in time slot i;
3: $\delta' = \delta, k_1[] = 0, k_2[] = 0$;
4: **for** $0 \leq i \leq T-1$ **do**
5: **for** $0 \leq j \leq i$ **do**
6: $\delta'[j] = \beta_1[j] \cdot ((t[j+1] - t[j]) - k_1[j] \cdot \tau) + \beta_2[j] \cdot ((t[j+1] - t[j]) - k_2[j] \cdot \tau)$;
7: $\delta' = \delta' - \delta'[j]$;
8: **if** $\delta' \leq 0$ **then**
9: $\delta' = \delta' + \delta'[i]$;
10: **if** $k_1[i] = k_2[i]$ **then**
11: $tt = t[i] + k_1[i] \cdot \tau + \delta'/(\beta_1[i] + \beta_2[i])$;
12: **else**
13: **if** $k_1[i] > k_2[i]$ **then**
14: $\delta' = \delta' - (k_1[i] - k_2[i]) \cdot \beta_2[i]$;
15: $tt = t[i] + k_1[i] \cdot \tau + \delta'/(\beta_1[i] + \beta_2[i])$;
16: **else**
17: $\delta' = \delta' - (k_2[i] - k_1[i]) \cdot \beta_1[i]$;
18: $tt = t[i] + k_2[i] \cdot \tau + \delta'/(\beta_1[i] + \beta_2[i])$;
19: $t_{\text{end}} = t[0] + tt$;
20: **Break;**
21: Compare $data[i, i+1]$ using different schemes in Table 1, and select the one with maximum $data[i, i+1]$;
22: **if** $p_1[i+1] \neq p_1[i]$ **then**
23: **if** $\beta_1[i+1] > \beta_1[i]$ **then**
24: $k_1[i] = k_1[i] + 1$;
25: **else**
26: $k_1[i+1] = k_1[i+1] + 1$;
27: **if** $p_2[i+1] \neq p_2[i]$ **then**
28: **if** $\beta_2[i+1] > \beta_2[i]$ **then**
29: $k_2[i] = k_2[i] + 1$;
30: **else**
31: $k_2[i+1] = k_2[i+1] + 1$;
32: **return** t_{end}.

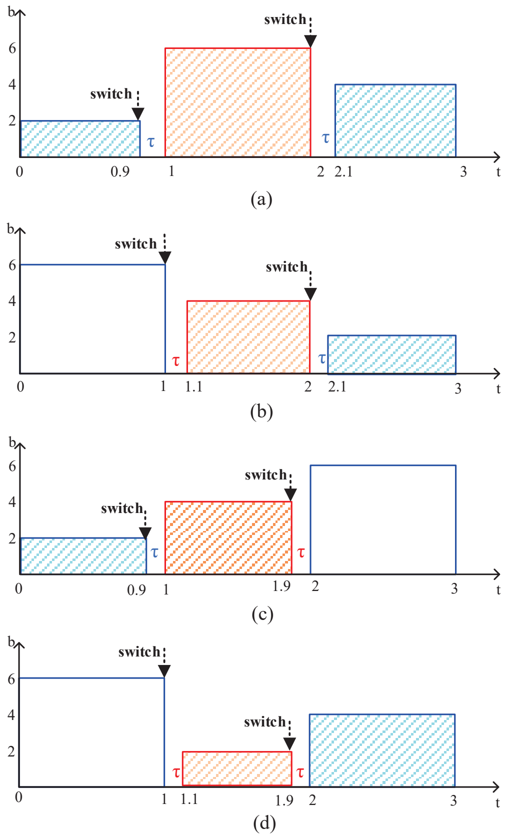


Fig. 4. Illustration of path switchings in time slot 1 on p_1: (a) $k_1[1] = 0$, (b) $k_1[1] = 1$, (c) $k_1[1] = 1$, and (d) $k_1[1] = 2$.

$t[i+1]$ (i.e., in time-slot i), we compute the remaining data size δ' at time point $t[i]$ and the transfer end time according to both the path-switching counts of the two node-disjoint paths:

i) when $k_1[i] = k_2[i]$, the concurrent transfer end time is $tt = t[i] + k_1[i] \cdot \tau + \delta'/(\beta_1[i] + \beta_2[i])$;
ii) when $k_1[i] > k_2[i]$, then $tt = t[i] + k_1[i] \cdot \tau + (\delta' - (k_1[i] - k_2[i]) \cdot \beta_2[i])/(\beta_1[i] + \beta_2[i])$;
iii) when $k_1[i] < k_2[i]$, then $tt = t[i] + k_2[i] \cdot \tau + (\delta' - (k_2[i] - k_1[i]) \cdot \beta_1[i])/(\beta_1[i] + \beta_2[i])$.

In Line 21, if the data transfer has not yet been completed in time-slot i, it continues to transfer in the next time-slot $i + 1$. We compute the maximum amount of data to be transferred in time-slot $[i, i+1]$ by analyzing possible path-pairs and their switching schemes. We first define several notations to facilitate the explanation of our path switching scheme:

- $p_1[i]$,$p_2[i]$: Two node-disjoint paths with maximum bandwidth in the current time-slot i.
- $\beta_1[i]$,$\beta_2[i]$: Bandwidth of $p_1[i]$ and $p_2[i]$, respectively.
- $\beta_1[i+1]$,$\beta_2[i+1]$: Bandwidth of $p_1[i+1]$ and $p_2[i+1]$, respectively.
- $p'_1[i+1]$: Path with the widest bandwidth after deleting $p_2[i]$ in time slot $i+1$.
- $p'_2[i+1]$: Path with the widest bandwidth after deleting $p_1[i]$ in time slot $i+1$.
- $\beta'_1[i+1]$: Bandwidth of path $p'_1[i+1]$.
- $\beta'_2[i+1]$: Bandwidth of path $p'_2[i+1]$.
- $data[i, i+1]$: Amount of data movement during time-slot $[i, i+1]$.

