Delay Optimization for Multi-source Multi-channel Overlay Live Streaming

Jie Dai Zhangyu Chang S.-H. Gary Chan
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{jdaiaa,zchang,gchan}@cse.ust.hk

***Abstract*—In order to provide scalable live streaming service, a content provider often deploys an overlay cloud consisting of distributed servers which collaboratively exchange streams with each other. We consider an overlay consisting of multiple live channels originating from multiple sources. Server bandwidth and end-to-end network bandwidth are shared among these channels. The stream of each channel is divided into multiple substreams of a certain bitrate, each of which is pushed via a tree to the servers that subscribe to the channel. The critical and challenging issue is then how to optimize the topology of the substream trees so as to minimize the maximum channel delay (defined as the delay from the source to the subscribing servers).**

There has been little work on the optimization of such multi-source multi-channel live streaming network. We first formulate the problem which comprehensively and realistically captures various delay and bandwidth components, and show that it is NP-hard. We then propose an efficient algorithm called COMMOS (Collaborative Multi-source Multi-channel Streaming Overlay) which achieves low channel delay. Extensive simulation results based on real Internet topologies show that COMMOS outperforms other state-of-the-art schemes by a wide margin (often by more than 40%), due to its better utilization of network resources.

***Index Terms*—live streaming, overlay routing, optimization, substream push**

I. Introduction

In order to deliver live contents in a scalable manner, a content provider often deploys distributed servers close to user pools. These servers form a collaborative overlay network, with a certain bandwidth capacity between any pair of them (which can be zero) [1]. A *source* is where a live *channel* originates, and users may subscribe the channel via their connected servers, the so-called "home" servers. The servers subscribe the channels requested by users, get the stream, and deliver the content to the users. To reduce the end-to-end streaming delay, the channels are *pushed* from their sources to the subscribing servers. With the improvement in edge bandwidth, we consider that the bandwidth between servers and users is not the bottleneck. Therefore, we focus on the overlay consisting of only the sources and servers in this paper, and study how to minimize the delay from the sources to the subscribing servers.

We consider a multi-source multi-channel live overlay streaming with multiple channels and multiple sources, where each of the sources originates one or more channels. Each server may subscribe to an arbitrary set of the channels (may be empty, in which case the server does not have users). The channels may have heterogeneous streaming rate. In order to better utilize network bandwidth, the full stream of a channel is divided into a certain number of substreams of a certain bitrate. The substreams are pushed from the source to the subscribing servers in the overlay via multiple trees, such that the subscribing servers have to receive all the substreams of the channel, i.e., the full stream, before the channel can be played back. Clearly, the aggregation of all the substream trees of a channel is a *mesh*. Without loss of generality, we consider that the sources subscribe to no channels and hence act as the roots of the substream trees of their channels. [1] Furthermore, a channel can only originate from one source. [2]

In order to more cost-effectively utilize network resources, the end-to-end network bandwidth between servers are *shared* among all the channels, i.e., there is no bandwidth partitioning for each channel. Furthermore, the server upload bandwidth to the other servers is also *shared*, i.e., its bandwidth may be used by other channels which it has not subscribed. In other words, the server may act as a *helper* by receiving and forwarding substreams of the other unsubscribed channels.

In live streaming, the end-to-end delay of a channel, or simply channel delay, is an important optimization objective. Such delay mainly consists of two components, the propagation delay and the scheduling delay. *Propagation delay* is the time that a packet travels from one server to the next over a physical connection, usually reflected by the round-trip time (RTT). *Scheduling delay* is defined as the time elapsed from a parent node fully received a packet to the time instant that the packet is fully departed from the node for its neighbor. We consider both delay components here, as they accumulate with the increase of the number of channels, servers and the hop counts from the source. Our objective is to minimize the maximum delay of all the channels.

This work was supported, in part, by Hong Kong Research Grant Council (RGC) General Research Fund (610713) and HKUST (FSGRF13EG15).

[1] If not, we may split the node into two parts, one being the "source" with all the channels and the other being the "server" subscribing to the channels. The two parts are then connected by a link of infinite bandwidth capacity and zero delay.

