

Large-Scale Live Media Streaming over Peer-to-Peer Networks through Global Internet*

Meng Zhang¹, Li Zhao², Yun Tang³, Jian-Guang Luo⁴, and Shi-Qiang Yang⁵

Department of Computer Science and Technology
Tsinghua University, Beijing, China, 100084

{¹zhangmeng00, ³tangyun98, ⁴luojg03}@mails.tsinghua.edu.cn,
{²zhaoli, ⁵yangshq}@tsinghua.edu.cn

ABSTRACT

We describe the design and implementation of unstructured peer-to-peer networks for large-scale live media streaming through global Internet in this paper. In so called GridMedia system, we adopt gossip-based protocol to organize end nodes into an application layer overlay. Each node in GridMedia independently selects its neighbors and utilizes a novel and efficient push-pull streaming mechanism to fetch data from neighbors with low latency and little redundancy. The traditional pull mode in the unstructured overlay has inherent robustness to high churn rate which is common in peer-to-peer environment while the push mode could efficiently diminish the accumulated latency observed at end users. A practical system based on this architecture has been developed, and we evaluate its performance on PlanetLab [10] in various rigorous conditions. All the results demonstrate that the proposed push-pull method in GridMedia achieves good performance even with high group change rate and very low upload bandwidth limitation. Furthermore, this system was provided for CCTV (the largest TV station in China) to live broadcast the Gala Evening of Spring Festival 2005 over global Internet at the bit rate of 300 Kbps. It is evidenced that more than 500,000 users were attracted all over the world with the peak concurrent online users of 15,239 during the night.

Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed System – Distributed Applications; H.3.5 [Information-Storage and Retrieve]: Online information Service – Data Sharing

General Terms

Design, Experimentation, Performance

Keywords

Peer-to-Peer, Streaming, Push-Pull, Delivery Ratio

*Some preliminary work of this paper has been accepted as a *short paper* (4 pages) by ACM Multimedia 2005 [11]. And this work is supported by the National Natural Science Foundation of China under Grant No. 60273008 and No. 60432030

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

P2PMMS'05, November 11, 2005, Singapore.

Copyright 2005 ACM 1-59593-248-8/05/0011...\$5.00.

1. INTRODUCTION

Due to the deployment difficulties of IP multicast, application layer multicast has attracted more research interests and efforts. Meanwhile, with the explosive growth of multimedia services and applications over Internet, there has been much work in recent years on the topic of streaming video to a large population of users. As remedies to IP multicast, some research pioneers advocate application layer multicast, involving peer-to-peer based multicast. Their honorable work includes NARADA [1], HMTP [2], NICE [3], ZIGZAG [4], CoopNet [5], SplitStream [6], to name a few. However, stringent playback continuity for comfortable quality of streaming service, the scalability to self-growing population, the robustness to high churn rate of nodes and the fluctuation of network conditions are all challenges to large scale peer-to-peer media streaming applications. Since the tree-based approaches are vulnerable with dynamic group variations [8] [9], gossip-based unstructured framework has recently become popular owing to its inherent robustness, such as PRO [7], DONet [8], PRM [9]. In this paper, we also adopt the gossip-based approach for the construction of the overlay. Moreover, a novel and efficient push-pull streaming mechanism is proposed to not only keep robust to dynamic peer evolution but also efficiently reduce the latency observed at end users. We also evaluate the performance of proposed architecture and streaming method over PlanetLab in various rigorous conditions, including high group churn rate and low upload bandwidth limitations. Almost all active PlanetLab nodes across five continents were involved in experiments. To the best of our knowledge, very few literatures have utilized as many nodes as ours over PlanetLab. More interestingly, the practical system was deployed by CCTV (the largest TV station in China) to live broadcast the Gala Evening for Spring Festival 2005 through Internet. Over 500,000 users all around the world had enjoyed the exciting show with satisfactory quality feedback. And the peak simultaneous online number reached 15,239, which was achieved by only one common streaming server with encoding rate of 300 Kbps. Only 200 nodes directly connected to the streaming server. So far as we know, systems that are able to support users of a par scale at a rate of 300 Kbps by only one server seldom have been published before in both academic and industrial area.

The remainder of this paper is organized as follows. Section 2 presents a brief overview of GridMedia, comprising two main components – the overlay manager and the streaming scheduler. Section 3 and 4 focus on the details of the streaming scheduler: Section 3 explains the weakness of traditional pull method,

followed by the proposed push-pull mechanism in Section 4. Section 5 provides the performance evaluation over PlanetLab in various conditions. Then, a preliminary statistical result over Internet is discussed in Section 6. Finally, we conclude this paper in Section 7 and discuss some future work.

2. GRIDMEDIA OVERVIEW

We utilize an unstructured overlay to accommodate high churn rate in peer to peer live media streaming application. GridMedia comprises two main components – the overlay manager and the streaming scheduler. Here we only give a brief introduction to the protocol of overlay manager component in Subsection 2.1. And subsection 2.2 provides an overview of streaming scheduler component which will be explained in Section 3 and 4.

