iGridMedia: Providing Delay-Guaranteed Peer-to-Peer Live Streaming Service on Internet

Meng Zhang
Tsinghua University
Email: zhangmeng00@mails.tsinghua.edu.cn

Lifeng Sun
Tsinghua University
Email: sunlf@tsinghua.edu.cn

Shiqiang Yang
Tsinghua University
Email: yangshq@tsinghua.edu.cn

***Abstract*—Almost all existing peer-to-peer (P2P) streaming systems can only provide non-interactive service on Internet. In this paper, we focus on providing delay-guaranteed service to support delay-tolerant interactive applications (such as online auction, person interview, video sharing & commenting, etc.) using P2P streaming technology. In an interactive channel, all or part of the participants can interact with the presenter or publisher at the source. Unlike existing P2P streaming service, there is different level of delay and synchronization requirement in the interactive applications, meanwhile, the participant number in each channel is relatively small but the concurrent channel number is large. With such challenges, how much server bandwidth can be saved in P2P streaming is an interesting problem. We propose a very practical protocol iGridMedia and fully implement it. The basic tradeoff between consumed server bandwidth and the required delay is carefully studied. Simulation and real-world experiment show that our system consumes only about 4 times streaming rate with 5 sec guaranteed delay even if the peer churn rate is high. So far as we know, our work represents the state-of-the-art approach that using P2P technology to support emergent interactive live applications on Internet with delay-guaranteed requirement.**

I. Introduction

During recent three years, a large number of commercial peer-to-peer live streaming systems have emerged on Internet such as PPLive [1] to name a few and the technology has been getting mature. Almost all these systems target to provide non-interactive live broadcasting service without delay guaranteed, such as TV broadcasting, etc. And most systems employ a data-driven/mesh-based P2P streaming protocol [2], [3], [4]. This type of protocol can work fairly well with large scale users. Some service providers [1] declare that they have the record to support over 1 million concurrent users to watch an event. On the other hand, in academia, many new ideas have also been proposed to improve the performance of large scale non-interactive live P2P streaming [5], [6], [7] recently.

Nevertheless, beyond non-interactive applications, there is an urgent demand of technology support for interactive live application, such as online gaming, online auction, person interview & watch, distant learning, new product release & training, video sharing & commenting, etc. In these type of applications, a number of end users (viewers) should watch the same video at the same time, and the viewers can interact with the presenter/publisher at the source. Usually, many of these applications have different level of delay tolerance from several seconds to more than 10 seconds rather than strictly real-time, and hence it brings potential possibility to use peer-to-peer technology for saving server bandwidth. Meanwhile, although many applications can tolerate a certain amount of delay, however, the delay should be predicable and guaranteed. Actually, the aim of this paper is to find a practical way to provide efficient low-cost open live streaming service with guaranteed delay on Internet for such interactive applications.

Besides, in many interactive applications, the user number in each channel is frequently not large, usually between 5$\sim$200. However, there may be many concurrent channels at the same time. With so many small group channels, to improve the system scalability, the best way is to minimize the server bandwidth consumed with respect to the guaranteed delay constraint in every channel. Can live P2P streaming technology help much in such applications? How much server bandwidth is consumed under the constraint of delay? In this paper, we propose and implement a highly efficient protocol - iGridMedia. The basic tradeoff between the required delay and the consumed server bandwidth is studied. Both simulation and PlanetLab experiment show that the performance of the protocol is good. With 5-sec guaranteed delay, in static environment (no peer quits after joining), the server bandwidth consumed is only 1.4 times streaming rate even if the peer resource index (defined in Sec. III-C) is only 1.4 and the group size is 200; while, in very high churn rate, only about 4 times streaming rate of server bandwidth is consumed when the group size is 100. Our protocol is simple and practical. So far as we know, this work is the current state-of-the-art approach which uses P2P streaming technology to support interactive live applications with guaranteed delay constraint.

The paper is organized as follow. In Section II, we discuss the related work. Then in Section III, the detailed protocol of iGridMedia is presented. Its performance is evaluated in Section IV by both simulation and real-world experiment. We conclude this paper and give our future work in Section V.

