MixNStream: Multi-Source Video Distribution with Stream Mixers*

C.-H. Philip Yuen S.-H. Gary Chan
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong, China
chyuen@cse.ust.hk gchan@cse.ust.hk

ABSTRACT

Many Internet streaming applications, such as distributed surveillance, multimedia webcasting and video thumbnails for Internet TV channel browsing, require video *mixers* where streams are aggregated as a single stream before presenting to users. We consider in this paper a multi-source streaming network with distributed mixers, where streams originated from multiple sources are mixed before presented to distributed users. We are interested in minimizing the worst-case delay from the source to users via the mixers. We propose an adaptive and distributed protocol called MixNStream, which continuously reduces the network diameter in the presence of churns. Through simulations on Internet-like topologies, we show that MixNStream achieves substantially better performance as compared with the state-of-the-art in terms of network delay and network stress.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Algorithms, Design, Performance

Keywords

Distributed protocol, mixers, multi-source, peer-to-peer network, proxies

*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 the Hong Kong Innovation and Technology Fund (ITS/097/09FP).

AVSTP2P'10, October 29, 2010, Firenze, Italy.

1. INTRODUCTION

The Internet has become an important vehicle to carry multimedia traffic. We have witnessed in recent years the development and deployment of many Internet streaming applications, such as Internet TV, conferencing, surveillance, etc. These applications often involve multiple video sources generating streams to be distributed to users. These streams may need to be aggregated as a single stream before presenting to users. Such aggregation can be done with a stream *mixer*, which combines multiple incoming streams and outputs a single integrated stream on-the-fly (using techniques such as sub-sampling and transcoding).

As streaming applications continue to grow with more and more generating sources and distributed users, efficient stream mixing from distributed sources becomes an important issue. These "mixing" applications include:

- *Distributed surveillance*: For monitoring road traffic or large enterprises (such as airports or campuses), many distributed surveillance cameras may be set up. Some of these video streams may need to be aggregated (i.e., "mixed") and then streamed to many monitoring users distributed in the network.
- *Conference webcasting*: In a conferencing session (such as a share-holder meeting, discussion forum or panel), people holding the session may be distributed in the network. If the session is to be webcasted/broadcasted to distributed viewers, the multimedia streams of these people need to be mixed as a single stream before being received by users.
- *Video thumbnails for Internet TV channel browsing*: In a multi-channel Internet TV application, the streams of the channels are usually generated by sources distributed in the network. To support a stream of video thumbnails for channel browsing and selection, the streams from these sources should be mixed before presenting to users.

In a multi-source streaming network with distributed users, proxies are often set up to serve local users. These rather stable proxies play the role of better localizing/isolating the adverse effect of user churns (joins, leaves and failures). If the streams from different sources *were* to be aggregated at a fixed mixer before distributing to the proxies, it would not be network-efficient and vulnerable to single point of failure at the mixer (because streams have to first fan into the mixer before fanning out again). To address this, we consider the mixing function being *embedded and implemented*

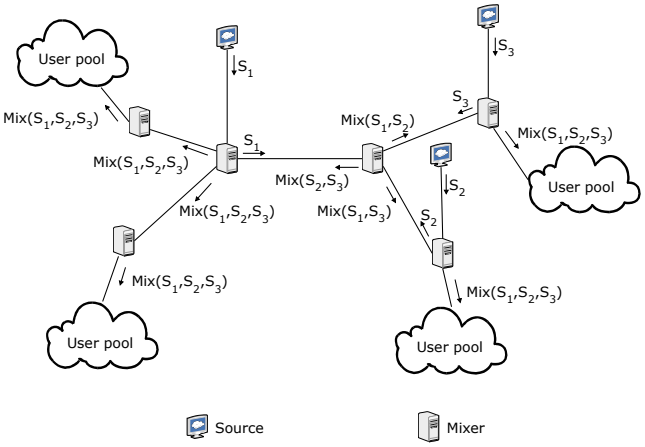

Figure 1: A multi-source network with stream mixing at proxies.

into the distributed proxies, so that stream mixing is done at the proxy network. Such integration of mixer function into proxies is not an issue because the proxies are usually of high processing capability. We hence will use "proxy" and "mixer" interchangeably in the remainder of this paper.