[2] If not (i.e., the channel has multiple sources), we may form a "virtual" source connecting to all the sources as its first-hop neighbors with infinite bandwidth and zero delay. In other words, the sources become the first-hop servers subscribing the channel originating at the virtual source.

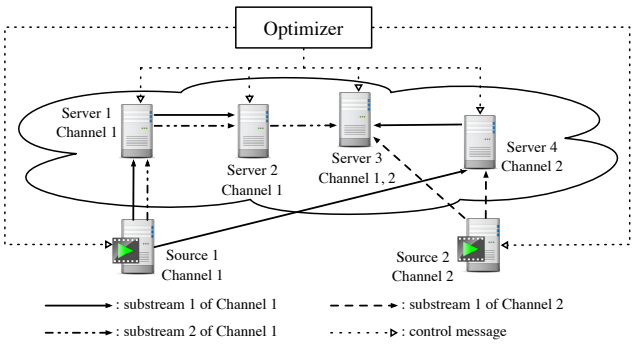

Fig. 1. A multi-source multi-channel overlay live streaming network with server collaboration.

We show in Figure 1 an example of the streaming network under consideration. Source 1 and Source 2 are streaming channels 1 and 2, respectively, where Channel 1 has two substreams (higher bitrate) while Channel 2 has only one (lower bitrate). Server 4 does not subscribe to Channel 1 but acts as its helper: it receives a substream from Source 1 and forwards it to Server 3. In this way, the bandwidth bottleneck between Server 2 and Server 3 is overcome by such multi-path streaming. Topology optimization is carried out by a central optimizer, which continuously sends control messages to sources and servers to collect network information. Based on such information and network capacity constraints, it computes the substream trees and informs the servers of the optimized topology.

The live overlay network we consider is general and realistic. It uniquely considers substreams and its optimization under such setting. Such dividing each channel into multiple substreams leads to better utilization of network bandwidth as compared with the single-stream approach. The collaboration of servers and bandwidth sharing increase the reservoir of available bandwidth in the network, and provide rich path diversity to overcome bandwidth bottlenecks to achieve low delay.

There has been little work on multi-source multi-channel overlay live streaming. We study delay optimization problem for such network through substream tree construction. Our contributions are as follows:

- *Problem formulation and complexity analysis:* We present a realistic delay model for multi-source multi-channel overlay live streaming with collaborative servers, which captures various network components and constraints. We formulate the optimization problem of constructing multiple substream trees, which is to minimize the maximum delay of the channels while meeting the constraints. We analyze its complexity and prove that it is NP-hard.
- *COMMOS: A novel algorithm for low-delay multi-source multi-channel overlay live streaming:* We propose an efficient and implementable heuristic termed COMMOS (**Co**llaborative **M**ulti-source **M**ulti-channel **O**verlay **S**treaming). It achieves overall low streaming delay while meeting bandwidth capacity constraints by constructing multiple substream trees.
- *Extensive simulation results:* We conduct extensive simulation study based on real Internet topologies to evaluate the performance of our COMMOS, and show that server collaboration can significantly reduce the streaming delay while meeting high streaming rate.

We briefly review related work here. While much work has been done on peer-to-peer (P2P) live streaming, most of them [2]–[7] focus on single-source single-channel overlay streaming. These works cannot be extended to multi-source multi-channel live streaming due to the shared resources such as link capacity and server upload bandwidth. Though some works have been done on multi-channel P2P live streaming [8]–[12], they consider different objective functions, such as maximizing bandwidth to support more users or minimizing the load of servers through collaboration of users. Few have addressed how to optimize the overall streaming delay, which is an important metric for user experience in live streaming.

This paper is organized as follows. We first formulate the optimization problem and prove its NP-hardness in Section II. We then present the Collaboration Multi-source Multi-channel Streaming Overlay (COMMOS) algorithm that constructs the streaming overlay network in Section III. Illustrative simulation results and comparison with other state-of-the-art schemes are presented in Section IV. We conclude in Section V.

II. Problem Formulation and Complexity Analysis

We present our optimization problem, Minimum-Delay Streaming with Server Collaboration, in Section II-A. We show the NP-hardness of the problem in Section II-B.

