Beyond 1Mbps Global Overlay Live Streaming: The Case of Proxy Helpers

DONGNI REN, YISHENG XU, and S.-H. GARY CHAN, The Hong Kong University of Science and Technology

In order to provide live streaming over the global Internet, a content provider often deploys an overlay network consisting of distributed proxies placed close to user pools. Streaming of multi-Mbps video over such an overlay is challenging because of bandwidth bottlenecks in paths. To effectively overcome these bottlenecks, we consider employing proxy *helpers* in the overlay to provide rich path diversity. The helpers do not have any attached users, and hence may forward partial video streams (or not at all) if necessary. In this way, the helpers serve as stepping stones to supply full streams to the servers. The issue is how to involve the helpers in the overlay to achieve low streaming delay meeting a certain high streaming bitrate requirement.

To address the issue, we first formulate the problem which captures various delay and bandwidth components, and show that it is NP-hard. We then propose an efficient algorithm called *Stepping-Stones* (SS) which can be efficiently implemented in a controller. Given the encouraging simulation results, we develop a novel streaming testbed for SS and explore, through sets of Internet experiments, the effectiveness of helpers to achieve high bitrate (multi-Mbps) global live streaming. In our experiments, proxies are deployed with a reasonably wide global footprint. We collect more than a hundred hours of streaming traces with bitrate ranging from 500kbps to a few Mbps. Our experimental data validates that helpers indeed play an important role in achieving high bitrate in today's Internet. Global multi-Mbps streaming is possible due to their *multihop* and *multipath* advantages. Our experimental trials and data also provide valuable insights on the design of a global push-based streaming network. There are strong benefits of using proxy helpers to achieve high bitrate and low delay.

Categories and Subject Descriptors: H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Video*; C.2.1 [**Computer-communication Networks**]: Network Architecture and Design—*Network communications*; C.4 [**Performance of Systems**]: Design studies

General Terms: Measurement

Additional Key Words and Phrases: Multi-Mbps video, global live streaming, proxy helpers, testbed experiment

ACM Reference Format:
Dongni Ren, Yisheng Xu, and S.-H. Gary Chan. 2014. Beyond 1Mbps global overlay live streaming: The case of proxy helpers. ACM Trans. Multimedia Comput. Commun. Appl. 11, 2, Article 26 (December 2014), 22 pages.
DOI: http://dx.doi.org/10.1145/2652485

1. INTRODUCTION

The last-mile bandwidth of residential networks has been improved substantially in recent years. With the gradual lifting of this "last-mile bottleneck," there has been

This work was supported, in part, by Hong Kong Research Grant Council (RGC) General Research Fund (610713), HKUST (FSGRF13EG15) and the Hong Kong Innovation and Technology Fund (UIM/246).
Authors' addresses: Rm. 2606, The Hong Kong University of Science and Technology, Kowloon, Hong Kong; email: {tonyren, yishengxu, gchan}@cse.ust.hk.

DOI: http://dx.doi.org/10.1145/2652485

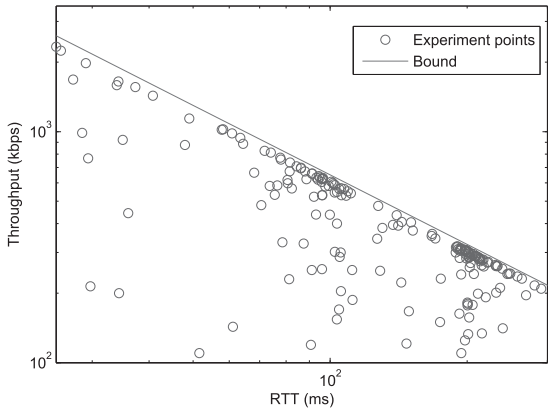


Fig. 1. End-to-end throughput versus RTT (log scale).

keen industrial and public interest in streaming high-quality multi-Mbps video to end users over the Internet. In order to effectively serve users with live contents, a content provider often deploys distributed proxy servers in proximity to user pools. These proxies form an overlay network to share video streams with each other, the so-called "over-the-top" (OTT) content distribution.

Streaming high-bitrate live video over the global public Internet still poses much challenge to content providers. This is because long connections between streaming proxies often suffer from unsatisfactory end-to-end throughput and jitter performance, adversely affecting the continuity of the streams. Furthermore, for live streaming the source-to-proxy delay has to be minimized to offer the best user experience. These concerns are markedly different from many of the asynchronous *stored* applications nowadays (such as YouTube, Hulu, Youku, etc.). How to globally distribute OTT *high-bitrate live* streams in a timely and efficient manner remains a rich and challenging research problem.

In a global network, proxies are often deployed where active users are leads to a relatively sparse network. The long-haul connections between proxies limit the throughput of the network, leading to bottlenecks. In order to support high bitrate streaming (e.g., multi-Mbps), we need to overcome such bottlenecks. Many previous studies assume that network edge is the only bottleneck. With the improvement of edge bandwidth, this assumption no longer holds well for a global network where end-to-end overlay bandwidth between proxies needs to be considered.

Nowadays streams are predominantly transmitted over TCP (e.g., using http as HLS or DASH [Wang et al. 2008; Müller et al. 2012; Concolato et al. 2011]). The round-trip time (RTT) is one of the most important factors affecting the end-to-end throughput between two nodes in the Internet. Figure 1 shows the relationship between RTT and end-to-end TCP throughput (in log scale), based on our extensive experiments collected from nodes at different locations on PlanetLab. Our experiments show that the maximum end-to-end throughput (bound) is inversely proportional to the RTT between the nodes, as expected from TCP throughput formula[Padhye et al. 1998]. The RTTs between nodes in Hong Kong and the United States are usually more than 200ms, leading to a TCP connection throughput less than 400 kbps. Clearly it is challenging to use a single long-haul connection to support global live streaming with high bitrate. A global network calls for multipath streaming.

In order to overcome the bottlenecks, we study in this article the use of *helpers*. Helpers are proxies deployed by the content provider in the overlay without (existing)

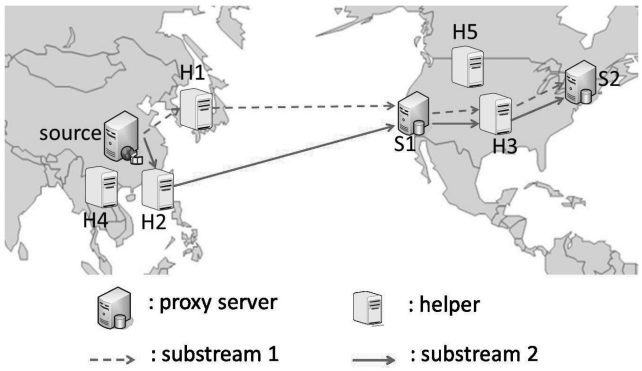


Fig. 2. A high-bitrate live streaming overlay with helpers.

end users. Therefore, they do not have to receive the full video stream. *Servers*, on the other hand, are proxies with active users and have to receive the full stream.

We consider that the full video stream is divided into a number of substreams by video packet multiplexing or encoding scheme. For example, the simplest way to divide a video into two substreams is to pack all odd packets of the video into one substream, and all even packets into the other substream. Each substream can be delivered via different network paths to the same destination. When a server receives both substreams, it can reassemble the full video stream, and delivers it to its local clients. To achieve low delay, we consider that each substream is *pushed* from the source to the servers as a delivery tree, that is, each proxy receives the substream from its parent, and then directly forward it to its children. To aggregate a full stream, the servers have to receive *all* the substreams, either from the streaming source, other servers or helpers. Helpers may also participate in streaming and serve as "stepping stones" (intermediate nodes) in these delivery trees so as to meet the bandwidth requirement of the servers.

We show in Figure 2 a possible deployment scenario of a global overlay network with helpers for live streaming. The video is divided into two substreams, which are pushed from the source to all servers via multiple trees (which may overlap on overlay edges). Helpers H_1 and H_2 receive and forward Substreams 1 and 2, respectively to server S_1. Helper H_3 receives both Substream 1 and Substream 2 (i.e., full stream) from S_1, and then delivers them to servers S_2. Some other helpers (H_4 and H_5) do not need to participate in the stream distribution as they do not contribute to the performance of the network.

Though helpers have been studied in the context of video-on-demand for storage purpose, there has been little work on how to make use of helpers for *live* overlay streaming. In this work we study, through algorithmic design and experimental studies, how to make use of helpers to achieve high bitrate global live streaming.

We first present a realistic delay model which captures various important delay components and bandwidth constraints for the overlay (such as end-to-end bandwidth, uplink bandwidth, scheduling delay, propagation, etc.). We formulate the optimization problem of constructing overlay trees for the substreams with proxy helpers, which is to minimize network diameter while meeting a certain streaming rate requirement. We analyze the problem complexity and prove that it is NP-hard.