II. Related Work

Most of the recent P2P streaming research work focuses on providing non-interactive live streaming service. CoolStreaming/DONet [3] employs a data-driven protocol to discover content availability among peers, eliminating the trees. PRIME [6] proposes mesh-based based protocol importing swarming content delivery used in file-sharing to P2P streaming systems. [8] shows that using stable peers to form a backbone in the

streaming delivery mesh can reduce the transmission delay. R^2[7] presents a protocol by randomly pushing network-coded blocks has better performance compared to traditional pull-based protocol with or without network coding. Outreach [9] seeks to maximize the utilization of available upload bandwidth on each participating peer. Chunkyspread [5] proposes an unstructured multiple tree protocol to degrade the transmission delay and to better adapt to the peer dynamics. However, very little work focuses on providing delay-guaranteed live streaming service to support interactive applications using P2P streaming technology, meanwhile, there is a great demand from the industry. We believe this is because it is very difficult to not only provide live streaming service with ensured small delay but also let it have good scalability. So far as we know, this is the first work to use P2P technology to support delay-guaranteed live streaming service.

III. iGridMedia Protocol

A. Architecture

We target to provide open delay-guaranteed live broadcasting service on Internet. To give such The service we assume the service provider has dedicated servers to support the delay-guaranteed interactive applications. As illustrated in Fig. 1, the *presenters* who want to cast their live show usually use DSL or cable to access Internet and hence have low upload bandwidth. To make full utilization of its upload bandwidth, the presenters should upload their stream to the deployed dedicated servers and the live content will be broadcast from the server to the end viewers. As shown in Fig. 1, each *viewer* maintains a *rescue connection* with the server. We assume the default transmit protocol is UDP. Once an absent streaming packet is about to pass its deadline, it will be requested through the rescue connection from the server.

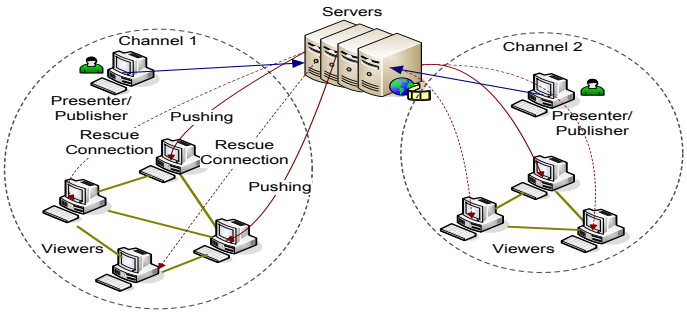

Fig. 1. Service Architecture

B. Basic Pull-Push Protocol

The basic protocol we used is pull-push protocol and the details can be found in [10]. The protocol is simple and the most important characteristic is its near optimality in upload bandwidth utilization. We briefly introduce this protocol here. For overlay construction, to join a P2P streaming session, nodes must first contact a rendezvous point (RP) which is a server maintaining a partial list of current online nodes in every channel. Then each node randomly finds some other 15 nodes as its neighbors to keep connections with so that a rich-connected unstructured overlay is built. For the streaming delivery, in the pull mechanism, the video streaming is packetized into fixed-length packets called streaming packets marked by sequence numbers. In all of our simulations and experiments, we pack 1250-byte streaming data in each streaming packet. Each node periodically sends buffer map packets to notify all its neighbors what streaming packets it has in the buffer. Each node then explicitly requests its absent packets from neighbors. Once a packet fails to be pulled, it will be requested again. To avoid duplicated requests, estimating the packet transmission timeout after requesting is important [10]. Since the transmission delay of UDP packet can be well predicted, we use UDP as default transmission protocol. And in the push mechanism, we evenly partition the stream into 16 ***sub streams***, and each sub stream is composed of the packets whose sequence numbers are congruent to the same value modulo 16. Once a packet in one sub stream is successfully pulled from a peer, the rest packets in this sub stream will be relayed directly from this peer. When a neighbor quits or packet loss occurs, the pull mechanism is started again.