A. The Minimum-Delay Streaming with Server Collaboration Problem

We model the overlay as a directed graph $G = (V, E)$, where V is the set of vertices corresponding to the *node indices* of both servers and sources. Let S be the set of sources and P be the set of servers, where $V = S \cup P$. Let $E \subseteq V \times V$ be the set of possible overlay connections between nodes in V. Note that the graph does not have to be complete.

Let M be the set of channels. Each channel is divided into multiple substreams of similar bandwidth. We consider that all the bandwidth (server upload bandwidth and end-to-end network capacity) is normalized by some unit denoted as τ (e.g., $\tau = 400$ kb/s). Let $K^{(m)}$ be the set of substream indices for channel $m \in M$, and $|K^{(m)}|$ be its cardinality, i.e., the number of substreams for channel m. For every node $i \in P$, let M_i be the set of channels that node i subscribes to.

Each substream is delivered to all the subscribed nodes in P by a delivery tree, and hence there are a total of $\sum_{m \in M} |K^{(m)}|$ delivery trees. Denote the delivery tree of the kth substream of channel m as $T^{(mk)}$.

For every node $i \in V$, it has an upload capacity of $U_i \in \mathbb{Z}^+$ units which is normalized by τ. Therefore, U_i represents the maximum total number of children it can serve in all substream trees. Note that each source server $l \in S$ has an

upload capacity of U_l units and has no parent. For each edge $\langle i,j\rangle \in E$, there is an end-to-end capacity $b_{ij} \in \mathbb{Z}^+$ units and a bandwidth $t_{ij} \in \mathbb{N}$ units normalized by τ. In other words, b_{ij} is the maximum number of substreams that can simultaneously accommodate in edge $\langle i,j\rangle$, and t_{ij} is the concurrent number of substreams in edge $\langle i,j\rangle$. Note that as t_{ij} is bounded by both the link capacity of edge $\langle i,j\rangle$ and the upload capacity of node i, t_{ij} has to satisfy

$$0 \le t_{ij} \le \min(b_{ij}, U_i) \tag{1}$$

and

$$\sum_{j \in C_i} t_{ij} \le U_i, \tag{2}$$

where C_i is the set of the children of node i in all the substream trees.

For the streaming session to be feasible, the network has to satisfy the following constraints:

1) The total upload capacities must be larger than the total streaming bandwidth:

$$\sum_{i \in V} U_i \ge \sum_{i \in P} \sum_{m \in M_i} |K^{(m)}|. \tag{3}$$

2) The aggregate incoming capacities of each node must be larger than the streaming rate:

$$\sum_{i \in V} \min(b_{ij}, U_i) \ge \sum_{m \in M_j} |K^{(m)}|, \quad \forall j \in P. \tag{4}$$

For any node $i \in P$, if node i receives all substreams in $K^{(m)}$, it is *fully served* for channel m and can play back the channel m with continuity. Let $a_i^{(mk)}$ be an indicator variable indicating whether node i is in tree $T^{(mk)}$. $a_i^{(mk)} = 1$ if $i \in T^{(mk)}$; otherwise $a_i^{(mk)} = 0$. Therefore, node i is *fully served* for channel m if and only if $a_i^{(mk)} = 1$ for every $k \in K^{(m)}$.

Denote the propagation delay of edge $\langle i,j\rangle$ as d_{ij}^{Prp} and the worst-case scheduling delay of node i as d_i^{Sch}. d_i^{Sch} is given by

$$d_i^{\mathrm{Sch}} = \sum_{j \in C_i} \frac{L \cdot t_{ij}}{\min(b_{ij}, U_i)\tau}, \tag{5}$$

where L (bits) is the segment size used in streaming.

Denote the source-to-end delay of node j in substream tree $T^{(mk)}$ as $D_j^{(mk)}$, which equals to the delay of its parent i in tree $T^{(mk)}$ plus the scheduling delay of i and propagation delay between i and j, i.e., $D_j^{(mk)} = D_i^{(mk)} + d_i^{\mathrm{Sch}} + d_{ij}^{\mathrm{Prp}}$.