To improve the bandwidths of path-pair in time-slot $i+1$ and avoid path switching delay between these two time-slots, we design six different types of path switching schemes between time-slot i and $i+1$ as shown in Table 1, and select the one with the largest amount of data movement $data[i, i+1]$. For each scheme, the amount of data movement $data[i, i+1]$ is calculated as the sum of transferred data by two paths during time-slot $[i, i+1]$.

In Table 1, we consider the following path switching scenarios:

i) When the bandwidth sum of two paths retains their originally computed value in time-slot $i+1$, we further consider two cases: Type 1 keeps both originally computed paths invariable in $i+1$ time-slot while Type 2 exchanges two paths in $i+1$ time-slot.
ii) When path p_1 does not switch between time-slot i and $i+1$ (i.e., in time-slot $i+1$, we still use path $p_1[i]$, although the bandwidth of this path may be changed), we reduce the path switching delay of p_1 and further consider two cases: Type 3 computes a new path $p'_2[i+1]$ disjoint from $p_1[i]$ while Type 4 keeps originally computed path $p_2[i+1]$ invariable in $i+1$ time-slot.
iii) When path p_2 does not switch between time-slot i and $i+1$ (i.e., in time-slot $i+1$, we still use path $p_2[i]$, although the bandwidth of this path may be changed), we reduce the path switching delay p_2 and further consider two cases: Type 5 computes a new path $p'_1[i+1]$ disjoint from $p_2[i]$ while Type 6 keeps originally computed path $p_1[i+1]$ invariable in $i+1$ time-slot.

In each case, we compute the amount of moved data $data[i, i+1]$ during time-slot $[i, i+1]$. The calculation formulas are provided in Table 1. After comparing the amount of moved data $data[i, i+1]$ in all these schemes, we select an optimal scheme with the largest amount of moved data during time-slot $[i, i+1]$.

In Lines 22–31, according to the selected path switching scheme, we compute the number of path switchings on each path in time-slots i and $i+1$ for the next round of calculation.

Since the time complexity of Dijkstra's algorithm is $O(|V|^2)$, the time complexity of Imp2VPVB-1 is $O(T \cdot |V|^2 + T^2)$, where T is the total number of new time slots in the ATB list, and $|V|$ is the number of nodes.

V. PERFORMANCE EVALUATION

For performance evaluation, we implement the proposed algorithms and conduct i) proof-of-concept experiments on an emulated SDN testbed based on the Mininet [6] system, and ii) extensive simulations in randomly generated networks as well as a real-life HPN topology.

Table 1. Different path switching schemes.

