Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better?

Meng ZHANG, *Student Member, IEEE,* Qian ZHANG, *Senior Member, IEEE,* Lifeng SUN, *Member, IEEE,* and Shiqiang YANG, *Member, IEEE*

***Abstract*—Most of the real deployed peer-to-peer streaming systems adopt pull-based streaming protocol. In this paper, we demonstrate that, besides simplicity and robustness, with proper parameter settings, when the server bandwidth is above several times of the raw streaming rate, which is reasonable for practical live streaming system, simple pull-based P2P streaming protocol is nearly optimal in terms of peer upload capacity utilization and system throughput even without intelligent scheduling and bandwidth measurement. We also indicate that whether this near optimality can be achieved depends on the parameters in pull-based protocol, server bandwidth and group size. Then we present our mathematical analysis to gain deeper insight in this characteristic of pull-based streaming protocol. On the other hand, the optimality of pull-based protocol comes from a cost - tradeoff between control overhead and delay, that is, the protocol has either large control overhead or large delay. To break the tradeoff, we propose a pull-push hybrid protocol. The basic idea is to consider pull-based protocol as a highly efficient bandwidth-aware multicast routing protocol and push down packets along the trees formed by pull-based protocol. Both simulation and real-world experiment show that this protocol is not only even more effective in throughput than pull-based protocol but also has far lower delay and much smaller overhead. And to achieve near optimality in peer capacity utilization without churn, the server bandwidth needed can be further relaxed. Furthermore, the proposed protocol is fully implemented in our deployed GridMedia system and has the record to support over 220,000 users simultaneously online.**

***Index Terms*—p2p streaming, pull-based, pull-push hybrid, capacity utilization, throughput, delay**

I. Introduction

With more and more peer-to-peer (P2P) streaming systems (e.g. [1], [2], [3]) deployed on Internet, P2P streaming technology has achieved a great success. Most of these systems employ pull-based streaming protocol. In this type of protocol, each node independently selects its neighbors so as to form an unstructured overlay network. The live media content is divided into segments and every node periodically notifies its neighbors of what packets it has. Then each node explicitly requests the blocks of interest from its neighbors according to their notification. The well-known advantages of pull-based protocol are its simplicity and robustness [4], [5]. In peer-to-peer streaming study, there are a lot of efforts made on enhancing the throughput of a P2P overlay. The maximum throughput that a P2P overlay can achieve is the total upload capacity of all peers divided by the peer number. Therefore, how to utilize all the peer upload capacity is the key issue. Traditional single tree based protocol can not achieve this because the upload capacity of the leaf peers and those peers with upload capacity less than the streaming rate cannot be utilized. So existing approaches slice the stream into multiple sub streams and leverage optimization theory to distributedly solve "packing spanning trees" problem [6], [7] or directly build multiple trees [8], [9] to utilize the peer upload capacity as much as possible. Recent study also employs network coding to deal with the issue of maximizing throughput [10]. However, our study by both simulation and real-world experiment demonstrates that, when the server bandwidth is above a reasonable number of times of the raw streaming rate, with appropriate parameter settings, simple pull-based protocol is near-optimal in terms of peer bandwidth utilization and system throughput. The most interesting part is that the pull-based protocol can achieve this property without intelligent scheduling and proactive bandwidth measurement. Yet this characteristic has been paid little attention to in the literature. To explain such near optimality of pull-based protocol, we further give mathematical analysis to help understand why pull-based protocol is effective in utilizing peer upload capacity. Moreover, we also point out that this near optimality in bandwidth utilization depends on the parameters in the pull-based protocol, the server bandwidth and the group size. We see that when the server bandwidth is at least 3 times of the raw streaming rate and the group size is less than 10,000 peers, such near optimality holds.

However, the benefit of pull-based protocol results from a cost, that is, the tradeoff between control overhead and delay. As indicated in the literature [5], to minimize the delay, each node notifies its neighbors of packet arrivals as soon as it receives the packet and the neighbor should request the packet immediately after receiving the notification, resulting in a remarkable control overhead. On the other hand, to diminish the overhead, a node can wait until dozens of packets arrived and inform these packets to its neighbors once in a packet meanwhile the neighbors can also request a bunch of packets once, which obviously leads to a considerable delay. Due to

Manuscript received March, 2007; revised August, 2007. Supported by the National Basic Research Program of China (973) under Grant No. 2006CB303103 and National High-Tech Research and Development Plan of China (863) under Grant No. 2006AA01Z321, the HKUST Nansha Research Fund NRC06/07.EG01, the HKUST RPC06/07.EG05, and CERG 622407.

