Supporting Multi-Party Voice-Over-IP Services with Peer-to-Peer Stream Processing

Xiaohui Gu, Zhen Wen, Philip S. Yu, Zon-Yin Shae
IBM T. J. Watson Research Center
Hawthorne, NY 10532
{xiaohui, zhenwen, psyu, zshae}@ us.ibm.com

ABSTRACT

Multi-party voice-over-IP (MVoIP) services provide economical and natural group communication mechanisms for many emerging applications such as on-line gaming, distance collaboration, and tele-immersion. In this paper, we present a novel peer-to-peer (P2P) stream processing system called peerTalk to provide resource-efficient and failure-resilient MVoIP services. Different from previous work, our solution is fully distributed and self-organizing without requiring specialized servers or IP multicast support. Particularly, we decouple the stream processing in MVoIP services into two phases: (1) *aggregation phase* that mixes audio streams from active speakers into a single stream; and (2) *distribution phase* that distributes the mixed audio stream to all listeners. The decoupled model allows us to optimize and adapt the P2P stream mixing and distribution processes separately. Specifically, we can adaptively spread stream mixing workload among resource-constrained peer hosts according to current speaking activities. We have implemented a prototype of the peerTalk system and conducted experiments in real-world wide-area networks. The results show that peerTalk can achieve lower resource contention and better service quality than previous common solutions.

Categories and Subject Descriptors: C.2.4: Distributed applications

General Terms: Design, Performance, Algorithm

1. INTRODUCTION

Internet has evolved into an indispensable service delivery infrastructure instead of merely providing host connectivity. IP telephony [1, 7, 8] is one of the most promising Internet services that can greatly reduce the cost of traditional telephony services. A simple IP telephony system includes two participants, where the original voice signal is periodically sampled, encoded into a bit stream, and sent over the Internet to the receiving end. However, many emerging applications call for multi-party voice-over-IP (MVoIP) services that can include three or more participants. Such application examples include multi-player Internet gaming [6], distance collaboration systems, and on-line chatting. In Internet gaming, MVoIP services allow game players to easily communicate with each other for deploying strategies, and game spectators to cheer up players. Distance collaboration systems allow people to work as a team without costly travel expenses, where MVoIP services can provide natural communication mechanisms. Different from conventional conferencing systems that impose explicit or implicit floor controls, the emerging applications demand more flexible MVoIP services that allow any participants to “speak” at anytime. By speaking, we mean not only uttering words, but also nonverbal activities such as shouting, singing, cheering, and laughing that are common in interactive and spontaneous applications.

Traditional conferencing systems often employ IP multicast (e.g., [4]) or overlay multicast (e.g., [5, 3]), illustrated by Figure 1 (a). The system needs to distribute multiple audio streams concurrently through different multicast trees from all active speakers to all listeners. Although the multicast approach is well suited for broadcast applications that usually involve one active speaker, it becomes inefficient for interactive and spontaneous applications (e.g., Internet gaming, chatting) that often include many simultaneous speakers. Alternatively, we can employ a centralized approach that first mixes the audio streams of all active speakers into a single stream and then distributes the mixed stream to all participants, illustrated by Figure 1 (b). Stream mixing can effectively reduce network traffic by reducing the number of audio streams distributed across networks. However, the centralized approach lacks scalability and resilience that are required by the P2P applications. The most popular real-world P2P VoIP system Skype can only support conferencing sessions with at most five people[1]. Distributed audio mixing (e.g., [8, 7]) has been proposed to provide MVoIP services, illustrated by Figure 1 (c). The distributed audio mixing approach mingles the stream mixing process with the stream distribution process, which is called coupled distributed processing (CDP) in this paper. However, CDP can be sub-optimal since it fails to explore the asymmetric properties of MVoIP services such as distinct speaking/listening activities and unequal in-bound and out-bound bandwidth at each peer host.

In this paper, we propose a new P2P audio stream processing system called peerTalk to provide resource-efficient and failure-resilient MVoIP services. Compared to previous work, our solution presents three unique features. First, we decouple the audio stream processing into two phases,

MM’05, November 6–11, 2005, Singapore.