C. Server Scheduling

The packet scheduling on server is different from the peers. The packets sent out from the server is either pushed or relayed proactively or be pulled by peers passively. The server pushes/relays each fresh packet received from the presenter to part of the peers in every channel. For any packet, if the server directly pushes/relays only one copy to any of the peers, we call it a **1-time-server-push**. We say that the ***server push times*** is 2 or it is **2-times-server-push**, if the server pushes/relayes two copies of each packet to any of the peers, and so on. When the packets are pushed to the peers by the server, the packets will be transmitted between the peers using pull-push protocol. Once an absent packet is about to pass the deadline (described more detailedly below), the peer will directly request this packet from the server through the rescue connection established between the peer and the server. In one word, the server always relays the most fresh packets to the peers and sends back the late packets requested from the peers and within the deadline, the packets will be requested from peers (not server) as much as possible.

To get the packet precise playback deadline, all nodes including the presenters and the viewers should perform clock synchronization with the server when they start to join any channel, using network time protocol (NTP) or just a simple "ping-pong" synchronizing message if the precision requirement is not high. We use t_{gen} to denote the generated clock time of packet i at the presenter. We let t_{rtt} represent the round-trip time between the viewer and the server and let t_e denote the end-to-end delay from the presenter to the server. If the required guaranteed delay is T, the viewer will send a packet request to the server, once the packet does not arrive until clock time $t_{gen} + T - t_e - t_{rtt}$. The server will send the packet to the viewer as soon as it receives the request.

Then we introduce how the server chooses peers to relay the fresh packets. The server select one or several peers for each

sub stream to relay. Each peer reports to the server its current outgoing traffic rate every 15 sec. We assume the streaming rate is r and each sub stream rate is $r/16$. We use $1, \cdots, n$ to denotes each peer and use $O_1, \cdots, O_n$ to represent their current outgoing rate. For sub stream 1, The server finds a peer i_1 who satisfies $i_1 = \max_i\{\frac{O_i}{r/16}\}$. Actually, we want to relay the sub stream to a peer who has the most contribution. After that, we let $O_{i_1} \leftarrow O_{i_1} - r/16$. Then we use the same method to find a peer i_2 to relay sub stream 2, and so on.

Here, we define term ***Peer Resource Index*** (PRI) [11] in every channel here. It is defined as the ratio of the total peer upload capacity $\sum_{i=1}^{n} u_i$ to the minimum bandwidth resource demand (i.e, streaming rate r times the viewer number n), that is., $PRI = \sum u_i/nr$.

And we define ***Minimum Server Bandwidth Needed*** ($MSBN$) here. If the total peer upload capacity is large enough, that is, $\sum u_i \geq nr - r$, i.e., $PRI \geq 1 - 1/n$, $MSBN$ is just the streaming rate r (the server should sent out at least one copy for each packet); while, if the total peer upload capacity is low, $\sum u_i < nr - r$, i.e., $PRI < 1 - 1/n$, the $MSBN$ is the minimum bandwidth resource demand nr minus the total peer upload capacity, that is, $MSBN = nr - \sum u_i = nr(1 - PRI)$. This is actually the amount of additional server bandwidth that should be supplied except for the total peer upload bandwidth capacity. On the whole, $MSBN = \max\{r, nr - \sum u_i\} = \max\{r, nr(1 - PRI)\}$

Further, to measure the consumed server bandwidth, we use the term ***Normalized Consumed Server Bandwidth*** ($NCSB$), which is defined as the ratio of the consumed server bandwidth to the minimum server bandwidth needed ($MSBN$). The normalized server bandwidth consumed is always above 1, and the more it is close to 1, the better the performance is.

Besides, we define ***Bandwidth Gain*** as the ratio of the bandwidth capacity demand nr to the consumed server bandwidth, that is, if client-server architecture rather than P2P streaming was employed, how many times server bandwidth would be consumed compared to using P2P streaming. The larger the bandwidth gain is, the better the performance is.

D. Adaptive Server Push

Generally speaking, more server bandwidth will be consumed if the peer bandwidth capacity is not enough or the guaranteed delay is small. The packets sent out by the server include pushed packets and pulled packets requested from the peers. So when the server outgoing rate is getting higher, it is reasonable to push more copies of a packet directly to peers rather than waiting for the requests of late packets from the peers. In iGridMedia, we use a simple way to adjust server push times. Assume the current server outgoing traffic rate is r_s and streaming rate is r, and the current server push times is γ. If $r_s/r > 2\gamma$, we let $\gamma \leftarrow 2\gamma$; while if $r_s/r < 1.2\gamma$, we let $\gamma \leftarrow \gamma/2$. This means once the server outgoing traffic rate exceeds 2 times pushed packet bit rate, we double the server push times; while if the server outgoing traffic rate goes down to 1.5 times pushed packet bit rate, we reduce the server push times to half.

