Scheduling Inter-Datacenter Video Flows for Cost Efficiency

Haiying Shen*, *Senior Member, IEEE*, Chenxi Qiu

Abstract—As video streaming applications are deployed on the cloud, cloud providers are charged by ISPs for inter-datacenter transfers under the dominant percentile-based charging models. In order to minimize the payment costs, existing works aim to keep the traffic on each link under the charging volume. However, these methods cannot fully utilize each link's available bandwidth capacity. As a solution, we propose an economical and deadline-driven video flow scheduling system, called EcoFlow. Considering that different video flows have different transmission deadlines, EcoFlow transmits videos in the order of their deadline tightness and postpones the deliveries of later-deadline videos to later time slots. The flows that are expected to miss their deadlines are divided into subflows to be rerouted to other under-utilized links. We also propose setting each link's initial charging volume to reduce the scheduling latency at the beginning of the charging period and discuss how to deal with issues such as the prediction errors of link available bandwidth and the lack of charging volume's prior knowledge. Furthermore, we designed implementation strategies for using EcoFlow in both centralized and distributed situations. Experimental results demonstrate that EcoFlow achieves lower bandwidth costs and higher video flow transmission rates when compared to existing methods.

Index Terms—Video streaming; Bandwidth cost; Inter-datacenter traffic; Percentile-based charging models

1 INTRODUCTION

Cloud providers (e.g., Amazon) offer various pay-as-you-use cloud based services (e.g., Amazon Web Services) to cloud customers (e.g., Netflix) [1], [2]. The Cloud has proved to be an effective infrastructure to host video streaming services with many benefits [3]–[5]: First, cloud based infrastructures provide easy to use scalablity for hosting additional users and content by allowing cloud customers to buy additional resources from the cloud provider [6]. Second, distribution is similarly scalable and in this context is provided by a large number of globally distributed servers and streaming service providers (VSSPs) that allow for services to be provided in a vast array of different areas [7]. Lastly, cloud based infrastructures provide data-center management for their cloud customers, and as such are low-cost way for VSSPs to deliver their streaming services [8]–[10]. As a result, a progressively higher number of VSSPs, such as Netflix, are deploying theri web applications on the cloud.

In order to enhance service availability and scalability, cloud providers generally deploy a number of datacenters across different geographical regions, which are interconnected by high-capacity links leased from internet service providers (ISPs). Newly published videos and their replicas are allocated to these distributed datacenters to serve users from different regions. Specifically, when a new video is uploaded to a datacenter, the datacenter disseminates it to other datacenters to serve users. When the number of video replicas is insufficient to allow for a streaming service that is scalable, accessible, and widely available, more video replications occur between datacenters. Video dissemination and video replication leads to a substantial amount of inter-datacenter traffic [11]. It is important to note, however, that this traffic does not include datacenter-to-customer video traffic.

ISPs charge cloud providers for transit bandwidth using a wide array of pricing schemes. The most common of these, and the model adopted by most ISPs, is the 95th percentile charging model [12]. In this model the bandwidth cost is charged based on the 95th percentile value in all traffic volumes (data sizes) recorded in 5-minute intervals generated within a charging period (e.g., 1 month [12]). In the 95th percentile charging model, "Charging Volume" is defined as the volume of traffic from the beginning of the charging period up to the current time. Numerous previous studies focused on controlling new traffic volumes to ensure that they stayed below the charging volume [11]–[17] in order to minimize bandwidth payment costs on inter-datacenter video traffic to ISPs. These previous studies can primarily be classified into to groups: *store-and-forward* and *optimal routing path*.

The *store-and-forward* methods [13]–[15] take advantage of unique spatial and temporal attributes of inter-datacenter video traffic. Spatial attributes refer to datacenters in different geographic areas and their traffic loads and available bandwidth capacities, which are highly dependent upon the user demands within those different areas. Temporal attributes relate to the loads placed on a datacenter during a given time period, and more specifically regard the strong diurnal patterns that correlate with a given local time [18]. Store-and-forward methods predefine peak and off-peak hours for each datacenter based primarily on these two aspects - local time and geographic area - and subsequently utilize leftover traffic volume (charging volume minus actual traffic volume) during off-peak hours to transfer delay-tolerant data flows. A hypothetical datacenter, i, on the East Coast of the United States (EST/GMT-5), along with datacenters j, in Chile (EST+1/GMT-4), and k, in Belgium (EST+6/GMT+1), each with peak hours of 9-12pm local time and off-peak hours of 3-6am local time, will serve as an example. Suppose that a number of delay-tolerant data flows need to be sent from datacenter i to datacenter j, however, it happens to be 9pm EST meaning that both i and j are in peak hours. It can subsequently be inferred that no overlap exists between their respective peak and off-peak hours. That is, the peak hours of i do not align with the off-peak hours of j and that the reverse is also

* *Corresponding Author. Email: hs6ms@virginia.edu; Phone: (434) 924-8271; Fax: (434) 982-2214.*

- *Haiying Shen is with the Department of Computer Science, University of Virginia, Charlottesville, VA, 22904. E-mail: hs6ms@virginia.edu. Chenxi Qiu is with the College of Information Science and Technology, Pennsylvania State University, State College, PA, 16801. E-mail: czq3@psu.edu.*

true. The delay-tolerant data flows will then need to be sent to an intermediate datacenter, in this case datacenter k, which does have off-peak hours that overlap with the peak hours of datacenter i. These data flows will then be temporarily stored in datacenter k which will forward them to datacenter j when both k and j are in off-peak hours.

The *optimal routing path* methods [11], [12], [16], [17] optimize the routing paths for video flows to minimize the charging volume on each link. The bandwidth costs of transmitting the same amount of videos vary across different inter-datacenter links. Accordingly, if the transmission of a video is expected to exceed the charging volume on a link, the video can be transferred to an alternative path to maximize the utilization of other links without increasing their charging volumes. However, these methods transmit each video immediately when the video transmission request arrives at the source datacenter regardless of its deadline. As a result, these methods can easily reach the charging volume of a current link and create a large number of reroute requests when a large number of video transfer requests arrive simultaneously. This may also increase the charging volumes of some links. An additional consequence of using optimal routing path methods concerns the difficulty in fully utilizing a link's available bandwidth capacity. That is, a links available bandwidth capacity is not fully utilized when the cumulative transmission rate of all the currently transmitted videos is less than the link's available bandwidth capacity.

We propose EcoFlow, an economical and deadline-drive video flow scheduling system, to address the problems of previous scheduling methods. It is based on the fact that different video flows have different deadlines. Different applications from cloud customers have different service-level agreements (SLAs) that specify data Get/Put bounded latency [19] or a service probability [20] by ensuring a certain number of replicas in different locations [21]. Thus, cloud providers would like to assign shorter transmission deadlines (deadline) to videos in applications with more stringent SLAs in order to minimize the SLA violation penalty to maximize their profits [11], [22]. Different videos in one application also have different deadlines. For example, the flows for new video dissemination to a datacenter to serve user requests should have more stringent deadlines than the flows for video replication backups to boost availability. Based on the different deadlines of video flows, the key idea of EcoFlow is to postpone the transfers of some delay-tolerant videos while still ensuring their transmissions within deadlines if the transmission of these videos will increase the current charging volume. This is the novelty of EcoFlow – to significantly reduce the payment cost of the previous flow scheduling methods. The EcoFlow system includes three key steps.

Step 1: **Available Bandwidth Capacity Estimation**. We estimate the available bandwidth capacity on each link by comparing the charging volume and the expected traffic volume. This provides the maximum transmission rate that a link can provide in the next time interval without increasing the current bandwidth cost.

Step 2: **Deadline-Driven Flow Scheduling**. We sort the flows on each link based on their deadline tightness. In the sorted flow queue, we generally give videos with .earlier deadlines higher priority in finishing transmission, and postpone the transfers of some delay-tolerant videos while still ensuring their transmissions occur within deadlines if the transmission of these videos will increase the current charging volume. Flows that are expected to miss their deadlines are split into subflows, which will be rerouted to other underutilized links in order to meet their deadlines.

Step 3: **Alternating Routing Path Identification**. In order to deliver these subflows by their deadlines, we rely on Dijkstra's algorithm [23] to find the shortest path between the source and the destination datacenters in the inter-datacenter network that guarantees the successful transmission by flow deadlines.

In summary, existing flow scheduling methods fail to fully utilize each link's available bandwidth capacity, and may increase the charging volumes, and thus cannot reduce bandwidth costs maximally for cloud providers.

Compared to existing methods, the advantage of EcoFlow lies in 1) the reduction in overall bandwidth cost as much as possible, and 2) video flows can finish transmission before their deadlines through fine-grained (i.e., hourly) estimation of the traffic load on each link and taking advantage of various flow deadlines in flow scheduling. Note that EcoFlow is a heuristic rather than an optimal solution and it outperforms existing methods.

The remainder of the paper is organized as follows. Section 2 gives an overview on related work. Section 3 provides an overview of the EcoFlow design. Section 4 introduces the detailed design of EcoFlow. The performance evaluation on both PlanetLab and EC2 is presented in Section 5. Section 6 concludes the paper with remarks on our future work.

2 Related Work

Many methods have been proposed to schedule the inter-datacenter traffic in order to minimize the ISP bandwidth costs of cloud providers, which can be classified in two groups: store-and-forward and optimal routing path.

Store-and-forward. The methods in this group postpone the transmissions of the delay-tolerant data flow from peak hours to off-peak hours, so as to utilize the leftover traffic during off-peak hours. Laoutaris *et al.* [13], [14] proposed to employ a number of storage servers to collect delay-tolerant traffics and perform data transmission only when the destination datacenter is in predefined off-peak hours, so that the charging volume will not increase during peak hours. NetSticher [15] performs transmissions of delay-tolerant data between two datacenters only when both datacenters are in off-peak hours. If there are no common off-peak hours between both datacenters, it uses an intermediate datacenter that has an overlap in off-peak hours with the destination datacenter as a relay datacenter to store the delay-tolerant data temporarily.

Such store-and-forward transfer systems predefine off-peak hours of each datacenter. Delaying the transmission of delay-tolerant videos from peak hours to off-peak hours is a coarse-grained scheduling strategy. It does not fully utilize the link's available bandwidth capacity when actual traffic load is light during peak hours. Additionally, the transmission of a large number of non-delay-tolerant videos during the peak hours will increase the link's charging volume. Instead, EcoFlow is a fine-grained video flow scheduler which estimates the available bandwidth capacity on each link during a short time interval (i.e., 1 hour), and schedules the pending flows using a link's available bandwidth capacity in an earliest-deadline-first manner. The flows expected to miss their deadlines are rerouted to other under-utilized links to avoid increases in the current charging volume.

Optimal routing path. The optimal routing path methods identify routing paths for video flows with the objective of minimizing the bandwidth payment costs. Multihoming is a scenario in which a user is connected to the internet through multiple links operated by different ISPs, and the links have different bandwidth capacities, availabilities and prices. In order to optimize the user's bandwidth costs and network performance in multihoming, Goldenberg *et al.* [12] used liner programming techniques to dynamically assign traffic among different links. Entact [16] applies a route-injection mechanism to estimate the bandwidth costs of alternating paths that are not being currently used. Based on information of payment costs, traffic, and link capacity, it jointly computes the optimal routing path for online service providers. In order to reduce users' bandwidth costs in multihoming, Wang *et al.* [17] applied both a dynamic programming algorithm and a greedy algorithm to select links operated by different ISPs to transfer the user data. D-hamdhere *et al.* [24] considered monetary cost and network availability in multihoming, and used the first fit decreasing algorithm to select an optimal set of ISPs. Jetway [11] aims to control the transmissions of video flows under the link's charging volume. When a video flow is expected to increase a link's current charging volume, the video will be split into subflows, and the subflows are transmitted on a multi-hop path to utilize the available bandwidth capacity of each link. These methods do not take advantage of the fact that some delay-tolerant videos are elastic to delay when scheduling the traffics. The charging volume of all option links will be increased when a large amount of videos need to be sent concurrently. Instead, EcoFlow takes advantage of different deadlines of flows and postpones the delivery of delay-tolerant videos to utilize a link's available bandwidth capacity when the current traffic load is high, so that the charging traffic volume will not further increase.

