Achieving High-Bitrate Overlay Live Streaming with Proxy Helpers

Dongni Ren *S.-H. Gary Chan*
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
Email: {tonyren, gchan} @cse.ust.hk

***Abstract*—Meeting a high bitrate requirement (say, 1 Mbps) in overlay live streaming is challenging. We consider the design and optimization of an overlay network formed by distributed proxies for high-bitrate live streaming. The video stream is divided into substreams and pushed via multiple trees to all the proxy *servers* with users. To effectively overcome bandwidth bottlenecks, we employ proxy *helpers* to provide rich path diversity. They do not have any attached users, and hence may forward any arbitrary subset of the substreams. In this way, the helpers serve as "stepping stones" to provide full streams to the servers.**

A critical issue is *how* to best use the proxy helpers to minimize delay meeting a certain streaming rate requirement. We first model the network by capturing various delay and bandwidth components. We formulate the problem and show that it is NP-hard. We then propose an efficient algorithm called *Stepping-Stones*. Our results based on simulation on real Internet topologies show that the algorithm outperforms other overlay protocols by effectively making use of helpers to achieve low delay and high streaming rate.

I. Introduction

Achieving overlay live streaming of high bitrate over global public Internet is challenging. This challenge comes from two factors. First, multimedia streaming requires high sustained bandwidth overcoming network bottlenecks. Second, overlay delay has to be minimized so that live streams can be delivered to end users in a timely manner. Therefore, although we have witnessed many successful stored streaming applications (such as YouTube, Hulu, Youku, etc.), there is still much room for improvement regarding the delivery of *high* bitrate *live* streams over the global Internet.

In order to provide overlay live services, content or service providers usually deploy proxies at different locations of the Internet to serve end users, the so-called "media cloud" [1]. These proxies are mostly stable, though they may be introduced or removed at any time. They form an overlay to receive live streams from the source or other proxies to serve their local users. To achieve low delay, the stream is *pushed* from the source to the proxies. The most challenging issue is how to minimize the delay from the source to all the proxies with users while meeting the streaming rate requirement.

This work was supported, in part, by the General Research Fund from the Research Grant Council of the Hong Kong Special Administrative Region, China (611209), and Proof-of-Concept Fund at the HKUST (PCF.005.09/10 & PCF.005/10/11).

User delay in the network is due to the aggregation of two components on the overlay, scheduling delay and propagation delay. *Scheduling delay* is defined as the time elapsed from a parent node receiving a segment (or chunk) to the instant that the segment is transmitted to its children. It is clear that the more children a proxy serves, the higher the scheduling delay is. *Propagation delay* is the time for the segment to travel from one proxy to the next one. It is usually reflected by the "ping" or round-trip time (RTT) distance. For push-based streaming (i.e., typically with small segments) over global network, both delays have to be considered. [1] Both delays accumulate with the number of proxies and number of hops from the source. If the overlay is not carefully designed, high overall delay will lead to undesirable results.

Traditionally, proxies are deployed at where active users are, which often leads to a relatively sparse network. Because of the long-haul connections between proxies, the end-to-end throughput would be limited. In order to support global high bitrate streaming (e.g., higher than 1 Mbps), we need to overcome such end-to-end bandwidth bottleneck [2]. Many previous studies assume that network edge is the only bottleneck. This assumption hence no longer holds for a global network, where end-to-end bandwidth needs to be considered as well.

In order to overcome bandwidth bottlenecks to support high bitrate, we study the use of *helpers*. Helpers are proxies deployed in the overlay with no (existing) end users. Therefore, they do not have to receive the full stream to serve users. *Servers*, on the other hand, are proxies with active users and have to receive full streams. Previous work has not considered enough the use of helpers for live streaming. In this paper, we show how helpers can improve streaming performance through the following:

- *Providing rich path diversity:* Helpers provide path diversity in the network, by offering a rich set of alternate paths as "stepping stones" from one server to another. As bandwidth can be aggregated along multiple paths, this overcomes bandwidth bottlenecks of end-to-end connections.