We then propose an efficient heuristic making use of helpers called *Stepping-Stones* (SS) which meets bandwidth requirement while achieving low streaming delay. SS is efficient, and can be easily implemented in a controller for the overlay construction.

Through extensive simulation studies we show the impact of helpers in improving streaming performance in the network.

Given our promising simulation results, we implement a testbed and experimentally explore the effectiveness of SS with helpers to provide multi-Mbps live streaming in the public global Internet. We set up a cross-continent network and collect more than a hundred hours of data traces. Our network is deployed at various sites with diverse geographical footprints. We conduct extensive measurement studies with streaming rate ranging from 500kbps to a few Mbps. We show in realistic Internet environment how helpers play a role in achieving high streaming rate to overcome end-to-end bandwidth bottlenecks. As compared with much previous experimental studies on streaming, the uniqueness of this work is that the design, implementation and study are from the same group of researchers. As a result, we are able to explore system details to obtain much fine-grained visibility and evaluation on system performance. Our implemented streaming network is novel in that, besides its use of helpers, it constructs *multiple substream trees* and is *purely push-based* (previous implementations are often based on predominately pull or hybrid approach). With SS, we organize participating proxies as a high-bandwidth backbone to achieve low-delay streaming. Our work shows that it is feasible to implement a push-based streaming network with helpers which is commercially deployable to large scale.

Admittedly, as compared with many commercially deployed systems with tens to hundreds of thousands of users, our testbed appears relatively small in scale. However, in the absence of any existing system deploying helpers for push-based streaming, we need to resort developing such novel testbed ourselves for experimentation. To the best of our knowledge, this is the first piece of experimental work implementing and exploring the impact of helpers for live streaming on a push-based overlay. Our experiments span multiple countries, and despite of its relatively small scale, we obtain many appealing and exciting data which are general enough to shed insights on how a global high bitrate streaming network should be designed for the future large-scale deployment of the network.

Our testbed and experimental data validates and confirms our algorithm and simulation results, and reveals that helpers are able to significantly improve streaming performance through the following.

—*Providing rich path diversity to overcome network bottlenecks.* Helpers are able to provide path diversity in the network, by offering a rich set of alternative paths as "stepping stones" from one proxy to another. As bandwidth can be aggregated along multiple paths, this overcomes bandwidth bottlenecks of end-to-end connections.
—*Improving throughput and jitter with shorter connections.* Helpers can be deployed in the middle of a long-haul connection, breaking the connections into multiple shorter overlay segments. This "multihop" nature of shorter connectivity greatly improves the end-to-end throughput and jitter of the network.
—*Increasing system capacity.* Helpers donate bandwidth to the network. Their presence increases the reservoir of available uploading bandwidth of the network. This bandwidth can be drawn upon when needed to increase system capacity by accommodating more servers (and hence more users).

The rest of the article is organized as follows. We discuss the previous work in Section 2. Then we present the problem formulation and its complexity analysis in Section 3. Section 4 presents the *Stepping-Stones* algorithm that constructs the proxy streaming network with helpers, and its illustrative simulation results. Section 5 gives an overview of the design and implementation details of our experimental proxy network with helpers, which include system settings and parameters, as well as

measurement metrics. Section 6 presents the analysis of our collected streaming data and our major observations. We conclude in Section 7.

2. RELATED WORK

The concept of helpers has been discussed in various contexts. In P2P Video-on-Demand (VOD), nodes with residual bandwidth are used to reduce server loads and distribute resources [He and Guan 2010; Li et al. 2010]. In file sharing, powerful peers are added to accelerate the download rate [Wang et al. 2007]. In hybrid CDN-P2P streaming, helpers (called repeaters) are assigned to channels short of bandwidth [Chang et al. 2009, 2011]. These nodes receive and forward all video substreams, and function as bandwidth multipliers to increase the amount of available bandwidth in the network. Sengupta et al. [2011] show how helpers can be used to increase the streaming capacity. In CDN live streaming, helpers (also known as reflectors) act as intermediaries between the entry-points and the edge clusters, where each reflector can receive one or more streams from the entry-points and can send those streams to one or more edge clusters [Kontothanassis et al. 2004; Nygren et al. 2010; Su and Kuzmanovic 2008; Adler et al. 2011, Ren et al. 2009, 2008; Jin et al. 2009]. Despite of these works, there has not been optimization and experimental work exploring the design, usage and impact of helpers in high-bitrate global live streaming.

There is a large body of work studying the placement of servers (proxies) in an overlay streaming network. Yuan et al. [2013] propose a server placement model that targets to reduce cross-ISP traffic. Yin et al. [2013] jointly optimizes the trade-off between the delay performance and the deployment cost under the constraints of client locations and server capacity. In our study we assume the locations of the proxies are given (e.g., precalculated by any of the server placement model), and focus on the construction of the streaming overlay with helpers.

There have been many experimental studies on P2P or overlay streaming [Jiang et al. 2012; Alessandria et al. 2009; Yin et al. 2009; Wu et al. 2011; 2008; Vu et al. 2010]. The experiments are often conducted on some commercially deployed systems, for instance, PPLive, UUSee, Zattoo, etc. While the user pools are impressive in the studies, due to its proprietary and commercial nature, they often lack visibility and tunability on system parameters. Our approach here provides another trade-off point where the experiments are conducted on a system which we implemented, thus giving us much visibility into operation details of the streaming network. Much of the previous measurement studies are conducted on mesh overlay, which uses predominantly pull-based data delivery and relatively low bitrate videos (several hundreds kbps). There has been little work studying and measuring push-based multitree streaming networks.

There are studies that propose different approaches to overcome the long-haul TCP throughput limitation, for instance, to use parallel TCP connections to increase source-to-end transmission rate [Kuschnig et al. 2010b; Chen and Chan 2001], or override congestion control algorithms to achieve highspeed TCP [Alvarez-Horine and Moh 2012]. These techniques are orthogonal to our study, and can be directly adopted in our proposed proxy helper network to further boost the streaming rate.

Multiple description coding (MDC) has been widely used in media streaming to address the bandwidth heterogeneity issue. The video source encodes data into multiple descriptions. At the receiver end, the streaming quality is improved with the number of descriptions received [Hsiao and Tsai 2010; Castro et al. 2003; Kostic et al. 2003]. However, MDC suffers from coding penalty. We consider meeting the quality requirement, that is, the video has to be fully received by a proxy server before its playback, and hence there is no coding penalty as compared with MDC.

In our previous work [Jiang et al. 2012], we explore the performance of a Proxy-P2P system under various configurable settings, that is, tree-depth, IP-multicast, churns,

etc. However, helpers have not been investigated, We study the impact of helpers in global streaming, and the feasibility of multi-Mbps live streaming. In order to stream multi-Mbps globally, we explore the multipath effect that helpers provide, and how it can increase the streaming rate and reduce the playback delay. We have employ different design principles, software implementation and experiment settings from Jiang et al. [2012]. To the best of our knowledge, this is the first piece of experimental work exploring the feasibility of a helper-assisted push-based live streaming network with multi-Mbps streaming rate.

Our previous work [Ren and Chan 2012] discusses the potential advantages of using helpers. In this work we further explore the impact of helpers in live streaming over global Internet, and how it is used to overcome the throughput limitations. We also design and develop the proxy streaming network with helpers. The network is deployed at 10 collaborator sites with various geographical footprints, and extensive measurement studies are conducted.

3. PROBLEM FORMULATION AND COMPLEXITY

3.1. The Problem of Minimum-Delay Streaming with Helpers

We formulate the overlay as a directed graph $G = (V, E)$, where V is the set of vertices containing the overlay *nodes* of servers, helpers, and the streaming source. Let S be the streaming source, H be the set of helpers and P be the set of servers; therefore, $V = \{S\} \cup P \cup H$. $E = V \times V$ is the set of possible overlay connections between nodes in V (does not have to be complete). For every edge $\langle i, j \rangle \in E$, there is a propagation delay d_{ij}^p from node i to node j in the physical network.

The video is divided into multiple substreams of similar bandwidth. We consider that the bandwidth is normalized to some unit equal to the streaming rate of a substream denoted as b_s (e.g., $b_s = 400$ kb/s). Let $s \in Z^+$ be the streaming rate in that unit, that is, s is the number of substreams of the full video stream. Each unit of stream (hence a substream) is delivered to all the nodes in P by a spanning tree and there are a total of s delivery trees. Denote the spanning tree of the k^{th} substream as T_k.

For every node i in V, it has an uplink bandwidth of U_i units, $U_i \in Z^+$, which represents the maximum total number of children it can serve in all spanning trees. The end-to-end throughput of the edge $\langle i, j \rangle$ is denoted as $w_{ij} \in Z^+$, which is the maximum number of substreams that can simultaneously accommodate in edge $\langle i, j \rangle$. For any node in V, if it gets an aggregated stream of s units from its parents, we call the node *fully served*. In other words, if node i receives streams from all s spanning trees, it is fully served and can play back the video with continuity. Note that S has an uplink bandwidth of U_S units and has no parent.