Denote the source-to-end delay of node i of channel m as $D_i^{(m)}$. It equals to the maximum source-to-end delay of node i in all the substream tree $T^{(mk)}$, $\forall k \in K^{(m)}$, i.e., $D_i^{(m)} = \max_{k \in K^{(m)}} D_i^{(mk)}$.

The channel delay $D^{(m)}$ of channel m is defined by its maximum source-to-end delay of channel m for all the nodes i that has subscribed to channel m, i.e., $D^{(m)} = \max_i D_i^{(m)}$.

The streaming diameter D is the maximum channel delay in M, i.e., $D = \max_{m \in M} D^{(m)}$.

We finally state below the problem under study:

Minimum-Delay Streaming with Server Collaboration (MDSSC) problem: The MDSSC problem is to find an overlay of substream deliver trees $\{T^{(mk)}\}$ which minimizes the maximum delay (i.e., minimizes the streaming diameter),

$$\min D = \min \max_{m \in M} D^{(m)}, \tag{6}$$

subject to the subscription requirement $a_i^{(mk)} = 1 \; \forall i \in P, m \in M_i, k \in K^{(m)}$ (i.e., each server $i \in P$ receives all substreams $k \in K^{(m)}$ for each channel $m \in M_i$), and the bandwidth requirements given in Equations (1) to (4).

B. Complexity Analysis

The complexity of this MDSSC problem is NP-hard since the Travelling Salesman Problem (TSP) is reducible to MDSSC problem in polynomial time. Let $G'(V', E')$ be an instance of TSP. We transform the instance of TSP $G'(V', E')$ to an instance of MDSSC $G(V, E)$ by adding a vertex S_{end} and edges from all vertices to S_{end}. The vertices in V represent servers and the weights on the edges are the propagation delay plus the transmission delay between two adjacent servers. In this particular instance, there is only one streaming source existing in the network. Let S_{src} be the source, and consider the special case that the streaming rate is 1 unit of the substream and upload capacity of each server is also 1 unit except that S_{end} has zero upload capacity. The resulting overlay topology must be a chain starting from the streaming source S_{src} to S_{end}. The streaming diameter D is equal to the delay at S_{end}, which is the sum of all the delays along the chain. Hence, D in G is the minimum if and only if the cost of the Hamiltonian cycle is the minimum. Hence, TSP is polynomial-time reducible to MDSSC.

III. COMMOS: Collaborative Multi-source Multi-channel Overlay Streaming

In this section, we first present COMMOS (Collaborative Multi-source Multi-channel Overlay Streaming), which constructs a streaming overlay given network information on server distance and bandwidth capacities. Due to the NP-hardness of the problem, the algorithm is heuristic in nature, which is to construct delivery trees for each substream and involve helpers if necessary to meet the streaming bitrate requirement. We present the preliminary in Section III-A, and the algorithmic details in Section III-B.

A. Preliminary

Let $r_{ij} \in \mathbb{N}$ be the residual traffic that can be transmitted from i to $\langle i,j\rangle$, which is given by $r_{ij} = b_{ij} - t_{ij}$. Recall that t_{ij} is the number of concurrent substreams and b_{ij} is the end-to-end capacity on edge $\langle i,j\rangle$.

Let $R_i^{\mathrm{out}} \in \mathbb{N}$ be the residual uploading capacity that node i can stream out, which is given by $R_i^{\mathrm{out}} = U_i - \sum_{j \in C_i} t_{ij}$. A node i can be a *potential parent* and serve node j if $R_i^{\mathrm{out}} > 0$ and $r_{ij} > 0$.

A node i can be a *potential helper* for other channel if it has surplus incoming capacity to serve the others after it fulfils its subscriptions. Denote $R_i^{\mathrm{in}} \in \mathbb{Z}$ as the residual incoming

capacity that node i can receive beyond its own subscription requirement, which is defined by

$$R_i^{\text{in}} = \sum_{\langle j,i\rangle \in E} b_{ji} - \sum_{m \in M_i} K^{(m)}. \quad (7)$$

Node i is a valid candidate of being a helper if $R_i^{\text{in}} > 0$.

Define the *subtree delay* of node i as the maximum delay of all the nodes in all the subtrees rooted at node i. It is given by

$$\delta_i = \max_{j \in Q_i, m \in M_i, k \in K^{(m)}} D_j^{(mk)}, \quad (8)$$

where Q_i denotes the set of all the descendant nodes of i in all substream trees.