Meng ZHANG, Prof. Lifeng SUN and Prof. Shiqiang YANG are with the Department of Computer Science and Technology in Tsinghua University (e-mail: zhangmeng00@mails.tsinghua.edu.cn, and {sunlf,yangshq}@tsinghua.edu.cn).

Prof. Qian ZHANG is with the Department of Computer Science in Hong Kong University of Science and Technology (e-mail: qianzh@cse.ust.hk).

Digital Object Identifier 10.1109/JSAC.2007.071207.

this limitation, most of the current deployed P2P streaming systems [1], [2] only target at the delay-tolerant non-interactive applications. A natural question is whether it is possible to not only achieve nearly optimal throughput but also obtain low delay and overhead. To break this tradeoff, in this paper, we propose the pull-push hybrid protocol. The basic idea of this protocol is to consider pull-based protocol as a highly efficient bandwidth-aware multicast routing approach and to push down the packets along the trees formed by pull-based protocol. Both simulation and real-world experiment results indicate that this protocol can not only achieve nearly optimal throughput but also has very low delay and small overhead in typical scenarios. Furthermore, pull-push hybrid protocol can achieve optimal bandwidth utilization with less server bandwidth than pull-based protocol. When the server bandwidth is at least 1.6 times of the raw streaming rate and the group size is less than 10,000 peers, the near optimality keeps true. We summarizes our contributions as follow:

- We give detailed simulation analysis and real-world experiments on *PlanetLab* of pull-based peer-to-peer streaming protocol and discover an interesting fact that, when the server bandwidth is above 3 times of the raw streaming rate, with appropriate parameter settings, simple pull-based P2P streaming can nearly utilize all the upload capacity of peers so that almost maximum throughput is achieved. We also clarify the parameters that have the most impact on the performance of pull-based method which is not well understood before.
- We give mathematical analysis to pull-based peer-to-peer streaming protocol to estimate the lower bound of the delivery ratio. So far as we know, we are the first to analyze the near optimality in terms of peer bandwidth utilization and throughput of pull-based P2P streaming.
- Based on the understanding that pull-based protocol is a highly efficient bandwidth-aware multicast routing scheme, we proposed a novel pull-push hybrid protocol, which can not only achieve nearly optimal throughput and bandwidth utilization but also has far lower playback delay and much smaller overhead. And the server bandwidth needed can be also relaxed to about 1.6 times of the raw streaming rate.
- We have totally implemented the proposed pull-push hybrid protocol and the system has been adopted by the largest TV station in China (CCTV) for TV online broadcasting. Our system has the record to support over 220,000 users concurrent online to watch high-quality Internet TV by only one server.

The remainder of this paper is organized as follows: in section II we briefly discuss the related work. And in section III, we give our deep insight into pull-based protocol and present our interesting findings by both simulation and real-world experiment. We also give mathematical analysis to explain our observations. Section IV describes the pull-push hybrid protocol and gives detail performance evaluation by both simulation and real-world experiment. In Section V, we present our insights obtained from our findings and give our suggestions for future study in this area. Finally, we conclude this paper in Section VI.

II. Related Work

Unlike traditional tree(s)-based protocols [11], from 2005, a new category of data-driven or pull (or swarming) based P2P streaming protocols (very similar to the BitTorrent[12] mechanism) has been proposed, such as Chainsaw [13], DONet/CoolStreaming [4], PRIME [14] and [15]. Most of the real P2P streaming system deployed over Internet are based on this type of protocol ([1], [16]). However, little attention has been paid to the near optimality of pull-based protocol in peer bandwidth utilization and throughput. To utilize all bandwidth capacity in P2P streaming, [8] and [17] mentioned a simple strategy to construct multiple height-2 trees within a fully connected graph, which can be only apply to small group since each node has to establish connections with all other nodes certainly resulting in a huge overhead. Besides, there is also much theoretical work to achieve the maximum overlay multicast throughput using optimization theory and network coding technique, such as [6], [7], [10]. However, our discovery shows that random packet scheduling strategy in pull-based P2P streaming protocol can run near-optimally in terms of peer upload capacity utilization. Some results in [18], [19] confirm that the simple rarest-first policy in Bit-Torrent system performs near-optimally in terms of upload capacity utilization and thereby the downloading time. Why random strategy in P2P streaming is near-optimal in peer bandwidth utilization? An intuitive explanation is that, unlike file sharing, there are always new block generated at the server, even if the random packet scheduling policy is used, the block diversity can usually be satisfied.