We show in Figure 1 the multi-source network with stream mixers under consideration, where *Mix(S_1, S_2)* means forming a stream with streams S_1 and S_2. We will mainly focus on the mixer-tier with multiple sources. Note that the user pool can be either a client-server or any peer-to-peer (P2P) network, and can be associated with the mixers by any joining protocol. Obviously for protocol simplicity and scalability, we should not maintain individual tree rooted at each source, but a single spanning tree among all the mixers so that streams are always forwarded in the same tree. A mixer sends to its mixer neighbor a mixed stream formed by all of the other neighbors. A mixer or proxy aggregates all the streams from its neighbors as a single stream before forwarding it to its local users. (In this way, the sources do not directly serve users. The mixers act as an intermediary for user requests, which supports anonymity of the sources and access control to the source streams.)

In the network, user delay is the sum of two parts: (1) the delay for a mixer to get the streams from all the sources in the mixer-tier; and (2) the path delay from that mixer to the user. We study the protocol at the peers[1] to minimize such delay subject to degree bounds of the mixers (the maximum connections it can establish with the sources and other mixers). Note that previous results on traditional streaming networks (without mixers) cannot be directly applied here. This is because in our mixer network streams are dynamically combined; consequently, incoming and outgoing bandwidths need to be accounted for differently. This fundamentally changes the design of the streaming protocol.

The key contributions of this work are: (1) we consider a mixer network and propose an adaptive and distributed algorithm called MixNStream, which builds a low-delay streaming network with mixers in the presence of churns (i.e., mixer joins and leaves). MixNStream continuously achieves lower overall delay with peers moving into better positions in a distributed fashion; (2) We conduct extensive simulation study on MixNStream. We show that MixNStream achieves substantially better performance as compared with an existing approach (in terms of delay, stress, etc.).

We briefly discuss previous work below. Many algorithms to reduce the network diameter are centralized in nature (see, for examples, [3, 4, 8]). Therefore, they are not applicable to large and dynamic group which we consider here. A distributed algorithm for a single-source two-tier network is presented in [2], which adapts the tree structure in upper-tier in response to the change in lower-tier. However, the solution cannot be directly extended to a *multi-source* network with *mixers* we consider here. STS (Shared Tree Streaming) is a distributed algorithm to construct a multi-source degree-bounded minimum-diameter spanning tree [1]. It adapts the overlay only when peer joins or leaves. In contrast, MixNStream *continuously* improves the overlay with peers moving into better positions in a distributed fashion. Although many distributed algorithms for overlay construction have been proposed (see, for examples, [1, 2, 5, 6, 7]), none of them fully addresses the problem of how to adaptively build a low-delay multi-source mixer overlay with peer churns. Hence they cannot be directly applied in this setting.

The rest of this paper is organized as follows. In Section 2, we define the terminology used in this paper. In Section 3 we present how peer dynamic are handled in MixNStream, and in Section 4 we present how MixNStream adaptively builds an overlay to reduce delay. Illustrative simulation results and comparison are discussed in Section 5. We conclude in Section 6.

2. TERMINOLOGY

Denote $\mathcal{S}$ and $\mathcal{M}$ the disjoint sets of all the sources and mixers, respectively. We model the mixer overlay as an undirected graph $G = (V, E)$, where V is the set of vertex representing the participating nodes given by $V = \mathcal{S} \cup \mathcal{M}$ and E is the set of overlay edges. Let l_{ij} be the latency of the edge $\langle i, j\rangle \in E$. Denote $D(u, v)$ the delay of the streaming path from node u to node v. We show in Table 1 some important symbols used in this paper.

Let $T(V, E_T)$ be the entire overlay tree constructed out of $G(V, E)$, where $E_T \subseteq E$. Denote $T_{\langle i,j\rangle}$ (or $T_{\langle j,i\rangle}$) the partial tree of T that contains mixer j (or i) after the removal of the overlay link $\langle i, j\rangle$. Clearly,

$$T_{\langle i,j\rangle} \cup \langle i, j\rangle \cup T_{\langle j,i\rangle} = T(V, E_T). \quad (1)$$

Further denote $\mathcal{S}_{\langle i,j\rangle}$ and $\mathcal{M}_{\langle i,j\rangle}$ sets of all the sources and mixers, respectively, in $T_{\langle i,j\rangle}$.