For every node $i \in P$, we define the incidence matrix $A = [a_{ik}]$ indicating whether node i is on tree k, that is,

$$a_{ik} = \begin{cases} 1, & \text{if } i \in T_k; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Therefore, $a_{i1} = a_{i2} = \cdots = a_{ik} = 1$ if and only if node i is fully served.

For the streaming session to be feasible, the total uplink bandwidth must be larger than the total streaming bandwidth, that is,

$$\sum_{i \in V} U_i \geq (|P| - 1) \times s, \tag{2}$$

otherwise there is not enough resource to fully serve all servers. The "-1" is due to the source (which does not need to be supplied with any substream).

Similarly, the aggregate incoming bandwidth of each node must be larger than the streaming rate, that is,

$$\sum_{\forall i \in V} \min(w_{ij}, U_i) \geq s, \forall j \in P. \tag{3}$$

Note that the maximum throughput between two node is $\min(w_{ij}, U_i)$, which is bounded by the minimum of edge bandwidth of node i and core bandwidth of edge $\langle i, j \rangle$.

We define the worst-case scheduling delay d^s_{ji} from node j to node i as the maximum amount of time that node i has to wait until it receives the substream(s) from node j. It is given by

$$d^s_{ji} = \sum_{k \in C(j)} \frac{L}{\min(w_{jk}, U_j) b_s / t_{jk}}, \tag{4}$$

where L (bits) is the segment size used in streaming, $C(j)$ is the set of children of node j in all spanning trees, and t_{jk} is the number of concurrent substreams on edge $\langle j, k \rangle$. Equation (4) represents the total amount of time that node j needs to *push* the substreams to all of its children. Note that any specific scheduling mechanisms, that is, the substreams are pushed one by one or simultaneously. The worst scheduling delay is the same for different scheduling algorithms.

Denote the source-to-end delay of node i in spanning tree T_k as D^k_i, which equals to the delay of its parent j in tree T_k plus the propagation delay and scheduling delay between j and i, that is,

$$D^k_i = D^k_j + d^p_{ji} + d^s_{ji}. \tag{5}$$

The total delay D_i of node i is given by its maximum source-to-end delay D^k_i among all spanning trees, that is,

$$D_i = \max_{k \in [1,s]} D^k_i. \tag{6}$$

We now state the problem of our study.

Minimum-Delay Streaming with Helpers (MDSH) problem. The MDSH problem is to find an overlay which minimizes the maximum of the server delay (i.e., minimizes the streaming diameter),

$$\min \max_{i \in P} D_i \tag{7}$$

subject to the streaming rate requirement, that is, all servers receive an aggregate incoming stream of s units, that is, $A = [a_{ik}] = 1, \forall i \in S$.

3.2. Complexity Analysis

We show in this section that the complexity of our problem is NP-hard using *Proof-by-restriction* [Garey and Johnson 1990; Cormen et al. 2009]. MDSH is obviously in P. This is because we can compute the maximum delay of a given streaming cloud in polynomial time. Given a graph $G(V, E)$ and its corresponding optimal delay, we hence can verify whether the constructed overlay is the optimal overlay.

The well-known NP-hard travelling salesman problem (TSP) is reducible to MDSH problem in polynomial time. Let $G'(V', E')$ be the graph of a TSP instance. We transform $G'(V', E')$ into $G''(V'', E'')$ by adding a vertex S_{end} and edges from all the vertices to S_{end}. In this way, the vertices in V'' represent servers and the weight on the edges are the propagation delay plus the transmission time of a segment between the two adjacent servers. We let S be the source, and consider the special case that the streaming rate

is 1 unit of substream, uplink bandwidth of each peer is also 1 unit, and S_{end} has zero uplink bandwidth. In this way the resulting overlay topology must be a chain starting at S and ending at S_{end}. D_{max} is equal to the delay of S_{end}, which is the sum of all delays preceding it. Hence, it is obvious that D_{max} in G'' is minimum if and only if the cost of the Hamiltonian cycle in G' is minimum. Therefore TSP is polynomial reducible to MDSH.

4. STEPPING-STONES ALGORITHM AND SIMULATION

4.1. Algorithm Details

In this section we present the Stepping-Stones (SS) algorithm we implement at a server to arrange the proxies to a low-delay streaming overlay. Due to the NP-hardness of the problem, the algorithm is heuristic in nature, which is to construct multiple trees for each substreams spanning all the servers and involves helpers if necessary to meet the streaming bitrate requirement. At the end of the section we discuss the complexity of SS, which is shown to be polynomial and run-time efficient.

To construct an overlay with low delay, we need to determine which helpers to include in the streaming overlay, how many substreams each helper receives, and which proxies it forwards to. To do that, we construct s delivery trees spanning all servers through iterations, where $s \in Z^+$ is the normalized streaming rate which is also the total number of substreams. Delivery trees are expanded from the source to the destinations. In each iteration, it adds one server into one partially constructed delivery tree. The intuition behind our algorithm is that we only include helpers that could reduce the overall delay in each substream tree, and hence helpers only participate in the overlay if they improve streaming. Furthermore, helpers may only receive partial stream. This is in remarkable contrast with previous work, where helpers either receive full stream or not at all.

In the tree construction step, each delivery tree, or T_k, is initialized containing only the streaming source S. For every node i not in T_k, we calculate the potential delay of node i in tree T_k as

$$D_i^k = \min_{\forall j \in T_k} D_i^k(j), \tag{8}$$

where $D_i^k(j)$ is the delay of node i in tree k if it connects to node j as its parent. Let d_{ij} be the end to end propagation delay from node i to node j. Each connection $\langle i, j \rangle$ has a certain maximum transmission rate (normalized to substream bitrate), denoted as $w_{ij} \in Z^+$. In other words, w_{ij} is the maximum number of substreams that can be accommodate from i to j. Let $t_{ij} \in Z^+$ be the existing traffic (in number of substreams) from i to j and $r_{ij} \in Z^+$ be the residual bandwidth from i to j. r_{ij} clearly can be written as

$$r_{ij} = w_{ij} - t_{ij}. \tag{9}$$

If i directly connects to j in tree k, its delay can be calculated as

$$\hat{D}_i^k(j) = D_j^k + d_{ji}, \quad \text{if } r_{ji} > 0. \tag{10}$$

Because the link between nodes j and i may be congested, or may not have enough throughput to support the substream, we employ helpers to bypass the bottleneck link. In this case, the delay from node j to node i through helper h can be calculated as

$$\hat{D}_i^k(j, h) = D_j^k + d_{jh} + d_{hi}, \quad \text{if } r_{jh} > 0, r_{hi} > 0. \tag{11}$$

If the direct connection of $\langle i, j \rangle$ is not available or has higher delay, a two-hop path through a helper is selected. Therefore $D_i^k(j)$ can be calculated as

$$D_i^k(j) = \min\left(\hat{D}_i^k(j, h), \min_{\forall h} D_i^k(j, h)\right), \tag{12}$$

Table I. Baseline Parameters in Our Simulation

Parameter	Baseline value
Number of proxies	200
Number of servers	Half of proxies
Streaming rate	1.2 Mbps
Substream bandwidth	400 kbps
Proxy uplink bandwidth	3 Mbps
Edge bandwidth	3 Mbps

which is the minimum delay from node j to node i in tree k. We then choose the node i with lowest delay and connect it to the corresponding tree T_k, that is,

$$\arg_{\{i,k\}} \left(\min_{\forall i \notin T_k, \forall k} \hat{D}_i^k \right). \tag{13}$$

We continue this process until every server is connected to all delivery trees.

The run-time complexity of the heuristic algorithm is $O(s^2|P|^3|H|)$, where $s \in Z^+$ is the (normalized) streaming rate, P is the set of servers and H is the set of helpers. This is proved as follows. In the tree construction step, adding one server to one delivery tree takes $O(s|P|^2|H|)$ time and there are $O(s|P|)$ iterations in total. Therefore the tree construction step takes $O(s^2|P|^3|H|)$ time.

4.2. Simulation Results

We have conducted extensive simulation on *SS*, and present illustrative simulation results[1] here. The simulation is carried out on a real Internet topology provided by CAIDA, which was collected on June 12th, 2011, and contains 1,747 routers and 3,732 links. The round trip times (RTTs) between interconnected routers are also given in the topology. We use Distance-vector routing to compute the latencies between any two router nodes in the network. Proxies (servers and helpers) are attached to the routers randomly and their uplink bandwidth is normally distributed with mean $\mu = 3$ Mbps and standard deviation $\sigma = 1.2$ Mbps (accepting only the positive values).[2] We set the segment size as 100 kbits and the streaming rate of a substream as 400 kbps[3]; [Vieira et al. 2013]. Unless otherwise stated, the baseline parameters used in our simulation is shown in Table I. We have also run our simulations on 10 different two-levels top-down hierarchical Internet topologies generated by BRITE [Medina et al. 2001]. The results of those simulations are qualitatively the same as what is presented here, and hence are not shown for brevity.