Type	Path-pair in time-slot i	Path-pair in time-slot $i+1$	Compare paths between time-slot $[i,i+1]$			Switch time point	Data movement during time-slot $[i,i+1]$
1	$p_1[i]$	$p_1[i+1]$	$p_1[i+1]=p_1[i]$			no switching	$\beta_1[i](t[i+1]-t[i])+\beta_1[i+1](t[i+2]-t[i+1])$
			$p_1[i+1]\neq p_1[i]$	$\beta_1[i+1]>\beta_1[i]$		end of i	$\beta_1[i](t[i+1]-t[i]-\tau)+\beta_1[i+1](t[i+2]-t[i+1])$
				$\beta_1[i+1]<\beta_1[i]$		beginning of i+1	$\beta_1[i](t[i+1]-t[i])+\beta_1[i+1](t[i+2]-t[i+1]-\tau)$
	$p_2[i]$	$p_2[i+1]$	$p_2[i+1]=p_2[i]$			no switching	$\beta_2[i](t[i+1]-t[i])+\beta_2[i+1](t[i+2]-t[i+1])$
			$p_2[i+1]\neq p_2[i]$	$\beta_2[i+1]>\beta_2[i]$		end of i	$\beta_2[i](t[i+1]-t[i]-\tau)+\beta_2[i+1](t[i+2]-t[i+1])$
				$\beta_2[i+1]<\beta_2[i]$		beginning of i+1	$\beta_2[i](t[i+1]-t[i])+\beta_2[i+1](t[i+2]-t[i+1]-\tau)$
2	$p_1[i]$	$p_2[i+1]$	$p_2[i+1]=p_1[i]$			no switching	$\beta_1[i](t[i+1]-t[i])+\beta_2[i+1](t[i+2]-t[i+1])$
			$p_2[i+1]\neq p_1[i]$	$\beta_2[i+1]>\beta_1[i]$		end of i	$\beta_1[i](t[i+1]-t[i]-\tau)+\beta_2[i+1](t[i+2]-t[i+1])$
				$\beta_2[i+1]<\beta_1[i]$		beginning of i+1	$\beta_1[i](t[i+1]-t[i])+\beta_2[i+1](t[i+2]-t[i+1]-\tau)$
	$p_2[i]$	$p_1[i+1]$	$p_1[i+1]=p_2[i]$			no switching	$\beta_2[i](t[i+1]-t[i])+\beta_1[i+1](t[i+2]-t[i+1])$
			$p_1[i+1]\neq p_2[i]$	$\beta_1[i+1]>\beta_2[i]$		end of i	$\beta_2[i](t[i+1]-t[i]-\tau)+\beta_1[i+1](t[i+2]-t[i+1])$
				$\beta_1[i+1]<\beta_2[i]$		beginning of i+1	$\beta_2[i](t[i+1]-t[i])+\beta_1[i+1](t[i+2]-t[i+1]-\tau)$
3	$p_1[i]$	$p_1[i]$	$p_1[i+1]=p_1[i]$			no switching	$\beta_1[i](t[i+1]-t[i])+\beta_1[i+1](t[i+2]-t[i+1])$
	$p_2[i]$	$p_2^{'}[i+1]$	$p_2^{'}[i+1]=p_2[i]$			no switching	$\beta_2[i](t[i+1]-t[i])+\beta_2^{'}[i+1](t[i+2]-t[i+1])$
			$p_2^{'}[i+1]\neq p_2[i]$	$\beta_2^{'}[i+1]>\beta_2[i]$		end of i	$\beta_2[i](t[i+1]-t[i]-\tau)+\beta_2^{'}[i+1](t[i+2]-t[i+1])$
				$\beta_2^{'}[i+1]<\beta_2[i]$		beginning of i+1	$\beta_2[i](t[i+1]-t[i])+\beta_2^{'}[i+1](t[i+2]-t[i+1]-\tau)$
4	$p_1[i]$	$p_1[i]$	$p_1[i]$ intersects $p_2[i+1]$				0
			$p_1[i]$ disjoint $p_2[i+1]$	$p_1[i+1]=p_1[i]$		no switching	$\beta_1[i](t[i+1]-t[i])+\beta_1[i+1](t[i+2]-t[i+1])$
	$p_2[i]$	$p_2[i+1]$		$p_2[i+1]=p_2[i]$		no switching	$\beta_2[i](t[i+1]-t[i])+\beta_2[i+1](t[i+2]-t[i+1])$
				$p_2[i+1]\neq p_2[i]$	$\beta_2[i+1]>\beta_2[i]$	end of i	$\beta_2[i](t[i+1]-t[i]-\tau)+\beta_2[i+1](t[i+2]-t[i+1])$
					$\beta_2[i+1]<\beta_2[i]$	beginning of i+1	$\beta_2[i](t[i+1]-t[i])+\beta_2[i+1](t[i+2]-t[i+1]-\tau)$
5	$p_1[i]$	$p_1^{'}[i+1]$	$p_1^{'}[i+1]=p_1[i]$			no switching	$\beta_1[i](t[i+1]-t[i])+\beta_1^{'}[i+1](t[i+2]-t[i+1])$
			$p_1^{'}[i+1]\neq p_1[i]$	$\beta_1^{'}[i+1]>\beta_1[i]$		end of i	$\beta_1[i](t[i+1]-t[i]-\tau)+\beta_1^{'}[i+1](t[i+2]-t[i+1])$
				$\beta_1^{'}[i+1]<\beta_1[i]$		beginning of i+1	$\beta_1[i](t[i+1]-t[i])+\beta_1^{'}[i+1](t[i+2]-t[i+1]-\tau)$
	$p_2[i]$	$p_2[i]$	$p_2[i+1]=p_2[i]$			no switching	$\beta_2[i](t[i+1]-t[i])+\beta_2[i+1](t[i+2]-t[i+1])$
6	$p_1[i]$	$p_1[i+1]$	$p_1[i+1]$ intersects $p_2[i]$				0
			$p_1[i+1]$ disjoint $p_2[i]$	$p_1[i+1]=p_1[i]$		no switching	$\beta_1[i](t[i+1]-t[i])+\beta_1[i+1](t[i+2]-t[i+1])$
				$p_1[i+1]\neq p_1[i]$	$\beta_1[i+1]>\beta_1[i]$	end of i	$\beta_1[i](t[i+1]-t[i]-\tau)+\beta_1[i+1](t[i+2]-t[i+1])$
					$\beta_1[i+1]<\beta_1[i]$	beginning of i+1	$\beta_1[i](t[i+1]-t[i])+\beta_1[i+1](t[i+2]-t[i+1]-\tau)$
	$p_2[i]$	$p_2[i]$		$p_2[i+1]=p_2[i]$		no switching	$\beta_2[i](t[i+1]-t[i])+\beta_2[i+1](t[i+2]-t[i+1])$

A. Experiment-based Performance Evaluation

A.1 Mininet Testbed Setup

The Mininet emulation tool has been widely used for constructing a virtual network topology [9]. It is also suitable for evaluating the performance of scheduling algorithms as the overhead introduced by the scheduling process in Mininet emulation is marginal compared to the scheduling algorithm running time, and is almost negligible in large cases [11].

Based on the Mininet system, we emulate each switch in the virtual network topology using a virtual instance of Open vSwitch [7], and choose OpenDaylight [8] as the OpenFlow controller, which is a Java-based modular open platform for customizing and automating networks of any size and scale. We deploy a small virtual network testbed emulating a 7-site wide-area network, as shown in Fig. 5. On this testbed, we set the bandwidth capacity of each link between two OpenFlow switches to be 10 Gb/s, and the path switching delay is measured to be $\tau = 0.1$ s.

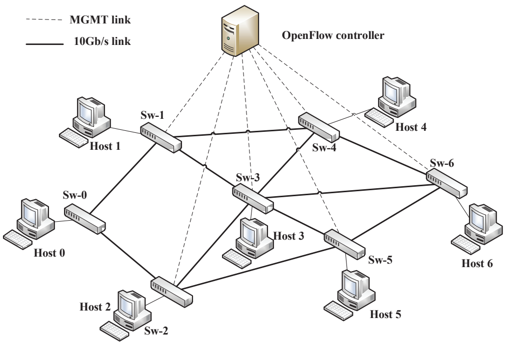


Fig. 5. A Mininet emulated network testbed.