EcoFlow adopts some existing techniques such as sub-flow (e.g., MPTCP [25]) and rerouting (e.g., Hedera [26], DevFlow [27]). It also adopts the earliest-deadline-first flow scheduling strategy [28], [29]. While these techniques have previously been applied with goals such as minimizing mean flow completion time or meeting flow deadlines, they do not inherently consider the payment costs for flow transfers between datacenters. In this regard, one of the contributions of the work on EcoFlow is the adoption of these techniques in conjunction with developing an integral video flow scheduling system that aims to minimize the payment costs of inter-datacenter video transfers..

3 Overview of EcoFlow

3.1 Problem Statement

In order to examine the need for EcoFlow consider a cloud with multiple geographically distributed datacenters, operated by a single cloud provider, where ever datacenter in the cloud is connected to every other datacenter. Important notations used in this paper are listed in Table 1, however, a few critically important notations to begin this discussion include using a complete direct graph $G = (V, E)$ to represent the inter-datacenter network, where V is the set of datacenters and E denotes the set of direct links connecting datacenters. We use $e_{j,k} \in E$ to denote a link from datacenter j to datacenter k. For each link $e_{j,k}$, we use a positive value a to denote the cost per traffic unit, a non-negative value c to denote the maximum link capacity (the maximum available transmission rate), and $\hat{v}$ to denote the charging volume.

We use T_r to denote the time window to record traffic volume (i.e., 5-minute interval) for calculating the charging

TABLE 1: Table of important notations.

e	a direct link connecting datacenter i and j
f_k	video flow k
f_k^I	an indirect video flow
f_k^D	a direct video flow
$F(t)$	a set of video flow at time t
$F^I(t)$	a set of indirect video flow at time t
$F^D(t)$	a set of direct video flow at time t
T_p	time window for traffic volume prediction
T_r	time window to record actual traffic volume
$[t_i, t_{i+T_p/T_r})$	time interval for traffic volume prediction
$[t_i, t_{i+1})$	time interval to record traffic volume, t_{i+1}-t_i=T_r
a	bandwidth cost per unit traffic on e
$v(t_i, t_j)$	traffic volume on link e at time interval $[t_i, t_j)$
$\hat{v}(t_i)$	charging volume on link e at time t_i
$P^c(t_i)$	bandwidth cost on link e at time t_i
$\tilde{v}(t_i, t_j)$	estimated traffic volume on link e at $[t_i, t_j)$
c	maximum bandwidth capacity on link e
$\Delta c(t_i, t_j)$	available bandwidth capacity on link e at $[t_i, t_j)$
t_k^{start}	starting time for video flow f_k
t_k^{end}	completion time for video flow f_k
P	reroute path for an indirect video flow
s_k	flow size of video flow f_k
d_k	transmission deadline of video flow f_k
$\hat{v}(t_0)$	initial charging volume on each link e
$\bar{V}(t_{end})$	average charging volume at time t_{end} of all links

volume, and use $P^c_{j,k}(t_i)$ to denote the bandwidth cost on link $e_{j,k}$ at time t_i. We let $\hat{v}(t_i)$ represent the charging volume on link e at time t_i and let $v(t_{i-1}, t_i)$ present the total actual traffic during time interval $[t_{i-1}, t_i)$. Assuming the charging period begins at time t_0, the bandwidth cost on link e at time t_i can be calculated by:

$$P^c_{j,k}(t_i) = \begin{cases} a\frac{\hat{v}(t_{i-1})}{T_r}(t_i - t_{i-1}) & \text{If } v(t_{i-1}, t_i) < \hat{v}(t_{i-1}) \\ a\frac{\hat{v}(t_i)}{T_r}(t_i - t_{i-1}) & \text{otherwise} \end{cases} \quad (1)$$

As shown in Equation (1), when the actual traffic volume during $[t_{i-1}, t_i)$ is smaller than the charging volume at time t_{i-1}, the bandwidth cost at time t_i is calculated by applying the cost function to the charging volume at time t_{i-1}; otherwise, the new bandwidth cost at time t_i is calculated by applying the cost function to the new charging volume at time t_i.

Suppose that there are M video flows f_1, ..., f_m, ..., f_M to be scheduled, where the starting time and the completion time of each f_m are represented by t_m^{start} and t_m^{end}, respectively. We use u_m^{src} and u_m^{dst} to denote the source and the destination of f_m, and use s_m to denote the size of f_m. Each flow is allowed to be partitioned into several subflows and to be delivered via multiple paths. We use $x_{m,i}^{j,k}$ to represent the size of f_m's subflow that is allocated to link $e_{j,k}$ at time $[t_{i-1}, t_i)$. Then, the following conditions must be satisfied:
For each flow f_m,
C1: The total data flowing out of the source u_m^{src} after t_m^{start} equals to s_m: $\sum_{i \geq t_m^{start}} \sum_{j \in V \setminus u_m^{src}} x_{m,i}^{u_m^{src},j} = s_m$.
C2: The total data flowing into the destination u_m^{dst} by time t_m^{end} equals to s_m: $\sum_{i \leq t_m^{end}} \sum_{j \in V \setminus u_m^{dst}} x_{m,i}^{j,u_m^{dst}} = s_m$.

For each flow f_m and each datacenter j that is neither source nor destination of f_m,
C3: The total data in f_m flowing into j equals to the data flowing out of j for the whole time span: $\sum_i \sum_{l \in V \setminus j} x_{m,i}^{l,j} = \sum_i \sum_{k \in V \setminus j} x_{m,i}^{j,k}$
C4: At each time t_τ, the total data in f_m flowing into j should be no smaller than the data flowing out of j: $\sum_{i \leq \tau} \sum_{l \in V \setminus j} x_{m,i}^{l,j} \geq \sum_{i \leq \tau} \sum_{k \in V \setminus j} x_{m,i}^{j,k}$.
C5: The total size of all subflows on each link $e_{j,k}$ can not

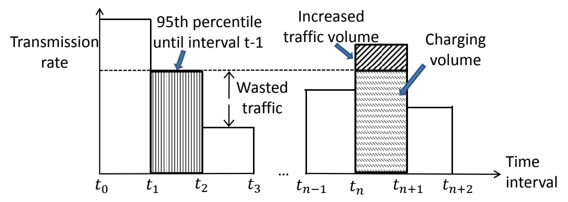

Transmission rate
95th percentile until interval t-1
Wasted traffic
Increased traffic volume
Charging volume
Time interval
t_0 t_1 t_2 t_3 ... t_{n-1} t_n t_{n+1} t_{n+2}

Fig. 1: Example: inter-datacenter video traffic bandwidth cost.

exceed the bandwidth capacity c at each time t_i: $\sum_m x^{l,j}_{m,i} \leq c$.

Additionally, the total actual traffic on the link between data centers j and k during the time interval $[t_{i-1}, t_i)$ can be obtained using the following formula:

$$v(t_{i-1}, t_i) = \sum_{m=1}^{M} \left(x^{j,k}_{m,i} + x^{k,j}_{m,i} \right). \tag{2}$$

The equation can then be rewritten:

$$P^{c}_{j,k}(t_i) = \max_{l \leq i} \sum_{m} \left(x^{j,k}_{m,l} + x^{k,j}_{m,l} \right) \tag{3}$$

The objective of EcoFlow is to design a video flow scheduling strategy to specify each decision variable $x^{j,k}_{m,i}$ to minimize the *overall bandwidth cost*:

$$\min \quad \sum_{i} \sum_{e_{k,j} \in E} \max_{l \leq i} \sum_{m} \left(x^{j,k}_{m,l} + x^{k,j}_{m,l} \right) \tag{4}$$

$$\text{s.t.} \quad \text{Constraints C1-C5 are satisfied.} \tag{5}$$

The bandwidth cost optimization problem above is a convex optimization problem that is similar to the typical minimum-cost multi-flow problem [30] and can be solved in polynomial time. We cannot, however, directly apply existing solutions to solve this problem as doing so requires having complete information on every traffic flow (starting time, deadline, flow size) for the entire time span, which is difficult to predict, particularly at the outset. Accordingly, in the following sections, we introduce a time-efficient heuristic video flow scheduling solution designed to reduce the inter-datacenter bandwidth costs. We will demonstrate the effectiveness of our proposed method by comparing it with some existing methods through extensive experiments in Section 5. For simplicity, we omit subscripts j and k in all notations related to link $e_{j,k}$ throughout the paper.

3.2 Overview of EcoFlow

Constrain charging volume. Many recent methods try to constrain the charging volume on each link to reduce the bandwidth costs. We first present an example to explain the basic idea of these methods. Figure 1 shows an example of an inter-datacenter link's bandwidth cost under the 95th percentile charging model. The traffic volume in time interval $[t_1, t_2)$ is the 95th percentile value until time t_n, and is marked as the charging volume that needs to be paid by the cloud provider at t_n. When a larger traffic volume v comes up in time interval $[t_n, t_{n+1})$, it becomes the new charging volume at time t_{n+1}. Then, from time t_0 to t_{n+1}, the unused bandwidth below the traffic volume v is wasted, that is, the cloud provider does not fully utilized the charging volume. Given this observation, a feasible way to reduce bandwidth cost is to maximize the utilization of the charging volume at different time intervals. For this purpose, when the bandwidth needed is greater than current charging volume, EcoFlow postpones the delivery of later-deadline videos to the time when the traffic load is light. For example, the increased traffic volume in time interval $[t_n, t_{n+1})$ can be postponed to time interval $[t_{n+1}, t_{n+2})$. In this way, when a fixed amount of video flow is transmitted between two datacenters over a given period, the 95th percentile of video volumes over all time intervals is minimized. If multiple video flows have the same deadline, we can randomly order these video flows. Note that here we only consider the deadline to order the video flows, but we can jointly consider other factors such as the importance of the video flow.

Three steps of EcoFlow. Specifically, EcoFlow schedules the video flow transfers on a link to different time slots or to other links in order to fully utilize the charging volume while guaranteeing the successful flow transfer within deadlines. The EcoFlow scheduling mechanism can be divided into three steps as explain in Section 1. We first briefly introduce the three steps using an example in Figure 2, which demonstrate the flow scheduling on two links.