We evaluate the performance of our proposed algorithms with the following metrics.

—*Worst-case delay.* The worst-case delay is the maximum time taken for a packet to travel from the streaming source to the servers in the overlay network. Our algorithm is to minimize this delay while meeting bandwidth requirement.
—*Delay components and distribution.* Besides the worst-case delay, we are also interested in the delay distribution of the servers, and delay components for scheduling and propagation.
—*Helper involvement.* The use of helpers leads to a better streaming cloud with the cost of additional machines and resources. We study the number of helpers involved in *Stepping-Stones*.

[1] http://www.caida.org.
[2] http://www.netindex.com/.
[3] http://pptv.com.

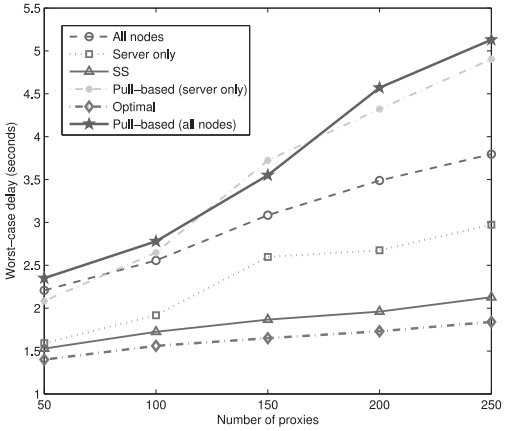


Fig. 3. Worst-case delay versus number of proxies with of them servers.

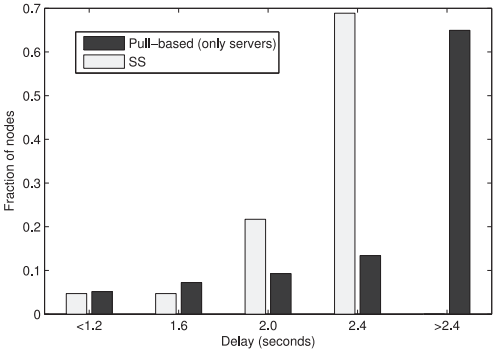


Fig. 4. Delay distribution.

We compare SS with the optimal solution and three other schemes. The optimal solution is obtained by exhaustive search. The "All nodes" scheme is a common one with all proxies, servers or helpers, receiving full streams. It is implemented as applying *SS* algorithm with all helpers treated as servers. The Servers only scheme means that no helpers is involved in the streaming, and hence only servers help each other in the overlay. It is implemented as applying the *SS* algorithm on the servers only (which is half of the proxies). The *pull-based algorithm* has been commonly implemented nowadays in overlay networks, where the servers periodically exchanges its buffermap with other servers. They randomly pull the available segments from their neighbors and then reassemble a full stream.

Figure 3 shows the worst-case delay of *SS* algorithms versus number of proxies where the number of servers is the same as the number of available helpers. From the figure, we see that as the number of proxies (and hence servers) increases, the worst-case delay increases. This is because of more servers and hence more hops in the network. *SS* performs the best because it uses a partial set of helpers depending on the network condition to achieve low delay. "Pull-based" algorithm performs worst since the overlay is not optimized and helpers are not involved in the streaming network. "All nodes" scheme also has long delay because all helpers are included in all substream trees and a large portion of system bandwidth is wasted delivering redundant data to the helpers. *SS* decides whether a helper node would help in a specific substream tree, and then places it to an optimal position in the tree. Because the participating helpers provide additional throughput to the system and reduce the scheduling delay, *SS* achieves better delay than "Servers only."

Figure 4 compares the delay distribution of servers with different schemes (for 200 proxies and 100 servers). Clearly, servers in *SS* achieve low delay as compared with the pull-based algorithm (server only). *SS* tries to arrange the overlay in a way that most of the servers share rather similar delays, and the worst-case delay is not much larger than the average delay. The performance of the pull-based algorithm with all nodes is qualitatively the same as the "server only" version, and hence omitted here for brevity.

We next study the number of helpers actually participate in streaming by plotting in Figure 5 the number of helpers actively involved in forwarding substream(s) versus the number of available helpers for *SS* (with number of servers equal to 100). Clearly, not all helpers are involved in streaming. Only a low fraction of the helpers (about 10–15% in the figure) are needed to be activated to help delivering the substreams.

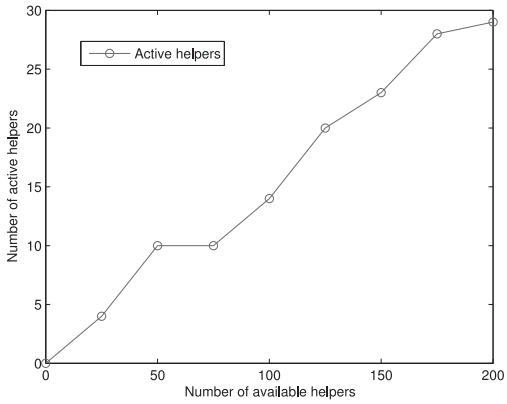


Fig. 5. Activated helpers versus available helpers.

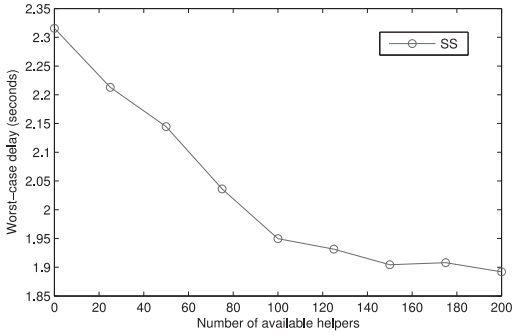


Fig. 6. Worst-case delay versus the number of helpers.

We show in Figure 6 the worst-case delay versus the number of helpers in the streaming cloud (with number of servers equal to 100). As the number of helpers increases, the delay decreases. This is because more helpers means that there are more space to optimize the delay. The marginal benefit of adding more helpers, however, decreases with the number of helpers. This is because there is no need to add more helpers if the helpers are dense enough.

We show in Figure 7 the components of scheduling and propagation delays in the worst-case delay (for *SS*). Our results show that scheduling delay is the major component, though propagation delay also plays a significant role. This validates that both delays have to be considered in order to optimize the network. Normally, as observed in other experimental studies [Jiang et al. 2010], the scheduling delay increases more quickly than propagation delay because whenever a proxy serves a new child, such increase in scheduling delay affects all of its existing descendants in all substream trees. As the increase in both scheduling and propagation delays is not sharp, *SS* effectively controls both delays.

We show in Figure 8 the maximum streaming rate *SS* can support given the number of available helpers (with number of servers equal to 100). The streaming rate is normalized with 400kbps per unit. The result shows that with more helpers available in the cloud, the maximum streaming rate *s* improves significantly. The helpers increase the total bandwidth and capacity of the system, and at the same time offer a richer set of alternate paths between nodes. Given the path diversity in the network, servers can aggregate bandwidth along multiple paths and overcomes bandwidth bottlenecks of end-to-end connections. The maximum streaming rate of All nodes does not increase with the number of helpers. This is because when all nodes need to be fully served, extra helpers do not bring additional resources to the system.

5. GLOBAL TESTBED DESIGN

5.1. Software Architecture and Implementation

In this section we describe the software architecture and framework of our proxy streaming network. Figure 9 shows the major components of the network we implemented. The live videos are captured and encoded at the streaming source. The encoded video is split into multiple substreams by the stream splitter for transmission. The stream splitter use packet multiplexing as the splitting strategy. It could also adopt more complex splitting strategies, such as MDC. All proxies (both servers and

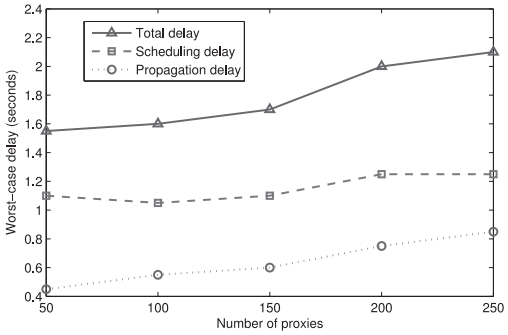


Fig. 7. Delay components.

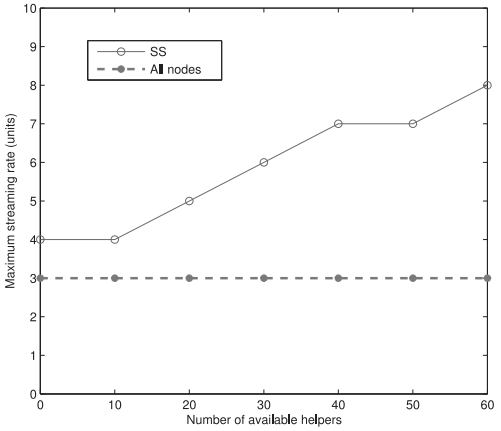


Fig. 8. Streaming rate versus number of available helpers.