For the mathematical analysis of the delivery ratio of pull-based P2P streaming protocol, little work has been done to analyze this near optimality of pull-based P2P streaming. [20] gives a fluid model of BitTorrent-like file sharing system. [17] discusses the impact of high churn rate on the performance of P2P streaming. [21] gives a random distributed to solve the broadcast problem in edge-capacitated networks. However, none of them do analysis to explain the effectiveness of pull-based protocol in peer bandwidth utilization and throughput.

Interestingly, we can see that, the recent debate on peer-to-peer streaming focuses on whether mesh protocol (i.e. pull-based protocol) or multi-tree protocol is better [5], [22]. The main argument on the anti-tree side is that tree construction is complex, and that trees are fragile. The main counter-argument is that mesh/pull based protocol has a lot of overhead and large delay (it is validated by our results). However, our proposed pull-push hybrid protocol not only inherits the robustness of pull-based protocol and can achieve nearly optimal throughput but also has much lower overhead and delay. We note that SplitStream [23], ESM [24] and Chunkyspread [5] use multi-tree based protocol. However, it is not clear that whether these protocols can achieve nearly optimal throughput of the system, which is a very important feature of our proposed pull-push protocol.

III. Deep Insight into Pull-based Method

A. Protocol

In this section, we introduce the pull-based protocol we use, including overlay construction and streaming delivery.

For overlay construction, to join a P2P streaming session, nodes must first contact a rendezvous point (RP) which can be a server maintaining a partial list of current online nodes. Then each node randomly finds some other nodes as its neighbors to keep connections with so that an unstructured overlay is built.

For the streaming delivery, 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 (not including 40-byte packet header). 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. We call the packets used for protocol control *control packets*, including buffer map packets, request packets and packets for overlay construction, etc. To reduce the control packets, in our simulation and implementation, the control packets will be piggybacked in the streaming packets as much as possible.

More detailedly, in each *request interval* τ, the node asks its neighbors for all the absent packets in the current *request window*. The request window slides forward continuously. The head of the request window is the latest packet with the maximum sequence number the node has ever known, which is acknowledged by the buffer map received from its neighbors. The buffer size (usually 1 minute, denoted by B) is larger than the request window size (denoted by W). An absent packet that gets out of the request window (i.e. beyond the playback deadline) will not be requested further. If multiple neighbors own the same absent packet, the packet will be assigned to one of the neighbors *randomly with the same probability*. The request interval τ is set to much smaller than the request window W. When a packet does not arrive after its request has been sent out for a time of T_w and it is still in the request window, it will be requested again (i.e., a retransmission happens). T_w is called the waiting timeout for a requested packet.

We would like to emphasize our packet request and retransmission control here, which have some important difference from those described in literature, such as [13], [4], [25]. The key point here is to use *unreliable* transmission for streaming packets. If UDP is used, the unreliability is inherent. While, if TCP is used, we set the sending buffer of TCP socket to a small value, such as 2KB, and once the sending rate is beyond the bandwidth, the TCP sending buffer will get overflow and the packet is dropped. The benefit of using unreliable transmission for streaming packet is that we can use relatively small timeout to conclude whether a packet is dropped without encountering redundancy. Nevertheless, if reliable transmission is used, packets are usually queuing at the sender if the sending rate is high and we should use a very large timeout to judge whether the packet is dropped or will be received later. In our pull-based protocol, once a node receives a request (containing requests for multiple packets) from one neighbor, it will deliver all the requested packets within a constant bit rate exactly in time of one request interval τ. If the sending rate exceeds the upload capacity or end-to-end bandwidth, the overloaded packets will be dropped directly and never be retransmitted until a new request of this packet is received. Hence a packet will either arrive within time of $rtt + \tau$ or be dropped after it is requested, where rtt is the round-trip time to the neighbor. Moreover, considering the Internet transmission delay jitter, we set the waiting timeout T_w for a requested packet as $rtt + \tau + t_w$, where t_w is an additional waiting time to further prevent duplicated requests for a packet.

In a nutshell, in our streaming delivery protocol, for the receiver side, it selfishly requests an absent packet again and again until this packet eventually arrives or is out of the request window; for the sender side, it tries to send all the packets requested with the best effort.