2.1 Overlay Manager

In short, the basic task of the overlay manager component on each node is to take charge of finding appropriate neighbors for each node by gossip method so that the application layer network can be successfully built up. A well-known *Rendezvous Point* (RP) is deployed to assist the construction of the overlay. During the startup, a participating node firstly contacts the RP to get a list of the nodes already in the overlay (called candidates list), which could be regarded as the login process. The newly participants will select several nodes from the candidates list as its initial neighbors. It first measures the Round-Trip Time (RTT) to each candidate and then chooses some nodes with the minimum RTT as one part of its initial neighbors. To avoid overlay division, the other part of initial neighbors will be selected randomly.

In the configuration of this overlay, the neighborhood of each node will be maintained in traditional distributed way after login process. And all the nodes will self-organize into an unstructured mesh. Each node has a member table initially obtained from RP as said, in which each item represents an active node that is currently in the overlay. The information of member tables is encapsulated into a UDP packet and exchanged among neighbors periodically. To diminish the control overhead of the gossip protocol, not all the items in the member table are included in the exchanged packet so that the swapped member table has certain constraint. Each node updates its member table in accordance with the member table sent by its neighbors. Note that each item in the table has a field called *life-time*. It denotes the elapsed time since the latest message was received from this member. When the life-time of a node exceeds the predefined threshold, it will be removed from member table. Once a node quits, it will broadcast a “quit message” to all its neighbors. Further, this message will be flooded within limited hop counts. The nodes who receive this message will delete the corresponding node from its member table. Besides, each node delivers an ‘alive’ message to all its neighbors periodically to declare its existence. Thus, the failure of any neighbors can be detected, that is, if a node hasn’t received any message from a neighbor for a while, this neighbor then will be erased from member table. Figure 1 illustrates an instance of the unstructured mesh in GridMedia.

Aiming to calculate the life time of each node in member table and deal with the synchronization issue among nodes more lightly, the local clock of the newly joined node should be synchronized with RP in login process. That is, the participating node sends a UDP packet containing its local time. Then RP returns the difference between that clock time and the clock on RP immediately. If those UDP packets are lost, this process will be

repeated. Although the synchronization method is not extremely precise, it is clear that the error will not exceed the RTT value between the node and RP, which is accurate enough for this system.

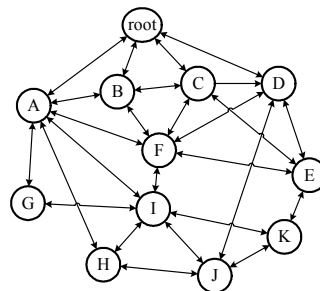


Figure 1. The structure of GridMedia

If there are some neighbors of one node quitting or failing, the node will retrieve alternative neighbors in its member table. The selection of substitutes is similar to the selection procedure of initial neighbors. Moreover, every node will refine its neighbors iteratively. Each neighbor will obtain an evaluation index in one term. The evaluation index of a neighbor here means the sum of packets received from this neighbor and the packets sent to this neighbor, i.e., the total bidirectional traffic between the node and its neighbor in one bout. If the minimum evaluation index of the neighbors is less than a threshold, the neighborhood of the poorest will be discarded, and at the meantime, a new neighbor will be probed from the member table.

2.2 Streaming Scheduler

Streaming scheduler component of each node is responsible for exchanging packets with all its neighbors. It fetches absent packets from its neighbors and in turn tries its best to deliver packets requested by the neighbors. GridMedia employs two schemes in packets streaming, namely the pull mode and the push mode.

The pull mode in this paper is similar to the data-driven approach in DONet [8]. However, here we deploy one UDP packet as a unit of transmission instead of a segment containing one second media content. RTP protocol [13] is adopted. The sequence number field in RTP packet can be used for buffer management, while the time stamp field is used to synchronize among nodes since the packet rate is variable in general. In DONet [8], buffer maps are swapped between neighbors continuously. Similarly, a buffer map in GridMedia has three fields: the maximum sequence number of buffer map s_{BM} , the length of buffer map l_{BM} , and a bit vector representing the availability of all packets with the sequence number from $s_{BM} - l_{BM} + 1$ to s_{BM} . Every node periodically sends request to its neighbor to fetch a bunch of absent packets. When a node receives a request from one of its neighbors, it will try its best to deliver the requested packets to this neighbor.

In GridMedia, the transmission protocol between neighbors can be UDP, TCP or TFRC [14]. Because of the real-time characteristics in live media streaming, UDP is the default protocol in the system. Nevertheless, UDP has no traffic control mechanism. Since the maximum downloading bandwidth of many hosts over Internet are not sufficient enough (such as the hosts accessing Internet by DSL or cable), any bursty arrival of packets to those hosts possibly lead to packet loss. Accordingly, it is significant to keep the downloading rate smooth. Motivated by

this, in our system a traffic shaper is used to smooth the traffic between any two neighbors. Every node in GridMedia assigns a traffic shaper for each neighbor. Once a request is received from one neighbor, dozens of requested packets will be delivered at an even packet rate rather than all packets sent at once. If let τ denote the interval between the request packets, and n denote the requested packets number from one neighbor, then the packet rate to this neighbor is equal to n/τ .