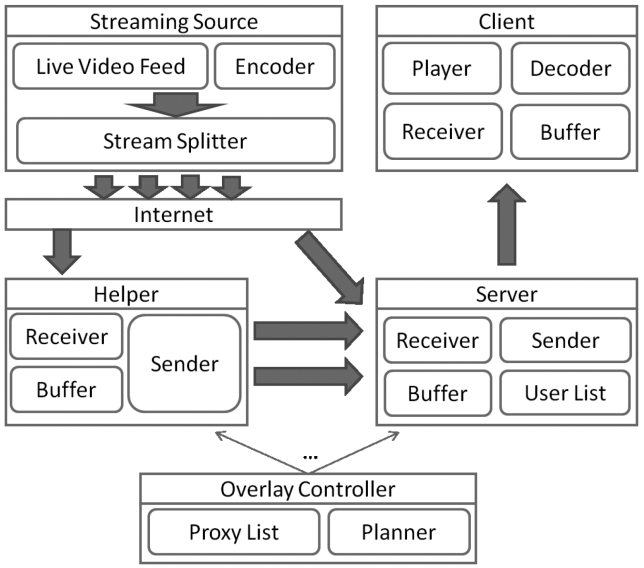


Fig. 9. A software framework of the proxy streaming network.

helpers) are registered at the overlay controller, which uses the heuristic algorithm as discussed in Section 4 to optimize the overlay by organizing the substream delivery between proxies. When there is a new proxy joining the system, the controller informs the proxy the nodes it gets data from, and the nodes it forwards data to. Proxy servers are responsible to keep track of its clients as well as deliver the full video stream to them. At the client side, the received substreams are reassembled and fed to the decoder so that the user is able to playback the real-time live video.

We develop a novel network with a purely push-based strategy to distribute the substreams. We define the minimum inseparable data unit as a "chunk," which is a number of consecutive video packets. Proxies do not need to exchange buffermap nor wait till a full buffer of chunks to arrive. Whenever they receive a chunk, it is immediately pushed to the downstream children without any delay. In this way the

Table II. Deployed Proxies and Their Locations

Number of PCs	Site Location	Country/Region
3	HKUST	Hong Kong
1	CUHK	Hong Kong
1	POLYU	Hong Kong
1	HKU	Hong Kong
1	HKBU	Hong Kong
4	KAIST	Korea
4	Princeton	U.S.
4	Caltech	U.S.
1	Orange, CA	U.S.
1	AT&T, NY	U.S.

scheduling delay and control overhead are both significantly lower than the data-driven, or pull-based, approach.

Although UDP is the conventional approach for media streaming in research community, TCP is much more widely used in the commercial streaming systems (e.g., youtube, hulu, etc). A number of UDP applications have recently changed their transport protocol to HTTP/TCP recently as well (e.g., PPTV, QQLive, etc.). We hence use TCP as the transport protocol in stream distribution, due to its following advantages. First, TCP is more reliable and equipped with natural loss recovery mechanism. This effectively minimizes unintended random loss which seriously affects video quality. Second, with TCP it is easier to get through firewalls and NATs. It is much more friendly to cross-platform client programs, for instance, browser-based player, mobile streaming on iOS and Android devices, etc. Studies have shown that TCP streaming provides good streaming performance with low startup delay [Wang et al. 2008; Kuschnig et al. 2010a].

In our experiments, we deploy 21 proxies located on 10 collaborator sites. Their geographical locations and node statistics are given in Table II.

5.2. Measurement Metrics and Methodology

We have implemented the proxy network and a packet-level monitoring system to study the impact of helpers under different settings and environments. The monitoring module is embedded in all components of the proxy network, that is, streaming source, helpers, servers and client programs, so that we can study the system in details. We generate a log entry for chunks departing and arriving at the node. We send the source stream into multiple substreams using packet multiplexing. Figure 10 shows the information contained in each log entry. The field *node_id* and *chunk_id* specifies which node and which chunk this entry belongs to. *stream_id* records the substream that the chunk belongs to. In a sender entry, *depart_t* records when the chunk is scheduled to send and *receiver_id* is the destination of this chunk. In the receiver, *arrive_t* logs the arrival timestamp and *sender_id* is the ID of the proxy (or streaming source) that sends the chunk to the node. We use NTP servers to synchronize the local clocks of the proxies. By collecting the records from the streaming source and all proxies, we can reconstruct the lifetime for every data chunk in the network. The proxy streaming network also has a list of toolkits that monitors the RTT and end-to-end throughput between nodes. As the development and study of the system are from the same group of researchers, we are able to investigate its performance in details. We define the important system parameters and metrics as follow.

—*Streaming rate.* The streaming rate of the video source ranges from 500kbps to a few Mbps. We set the bitrate of each substream as 250kbps, denoted as b, and

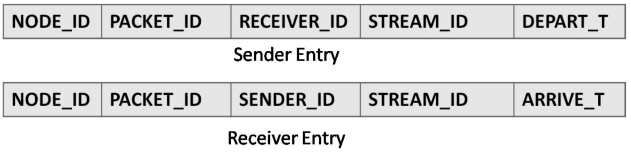


Fig. 10. Log entry format.

define the normalized streaming rate as s, which is the number of substreams. More substreams are generated at the source for videos with higher bitrate rate. We use a video source with bitrate of 2 Mbps as baseline in our experiments, which is composed of 8 substreams, that is, $s = 8$. 2Mbps satisfies the bitrate of SDTV video streaming in commercial IPTV services.

—*Sampling ratio.* The sampling ratio affects the number of log entries we record during an experimental run. It trades off overhead with details. Denote the sampling ratio as $r, 0 \leq r \leq 1$. For every n chunks generated, we collect logs of $n \times r$ chunks. To reduce the high overhead of logging every single chunk, we sample the chunks by first collecting logs for s consecutive chunks, one for each substream. After that we skip the next $s \times (1/r - 1)$ chunks, and then collect logs for another s consecutive chunks. In this way there are total $n \times r$ out of n chunks sampled.

—*Chunk delay.* The source-to-end delay of a chunk can be calculated with the log entries recorded at the streaming source and the server. We denote the chunk delay of chunk k from the streaming source to proxy i as $T^k(i)$, which is the time elapsed from chunk k's generation at the source to its complete receipt at proxy i. It is calculated as

$$T^k(i) = arrive\_t^k(i) - depart\_t^k(S), \tag{14}$$

where $arrive\_t^k(i)$ is chunk k's arrival time at proxy i, and $depart\_t^k(S)$ is the departure time of chunk k from the streaming source S.

—*Chunk interarrival time.* The chunk interarrival time is the time elapsed between the arrivals of two consecutive chunks at a proxy. Denote the interarrival time between chunk k and chunk $k-1$ at proxy i as $IA^k(i)$, which can be calculated as

$$IA^k(i) = arrive\_t^k(i) - arrive\_t^{k-1}(i). \tag{15}$$

Note that in the results, we only take the interarrival time of consecutive chunks into statistics.

—*Buffering time.* There are jitters in the chunk arrival time due to various reasons, for instance, network congestion, background traffic, etc. It may lead to continuity problem if the video is played back immediately after the first few chunks arrive. Therefore a buffering time T should be added, and the video is played back at the proxy after waiting T seconds since the first chunk arrival.

—*Playback delay.* Playback delay is defined as the time elapsed from the timestamp that the live video is streamed at the source, to the timestamp that the video can be played back at the proxy. It is determined by the chunk delay and the buffering time. The playback delay $\hat{T}(i)$ of proxy i can be calculated as

$$\hat{T}(i) = T^l(i) + T, \tag{16}$$

where l is the first chunk that, arrives at proxy i and $T^l(i)$ is its delay.

—*Playback moment of a chunk.* The playback moment is the timestamp that a specific chunk is needed at the player for playback. It is also the arrival deadline of this chunk at the proxy. The playback moment $PB^k(i)$ of chunk k at proxy i is

$$PB^k(i) = arrive\_t^l(i) + T + \frac{C \times (k-l)}{b \times s}, \tag{17}$$

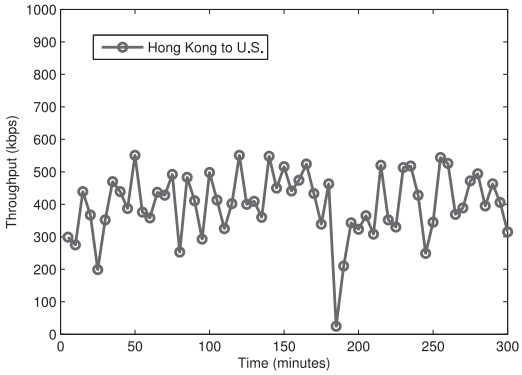


Fig. 11. End-to-end throughput from Hong Kong to U.S.A.