Let $\Delta(m)$ be the worst-case delay from all the source(s) directly attached to mixer m to m given by

$$\Delta(m) = \max_{s:\forall s\in\mathcal{S},\langle s,m\rangle\in E_T} l_{sm}. \quad (2)$$

Define $\delta(m)$ the worst-case delay from *any* source in T to mixer m given by

$$\delta(m) = \max_{s\in\mathcal{S}} D(s, m). \quad (3)$$

Clearly, $\Delta(m) \le \delta(m), \forall m \in \mathcal{M}$. Further define $\delta_{\langle i,j\rangle}(m)$ the worst-case delay from *any* source in $T_{\langle i,j\rangle}$ to mixer m given by

$$\delta_{\langle i,j\rangle}(m) = \begin{cases} \max_{s\in\mathcal{S}_{\langle i,j\rangle}} D(s, m), & \text{if } \mathcal{S}_{\langle i,j\rangle} \neq \emptyset; \\ -\infty, & \text{otherwise.} \end{cases} \quad (4)$$

[1]In this paper, a "peer" means either the source or mixer.

Table 1: Table of Nomenclature.

Symbols	Meanings
$T(V, E_T)$	Entire overlay tree of $G(V, E)$
$T_{\langle i,j \rangle}$	Partial tree of T that contains mixer j after the removal of the overlay link $\langle i, j \rangle$
$D(u, v)$	Delay of the streaming path from node u to node v
$\Delta(m)$	Worst-case streaming delay from the source directly attached to mixer m to m
$\delta(m)$	Worst-case streaming delay from *any* source in T to mixer m
$\delta_{\langle i,j \rangle}(m)$	Worst-case streaming delay from *any* source in $T_{\langle i,j \rangle}$ to mixer m
$\Gamma(m)$	Worst-case streaming delay from mixer m to the user in its user-pool
$\gamma(m)$	Worst-case streaming delay from mixer m to *any* user in T
$\gamma_{\langle i,j \rangle}(m)$	Worst-case streaming delay from mixer m to *any* user in $T_{\langle i,j \rangle}$
$\Pi(m)$	Worst-case delay of all the source-to-user paths that pass through mixer m

Let $\Gamma(m)$ be the worst-case delay from mixer m to the user in its user pool. We define $\gamma(m)$ the worst-case delay from mixer m to *any* user in T given by

$$\gamma(m) = \max_{m^* \in \mathcal{M}} \left(D(m, m^*) + \Gamma(m^*) \right). \quad (5)$$

Further define $\gamma_{\langle i,j \rangle}(m)$ the worst-case delay from mixer m to *any* user in $T_{\langle i,j \rangle}$ given by

$$\gamma_{\langle i,j \rangle}(m) = \max_{m^* \in \mathcal{M}_{\langle i,j \rangle}} \left(D(m, m^*) + \Gamma(m^*) \right). \quad (6)$$

Consider all the source-to-user streaming paths that pass through mixer m. There can be two cases: (1) the user is in mixer m's user pool; and (2) the user is *not* in mixer m's user pool. We define $\pi(m)$ worst-case delay of the paths in case (1), which is given by

$$\pi(m) = \delta(m) + \Gamma(m). \quad (7)$$

Further define $\widehat{\pi}(m)$ the worst-case delay of the paths in case (2), which is given by

$$\widehat{\pi}(m) = \max_{m^*: \forall m^* \in \mathcal{M}, \langle m, m^* \rangle \in E_T} \left(\delta_{\langle m^*, m \rangle}(m) + \gamma_{\langle m, m^* \rangle}(m) \right). \quad (8)$$

We define $\Pi(m)$ the worst-case delay of all the source-to-user paths that pass through mixer m, which is given by

$$\Pi(m) = \max \left(\pi(m), \widehat{\pi}(m) \right). \quad (9)$$

We show in Figure 2 an illustration example of the above symbols. In the figure, the delay on all the overlay links are the same. The path for $\delta(i)$ is shown in dashed line; the path for $\gamma(i)$ is shown in "-·-" line; the path for $\delta_{\langle i,j \rangle}(i)$ is shown in "-··-" line; the path for $\gamma_{\langle i,j \rangle}(i)$ is shown in dotted line; and the path for $\Pi(i)$ is shown in double line.