[1] Consider a practical push-based streaming network with segment size of 100 kbits and end-to-end bandwidth of 500 kb/s. A proxy with two children then has scheduling delay of 400ms. Given that typical ping distances in a global setting are 50ms to 100ms, neither delays can be ignored.

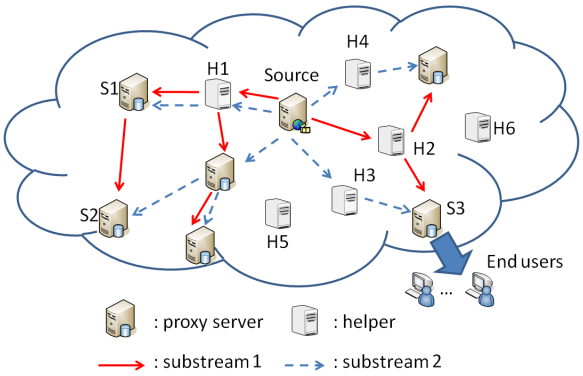


Fig. 1: A high-bitrate live streaming network with helpers.

- *Reducing scheduling delay:* The worst-case scheduling delay is proportional to the number of children that a proxy (i.e., either server or helper) has. The use of helpers can reduce the average number of children of the proxies, and hence the scheduling delay. This reduction may reduce the overall delay of the network.
- *Increasing system throughput and capacity:* The presence of helpers increases the available uploading bandwidth of the network. This bandwidth can be used to serve the servers in need. This increases system capacity to accommodate more servers.

We show in Figure 1 the streaming network under consideration. The video is divided into multiple substreams, which are pushed from the source to all servers via multiple spanning trees (which may overlap on overlay edges). The aggregation of all the trees is hence a *mesh*. The servers have to receive all the substreams, either from streaming source, other servers or helpers, to aggregate as a full stream. Helpers also participate in streaming and serve as stepping stones (intermediate nodes) in one or more delivery trees so that the substreams can reach servers meeting bandwidth requirement with low delay. For examples, helper H_1 receives both substream 1 and substream 2 (i.e., full stream in this example), and then delivers them to proxy server S_1. Helpers H_2, H_3 and H_4 receive and forward only partial stream (one substream in this example), while some other helpers (H_5 and H_6) does not participate in the stream distribution.

There has not been work studying the *construction and optimization* of the helper-based streaming cloud as discussed above. We investigate such a network, and our contributions are as follows:

1) *Problem formulation and complexity analysis:* We present a realistic delay model with helpers. The model captures various important delay components and bandwidth constraints for the streaming cloud, such as end-to-end bandwidth, uplink bandwidth, scheduling delay, propagation, etc. We formulate the optimization problem of constructing overlay trees for each substream, which is to minimize network diameter while meeting the streaming rate requirement. We analyze its complexity and prove that it is NP-hard.
2) *A heuristic (Stepping-Stones):* We propose an efficient heuristic making use of helpers called *Stepping-Stones (SS)* which meets bandwidth requirement while achieving low streaming delay.
3) *Simulation studies:* We conduct extensive simulation study real Internet topologies to evaluate the performance of our proposed algorithm. We show that helpers can significantly reduce streaming delay, achieve high streaming rate, and improve system capacity.

The rest of this paper is organized as follows. We first review related work in Section II. Then we present the formulation of the problem and its complexity analysis in Section III. In Sections IV, we discuss the algorithm *Stepping-Stones*, which is to construct a low-delay overlay streaming backbone with helpers. Illustrative simulation results and comparisons are presented in Section V. We conclude in Section VI.

II. Related Work

To build an overlay for live streaming, one common approach is to use CDN (Content Distribution Network). There is a large body of work on CDN, which may be broadly grouped as follows: Data replication and cooperative caching [3]; User request redirection [4]; CDN deployment and configuration [5]. Despite this large body of work, there is little, if any, work studying and optimizing CDN overlay to *live streaming* issues. As opposed to caching static data content, live streaming calls for a careful design of overlay to support *sustained* connections between proxies so as to meet *streaming rate* requirement with *low delay*.

There has been much work on peer-to-peer (P2P) streaming [6], [7]. While this body of work addresses different aspects of streaming issues, it has not addressed how to *optimize* streaming delay making use of *helpers* while meeting a certain bitrate requirement.