B. Evaluation by Simulation

1) Simulation Setup: 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. We run our simulator on two 4-cpu 8GB-memory machines. For the end-to-end latency setup, we employ real-world node-to-node latency matrix (2500×2500) measured on Internet [26], in which, the average end-to-end delay is 79ms. We do not consider the queuing management in the routers. The default number of nodes (i.e. the group size) is set to 10,000. Since the node number is larger than the latency matrix dimension, we simply map each node pair in our simulation to each pair in the latency matrix randomly. And the default raw streaming rate (not including 40-byte/packet header overhead) is set to 300kbps. We assume all the control messages can be reliably delivered (streaming packets do not). 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 host. The default upload capacity of the source node is 2Mbps. To simulate the bandwidth heterogeneity of the peers, we use three different typical DSL nodes. Their upload capacities are 1Mbps, 384kbps and 128kbps and download capacities are 3Mbps, 1.5Mbps, 768kbps respectively. Note that the upload capacity are far lower than the download capacity, hence the upload link is usually the bottleneck. We use the term *capacity supply ratio* (also known as resource index in [27]) to represent the bandwidth supply "tightness", which is defined as the ratio of the total upload capacity to the raw streaming rate (default value 300kbps) times the receiver number, i.e., the ratio of bandwidth supply to the minimum bandwidth demand. It is obvious that the necessary condition of supporting a peer-to-peer streaming system is that the total bandwidth supply should outstrip the minimum bandwidth demand [17], i.e., the capacity supply ratio should be greater than 1. In Table I, by adjusting the fractions of different types of the peers, we obtain several capacity supply ratios. We will investigate the performance of pull-based method under these different capacity supply ratios.

As the bottleneck of P2P systems are always at the upload of the peers. The maximum throughput that can be achieved is

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

TABLE I
DIFFERENT CAPACITY SUPPLY RATIOS BY ADJUSTING THE FRACTIONS OF DIFFERENT PEER TYPES

Capacity Supply Ratio	Fractions			Capacity Supply Ratio	Fractions		
	1M	384k	128k		1M	384k	128k
1.02	0.1	0.346	0.554	1.2	0.15	0.39	0.46
1.05	0.1	0.38	0.52	1.25	0.16	0.405	0.435
1.1	0.1	0.45	0.45	1.3	0.18	0.4	0.42
1.15	0.1	0.5	0.4	1.4	0.2	0.45	0.35

the total upload capacity (including the source node) divided by the receiver number. Therefore, if every node in a system with capacity supply ratio of 1 can get the entire streaming, the optimal throughput is achieved.

2) Metrics:

- *Average deliverable rate* and *average delivery ratio*: we define the *deliverable rate* of a node as the available streaming rate received (excluding redundant streaming packets and control packets but including streaming packet header) by the node. And only the packets that arrive before the playback deadline are available ones. And we define the *delivery ratio* of a node as the ratio of its deliverable rate to the packetized streaming rate sent out from the source node. In our simulation and experiment, we use the total received streaming packets by a node and the total packets sent out from the source node during the latest 10 seconds to derive the *sampled* deliverable rate and delivery ratio of the node. The average sampled deliverable rate and average sampled delivery ratio of a session are the average values among the sampled deliverable rates and delivery ratios of all the nodes at that time. And when the system gets steady state, we average all the average sampled deliverable rate and delivery ratio throughout the whole duration as the *average deliverable rate* and *average delivery ratio* of the session. Both the two metrics are measures of the system throughput. The average numbers of the following metrics of the system are similar to that of the the deliverable rate and delivery ratio.
- *Average upload rate*: the upload rate of a node consists of all streaming packets and control packets that are *successfully* uploaded. The method to derive average upload rate of the whole session is similar as that of average delivery ratio. The more the average upload rate approaches average upload capacity, the more efficient the upload capacity is used
- *Packet arrival delay*: the packet arrival delay is the delay between the time when it is sent out from the source node and when it is finally arrived at a node after one or several hop(s).
- *α-playback delay* ($\alpha \leq 1$): in video streaming systems, usually, the packets should be buffered before they are played, and the longer time they are buffered, with higher possibility the better playback quality (i.e. delivery ratio) may be achieved. We call the minimum buffered time when the delivery ratio reaches α as the α-playback time. And we define *α-playback delay* as the delay between the time when a packet is sent out from the source node and α-playback time of that packet. This metric captures the true delay experienced by the end users. The default value of α in our simulation is 0.99.
- *Control packet rate*: the control packets include buffer map packets, request packets, member table packets, connection setup and heart-beat packets, etc. The control packet rate representing the control overhead is the control packet sent out per second from each node..