3. JOINS AND LEAVES IN MIXNSTREAM

Peer traffic can be highly dynamic. A peer may join, leave and failure at anytime. The overlay hence has to adapt to this network dynamic.

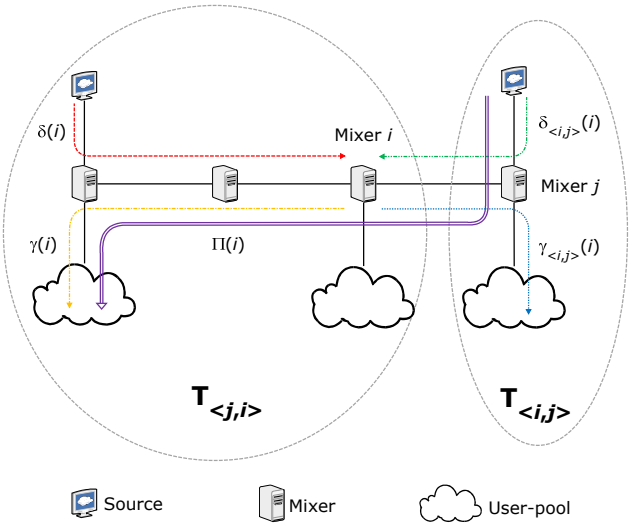

Figure 2: Terminology

3.1 Source

A joining source should connect to a mixer neighbor which leads to low delay of all the users. Upon the arrival of a new source s, it contacts a *Rendezvous Point* (RP) which returns a random list of mixers as the potential mixer neighbors. After receiving the list, the source s requests $\gamma(m)$ of each of the potential mixer neighbor m and evaluates $(\gamma(m) + l_{sm})$. It connects to the mixer neighbor m with available degree of the lowest $(\gamma(m) + l_{sm})$. It may request the mixer neighbors from the list (through gossip). Then the above process is repeated several times until a sufficiently good mixer is found.

When a source is about to leave, it informs its attached mixer. Upon detecting a source has left, the mixer it attached to updates its set of neighbors.

3.2 Mixer

A newly arrived mixer should look for a mixer neighbor which leads to low delay of all the source streams. Upon the arrival of a new mixer m, it receives a list of potential mixer neighbors from RP. It then requests $\delta(m^*)$ of each of the potential mixer neighbor m^* and evaluates $(\delta(m^*) + l_{mm^*})$ with respect to each of them. It may get more candidates through gossip. After computing the delays, m connects to mixer neighbor m^* with available degree of the lowest $(\delta(m^*) + l_{mm^*})$.

When a mixer is about to leave, it initiates a leave message to all of its source neighbors and mixer neighbors, asking them to re-join the network. In order to guarantee connectivity in the spanning tree after the re-join process, we put a special node, termed *virtual root*, in the mixer-tier network which acts as the root of mixer-tier (the virtual root may be co-located with the RP). The virtual root connects to exactly one of the mixers and will not leave the network (like RP). The only responsibility of the virtual root is to send the *Root-Path* which contains its identity to its mixer neighbor m. Mixer m will then concatenate its identity to the *Root-Path* and send to its mixer neighbors. Keeping the *Root-Path* is important as in the re-join process, the potential mixer neighbors returned by RP may be a descendant

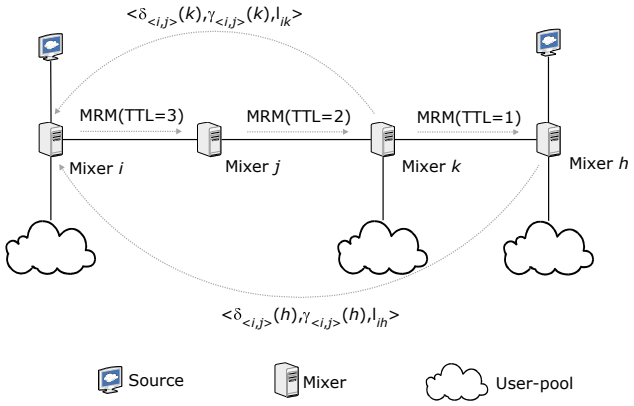

Figure 3: An example of mixer adaptation

of the repairing client. A loop will exist if a client takes its descendant as its parent. The repairing client eliminates looping (hence overlay disconnection) in the re-join process by examining whether the *Root-Path* of the potential parents contains its own identity or not.