The details of packets scheduling in streaming scheduler component will be introduced in Section 3 and Section 4: the traditional pull mode and its weakness are presented in Section 3, while the proposed push-pull mechanism in GridMedia is interpreted in Section 4.

3. PULL METHOD AND ITS WEAKNESS

DONet [8] advocates a data-driven streaming approach upon the multi-partner overlay. Each node in DONet periodically exchanges buffer map (BM) of media segments with partners, and then retrieves absent segments from partners which report to possess the segments. This traditional pull method will introduce tremendous latency from root to each node, i.e., the huge delay between the sampling time at the server and the playback time at users. We define this delay as *absolute delay*, and the playback time as *absolute playback deadline*. Note that the absolute delay defined here is totally different from the deadline used in [8] which refers to the elapsed time after receiving the first packet at each node. A number of applications are delay-sensitive, such as interactive IP TV, distance education, etc. Streaming multicast using data-driven approach may not meet these demands very well. To exactly evaluate the quality performance, we term *delivery ratio* to represent the number of packets that arrive at each node before or right on absolute playback deadline to the total number of packets. Apparently, a greater (at least no smaller) delivery ratio can be reached at a node with the absolute delay increasing. Another metric *α -playback-time* should also be defined here to denote the minimum absolute delay at which the delivery ratio is larger than α , where $0 \leq \alpha \leq 1$. In this section, we investigate the delivery ratio as a function of absolute delay when using a pure pull method.

3.1 Simple Analysis of the Pull Method

At first, we review the detailed process of the pull mode to get insight of striking latency issue.

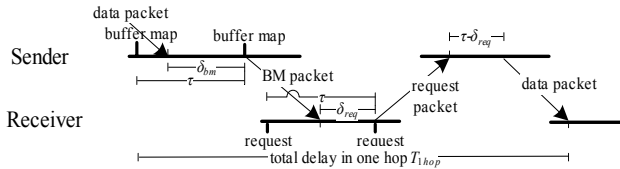


Figure 2. Delay of a packet transmitted in one hop using pure pull method

In the pull mode, when a packet transmits from one node to another, the following three steps should be accomplished as depicted in Figure 2. First, the sender informs the receiver that the packet has already existed in its buffer. Then if the receiver needs this packet, request for the packet will be sent from the receiver. At last, the sender will deliver all the requested packets to the receiver at a smooth rate. Intuitively, at least three end-to-end delays are involved in these steps. Furthermore, for the efficiency of information exchange between nodes, buffer map and requests

will only be sent in certain time interval so that dozens of packets can be mapped into one single packet. As a result, the delivery of most packets will have extra delays in one hop. More formally, we denote the interval between two buffer map packets and two request packets is τ . In the first step, the arrival of packets will not be notified to the receiver until the next buffer map packet is generated. We use δ_{bm} to denote this waiting time. Similarly, in the second step, the notified packets will not be requested as immediately as the buffer map packet is received. Let δ_{req} denote this waiting time. In the third step, packets also have to wait due to the smooth control. Since the packet with the largest waiting time in the second step will be sent first and the transmission of the requested packets should be finished in one cycle if the bandwidth is sufficient, the waiting time is $\tau - \delta_{req}$. Apparently, δ_{req} and δ_{bm} are independent, and their average values are both $\tau/2$ if the packet rate is invariable. Thus, the total average latency for a packet transmitted in one hop T_{1hop} can be approximately computed as $\delta_{bm} + \delta_{req} + (\tau - \delta_{req}) + 3\overline{\delta_{EED}} = 3\tau/2 + 3\overline{\delta_{EED}}$, where $\overline{\delta_{EED}}$ is the average end-to-end delay between nodes.

In the pull mode, every node maintains a packets owner table. Each item in this table records the corresponding neighbors of a packet in the current sliding-window. The table will be updated once a buffer map is received from a neighbor. If the buffer map contains packets which have larger time stamps than the largest one in local window, the sliding-window will move forward. In such case, the absent packets with larger sequence number will be preemptively contained in the request packet. This aims to reduce the transmission delay. As a result, each packet will be first fetched from the neighbor who possesses it earliest. When the streaming transmission comes into steady, it can be imagined that the transport paths through which the streaming is delivered tends to form a shortest sub-tree of the random-built mesh. We know that if the degree of each node is limited to n , a tree whose internal nodes have $n-1$ children except that the root node has n children is the shortest one. Hence, we can employ this tree structure to estimate the upper bound of the average absolute delay in the pull mode.