IV. Performance Evaluation

A. Simulation

We implement an event-driven packet-level simulator coded in C++ to conduct a series of simulations in this section[1]. In our simulation, all streaming and control packets and node buffers are carefully simulated. For the end-to-end latency setup, we employ real-world node-to-node latency matrix (2500×2500) measured on Internet [12]. We do not consider the queuing management in the routers. The default streaming rate is set to 500kbps. The default neighbor count is 15 and the default request window size is 20 seconds. We assume that all peers are DSL nodes and assume the bandwidth bottleneck happens only at the last hop, i.e., at the end viewer. To simulate the bandwidth heterogeneity of the peers, we use three different typical DSL nodes. Their upload capacities are 1Mbps, 384kbps and 128kbps respectively. Note that the upload link is the bottleneck. In the simulation, we adjust the fraction of each type of node to obtain different peer resource index (PRI). And for each point in the figures, we average the results by repeating 10 runs with different random seeds.

In our simulation, we assume that the bottleneck is always at the last hop and the server bandwidth is large enough, hence the end viewer can always get sufficient streaming packets to play back due to the rescue connection with the server, that is, all the viewers can watch a full quality of video within the required. Therefore, in this section, we mainly study the consumed server bandwidth with respect to different guaranteed delay requirement under different user behavior and network conditions. By default, we use fixed server push times and its default value is 1 in our simulation.

We first investigate the performance in static environment, i.e., no peer quits after joining. Fig. 2 shows the normalized consumed server bandwidth (NCSB) with respect to the guaranteed delay in different user number. We set peer resource index (PRI) to 1.4. As shown, when the user number is small such as 20, the NCSB is very close to 1 even if the guaranteed delay is only 2 sec. This means that the consumed server bandwidth is nearly equal to the minimum server bandwidth needed (MSBN), i.e., 500kbps. When the user number in a channel is up to 200, the consumed server bandwidth is only 1.2 times the streaming rate 500kbps when the guaranteed delay is limited to 5 sec and the bandwidth gain is $200/1.5 = 133$ times. And we can also see that, to support 200 users with 2 second guaranteed delay, no more than 3.5 times streaming rate of server bandwidth is consumed.

Fig. 3 shows the NCSB with respect to the guaranteed delay in different peer resource index. The user number is 100. We see that when the PRI is only 1.2, that is, the total upload capacity is only 20% more than the minimum bandwidth resource demand, the consumed server bandwidth is about 2 times the minimum server bandwidth needed if the required delay is 5 sec. The more sufficient the upload capacity of the peers is, the less server bandwidth is consumed. And note

[1]The simulator is available online for free downloading at http://media.cs.tsinghua.edu.cn/∼zhangm

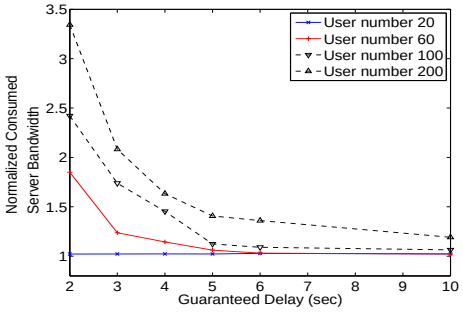

Fig. 2. NCSB with respect to the guaranteed delay and user number in static environment. PRI = 1.4. MSBN = 500kbps.

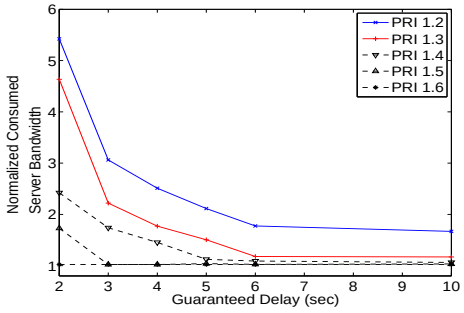

Fig. 3. NCSB with respect to the guaranteed delay and peer resource index (PRI) in static environment. User number 100. MSBN = 500kbps.