A mixer may fail at anytime. To handle this, each mixer regularly sends its heartbeat to its neighbors. When a peer finds its neighbor fails, it runs the repair process.

4. ADAPTATION AND CONTROL MESSAGING

We define an adaptation as a peer moving into better positions (i.e., achieving lower delay) in the mixer-tier. Each peer periodically runs the adaptation algorithm to reduce delay. An adaptation is performed if and only if the worst-case source-to-end delay can be reduced.

4.1 Source Adaptation

A source s, which is attached to a mixer m, periodically requests $\gamma(m^*)$, where m^* is the neighbors of m. It evaluates $(\gamma(m^*) + l_{sm^*})$. The source picks the mixer neighbor with the lowest delay by disconnecting from mixer m and takes mixer m^* as its mixer neighbor.

4.2 Mixer Adaptation

A mixer m floods a *Migration Request Message* (MRM) to its mixer neighbors with a time to live (TTL) value which is decremented by 1 each time it is forwarded until it hits 0. Upon a mixer $\hat{m}$ receiving MRM, it replies m with a GRANT message which is a tuple of $\langle \delta_{\langle m,m^*\rangle}(\hat{m}), \gamma_{\langle m,m^*\rangle}(\hat{m}), l_{m\hat{m}} \rangle$ if it has available degree, where m^* is m's 1-hop mixer neighbor. We show in Figure 3 an example of this case.

Mixer m may receive a number of GRANT messages. For each replier $\hat{m}$, mixer m considers a new mixer network constructed by replacing $\langle m, m^* \rangle$ with $\langle m, \hat{m} \rangle$, and calculates $\Pi(m)$ for the network using the information in the tuple. The mixer neighbor which yields the smallest $\Pi(m)$ is chosen as the new neighbor.

4.3 Control Messaging

Peers gather the required information for making adaptation decisions using the idea in Distributed Aggregation Tree [9]. Each peer periodically sends its neighbors a control message for aggregation. As an example of aggregating $\delta(m)$, mixer m first collects l_{sm} from each of its source neighbors s. Mixer m hence computes $\Delta(m)$ according to Equation (2). Afterward, mixer m collects $\delta_{\langle m,m^*\rangle}(m^*)$ and l_{mm^*} from its mixer neighbor m^*, and computes

$$\delta_{\langle m,m^*\rangle}(m) = \delta_{\langle m,m^*\rangle}(m^*) + l_{mm^*}. \tag{10}$$

After computing $\delta_{\langle m,m^*\rangle}(m)$ with respect to each of its mixer neighbor m^*, it computes

$$\delta(m) = \max\left(\Delta(m), \max_{m^* \in m\text{'s mixer neighbors}} \delta_{\langle m,m^*\rangle}(m)\right). \tag{11}$$

Mixers gather other information (for example, Equations (5) and (9)) and compute the tuple in the GRANT message using similar mechanism.

Table 2: Baseline Parameters.

Parameters	Baseline Values
λ	2 req./minute
$1/\mu$	50 minutes
Adaptation interval	1 minute
Degree-bound	5
TTL	5
Number of sources	8
Diameter of user-pool	Truncated normal with mean 30 msec and standard deviation (s.d.) 20 msec

5. ILLUSTRATIVE SIMULATION RESULTS

We have carried out simulation to compare the performance of MixNStream with STS (Shared Tree Streaming) [1]. Note that STS does not consider the diameter in user-pools, and adapts the overlay only when peer joins or leaves (instead of continuously as in MixNStream). We first present the simulation environment and performance metrics in Section 5.1, followed by illustrative results in Section 5.2.