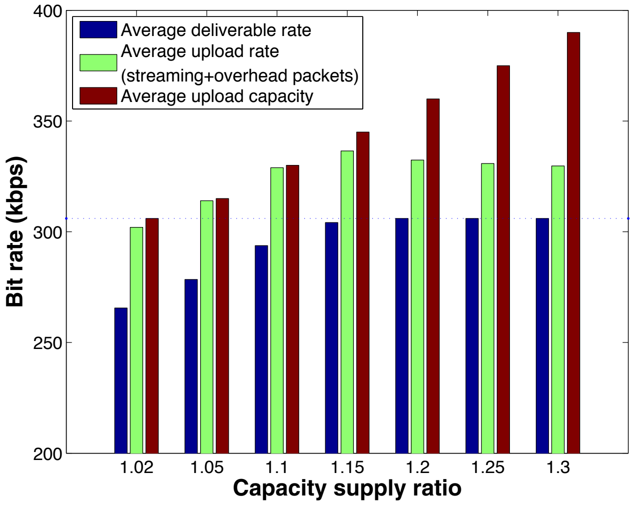


Fig. 1. Average deliverable rate, average upload rate and average upload capacity in pull-based protocol

3) Simulation results: We first investigate the performance of pull-based protocol in static environment. In static environment, users do not quit after they join the session. Our simulation results here indicate the system behavior at the steady state. For each point, we simulate s session for half an hour. It takes about one day to perform one run for one point. Furthermore, we have repeated several simulation runs for each of some points over different random seeds and we found that the results have been very similar.

Fig. 1 gives a comparison of i) the average deliverable rate, ii) the average upload rate and iii) the average upload capacity under different capacity supply ratios. The request interval is set to 200ms. Generally speaking, the average upload rate should be equal to the average download rate regardless of the packet loss in the non-last-hop links. Obviously, the average upload rate including both streaming and control packets, should be greater than the average deliverable rate, and both of them are upper bounded by the average upload capacity. The dotted horizontal line in each sub figure indicates the packetized streaming rate sent out from the source node (i.e. the best deliverable rate that can be achieved). As indicated in the figure, we note that when the capacity supply ratio is 1.15 or above (i.e., as long as the bandwidth supply is higher by only 15% than the minimum bandwidth demand, that is, the raw streaming rate 300kbps times the receiver number), the average deliverable rate can reach the nearly best deliverable rate. Considering the 3% header overhead and 5%$\sim$10% control overhead, the deliverable rate achieves nearly the optimal throughput. And when the capacity supply ratio is lower than 1.15, we see that the average upload rate is very close to the average upload capacity, namely, the capacity is almost fully utilized.

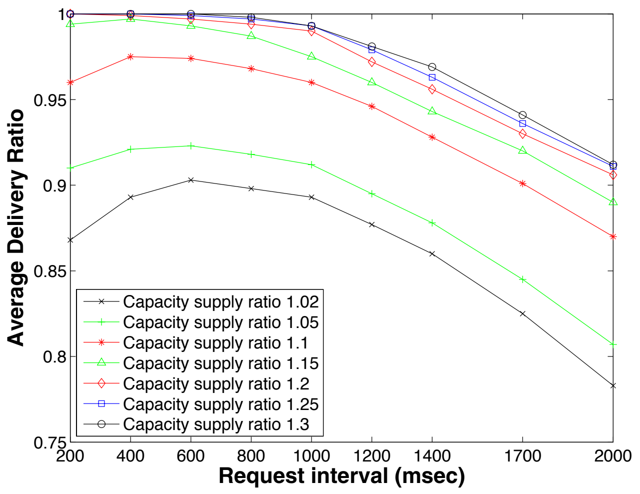


Fig. 2. Average delivery ratio with respect to different request intervals and capacity supply ratios in pull-based protocol