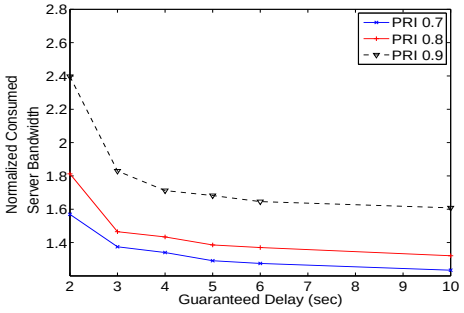

Fig. 4. NCSB with respect to the guaranteed delay in static environment when the peer upload capacity is low. User number 100.

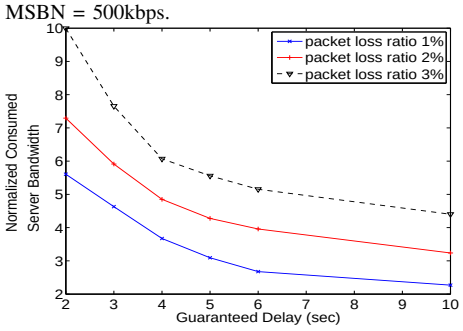

Fig. 5. NCSB with respect to the guaranteed delay and link packet loss rate in static environment. User number 100. PRI = 1.4. MSBN = 500kbps.

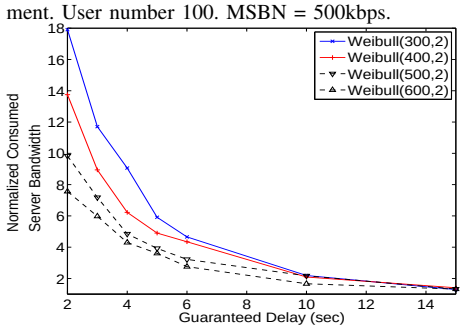

Fig. 6. NCSB with respect to the guaranteed delay in dynamic environment. PRI = 1.4. User number 100. MSBN = 500kbps.

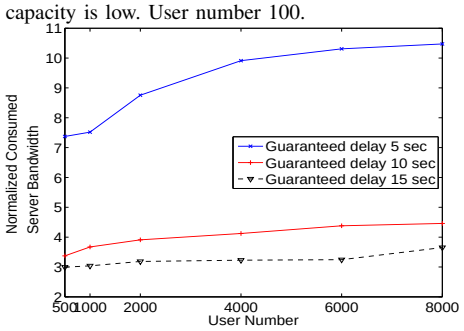

Fig. 7. NCSB with respect to the large user number and the guaranteed delay in Weibull(600,2). Packet loss rate 1%. PRI = 1.4. MSBN = 500kbps.

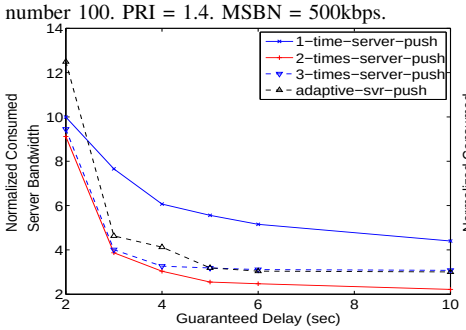

Fig. 8. NCSB with respect to the guaranteed delay and server push times in static environment. Link packet loss rate 3%. PRI = 1.4. User number 100. MSBN = 500kbps.

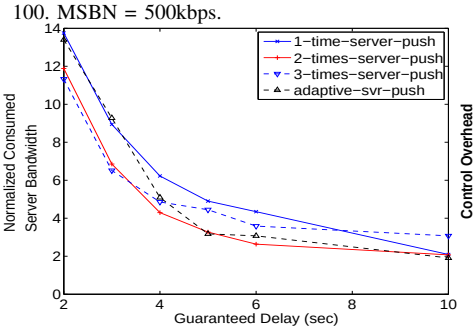

Fig. 9. NCSB with respect to the guaranteed delay and server push times in dynamic environment (Weibull(400,2)). PRI = 1.4. MSBN = 500kbps.