Step 1: Available Bandwidth Capacity Estimation. We use T_p to denote the time window used to estimate the available bandwidth capacity on each link, and use T_r ($T_r < T_p$) to denote the time window to record traffic volume in current charging model. Based on historical data, we estimate the total volume of video traffic needed to be transmitted on each link during time interval $[t_0, t_n)$, $t_n - t_0 = T_p$, denoted by $\tilde{v}(t_0, t_n)$. Assume link e_1's charging volume at time t_0 is $\hat{v}_1(t_0)$, it then can transfer a volume of $\hat{v}_1(t_0) \times T_p/T_r$ video during time interval $[t_0, t_n)$. We define **a link's available bandwidth capacity** as the maximum transmission rate that can be used to transfer videos without increasing the current charging volume during a certain time interval. We then calculate the available bandwidth capacity $\Delta c_1(t_0, t_n)$ on link e_1 during time interval $[t_0, t_n)$:

$$\Delta c_1(t_0, t_n) = \hat{v}_1(t_0)/T_r - \tilde{v}_1(t_0, t_n)/T_p. \tag{6}$$

Step 2: Deadline-Driven Flow Scheduling. On each link, the pending video flows are scheduled on an earliest-deadline-first base. When the traffic capacity is fully occupied at the current interval, we postpone the transfer of flows with later deadlines to later time interval but still guarantee their deliveries by deadlines. In Figure 2, on link e_2, the transmission of flow f_{21} fully utilizes the available bandwidth capacity on link e_2 in time interval $[t_0, t_1)$, so f_{22} with a later deadline than f_{21} will be sent after f_{21} finishes transmission. However, when f_{24} is scheduled after f_{23}, its expected transmission time is at t_5, which is later than its deadline. We divide f_{24} into two subflows: f^{D}_{24} and f^{I}_{24}. On link e_1, all pending videos are scheduled to finish transmission before t_3, its available capacity during $[t_3, t_n)$ is not utilized (highlighted in dashed fill). We call the available bandwidth capacity that are not utilized during $[t_3, t_n)$ **extra bandwidth capacity (**$\delta c_1(t_3, t_n)$**)**, $\delta c_1(t_3, t_n) = \Delta c_1(t_0, t_n)$. We define the links with extra bandwidth capacity during a time interval as the under-utilized links. The extra bandwidth capacity on link e_1 can be utilized to reroute subflow f^{I}_{24} from e_2 by its deadline.

Step 3: Routing Path Identification. For the video rerouting, we aim to identify an alternating path that has extra bandwidth capacity to transmit the video by its deadline. To this end, we apply the Dijkstra's algorithm [23] to identify the alternating routing path.

Advantages of EcoFlow. We then use an example to show the advantage of EcoFlow compared to existing methods. In Figure 3, the time interval in X axis is one second, that is, t_1-t_0=1s. Y axis denotes the available bandwidth capacity on a link. Four videos of 1GB in size need to be transferred on a link, and the link's available bandwidth

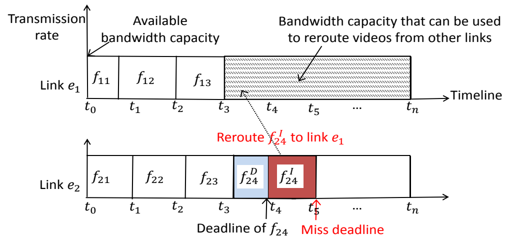


Fig. 2: An overview of EcoFlow.

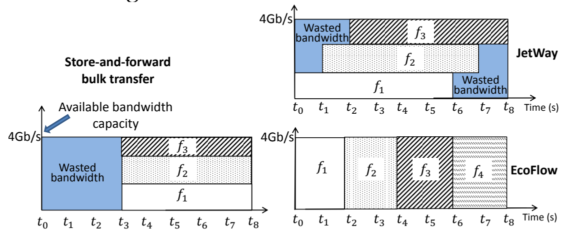


Fig. 3: A comparison of bandwidth utilization between different methods.

capacity is 4Gb/s. We assume that time interval $[t_0, t_3)$ is the link's peak hours, while $[t_3, t_8)$ is in the link's off-peak hours defined in the store-and-forward methods [13]–[15]. The transmission requests of video flows f_1, f_2, f_3 and f_4 arrive at the source datacenter at time t_0, t_1, t_2 and t_3. All four videos are delay-tolerant with transmission deadlines at time t_6, t_7, t_8, and t_9, respectively. In the store-and-forward method, the delay-tolerant videos temporarily wait during peak hours during time interval $[t_0, t_3)$. As no video transmission is performed during peak hours, the link's bandwidth capacity is wasted. The videos are sent out during off-peak hours in $[t_3, t_8)$. However, as a total amount of 2.5 GB data can be transmitted using the link's available bandwidth capacity during the off-peak time, thus only 2.5 videos can finish transmission.

We take Jetway [11] as a representative method of the routing path optimization methods. In JetWay, all videos are sent out when their transmission requests arrive at the source datacenter. The transmission rate of each video flow is calculated by dividing the size of video by the time span between the transmission request arrival time and the video's deadline. Thus, each video is transmitted at the rate of 1GB/6s=1.33Gb/s. f_1, f_2 and f_3 will be transmitted at the rate of 1.33Gb/s at time t_0, t_1 and t_2, respectively. When f_4 arrives at time t_3, the link's bandwidth capacity is fully occupied by other videos, and then f_4 has to be rerouted to other links. In this example, the bandwidth capacity marked in blue is wasted.

In EcoFlow, for contrast, all videos are transmitted using the available bandwidth capacity, and the transmission of later-deadline videos will be postponed to future time slots if current traffic increases the charging volume. In this example, f_2 will be postponed until f_1 finishes its transmission, and f_3 and f_4 are then transferred one by one. All four videos can be transmitted before their deadlines and the link's bandwidth capacity is fully utilized. Therefore, EcoFlow can fully utilize the link bandwidth capacity to reduce the bandwidth payment cost of existing methods.

4 System Design

4.1 Available Bandwidth Capacity Estimation

When the total traffic volume at current time interval $[t_i, t_{i+1})$ $(t_{i+1} - t_i = T_r)$ exceeds the current charging volume, EcoFlow postpones the transmission of later-deadline videos to utilize the link's available bandwidth capacity in future time intervals. In order to predict whether a link has enough bandwidth capacity to transmit these videos, EcoFlow estimates each link's available bandwidth capacity during T_p with two steps: 1) traffic volume prediction during T_p, and 2) available bandwidth capacity estimation, which is the maximum volume of traffic a link can transfer without further increasing the current bandwidth cost.

Traffic volume prediction. Note that the traffic transmitted on link e includes traffic transmitted from datacenter i to j and from datacenter j to i. Exponentially weighted moving average (EWMA) [31] is widely used for prediction for a given series of data points. We use EWMA to estimate the traffic volume during $[t_i, t_i + T_p]$ on link e (denoted by $\tilde{v}(t_i, t_i + T_p)$) based on the actual historical traffic volume (denoted by $v(\cdot)$):

$$\tilde{v}(t_i, t_i + T_p) = \beta \times v(t_i - T_p, t_i) + (1 - \beta) \times \tilde{v}(t_i - T_p, t_i). \quad (7)$$

$\tilde{v}(t_i - T_p, t_i)$ is the estimated traffic volume in time interval $[t_i - T_p, t_i)$, and β $(0 < \beta < 1)$ is a constant used to control the degree of weighting decrease.

Available bandwidth capacity estimation. From historical flow records, we calculate the charging volume on link e at time t_i, $\hat{v}(t_i)$. Thus, during time interval $[t_i, t_i + T_p)$, a total volume of $\hat{v}(t_i) \times T_p/T_r$ video traffic can be transferred under the current bandwidth cost. Given estimated traffic volume $\tilde{v}(t_i, t_i + T_p)$, we can calculate the available bandwidth capacity in time interval $[t_i, t_i + T_p)$:

$$\Delta c(t_i, t_i + T_p) = \min\{c, \hat{v}(t_i)/T_r - \tilde{v}(t_i, t_i + T_p)/T_p\}. \quad (8)$$

When $\Delta c(t_i, t_i + T_p) > 0$, the current charging volume on link e is larger than the expected traffic volume, and the available bandwidth capacity can be used to reroute video flows from other links. The accuracy of EWMA in prediction was studied in previous works such as in [32]. Its true positive rate achieves over 90% while its false positive rate is under 5%.

4.2 Deadline-driven Flow Scheduling

Like existing works [11], [33] that assume the existence of a centralized server (which connects to all the datacenters) as a scheduler to globally schedule the video flows across different datacenters, we first introduce EcoFlow in a centralized manner. We will further introduce a distributed way to implement EcoFlow in Section 4.8.

In order to maximize the number of videos that can be transmitted by their deadlines, we use the earliest-deadline-first strategy, that is, the video with the earliest deadline will be put at the front of the sending queue. The network scheduler maintains a sending queue $Q(t_i)$=($< f_1, d_1, s_1 >$,$< f_2, d_2, s_2 >$,... $< f_m, d_m, s_m >$) to store all pending flows on each link e at time t_i, which are ordered based on their deadlines. Note that flows on link e include all flows that are transmitted bidirectionally between datacenter i to j (i.e., from i to j or from j to i). Each triple $< f_k, d_k, s_k >$ in $Q(t_i)$ contains the flow information of f_k, where d_k and s_k are the deadline and size of f_k, respectively.

All pending videos in $Q(t_i)$ are sent out sequentially, that is, f_k will be sent only when all videos f_1, f_2, ..., f_{k-1} have finished transmission. As we see from Figure 3, when a number of videos are transmitted on a link simultaneously and their cumulated transmission rate is less than the link's available bandwidth capacity, the bandwidth resource of this link is wasted as the extra bandwidth capacity is not utilized. Thus, EcoFlow aims to maximize the bandwidth utilization by sending video at a rate that fully utilizes the available bandwidth capacity. The estimated flow transmission time for flow f_k can be computed as:

$$T_k = \mathcal{A}/\Delta c(t_i, t_i + T_p). \tag{9}$$

$\mathcal{A} = \sum_{f_p \in F_{<k}} s_p$, where $F_{<k}$ is a subset of flows in $Q(t_i)$ that have earlier deadlines than flow f_k (including f_k), and $\Delta c(t_i, t_i + T_p)$ is the estimated available bandwidth capacity on link e in time interval $[t_i, t_i + T_p)$. As the flows in $Q(t_i)$ are sent sequentially, the flow completion time for f_k is t_k^{end}:

$$t_k^{end} = \begin{cases} t_i + T_k & \text{if } k = 1 \\ t_{k-1}^{end} + T_k & \text{otherwise} \end{cases} \tag{10}$$

The flow start time for f_1 is t_i. For flow f_k ($k > 1$), the flow start time is the completion time of the previous flow, that is, $t_k^{start} = t_{k-1}^{end}$. d_k is the deadline of video f_k. When $t_k^{end} \le d_k$, flow f_k is expected to finish transmission before its deadline, and we call it a **Direct Flow** (DF). When $t_k^{end} > d_k$, flow f_k is likely to miss the transmission deadline under the expected bandwidth capacity. Then, f_k can be split to two subflows: f_k^D and f_k^I. f_k^D is the volume with size s_k^D that can be transmitted directly on link e before its deadline; while f_k^I is the residual volume with size s_k^I (s_k^I=$s_k - s_k^D$), which should be rerouted in an alternating path before its deadline. We call f_k^I an **Indirect Flow** (IF).

$$s_k^D = max(0, d_k \times \Delta c(t_i, t_i + T_p) - \mathcal{A}). \tag{11}$$

Using the alternating routing path identification method in Section 4.4, we identify alternating paths for each indirect flow f_k^I which can transmit f_k^I before its deadline.