A.2 Performance Comparison

A.2.a Illustration of a Scheduling Instance. For illustration, we first conduct a scheduling experiment on the emulated testbed over a period of total 4 time slots, among which the smallest one is of 1 time unit. The available bandwidths of the network links across $[0,3]$ time slots are provided in Table 2.

In this experiment, one bulk data transfer request r_{0-6} with data size $\delta = Gbits$, source s_0, and destination s_6 is submitted. We calculate two node-disjoint paths using different algorithms in deferent scheduling models, i.e., 2VPFB-0, 2VPFB-1, and 2VPVB-1. The corresponding scheduling results are provided in Tables 3, 4, and 5, respectively.

From Table 3, we observe that the proposed Imp2VPFB-0 algorithm outperforms Greedy2VPFB-0 by 20% in the scheduling model of 2VPFB-0 in terms of earliest completion time (ECT).

From Table 4, we observe that the proposed Imp2VPFB-1 algorithm outperforms Greedy2VPFB-1 by 25% in the scheduling model of 2VPFB-1 in terms of ECT.

From Table 5, we observe that the proposed Imp2VPVB-1 algorithm outperforms Greedy2VPVB-1 by about 3% in the scheduling model of 2VPFB-1 in terms of ECT. Note that Imp2VPVB-1 only reduces the number of switching times compared with Greedy2VPVB-1.

Table 2. Link bandwidths in Gb/s across [0,3] time slots on the network testbed in Fig. 5.

Time slots \ Links	S_0-S_1	S_0-S_2	S_1-S_3	S_1-S_4	S_2-S_3	S_2-S_5	S_3-S_4	S_3-S_5	S_3-S_6	S_4-S_6	S_5-S_6
0	4	3	3	4	2	4	1	2	4	1	1
1	2	3	2	3	3	1	1	2	1	2	1
2	7	8	10	6	8	2	10	10	8	6	6
3	5	7	5	9	6	2	2	5	2	5	10

Table 3. Scheduling results of different algorithms for request r_{0-6} in the scheduling model of 2VPFB-0 on the testbed.

Algorithms	Time slots	Paths	Bandwidths (Gb/s)	Transfer bandwidths (Gb/s)	ECT (s)
Greedy2VPFB-0	0	$p_1: S_0-S_1-S_3-S_6$	3	3	3.33
		$p_2: S_0-S_2-S_5-S_6$	1		
	1	$p_1: S_0-S_1-S_4-S_6$	2		
		$p_2: S_0-S_2-S_3-S_6$	1		
	2	$p_1: S_0-S_2-S_3-S_6$	8		
		$p_2: S_0-S_1-S_4-S_6$	6		
	3	$p_1: S_0-S_1-S_4-S_6$	5		
		$p_2: S_0-S_2-S_3-S_5-S_6$	5		
Imp2VPFB-0	2	$p_1: S_0-S_2-S_3-S_6$	8	14	2.71
		$p_2: S_0-S_1-S_4-S_6$	6		

Table 4. Scheduling results of different algorithms for request r_{0-6} in the scheduling model of 2VPFB-1 on the emulated network testbed.

Algorithms	Time slots	Paths	Bandwidths (Gb/s)	Transfer bandwidths (Gb/s)	Switching times	ECT (s)
Greedy2VPFB-1	0	$p_1: S_0-S_1-S_3-S_6$	3	3	p_1, p_2 both 3	3.63
		$p_2: S_0-S_2-S_5-S_6$	1			
	1	$p_1: S_0-S_1-S_4-S_6$	2			
		$p_2: S_0-S_2-S_3-S_6$	1			
	2	$p_1: S_0-S_2-S_3-S_6$	8			
		$p_2: S_0-S_1-S_4-S_6$	6			
	3	$p_1: S_0-S_1-S_4-S_6$	5			
		$p_2: S_0-S_2-S_3-S_5-S_6$	5			
Imp2VPFB-1	2	$p_1: S_0-S_2-S_3-S_6$	8	14	0	2.71
		$p_2: S_0-S_1-S_4-S_6$	6			

Table 5. Scheduling results of different algorithms for request r_{0-6} in the scheduling model of 2VPVB-1 on the emulated network testbed.

Algorithms	Time slots	Paths	Bandwidths (Gb/s)	Transfer bandwidths (Gb/s)	Switching times	ECT (s)
Greedy2VPVB-1	0	$p_1: S_0-S_1-S_3-S_6$	3	4	p_1, p_2 both 2	2.26
		$p_2: S_0-S_2-S_5-S_6$	1			
	1	$p_1: S_0-S_1-S_4-S_6$	2	3		
		$p_2: S_0-S_2-S_3-S_6$	1			
	2	$p_1: S_0-S_2-S_3-S_6$	8	14		
		$p_2: S_0-S_1-S_4-S_6$	6			
Imp2VPVB-1	0	$p_1: S_0-S_1-S_3-S_6$	3	4	p_1 1 p_2 1	2.23
		$p_2: S_0-S_2-S_5-S_6$	1			
	1	$p_1: S_0-S_1-S_4-S_6$	2	3		
		$p_2: S_0-S_2-S_5-S_6$	1			
	2	$p_1: S_0-S_1-S_4-S_6$	6	14		
		$p_2: S_0-S_2-S_3-S_6$	8			

A.2.b Performance Evaluation With Varying Data Sizes. For a thorough performance evaluation, we conduct more emulation-based experiments with varying data sizes on the testbed. Each of these experiments spans across total 10 time slots, among which, the smallest one is of 1 time unit, and the link bandwidth between any two switches is initialized to be the link capacity of 10 Gb/s, as determined by each switch's line card speed. The available bandwidths of the network links across $[0, 9]$ time slots are provided in Table 6.