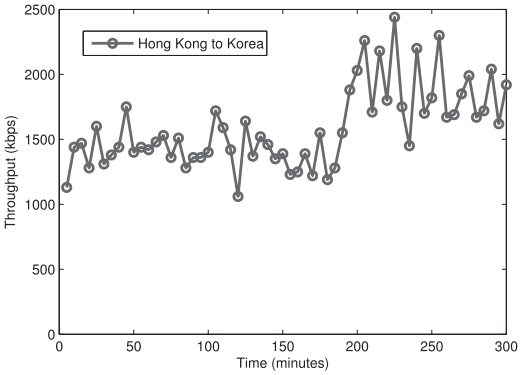


Fig. 12. End-to-end throughput from Hong Kong to Korea.

where l is the first chunk that arrives at proxy i and C is the size of a data chunk. In our experiment we use $C = 4KB$ as baseline.

—*Loss rate.* We define a chunk is lost if it arrives after its playback moment (i.e., missing its deadline). The loss rate is the percentage of lost chunks among all the chunks examined.

6. EXPERIMENTAL STUDIES AND RESULTS

In this section we analyze the data collected from our global testbed experiments and present our major findings. We first explore the characteristics of end-to-end throughput in Section 6.1. Then, in Section 6.2, we show the impact of helpers to provide global high-bitrate streaming. Section 6.3 shows the performance of chunk delay. In Section 6.4 we explore the design of buffering and its impact on stream continuity.

6.1. End-to-End Throughput

Figures 11 and 12 show the typical throughput fluctuation of a long-haul and a short-haul connections, respectively, over time. Live streaming requires sustainable bandwidth so that the transmission rate can be maintained during the streaming session. We observe that the throughput of long-haul links (e.g., from Hong Kong to U.S.A.) have higher jitters and fluctuates more than the connections with smaller RTT (e.g., Hong Kong to Korea and Korea to U.S.A). (See Figure 13). The figures show that it is challenging to sustain a continuous stream on long-distance connections.

6.2. Impacts of Helpers

We explore the effect of helpers in multi-Mbps global streaming through a set of experiments. The streaming source is located at HKUST, Hong Kong, and a server is deployed at Princeton, U.S.A. We use our proposed heuristic algorithm to construct the streaming network.

6.2.1. Control Experiment without Helpers. We first consider the case of direct connection between source and server without any helpers in between. Figure 14 shows the streaming topology from HKUST to Princeton. The maximum streaming rate that the direct link can carry is very small, that is, less than 300kbps. It is mainly due to the limited end-to-end throughput and heavy bandwidth fluctuation. Therefore it is not feasible to conduct streaming with high bitrate on a single long-haul connection.

6.2.2. Streaming with Helpers. We then conduct an experiment with several helpers deployed in the same region as the source, that is, Hong Kong. The streaming topologies

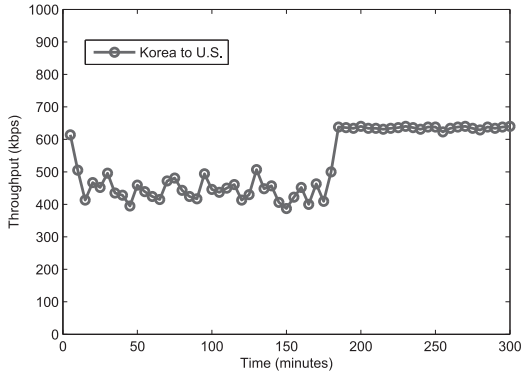
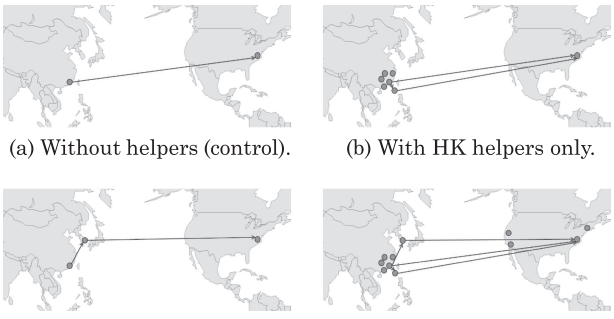


Fig. 13. End-to-end throughput from Korea to U.S.A.



(a) Without helpers (control).

(b) With HK helpers only.

(c) With Korea helpers only.

(d) With both HK and Korea helpers.

Fig. 14. Streaming topology.

are shown in Figure 14(b). Helpers are used to increase the throughput from Hong Kong to the server in Princeton. However, not all helpers in Hong Kong are included in the overlay. This is because the end-to-end bandwidth from some of the helpers to the server is less than the bitrate of a single substream due to the large RTT. In this case they cannot participate in the stream delivery. Although the streaming rate is increased with the use of helpers in Hong Kong, we still cannot achieve multi-Mbps bitrate in a cross-continent Internet environment.

Next we stream with a few helpers deployed in Korea, as shown in Figure 14(c). The direct link from HK to Princeton is split into two shorter links, that is, HK to Korea and Korea to Princeton. RTTs from HK to Korea and from Korea to Princeton are both much smaller than from HK to Princeton directly. Therefore, the throughput of both hops, that is, HK to Korea and Korea to Princeton are larger than the long-haul connection from HK to Princeton. With the helpers in Korea, the streaming rate is significantly increased due to this "multihop" effect.

Figure 14(d) shows a topology with helpers in both Hong Kong and Korea. The resulting overlay achieves a higher aggregated streaming rate than the first three cases. The video stream is delivered to the streaming server via multiple paths, that is, Hong Kong to Princeton and Korea to Princeton. We compare in Figure 15 the maximum achievable streaming rate of the four topologies. The direct link has the smallest streaming rate due to the limited end-to-end throughput. With helpers in Hong Kong and Korea, the streaming rate significantly increases.

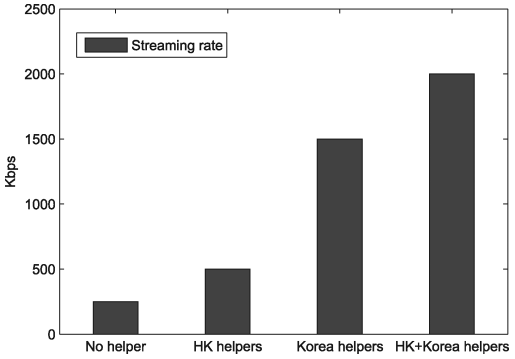


Fig. 15. Streaming rate with helpers.

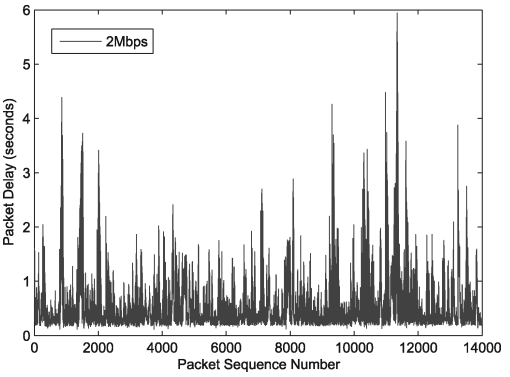


Fig. 16. Chunk delay versus sequence number.

6.3. Delay Performance

In this section we review the statistics we have collected from the experiments. Unless otherwise stated, we use 2Mbps video source and the topology in Figure 14(d) as our baseline.

Figure 16 shows the delay of the chunks from the streaming source to the streaming server. Since chunks goes from the source to the destination via different delivery path, the delay of each chunk differ from each other by significant amounts. As we can see in the figure, the delay of the chunks also fluctuates heavily with time due to change of network condition on a specific link.

Figure 17 shows the chunk delay CDF with different streaming rate. The overlays are constructed using the heuristic algorithm. For 500kbps, 1Mbps and 1.5Mbps videos, the topology shown in Figure 14c is used, and 14d is used for 2Mbps video. We observe that most of the chunks (70–80%) have a delay of 800 milliseconds for all streaming rates. Chunks in 500kbps video has relatively smaller delay than the videos with a larger bitrate. There is a long tail effect in delay CDF due to the late chunks and network fluctuation.

Figure 18 shows the delay components of different streaming rate. High streaming rate, that is, over 1Mbps, leads to higher delay. The queuing delay makes up a significant proportion of the total chunk delay. It is because at the sender side, that is, streaming source and the helpers, chunks need to wait until it is scheduled to be transmitted. When there is bandwidth jitter or chunk loss, it takes longer time for the previous chunks to be delivered.

Figure 19 shows the interarrival time between chunks at the streaming server versus their sequence number. We observe that some of the chunks have negative inter arrival time or out of order delivery. This is because chunks are delivered via multiple paths, and some chunks with larger sequence number (which will be played later than the previous chunk) arrive earlier than their previous chunks. We will explain the impact of interarrival jitters to playback delay, buffering time and chunk loss in Section 6.4.