Algorithm 1 Pseudocode for identifying alternating path for flow f_k^I.

```
1:  Input: $G = (V, E)$; $s_k$, $\delta c(t_i, t_i + T_p)$, $\forall e \in E$;
2:  Output: alternating path $P$ from source $i$ to destination $j$
3:    for each vertex $u$ in $V$
4:      $rate[u]$ := 0  //The maximum transmission rate on each path
5:      $pre[u]$ := null  //Record last hop on the path
6:      $t_k^{end}[u]$ := $t_i$  //Transmission completion time
7:      $Q+ = j$  //$Q$ is a temporal set
8:    end for
9:    $rate[i]$ := infinity
10:   while $Q$ is not empty do
11:     u := vertex in Q with max rate[u]
12:     remove u from Q
13:     for each neighbor $v$ of $u$ with $\delta c_{uv}(t_i, t_i + T_p) > 0$ do
14:       $t_k^{end}[v]$:=$s_k^I/\min\{rate[u], \delta c_{uv}(t_i, t_i + T_p)\} + t_i$
15:       alt :=max$\{rate[v], \delta c_{uv}(t_i, t_i + t_k^{end}[v])\}$
16:       if alt $\ge rate[v]$:  // A path with higher transmission rate
17:         $rate[v]$ := alt
18:         $pre[v]$ := u
19:       end if
20:   $P$ := empty sequence
21:   while $pre[j]$ is defined do  //Construct the alternating path
22:     insert $j$ at the beginning of $P$
23:     $\delta c_{j,pre[j]}(t_i, t_i + t_k^{end}[j])$ := 0
24:     $j$ := $pre[j]$  // Traverse from destination to source
25:   if $t^{end}[v]$<$d_k$ Return $P$
26:   if $t^{end}[v]$>$d_k$  //$P$ cannot transmit $f_k^I$ before its deadline
27:     split $f_k^I$ into $f_{k1}$, $f_{k2}$
28:     Return $P$, $f_{k1}$, $f_{k2}$
29:   end if
```

After all IFs find alternative paths, the scheduler creates a schedule table for all pending flows, in which each item is expressed in a quadruple $S(t_i) =< f_k, S, D, t_k^{start} >$, which denotes the flow ID, source datacenter, destination datacenter and flow start time. This table is used to guide the transfers of flows initiated from all datacenters.

4.3 Alternating Routing Path Identification

$F^I(t_i)$ denotes the set of all IFs in the network at time t_i, which are sorted by their deadlines in ascending order. Assume f_k^I is an IF from datacenter i to datacenter j, in this section, we describe how the scheduler identifies an alternating routing path P for f_k^I, $P = (v_1, v_2, \ldots, v_p)$. The centralized scheduler identifies an alternating path for each IF in an earliest-deadline-first manner.

The selected alternating path P uses extra bandwidth capacities of its constituent links to transmit f_k^I, with the requirement of finishing the transmission before its deadline. The path that can transmit f_k^I with the minimum transmission time among all possible alternating paths is the best path to satisfy this requirement. Thus, we first identify the alternating path P that leads to the minimum transmission time. If the identified path can transmit f_k^I before its deadline, we reroute f_k^I to this alternating path; otherwise we split f_k^I into two parts: f_{k1} and f_{k2}, where f_{k1} is the part of f_k^I that is expected to finish transmission on P. We reroute f_{k1} along P, and identify another alternating path for f_{k2} by using the same process. If the new identified alternating path cannot transmit f_{k2} before its deadline, f_{k2} is further split into two parts: f_{k2}^1 and f_{k2}^2. f_{k2}^1 is transmitted on the new alternating path and f_{k2}^2 is transmitted on e by increasing the charging volume on e.

When f_k^I is transmitted on path P, its transmission rate is the minimum extra bandwidth capacity on all P's constituent edges [34], that is $\min_{\forall i \in (1,p-1)}\{\delta c_{i,i+1}(t_i, t_i + T_p)\}$. f_k^I's transmission completion time t_k^{end} is calculated by:

$$t_k^{end} = s_k^I / \min_{\forall i \in (1,p-1)}\{\delta c_{i,i+1}(t_i, t_i + T_p)\} + t_i, \tag{12}$$

where s_k^I denotes the flow size of f_k^I and $\delta c_{i,i+1}(t_i, t_i + T_p)$ denotes the extra bandwidth capacity on link $e_{i,i+1}$. We then express the requirement that the transmission of flow f_k^I on path P would be finished before its deadline by: $t_k^{end} \le d_k$. Therefore, transmitting subflows along the alternating path will not interfere with the other video flows being transmitted in the constituent edges of the path or violate the deadline requirement of the flow for the subflows. The path that can transfer f_k^I with the minimum transmission time is the best path to satisfy this requirement, which is shown in Equation (13):

$$\min_{\forall P}\{t_k^{end}(P)\} \tag{13}$$

The Dijkstra's algorithm can be used to find the path that can transmit f_k^I with the minimum transmission time. Then, we use a modified Dijkstra's algorithm to identify an alternating routing path for f_k^I but we modify this algorithm to meet our need, i.e., meeting the deadline, as shown in Algorithm 1. In this algorithm, we input the flow information including its size, deadline, source datacenter and destination datacenter, together with network information including all link's extra bandwidth capacity. Algorithm 1 will return an alternating path P for f_k^I, and splits f_k^I into two parts if P cannot finish transmission before its deadline. In Algorithm 1, we modify the original Dijkstra's algorithm to check whether the identified alternating path P meets the deadline requirement through the addition of Lines 25-29. If the identified path can transmit f_k^I before its deadline, alternating path P is returned (Line 25); otherwise f_k^I is split into f_{k1} and f_{k2} (Line 26-29). The simplest implementation of the Dijkstra's algorithm requires a running time of $O(|E| + |V|^2) = O(|V|^2)$.

If P cannot finish f_k^I's transmission before its deadline and f_k^I is split into f_{k1} and f_{k2}, we will use Algorithm 1 to identify an alternating path for f_{k2}. If the new identified alternating path cannot transmit f_{k2} before its deadline, f_{k2} is further split into two parts: f_{k2}^1 and f_{k2}^2. f_{k2}^1 is transmitted on the new alternating path and f_{k2}^2 is transmitted on e by increasing the charging volume on e. The new charging volume $\hat{v}(t_i)$ at time t_i is calculated by:

$$\hat{v}(t_i) = (\mathcal{A} - s_k + s(f_{k2}^2) \times T_r/(d_k - t_i). \tag{14}$$

Where $s(f_{k2}^2)$ is the size of f_{k2}^2. The flow start time and completion time of all flows on link e will then be updated based on Equation (9), and the schedule table $S(t_i) =< f_k, S, D, t_k^{start} >$ will also be updated.

4.4 Forwarding Subflows with Rate Limiters

When sending video flows in the inter-datacenter network, we need to control the transmission rate so that the flows are sent out by using the links' available bandwidth capacities. Each datacenter i deploys a scheduler C_i to organize the sending queue of video flows, which attaches labels to all packets of each flow that record their corresponding transmission paths. For each video flow, the datacenter also uses a rate limiter [35], [36] to control its sending rate within the available bandwidth capacity, so that the links' current charging volume on the flow's transmission path will not increase.

If flow f_k is a DF sending from datacenter i to datacenter j, all packets of f_k are transmitted using available bandwidth capacity on link e (i.e., $\Delta c(t_i, t_i + T_p)$). The number of packets transmitted per second r_k is calculated by:

$$r_k = \Delta c(t_i, t_i + T_p)/\mu, \tag{15}$$

where μ is the size of a packet. The cloud provide specifies the value of μ, typically, μ is set to 1KB [37], [38].

If flow f_k is split into multiple subflows, (e.g., f_k is split into f_k^D and f_k^I, and f_k^I is further split into f_{k1} and f_{k2}), the rate limiter needs to split the packets proportionally according to each link's available bandwidth capacity. Assume f_k is split into (f_{k1}, f_{k2},...,f_{km}), which are transmitted using links (e_{i1}, e_{i2},...,e_{im}). The total number of packets sent per second for f_k is:

$$r_k = \sum\nolimits_{j=1}^{m} \Delta c(t_i, t_i + T_p)/\mu. \tag{16}$$

In this way, the f_k's packets are proportionally distributed across all paths.

4.5 Setting Initial Charging Volume

According to the 95th percentile charging model, the charging volume at the beginning of the charging period is 0 and it increases gradually as the bandwidth usage goes up. Figure 4 shows an example where a link's charging volume increases over time. In this example, the charging volume rises drastically in early time intervals and remains relatively stable after a given time interval. This property leads to a problem in our design. During early time intervals, as available bandwidth capacity of a direct link is 0 and cannot transmit a video flow f_k on this path, f_k needs to look for an alternating path. f_k cannot find any alternating path because the available bandwidth capacities of all links are 0. Finally, the direct link needs to increase its charging volume in order to transmit f_k. This process leads to extra scheduling latency due to insufficient available bandwidth capacities in all links.

To reduce the scheduling latency and make the schedule process simple during early time intervals of a charging period, we set an initial charging volume on each link, as shown in the dash line in Figure 4. Assume each charging period is divided into a number of time points $< t_0, t_1, ...t_{end} >$, i.e., it starts from time t_0 and ends at time t_{end}. The initial charging volume on each link e at time t_0 is denoted by $\hat{v}(t_0)$. $\hat{v}(t_0)$ should be set properly. If $\hat{v}(t_0)$ is

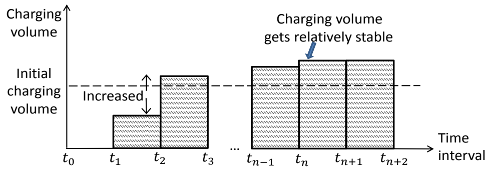


Fig. 4: An example of setting an initial charging volume at the beginning of a charging period.

too small, the scheduling latency cannot be reduced since a direct link does not have enough available bandwidth capacity to transmit flow f_k, and also we can hardly find an alternating path because other links' initial charging volumes are small. On the other hand, if $\hat{v}(t_0)$ is too large, the initial charging volume is not fully utilized throughout the charging period, and the proposed scheduling scheme will not be effective in reducing each link's bandwidth cost. In this section, we introduce how to set the initial charging volume on each link based on the historical data, i.e., actual charging volume at the end of the last charging period (denoted by t_{end}), since it can be an indicator of how much traffic volume will be transmitted during current charging period.

We consider two factors in setting $\hat{v}(t_0)$, which are the actual charging volume at time t_{end} on link e ($\hat{v}(t_{end})$) and the average actual charging volume at time t_{end} on all links in the inter-datacenter network (denoted by $\bar{V}(t_{end})$). $\bar{V}(t_{end})$ is calculated as:

$$\bar{V}(t_{end}) = \sum\nolimits_{e \in E} \hat{v}(t_{end})/2|E|. \tag{17}$$

$\hat{v}(t_{end})$ plays a major role in determining the initial charging volume on e, because EcoFlow encourages sending video flows using direct link e to reduce transmission time. As introduced in Section 4.1, $\hat{v}(t_{end})$ can be calculated based on the actual historical traffic volume during previous charging period. We consider $\bar{V}(t_{end})$ because EcoFlow aims to transmit video flows using the available bandwidth capacities of all links in the inter-datacenter network so as to control the charging volume on a specific link and offload partial traffic to alternating paths. Combining these two factors, we calculate $\hat{v}(t_0)$ by:

$$\hat{v}(t_0) = \phi \hat{v}(t_{end}) + \varphi \bar{V}(t_{end}). \tag{18}$$

$\phi \in (0, 1)$ is a weight placed on $\hat{v}(t_{end})$; $\varphi \in (0, 1)$ is a weight placed on $\bar{V}(t_{end})$. At the beginning of each charging period, the charging volume on link e is set to $\hat{v}(t_0)$. In Section 5.4, we evaluate the performance of this strategy by setting different values of ϕ and φ.