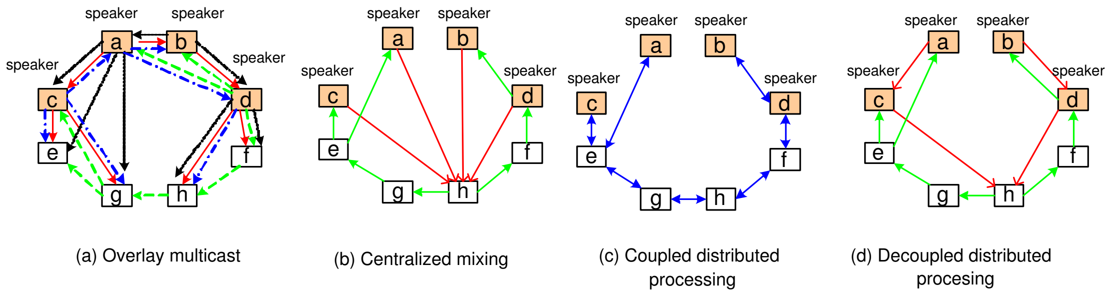


Figure 1: Different implementations of P2P MVoIP services.

illustrated by Figure 1 (d): (1) *aggregation phase* that mixes audio streams of all active speakers into a single stream via a *mixing tree*; and (2) *distribution phase* that distributes the mixed audio stream to all listeners via a *distribution tree*. The decoupled processing model allows us to optimize and adapt the stream mixing process and stream distribution process separately by fully exploring the asymmetric properties of MVoIP services. Second, the peerTalk system is *fully distributed* and *self-organizing*, which does not require any specialized servers or IP multicast support. The system provides scalable MVoIP services by aggregating resources of all peer hosts in the system. Thus, the peerTalk system can naturally scale up as more peers join the system. Third, the peerTalk system is *adaptive*, which can dynamically distribute the audio processing workload among different peer hosts. The system performs *continuous* optimization to adaptively improve the quality of MVoIP services. To achieve failure resilience, the peerTalk system adopts an overlay-based approach for failure resilience. We first connect peer hosts into an overlay mesh on top of IP network. The mixing and distribution trees are then built on top of the overlay mesh.

We have implemented a prototype of the peerTalk system and conducted extensive experiments in both wide-area networks PlanetLab [2] and simulated P2P networks. Our experiments reveal several interesting results. First, peerTalk can greatly reduce resource contentions in P2P environments compared to the overlay multicast approach [5], especially for MVoIP sessions with large group sizes and heavy workloads. Second, peerTalk achieves much lower service delay than the CDP approach by separating the stream mixing process from the distribution process. Third, peerTalk can quickly recover MVoIP service failures while maintaining low resource contention and service delay among live peers as long as the overlay mesh is connected.

The rest of this paper is organized as follows. Section 2 introduces the adaptive stream mixing algorithm. Section 3 presents the experimental results. Finally, the paper concludes in Section 4.

2. ADAPTIVE STREAM MIXING

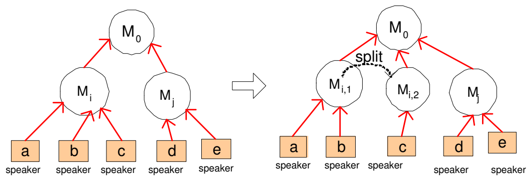
We now present a fully distributed algorithm for dynamically constructing and adapting the mixing tree used by a MVoIP service session. The basic idea of our approach is to adaptively distribute the multi-stream audio mixing workload among multiple selected peer hosts. Different from the stream distribution workload that is proportional to the number of listeners, the workload of audio mixing is decided by the number of active speakers that can dynamically change over time. Thus, our scheme continuously monitors the number of active speakers and dynamically adjust the mixing tree to adapt to the changing workload.

At the beginning of a MVoIP service session, the mixing tree contains a single mixer called the root mixer M_0. All participants are initially assigned as the children of the root mixer. The system runs an election algorithm to make all participants initially connected to the same root mixer. Since the root mixer is also the root of the distribution tree, we place the root mixer on the peer host that is the source of the best multicast tree with the minimum worst-case delay between the source and all other participates. Specifically, all peers concurrently run the DVMRP algorithm to construct multicast trees rooted at themselves. Each peer then calculates the worst-case delay of its own multicast tree and then propagates the information to all other members via the overlay mesh. All peers then select the same best peer as the root mixer whose multicast tree has the minimum worst-case delay.