Following the primary analysis above, we can figure out the approximate average delivery ratio \bar{r} at any specific absolute delay. Assume that the group size is N and each node has at most n neighbors, while the media stream transmits through a shortest tree described previously. Let T_{1hop} denote the average delay of each packet transmitted in one hop. Thus, a time of iT_{1hop} ($i \geq 0$) after a packet is sent out from the root node, the nodes whose depth are less than or equal to i will receive it if no packets is lost. We denote the number of these nodes as N' . Therefore, at this absolute delay of iT_{1hop} , the average delivery ratio is the ratio of N' to the group size N . The result is written formally below:

$$\bar{r}(iT_{1hop}) = \begin{cases} \sum_{k=0}^i [n^k/N] & i = 0, 1 \\ (1+n)/N + \sum_{k=1}^{i-1} [n(n-1)^k/N] & 2 \leq i \leq \lceil \log_{n-1}(N/n) \rceil \\ 1 & i > \lceil \log_{n-1}(N/n) \rceil \end{cases}$$

Obviously, the average delivery ratio at other absolute delay can be approximately calculated by linear interpolation.

3.2 Analytical Experiment of the Pull Method

An analytical experiment has also been setup over PlanetLab for the performance of the pull method. In this set of experiments, only the result in static environment (that is, all the nodes are continuously online during the experiment) is presented in order to highlight the comparison with the above numerical result. The total nodes number or the group size is 310 ($N=310$), which is almost the largest scale of nodes available on PlanetLab (details of our experiments on PlanetLab can be found in Section 5). Each node has 5 neighbors at most ($n=5$). The period of exchanging buffer map packet and request packet is 1 second ($\tau=1sec$). The average packet rate is 30 packets/sec and the average packet size is 1300 bytes. Thus, the bit rate is about 310 Kbps. Simple RTT measurement shows that the average RTT between nodes on PlanetLab is approximately 120ms, as a result $\overline{\delta_{EED}} \approx 60ms$. In order to obtain the average delivery ratio at various absolute delays, each node will synchronize its local clock with RP including the root node when login. A log server collects all the report packets that contain the delivery ratio information.

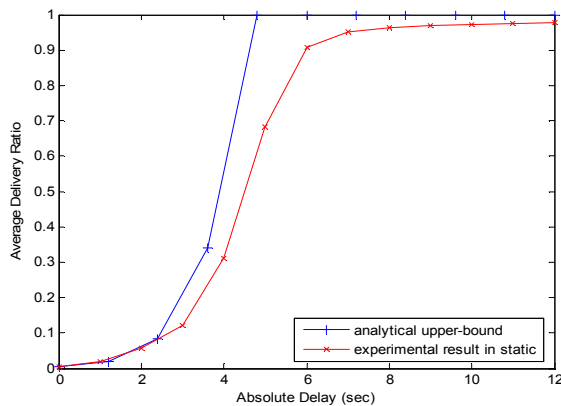


Figure 3. Delivery ratio as a function of absolute delay by analysis and experiment, group size 310

Figure 3 illustrates the analytical upper-bound and experimental result of the average delivery ratio as a function of absolute delay in static environment. Observe that both analytical and experimental results reveal that pure pull method will cause a striking latency between the sampling time and the playback time. Observe that the α -playback-time ($\alpha=0.97$) of the experimental result is about 10 seconds.

4. PROPOSED PUSH-PULL MECHANISM

Although the traditional pull or data-driven method works well with high churn rate and dynamic network conditions, it can't meet the demands of delay-sensitive applications because of the striking latency accumulated hop by hop. Additionally, incidental buffer usage at each node is needed to store the exchanging data. To this end, we propose an efficient push-pull streaming mechanism to greatly decrease the latency and inherit simplicity and robustness of traditional pull method. Concretely, the proposed push-pull mechanism can obtain the same delivery ratio with much smaller absolute delay than the pull method.

4.1 Packets Scheduling

In push-pull streaming mechanism, each node uses the pull method as a startup, and after that each node will relay a packet to

its neighbors as soon as the packet arrives without any explicit request from the neighbors. More detailedly, we classify the streaming packets into pulling packets and pushing packets. Just as the name implies, a pulling packet of a node is delivered by a neighbor only when the packet is requested, while a pushing packet is relayed by a neighbor as soon as received. Each node works in pull mode in the first time interval after the participation. Then, based on the traffic from each neighbor, the node will subscribe the pushing packets from its neighbors accordingly at the end of each time interval. A roulette wheel selection scheme is employed to allocate pushing packets in the next time interval to each neighbor. The selection probability of each neighbor is equal to the percentage of traffic from that neighbor in the previous time interval.

Meanwhile, the lost packets caused by the unreliability of the network link or the neighbors failure will be pulled as well from the neighbors, where the roulette wheel selection scheme is also used to select the suppliers of each packet from neighbors. Thus, most of the packets received will be pushing packets from the second time interval. With the purpose of easy control, the start of time interval on each node should be synchronized. Therefore in our system each node synchronizes with RP during the first login.

4.2 Pushing Packets Map and Tracker

Since each neighbor of a node should stream different packets, we use a *Pushing Packets Map* (PPMAP) to represent all the pushing packets in the next time interval. First, the stream is partitioned into P parts (numbered by $0 \dots P-1$), and each packet is hashed into only one part. As mentioned above that an RTP packet has a field of sequence number, a hashing function (mod P function) maps each packet to a part of the stream. In PPMAP, every bit represents one part of streaming. And all the packets which can be hashed into any part of the streaming contained in the PPMAP should be pushed. For each neighbor of a node, there are two relevant PPMAPs – incoming PPMAP and outgoing PPMAP, where the incoming PPMAP represents the pushing packets from this neighbor to local node while the outgoing PPMAP represents packets from local node to this neighbor.