4.6 Deal with Prediction Errors and Lack of Prior Knowledge

Our design is based on prediction. That is, we estimate the available bandwidth capacity on each link and utilize it to reroute videos. Prediction errors, (e.g., there are more video flows waiting to be transmitted on a link than the estimated value due to overestimating the available bandwidth capacity), can lead to an excessive backlog of videos left waiting in the sending queue. To handle the prediction errors, our design can adapt by revising the transmission schedule. When an excessive number of pending videos appear in the sending queue and EcoFlow cannot find alternating paths for these videos, it then uses Equation (14) to calculate a new charging volume on this link.

When we lack prior knowledge of the charging volume that existed during previous charing periods, as describe in Section 4.5, the cloud service provider will set each link's initial charging volume based on how much bandwidth cost it is willing to pay, or it can set the initial charging volume to 0 and let the charging volume increases gradually as more videos are transmitted.

4.7 Centralized Implementation of EcoFlow

In this section, we pull together the various constituent parts of EcoFlow and describe how it operates in situations of centralized implementation. As mentioned in Section 4.2, each scheduler maintains a sending queue $Q(t_i)$=($<f_1, d_1, s_1>$, $<f_2, d_2, s_2>$,...,$<f_m, d_m, s_m>$) to store all pending video flows on each link e at time t_i, and the flows are sorted in ascending order based on their deadlines. The goal of EcoFlow is to calculate a schedule table $S(t_i)$. That is, it decides a transmission path for each flow, and splits a flow into subflows and identifies alternating paths for them if a flow is estimated to miss its transmission deadline. We describe the process of scheduling video flows on link e in Algorithm 2. If t_i is the beginning of a charging period, the scheduler first sets an initial charging volume on e (Line 3-5). It then calculates e's available bandwidth capacity (Line 6). For each f_k in the sending queue, EcoFlow calculates expected transmission time t_k^{end}. If t_k^{end} is earlier than f_k's transmission deadline, f_k will be transmitted on link e (Line 9-10). Otherwise, EcoFlow splits f_k into f_k^D and f_k^I and identifies an alternating path for f_k^I (Lines 12-13). If an alternating path P is found to transmit f_k^I, we update the available bandwidth capacity on each edge of P (Line 14). If no alternating path can be found to transmit the whole volume of f_k^I, we increase charging volume on e at time t_i according to Equation (14) (Lines 15-17). Finally, the scheduling table $S(t_i)$ is updated based on the scheduling results (Line 20).

4.8 Distributed Implementation of EcoFlow

In the centralized implementation of EcoFlow, when the centralized scheduler fails, the scheduling function comes to a halt. In order to prevent the single point of failure problem, we propose a distributed implementation of EcoFlow. Thus, when one scheduler fails, the scheduling system can still function by utilizing other normal schedulers. As mentioned before, each datacenter i has a scheduler C_i. For flow scheduling on link e, we select a scheduler on datacenter i or j as a master scheduler, denoted by scheduler C, and the other schedule then becomes the slave scheduler. Scheduler C is responsible for scheduling transmission of flows on e, calculating the available bandwidth capacity (Δc) on link e using the same technique as in Section 4.1, and broadcasts this information to all schedulers in the network for alternating path identification.

At each time interval T_r, scheduler C_i and scheduler C_j report information of their pending flows on link e (including flow ID, size and deadline) to scheduler C. Scheduler C then orders the flows by their deadlines in ascending order, and calculates start time and completion time for each flow using the same technique in Section 4.8. All flows on link e are divided into DFs ($F^D(t_i)$) and IFs ($F^I(t_i)$). Scheduler C builds a schedule table $S(t_i) =< f_k, S, D, t_k^{start} >$ for flows in $F^D(t_i)$ and forwards it to both scheduler C_i and scheduler C_j. As DFs can be transmitted directly through e, scheduler C_i and scheduler C_j transfer flows in DFs according to the schedule table. Scheduler C also needs to find the alternating paths for $F^I(t_i)$ and notifies scheduler C_i and scheduler C_j the alternating paths.

Assume f_k^I is an IF flow from datacenter i to datacenter j, we need to identify an alternating path that can transfer f_k^I before its deadline. Due to the lack of a centralized scheduler, the challenge of the distributed identification method lies in finding an alternating path through the cooperation of multiple schedulers. Under this scenario, identifying a path with the minimum transmission time f_k^I is complicated, so we aim to find a path that can transfer f_k^I before its deadline. As each datacenter has direct links connected to all other datacenters, a sufficient number of datacenters can be chosen as relay datacenters in the alternating paths. With a high probability, we can find a relay datacenter and build a 2-hop alternating path that can transfer f_k^I before its deadline. While identifying a multi-hop (more than 2-hop) alternating path requires cooperation of multiple schedulers and is not efficient in time complexity, we aim to simplify the implementation, and achieve algorithm time efficiency, we identify a 2-hop alternating path $P = (i, h, j)$ for f_k^I in our proposed method, that is, find an intermediate datacenter h and transfer f_k^I on path $P = (i, h, j)$. Multiple candidate datacenters might be able to relay and transfer f_k^I before its deadline, we then contact each datacenter's scheduler and randomly choose a datacenter h who are able to relay f_k^I as the intermediate datacenter. Note that this routing path identification method can be extended to identify alternating paths with more than two hops.

Algorithm 2 Pseudocode for scheduling video flows on link e.

```
1:  Input: $G = (V, E)$; $Q(t_i)$;
2:  Output: schedule table $S(t_i)$ for all flows in $Q(t_i)$
3:    if $t_i$ is the beginning of a charging period
4:       set initial charging volume $\hat{v}(t_0)$
5:    end if
6:    calculate available bandwidth capacity $\tilde{v}(t_i, t_i + T_p)$
7:    for each $f_k \in Q(t_i)$
8:       calculate expected transmission time $t_k^{end}$
9:       if $t_k^{end}$ is smaller than deadline $d_k$
10:         $f_k$ is transmitted directly on link $e$
11:      else
12:         split $f_k$ into subflows: $f_k^D$ and $f_k^I$
13:         find alternating path $P$ for $f_k^I$ using Algorithm 1
14:         update available bandwidth capacity on each link of $P$
15:         if $P$ cannot transmit whole volume of $f_k^I$
16:            increase $\hat{v}(t_i)$ on $e$ according to Equation (14)
17:         end if
18:      end if
19:   end for
20:   update scheduling table $S(t_i)$ for $f_k$ and subflows
```

Algorithm 3 Pseudocode of finding intermediate datacenter h.

```
1:  Input: $\delta c(t_i, t_i + T_p)$, $\forall ij \in E$;
2:  Output: intermediate datacenter $h$ between source $i$ and dest. $j$
3:    for each vertex $q$ in $V \backslash j$:
4:       $t_k^{end}$:=$s_k/\min\{\Delta c_{iq}, \Delta c_{qj}\} + t_i$ //Calculate completion time
5:       if $t_k^{end} < d_k$: //Guarantee the transmission deadline
6:          scheduler $C_i$ contacts scheduler $C_q$
7:          if $\delta c_{iq}(t_i, t_i + t_k^{end}) > 0$ and $\delta c_{qj}(t_i, t_i + t_k^{end}) > 0$:
8:             $h := q$
9:          end if
10:      end if
11:   end for
```

The information of available bandwidth capacity on each link at each time interval is shared among all schedulers by broadcasting. The intermediate datacenter h is selected according to Algorithm 3. In this algorithm, we try each datacenters in $V \backslash j$ to build a candidate path (Line 3). For each candidate path, we then calculate f_k^I's transmission time on this path (Line 4). If path P can transfer f_k^I before its deadline, we further check if the links on P have extra bandwidth capacities (Line 5-10). When intermediate datacenter h is found, scheduler C_i

then forwards f_k^I to datacenter h, and scheduler C_h further forwards it to its destination j. If scheduler C_i cannot find a transit datacenter h, scheduler C_i increases the charging volume on e according to Equation (14).

5 Performance Evaluation

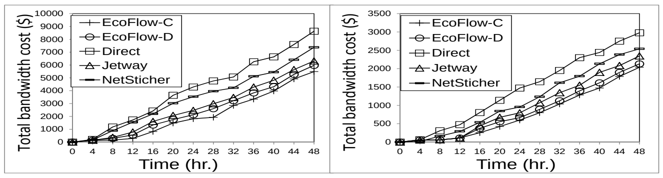
We conducted experiments on the PlanetLab [39] real-world testbed and Amazon EC2 platform [40] to evaluate the performance of EcoFlow in comparison with other systems. For EcoFlow, We tested EcoFlow as both a centralized scheduler (denoted as EcoFlow-C) and as a distributed scheduler (denoted as EcoFlow-D). We compare the performance of EcoFlow with three datacenter traffic scheduling strategies:
1) Direct transfer (denoted as Direct): This transfer type directly transfers video flows to the destination whenever the video transfer requests are initated by the cloud provider without considering each link's charging volume. We use it to represent optimal routing path methods to be compared wtih EcoFlow in our evaluation.
2) JetWay [11]: Jetway transfers video flows whenever the video transfer requests are initiated by the cloud provider, at a rate calculated by its size divided by its corresponding maximum tolerable transfer time. When a video flow is expected to increase a link's current charging volume, it splits the video flow into two sub-flows, and the subflows are transmitted along alternating paths to utilize the available bandwidth capacity of each link.
3) NetSticher [15]: NetSticher is a store-and-forward method that transfers delay-tolerant data between two datacenters only when both datacenters are in off-peak hours. When there are no common off-peak hours between both datacenters, an intermediate datacenter is used to store the data temporarily and then forward it to the destination datacenter.

In both PlanetLab and EC2 experiments, we defined two types of videos: Standard Definition (SD) videos with sizes randomly selected in [500, 800] MB, and High Definition (HD) videos with sizes randomly selected in [2, 4] GB [11]. We assumed that the traffic load for each datacenter displays a periodic diurnal pattern [15]. For simplicity, we further assumed that 10-12am and 6pm-12am of a node's local time are peak hours. A datacenter transfers x and y videos per hour (including both SD and HD videos) to all other datacenters during its peak hours and off-peak hours, respectively, where x and y were randomly selected from [2, 5] and [0, 1], respectively. The transfer request of each video is initiated at a random time during the selected hours, and its deadline is chosen in [30, 120] minutes after the transfer request's initiated time. We assumed a video with maximum tolerable transfer time longer than 60 minutes to be a delay-tolerant. We set $T_p = 1$ hour and $T_r = 5$ minutes. We simulated an inter-datacenter network running for 48 hours for all methods. We set this 48 hour period as an independent charging period and calculated the bandwidth cost on each link at the end of the experiment. In EcoFlow-C and EcoFlow-D, we had a 48 hour warmup period and used the traffic records in this period to predict the traffic volume on each link during the charging period. We also set the initial charging volume based on the charging volume in this warmup period according to Section 4.5. We calculated the bandwidth costs under the 95th percentile charging model.

