IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 10, 2018, accepted May 2, 2018, date of publication May 8, 2018, date of current version June 5, 2018.

*Digital Object Identifier 10.1109/ACCESS.2018.2834482*

# A Completion Time-Based Flow Scheduling for Inter-Data Center Traffic Optimization

**WENBO HU, JIANG LIU, TAO HUANG, AND YUNJIE LIU**
Future Network Theory and Application Laboratory, School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China

Corresponding author: Wenbo Hu (hu6360567@bupt.edu.cn)

This work was supported in part by the National High Technology Research and Development Program (863) of China under Grant 2015AA016101, in part by the National Natural Science Funds under Project 61671086, in part by the 111 Project under Grant B17007, and in part by the Director Funds of the Beijing Key Laboratory of Network System Architecture and Convergence under Grant 2017BKL-NSAC-ZJ-01.

**ABSTRACT** With the rise of cloud computing service, traffic in inter-data center wide area networks (inter-DC WANs) has been growing rapidly. Inter-data center traffic contains multitudinous data transfers that require to be accomplished within certain time periods or deadlines. The different completion time of the transfer tasks has distinct impacts on the performance of the whole system. However, very little work takes the completion time into account while guarantees the hard deadlines. Besides, most previous works arrange transfers exclusively, which does not take fairness into account. In this paper, we introduce a completion time-based model, which takes multiple deadlines and their impacts into consideration to profile a more accurate characterization of the data transfers. We propose a novel mechanism to achieve the goals of both maximizing utility and fairness. We also develop an inter-DC WAN emulation tool which enables a single commodity server to emulate arbitrary topology and dynamic configuration on the WAN links. Our evaluation demonstrates that our algorithm gains 110%–250% utility than current deadline focus mechanisms, while achieves 12% higher throughput. Fairness is improved over four times at best by adjusting the fairness coefficient.

**INDEX TERMS** Inter-DC WAN, software defined networking, traffic engineering.

## I. INTRODUCTION

Cloud service has been growing rapidly in recent years due to the continuous rise in numbers of data-hungry customers and applications. Cisco predicts that annual global data center IP traffic will reach 10.4 zettabytes (ZB) by the end of 2019, up from 3.4 zettabytes per year in 2014 [1].

It is crucial to understand and optimize the inter-data center traffic. Despite the high cost of wide area transportation, cloud service providers deploy dedicate wide area networks (WANs) to connect data centers [2]. These providers need to provision more bandwidth capacity than average for various reasons, such as burst transmission, or deploy extra links between data centers in case of link failure.

Recent research shows the interest in traffic engineering of inter-data center communication. Chen *et al.* [3] reveal that datacenter-to-datacenter (D2D) traffic is composed of datacenter-to-client (D2C) triggered traffic and background D2D traffic. Inter-data center traffic can be classified into three categories [4], based on time sensitivity: *interactive* traffic which needs to be done in a short time, *elastic* traffic that should be transferred within a duration that from a quarter hour to a couple of hours and *background* traffic that has no specific deadline or a very long one.

One fundamental property of the tasks in inter-data center communication is the deadline, either hard or soft. Missing the deadline may lead to unexpected consequences and severe performance drain, such as [5]. Therefore, it is important for an efficient traffic engineering to provide deadline guarantees for inter-data center transmission.

However, state-of-the-art techniques are not intended to guarantee the deadlines. B4 [6] tries to work along with existing routing protocols in Software Defined Networking (SDN) [7], [8] environment by building custom switches and mechanisms, while SWAN [4] gives a more detailed perspective by using commodity switches with limited forwarding table capacity. These coarse-grained classification based solutions make no difference to transfers within the same category and may result in poor performance on elastic traffic

due to missing the deadlines. Tempus [9] and Amoeba [10] try to provide mechanisms to promise the deadlines. However, both Tempus and Amoeba are solving the allocation problem by using Linear Programming (LP), the size of which is typically enormous. Tempus only maximizes the minimum fraction of transfer tasks before deadline, rather than completing. Amoeba arranges transfers with shorter time to achieve a better bandwidth utilization, but fairness is ignored. Furthermore, it does not distinguish the transfers with similar volume and deadlines, though they have different impacts on disparate tenants.

To address this problem, we extend Deadline-based Network Abstraction [10] into a Completion Time-based Model which enables tenants to express how much the completion time affects their services precisely. A utility function is added to the Completion Time-based Model, which is a monotone decreasing function with the completion time to indicates the benefit of completing them or the impacts of missing them on different time, and the completion time depends on how bandwidth resources are allocated. The Completion Time-based Model provides a more detailed perspective for the system to measure the effect of allocating bandwidth resources to different transfer tasks over time.

In this paper, we propose a novel mechanism for online bandwidth allocation based on Completion Time-based Model. The system calculates the probability of disparate completion time and the predicted utility, according to the declared utility values of different completion time for various transfer tasks. It can efficiently improve the flexibility and scalability of allocation and reduce the computation of LP which is used in Tempus [9] and Amoeba [10]. Also, our solution takes fairness into consideration. Greater fairness means more similar flows transferred in parallel. We introduce the fairness coefficient, which provides a flexible way for network administrators to express the trade-off between the performance (total utility) and fairness. We develop an inter-DC WAN emulation tool which can emulate arbitrary topology within one commodity server. It provides the ability to customize WAN topology and dynamically configure bandwidth of WAN links.

Our evaluation shows that our algorithm gains 10%-150% more total utility than current deadline focus mechanism while producing 12% higher throughput. Furthermore, by adjusting the fairness coefficient, our algorithm improves the fairness over four times at best.

## II. BACKGROUND AND MOTIVATION

### A. DEADLINE

Deadline is crucial for inter-DC transfers. The primary reason is that the available capacity is far from meeting the total demand for inter-DC transfers. Flows generated by online services are used for geo-replication to improve performance and reliability. Therefore, providers of data centers offer different network service by setting different Service Level Agreements (SLAs) to tenants based on the price or the delay tolerance.

Remote storage access for computation over inherently distributed data sources is the most common usage in the elastic traffic [6]. For example, a distributed computation application consumes data from the workers across the DCs. If one of the workers fails to transfer, the application has to wait for it while other workers are idle.

The key point of the elastic traffic is that the utility of the transfer reduces depending on how late it finishes. Accordingly, an attribution should be attached to each flow, which indicates the profit of completing it at certain times based on its deadlines.

However, most of the current solutions, such as rate limiting and Traffic Engineering (TE) [4], [6], [11], are unsatisfactory in handling the deadline-oriented inter-DC transfers.

### B. INTRA-CLASS PRIORITY

The main reason for the poor performance on the deadline-oriented inter-DC transfer is lacking an efficient priority mechanism within the same category.

Tenants in data centers run various businesses by deploying different private services distributed over different data centers. Inter-data center flows generated by these private services may have the same traffic characteristics but different effects on each tenant, which makes defining priority tricky.

What makes defining priority more difficult is that the priority of a transfer dynamically changes along the time. The priority of a transfer task should increase higher as the time lapses because it is closer to the deadline. If the transfer task misses its soft deadline, the priority drops, since its next deadline is far from now.

### C. DNA AND AMOEBA

Amoeba [10] tries to define priority based on deadline by proposing Deadline-based Network Abstract (DNA) to represent transfer requests. However, DNA is insufficient to present impacts to tenants, which should be taken into account by the priority setting mechanism.

DNA is proposed to allow tenants to specify deadlines for transfers. DNA defines a standard interface for interactions between the central controller and tenants to state tenants' deadline requirements for each flow. A transfer $T$ is specified as a tuple which contains the source and destination, data volume, start time and deadlines. Transfers are grouped into one request which is atomically accommodated by Amoeba. Amoeba performs admission process on a first-come-first-served (FCFS) basis. It tries to reschedule earlier admitted requests to make room for new request arrives at a later time. Besides, Amoeba handles mispredictions by setting aside headrooms for further time slots.

The problem of grouping transfers is ignoring different priorities of the transfers. Tenants have to divide a number of transfers into several requests to distinguish the transfers with different priorities, which introduces extra work on tenants side.

### D. OUR GOAL

Our goal is archiving maximum utility while accommodating more transfer requests before their deadlines, achieving higher bandwidth utilization and arranging their transfers more fairly.

In this paper, we focus on *elastic* traffic optimization, since *interactive* traffic always meets its requirements for its high priority and *background* traffic has no specific deadline.

Inter-data centers from various providers may have different multi-path transportation implementations. For this reason, multi-path transportation optimization is beyond the scope of this paper.