The concept of helper has been mentioned in the context of P2P Video on Demand (VOD), where nodes with residual bandwidth are used to reduce server loads, increase streaming capacity and distribute video resources [8]. However, how to make use of helpers to support *live* streaming has not been addressed, where meeting bandwidth requirement with minimum delay is the major concern. While much of the traditional work constructs the overlay in a rather ad hoc manner, ours focuses on its optimization to support high streaming rate and low delay. Our study shows that helpers can substantially improve streaming rate, reduce user delay and increase system capacity.

III. Problem Formulation and Complexity

A. 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^p_{ij} from node i to node j in the physical network.

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, i.e., s is the number of substreams of the 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 aggregate 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, i.e.,

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

Therefore, $a_{i1} = a_{i2} = \ldots = a_{ik} = 1$ 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, i.e.,

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

where the "1" is due to the source (which does not need to be supplied with any substream). The aggregate incoming bandwidth of each node must be larger than the streaming rate, i.e.,

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

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

The worst-case scheduling delay from node j to node i, denoted as d^s_{ji}, is given by

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

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 $\langle j, k\rangle$.

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, i.e.,

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

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

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

We state below 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),

$$\min \max_{i \in P} D_i \quad (7)$$

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

B. Problem Complexity

We show in this section that the complexity of our problem is NP-hard. 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 proxy 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 polynomially reducible to MDSH.

IV. Stepping-Stones Algorithm

In this section we present our heuristic called *Stepping-Stones (SS)*, which constructs a streaming overlay given network information (i.e. the inter-proxy distances and bandwidths). We first present the details of the heuristic in Section IV-A, followed by its complexity analysis in Section IV-B

A. Algorithmic Details

To construct an overlay with minimum 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, *Stepping-Stones (SS)* uses two steps to construct s delivery trees. It first constructs s delivery trees spanning all servers through iterations. In each iteration, it adds one server into one partially constructed delivery tree. After s delivery trees are constructed, helpers are then added

to the trees in order to reduce delay. The intuition behind *SS* is that we only include those useful helpers in each substream tree, and hence helpers only participate in streaming 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 is initialized containing only the streaming source S. For every node i that is not in T_k, we calculate the potential delay of node i in tree T_k as

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

where $D_i^k(j)$ is the delay of node i in tree k if it connects to node j as its parent. Let $d_{ij} = d_{ij}^s + d_{ij}^p$. Recall that each connection $\langle i, j \rangle \in E$ has a maximum transmission rate w_{ij}, which is the value that the total number of substreams from i to j cannot exceed. Let t_{ij} be the existing traffic (in number of substreams) from i to j and r_{ij} be the residual bandwidth from i to j. r_{ij} clearly can be written as

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

Since the link between node j and i may not have sufficient throughput to support the substream, in this case we employ helpers to bypass the bottleneck link, and $D_i^k(j)$ can be calculated as

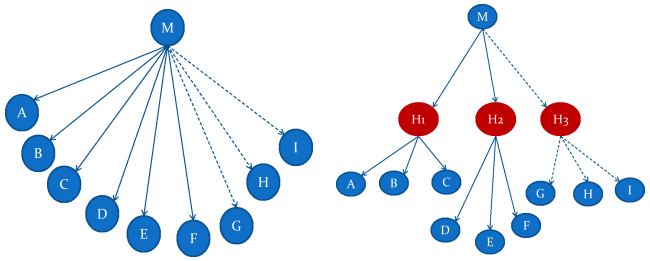
$$D_i^k(j) = \begin{cases} D_j^k + d_{ji} & \text{if } r_{ji} > 0, \\ \min\limits_{\forall h \in H, r_{jh}>0, r_{hi}>0} (D_j^k + d_{jh} + d_{hi}) & \text{otherwise.} \end{cases} \tag{10}$$

We then choose the node i with lowest potential delay and connect it to the corresponding tree T_k, i.e.,

$$\arg_{\{i,k\}} (\min_{\forall i \notin T_k, \forall k} \hat{D_i^k}). \tag{11}$$