5.1 Experimental Results for Overall Performance

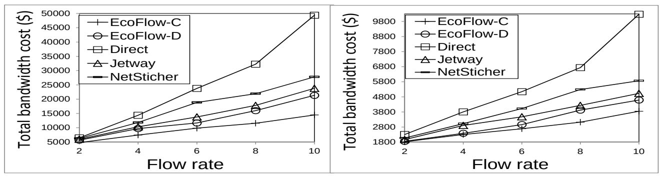
We first present the overall performance of EcoFlow in terms of bandwidth cost, percentage of flows transmitted within the charging volume and percentage of transferred flows within deadlines. In order to compare EcoFlow with other scheduling methods, we set the initial charging volume to 0 in these experiments.
Settings on PlanetLab. We used 15 distributed nodes worldwide to simulate 15 datacenters, including 7 nodes in North America, 5 nodes in East Asia and 3 nodes in Europe. On each link between two datacenters, the bandwidth capacity is randomly selected in [10, 600] MB, and the bandwidth cost per unit (MB) is randomly selected in [50, 400] [11]. Each node's time zone is determined based on its location. In the experiment, we used the TCP protocol to transfer data between different nodes.
Settings on EC2. We have conducted our experiments in the Amazon EC2 platform, which is one of the dominant Infrastructure as a Service (IaaS) cloud providers. There are a total of 7 datacenters on EC2, the capacity and cost per traffic unit of each link is set according to the studies in [11]. We assigned the diurnal load described above to each datacenter based on the time zone it resides in.

We first defined a metric of *total bandwidth cost* as the sum of bandwidth payment cost on all links in the network. Figure 5(a) and Figure 5(b) show the total bandwidth cost at each time interval in PlanetLab and EC2, respectively. We see that as time evolves, bandwidth payment cost for each method is increasing due to the reason that bandwidth payment cost is a function of how long the link's bandwidth is used according to Equation (1). The result also follows: EcoFlow-C<EcoFlow-D<JetWay<NetSticher<Direct. Direct results in the highest bandwidth cost. When a video transfer request arrives at the source datacenter, it immediately transfers the video by using only the direct link between two datacenters without considering the current charging volume on the link. NetSticher postpones the transmission of delay-tolerant videos until both source datacenter and destination datacenter are during off-peak hours, so that the traffic load during peak hours is alleviated and it generates less average bandwidth cost than Direct. However, as the available bandwidth capacity is not fully utilized during peak hours, there is still room for NetSticher to further reduce the bandwidth cost. JetWay is able to incur less bandwidth cost than NetSticher by controlling the transmissions of current videos within the charging volume. Also, when a video is expected to increase a link's charging volume, it splits the video into subflows and reroutes the subflows to links that are under-utilized. However, as videos are transmitted immediately when the video transfer requests are initiated by the cloud provider, the bandwidth cost will increase when a large number of video transfer requests arrive simultaneously. EcoFlow generates the least bandwidth cost among all comparison methods. Figure 5(a) shows that EcoFlow-C generates around an \$800 reduction in total bandwidth cost when compared to Jetway for the 48 hour charging period in PlanetLab, while Figure 5(b) shows that EcoFlow-C generates about a \$300 reduction in total bandwidth cost when compared to Jetway for the 48 hour charging period in EC2. This is due to the fact that EcoFlow schedules the flows on each link based on their deadline tightness, and postpones the transmission of video flows to make the current traffic within the charging volume. Flows that are expected to miss their deadlines are split into subflows, which will be rerouted to alternate paths that are constructed by under-utilized links. Also, EcoFlow transmits each video with the link's available bandwidth capacity, so that the charging volume is fully utilized. Note that EcoFlow-C performs better than EcoFlow-D as it gains



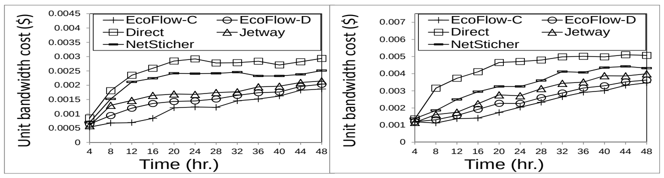
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 5: Total bandwidth cost at different time intervals.



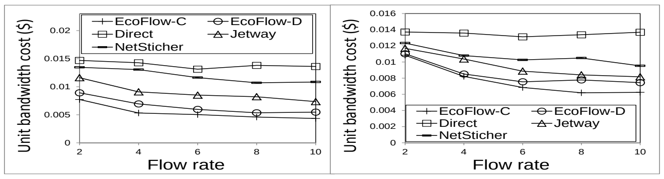
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 6: Total bandwidth cost at different flow rates.



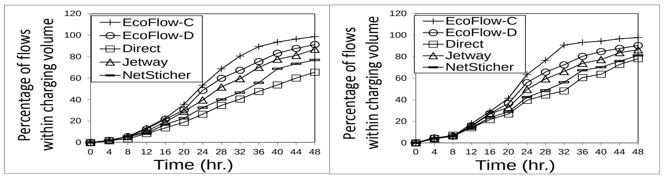
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 7: Average unit bandwidth cost at different time intervals.



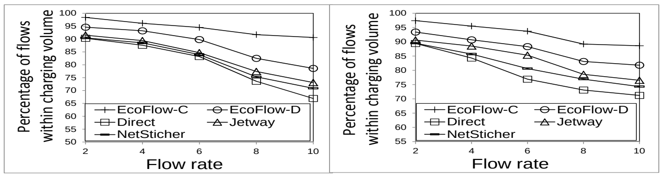
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 8: Average unit bandwidth cost at different flow rates.



(a) Results on PlanetLab. (b) Results on EC2.

Fig. 9: Average percentage of flows transmitted within the charging volume at different time intervals.



(a) Results on PlanetLab. (b) Results on EC2.

Fig. 10: Average percentage of flows transmitted within the charging volume at different flow rates.

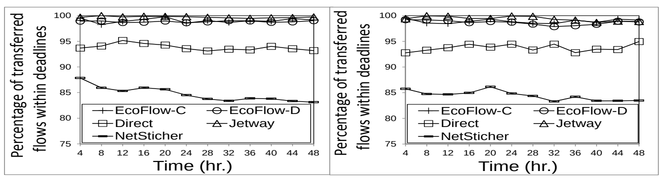
full knowledge of all under-utilized links in the network, and thus has higher probability to identify a reroute path for IFs using under-utilized links.

Next, in order to test the performance of each method in the presence of different traffic loads, we changed the flow arrival rates during a link's peak hours from 2 to 10 flows per hour on each link. Figure 6(a) and Figure 6(b) show the total bandwidth cost at the end of the 48-hour charging period at different flow rates in PlanetLab and EC2, respectively. We see that as more flow transmission requests are initiated hourly, the total cost tends to increase. This is due to the additional bandwidth that is generally required on each link to transmit every video when a larger number of videos need to be transferred between datacenters during the charging period, thus increasing the charging volume on each link. The relative performance of different methods in Figure 6(a) and Figure 6(b) concurs with that in Figure 5(a) and Figure 5(b). Figure 6(a) shows that EcoFlow-C generates about \$9000 reduction in total bandwidth cost for the 48 hour charging period than Jetway in PlanetLab when the flow rate is 10 flows per hour. Figure 6(b) shows that EcoFlow-C generates about \$1000 reduction in total bandwidth cost for the 48 hour charging period than Jetway in EC2 when the flow rate is 10 flows per hour. These observations are due to the same reason in Figure 5(a) and Figure 5(b).

We then present a performance metric of *average unit bandwidth cost*, which is defined as the sum of bandwidth payment cost on all links divided by the total volume of video flows (MB) transmitted in the network. An effective scheduling system should be able to reduce the average unit bandwidth cost, i.e., use the same bandwidth cost to transmit a larger size of videos. Figure 7(a) and Figure 7(b) plot average unit bandwidth cost at each time interval in PlanetLab and EC2, respectively. We see that as time evolves, the average unit bandwidth cost for all methods generally increases because bandwidth payment cost is increased as explained in Figure 5(a) and Figure 5(b). The relative performance between different methods follows: EcoFlow-C<EcoFlow-D<JetWay<NetSticher<Direct. EcoFlow generates the unit bandwidth cost among all methods as it postpones the transmission of video flows to make the current traffic within the charging volume, and it splits the flows that are expected to miss their deadlines into subflows and utilizes other links' available bandwidth capacities to reroute these subflows. Thus, EcoFlow can efficiently reduce the average unit bandwidth cost.

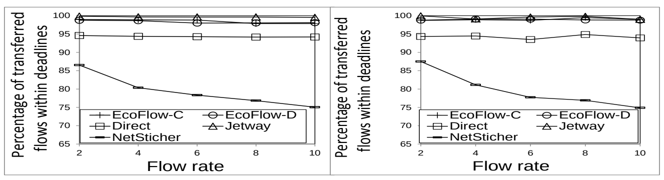
As shown in Figure 6(a) and Figure 6(b), we changed the flow arrival rates during a link's peak hours from 2 to 10 flows per hour and tested EcoFlow's performance with respect to average unit bandwidth cost. Figure 8(a) and Figure 8(b) plot the average unit bandwidth cost at the end of the 48-hour charging period at different flow rates in PlanetLab and EC2, respectively. We see that the average unit bandwidth cost generally drops when the flow arrival rate increases from 2 to 8 flows per hour, and it then keeps stable when the flow arrival rate increases from 8 to 10 flows per hour. This is due to the reason that when a larger number of videos are transmitted between datacenters during the charging period, the links connected to the datacenters have higher utilization and more videos are sent by using current charging volume, which reduces the bandwidth payment cost per video unit. However, when the flow arrival rate reaches a specific point (8 flows per hour in this case), the links' available bandwidth capacities are overutilized and they need to increase the charging volumes in order to transmit higher rates of video flows. Thus, the average unit bandwidth cost remains stable. The relative performance of different methods in Figure 8(a) and Figure 8(b) concurs with that in Figure 6(a) and Figure 6(b) due to the same reason.

Figure 9(a) and Figure 9(b) show the average percentage of flows transmitted within the current charging volume at different time intervals in PlanetLab and EC2, respectively. If a large portion of the flows are transmitted by utilizing the current charging volume, a link's charging volume will not further increase. An effective flow scheduler should provide



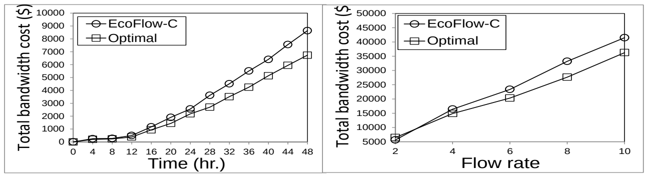
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 12: Percentage of transferred flows within deadlines at different time intervals.



(a) Results on PlanetLab. (b) Results on EC2.

Fig. 13: Percentage of transferred flows within deadlines at different flow rates.



(a) Different time intervals. (b) Different flow rates.

Fig. 11: Comparison of the total bandwidth cost between EcoFlow-D and the optimal solution.

a large percentage of flows transmitted within the current charging volume, so that the bandwidth cost during current time interval will not further increase. We see that performance of different methods with respect to average percentage of flows transmitted within the charging volume follows: EcoFlow-C>EcoFlow-D>JetWay>NetSticher>Direct. In JetWay and Direct, if a large number of video transfer requests arrive at a specific time interval, the videos will be transmitted immediately. And these videos are likely to result in high bandwidth usage at the time interval and increase the charging volume. NetSticher performs transmissions of delay-tolerant videos only during off-peak time, and the charging volume is likely to increase when a large number of non-delay-tolerant videos are transmitted during the peak hours. EcoFlow-C and EcoFlow-D aim to transfer flows within the charging volume by postponing the transmission of flows with late deadlines, thus yield the highest percentage of flows transmitted within the charging volume.