## III. COMPLETION TIME-BASED FLOW SCHEDULING

Equivalent problems, e.g., variants of the packing-covering issue, have been proposed and solved by LP [12] [13] [14]. However, the size of LP can be extraordinary huge when large amounts of transfer tasks are involved [9]. Our solution is based on the greedy algorithm which can naturally reduce the time complexity with asymptotically optimal approximations.

In this section, we first introduce a Completion Time-based Model to have a further detail description of tenant transfer tasks. Then, we provide a solution to a simplified problem, end-to-end flow scheduling, which schedules all transfer tasks from the same pair of source and destination. Additionally, a complete solution to the overall flow scheduling optimization is proposed based on the algorithm in end-to-end flow scheduling. Finally, we introduce the fairness coefficient to improve the parallel of transportation with little total utility loss.

### A. COMPLETION TIME-BASED MODEL

Transfer tasks are generated by tenants in data centers delivering data from one data center to another for various purposes. A transfer task (flow) in the Completion Time-based Model is defined as a tuple $\{src, dst, Q, ts, u\}$, where *src* and *dst* are the source and destination node, $Q$ is the data volume of this transfer, $ts$ is the starting time, and $u$ is the utility function.

The utility function in the Completion Time-based Model describes the profit on transfer completion. Different transfer durations result in different benefits. It should reduce to zero eventually. The slope of the reduction, steep at first, plummets near to zero and then changes moderately over time. A typical utility function of a transfer task is shown in Figure 1.

### B. END-TO-END FLOW SCHEDULING

In end-to-end scheduling, flows share the same *src* and *dst*. Having all available bandwidth to these flows, the controller can allocate bandwidth resource iteratively for each time slot in a greedy manner.

Given k-shortest paths $P$ and a set of $n$ flows $F$, the goal of our algorithm is to maximize the total utility of flows under the restriction of bandwidth,

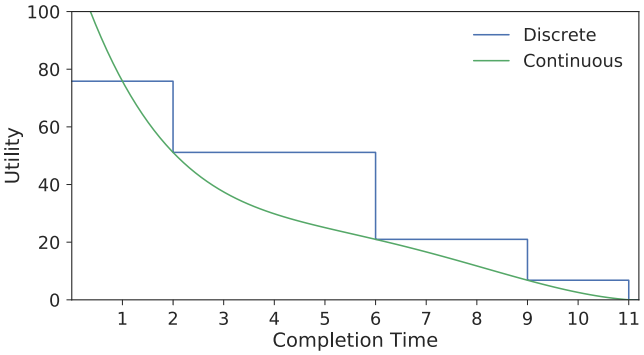



**FIGURE 1.** Two different types of utility function.

$$
\begin{aligned}
&\textit{maximize } T \triangleq \textstyle\sum_i U_i, \\
&s.t.\ \forall i, t, p, \quad A_{i,t,p} \geqslant 0 \\
&\qquad \forall i \in [1, n], \quad \textstyle\sum_t \sum_p A_{i,t,p} = Q_i \\
&\qquad \forall t, p, \qquad \textstyle\sum_i A_{i,t,p} \leqslant B_{t,p}
\end{aligned}
$$

where $U_i$ is the utility of flow $f_i$, $A_{i,t,p}$ is the allocated volume of flow $f_i$ at time $t$ over path $p$, $Q_i$ is the total data volume of flow $f_i$ and $B_i$ is the available bandwidth of path $p$ at time $t$.

To further reduce computation load, we divide time into time slots and data volume into identical size data blocks. With a smaller size of a data block, the system can allocate bandwidth resource with finer granularity. In consequence, a time series block queue should be built to allocate bandwidth at first. In this way, bandwidth restriction equations in LP turn into an urn problem [15], [16].

#### 1) TIME SERIES BLOCK QUEUE

Given a graph $G$ of network topology, k-shortest paths $\{p_1, p_2, \ldots, p_k\}$ from *src* to *dst* can be measured by assigning each link a cost value. The available bandwidth of k-shortest paths at each time slot can be evaluated, as shown in Figure 2(a). Consequently, the overall available bandwidth (blocks) can be gathered, which are ordered from the shortest path to the longest, as shown in Figure 2(b). Then, a time series data block queue $Q$ can be built depending on overall bandwidth in time order, as shown in Figure 2(c).

The block $b_i$ in Time Series Block Queue is specified by a tuple $\{p, t\}$, where $i$ is the index, $p$ is the path, $t$ is the time slot. For instance, $b_5$ is specified as below,

$$b_5 : \{p_5 : 2, t_5 : I\}$$

#### 2) END-TO-END BLOCK ALLOCATION

In this step, blocks will be assigned to separate flows iteratively until all flows are satisfied or missed their hard deadlines, see Algorithm 1.

Our object is to find the max total predicted utility $\overline{T}_i(x)$ if assigning block $b_x$ to flow $f_i$,

$$\underset{i}{\textit{maximize}}\ \overline{T}_i(x) \triangleq \textstyle\sum_k \overline{U}_{x,k|i}$$

**TABLE 1. Notations used in algorithm description.**

| Notation | Definition |
|---|---|
| $\overline{U}_{x,j\|i}$ | The predicted utility of flow $f_j$ if assigning block $b_x$ to flow $f_i$ |
| $\overline{U}_{x,j\|\bar{j}}$ | The predicted utility of flow $f_j$ if **NOT** assigning block $b_x$ to flow $f_j$ |
| $p_{x\leftarrow i}(y)$ | The probability of completing flow $f_i$ at block $b_y$ if assigning block $b_x$ to flow $f_i$ |
| $p_{x\leftarrow\bar{i}}(y)$ | The probability of completing flow $f_i$ at block $b_y$ if **NOT** assigning block $b_x$ to flow $f_i$ |
| $u_i : timeslot \rightarrow utility$ | The utility function $u_i$ of flow $f_i$ |
| $r_i$ | The number of unallocated blocks of flow $f_i$ |
| $t_x$ | The time slot of block $b_x$ |
| $Q_i$ | The data volume of flow $f_i$ |
| $N_x$ | The number of total unallocated blocks at block $b_x$ |
| $P_{x\leftarrow i}(t)$ | The probability of completing flow $f_i$ before time $t$ if assigning block $b_x$ to flow $f_i$ |
| $P_{x\leftarrow\bar{i}}(t)$ | The probability of completing flow $f_i$ before time $t$ if assigning block $b_x$ to flow $f_i$ |

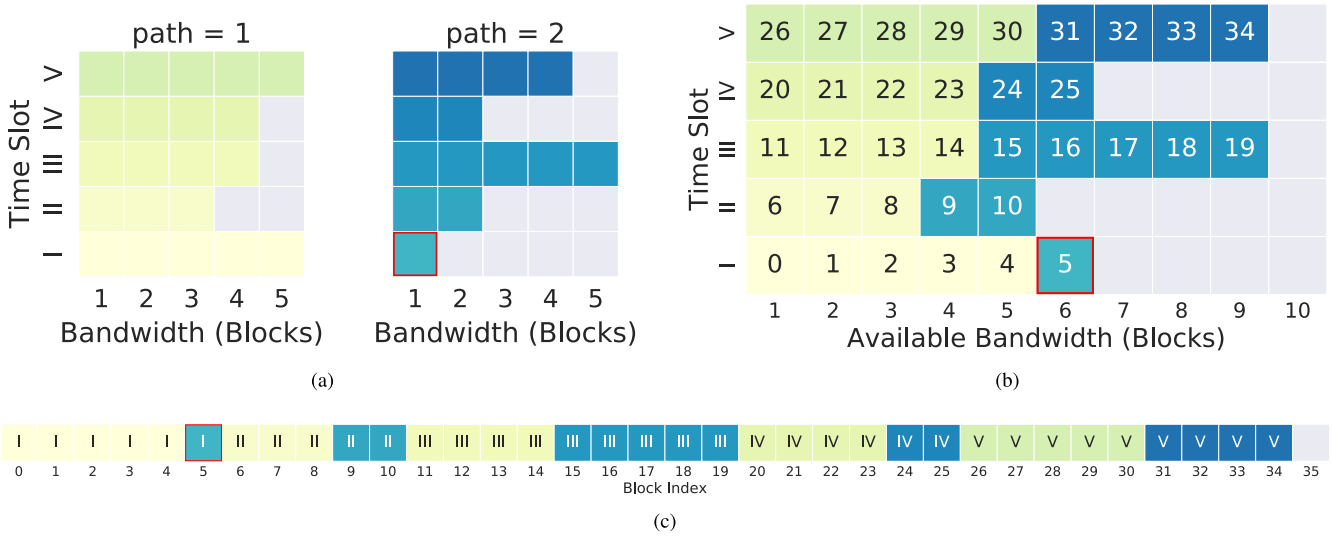



**FIGURE 2. Illustration of building Time Series Block Queue. Time slots are presented by Rome numerals, while blocks are indexed by Arabic numerals. In (a) and (b), the overall bandwidth can be accumulated by those in the k-shortest path ($k = 2$). In (c), the controller can build a time series data block queue according to the overall bandwidth. Moreover, blocks in each time slot are ordered from the shortest path to the longest. For example, block $b_5$ is in Time Slot *I*, from Path 2 and rounded by red rectangles. (a) Available Bandwidth of 2-Shortest Paths. (b) Total Available Bandwidth at each time slot. (c) Time Series Block Queue.**