We repeat the scheduling experiments under different data sizes increasing from 10 Gbits to 100 Gbits with a step of 10 Gbits. The transfer end time measurements in different scheduling models are plotted in Figs. 6, 7, and 8, respectively. In all these experiments, we observe that the proposed algorithms consistently outperform the other algorithms in comparison.

B. Simulation-based Performance Evaluation

B.1 Simulation Setup

For performance evaluation, we generate a set of networks containing different numbers of nodes and links of random bandwidths within an appropriate range. We randomly select a source node v_s and a destination node v_d in each network. There are 100 time slots in total and the start time $t[0] = 0$. The link bandwidths follow a normal distribution: $b = b_{max} \cdot e^{-\frac{1}{2}(x)^2}$, where b_{max} is set to be 100 Gb/s, which is the capacity of most production backbone networks nowadays, and x is a random variable within the range of $[0, 1]$.

We investigate the scheduling performance of the proposed algorithms as network size scales up in Section V.B.2, and as both network size and data volume increase simultaneously, as faced by many big data applications, in Section V.B.3.

Table 6. Link bandwidths in Gb/s across [0, 9] time slots on the emulated network testbed in Fig. 5.

Time slots \ Links	$S_0 - S_1$	$S_0 - S_2$	$S_1 - S_3$	$S_1 - S_4$	$S_2 - S_3$	$S_2 - S_5$	$S_3 - S_4$	$S_3 - S_5$	$S_3 - S_6$	$S_4 - S_6$	$S_5 - S_6$
0	3	3	3	1	1	1	2	3	2	2	2
1	2	3	1	3	3	2	1	1	3	2	3
2	6	10	9	10	7	9	8	6	10	8	8
3	10	6	7	10	7	6	10	6	6	8	9
4	8	10	10	7	6	9	7	8	10	10	8
5	9	8	8	10	9	7	8	8	6	10	10
6	9	10	8	10	8	6	10	6	10	7	7
7	8	10	8	6	9	6	8	7	9	8	10
8	10	8	8	10	8	7	6	9	10	6	9
9	9	7	6	8	7	10	7	7	7	10	6

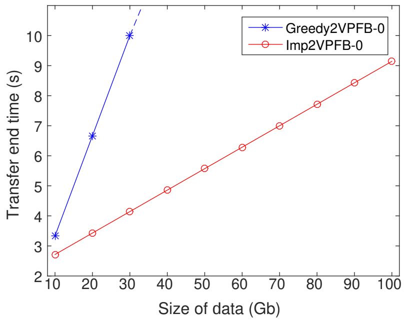


Fig. 6. Performance comparison for 2VPFB-0 as data sizes vary on the testbed.

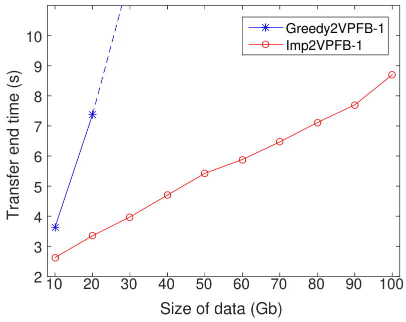


Fig. 7. Performance comparison for 2VPFB-1 as data sizes vary on the testbed.

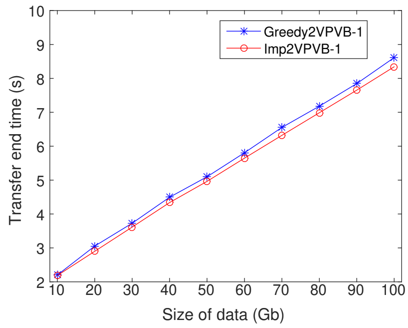


Fig. 8. Performance comparison for 2VPVB-1 as data sizes vary on the testbed.

B.2 Evaluation of Scheduling Performance for 2VPFB-0/1 and 2VPVB-1 with Varying Network Sizes

For performance evaluation, we randomly generate 15 different large-scale networks, indexed from 1 to 15, as shown in Table 7. In these randomly generated networks, we set the data volume for transfer to be 1000 GByte. In each of these network instance, we run Greedy2VPFB-0/1, Imp2VPFB-0/1, Greedy2VPVB-1, and Imp2VPVB-1 for 10 times with different random seeds, and measure each algorithm's average performance and standard deviation.

For 2VPFB-0/1, we plot the mean and standard deviation of the data transfer end time obtained by Greedy2VPFB-0/1 and Imp2VPFB-0/1 in Figs. 9 and 10, respectively. We observe that Imp2VPFB-0/1 achieve about 10–20% and 12–35% performance improvement on average over Greedy2VPFB-0/1, respectively.

We also evaluate the performance of Greedy2VPVB-1 and Imp2VPVB-1 for 2VPVB-1 in the same set of networks, and plot the performance measurements in Fig. 11. We observe that Imp2VPVB-1 also consistently outperforms Greedy2VPVB-1 in all the cases with about 5% performance improvement.

We would like to point out that bandwidth is the dominating factor that determines data transfer end time for a given data volume in a given network. We notice that the algorithm performance improvement for 2VPVB-1 is less significant than 2VPFB-0/1. This is because Greedy2VPVB-1 has already considered maximum data transfer in every time slot, and Imp2VPVB-1 only reduces the number of path switchings between different time-slots, hence yielding a limited performance gain. For 2VPFB-0, Imp2VPFB-0 may postpone data transfer to a later time-slot to use a higher bandwidth than Greedy2VPFB-0; while for 2VPFB-1 with path-switching delay, Imp2VPFB-1 further reduces the number of path switchings between different time-slots to improve the performance over Greedy2VPFB-0.