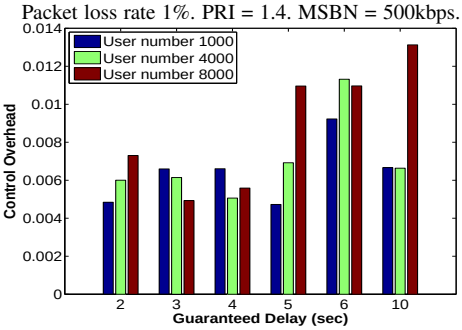

Fig. 10. Control overhead. Weibull(400,2). PRI = 1.4

that, when the PRI is above 1.6, the real consumed server bandwidth is just the MSBN.

Fig. 4 indicates the scenario when the peer upload capacity is less than the minimum bandwidth resource demand. The user number is 100. As shown, when the PRI is only 0.7, the server should supply the additional 30% bandwidth resource needed. We can easily compute the MSBN in such condition is $nr(1 - PRI) = 15Mbps$. We see that if the required delay is 10 sec, the NCSB is only 1.2. This implies that the iGridMedia protocol can ensure the all the upload capacity at peers is fully utilized and the server only provide the additional bandwidth resource needed. When the PRI is 0.9 and the guaranteed delay is 10 sec, the MSBN is about 5M and the NCSB is about 1.6. Note that the absolute server outgoing rate decreases with the increment of the PRI.

Fig. 5 gives the impact of link packet loss ratio. The default protocol in iGridMedia is UDP, so link packet loss ratio is considered here. In this figure, the user number is 100. The figure shows that the performance of iGridMedia gets worse when the link loss ratio is larger. This is because more packets will be pulled rather than be pushed. We see that when the link packet loss ratio is 3%, the bandwidth gain of 5-sec guaranteed delay is about 47 times. Later, we will show that using adaptive server push can improve the performance.

Then we investigate the performance in high peer churn rate in Fig. 6. We use Weibull(λ, k) distribution with a CDF $f(x) = 1 - e^{-(x/\lambda)^k}$ to randomly generate the lifetime of the viewers. And we assume the peer joining process is a Poisson Process with rate of 20 per sec and the maximum online user number is 100 in this figure. The median values of Weibull(300,2), Weibull(400,2), Weibull(500,2), Weibull(600,2) are respectively 104 sec, 138 sec, 173 sec and 208 sec. Fig. 6 shows the NCSB with respect to the guaranteed delay in different user lifetime distribution. Even in very high churn rate of Weibull(300,2), we get a 5-sec guaranteed delay with 6 times streaming rate of server bandwidth consumed.

We will show the improvement by adaptive server push later.

In Fig. 7, we study the scalability of iGridMedia. Although we mainly target the interactive applications with small group, we also would like to see the performance with very large user number. As shown, the user number is ranging from 500 to 8000. And we use dynamic environment with a lifetime distribution of Weibull(600,2). The link packet loss ratio is 1% and PRI is set to 1.4. It is interesting that the consumed server bandwidth do not increase much with rise of the user number, although there is strict guaranteed delay limitation. Even if the guaranteed delay is 5 sec, the server bandwidth consumed is only about 10.5 times streaming rate, i.e., 5.25Mbps.

And then, we study the impact of server push times. We vary the server push times in Fig. 8. It shows the NCSB with respect to the guaranteed delay in different server push times in static environment. The link packet loss ratio is 3%. As shown, when we use 1-time-server-push, the consumed server bandwidth is high. With 2-time-server-push, the server bandwidth consumed can be degraded to about half. This is because more direct pushed packets from server results in less requests of late packet to the server. However, when the server push times is up to 3, the server bandwidth consumed is again getting larger. The black curve in the figure shows the performance of our adaptive server push method spends relatively low server bandwidth in most cases.

Fig. 9 shows the impact of server push times in dynamic environment. The user number is 100. And the user lifetime distribution is Weibull(400,2). We can see that the adaptive server push method can usually get low server bandwidth consumed. With 5-sec delay guaranteed, the spent server bandwidth is about 4 times streaming rate and the bandwidth gain is 25.

Finally, we give the control overhead of iGridMedia. As shown in Fig. 10, The maximum user number is set to 1000, 4000, 8000 user number and PRI is set to 1.4. Dynamic environment is used and the user lifetime distribution is Weinull(400, 2). The control overhead is defined as the ratio of the control traffic amount to the total traffic. We see that the control overhead is always around or below 1%, which implies a low control traffic amount.