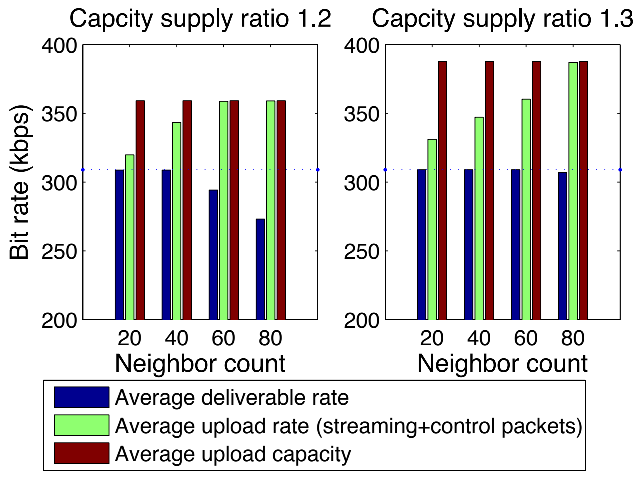


Fig. 4. Delivery ratio with respect to neighbor count and capacity supply ratio. Capacity supply ratio is 1.2 and request interval is 500ms.

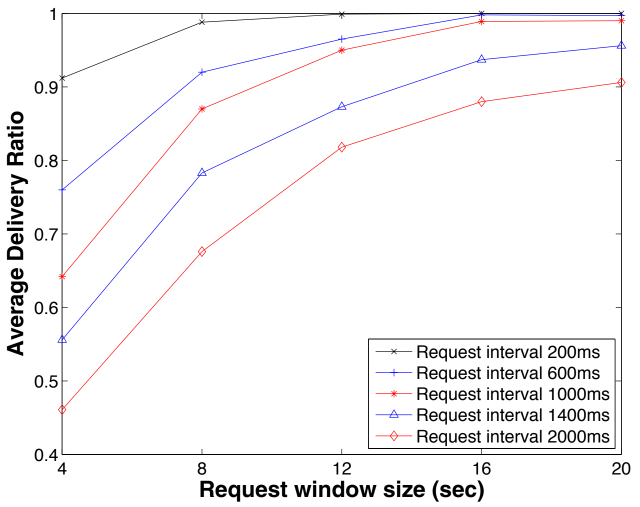


Fig. 3. Delivery ratio with respect to different request interval and request window size in pull-based protocol

Fig. 2 shows the average delivery ratio under different capacity supply ratio and request interval. We see that when the capacity supply ratio is 1.15 or above and request interval is between 200ms and 800ms, the average delivery ratio is very close to 1 (i.e., the best delivery ratio). And we note that when the capacity supply ratio is higher than 1.15, the smaller the request interval is, the better delivery ratio is achieved. Intuitively, small request interval implies that there are more chances for a packet to be successfully retransmitted until it gets out of the request window. On the other hand, if the capacity supply is very limited (below 1.15), the delivery ratio gets worse when the request interval is very small (such as under 400ms). This is because small request interval causes heavier control overhead traffic, which consumes more bandwidth resulting in lowering delivery ratio.

Fig. 3 shows the impact of request window size under different request window interval. Here, the capacity supply ratio is set to 1.2. When the request windows size is small, the delivery ratio gets poor. While, we notice that, with enlarging the request window size, the delivery ratio can be remarkably improved under the same request interval. And the delivery ratio can achieve nearly 1 when the request interval is below 1000 msec and the request window size reaches 20 sec. In fact, expanding the request windows size will also increase the chances of repeated requests for a packet till an eventual successful request. On the other hand, note that with the same window size, the delivery ratio increases much when the request interval gets smaller. The reason is almost the same: smaller request interval also results in more request retries. In fact, request interval and request window size are the two most important parameters having impact on the delivery ratio of pull-based protocol. We analyze the relationship between delivery ratio and these two parameters in our mathematical analysis in Section III-D.

Fig. 4 shows the impact of the neighbor count (i.e. overlay construction) on the performance when the capacity supply ratio is 1.2 and 1.3. Note that when the capacity supply ratio is 1.2 and the neighbor count exceeds 60, the deliverable rate drops below packetized streaming rate (optimal rate). And the upload rate is almost fully utilized. On the other hand, we see that when the capacity supply ratio is 1.3, the deliverable rate is nearly optimal, even if the neighbor count is up to 80. So the observation is that, if the upload capacity can support all the streaming data and all control overhead, the deliverable rate can achieve the near-optimal rate with a very wide rage of neighbor count.

In Fig. 5, we plot the average control packet rate (not including the piggyback control packets). It is found that the control packet rate is usually above 30 packets/sec. We know that the streaming packet rate is 30 packets/sec as the raw streaming rate 300kbps and raw packet size is 1250 bytes. Therefore, there are a dramatic number of control packets that are even more than the streaming packets. And smaller request interval and capacity supply ratio causes more control packets.