5.1 Simulation Setup and Metrics

In the simulation, we use Brite to generate a two-level top-down hierarchical topology consisting of 8 autonomous systems each of which has 625 routers. This gives us a total of 5,000 routers and about 20,000 links. Peers are attached to the routers randomly. Mixers arrive according to a Poisson process with rate λ (req./minute), and the sojourn time of the mixers are exponentially distributed with mean $1/\mu$ (minutes). Clearly, the number of mixers in the system is Poisson with mean λ/μ. Unless otherwise stated, we use the baseline values of the parameters according to Table 2.

We use the following evaluation metrics for our comparison with the previous work of STS [1]:

- *Source-to-end delay:* Source-to-end delay is the path delay from multiple sources to a mixer's user pool. We are mainly interested in its maximum (i.e., diameter) among all the mixers in this paper.

- *Network stress:* Network stress is defined as the average number of streams in an used underlay link.

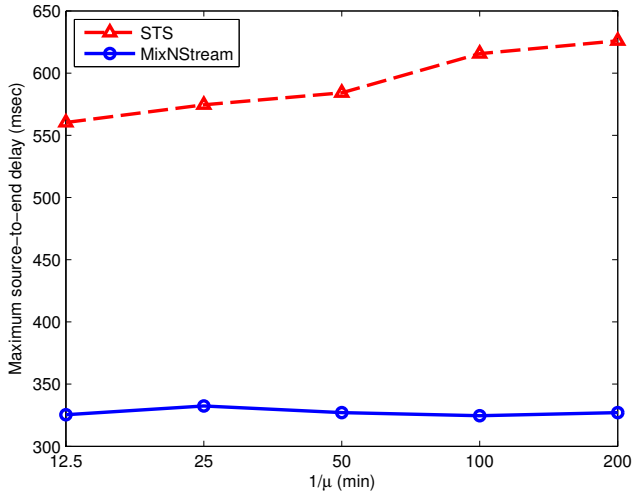

Figure 4: Maximum source-to-end delay versus $1/\mu$.

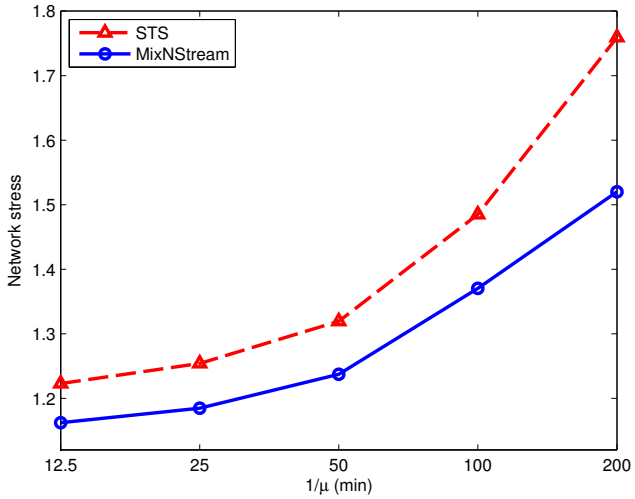

Figure 5: Network stress versus $1/\mu$.

5.2 Illustrative Results

We plot in Figure 4 the maximum source-to-end delay against average holding time $1/\mu$ (because λ fixed, the x-axis is also proportional to the average number of mixers). The delay in MixNStream remains rather constant and low as $1/\mu$ increases. This is because even the expected number of mixers in the system (λ/μ) increases, MixNStream effectively pushes mixers with larger diameter closer to the sources. The delay in STS is substantially higher. This is because its lack of consideration of user diameter. This shows that MixNStream is very effective in achieving low delay, due to its more optimized connections and connection improvements through periodic adaptation in the overlay.

We plot in Figure 5 the network stress against $1/\mu$. The stress in all schemes increase as number of mixers increases. This is because the number of connections increases when the network size increases. The stress in MixNStream is substantially lower than that of STS. This is because in MixNStream, mixers continuously move into better positions in the mixer-tier. This eliminates many long connections and saves much bandwidth in the network. More efficient routing leads to lower network stress.