Figure 20 shows the CDF of chunk interarrival times. The interarrival time of most packets are between -0.2 to 0.2. We observe that the jitters of interarrival time increase with the streaming rate. This is because the number of substreams and number of delivery path both increase with the streaming rate. There is a larger chance that the chunks are delayed due to the network environment change. We also show the standard deviation of chunk interarrival times in Figure 21. The standard deviation (jitter) increases significantly with the streaming rate.

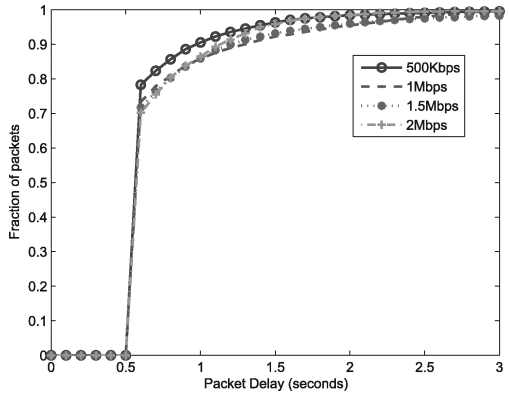


Fig. 17. Chunk delay CDF.

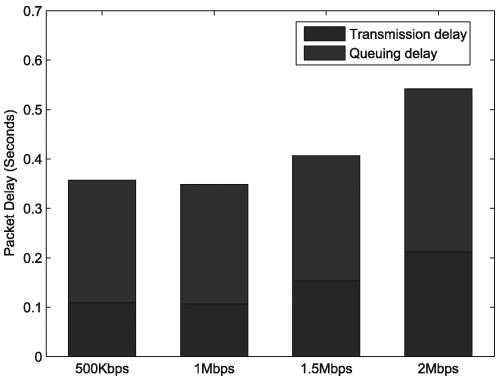


Fig. 18. Delay component.

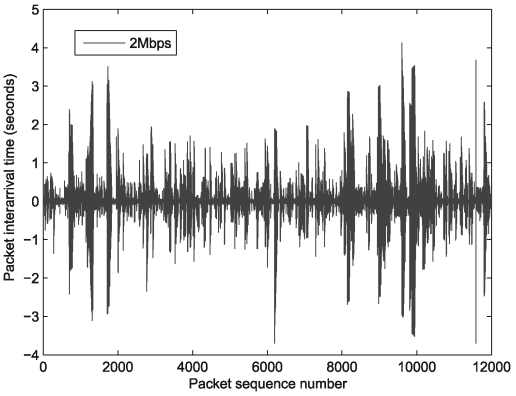


Fig. 19. Chunk interarrival time versus sequence number.

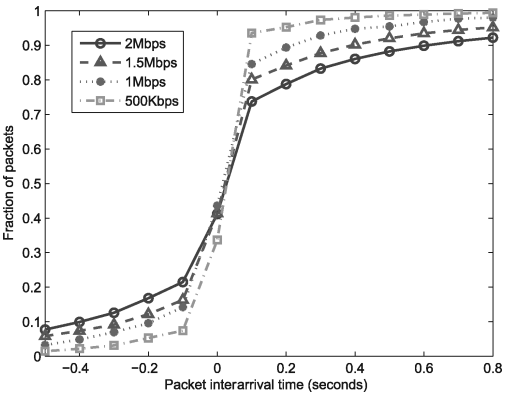


Fig. 20. Chunk interarrival time CDF.

Figures 22 and 23 shows the chunk interarrival time of a single substream (with bitrate 250kbps). The substream delivered by the direct connection between the source and the server has higher jitters than the two-hop connection, that is, from source to helper, then to server. It confirms with our observation in Section 6.1 that long-haul connections tend to have more bandwidth fluctuation. Therefore deploying helpers in the middle of a link with large RTT can significantly reduce the jitters of the chunk interarrival time.

6.4. Buffer Sizing

In this section we study the sizing of the buffer at servers and its impacts in global high-bitrate streaming. When the first chunk arrives at the server, it is delayed by a short period of time before it is delivered to the end users to accommodate the jitters between chunk arrivals, that is, chunk interarrival time. We call this period of delay "buffering time." Figure 24 shows the chunk loss rate versus buffering time. Recall from Equation (17), a chunk is considered "lost" if it arrives later than its play playback moment. The larger buffering time will lead to lower chunk loss. However on the other hand, it increases the playback delay of the end users since the chunks need to wait longer till they are played back. We observe that given a certain buffering time, the loss of a high-bitrate video, that is, 2 Mbps, is significantly larger than videos with

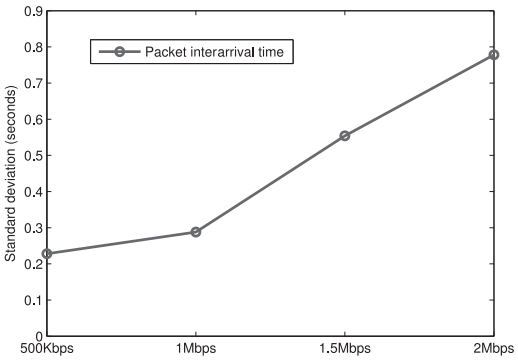


Fig. 21. Standard deviation of chunk interarrival time.

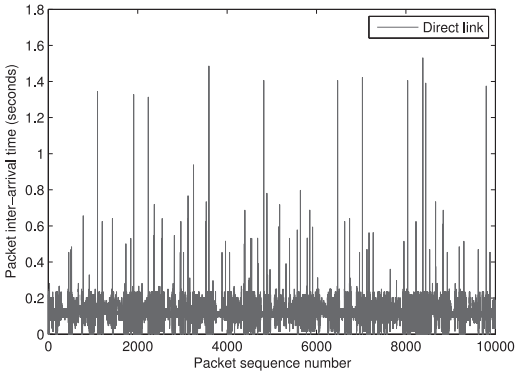


Fig. 22. Chunk interarrival time of a single substream on direct link.

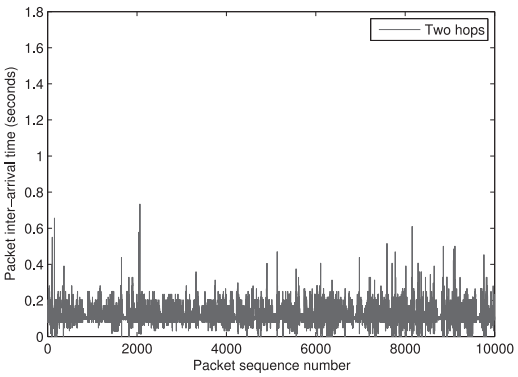


Fig. 23. Chunk interarrival time of a single substream on two-hop link.

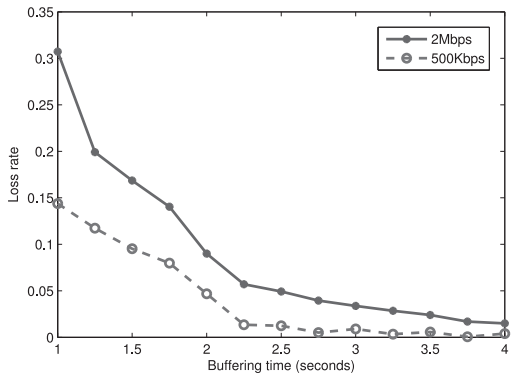


Fig. 24. Loss rate versus buffering time.

a lower bitrate, that is, 500kbps. Therefore, to maintain stream continuity, a larger buffering time must be given to multi-Mbps videos (3.5 seconds to achieve less than 3% loss rate). Comparing to transmission delay and queuing delay, the buffering time makes a major component in the playback delay of global high-bitrate live streaming.

In order to achieve a certain stream continuity, videos with different bitrates need different buffering time. Figure 25 shows the total playback delay of different streaming rates given 3% loss rate. Recall that the playback delay can be calculated with Equation (16). Given a target loss rate, streaming high-bitrate video requires longer buffering time in order to accommodate the high jitters of chunk arrival. Therefore the total playback delay is also higher for high-bitrate streaming.

7. CONCLUSIONS

In this article we address the design and optimization of a global live streaming network to achieve high streaming rate. In order to achieve low delay and high bitrate, we use overlay helpers (i.e., proxies which have no attached users) to provide rich path diversity, reduce scheduling delay, and improve system throughput and jitters in streaming. We present a realistic delay model of the network and formulate the delay optimization problem making use of helpers. We show that the problem is NP-hard. We propose an efficient algorithm called Stepping-stones which constructs an overlay

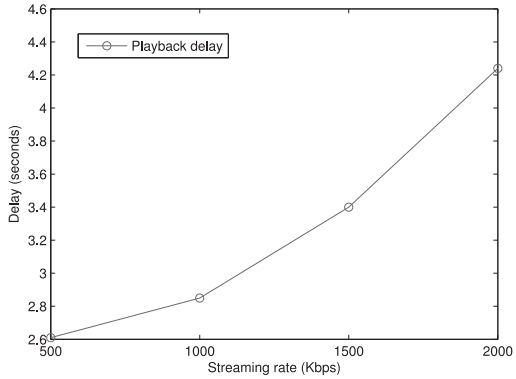


Fig. 25. Playback delay given a continuity requirement.

network involving helpers as needed to achieve low-streaming delay while meeting a certain high-streaming rate requirement.