Especially, the predicted utility is all the same as long as it does not get the assignment of the block.

$$\overline{U}_{x,k|i} = \overline{U}_{x,k|\bar{k}}, \quad \forall i \neq k$$

Suppose max total predicted utility occurs by assigning block $b_x$ to flow $f_i$, then for any $j \neq i$

$$\begin{aligned}
\overline{T}_{i,x} &\geqslant \overline{T}_{j,x} \\
\sum_k \overline{U}_{x,k|i} &\geqslant \sum_k \overline{U}_{x,k|j} \\
&\quad \times \overline{U}_{x,i|i} + \overline{U}_{x,j|i} + \sum_{k\neq i,j} \overline{U}_{x,k|i} \\
&\geqslant \overline{U}_{x,i|j} + \overline{U}_{x,j|j} + \sum_{k\neq i,j} \overline{U}_{x,k|j} \\
&\quad \times \overline{U}_{x,i|i} + \overline{U}_{x,j|\bar{j}} + \sum_{k\neq i,j} \overline{U}_{x,k|\bar{k}} \\
&\geqslant \overline{U}_{x,i|\bar{i}} + \overline{U}_{x,j|j} + \sum_{k\neq i,j} \overline{U}_{x,k|\bar{k}} \\
&\quad \times \overline{U}_{x,i|i} - \overline{U}_{x,i|\bar{i}} \\
&\geqslant \overline{U}_{x,j|j} - \overline{U}_{x,j|\bar{j}}
\end{aligned}$$

Consequently, the objective function is simplified as

$$\underset{i}{maximize}\ \overline{T}_i^*(x) \triangleq \overline{U}_{x,i|i} - \overline{U}_{x,i|\bar{i}}$$

Assigning the block $b_x$ assigns to flow $f_i$ means that if assigning the block to other flows will lead to a lower value of the total predicted utility. Then, we name $\overline{T}_i^*(x)$ as *Utility Contribution*.

Additionally, $\overline{U}_{x,i|i}$ and $\overline{U}_{x,i|\bar{i}}$ are determined as,

$$\overline{U}_{x,i|i} = \begin{cases} \dfrac{r-1}{r} \displaystyle\sum_{y=x+1}^{\infty} p_{x\leftarrow i}(y) \cdot u_i(t_y) \\ \qquad + \dfrac{1}{r} \cdot u_i(t_{x+r-1}) & r_i > 1 \\ u_i(t_x) & r_i = 1, \end{cases}$$

$$\overline{U}_{x,i|\bar{i}} = \sum_{y=x+1}^{\infty} p_{x\leftarrow\bar{i}}(y) \cdot u_i(t_y),$$

As mentioned earlier, allocation problem can be solved as an urn problem, more precisely, hypergeometric distribution problem.

**Algorithm 1** End-to-End Block Allocation

**E2EBandwidthAllocation** $(Q, F)$
  **inputs**: Time Series Block Queue $Q = \{b_0, b_1, \dots\}$; Set of flows to be allocated $F = \{f_1, \dots, f_n\}$;
  **output**: Allocated Time Series Block Queue $Q'$
  $Q' \leftarrow \{\}$;
  $F' \leftarrow \{\}$ ; `// Processing flow set`
  **foreach** $b_x \in Q$ **do**
    $F' \leftarrow F' \cup \{flows \text{ start at } b_x\}$;
    $F' \leftarrow F' \setminus \{flows \text{ meet hard deadline}\}$;
    $max \leftarrow 0$;
    $alloc \leftarrow 0$;
    **foreach** $f_i' \in F'$ **do**
      $\overline{T}_i(x) \leftarrow \sum_k \overline{U}_{x,k|i}$;
      **if** $\overline{T}_i(x) \geqslant max$ **then**
        $max \leftarrow \overline{T}_i(x)$;
        $alloc \leftarrow i$;
      **end**
    **end**
    $Q'_x \leftarrow alloc$;
    $r_{alloc} \leftarrow r_{alloc} - 1$;
    **if** $r_{alloc} = 0$ **then**
      $F' \leftarrow F' \setminus \{f_{alloc}\}$
    **end**
  **end**
  **return** $Q'$;

Then, the probabilities are

$$p_{x\leftarrow i}(y) = \begin{cases} \frac{\binom{N_x - r_i}{(y-x)-(r_i-1)}}{\binom{N_x-1}{y-x}} \frac{r_i - 1}{y-x} & r_i - 1 \leqslant y - x < N_x \\ 0 & otherwise \end{cases},$$

$$p_{x\leftarrow \bar{i}}(y) = \begin{cases} \frac{\binom{(N_x-1)-r_i}{(y-x)-r_i}}{\binom{N_x-1}{y-x}} \frac{r_i}{y-x} & r_i \leqslant y - x < N_x \\ 0 & otherwise, \end{cases}$$

both of whose time complexity is $O(y)$. Accordingly, the time complexity of $\overline{T}_i'(x)$ is up to $O({N_x}^2)$.

$N_x$ is the amount of all unallocated blocks of the flows whose start time is no later than the time of block $b_x$ while the hard deadline is later than that. It can be calculated recursively,

$$N_x = \sum_i^{F'} r_i$$

$$= \begin{cases} max(0, n_{x-1} - 1) + Q_i \cdot I_{i,x} - Q_i \cdot I'_{i,x} & x > 0 \\ \sum_i Q_i \cdot I_{i,0} & x = 0 \end{cases}$$

$I_{i,x}$ : 1 if flow $f_i$ starts at block$b_x$ and 0 otherwise ,

$I'_{i,x}$ : 1 if flow $f_i$ meets hard deadline at block$b_x$ and 0 otherwise.

Additionally, for all $x + r_i < y < x + N_x$,

$$\frac{p_{x\leftarrow i}(y)}{p_{x\leftarrow i}(y-1)} = \frac{(y-x)-1}{(y-x)-r_i+1},$$
$$\frac{p_{x\leftarrow \bar{i}}(y)}{p_{x\leftarrow \bar{i}}(y-1)} = \frac{(y-x)-1}{(y-x)-r_i}$$

Then, the time complexity of each iteration step decreases to $O(1)$, and that of $\overline{T}_i^*(x)$ falls to $O(N_x)$.

Furthermore, for most of the flows which have discrete utility functions, the block indexes of soft deadlines are presented as $\{y_1, \dots, y_n\}$. Then, $\overline{U}_{x,i|i}$ and $\overline{U}_{x,i|\bar{i}}$ are determined as,

$$\overline{U}_{x,i|i} = \begin{cases} \frac{r-1}{r} \sum_{m=1}^{n} \left( P_{x\leftarrow i}(y_m) - P_{x\leftarrow i}(y_{m-1}) \right) \\ \quad \cdot u_i(y_m) + \frac{1}{r} \cdot u_i(y_{x+r-1}) & r_i > 1 \\ u_i(t_x) & r_i = 1, \end{cases}$$

$$\overline{U}_{x,i|\bar{i}} = \sum_{m=1}^{n} \left( P_{x\leftarrow \bar{i}}(y_m) - P_{x\leftarrow \bar{i}}(y_{m-1}) \right) \cdot u_i(y_m),$$

and the possibilities

$$P_{x\leftarrow i}(y_m) = \begin{cases} 0 & m = 0 \text{ or } y_m < x + r_i \\ \frac{\binom{N_x - r_i}{(y_m - x)-(r_i-1)}}{\binom{N_x-1}{y_m-x}} & otherwise \end{cases},$$

$$P_{x\leftarrow \bar{i}}(y_m) = \begin{cases} 0 & o = 0 \text{ or } y_m < x + r_i + 1 \\ \frac{\binom{N_x-1-r_i}{(y_m-x)-r_i}}{\binom{N_x-1}{y_m-x}} & otherwise \end{cases}$$

If $b_{x-1}$ assigns to $f_i$,

$$P_{x\leftarrow i}(y_m) = \frac{N_x}{y_m - x + 1} P_{x-1\leftarrow i}(y_m),$$
$$P_{x\leftarrow \bar{i}}(y_m) = \frac{N_x}{y_m - x + 1} P_{x-1\leftarrow \bar{i}}(y_m).$$

Otherwise,

$$P_{x\leftarrow i}(y_m) = \frac{N_x}{N_x - r_i + 1} \frac{y_m - x - r_i + 2}{y_m - x + 1} P_{x-1\leftarrow i}(y_m),$$
$$P_{x\leftarrow \bar{i}}(y_m) = \frac{N_x}{N_x - r_i} \frac{y_m - x - r_i + 1}{y_m - x + 1} P_{x-1\leftarrow \bar{i}}(y_m).$$

Then, the time complexity of $\overline{T}_i^*(x)$ reduces to $O(m)$, which is much smaller than $O(N_x)$.