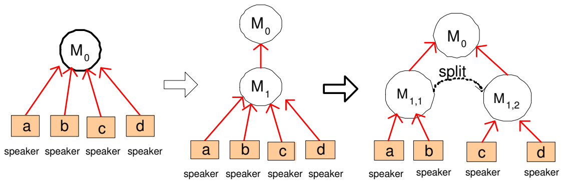
During runtime, the system adaptively grows or shrinks the mixing tree based on the audio mixing workload using a fully distributed algorithm. First, the root mixer monitors the number of active speakers among all participants. If the number of active speakers exceeds the capacity of the root mixer, the root mixer spawns new child mixers on other peer hosts to offload the audio mixing workload. The basic idea of mixing tree adaptation is that each mixer can either split itself if it is overloaded or merge with its sibling mixers if it is under-loaded. The mixer is also dynamically migrated among different peer hosts to achieve improved service quality. We now describe the distributed algorithms for mixer splitting, mixer merging, and mixer migration, respectively.

2.0.1 Mixer Splitting

Each mixer M_i in the mixing tree monitors the number of audio streams concurrently arrived at its input ports. Since peers can perform silence suppression, a leaf node on the mixing tree generates an audio stream only if the local participant produces any sound. An internal node on the mixing tree generates an output audio stream if any of its input ports receives an input stream. Suppose the mixer M_i has n input ports denoted by $I_1, I_2, ..., I_n$. We use time-serials $A_k, 1 \leq k \leq n$ to describe the data arrival pattern at the input port I_k. The time-serials A_k consist of a sequence of time-stamped number denoted by $a_k \in A_k$. At time t, we set $a_k = 1$ if there are data arriving at the input port I_k, or $a_k = 0$ if no data arrives. Hence, the



(a) Non-root mixer splitting



(b) Root mixer splitting

Figure 2: Mixer splitting operation.

total number of audio streams, denoted by N_i, concurrently arrived at the mixer M_i at time t can be calculated by $N_i(t) = \sum_{k=1}^{n} a_k$. To achieve stability, we use the exponential smoothing algorithm to update the value of N_i at periodical intervals, i.e., $N_i = \alpha \cdot N_i + (1-\alpha) \cdot N_i(t), 0 < \alpha < 1$. The length of the period can be decided based on the trade-off between stability and responsiveness.

Since peer hosts are often resource constrained, they can only process a limited number of audio streams while keeping up with the input stream rate. Let us consider the mixer M_i located on the peer host v_i that can process at most C_i streams. If the number of arriving audio streams exceeds its processing limit, i.e., $N_i > C_i$, the mixer M_i triggers the splitting process. If the overloaded mixer M_i is not the root mixer, it splits itself into two mixers $M_{i,1}$ and $M_{i,2}$, illustrated by Figure 2 (a). One of them $M_{i,1}$ remains on the host v_i and is assigned a subset of the children of M_i whose aggregate workload is within C_i. The rest of the children are assigned to the new mixer $M_{i,2}$. The peer host v_i then selects one of its neighbors v_j with the largest processing capacity to host $M_{i,2}$. If the workload of $M_{i,2}$ still exceeds the processing limit of v_j, the mixer $M_{i,2}$ continues to split itself until the workload of each new mixer is within the processing limit of its hosting peer. Note that the above process may trigger the parent of M_i to split since the number of its children is increased.

If the overloaded mixer M_i is the root mixer, i.e., $M_i = M_0$, the peer host v_i first creates a new mixer M_1 and transfers all of M_0's children to M_1, illustrated by Figure 2 (b). The new mixer M_1 then becomes the only child of M_0 and is migrated on one of the neighbors of v_i that has the largest stream processing capacity. By doing so, the height of the mixing tree is thus increased by one. Let us assume M_1 is placed on the peer host v_j. If the workload of M_1 still exceeds the capacity of v_j, M_1 performs the splitting as the previous case since M_1 is not the root mixer. All spawned new mixers become the children of the root mixer M_0.