We also implement a global streaming testbed based on Stepping-Stones, and conduct extensive experiments on the testbed to collect over a hundred hours streaming traces with bitrate ranging from 500kbps to 2Mbps. The results show that global multi-Mbps overlay live streaming over public Internet is achievable with the use of helpers due to the "multipath" and "multihop" benefits. Our experimental studies also provide important insights on the characteristics and design of push-based streaming in terms of packet jitter, loss and playback delay for video streaming on a global scale.

REFERENCES

Micah Adler, Ramesh K. Sitaraman, and Harish Venkataramani. 2011. Algorithms for optimizing the bandwidth cost of content delivery. *Comput. Netw.* 55, 18, 4007–4020.

E. Alessandria, M. Gallo, E. Leonardi, M. Mellia, and M. Meo. 2009. P2P-TV systems under adverse network conditions: A measurement study. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies* (*INFOCOM'09)*. 100–108.

Rafael Alvarez-Horine and Melody Moh. 2012. Experimental evaluation of Linux TCP for adaptive video streaming over the cloud. In *Proceedings of the IEEEGlobecom Workshops*. 747–752.

M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. 2003. SplitStream: High-bandwidth multicast in cooperative environments. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. 298–313.

Hyunseok Chang, Sugih Jamin, and WenjieWang. 2009. Live streaming performance of the Zattoo network. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference (IMC'09)*. ACM, New York, 417–429.

Hyunseok Chang, Sugih Jamin, and Wenjie Wang. 2011. Live streaming with receiver-based peer-division multiplexing. *IEEE/ACM Trans. Netw.* 19, 1, 55–68.

Jiancong Chen and S.-H. Gary Chan. 2001. Multipath routing for video unicast over bandwidth-limited networks. In *Proceedings of the IEEE Global Telecommunications Conference* (*Globecom'01)*. 1963–1967.

Cyril Concolato, Jean Le Feuvre, and Romain Bouqueau. 2011. Usages of DASH for rich media services. In *Proceedings of the 2nd Annual ACM Conference on Multimedia Systems (MMSys'11)*. 265–270.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* 3rd Ed. MIT Press.

Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York.

Yifeng He and Ling Guan. 2010. Solving streaming capacity problems in P2P VoD systems. *IEEE Trans. Circuits Syst. Video Technol.* 20, 11, 1638–1642.

Chia-Wei Hsiao andWen-Jiin Tsai. 2010. Hybrid multiple description coding based on H.264. *IEEE Trans. Circuits Syst. Video Technol.* 20, 1.

Joe W. Jiang, S.-H. Gary Chan, Mung Chiang, Jennifer Rexford, Dongni Ren, and Bin Wei. 2012. Global 1Mbps peer-assisted streaming: Fine-grain measurement of a configurable platform. *IEEE Trans. Multimedia* 14, 5, 1456–1468.

Wenjie Jiang, S.-H. Gary Chan, Mung Chiang, Jennifer Rexford, K.-F. Simon Wong, and C.-H. Philip Yuen. 2010. Proxy-P2P streaming under the microscope: fine-grain measurement of a configurable platform. In *Proceedings of the 19th International Conference on Computer Communications and Networks*.

Xing Jin, Kan-Leung Cheng, and S.-H. Gary Chan. 2009. Island Multicast: Combining IP Multicast with Overlay Data Distribution. *IEEE Trans. Multimedia* 11, 5, 1024–1036.

L. Kontothanassis, R. Sitaraman, J. Wein, D. Hong, R. Kleinberg, B. Mancuso, D. Shaw, and D. Stodolsky. 2004. A transport layer for live streaming in a content delivery network. *Proc. IEEE* 92, 9, 1408–1419.

D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. 2003. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. 282–297.

Robert Kuschnig, Ingo Kofler, and Hermann Hellwagner. 2010a. An evaluation of TCP-based rate-control algorithms for adaptive internet streaming of H.264/SVC. In *Proceedings of the 1st Annual ACMSIGMM Conference on Multimedia Systems (MMSys'10)*. ACM, New York, 157–168.

R. Kuschnig, I. Kofler, and H. Hellwagner. 2010b. Improving Internet Video Streaming Performance by Parallel TCP-Based Request-Response Streams. In *Proceedings of the 7th IEEE Consumer Communications and Networking Conference (CCNC'10)*. 1–5.

Xia Li, Rua Zou, Xinchao Zhao, and Fangchun Yang. 2010. A grouping algorithm of helpers in peer-to-peer video-an-demand systems. In *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT'10)*. IEEE Press, 497–501.

A. Medina, A. Lakhina, I. Matta, and J. Byers. 2001. BRITE: Universal topology generation from a user's perspective. In *Proceedings of the* IEEE *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*.

Christopher Müller, Stefan Lederer, and Christian Timmerer. 2012. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. In *Proceedings of the 4th Workshop on Mobile Video (MoVid'12)*. 37–42.

Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. 2010. The Akamai network: a platform for highperformance internet applications. *SIGOPS Oper. Syst. Rev.* 44, 3, 2–19.

Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. 1998. Modeling TCP throughput: a simple model and its empirical validation. *SIGCOMM Comput. Commun. Rev.* 28, 4, 303–314. DOI:http://dx.doi.org/10.1145/285243.285291

Dongni Ren and S.-H. Gary Chan. 2012. Achieving high-bitrate overlay live streaming with proxy helpers. In *Proceedings of 19th International Packet Video Workshop (PV'12)*.

Dongni Ren, Y.-T. Hillman Li, and S.-H. Gary Chan. 2008. On reducing mesh delay for peer-to-peer live streaming. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'08)*.

D. Ren, Y. T. H. Li, S. H. G. Chan, et al. 2009. Fast-mesh: a low-delay high-bandwidth mesh for peer-to-peer live streaming. *IEEE Trans. Multimedia* 11, 8.

S. Sengupta, Shao Liu, Minghua Chen, Mung Chiang, Jin Li, and P. A. Chou. 2011. Peer-to-Peer Streaming Capacity. *, IEEE Trans. Inf. Theory* 57, 8, 5072–5087.

Ao-Jan Su and Aleksandar Kuzmanovic. 2008. Thinning akamai. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC'08)*. ACM, New York, 29–42.

Alex Borges Vieira, Ana Paula Couto da Silva, Francisco Henrique, Glauber Goncalves, and Pedro de Carvalho Gomes. 2013. SopCast P2P live streaming: Live session traces and analysis. In *Proceedings of the 4th ACM Multimedia Systems Conference (MMSys'13)*. 125–130.

Long Vu, Indranil Gupta, Klara Nahrstedt, and Jin Liang. 2010. Understanding overlay characteristics of a large-scale peer-to-peer IPTV system. *ACM Trans. Multimedia Comput. Commun. Appl.* 6, 4, Article 31.

Bing Wang, Jim Kurose, Prashant Shenoy, and Don Towsley. 2008. Multimedia streaming via TCP: An analytic performance study. *ACM Trans. Multimedia Comput. Commun. Appl.* 4, 2, Article 16.

JiajunWang, Chuohao Yeo, Vinod Prabhakaran, and Kannan Ramch. 2007. On the role of helpers in peer-to-peer file download systems: Design, analysis and simulation. In *Proceedings of the International Workshop on Peer-To-Peer Systems*.

Chuan Wu, Baochun Li, and Shuqiao Zhao. 2008. Exploring large-scale peer-to-peer live streaming topologies. *ACM Trans. Multimedia Comput. Commun. Appl.* 4, 3, Article 19.

Chuan Wu, Baochun Li, and Shuqiao Zhao. 2011. On dynamic server provisioning in multichannel P2P live streaming. *IEEE/ACM Trans. Netw.* 19, 5, 1317–1330.

Hao Yin, Xuening Liu, Tongyu Zhan, Vyas Sekar, Feng Qiu, Chuang Lin, Hui Zhang, and Bo Li. 2009. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky. In *Proceedings of the 17th ACM International Conference on Multimedia (MM'09)*. 25–34.

H. Yin, X. Zhang, T. Zhan, Y. Zhang, G. Min, and D. Wu. 2013. NetClust: A framework for scalable and Pareto-optimal media server placement. *IEEE Trans. Multimedia* 15, 8.

Xiaoqun Yuan, Hao Yin, Geyong Min, Xuening Liu, Wen Hui, and Guangxi Zhu. 2013. A suitable server placement for peer-to-peer live streaming. *J. Supercomputing* 64, 3, 1092–1107.

Received October 2013; revised July 2014; accepted July 2014