According to the algorithm described above, we will have a view of how many data blocks of each flow will be transferred in each time slot.

### C. OVERALL FLOW SCHEDULING

Different from end-to-end flow scheduling, bandwidth resources are not exclusive to the flows sharing same *src* and *dst* nodes. A path of k-shortest paths in one end-to-end flow group may overlap with the path in other end-to-end flow groups.

Considering a simple example, given a topology as shown in Figure 3, assume that capacity of link $AB$ and $BC$ are same.

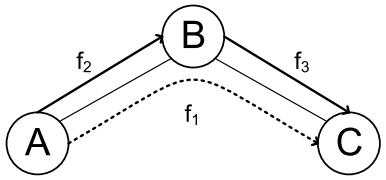



**FIGURE 3.** Example topology.

The data volume of $f_1, f_2$ and $f_3$ are equal, and the utility of $f_1$ is greater than that of both $f_2$ and $f_3$ but less than the sum of $f_2$ and $f_3$.

$$\begin{aligned} c_{AB} &= c_{BC}, \\ F &= \{f_1, f_2, f_3\}, \\ Q_1 &= Q_2 = Q_3, \\ U_2, \quad U_3 &\leqslant U_1 \leqslant U_2 + U_3 \end{aligned}$$

If merely comparing utilities of each flow, the system should arrange $f_1$ to transfer rather than $f_2$ and $f_3$. However, transferring one block of $f_1$ takes up two blocks in total, one block on each link. Apparently, transferring $f_2$ and $f_3$ in parallel is a better choice.

In order to operate an overall optimized flow scheduling, we introduce *Per-cost Utility Contribution* which can reveal the relationship between utility and cost.

### 1) PER-COST UTILITY CONTRIBUTION

Transferring a block along diverse paths will contribute different utility while having distinct costs. The per-cost utility contribution indicates the "earning rate" of utility that generated by transferring a specified block.

Per-cost utility contribution $PUC_i$ defines as below,

$$PUC_i(x) = \frac{\overline{T}_i^*(x)}{c(p_x)}$$

where $p_x$ is the path the block $b_x$ belongs to and $c(p_x)$ is the cost of transferring a block along the path $p_x$.

It can efficiently avoid the case that multiple short transfers with less utility cannot beat a single long transfer with greater utility, while the sum of short transfers' utilities is greater than that of the long one. Moreover, it can achieve the goal of contributing maximum utility while having the same cost.

### 2) OVERALL BLOCK ALLOCATION

Before allocation, the system classifies flows into end-to-end groups.

Consequently, the system collects network information to build Time Series Block Queue for each end-to-end flow group. Note that available bandwidth (blocks) are not exclusive to the flow group. They are shared by different groups due to sharing links in k-shortest paths. For this reason, blocks may be removed from the queue during the allocation. Further details will be discussed in the following section.

Additionally, the system maintains several states for each flow group during the allocation, including the number of unallocated blocks $N$, allocation index $x$, processing flow set $F'$ and allocated Block Queue $Q'$.

$$\begin{aligned} FG &= \{fg_1, \ldots, fg_n\}, \\ fg &= \{F, Q, N, x\} \end{aligned}$$

The detail of overall block allocation algorithm is described in Algorithm 2.

**Algorithm 2** Overall Bandwidth Allocation

**OverallBandwidthAllocation** ($FG$)
  **inputs**: End-to-End Flow Groups
    $FG = \{fg_1, \ldots, fg_n\}$
  **output**: Allocated Time Series Block Queue Groups
    $Q'G$
  $ts \leftarrow 0$;
  **while** $\exists fg \in FG \quad N! = 0$ **do**
    /* Loop until all flows are satisfied */
    update all $F', N, ksp$;
    **while** $\exists fg \in FG \quad t_x = ts$ **do**
      /* Loop until all blocks in current time slot are assigned */
      **foreach** $fg \in FG$ **do**
        $PUC_{fg} \leftarrow \frac{max\overline{T}_i(x_{fg})}{w(p_{x_{fg}})}$;
        $flow_{fg} \leftarrow i$;
      **end**
      $allocfg \leftarrow max(PUC_1, \ldots, PUC_n)$;
      $Q'_{allocfg} \leftarrow flow_{allocfg}$;
      $bw_{p_{x_{allocfg}}} \leftarrow bw_{p_{x_{allocfg}}} - 1$;
      $N_{allocfg} \leftarrow N_{allocfg} - 1$;
      $FG \leftarrow$ **UpdateFlowGroup**$(FG, p_{x_{allocfg}})$;
      **foreach** $l \in p_{xallocfg}$ **do**
        $c_l \leftarrow c_l - 1$;
      **end**
      $x_{allocfg} \leftarrow x_{allocfg} + 1$;
    **end**
    $ts \leftarrow ts + 1$;
  **end**
**return** $\{Q'_1, \ldots, Q'_n\}$

The objective of the overall block allocation is to find max per-cost utility contribution for each iteration.

For each flow group, flows are competing for a single block in each iteration. So, the cost of allocation current block in this group is the same to these flows. Consequently, the greatest *Utility Contribution* makes the greatest *Per-cost Utility Contribution*. Then, for each end-to-end flow groups, the objective function remains all the same.

Accordingly, the system will assign a block to the flow which has the highest per-cost utility contribution and updates states to fix the problem that the allocated block is occupied in different flow groups. Note that all these changes take effect within the current time slot.

For the flow group *allocfg* that got allocated, the number of unallocated blocks $N$ minus 1 and the allocate index $x$ moves forward.

For other flow groups, the system will update these flow groups to rebuild Time Series Block Queue $Q$ dynamically.

Besides, the path of the assigned block consumes a block on each block. Therefore, the available capacity $c_l$ of links along the path minus 1.

#### 3) FLOW GROUP UPDATE

As we have mentioned above, the consumed blocks along the path of the allocated block need to be removed out of queue as shown in Algorithm 3.

**Algorithm 3** Flow Group Update

**UpdateFlowGroup** ($FG$, $path$)
- **inputs**: End-to-End Flow Groups $FG = \{fg_1, \ldots, fg_n\}$; The path of allocated block $path$
- **output**: Updated Flow Groups $FG$
- **foreach** $fg \in FG$ **do**
  - $procLink \leftarrow link_{path} \cap \{l | l \in ksp, \sum bw_p = c_l\}$;
  - **while** $procLink! = \emptyset$ **do**
    - **foreach** $p \in ksp_{fg}$ *and* $bw_p > 0$ **do**
      - $ints_p \leftarrow procLink \cap link_p$;
    - **end**
    - $rmPath \leftarrow path$ which has the max $|ints|$;
    - $rmLink \leftarrow ints_{rmPath}$;
    - $bw_{rmPath} \leftarrow bw_{rmPath} - 1$;
    - $procLink \leftarrow procLink \setminus rmLink$;
  - **end**
  - rebuild $Q_{fg}$;
- **end**

**return** $FG$

The available bandwidth of k-shortest path is exclusive in each flow group, and the sum of available bandwidth $bw_p$ of a link $l$ shared by k-shortest paths $ksp$ in one end-to-end flow group cannot be greater than the available capacity $c_l$ of this link.

$$\forall l \in ksp, \quad \sum_{p}^{p \in ksp, l \in p} bw_p \leqslant c_l$$

In the case that $\sum bw_p < c_l$, the system regards that the block consumed is not occupied in this flow group. Then the system will ignore the link $l$ until they are equal.

In the case that $\sum bw_p = c_l$, the system will remove a block from a path which contains the link $l$. As a result, the system should only process links $procLink$ in the allocated path whose capacity equals the sum of available bandwidth.

Moreover, multiple paths may contain the link to be processed. The object of the system is to remove blocks as few as possible. Accordingly, the block from the path which contains the most processed links $ints_{path}$ should be removed first. Then the contained links $ints_{path}$ are removed from $procLink$. Repeat the above steps until $procLink$ is empty.

In the end, rebuild the Time Series Block Queue $Q$ and the flow group has been updated.