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

After the delivery trees span all servers, helpers are added to the overlay trees in order to reduce delay of the servers. Recall from Equation 4 that the worst-case scheduling delay is proportional to the number of children that a proxy serves. Therefore we use helpers to *offload* the busy proxies by taking over their children in each delivery tree. Figure 2 shows an example of the *offload* process and the adaptation of the delivery tree. Node M serves 6 nodes in delivery tree T_1 (solid lines) and 3 nodes in delivery tree T_2 (dashed lines) before offload. All 9 children of M have the same worst-case scheduling delay. In Figure 2B we employ 3 helpers, $H1, H2$ and $H3$, each of which serves a subset of M's children, and the scheduling delay of the nodes are significantly reduced. Therefore in the helper adding step, we iterate through every proxy in the delivery trees and check whether employing new helpers is benificial or not. If offloading the proxy leads to a reduce of overall delay, we connect the new helpers to the proxy, and then let the helpers serve the children of the proxy.



(a) Before offload. (b) After offload.

Fig. 2: An example of adding helpers.

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

B. Algorithmic Complexity

The complexity of the *SS* algorithm is $O(s^2|V|^2|H||P|)$. In the tree construction step, adding one server to one delivery tree takes $O(s|V|^2|H|)$ time and there are $O(s|P|)$ iterations in total. Therefore the tree construction step takes $O(s^2|V|^2|H||P|)$ time. In the helper adding step, we iterate through every node in the trees and it needs $O(s|V||H|)$ time. Therefore the total complexity of *SS* is $O(s^2|V|^2|H||P|)$.

V. Illustrative Simulation Results

In this section we present illustrative simulation results on the performance of our algorithm *SS*. 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 inter-connected 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). We set the segment size as 100 kbits and the streaming rate of a substream as 400 kbps. 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. 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 following metrics:

- *Worst-case delay:* The worst-case delay is the maximum time taken for a packet to travel from the streaming source

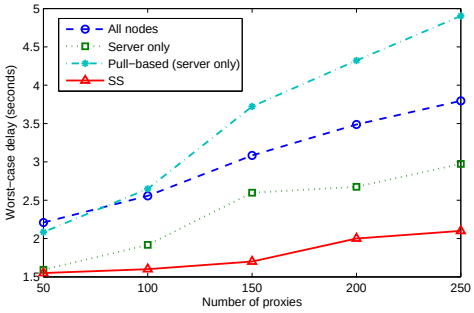


Fig. 3: Worst-case delay versus number of proxies with of them servers.

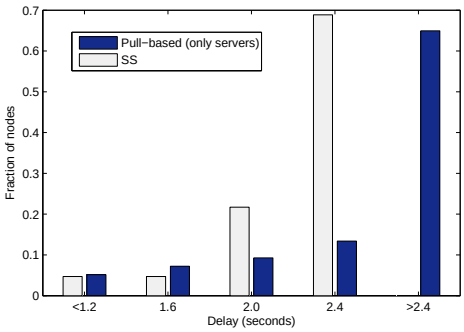


Fig. 4: Delay distribution.

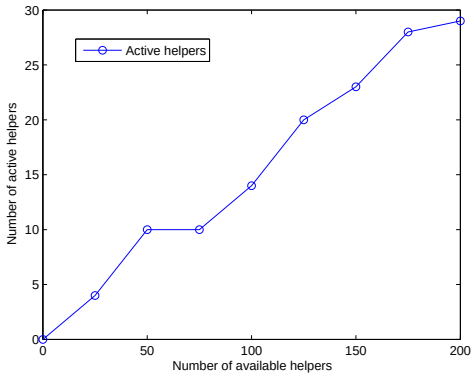


Fig. 5: Activated helpers versus available helpers.

to the servers in the overlay network. Our algorithms 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*.

We compare SS with three other schemes. 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 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. *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.

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.

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

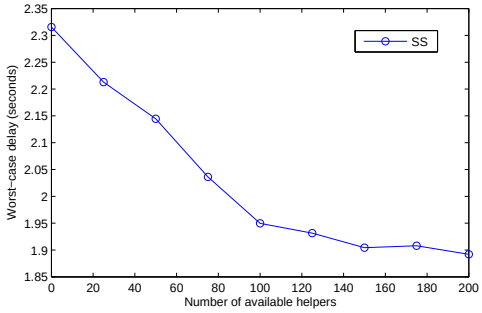


Fig. 6: Worst-case delay versus the number of helpers.