We plot in Figure 6 the maximum source-to-end delay against the s.d. of the diameter of user pool. Note that when the s.d. is 0, the problem becomes minimizing the delay in mixer network, and the two schemes achieve similar performance, showing the effectiveness of MixNStream. When the s.d. increases, the diameter in user pool has more variations. MixNStream pushes mixers with larger diameter closer to the sources, which makes the delay in MixNStream remains rather low. Without considering the diameter in user pools, the diameter in STS increases dramatically. This shows that MixNStream is effective in putting mixers in appropriate position in the network to achieve low delay.

We plot in Figure 7 the maximum source-to-end delay against the mean of the diameter in user-pool. The delay in two schemes increase as the mean increases. However, the growth rate of the delay in MixNStream is substantially lower than that of STS. This is due to routing efficiency of MixNStream in the mixer network. Without considering the diameter in user-pools, the delay in STS increases with the rate approximately the growth rate of the mean.

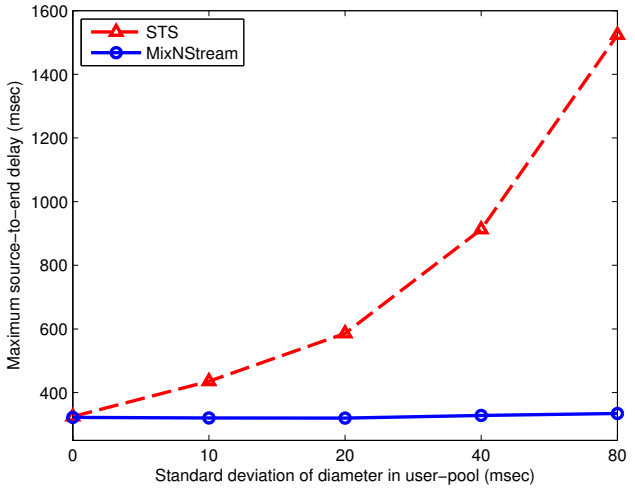

Figure 6: Maximum source-to-end delay versus standard deviation of diameter in user pool.

We plot in Figure 8 the graph of source-to-end delay against degree-bound. The delay decreases as degree-bound increases. This is because a larger degree-bound yields an overlay less "deep" and hence lower the delay. The delay of MixNStream is substantially lower than that of STS. This is because each peer in MixNStream periodically performs adaptation to move into better positions in the mixer-tier, while STS adapts the overlay only when peer joins or leaves.

We plot in Figure 9 the maximum source-to-end delay against TTL (flood scope). The delay decreases as TTL increases. This is because the larger TTL is, the better can the adaptation decision be made. Note that the case of TTL=2 corresponds to the gossip-based adaptation, while the case with TTL=∞ is the centralized scheme. Given that the delay reduction is not significant (about 10% in the figure), for simplicity and overhead consideration, TTL does not need to be high to achieve good performance.

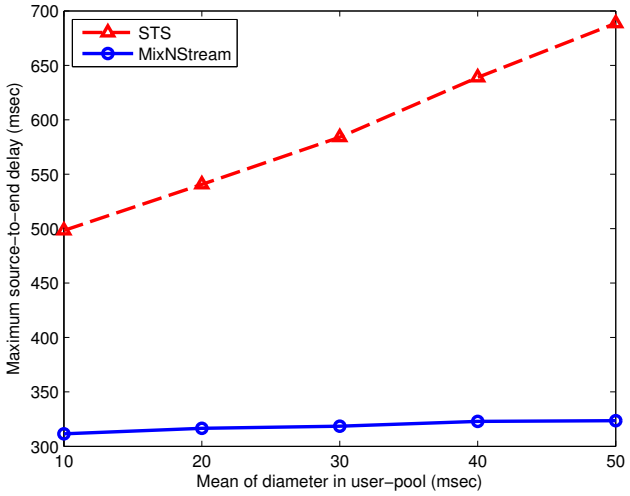

Figure 7: Maximum source-to-end delay versus mean of diameter in user-pool.