### D. FAIRNESS COEFFICIENT

According to the object functions, the bandwidth resources are allocated to separate transfer tasks to obtain a maximum utility.

Our algorithm prefers to allocate blocks to the same flow in a row to obtain the utility as soon as possible due to a greedy manner, even though it is far from the deadline and flows can be transferred in parallel without utility loss. This allocation strategy can effectively obtain utility with the high risk of missing deadline, e.g., link failure, since the utility is gained only at the time of completing transportation.

Nevertheless, exclusive transportation introduces extra cost on control plane to synchronize flow transportations, while the total utility may not be visibly increased when the risk of missing deadline is low because the system does not gain more utility no matter how early the transfer task completes before the deadline.

For this reason, in order to improve the fairness of transportation, as well as, parallelism of transportation, we introduce the fairness coefficient $fair(k)$ , which is defined as below,

$$fair(k) = \left(1 - \sum_{y=0}^{x-1} \left(\frac{k}{2}\right)^{x-y} \cdot Q'_y(i)\right)^{\frac{r}{2}},$$

$Q'_y(i)$ : 1if allocate block $b_y$ to flow $f_i$ and 0 otherwise.

$k$ is an argument ranges from 0 to 1. The system will allocate blocks more equitably with a higher value of $k$. For the case of $k = 0$, the system will allocate blocks strictly according to the D-value of predicted utilities.

Correspondingly, the objective functions turns into

$$\underset{i}{maximize}\ \overline{T}^*_i(x) \cdot fair(k)$$

A higher penalty is received on the value of $T^*$ of a flow if it has been allocated recently, which makes its completion time delayed and less valued flows have a chance to be transferred. Accordingly, flows with approximate values of $T^*$ will be arranged to be transferred in parallel, rather than one by one exclusively. In other words, fairness coefficient improves the fairness by delaying of completing flows, which may lead to missing their deadlines, either soft or hard, and a loss of total utility during the growth of $k$ value.

Besides, we define *allocation distance* is used as a metric to measure the fairness of our algorithm. *Allocation distance* is the distance between the locations of adjacent blocks from the same flow, where they are allocated in the queue. Correspondingly, a greater value of average allocation distance means more flows are transferring in parallel.

In summary, by setting a proper value of $k$ of the fair coefficient, the system can arrange bandwidth scheduling more fairly while keeps the loss of total utility to a minimum.

## IV. SYSTEM IMPLEMENTATION

In order to evaluate our algorithm, we develop a prototype system that implements all functions used in our algorithm. Additional, for the purpose of flexibly emulating arbitrary topology and dynamically reconfiguring the WAN link attributes, we develop an Inter-DC WAN emulation tool.

In this section, we describe details of the prototype architecture, rate limiting, failure handling and the emulation tool.

### A. ARCHITECTURE

The architecture of our prototype is shown in Figure 4. On the top of the architecture, a *central controller* implements all functions used in our allocation algorithm, including topology monitoring, bandwidth prediction, queue building and utility computation.

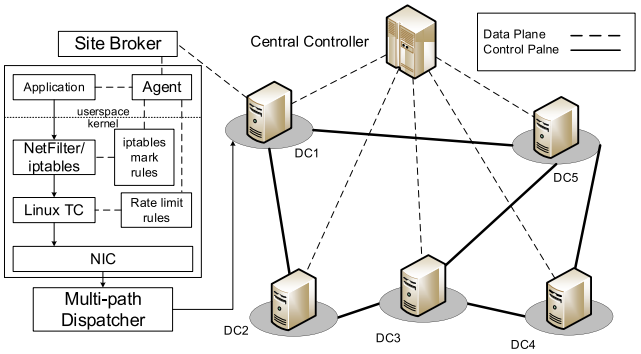



**FIGURE 4.** Prototype architecture & experiment topology.

On site level, *site broker* and *multi-path dispatcher* are deployed within each data center. *Site broker* exchanges information between central controller and agents deployed in virtual machines, including transfer requests and allocation result. *Multi-path dispatchers* take charge of multiple-path transportation according to the output of bandwidth allocation.

An *agent* installed in each virtual machine accepts transfer requests from user applications and sets up rules for rate limiting. Besides that, the *agent* periodically reports transferred data volume for each task to the *central controller* for further allocation.

### B. RATE LIMITING

Rate limiting is transparent to tenants' applications. Applications from tenants do not take control of sending rate. When packets enter into the kernel, rules of *iptables* set these packets with different marks to identify different flows. With *tc* tools, packets are split into different classes based on their marks. These classes are associated with distinct token buckets which have different sizes. As a result, flows are limited at specified rates.

### C. PER-TIME SLOT BANDWIDTH ALLOCATION & FAILURE HANDLING

Different from Amoeba, our algorithm allocates bandwidth for each time slot separately, rather than directly deciding whether accept or not when the system receives new requests as Amoeba does. This per-time slot allocation mechanism dramatically improves the flexibility and scalability of the system.

Since our allocation algorithm heavily relies on completion time prediction, especially when a high value of $k$ in fairness coefficient is set, bandwidth misprediction and link failure severely affect the performance of our allocation algorithm.

For bandwidth misprediction, the system sets aside headroom for future time slots. Furthermore, the system can dynamically estimate how much data volume left for each request from *agent*s to make a better allocation for next time slot.

Link failure means that the allocation has a high risk of missing deadline due to inadequate bandwidth resource than expected. To address this problem, a smaller value of $k$ should be set to push the completion time of each task as early as possible, based on the link quality.

### D. INTER-DC WAN EMULATION TOOL

In order to emulate WAN topology flexibly in our evaluation, we develop an inter-DC WAN emulation tool based on Mininet [17] in a virtual machine, as shown in Figure 5. For the purpose of emulating dynamic links, we create a *DynamicTCLink* class inherit from *TCLink*, which provides extra APIs to dynamically change the link properties such as delay, bandwidth, packet loss.

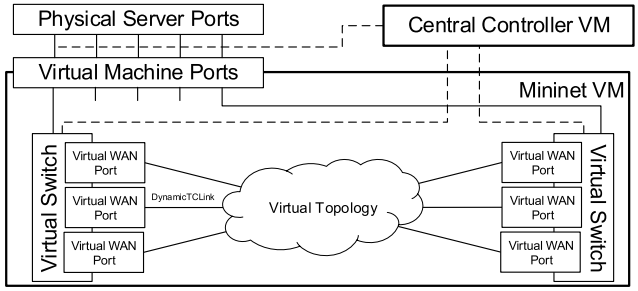



**FIGURE 5.** Inter-DC WAN topology emulation.

All physical ports, which connect to DC sites, are one-to-one mapped to emulation VM. These mapped ports are used to redirect traffic on data plane, which is tagged by multi-path dispatchers with VLAN IDs. While, traffic on the control plane, not tagged by any VLAN ID, redirects to central controller virtual machine directly.

In emulation VM, OVS switches are created as the gateways of DCs. Each switch is composed of a mapped port and several virtual WAN ports based-on the emulation topology. Virtual WAN ports are connected by *DynamicTCLink* to emulate links for inter-DC traffic. By doing that, the goal of emulating arbitrary topology is achieved.

The central controller installs rules on virtual switches to handle outgoing and incoming traffic. For outgoing traffic, it is filtered by VLAN ID to select the corresponding path to send traffic to correct destination. Meanwhile, incoming

traffic forwards directly to the mapped port since traffic can be routed to host without choosing the path.

According to our evaluation, our emulation tool is capable of handling traffic in our experiment topology with using 8% CPU.

## V. EVALUATION

### A. EXPERIMENT SETUP

#### 1) TESTBED SETUP

We create a small testbed with 6 servers to emulate an inter-DC WAN with 5 DCs, which topology is shown in Figure 6.

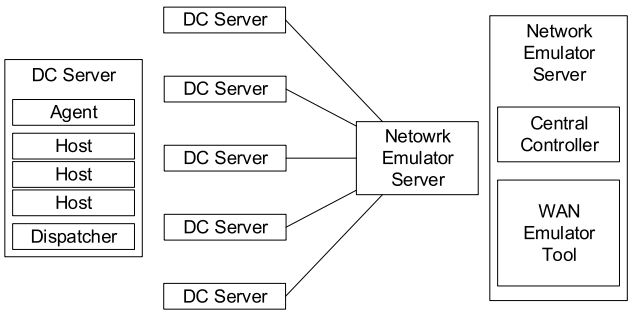



**FIGURE 6.** Physical topology of testbed.

Five servers act as DC sites. Within these servers, three virtual machines act as hosts in each DC site. Each DC server has a quad-core Intel E5-2609 2.40GHz CPU, 32GB memory, 500GB hard disk with four 1G Ethernet NICs. These severs directly connect to network emulator server with 1-4 physical 1G links on demand. The emulator sever runs the central controller and the Inter-DC WAN emulation tool. It has two E5-2609 CPU and 64GB memory, 500GB hard disk, and eight 1G Ethernet NICs. The CPU, memory or hard disk is not a bottleneck in our testbed evaluation. We use a modified iperf3 to generate TCP flows.