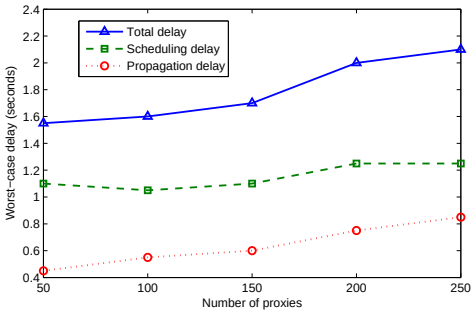


Fig. 7: Delay components.

the network. Normally, as observed in other experimental studies [9], 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 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.

VI. Conclusion

In this paper 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 helpers (i.e., proxies which have no attached users) to provide rich path diversity, reduce scheduling delay and increase system throughput and capacity. They work as "stepping stones" to forward substreams to the servers (i.e., proxies with attached users).

We present a realistic delay model of the network and formulate the delay optimization problem making use of

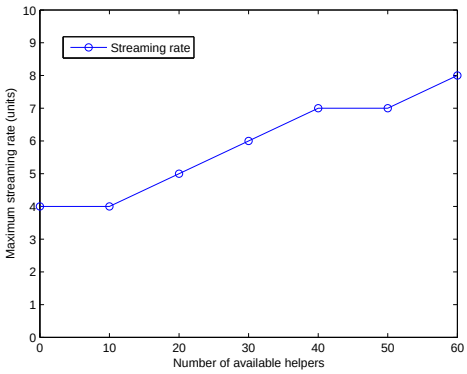


Fig. 8: Streaming rate versus number of available helpers.

helpers. We show that the problem is NP-hard. We propose a efficient algorithm *Stepping-stones*, or *SS*, to address the problem for a proxy live streaming network.

We conduct extensive simulation studies on real Internet topologies to evaluate the performance of our proposed algorithms. The results show that *SS* efficiently utilize helpers as "stepping stones" to achieve low delay and high streaming rate. The use of helpers can substantially improve streaming delay and bitrate.

In the future we will study the use of proxy helpers in streaming networks with multiple sources and multiple channels. We will also implement *Stepping-Stones* and carry out experimental studies on a global network to show the benefits of proxy helpers.

References

[1] W. Zhu, C. Luo, J. Wang, and S. Li, "Multimedia cloud computing: Directions and applications," *Special Issue on Distributed Image Processing and Communications, IEEE Signal Processing Magazine*, vol. 28, May 2011.

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

[3] S. Sivasubramanian, M. Szymaniak, G. Pierre, and M. van Steen, "Replication for web hosting systems," in *ACM Computing Surveys (CSUR)*, vol. 36, Sep. 2004.

[4] A. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante, "Drafting behind akamai: inferring network conditions based on CDN redirections," in *IEEE/ACM Transactions on Networking (TON)*, vol. 17, 2009.

[5] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao, "Moving beyond end-to-end path information to optimize CDN performance," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, ser. IMC '09. New York, NY, USA: ACM, 2009, pp. 190–201.

[6] Y. Zhou, D.-M. Chiu, and J. C. S. Lui, "A simple model for chunk-scheduling strategies in P2P streaming," *IEEE/ACM Trans. Netw.*, vol. 19, pp. 42–54, Feb. 2011.

[7] D. Ren, Y.-T. H. Li, and S.-H. G. Chan, "On reducing mesh delay for peer-to-peer live streaming," in *IEEE INFOCOM*. Phoenix, Arizona: IEEE, Apr. 2008.

[8] Y. He and L. Guan, "Solving streaming capacity problems in P2P voD systems," in *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, 2010, pp. 1638–1642.

[9] W. Jiang, S.-H. G. Chan, M. Chiang, J. Rexford, K.-F. S. Wong, and C.-H. P. Yuen, "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 (ICCCN) (Invited paper)*, 2-5 Aug. 2010.