B.3 Evaluation of Scheduling Performance for 2VPFB-0/1 and 2VPVB-1 With Varying Network Sizes and Data Volumes

We randomly generate 15 different large-scale networks. The data size to be transferred varies within a range from 1000 Gbytes to 3000 Gbytes.

For 2VPFB-0/1, in each of these 15 large-scale networks and for each data size, we run Imp2VPFB-0/1 and Greedy2VPFB-0/1 for 10 times, and plot the average data transfer end time in Figs. 12 and 13. We observe that Imp2VPFB-0/1 outperforms Greedy2VPFB-0/1 in all of the cases with about 10–20% and 12–35% performance improvement on average, respectively.

We also evaluate the performance of Imp2VPVB-1 and Greedy2VPVB-1 for 2VPVB-1 in the same set of networks, and plot their performance measurements in Fig. 14. Similarly, Imp2VPVB-1 outperforms Greedy2VPVB-1 in all the cases with about 5% performance improvement. Since Greedy2VPVB-1 has already considered maximum data transfer in every time slot, and Imp2VPVB-1 attempts to reduce the number of path switchings, the performance gain is limited.

These simulation results show that transfer end time is largely determined by data volume, not network size, and are qualitatively similar to those in Section V.B.2.

B.4 Performance Comparison for 2VPFB-0/1 and 2VPVB-1 in ESnet5

To further evaluate the performance of the proposed algorithms, we conduct scheduling experiments using the topology of a real-life HPN, ESnet5 of U.S. Department of Energy [34], with 57 nodes and 65 links, as shown in Fig. 15. We vary the vol-

Table 7. Index of 15 large-scale simulated networks.

Index of network size	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Number of nodes	40	50	60	70	80	90	100	120	150	200	230	260	290	320	350
Number of links	80	100	120	140	160	180	200	240	300	400	450	500	520	540	560

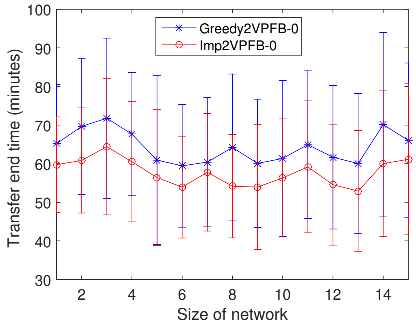


Fig. 9. Performance comparison for 2VPFB-0 as network size vary.

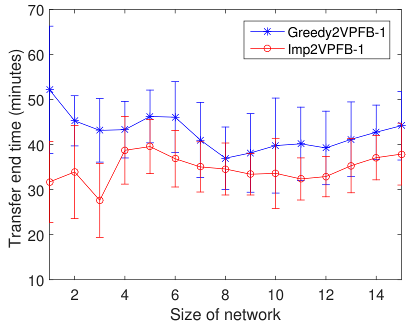


Fig. 10. Performance comparison for 2VPFB-1 as network size vary.

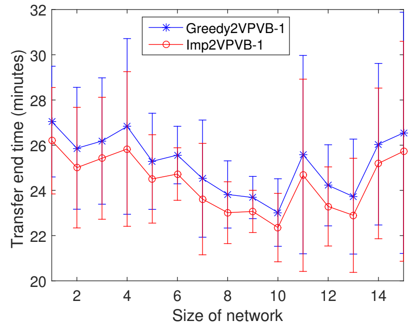


Fig. 11. Performance comparison for 2VPVB-1 as network size vary.

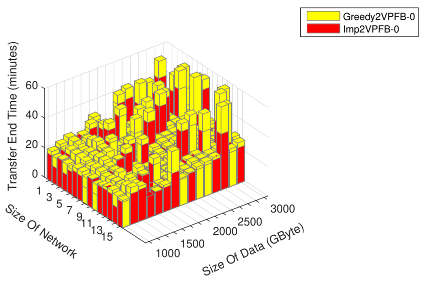


Fig. 12. Performance comparison of the algorithms for 2VPFB-0 in large networks as both data size and network size vary.

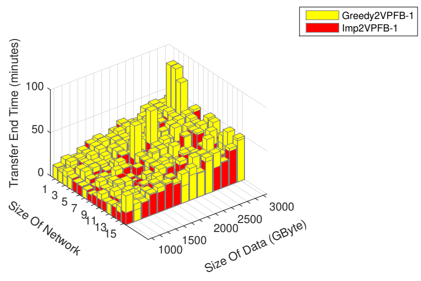


Fig. 13. Performance comparison of the algorithms for 2VPFB-1 in large networks as both data size and network size vary.

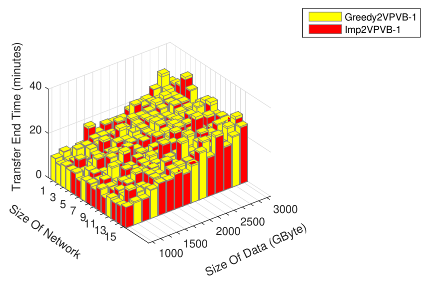


Fig. 14. Performance comparison of the algorithms for 2VPVB-1 in large networks as both data size and network size vary.

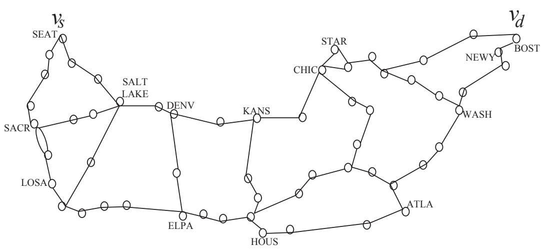


Fig. 15. The topology of ESnet5.

ume of data within a range from 1150 GBytes to 2750 GBytes in the experiments. The corresponding performance measurements for 2VPFB-0/1 in ESnet are plotted in Figs. 16 and 17, and the performance measurements for 2VPVB-1 in ESnet are plotted in Fig. 18. These results show that the improved algorithms consistently achieve better performance than greedy heuristic algorithms.