#### 2) EXPERIMENT SETUP

To emulate dynamically changing links, we use Inter-DC WAN emulation tool to emulate the virtual topology. The emulated topology is the topology shown in Figure 4 and all links have the same distance. Data center servers are connected logically by the virtual links, as described in Section IV(D).

We randomly generate from 5% to 15% of the link capacity as the interactive traffic on each link for each time slot. Based on such presupposition, we leave extra headroom as we discussed before. We perform experiments on our testbed for a duration of 50 3-minute time slots (2.5 hours).

We report the average of 5 runs.

#### 3) METRICS

We measure three main performance metrics: total utility, hard deadline accommodate (acceptance) rate and average allocation distance.

#### 4) WORKLOAD

The inter-DC transfer tasks are generated with the following parameters:

- Arrival time is modeled as a Poisson process with arrival rate $\lambda$ per time slot.
- Deadlines: The hard deadline is modeled under exponential distribution with a mean of one hour. We also generate 2 or 3 soft deadlines for each task, which follows a uniform distribution from 25% to 75% of the duration of the hard deadline.
- The data volume of each task should be related to its deadline. We expect the data volume is under exponential distribution with a mean of 100Mbps over the time of the hard deadline.
- Utility of a transfer task has a close correlation with data volume and deadlines. We calculate the utility as below:

$$U_t = Q * p * (\frac{hd}{t})^2, t \in \{sd_1, \ldots, hd\}$$

where $p$ is under exponential distribution with a mean of 1/Gb.

### B. TESTBED EVALUATION

We evaluate the performance of our algorithm for both end-to-end and overall flow scheduling.

For end-to-end evaluation, we select DC3 as the source node and DC5 as the destination node. 3-shortest paths are DC3-DC5, DC3-DC4-DC5 and DC3-DC2-DC1-DC5, which cover all links in the network. Besides, the total available bandwidth is 3Gbps.

For overall evaluation, we connect all 8 NIC interfaces from the emulation server to DC servers. The total throughput is 16Gbps. Besides, the cost of each links is modeled under Uniform Distribution from 1 to 10.

#### 1) END-TO-END FLOW SCHEDULING

##### a: TOTAL UTILITY

As shown in Figure 7(a), when $k = 0$, the total utility increases with the growth of $\lambda$, whereas, the growth of the total utility slows down as the $\lambda$ increases due to the link utilization reaches saturation gradually. Note that the total utility is not always increased as $\lambda$ rises, when a higher value of $k$ is set, e.g., 0.7, although the trend is upwards as the $\lambda$ increases. This is mainly because that our algorithm works in a greedy manner and our algorithm improves the fairness by delaying the completion time of allocated flows, which allocation may be not always the best choice from the view of a later time slot.

On the other side, the total utility tends to decline with the growth of $k$. Additionally, the rate of total utility decline from $k = 0$ to $k = 1$ ranges from 10% to 60%.

##### b: ACCEPTANCE RATE

According to Figure 7(b), the acceptance rate reduces with the growth of $\lambda$ by the reason that limited bandwidth can only accommodate limited transfer requests. For the same

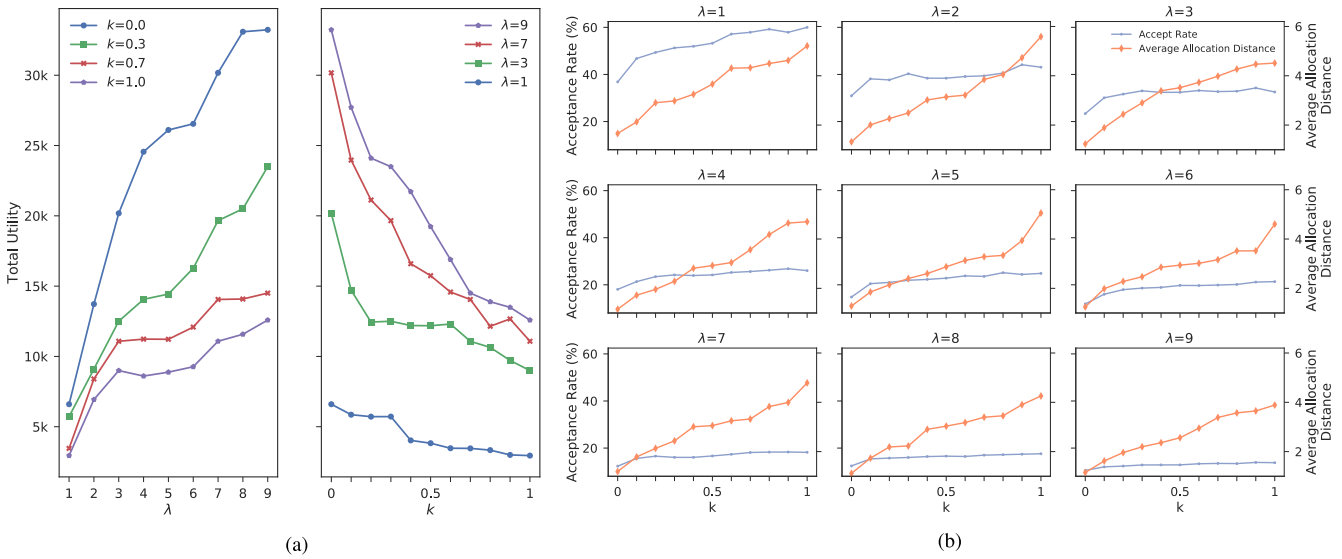



**FIGURE 7.** End-to-end Flow scheduling performance evaluation. (a) Total utility. (b) Acceptance rate and average allocation distance.

reason, the average allocation distance does not rise with the growth of λ.

Compared with the total utility and acceptance rate, higher acceptance rate does not lead to a higher total utility. On the contrary, for all λ, the minimum acceptance rate happens at $k = 0$ where achieves maximum total utility. In general, there is no correlation between utility and acceptance rate. Since the total utility is our first goal, the system prefers to arrange a more valuable flow to transfer rather than accepting many small flows that have less utility, which result in a lower acceptance rate.

### c: AVERAGE ALLOCATION DISTANCE

The average allocation distance is positively correlated with $k$. The system archives around 3.2-4.2 times of average allocation distance on the condition of $k = 1$ than that of $k = 0$.

On the other side, there is no specific link between the average allocation distance and λ. The average allocation distance is related to the available bandwidth. With the growth of λ, the number of accommodated flows is not obviously increased and extra arrived flows are dropped due to limited bandwidth. Accordingly, under the limited bandwidth resource, the average allocation distance will not have enormous change with λ.

### d: OVERALL FLOW SCHEDULING

We can find a similar result in Figure 8 for overall flow scheduling. The total utility increases with the growth of λ, whereas the acceptance rate drops. Notably, $k = 0.3$ is a proper value for the evaluation. For total utility, the system loses 7% on average when $k = 0.3$, while 25% when $k = 0.5$. For average allocation distance, it achieves 1.6 times on average when $k = 0.3$, while 1.8 times when $k = 0.5$.

The total utility declines up to about 50% when $k = 1$, which is better than that in the end-to-end evaluation. This is mainly because that the system gains more utility from other source/destination pairs. The system achieves around 2-3 times of average allocation distance with the increase of $k$, which is smaller than that in the end-to-end evaluation. This is because the allocated bandwidth for each source/destination in overall evaluation is smaller than that of end-to-end evaluation.

Then, we have a closer look at the accommodated flows. As shown in Figure 8(d), the number of accommodated flows increases with the growth of λ and then the trend of the improvement becomes flattened. This is because that the most valuable flows have been selected to be transferred under the restriction of limited bandwidth resource and it is unlikely to have flows with more valuable no matter how many more flows arrive.

Furthermore, we divide accommodated flows into three different classes {*short*, *mid*, *long*} based on the average hops of their transferred paths. As the stacked bar charts shown in Figure 8(d), the flows of *short* takes up to over half of accommodated flows, while the proportion of *long* is the least. This is because that, with the approximate utility contribution, a shorter path has a lower cost of transportation which leads to a greater value of per-cost utility contribution. The flow can be transferred along a long path, only when it has an extraordinary utility contribution.