Fig. 6 shows the cumulative distribution function (CDF) of 0.99-playback delay under different request interval, which indicates the "true playback latency" experienced by the users. The capacity supply ratio is set to 1.2. Note that almost all the peers have a playback delay over 16 seconds. In fact, in pull-based method, a packet is not relayed immediately after a node receives it, moreover, the retransmission is very frequent due

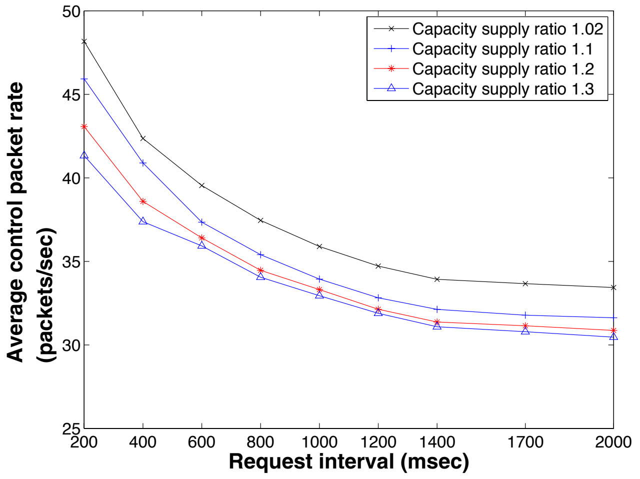


Fig. 5. Average control packet rate with respect to different request intervals and capacity supply ratios in pull-based protocol

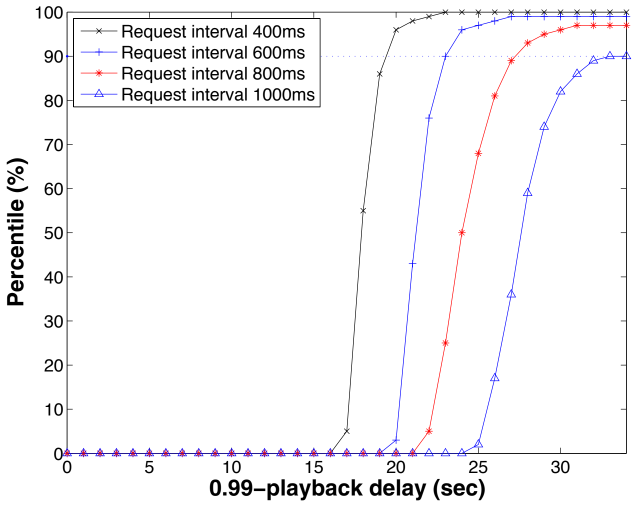


Fig. 6. CDF of 0.99-playback delay among all peers under different request interval (capacity ratio 1.2) in pull-based protocol

to the bandwidth limitation and packet loss. And these delays are cumulated hop by hop. The minimum playback delays in which 90% users have 99% delivery ratio with the request intervals of 400, 600, 800, and 1000 msec are respectively 19, 23, 27 and 33 seconds, all of which are considerable delays (note that these results are performed in the static environment). And from Fig. 5 and 6, we note that smaller request interval results in lower delay; however, it also causes more control packets.

Then we investigate the impact of the server bandwidth, group size and request window size to the delivery ratio in Fig. 7. We use capacity supply ratio 1.2 here. As shown, the delivery ratio increases with the increase of the server bandwidth. When the request windows size is 20 sec, the delivery ratio can reach nearly 100% if the server bandwidth is 900kbps, i.e., 3 times streaming rate for all types of group sizes. And we see that with the same server bandwidth, the delivery ratio of larger group size is slightly lower. This is because if the server bandwidth is low, there will be very competitive packet requests to the server and some of the unique packets are failed to be sent out. Moreover, many

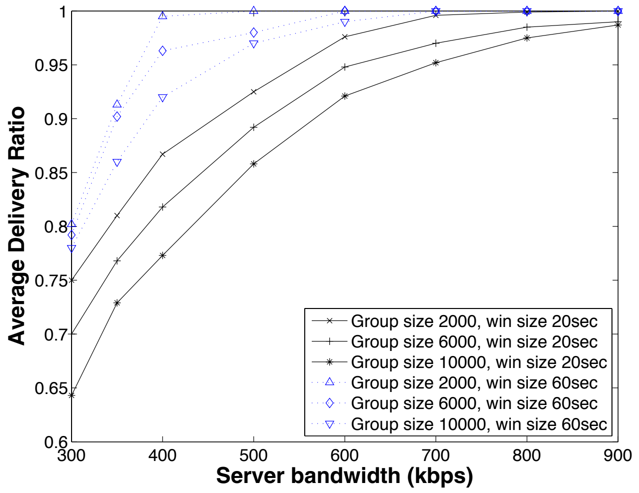


Fig. 7. Average delivery ratio with respect to the server bandwidth, group size and request window size