To minimize the average workload for all input streams, we distribute the children of M_i to each new spawned mixers $M_{i,1} ..., M_{i,k}$ based on the data arrival time serials $A_1, ..., A_n$. We calculate the correlation coefficient between every two data arrival time serials A_i and A_j, which indicates the possibility of concurrent data arrivals on the input ports I_i and I_j. We then allocate the least-correlated input streams to the same mixer to minimize the average aggregate workload at each mixer.

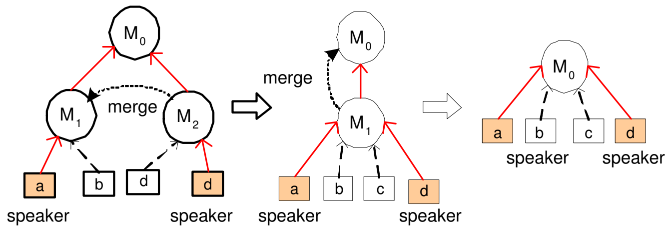


Figure 3: Mixer merging operation.

Merge M_i with M_j, M_p: parent of M_i and M_j
if $(N_i < [\frac{C_i}{2}]) \wedge (N_i + N_j \leq max(C_i, C_j))$
  if $C_i \leq C_j$
  then delete M_i
  else delete M_j
Operations when M_i is the only child of M_p
if M_p is not the root mixer
  if M_p can handle all workload
  then delete M_i
  else if M_i can handle all workload
  then delete M_p
if M_p is the root mixer $\wedge$ $(N_i + N_p \leq C_p)$
then delete M_i

Figure 4: Mixer merging algorithm.

2.0.2 Mixer Merging

We now present the mixer merging algorithm illustrated by Figure 3. The mixer merging process can effectively shrink the mixing tree to avoid excessive audio mixing overhead (delay, packet loss) by minimizing the number of mixers traversed by the audio streams. Similar to the mixer splitting process, each mixer M_i monitors the number of audio streams concurrently arrived at its input ports. If the total workload N_i is significantly less than the mixer's processing capacity C_i (e.g., $N_i < [\frac{C_i}{2}]$), the mixer seeks to merge with its succeeding sibling M_j in the mixing tree[1]. If the aggregate workload of M_i and M_j is within the processing limit of a single mixer, i.e., $N_i + N_j \leq max(C_i, C_j)$, we merge the two mixers into one mixer. If $C_i \leq C_j$, we delete M_i and connect the children of M_i to M_j. Otherwise, we delete M_j and connect the children of M_j to M_i. Note that the above process may trigger the parent of M_i and M_j to perform mixer merging since the parent's input stream number decreases. If a mixer M_i becomes the only child of its parent mixer M_p, we can merge M_i with M_p to reduce the height of the mixing tree. The situation occurs when the children of M_p merge with each other into one mixer. Figure 4 shows the psudo-code of the mixer merging algorithm.

[1]We organize the children list as a circular queue to avoid redundant merging (e.g., M_1 wants to merge with M_2 and M_2 wants to merge with M_1)

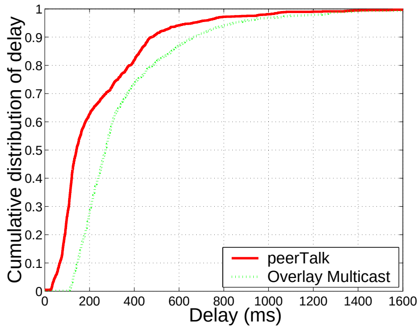


Figure 5: Cumulative distribution of delay with small number of speakers.

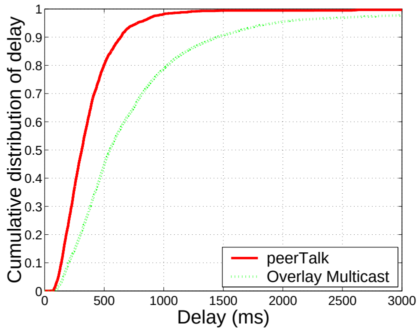


Figure 6: Cumulative distribution of delay with large number of speakers.