In peer-to-peer environment, the packets received by a node can not be expected to arrive in order. Hence a problem appears that how a node decides whether a packet is lost in the transmission or it will be pushed by one of the neighbors later. It can be imagined that if a node does not make a correct decision, some packets will be either missing or received more than once. With the aim of avoiding such a packet loss or redundancy, we must resort to an effective solution. In our system, each sender should record the packet with the maximum sequence number up-to-date pushed to each neighbor. When the sequence number of a just-received packet is far behind the largest sequence number up-to-date by a threshold, it will not be relayed. We call this threshold the *maximum lag gap* at sender, denoted by l_{sender} . Likewise, each receiver uses a *Pushing Packet Tracker* to record the packet with the maximum sequence number up-to-date pushed from each neighbor. When the Pushing Packet Tracker detects a packet whose sequence number falls behind the maximum one up-to-date by a threshold is still not in buffer, it will request for this packet from a neighbor in the next request cycle, i.e., the packet will be pulled from that neighbor. This threshold is the maximum lag gap at receiver, denoted by $l_{receiver}$. Moreover, when the Pushing Packet Tracker detects that no packets has been pushed from a neighbor for a timeout, all the packets which should have been

pushed by this neighbor will be pulled from other neighbors in the current time interval, against the failure of neighbors. Apparently, If $l_{receiver} \geq l_{sender}$, there will be no redundant packets pulled.

5. PERFORMANCE EVALUATION OVER PLANETLAB

5.1 Experiment System Setup over PlanetLab



Figure 4. Nodes over PlanetLab platform

PlanetLab is an open platform for experimenting, developing, deploying, and accessing planetary-scale services. We conducted extensive experiments on this platform. PlanetLab currently has more than 590 nodes over about 280 sites all over the world. Figure 4 shows all the nodes across five continents. All the nodes on PlanetLab can be utilized in the experiments, however usually only around 350 nodes are available online at the same time. Consequently, the number of nodes in our experiments will not exceed 350, which is almost the largest scale of nodes that are active on PlanetLab simultaneously.

Table 1. Dedicated nodes in our experiments on PlanetLab

Node	URL
Root Node	planetlab1.cs.cornell.edu
RP	planetlab-1.cs.princeton.edu
Log Collecting Server	planetlab6.millennium.berkeley.edu
Control Node	planetlab1.csail.mit.edu

Several specific nodes in PlanetLab are considered as the dedicated nodes, as shown in Table 1. A packet generator runs on the root to simulate the media streaming and transmits the packets

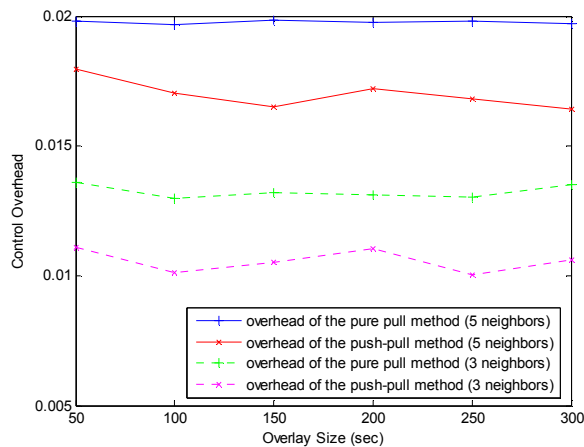


Figure 5. Control overhead of the gossip protocol in GridMedia

to the local peer directly. As mentioned, RP helps new nodes to participate the overlay. Log collecting server is deployed to gather log report packets sent by all the peers, such as control overhead, neighbor information, delivery ratio, traffic distribution, etc. Additionally, with the purpose of controlling all the nodes efficiently and delivering the executive program quickly, we manually clustered them into 12 sub-groups in terms of node location, and each group was equipped with a leader. The control node is responsible for commanding all the leaders, thus all the active nodes on PlanetLab could be controlled to participate or depart the overlay freely.

The delivery ratio at different absolute delays on each node is the most significant data to collect and analyze. Hence each node will synchronize with RP including root node, that is, all the nodes use the clock of RP. We call this clock absolute time. When the packet generator running on root node sends out a packet, it will encapsulate the absolute time in the packet header. Each node can use the absolute time when the packet is received and the absolute time encapsulated in the packet header to easily compute the absolute delay of this packet. To limit the uploading bandwidth of each node in some experiments, we use the token bucket algorithm to constrain the outgoing flow on each node.

Furthermore, we also measure the *link stress* of proposed method over PlanetLab. The stress of a physical link here is similar with [1] which represents the number of identical copies of a packet carried by a physical link. However, the accurate number of identical copies of packets is very difficult to obtain in a practical system. Instead, we refer to the ratio of the maximum packet rate on a physical link to the average streaming packet rate as the *link stress* on that physical link. In order to obtain this metric, each node reports the traffic from every neighbor to the log server periodically. Meanwhile, all the nodes utilize a tool called ‘tracepath’ on every host over PlanetLab to get the knowledge of the under-layer routers between neighbors. Likewise, the trace-path results will be reported to the log server. With these parameters, we can compute the stress of any physical link at any time.