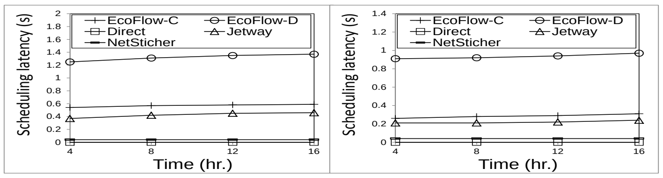
Figure 10(a) and Figure 10(b) show the average percentage of flows transmitted within the charging volume at different flow rates in PlanetLab and EC2, respectively. We see that as more videos need to transfer between datacenters hourly, smaller percentage of flows can be transmitted within the charging volume, i.e., the charging volumes on all links need to increase in order to accommodate higher flow rates. This is due to the reason that increased bandwidth is needed to transfer a large number of videos during each time interval and thus likely to increase the charging volume. As a result, a larger percentage of flows is likely to be transmitted by increased charging volume on the links. The relative performance of different methods mirrors that in Figure 9(a) and Figure 9(b) due to the same reason.

It is also interesting to check how close EcoFlow-C can achieve to the optimal. As we mentioned in Section 4.1, EcoFlow's bandwidth cost optimization problem is a convex problem, which can be solved optimally if we have the global information of all traffic flows (including their starting times, deadlines, and flow sizes) in the whole time span. However, the global traffic information is hard to predict at the beginning in reality. Hence, we conduct a simulation based on offline data, where we assume the global traffic information is known by the optimal approach. We create a larger network with 30 data centers, run the simulation for 20 times with Matlab [41], and take the average value of the total bandwidth cost of both EcoFlow-C and optimal solution. Figure 11(a) and Figure 11(b) compare the total bandwidth cost of EcoFlow-C and the optimal solution at different time slots and with different flow arrival rates, respectively. The results depicted in both figures show that the total bandwidth cost of EcoFlow-C is lightly higher than the optimal solution, which is not surprising as EcoFlow-C has to seek the local optimal in each period, and hence generates slightly lower performance than the optimal.

Figure 12(a) and Figure 12(b) show the percentage of video flows that are transferred within their deadlines across different time intervals in PlanetLab and EC2, respectively. Figure 13(a) and Figure 13(b) show the percentage of video flows that are transferred within their deadlines at different flow rates in PlanetLab and EC2, respectively. We see that the result follows: JetWay>EcoFlow-C>EcoFlow-D>Direct >NetSticher. NetSticher provides the least percentage of transferred videos within the deadlines due to the reason that it postpones the transmission of delay-tolerant videos from peak hours to off-peak hours, and if a link's available bandwidth capacity during the off-peak hours is not enough to transfer all waiting videos postponed from peak hours, a number of videos are likely to miss their transmission deadlines. Direct produces a higher percentage of transferred videos within the deadlines than NetSticher, due to the reason that a video begins transmission whenever the transfer request arrives at the source datacenter. EcoFlow and JetWay generate a comparably high percentage of transferred videos within the deadlines, as they both consider a video's transmission deadline when scheduling the video's transmission and aim to use the available bandwidth capacities from all links to finish the video's transmission before its deadline.

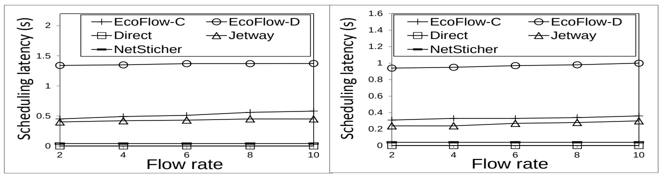
The average percentage of transferred flows within deadlines in EcoFlow-C is only 0.67% lower than that in Jetway; while the average percentage of transferred flows within deadlines in EcoFlow-D is only 0.87% lower than that in Jetway. This is due to link traffic volume prediction being potentially less accurate in EcoFlow. If the prediction is accurate, then EcoFlow will have the same results on meeting deadlines as JetWay. JetWay sends out videos immediately when the transmission requests arrive at the source datacenter and it transmits videos using sufficient bandwidth to guarantee that the videos are transmitted before their deadlines. On the other hand, EcoFlow may delay the transmission videos with late deadlines to avoid increasing the links' charging volumes. Because the link traffic volume prediction may not be accurate, when the actual traffic volume on a link is higher than the predicted traffic volume, the bandwidth capacity is insufficient to transmit all delayed videos and newly incoming videos, so some videos cannot be sent within their deadlines. Thus, EcoFlow generates a slightly higher percentage of transferred videos within the deadlines than JetWay.

We also tested the scheduling latency of different methods. Figure 14(a) and Figure 14(b) show the scheduling



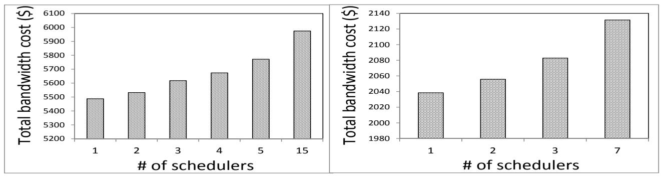
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 14: Avg. scheduling latency at different time intervals.



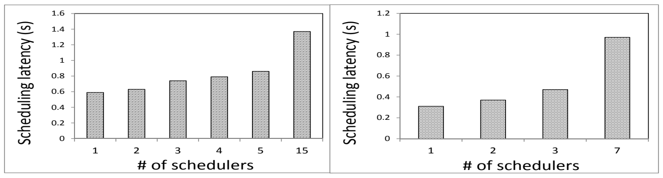
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 15: Average scheduling latency at different flow rates.



(a) Results on PlanetLab. (b) Results on EC2.

Fig. 16: Total bandwidth cost with different # of schedulers.



(a) Results on PlanetLab. (b) Results on EC2.

Fig. 17: Avg. scheduling latency with different # of schedulers.

latency from the 0th to 16th hour of the charging period in PlanetLab and EC2, respectively. Figure 15(a) and Figure 15(b) show the scheduling latency at the end of the 48 hour charging period at different flow rates in PlanetLab and EC2, respectively. We see that the scheduling latency is generally shorter on EC2 than on PlanetLab. It is because there are more datacenters on PlanetLab, so we need longer latency to calculate the available bandwidth capacities of all links and search alternating paths for indirect video flows when scheduling video flows. As Direct transfers video without scheduling, its scheduling latency is 0. We see that when time evolves or when the flow rate increases, the scheduling latency generally increases because the system needs to schedule a larger number of video flows. We also see that EcoFlow-D generates the highest scheduling latency (i.e., about 1.4 second latency). This is because each datacenter has its own scheduler, and schedulers need to communicate with each other in order to search alternating paths for indirect flows. EcoFlow-C reduces the scheduling latency of EcoFlow-D due to the reason that EcoFlow-D uses a centralized scheduler to avoid the communication overhead between different schedulers. JetWay generates a slightly shorter scheduling latency than EcoFlow-C because it does not delay the transmission of videos. NetSticher schedules the videos by postponing the delay-tolerant videos to off-peak hours, so its computation complexity is low and it generates shorter scheduling latency than JetWay. From these figures, we see that the scheduling latency of EcoFlow-C and EcoFlow-D is relatively short compared to the transmission time of videos.

5.2 Evaluations of Using Different Num of Schedulers

In this experiment, we used different numbers of schedulers to schedule video flows. Suppose there are y datacenters in the system and we used x schedulers, then each scheduler is responsible for scheduling video flows of $\lfloor y/x \rfloor$ or $\lfloor y/x \rfloor + 1$ data centers. Schedulers communicate with each other to finish the scheduling operation using EcoFlow-D. When there is only one scheduler in the system (i.e., centralized scheduler), the scheduling system is an implementation of EcoFlow-C; when the number of schedulers equals the number of datacenters, the scheduling system is an implementation of EcoFlow-D. Figure 16(a) and Figure 16(b) show the total bandwidth cost with different number of schedulers in PlanetLab and EC2, respectively. We see that as the number of schedulers increases, the total bandwidth cost increases gradually. This is due to the reason that the schedulers need to communicate with each other to identify a reroute path for IFs. When each scheduler is responsible for a smaller number of datacenters, it is more difficult to gain a full knowledge of all under-utilized links and less likely to find an optimal alternative path for routing indirect flows.

Figure 17(a) and Figure 17(b) show the average scheduling latency with different number of schedulers in PlanetLab and EC2, respectively. We see that the scheduling latency increases gradually as the number of schedulers increases due to the higher communication overhead between the schedulers that is required in order to find alternative paths for indirect flows.

5.3 Performance with Scheduler Failures and Rate Limiter Failures

In EcoFlow-D, we propose a distributed implementation of EcoFlow to deal with the single point of failure problem. When a scheduler or the rate limiter of a datacenter fails, the video flows sent from this datacenter are directly sent to their destination datacenters without applying the scheduling algorithm. In this experiment, we assume different numbers of schedulers or rate limiters fail and plot the total bandwidth cost in Figure 18. We see that when the number of failed schedulers or failed rate limiters increases from 0 to 3, the total bandwidth cost increases gradually. This is due to the reason that when a larger number of video flows are sent without applying the scheduling algorithm, they are likely to increase the links' charging volumes. As a large number of correlated failures are rare in datacenters and 95% of failures can be resolved in 10 min [42], scheduler failures or rate limiter failures will not significantly degrade the performance of our proposed EcoFlow system.

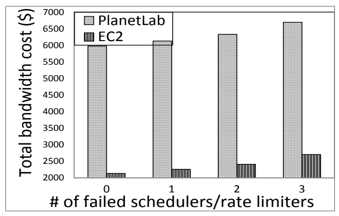
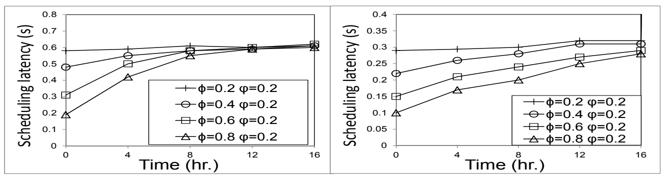


Fig. 18: Total bandwidth cost with different numbers of failed schedulers/rate limiters.

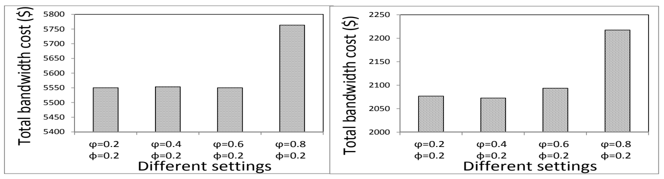
5.4 Effectiveness of Setting Initial Charging Volume

In this section, we tested the performance of both EcoFlow-C and EcoFlow-D when we set an initial charging volume at the beginning of the charging period according to Section 4.5. We set φ in Equation (18) to a fixed value of 0.2, and varied the value of ϕ from 0.2 to 0.8.



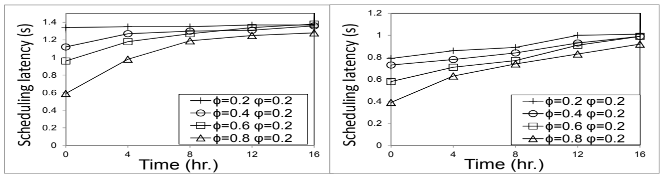
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 19: Average scheduling latency in EcoFlow-C.



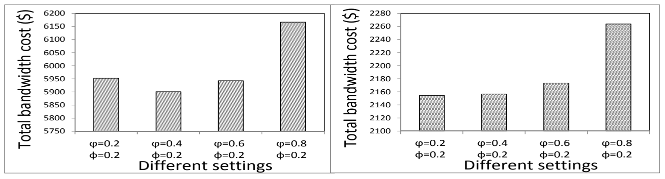
(a) Results on PlanetLab. (b) Results on EC2.

Fig. 20: Total bandwidth cost in EcoFlow-C.