3. PROTOTYPE EXPERIMENTS

The peerTalk system prototype is a multi-threaded distributed software system written in Java. The software at each node includes five major modules: (1) *mixer manager* executes the mixer splitting, the mixer merging, and mixer migration algorithms; (2) *overlay topology manager* maintains the neighbor set; (3) *monitoring* module is responsible for monitoring the network/service states of neighbors; (4) *session manager* maintains membership information about each active MVoIP service session; (5) *data transmission* module is responsible for sending, receiving, and forwarding audio data. We leverage the SCRIBE software [3] to realize P2P overlay multicast.

Our experiments use 51 Planetlab hosts that spread across U.S. and Europe. At the beginning, each peer sends a probe message to all other peers via the SCRIBE multicast interface and measures average delay between itself and all other peers. All peers then exchange with each other the worst-case delay from themselves to all other peers. All peers then select the best peer that has the minimum worst-case delay as the root mixer. We adopt the standard ON/OFF model to emulate the speaking activity. Each peer generates audio data when it is in ON state and generates no data when it is in OFF state. Each peer switches from ON state to OFF state with a probability P_1 and switches from OFF state to ON state with a probability P_2. The audio encodings are all 8KHz, 8bit, Ulaw, and mono. Our first set of experiments uses $P_1 = 0.6$ and $P_2 = 0.4$. Figure 5 plots the cumulative distribution of delays between all pairs of communicating participants using the peerTalk system and the SCRIBE overlay multicast system, respectively. Our second set of experiments uses $P_1 = 0.35$ and $P_2 = 0.65$ to emulate a more active MVoIP session, illustrated by Figure 6. We observe that peerTalk achieves smaller service delays than the overlay multicast approach, especially for highly interactive MVoIP sessions. The reason is that the peerTalk selects the best multicast tree for distributing streams and the audio mixing can effectively reduce resource contentions leading to less queuing delays.

4. CONCLUSION

In this paper, we have presented peerTalk, a P2P stream processing system supporting multi-party VoIP (MVoIP) services. Different from conventional VoIP systems, peerTalk is fully distributed and self-organizing without requiring any specialized servers or IP multicast support. To the best of our knowledge, this is the first work that studied the P2P stream processing problem in the MVoIP application domain. This paper makes three major contributions. First, we propose a fully distributed, decoupled stream processing model to provide efficient MVoIP services by separating the audio mixing and distribution processes. Second, we provide audio mixing adaptation algorithms to adjust the audio mixing process according to speaking activity changes. Third, we present the stream processing component migration algorithm to continuously optimize the quality of MVoIP services in dynamic P2P environments. We have implemented a prototype of the peerTalk system that are evaluated in both real-world wide-area networks and simulated P2P networks. Our results show that peerTalk outperforms previous common solutions in terms of resource contentions and service delays, especially for MVoIP sessions with large group sizes and heavy workload conditions.

5. ACKNOWLEDGMENT

We would like to thank Dr. Fan Ye at IBM T.J. Watson research center for his helpful input to our work.

6. REFERENCES

[1] Skype Internet telephony. *http://www.skype.com/*.
[2] the PlanetLab. *http://www.planet-lab.org/*.
[3] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC, 20(8)*, Oct. 2002.
[4] S. Deering. Multicast routing in internetworks and extended lans. *Proc. of ACM SIGCOMM, Stanford, CA*, Aug. 1988.
[5] Y. h. Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. *Proc. of ACM SIGCOMM, San Diego, CA*, Aug. 2001.
[6] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-Peer Support for Massively Multiplayer Games. *Proc. of IEEE INFOCOM 2004, Hong Kong*, Mar. 2004.
[7] J. Lennox and H. Schulzrinne. A Protocol for Reliable Decentralized Conferencing. *Proc. of ACM NOSSDAV, Monterey, CA*, June 2003.
[8] M. Radenkovic and C. GreenHalgh. Multi-party Distributed Audio Service with TCP Fairness. *Proc. of ACM Multimedia 2002, Juan-les-Pins, France*, Dec. 2002.