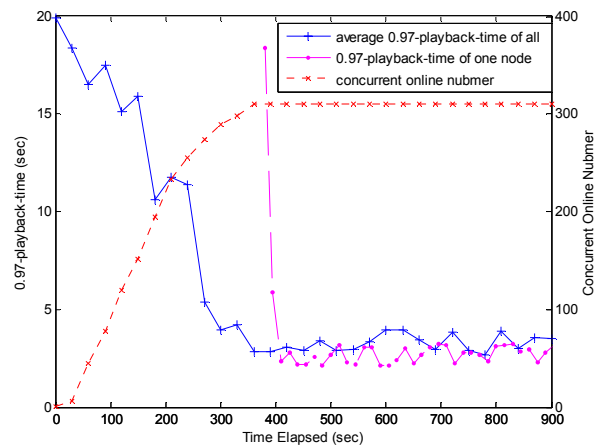


Figure 6. The convergence of the α -playback-time where $\alpha = 0.97$ in the push-pull method

5.2 Control Overhead of the Gossip Protocol in GridMedia

For the ease of deployment, the control overhead of the gossip protocol of GridMedia should be well examined. Here define the average ratio of the control traffic to the total traffic of each node as control overhead. The control messages include the messages to probe and keep neighbors, the exchanged member table packets, buffer map packets, the request packets, and PPMAP packets in push-pull method, where the buffer map packets swapped between neighbors contribute most of the control overhead. As each node only interacts with its neighbors after login and every control message has a scope constraint, the average control overhead of each node will not grow up with the overlay size. The main parameters are shown in Table 2 except that the neighbors in the experiments of this subsection are 3 and 5 so as to have a comparison.

As illustrated in Figure 5, the two solid lines show the control overhead with different overlay sizes when each node has 5 neighbors while the two dotted lines represent the control overhead when each node has 3 neighbors. We can discover from Figure 5 that the overhead has no relationship with the overlay size. In fact, the key factor impacting the control overhead is the

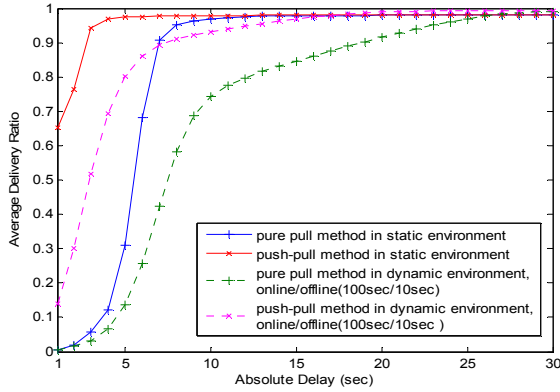


Figure 7. Comparison between pull and push-pull method in both static and dynamic environment, and the average online time and offline time in dynamic environment are 100 sec and 10 sec respectively.

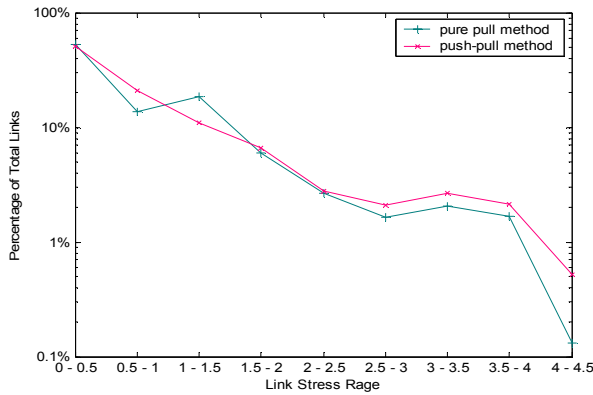


Figure 8. Link stress in static environment without upload bandwidth limitation

number of neighbors. The more neighbors each node has, the more control messages should be exchanged between them. Additionally, we note that the control overhead of pull method is always a little greater than the push-pull method. This is because in pure pull method each node will deliver a request packet per second. Yet, in push-pull method each node mainly works under push mode, and the request packet per second is replaced by the pushing packets map every 10 seconds. Hence the overhead in push-pull method becomes smaller.

5.3 Performance Evaluation without Upload Bandwidth Limitation

In this subsection, we depict the experiment results without upload bandwidth limitation and the parameters of the experiments in this subsection are summarized in Table 2.