A node j will incur an increment to the streaming diameter when it joins tree $T^{(mk)}$ via node i and we denote such increment as $\Delta D_{ij}^{(mk)}$.

Algorithm 1: COMMOS

```
Initialize the overlay parameters;
while there is unsatisfied subscription do
    (ΔD_ij^(mk), i, j, h, m, k) ← findMinDInc(P, M, K);
    if h = ∅ then
        Include server j into T^(mk) via i;
    else
        Include server j, helper h into T^(mk) via i;
    end
    Update the overlay parameters;
end
```

B. Algorithmic Details and Complexity

We construct $\sum_{m \in M} |K^{(m)}|$ delivery trees through iterations. Delivery trees are expanded from the source to the destinations. In each iteration, we add one server into one partially constructed delivery tree.

We outline COMMOS in Algorithm 1. We initialize residual uploading capacity R_i^{out}, residual incoming capacity R_i^{in} for each node $i \in V$ and the residual traffic r_{ij} for each edge $\langle i,j\rangle \in E$. Each delivery tree, or $T^{(mk)}$, is initialized containing only the streaming source $S^{(m)} \in S$ and the streaming diameter, D, is initialized as zero.

There are totally $\sum_{i \in p} \sum_{m \in m_i} |K^{(m)}|$ nodes to be joined into all the delivery trees. We loop through $\sum_{i \in p} \sum_{m \in m_i} |K^{(m)}|$ iterations and add a server into a delivery tree in each iteration. For a node j, we find the delivery $T^{(mk)}$ such that it increases the streaming diameter least when it joins $T^{(mk)}$ via node i.

Algorithm 2 states the process to find a node j for tree $T^{(mk)}$ with the least diameter increment. For each tree $T^{(mk)}$, there are two ways for node j to join it via i:

1) *directly connect a node i in tree $T^{(mk)}$:* its potential source-to-end delay is

$$D_j^{(mk)} = D_i^{(mk)} + d_{ij}^{\text{Prp}} + d_i^{\text{Sch}}, \quad (9)$$

if $r_{ij} > 0$ and $R_i^{\text{out}} > 0$. This new connection incurs an increment to the streaming diameter, denoted as $\Delta D^{(mk)}(i,j)$. We calculate this increment delay as

$$\Delta D^{(mk)}(i,j) = \max\left(\delta_i + \Delta d_i^{\text{Sch}}, D_j^{(mk)}\right) - D, \quad (10)$$

where Δd_i^{Sch} is the increment of the scheduling delay of node i.

2) *Connect node i through a potential helper node h:* Note that node $h \notin T^{(mk)}$ and $m \notin M_h$ and it has enough residual capacities to receive and server others. The potential source-to-end delay of node j is:

$$D_j^{(mk)} = D_i^{(mk)} + d_{ih}^{\text{Prp}} + d_i^{\text{Sch}} + d_{hj}^{\text{Prp}} + d_h^{\text{Sch}}, \quad (11)$$

if $r_{ih}, r_{hj}, R_i^{\text{out}}, R_h^{\text{out}}, R_h^{\text{in}} > 0$. We calculate the increment to the streaming diameter of node i through helper h, $\Delta D^{(mk)}(i,j,h)$ as

$$\max\left(\delta_i + \Delta d_i^{\text{Sch}}, \delta_h + \Delta d_h^{\text{Sch}}, D_j^{(mk)}\right) - D. \quad (12)$$

Therefore, if $R_i^{\text{out}} > 0$, the increment to streaming diameter of node i serving node j, denoted as $\Delta D_{ij}^{(mk)}$, is given by

$$\min\left(\Delta D^{(mk)}(i,j), \min_{h \in H} \Delta D^{(mk)}(i,j,h)\right), \quad (13)$$

where H is the set of *potential helpers*.

For each iteration, we choose the connection $\langle i,j\rangle$ incurring the lowest increment to the streaming diameter and connect node j to the corresponding tree $T^{(mk)}$, i.e.,

$$\arg_{i,j,m,k} \min \Delta D_{ij}^{(mk)}. \quad (14)$$

Then, we add the connection $\langle i,j\rangle$ into tree $T^{(mk)}$ to the overlay and update the networking parameters and streaming diameter.