(a) Results on PlanetLab. (b) Results on EC2.

Fig. 21: Average scheduling latency in EcoFlow-D.



(a) Results on PlanetLab. (b) Results on EC2.

Fig. 22: Total bandwidth cost in EcoFlow-D

Since the purpose of setting an initial charging volume is to reduce the scheduling latency during early time intervals of a charging period, we first present the average scheduling latency on all links. It is the time span from when a scheduler receives a sending queue of videos on a link until the time when the scheduler finished the scheduling of these video and updating the schedule table. Figure 19(a) and Figure 19(b) show the average scheduling latency of EcoFlow-C from the 0th to the 16th hour of the charging period in PlanetLab and EC2, respectively. We observe that the scheduling latency is generally shorter on EC2 than on PlanetLab because there are more datacenters on PlanetLab, so EcoFlow-C needs longer latency to calculate the available bandwidth capacities of all links and search alternating paths for indirect video flows when scheduling video flows. We also see that the scheduling latency gradually increases with time. This is because when few videos are pending at early time intervals, a large portion of videos can be transmitted directly by using available capacities of direct links, so the scheduling latency is short since the scheduler does not need to search alternating paths for these videos. As more video flows are transmitted in the network, available bandwidth capacities of some links are used up and videos on these links need to be split and rerouted to other links, so the scheduler needs longer latency to update the scheduling table. We also see that larger value of ϕ leads to shorter scheduling latency. According to Equation (18), large value of ϕ leads to large initial charging volume and high available bandwidth capacities on the links. The scheduling latency is short because most videos can be transmitted through direct links. On the other hand, small value of ϕ leads to longer scheduling latency as the scheduler needs to search alternating paths for some videos that are not able to be transmitted on direct links.

The negative effect of large value of ϕ is that it may lead to underutilization of the initial charging volume, and the bandwidth cost is not minimized at the end of the charging period. To further evaluate the effect of initial charging volume in bandwidth cost reduction, we then plot total bandwidth cost at the end of the 48 hour charging period in Figure 20(a) and Figure 20(b). We see that higher value of ϕ generally leads to higher bandwidth cost due to the reason that smaller charging volume may be adequate in transmitting all video flows. Therefore, it is important to determine an appropriate initial charging volume to reduce scheduling latency while constraining bandwidth cost.

We then evaluate the effectiveness of setting initial charging volume in EcoFlow-D. Figure 21(a) and Figure 21(b) show the average scheduling latency of EcoFlow-D from the 0th to 16th hour of the charging period in PlanetLab and EC2, respectively. Compared to Figure 19(a) and Figure 19(b), we observe that EcoFlow-D generates higher scheduling latency due to the reason that in EcoFlow-D. Figure 22(a) and Figure 22(b) show total bandwidth cost for EcoFlow-D at the end of the 48 hour charging period in PlanetLab and EC2, respectively. We see that the experimental results concur with that in Figure 20(a) and Figure 20(b) due to the same reason.

6 Conclusions

To provide video streaming services to users across different regions, cloud providers need to transfer video contents between different datacenters. These inter-datacenter transfers are charged by ISPs under percentile-based charging models. We take advantage of this particular characteristic of these models and propose EcoFlow to minimize cloud providers' payment costs on inter-datacenter traffics. EcoFlow is an economical and deadline-driven video transfer strategy. It first estimates the total volume of video traffic needed to be transmitted between any two datacenters within a time period, compares it with the charging volume and calculates the under-utilized traffic volume on each link. EcoFlow then schedules video flows with the objective that these flows do not incur additional charges on the link, guaranteeing that each video flow meets its transmission deadline. Finally, the under-utilized links with low traffic burden are used to build alternating paths for video flows that are estimated to miss their deadlines. To enhance EcoFlow, we also propose setting each link's initial charging volume to reduce the scheduling latency at the beginning of the charging period. We further discuss how to deal with link available bandwidth prediction errors and lack of prior knowledge of the charging volume. Moreover, we design the implementation of EcoFlow in both centralized and distributed manner. Experimental results on PlanetLab and EC2 show the effectiveness of EcoFlow in reducing bandwidth costs while guaranteeing that each video flow meets its transmission deadline for inter-datacenter video transfers. We will study a more tractable formulation of this bandwidth cost optimization problem in the future. In addition, considering the uncertainty of video flows over time, we will apply more sophisticated approaches like stochastic optimization, Lyapunov, or online algorithms to further improve the performance of our design. Also, we will study how to automatically generate the deadline that satisfies users when it is not indicated.

Acknowledgements

This research was supported in part by U.S. NSF grants OAC-1724845, ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proceedings of ACM Multimedia 2015 [43]. We thank Dr. Yuhua Lin and Dr. Liuhua Chen for their valuable work on this paper.

References

[1] A. Khanafer, M. Kodialam, and K. Puttaswamy. To rent or to buy in the presence of statistical information: The constrained ski-rental problem. *TON*, 23(4):1067–1077, 2014.

[2] S. Rajani and T. Rajender. Literature review: Cloud computing-security issues, solution and technologie. *International Journal of Engineering Research*, 3(4):221–225, 2014.

[3] V. Adhikari, Y. Guo, F. Hao, V. Hilt, Z. Zhang, M. Varvello, and M. Steiner. Measurement study of netflix, hulu, and a tale of three cdns. *TON*, 1(99):1–10, 2014.

[4] X. Liao, L. Lin, G. Tan, H. Jin, X. Yang, W. Zhang, and B. Li. Liverender: A cloud gaming system based on compressed graphics streaming. *TON*, 1(99):1–10, 2015.

[5] V. K. Adhikari, Y. Guo, F. Hao, M. Varvello, V. Hilt, M. Steiner, and Z. Zhang. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *Proc. of INFOCOM*, 2012.

[6] Auto scaling in the amazon cloud, http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html, [accessed in sep. 2015].

[7] Q. Zhu and G. Agrawal. Resource provisioning with budget constraints for adaptive applications in cloud environments. In *Proc. of HPDC*, 2010.

[8] F. Wang, J. Liu, M. Chen, and H. Wang. Migration towards cloud-assisted live media streaming. *TON*, 1(99):1–10, 2014.

[9] Y. Wu, C. Wu, B. Li, and L. Zhang. Scaling social media applications into geo-distributed clouds. *TON*, 23(3):689–702, 2015.

[10] H. Xu and B. Li. Joint request mapping and response routing for geo-distributed cloud services. In *Proc. of INFOCOM*, 2013.

[11] Y. Feng, B. Li, and B. Li. Jetway: Minimizing costs on inter-datacenter video traffic. In *Proc. of Multimedia*, 2012.

[12] D. Goldenberg, L. Qiu, H. Xie, Y. Yang, and Y. Zhang. Optimizing cost and performance for multihoming. In *Proc. of SIGCOMM*, 2004.

[13] N. Laoutaris and P. Rodriguez. Good things come to those who (can) wait or how to handle delay tolerant traffic and make peace on the internet. In *Proc. of HotNets-VII*, 2008.

[14] N. Laoutaris, G. Smaragdakis, P. Rodriguez, and R. Sundaram. Delay tolerant bulk data transfers on the internet. In *Proc. of SIGMETRICS*, 2009.

[15] L. Nikolaos, S. Michael, X. Yang, and R. Pablo. Inter-datacenter bulk transfers with netstitcher. In *Proc. of SIGCOMM*, 2011.

[16] Z. Zhang, M. Zhang, A. Greenberg, Y. Hu, R. Mahajan, and B. Christian. Optimizing cost and performance in online service provider networks. In *Proc. of NSDI*, 2010.

[17] H. Wang, H. Xie, L. Qiu, A. Silberschatz, and Y. Yang. Optimal isp subscription for internet multihoming: algorithm design and implication analysis. In *Proc. of INFOCOM*, 2005.

[18] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, 2009.

[19] A. Hussam, P. Lonnie, and W. Hakim. Racs: A case for cloud storage diversity. In *Proc. of SoCC*, 2010.

[20] Service Level Agreements. http://azure.microsoft.com/en-us/support/legal/sla/, [Accessed in Sep. 2015].

[21] Amazon S3. http://aws.amazon.com/s3/, [Accessed in Sep. 2015].

[22] A. Ashraf, F. Jokhio, T. Deneke, S. Lafond, I. Porres, and J. Lilius. Stream-based admission control and scheduling for video transcoding in cloud computing. In *Proc. of CCGrid*, 2013.

[23] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[24] A. Dhamdhere and C. Dovrolis. Isp and egress path selection for multihomed networks. In *Proc. of INFOCOM*, 2006.

[25] R. Van Der, S. Boele, F. Dijkstra, A. Barczyk, G. van Malenstein, H. Chen, and J. Mambretti. Multipathing with mptcp and openflow. In *Proc. of SCC*, 2012.

[26] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *Proc. of NSDI*, 2010.

[27] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review*, 41(4):254–265, 2011.

[28] C. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.

[29] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 435–446, 2013.

[30] J. Tomlin. Minimum-cost multicommodity network flows. *Operations Research*, 14(1):45–51, 1966.

[31] J. Lucas and M. Saccucci. Exponentially weighted moving average control schemes: Properties and enhancements. *Technometrics*, 32(1):1–29, 1990.

[32] Q. Xu, D. Cheng, and Y. Fu. Traffic feature distribution analysis based on exponentially weighted moving average. In *Prof. of the IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, 2012.

[33] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. Madhyastha. Spanstore: cost-effective geo-replicated storage spanning multiple cloud services. In *Proc. of SOSP*, 2013.

[34] J. Douceur, J. Mickens, T. Moscibroda, and D. Panigrahi. Collaborative measurements of upload speeds in p2p systems. In *Proc. of INFOCOM*, 2010.

[35] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proc. of NSDI*, 2011.

[36] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: trading a little bandwidth for ultra-low latency in the data center. In *Proc. of NSDI*, 2012.

[37] K. Xu, M. Zhang, J. Liu, Z. Qin, and M. Ye. Proxy caching for peer-to-peer live streaming. *Computer Networks*, 2010.

[38] Y. Huang, T. Fu, D. Chiu, J. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p-vod system. In *Proc. of SIGCOMM*, 2008.

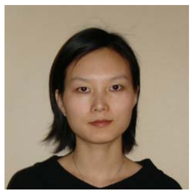
[39] PlanetLab. http://www.planet-lab.org/, [Accessed in Sep. 2015].

[40] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/, [Accessed in Sep. 2015].

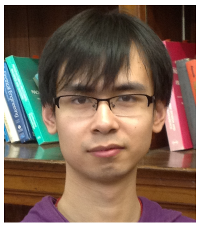
[41] Matlab. https://www.mathworks.com/products/optimization.html, [Accessed in Feb. 2018].

[42] A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM computer communication review*, volume 39, pages 51–62, 2009.

[43] H. Shen Y. Lin and L. Chen. Ecoflow: An economical and deadline-driven inter-datacenter video flow scheduling system, short paper. In *Proc. of ACM Multimedia*, 2015.



Haiying Shen received her BS degree in Computer Science and Engineering from Tongji University, China in 2000, and MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the CS Department at the University of Virginia. Her research interests include distributed computer systems and computer networks, cloud computing and cyber-physical systems. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE and a member of the ACM.



Chenxi Qiu received his BS degree in Telecommunication Engineering from Xidian University, China, in 2009 and Ph.D. degree in Electrical and Computer Engineering in Clemson University in 2015. He currently is a Postdoc scholar in the College of Information and Science at Pennsylvania State University, PA, United States. His research interests include cyber security, cyber physical systems, and cloud computing.