Additionally, with the growth of $k$, the number of accommodated flows is increased around 30% on average with the growth of $k$. With the growth of $k$, more flows with less data volume and less per-cost utility contribution are accommodated. By taking fairness into account, the completion time of flows is delayed. Therefore, these flows are delayed and maybe not completed at the time we collect the result from

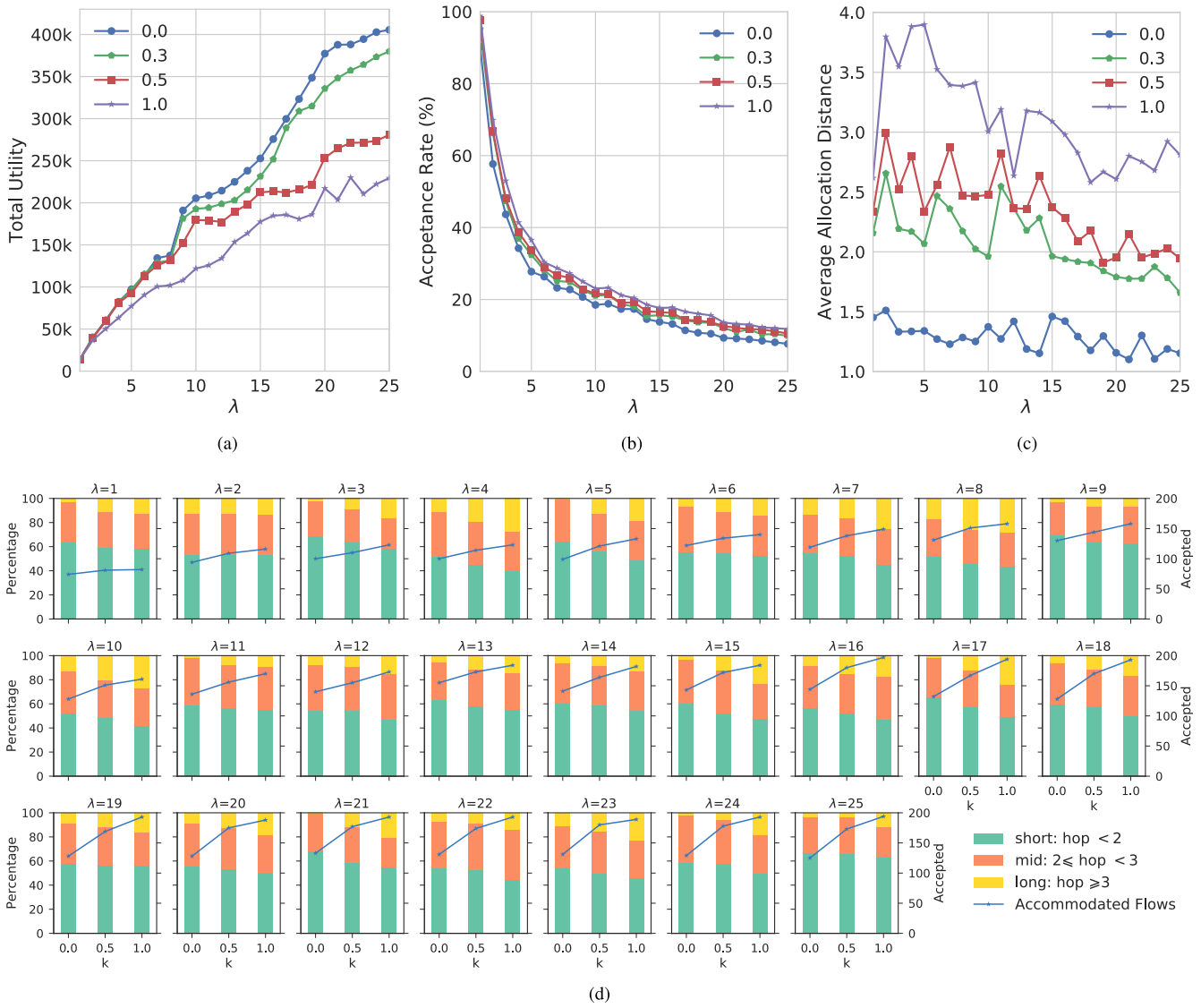



**FIGURE 8.** Overall flow scheduling performance evaluation. (a) Total utility. (b) Acceptance rate. (c) Average allocation distance. (d) Accepted flows.

the testbed, which should be completed when $k = 0$. Then the extra accommodated flows, when $k = 1$, are inserted into the blocks which are leased by delayed flows.

Besides, the proportion of the *short* declines 8% on average, while *long*'s increases 11% on average. This is mainly because that most of flows from *short* are accommodated when $k = 0$ and the number of unaccommodated flows from *long* is much more than other classes. In conclusion, the fairness coefficient improves not only the fairness within a single end-to-end flow group but also the fairness of different flow classes.

### C. FIXED MINIMUM BW GUARANTEE & AMOEBA

In this part, we compare our solution with fixed minimum bandwidth and Amoeba.

The fixed minimum bandwidth guarantee is set to satisfy the hard deadlines. Amoeba performs bandwidth scheduling based on the level of *request*, whereas our's on *transfer*. Therefore, we set the number of transfers per request to 1.

As shown in Figure 9(b), *Amoeba* accommodates most flows. When later arrived flows cannot meet their deadlines, Amoeba tries to reschedule on previously accepted flows to make room for these flows. *Fixed* accommodates less because the bandwidth will not be rescheduled if newly arrived flows cannot meet their deadlines. Later arrived flows can be accommodated only with less bandwidth. Additionally, both of *Amoeba* and *Fixed* prefer to accommodate flows with small data volume to achieve a higher acceptance rate with limited bandwidth. The acceptance rate of our solution is the least because that our solution is utility oriented which acceptance rate is ignored. In some cases, the system assigns a lot of bandwidth resource to a high data volume flow to complete before its soft deadline if it has high utility value, which leads

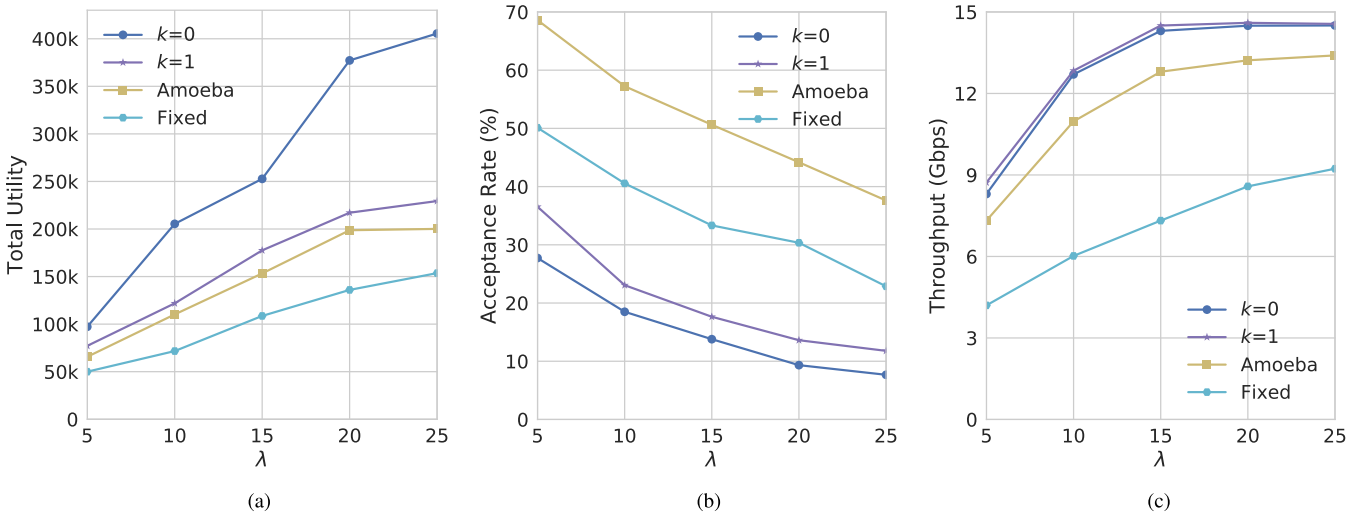



**FIGURE 9.** Our solution vs. amoeba & fixed minimum BW. (a) Total utility. (b) Acceptance rate. (c) Throughput.

to other flows with less data volume and utility miss their short hard deadlines.

Figure 9(a) shows the total utility obtains for each solution. *Amoeba* and *Fixed* gain less utility than our solution even when $k = 1$, although both of them accommodate more. If we set the total utility that our solution gains when $k = 1$ as the baseline, *Amoeba* loses 10% of total utility because the utility is not in a linear relationship with data volume and deadline. The per accommodated flow utility is much less than ours, although *Amoeba* runs OR to accommodate more flows. Due to the same reason, *Fixed* loses 37% of total utility on average. Additionally, accommodated flows of *Fixed* only meet their hard deadlines, which the system can only gain the least utility from them.