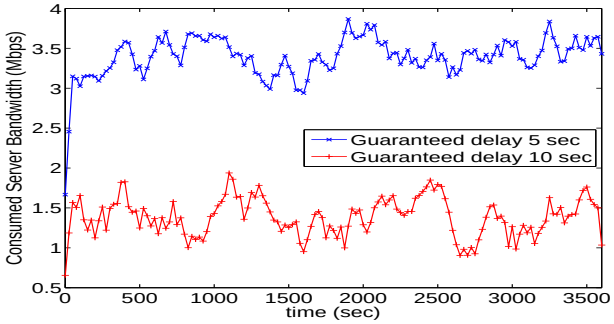

Fig. 11. PlanetLab experiment. 400 nodes used. Weibull(400,2). PRI = 1.4. MSBN = 500kbps.

B. PlanetLab Experiment

We have fully implemented iGridMedia protocol and conducted real-world experiments on PlanetLab [13]. The transmission protocol we used is UDP. We upload our viewer programs to 400 nodes on PlanetLab. We also limit the upload bandwidth of each node to 1Mbps, 384kbps and 128kbps and tune the PRI to 1.4. The upload bandwidth is limited at each node by token bucket method in our C++ code. The node lifetime distribution is Weibull(400,2) and the streaming rate is 500kbps. As shown in Fig. 11, when the delay is constrained to 5 sec, the server bandwidth consumed is around 3.4Mbps, i.e., 6.8 times streaming rate and the bandwidth gain is about 58. When the delay is limited to 10 sec, the spent server bandwidth is around 1.4 Mbps and the bandwidth gain is about 280.

V. Conclusion & Future Work

In this paper, we present our practical protocol - iGridMedia to support delay-guaranteed peer-to-peer live streaming service for delay-tolerated interactive applications. For future work, on Internet, not all the node can communicate with UDP due to firewall. The impact of TCP connection and firewall should be studied. Since TCP connection have unpredictable transmission delay because of the inherent retransmission mechanism in TCP, it will import more duplicate packets and longer delay, and hence leads to consuming higher server bandwidth. Besides, when the user number is small, the probability that the average peer upload bandwidth is less than the streaming rate is very high. In this case, we would like to use the channel with richer peer upload bandwidth to help the "lean" channel. Server can perceive the channel bandwidth richness by checking its outgoing traffic rate. Further, in some real application, when the user number in a channel is large, not all the user participate the interactive session (they only want to watch), how to provide different QoS service to different users and how to provide smooth user role transfer (such as, from non-interactive user to interactive one) are all future topics.

References

[1] "Pplive: http://www.pplive.com/," 2008. [Online]. Available: http://www.pplive.com/

[2] V. Pai and et al, "Chainsaw: Eliminating trees from overlay multicast," in *IPTPS 2005*.

[3] X. Zhang and et al, "Coolstreaming/donet: A data-driven overlay network for efficent media streaming," in *IEEE INFOCOM 2005*.

[4] N. Magharei and R. Rejaie, "Understanding mesh based peer-to-peer streaming," in *ACM NOSSDAV 2006*.

[5] V. Venkataraman and P. Francis., "Chunkyspread: Multi-tree unstructured end system multicast," in *IEEE ICNP 2006*.

[6] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in *IEEE INFOCOM 2007*.

[7] M. Wang and B. Li, "R^2: Random push with random network coding in live peer-to-peer streaming," *IEEE JSAC*, Dec 2007.

[8] F. Wang and et al, "Stable peers: Existence, importance, and application in peer-to-peer live video streaming," in *IEEE INFOCOM 2008*.

[9] T. Small and et al, "Outreach: Peer-to-peer topology construction towards minimized server bandwidth costs," *IEEE JSAC*, Jan. 2007.

[10] M. Zhang, Q. Zhang, L. Sun, and S. Yang, "Understanding the power of pull-based streaming protocol: Can we do better?" *IEEE Journal on Selected Areas in Communications*, Dec. 2007.

[11] Y. Chu and et al, "Early experience with an internet broadcast system based on overlay multicast," in *USENIX 2004*.

[12] "Meridian node to node latency matrix (2500×2500)," 2005, meridian project. [Online]. Available: http://www.cs.cornell.edu/People/egs/meridian/data.php

[13] "Planetlab: http://www.planet-lab.org/," 2008. [Online]. Available: http://www.planet-lab.org/