The complexity of the COMMOS algorithm is $O(|M|^2|P|^3|V|)$ where $|M|$ is the number of channels, $|P|$ is the number of servers and $|V|$ is the number of sources and servers. In the tree construction step, adding one server into one delivery tree takes $O(|M||P|^2|V|)$ time and the calculation of the increment of streaming diameter of the influenced servers takes $O(|M||V|^2)$; hence, each step takes $O(|M||P|^2|V|)$ time. There are $O(|M||P|)$ iterations in total. Therefore the tree construction takes $O(|M|^2|P|^3|V|)$ time.

IV. Illustrative Simulation Results

In this section, we first present our simulation environment and performance metrics in Section IV-A. Then we discuss illustrative simulation results in Section IV-B.

A. Simulation Setup and Performance Metrics

We have implemented the simulation of COMMOS using Python. The simulation is carried out on a real Internet topology provided by CAIDA, which was collected on June 12th, 2011. The round trip times (RTTs) between inter-connected routers are also given in the topology. In underlay routing,

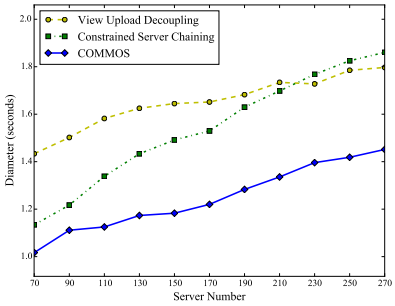

Fig. 2. Diameter versus server number.

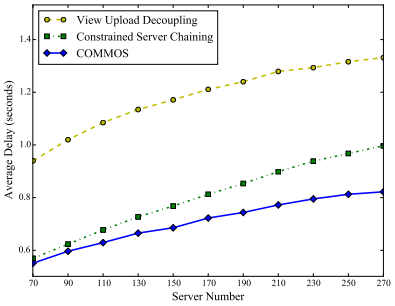

Fig. 3. Average delay versus server number.

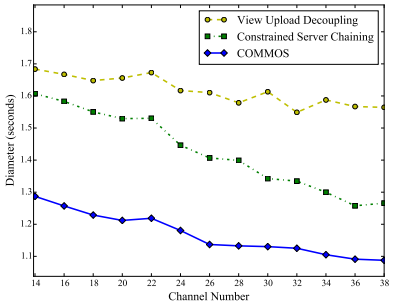

Fig. 4. Diameter versus channel number.

Algorithm 2: findMinDInc(P, M, K)

foreach $j \in P, m \in M_j, k \in K^{(m)}$ **do**
  foreach *potential parent* $i \in T^{(mk)}$ **do**
    $\Delta D_{ij}^{(mk)} \leftarrow \Delta D^{(mk)}(i, j)$;
    foreach *potential helper* $h \in H$ **do**
      $\Delta D_{ij}^{(mk)} \leftarrow \min(\Delta D_{ij}^{(mk)}, \Delta D^{(mk)}(i, j, h))$;
    end
  end
end
return $(\Delta D_{ij}^{(mk)}, i, j, h, m, k)$ correspond to $\Delta D_{ij}^{(mk)}$;

we use distance-vector to compute the latencies between any two router nodes in the network. To generate the simulation environment, sources and servers are randomly attached to the router nodes in the live streaming network. Without loss of generality, the link capacity between servers is generated by normal distribution (accepting only the positive values). Channels are randomly assigned to the sources. Each server subscribes to a channel based on the Zipf's law, i.e., the probability that a subscription for the nth most popular channel is $n^{-z} / \sum_{i=1}^{|M|} i^{-z}$, recalling that $|M|$ is the number of channels. (Note that COMMOS can be applied to any network environment. For concreteness in our simulation, we use the above settings as an example.) Unless otherwise stated, we use the following parameters in our simulation: number of servers = 150, number of channels = 20, the full streaming rate is 1.2 Mbps with a substream as 400 kbps, the segment size = 100 kbits, server upload capacity $U_i = 6.5$ Mbps, the mean and standard deviation of link capacity are $\mu = 4$ Mbps and $\sigma = 1$ Mbps, the Zipf shape parameter $z = 0.4$.