As shown in Figure 9(c), our solution achieves about 12% higher throughput than *Amoeba* and 80% than *Fixed*. This is mainly because that our solution does not reject flows when they arrive. It assigns available bandwidth to all candidate flows, which translate into higher total utility.

### D. FLEXIBILITY & SCALABILITY

To evaluate the flexibility of our solution, we measure the total utility under link failure by randomly failing one of the inter-DC links. We consider two cases: 100% link capacity loss and 50% link capacity loss. We calculate the remaining rate, which is the portion of the total utility we collect under different conditions of link failure over that without link failure.

Figure 10 shows the result. When λ is small, the total utility keeps almost the same, no matter the value of $k$. This is because that the system allocates bandwidth resource for each time slot separately. When link failure, the system can arrange flows, which should be transferred on the fail link, to transfer along with other paths where have surplus bandwidth. With the growth of λ, fewer data can be rearranged to transfer due to that the network is being saturated as shown in Figure 9(c).

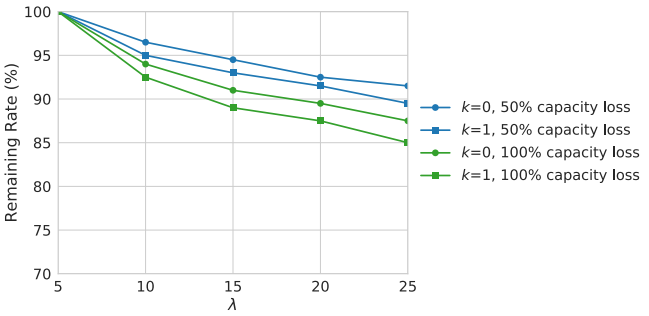



**FIGURE 10.** Performance under link failure.

As a result, more flows are failed to be accommodated, and lower total utility is gained by the system.

Additionally, $k$ affects the remaining rate as well. With the same condition of link capacity loss, the greater value of $k$ leads to a greater portion of loss on the total utility. This is caused by the same reason which causes greater $k$ results in less total utility without link failure. When a greater value of $k$ is set, the total utility declines because the completion time of flows is delayed. Additionally, with link failure, the completion time is further delayed which translates into lower utility. As a result, the total utility declines more.

Moreover, our solution achieves over 85% and 90% remaining rate respectively under high arrival rate.

We quantify the scalability by measuring the allocation time. As shown in Figure 11, the average allocation time increases accelerated with the growth of λ. This is because that the allocation time is mainly affected by the number of unserved flows, e.g., arrived but neither finished nor outdated.

The block size affects the allocation time as well. With a smaller size of a data block, the system can schedule bandwidth with finer granularity but takes a longer time to allocate. The system takes around double time when the size of data block is reduced from 1GB to 500MB. According to our practice of evaluation, the average allocation time should

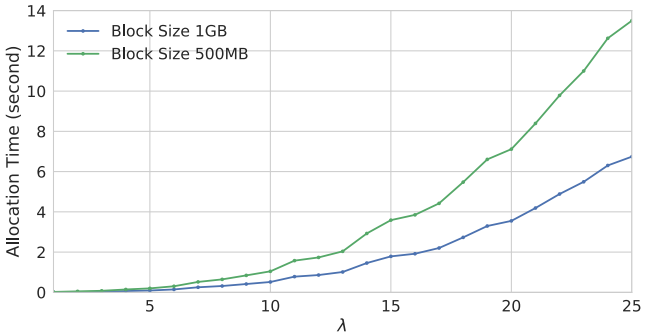



**FIGURE 11.** Average allocation time.

be around 1/20 of the duration of time slot to obtain a better balance between scalability and granularity.

## VI. CONCLUSION

A significant portion of Inter-DC transfer tasks have deadlines. Missing deadline may have various impacts on the whole system. However, current mechanisms mainly focus on deadline guarantee, rather than make a distinction between tasks to achieve an overall optimization on system performance. This paper introduces a completion time-based model which enables tenants to express how much missing the deadline of a transfer task impacts on their business, and a novel mechanism which efficiently allocates bandwidth resources to achieve a maximum utility in a flexible and scalable manner. Our evaluation shows that our solution effectively improves total utility than current deadline focus mechanisms while outperforming more throughput. Furthermore, by adjusting the fairness coefficient, our algorithm improves over 4 times fairness at best.

## REFERENCES

[1] "Cisco global cloud index: Forecast and methodology, 2014–2019," Cisco, San Jose, CA, USA, White Paper, Oct. 2015. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html

[2] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 68–73, Jan. 2008.

[3] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via Yahoo! Datasets," in *Proc. IEEE INFOCOM*, Apr.2011, pp. 1620–1628.

[4] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 15–26, Oct. 2013. [Online]. Available: http://doi.acm.org/10.1145/2534169.2486012

[5] M. Chen, Y. Qian, Y. Hao, Y. Li, and J. Song, "Data-driven computing and caching in 5G networks: Architecture and delay analysis," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 70–75, Feb. 2018.

[6] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, New York, NY, USA, 2013, pp. 3–14. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486019

[7] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "Research challenges for traffic engineering in software defined networks," *IEEE Netw.*, vol. 30, no. 3, pp. 52–58, May/Jun. 2016.

[8] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, to be published.

[9] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 515–526.

[10] H. Zhang *et al.*, "Guaranteeing deadlines for inter-datacenter transfers," in *Proc. 10th Eur. Conf. Comput. Syst. (EuroSys)*, 2015, Art. no. 20.

[11] A. Mahimkar *et al.*, "Bandwidth on demand for inter-data center communication," in *Proc. 10th ACM Workshop Hot Topics Netw.*, 2011, Art. no. 24.

[12] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto, "Optimal packing and covering in the plane are NP-complete," *Inf. Process. Lett.*, vol. 12, no. 3, pp. 133–137, 1981.

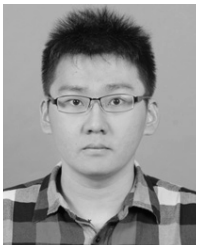
[13] S. A. Plotkin, D. B. Shmoys, and E. Tardos, "Fast approximation algorithms for fractional packing and covering problems," *Math. Oper. Res.*, vol. 20, no. 2, pp. 257–301, 1995.

[14] N. E. Young, "Sequential and parallel algorithms for mixed packing and covering," in *Proc. 42nd IEEE Symp. Found. Comput. Sci.*, Oct. 2001, pp. 538–546.

[15] P. Erdős, "On a classical problem of probability theory," *Magyar Tud. Akad. Mat. Kutató Int. Közl*, vol. 6, nos. 1–2, pp. 215–220, 1961.

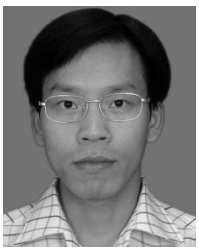
[16] B. Artur, Y. M. Ermol'ev, and Y. M. Kaniovskii, "A generalized URN problem and its applications," *Cybernetics*, vol. 19, no. 1, pp. 61–71, 1983.

[17] R. L. S. de Oliveira, A. A. Shinoda, C. M. Schweitzer, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Proc. IEEE Colombian Conf. Commun. Comput. (COLCOM)*, Jun. 2014, pp. 1–6.



**WENBO HU** received the B.S. degree in software engineering from the Dalian University of Technology, China, in 2011. He is currently pursuing the Ph.D. degree with the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications. His research interests include software-defined networking, future Internet architecture, and data center network.

**JIANG LIU** received the B.S. degree in electronics engineering from the Beijing Institute of Technology, Beijing, China, in 2005, the M.S. degree in communication and information system from Zhengzhou University, Zhengzhou, China, in 2009, and the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, in 2012. His current research interests include network virtualization, network architecture, and management.



**TAO HUANG** received the B.S. degree in communication engineering from Nankai University, Tianjin, China, in 2002, and the M.S. and Ph.D. degrees in communication and information system from the Beijing University of Posts and Telecommunications, Beijing, China, in 2004 and 2007, respectively. He is currently an Associate Professor with the Beijing University of Posts and Telecommunications. His current research interests include future Internet architecture, software-defined networking, and network virtualization.

**YUNJIE LIU** received the B.S. degree in technical physics from Peking University, Beijing, China, in 1968. He is currently an Academician of the China Academy of Engineering, the Chief of the Science and Technology Committee of China Unicom, and the Dean of the School of Information and Communication Engineering, Beijing University of Posts and Telecommunications. His current research interests include next-generation network, network architecture, and management.