Again, the simulation results from ESnet show that data size is the dominating factor that determines transfer end time in a given network, and are qualitatively similar to those in Section V.B.2.

VI. CONCLUSION

We investigated a bandwidth scheduling problem with two variable node-disjoint paths of fixed and variable bandwidth in dedicated networks, in each of which, we further considered two subcases according to the negligibility of path switching delay. We proved these four problem variants to be NP-complete, and designed a heuristic for each. The performance superiority of the proposed heuristics was verified by extensive simulation results in a large set of simulated and real-life networks in comparison with greedy strategies. It is of our future interest to incorporate and test these scheduling algorithms in the control plane of existing HPNs.

REFERENCES

[1] N. S. V. Rao, W. R. Wing, S. M. Carter, and Q. Wu, "UltraScience net: Network testbed for large-scale science applications," *IEEE Commun. Mag.,* vol. 43, no. 11, pp. 12–17, 2005.

[2] X. Zheng, M. Veeraraghavan, N.S.V. Rao, Q. Wu, and M. Zhu, "CHEETAH: Circuit-switched high-speed end-to-end transport architecture testbed," *IEEE Commun. Mag.,* vol. 43, no. 11, pp. 11–17, 2005.

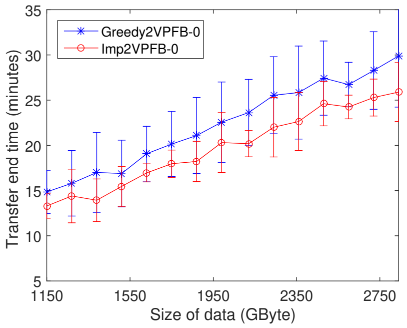


Fig. 16. Performance comparison for 2VPFB-0 in ESnet5.

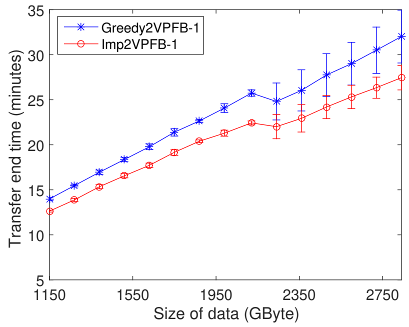


Fig. 17. Performance comparison for 2VPFB-1 in ESnet5.

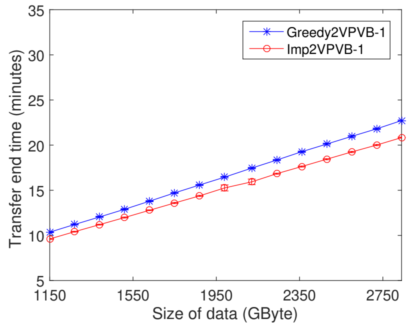
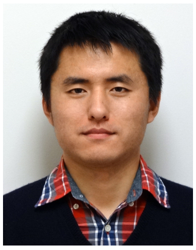


Fig. 18. Performance comparison for 2VPVB-1 in ESnet5.