We first show the convergence of the delivery ratio when the nodes join in the multicast group. Figure 6 illustrates the delivery ratio as a function of elapsed time when the 310 nodes take part in the group one by one in an initialization period. After that, they will stay for the life time (typically an hour in our experiment). We call it a static environment. The dotted line in Figure 6 depicts the concurrent online number at different time. The solid line

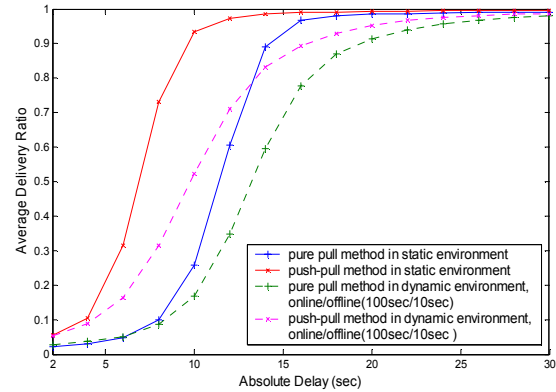


Figure 9. Comparison between pull and push-pull method in both static and dynamic environment with upload bandwidth limitation to 500 Kbps for each node

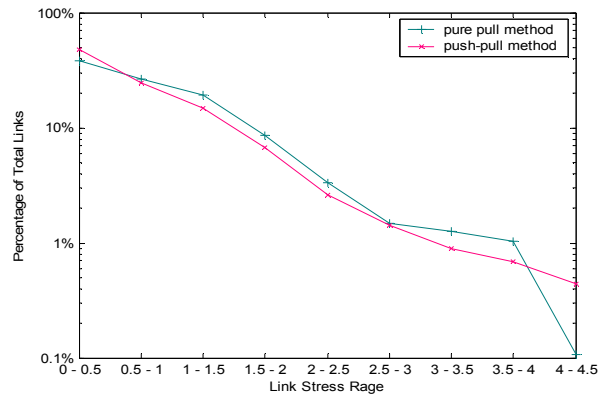


Figure 10. Link stress in static environment with upload bandwidth limitation to 500 Kbps for each node

represents the average α -playback-time ($\alpha=0.97$) of all nodes with the elapsed time. We can observe that the average α -playback-time goes down to about 2 to 3 seconds almost as immediately as all the nodes finish participating. Moreover, the dashed line in Figure 6 depicts the α -playback-time of the node joining the group at last that is considered to be far away from the root node. The α -playback-time ($\alpha=0.97$) of this node converges to 2~3 seconds within only about 20 seconds.

Table 2. Parameters in experiments without upload bandwidth limitation

Parameter	Value or Range
Group size	about 300 ~ 340
Number of neighbors for each node	5
Streaming bit rate	310 Kbps
Packet rate	30 packets/sec
Buffer map interval	1 second
Request interval	1 second
Subscribing pushing packets interval in push-pull method	10 seconds
Average online time in dynamic environment	100 seconds
Average offline time in dynamic environment	10 seconds

Figure 7 demonstrates the comparison between the traditional pull method and the push-pull method in static environment and in dynamic environment. In dynamic environment, 335 nodes on PlanetLab are utilized and each of them will continuously join and depart the group. The online and offline duration of each node are exponentially distributed with the average value of 100 and 10 seconds respectively, which is indeed a very high group change rate in practical system. The online number ranges from 300 to 320. We note that the push-pull method can reach an equivalent α -playback-time at a much smaller absolute delay than the pure pull method as shown in Figure 7. The two solid lines in Figure 7 show that the α -playback-times ($\alpha=0.97$) of the pure pull method and the push-pull method are 11 and 4 seconds separately in static environment while the two dotted lines illustrate that the α -playback-times ($\alpha=0.95$) of the two methods in dynamic environment are 22 and 13 seconds separately.

Figure 8 shows the percentage of links with different link stress ranges. Nearly 50% links has a link stress less than 0.5, and the percentage of links with link stress greater than 4 doesn't exceed

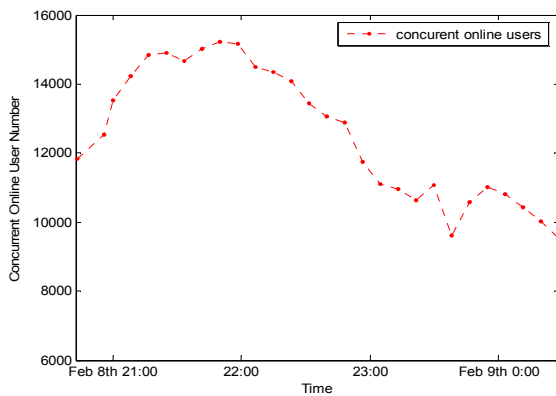


Figure 11. Number of users over time in Feb 8th.

1%. Attention should be paid that the trace path result between two nodes on the Internet is scarcely symmetric. Accordingly, the incoming traffic and the outgoing traffic of a node usually belong to two different access link of this node in our statistic.

5.4 Performance Evaluation with Upload Bandwidth Limitation

The parameters of the experiments in this subsection are listed in Table 3. Note that to emulate the DSL connections we limit upload bandwidth of all the nodes to 500 Kbps - a strictly low uploading bandwidth that is less than most DSL-access uploading bandwidth.

Figure 9 illustrates the comparison between the pull method and the push-pull method with 500 kbps uploading bandwidth limitation on each node in both static and dynamic environment. The online and offline duration of each node in dynamic environment are also exponentially distributed with an average of 100 and 10 seconds respectively. And the online number varies from 290 to 310. The two solid lines indicate that the α -playback-times ($\alpha=0.97$) of the pure pull method and the push-pull method are about 18 and 13 seconds separately in static environment while the two dotted lines show that the α -playback-times ($\alpha=0.95$) of the two methods in dynamic environment are about 24 and 20 seconds separately.