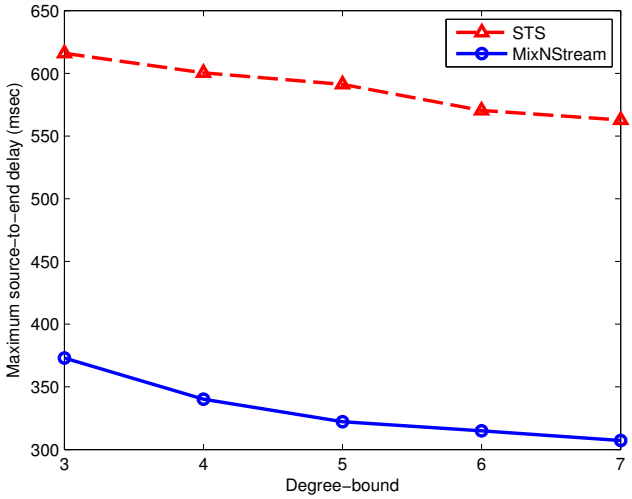

Figure 8: Maximum source-to-end delay versus degree-bound.

6. CONCLUSIONS

Many Internet streaming applications require video mixing from multiple sources. We consider in this paper a multi-source streaming network with distributed mixers serving local users, where streams originated from all the sources are aggregated before forwarding to users. We have studied how to minimize the network diameter (in terms of the worst-case source-to-end delay) by first formulating the problem. We propose and present an adaptive and distributed protocol called MixNStream, which continuously reduces the network diameter in the presence of peer churns. Simulation results show that MixNStream indeed achieves substantially lower network delay and network stress as compared with an existing approach of STS (Shared Tree Streaming).

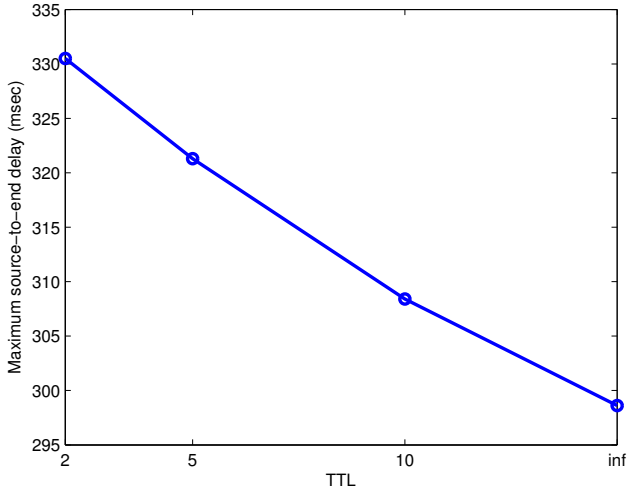

Figure 9: Maximum source-to-end delay versus TTL.

7. REFERENCES

[1] T. Baduge, A. Hiromori, H. Yamaguchi, and T. Higashino. A distributed algorithm for constructing minimum delay spanning trees under bandwidth constraints on overlay networks. *Systems and Computers in Japan*, 37(14):15–24, 2006.

[2] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. OMNI: an efficient overlay multicast infrastructure for real-time applications. *Computer Networks*, 50(6):826–841, 2006.

[3] N. Bansal, R. Khandekar, and V. Nagarajan. Additive guarantees for degree bounded directed network design. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 769–778. ACM, 2008.

[4] F. Huang, B. Ravindran, and V. Kumar. An Approximation Algorithm for Minimum-Delay Peer-to-Peer Streaming. In *Ninth International Conference on Peer-to-Peer Computing*, 2009.

[5] M. Khan, G. Pandurangan, and V. Kumar. Distributed algorithms for constructing approximate minimum spanning trees in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, pages 124–139, 2008.

[6] Y. Li, D. Ren, S. Chan, and A. Begen. Low-delay mesh with peer churns for peer-to-peer streaming. In *IEEE International Conference on Multimedia and Expo, 2009. ICME 2009*, pages 1546–1547, 2009.

[7] D. Ren, Y. Li, S. Chan, et al. Fast-mesh: a low-delay high-bandwidth mesh for peer-to-peer live streaming. *IEEE Transactions on Multimedia*, 11(8), 2009.

[8] K. Vik, P. Halvorsen, and C. Griwodz. Multicast tree diameter for dynamic distributed interactive applications. In *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1597–1605. Citeseer, 2008.

[9] B. Yu, J. Li, and Y. Li. Distributed Data Aggregation Scheduling in Wireless Sensor Networks. In *IEEE INFOCOM*. IEEE, 2009.