We compare the performance of COMMOS with the following state-of-the-art schemes:

- *Constrained Server Chaining (CSC)* [7], whose objective function is to minimize the source-to-end delay. This is a pull-based algorithm. A server requests live streams from those servers providing the content. No collaboration is involved in this scheme.
- *Video-Upload Decoupling (VUD)* [10], a P2P multi-channel overlay algorithm. With minor modifications, we can easily adapt this algorithm into our network setting. Servers are fully collaborative: each one is a helper in this scheme. A server receives and provides channels independent of what it subscribes to.

We evaluate the performance of our proposed algorithm mainly by several *delay* metrics. The *streaming diameter* is the maximum source-to-end delay of all the subscribed channels among all servers in the network. The *average delay* is the average of the channel delays in the network. Besides these overlay diameters, we are also interested in the *delay distribution* of the channels.

B. Illustrative Results

Figure 2 shows the diameter versus the server number and Figure 3 shows the average delay versus the server number. As the subscription number of each channel increases with the rise of server number, more hops are needed to deliver the contents. Therefore, both propagation delay and scheduling delay increase, which lead to increasing diameter and average delay. COMMOS achieves the lowest delay because it utilizes the surplus bandwidth properly. VUD keeps high diameter and average delay because it aims at maximizing bandwidth but ignores delay. CSC has a relatively low average delay but a high streaming diameter because it optimizes each channel without server collaboration. Therefore, CSC usually achieves low delay for not-so-popular channels, but popular channels with more subscribed servers suffer high delay due to lack of resources and server collaboration. For the same reason, the streaming diameter of CSC increases sharply with the increase of server number. As both streaming diameter and average delay show the same trend in Figure 2 and Figure 3, we will focus on streaming diameter in the following figures.

We plot in Figure 4 the diameter versus the channel number. The diameter decreases as the channel number increases for all schemes. With a fixed expected value of total subscription to all the channels, the number of subscription to each channel declines with the rise of channel number, which also decreases the depth of each tree and the streaming diameter. COMMOS enjoys more performance improvement with fewer channels because it comprehensively considers the delay components and utilizes the collaboration among servers. In such scenario, a tree contains more servers and has to face both higher

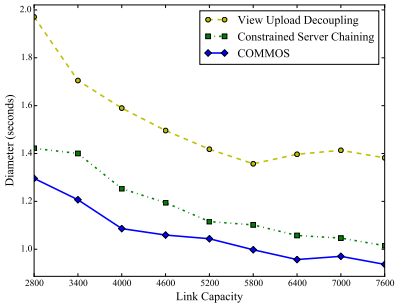

Fig. 5. Diameter versus link capacity.

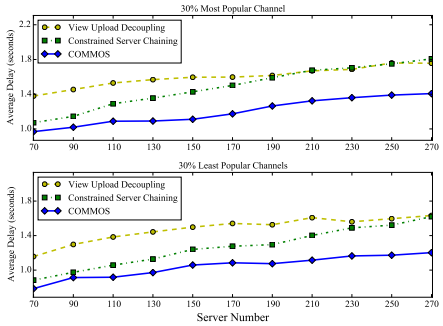

Fig. 6. Diameter versus server number with respect to channel popularity.

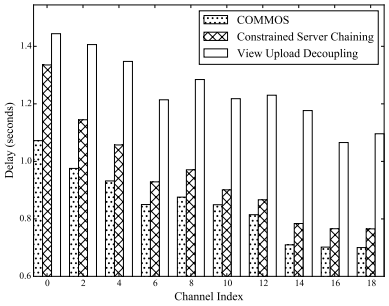

Fig. 7. Channel delay versus channel index.

scheduling and propagation delays. COMMOS achieves overall low delay by balancing between delay components and dredging extra bandwidth through server collaboration.