Table 3. Parameters in experiments with upload bandwidth limitation

Parameter	Value or Range
Group size	about 290 ~ 320
Upload bandwidth limitation at each node	500 Kbps
Number of neighbors for each node	5
Streaming bit rate	310 Kbps
Packet rate	30 packets/sec
Buffer map interval	1 second
Request interval	1 second
Subscribing pushing packets interval in push-pull method	10 seconds

Figure 10 presents the link stress distribution in static environment with upload bandwidth limitation to 500 Kbps for each node. Links with link stress greater than 4 occupies no more than 1%. More details about experiments can be found at my home page [12].

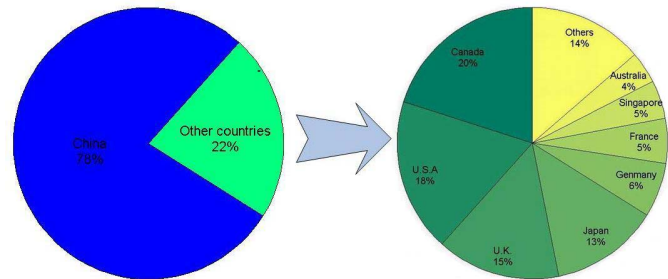


Figure 12. User location distribution

6. SERVICE OVER THE INTERNET

GridMedia system was adopted by CCTV to broadcast the Gala Evening for Spring Festival through global Internet and attracted more than **500,000** users all over the world on that night with the maximum concurrent users of **15,239**. Figure 11 gives the statistical result of online user number on the night of 8th Feb. 2005. The video encoding rate in the broadcast was 300 Kbps. Users from China, North America, Europe and other areas had enjoyed it. The peak online number 15,239 was an incredible result just supported by one single common server. Systems that are capable to sustain users of a par scale at a bit rate of 300 Kbps by only one server have seldom been published before as far as we know. The log on that night indicates that **60.8%** users were behind NAT and at least **16%** of our users accessed the Internet through ADSL.

We obtain the user location distribution on the night of Feb 8th in Figure 12 according to the IP addresses of the users' hosts. Most of them accessed from China, however, 22% of the users came from other countries, such as countries in North America, Europe and East Asia. In addition, some other demographic results of the users will be published soon.

7. CONCLUSION AND FUTURE WORK

In this paper, we provide a simple analysis of traditional pull method in peer-to-peer streaming multicast and reveal the tremendous latency observed at the user node when the pure pull method is employed. Then we propose an unstructured protocol with a novel and efficient push-pull mechanism to greatly diminish the latency and inherit most good features (such as simplicity and robustness) in pull method. After that the performance of our approach is evaluated over PlanetLab. Finally, we present a preliminary statistical result of our practical system over Internet. Our system demonstrates that peer-to-peer technology is really an efficient way for multimedia content delivery.

Supporting the group in which most nodes have limited uploading bandwidth and the optimal selection of the maximum lag gap in push-pull mechanism should be well studied. Besides, we need to evaluate some other important performance on PlanetLab. The experience of our service over Internet is also very interesting to further interpret.

8. REFERENCES

[1] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS 2000*, June 2000.

- [2] Beichuan Zhang, Sugih Jamin, and Lixia Zhang. Host multicast: a framework for delivering multicast to end users. In *Proceeding of IEEE INFOCOM 2002*, June 2002.
- [3] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM 2002*, August 2002.
- [4] Duc A. Tran, Kien A. Hua and Tai Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE INFOCOM 2003*, April 2003.
- [5] Venkata N. Padmanabhan, Helen. J. Wang, and Philip A. Chou. Distributing streaming media content using cooperative networking. In *Proceedings of NOSSDAV 2002*, May 2002.
- [6] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, et al. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of ACM SOSP 2003*, October 2003.
- [7] Reza Rejaie, Shad Stafford. A framework for architecting peer-to-peer receiver-driven overlays. In *Proceedings of NOSSDAV 2004*, June 2004.
- [8] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In *Proceedings of IEEE INFOCOM*, March 2005.
- [9] Suman Banerjee, Seungjoon Lee, Bobby Bhattacharjee, and Aravind Srinivasan. Resilient multicast using overlays. In *Proceedings of ACM SIGMETRICS 2004*, June 2004.
- [10] PlanetLab website: <http://www.planet-lab.org/>
- [11] Meng Zhang, Jian-Guang Luo, Li Zhao, and Shi-Qiang Yang. A Peer-to-Peer Network for Live Media Streaming - Using a Push-Pull Approach. In *Proceedings of ACM Multimedia 2005*, to appear, November 2005.
- [12] My homepage: <http://media.cs.tsinghua.edu.cn/~zhangm>
- [13] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson. RTP: a transport protocol for real-time applications, RFC 1889, January 1996
- [14] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448, January 2003.