[3] "OSCARS: On-demand secure circuits and advance reservation system," [Online] Available: http://www.es.net/oscars
[4] "Internet2 advanced layer 2 service," [Online] Available: https://goo.gl/4iAbQn
[5] A. Hou *et al.* "Bandwidth scheduling with multiple variable node-disjoint paths in high-performance networks," in *Proc. IEEE IPCCC*, 2016.
[6] "Mininet: An instant virtual network on your laptop (or other pc)," [Online] Available: http://mininet.org/
[7] "Open vSwitch: An open virtual switch," [Online] Available: http://openvswitch.org/
[8] "SDN controller," [Online] Available: http://www.opendaylight.org/
[9] P. Christoph, F. Simone, B. Olivier, and B.Olivier, "Experimental evaluation of multipath TCP schedulers," in *Proc. ACM SIGCOM CSWS,* 2014.
[10] I. Monga, E. Pouyoul, and C.Guok, "Software-defined networking for big-data science - Architectural models from campus to the WAN," in *Proc. IEEE SCC,* 2012.
[11] M. Aktas, G. Haldeman, and M. Parashar, "Scheduling and flexible control of bandwidth and in-transit services for end-to-end application workflows," *Future Generation Comput. Syst.,* vol. 56, pp. 284–294, 2016.
[12] M.Aktas, G. Haldeman, and M. Parashar, "Flexible scheduling and control of bandwidth and in-transit services for end-to-end application workflows," in *Proc. IEEE NDM,* 2014.
[13] J. Wang, M. Qiu, and B. Guo, "High reliable real-time bandwidth scheduling for virtual machines with hidden Markov predicting in telehealth platform," *Future Generation Comput. Syst.,* vol. 49, pp. 68–76, 2015.
[14] Y. Wang *et al.* "On periodic scheduling of fixed-slot bandwidth reservations for big data transfer," in *Proc. IEEE LCN,* 2015.
[15] A. Hou, C. Q. Wu, D. Fang, Y. Wang, and M. Wang, "Bandwidth scheduling with multiple fixed node-disjoint paths in high-performance networks," in *Proc. QSHINE,* 2016.
[16] A. Hou, C. Q. Wu, D. Fang, Y. Wang, and M. Wang, "Bandwidth scheduling for big data transfer using multiple fixed node-disjoint paths," *J. Network Comput. Applicat.,* vol. 85, pp. 47–55, 2017.
[17] M. Aihara, S. Kono, and K. Kinoshita, "Joint bandwidth scheduling and routing method for large file transfer with time constraint," in *Proc. IEEE NOMS,* 2016.
[18] P.Dharam, C. Q. Wu, and N. S. V. Rao, "Advance bandwidth scheduling in software-defined networks," in *Proc. IEEE GLOBECOM,* 2015.
[19] J. Domzal, Z. Dulinski, and M. Kantor, "A survey on methods to provide multipath transmission in wired packet networks," *Comput. Networks,* vol. 77, pp. 18–41, 2015.
[20] P. Mieghem and F. Kuipers, "On the complexity of QoS routing," *Comput. Commun.,* vol. 26, no. 4 pp. 376–387, 2003.
[21] F. Kuipers and P. VanMieghem, "Conditions that impact the complexity of QoS routing," *IEEE/ACM Trans. Netw.,* vol. 13, pp. 717–730, 2005.
[22] R. Bhandari, "Optimal diverse routing in telecommunication fiber networks," in *Proc. IEEE INFOCOM,* 1994.
[23] W. Liang, "Robust routing in wide-area WDM networks," in *Proc. IEEE IPDPS,* 2001.
[24] B. Shen, B. Hao, and A. Sen, "On multipath routing using widest pair of disjoint paths," in *Proc. IEEE HPSR Workshop,* 2004.
[25] M. Dahshan, "Maximum-bandwidth node-disjoint paths," *Int. J. Advanced Comput. Sci. Applicat.,* pp. 48–56, 2012.
[26] R. Bhatia, M. Kodialam, and T. V. Lakshman, "Finding disjoint paths with related path costs," *Springer Science+Business Media, LLC,* 2006.
[27] A. Andreas and J. C. Smith, "Exact algorithms for robust k-path routing problems," in *Proc. GO,* 2005.
[28] R. Loh, S. Soh, and M. Lazarescu, "Maximizing bandwidth using disjoint paths," in *Proc. IEEE AINA*, 2010.
[29] D. Sidhu, R. Nair, and S. Abdallah, "Finding disjoint paths in networks," in *Proc. ACM SIGCOMM*, 1991.
[30] S. Ramasubramanian, H. Krishnamoorthy, and M. Krunz, "Disjoint multipath routing using colored trees," *COMNET,* vol. 51, no. 8, pp. 2163–2180, 2007.
[31] Y. Lin and Q. Wu, "Complexity analysis and algorithm design for advance bandwidth scheduling in dedicated networks," *IEEE/ACM Trans. Netw.,* vol. 21, no. 1, pp. 14–27, 2013.
[32] Y. Lin and Q. Wu, "On design of bandwidth scheduling algorithms for multiple data transfers in dedicated networks," in *Proc. IEEE/ACM ANCS,* 2008.
[33] R. Guerin and A. Orda, "Networks with advance reservations: The routing perspective," in *Proc. IEEE INFOCOM,* 2000.
[34] "ESnet," [Online] Available: https://www.es.net
[35] L. Zuo and M. M. Zhu, "Concurrent bandwidth reservation strategies for big data transfers in high-performance networks," *IEEE Trans. Netw. Serv. Manage,* vol. 12, no. 2, pp. 232–247, 2015.
[36] L. Zuo, M. M. Zhu, and C.Q. Wu, "Concurrent bandwidth scheduling for big data transfer over a dedicated channel," *Int. J. Commun. Networks Distributed Syst.,* vol. 21, no. 1, 2014.
[37] L. Zuo, M. M. Zhu, C. Q. Wu, and J. Zurawski, "Fault-tolerant bandwidth reservation strategies for data transfers in high-performance networks," *Comput. Networks,* vol. 113, pp. 1–16, 2017.
[38] L. Zuo, M. M. Zhu, and C. Q. Wu, "Fast and efficient bandwidth reservation algorithms for dynamic network provisioning," *J. Network Syst. Manage.,* vol. 23, no. 3, pp. 420–444, 2015.

Aiqin Hou received the Ph.D. degree in school of information science and technology from Northwest University in 2018. She received the B.S. degree in Information Theory from XiDian University of China in 1989. She is currently an Associate Professor with the School of Information Science and Technology, Northwest University of China. Her research interests include big data, high performance network, and bandwidth scheduling.

Chase Qishi Wu received the Ph.D. degree in Computer Science from Louisiana State University in 2003. He was a research fellow at Oak Ridge National Laboratory during 2003-2006 and an Assistant and Associate Professor at University of Memphis during 2006–2015. He is currently a Professor with the Department of Computer Science and the Director of the Center for Big Data at New Jersey Institute of Technology. His research interests include big data, distributed and parallel computing, and computer networks.



Liudong Zuo received the Ph.D. degree in Computer Science from Southern Illinois University Carbondale in 2015. He received the B.E. degree in Computer Science from University of Electronic Science and Technology of China in 2009. He is currently an Assistant Professor in Computer Science Department at California State University, Dominguez Hills. His research interests include computer networks, algorithm design, and big data.

Xiaoyang Zhang received the M.S. degree in Communication and Information Systems from Northwest University of China in 2019 and the B.S. degree in Electronic Science and Technology from Xi'an University of Posts and Telecommunications of China in 2013. His research interests include bandwidth scheduling, high-performance networks, and big data management.

Tao Wang received the M.S. degree in Software Engineering from Northwest University of China in 2019, and the B.S. degree in Software Engineering from the Northwest University of China in 2016. He is now a Assistant Engineer in a software company in Xi'an, China. His research interests include bandwidth scheduling, high-performance networks and software networks.

Dingyi Fang is currently a Professor with the School of Information Science and Technology, Northwest University of China. His current research interests include mobile computing and distributed computing systems, network and information security, localization, social networks, and wireless sensor networks.