We show in Figure 5 the diameter versus the link capacity. The diameter decreases with the link capacity because more traffic can be transmitted through links with low propagation delay. COMMOS outperforms other schemes in all network environments. In the network with limited link capacity, COMMOS reduces delay through high bandwidth availability provided by server collaboration. In the network with abundant link capacity, however, the performance of VUD degrades because the servers with access to low-propagation-delay links have too many children and suffer higher scheduling delay. COMMOS considers delay components comprehensively and achieves overall low delay.

We plot in Figure 6 the average channel delay versus server number for the 30% leading popular and 30% least popular channels respectively. COMMOS keeps the lowest delay in both the most and least popular channels and the performance gap is wider with increasing number of servers. As the server number increases, more hops are needed to transmit the streaming data and the channels suffer higher delay. In COMMOS, servers with surplus capacity offer alternative paths to streams with high delay to reduce the overall delay.

We plot in Figure 7 the channel delay for each channel. The popularity of each channel is given by Zipf law and the channels are placed in the decreasing order of popularity. Channels with higher popularity are subscribed more and suffer higher delay. Hence, worst-case delay (streaming diameter) often happens in popular channels. COMMOS not only achieves low streaming diameter, but also outperforms the other schemes in each channel. This is because the servers collaboratively donate bandwidth even though they do not subscribe to those channels.

V. Conclusion

We have studied the delay optimization of a multi-source multi-channel live streaming network with server collaboration. The channel stream is divided into multiple substreams and pushed in multiple trees to the subscribing servers. We present a realistic delay model capturing both propagation and scheduling delays, and formulate the optimization problem to minimize the maximum channel delay. We show that the problem is NP-hardness, and propose an efficient algorithm, COMMOS (Collaborative Multi-source Multi-channel Overlay Streaming), to address the problem. We have conducted extensive simulation studies on real Internet topologies to evaluate the performance of our proposed algorithm, and show that COMMOS outperforms other state-of-the-art schemes by a wide margin (often by more than 40%).

References

[1] J. Chen, S.-H. Chan, and V. O. Li, "Multipath routing for video delivery over bandwidth-limited networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 10, pp. 1920–1932, 2004.

[2] M. Wang and B. Li, "R2: Random push with random network coding in live peer-to-peer streaming," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, pp. 1655–1666, 2007.

[3] T. Small, B. Li, and B. Liang, "Outreach: peer-to-peer topology construction towards minimized server bandwidth costs," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 35–45, 2007.

[4] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," *IEEE/ACM Transactions on Networking (TON)*, vol. 17, no. 4, pp. 1052–1065, 2009.

[5] D. Ren, Y.-T. H. Li, and S.-H. G. Chan, "Fast-mesh: A low-delay high-bandwidth mesh for peer-to-peer live streaming," *IEEE Transactions on Multimedia*, vol. 11, no. 8, pp. 1446–1456, Dec. 2009.

[6] H. Azarpira and S. Yousefi, "On optimal topology in hierarchical P2P live video streaming networks," in *Sixth International Symposium on Telecommunications (IST)*. IEEE, 2012, pp. 644–649.

[7] Z. Zhuang and C. Guo, "Optimizing CDN infrastructure for live streaming with constrained server chaining," in *9th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*. IEEE, 2011, pp. 183–188.

[8] Z. Wang, C. Wu, L. Sun, and S. Yang, "Strategies of collaboration in multi-channel P2P VoD streaming," in *IEEE Global Telecommunications Conference (GLOBECOM)*. IEEE, 2010, pp. 1–5.

[9] C. Wu, B. Li, and Z. Li, "Dynamic bandwidth auctions in multioverlay P2P streaming with network coding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 6, pp. 806–820, June 2008.

[10] D. Wu, C. Liang, Y. Liu, and K. Ross, "View-upload decoupling: A redesign of multi-channel P2P video systems," in *The 28th Conference on Computer Communications (INFOCOM)*. IEEE, 2009, pp. 2726–2730.

[11] C. Wu, B. Li, and S. Zhao, "Multi-channel live P2P streaming: Refocusing on servers," in *The 27th Conference on Computer Communications (INFOCOM)*. IEEE, 2008.

[12] J. Liu and G. Simon, "Fast near-optimal algorithm for delivering multiple live video channels in CDNs," in *22nd International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2013, pp. 1–7.