packets are received from the server after a lot of request attempts, resulting in a very large delay, and hence many packets will get out of the request window before spreading to the peers after several hops. And thereby we also check the delivery ratio when the request window is enlarged to 60 sec. As shown, under the same server bandwidth, the delivery ratio can be improved with increasing the request window size. However, this causes much control overhead and playback delay.

Finally, we investigate the performance of pull-based method under dynamic environment. We use the traces from real-deployed P2P streaming system - GridMedia [3], [28]. This system has been online to broadcast programs for CCTV (the largest TV station in China) since Jan. 2005. The traces are chosen from those of "CCTV1" channel that mainly shows news, sitcom, and interview programs. And we use the traces between 8:00PM and 9:00PM on 11th-13th Nov. 2005. The average user online time in the traces is 1500 sec. The total concurrent online number of this period on the 3 days is about 9000$\sim$10,000. To simulate a user behavior with a rough group size of 10,000, we use the superposition of the same period of this three days. In the simulation, the users join and depart from the group and the total online users increase form 0 to nearly 10,000. Note that the nodes that join the session before 8:00PM are not inclusively simulated even though they may depart within this period. So the arrival rate is greater than the departure rate leading to the rise of the online number. The capacity supply ratio is 1.2 and request interval is 500ms. Fig. 8 shows the results. The delivery ratio is always above 0.97 and in most time is over 0.99. The packet arrival delay is always around 17 sec and the playback delay is approximately 25 sec, larger than that in static (roughly 20 sec).

C. *Evaluation on* PlanetLab

We have implemented both pull-based and pull-push hybrid protocol and conducted real-world experiments on PlanetLab [29]. We upload our client program to 409 nodes[2] on Plan-

[2]As not all nodes are active at the same time on PlanetLab, it is almost the most nodes that we can control

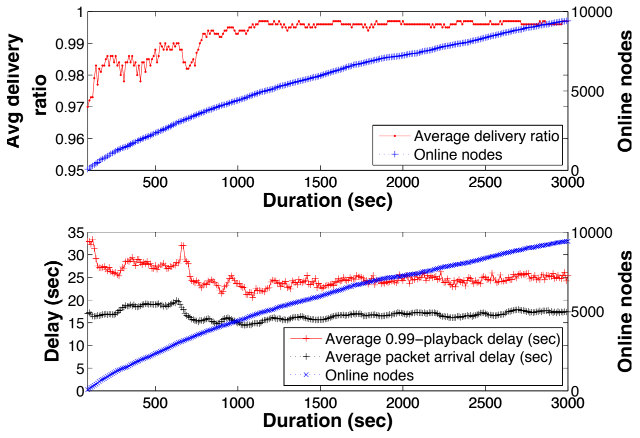


Fig. 8. Average delivery ratio and average playback delay driven by real P2P streaming system user trace

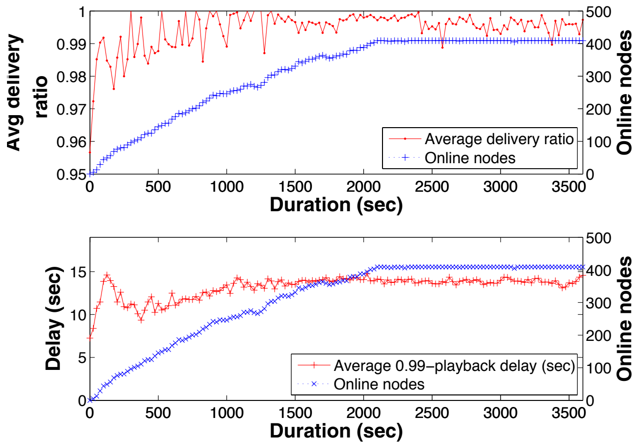


Fig. 9. **PlanetLab** experiment with 409 nodes. Average delivery ratio and average playback delay driven by real P2P streaming system user trace

etLab. We also limit the upload bandwidth of each node to 1Mbps, 384kbps and 128kbps and the capacity supply ratio is 1.2. Some special nodes are shown in Table II. We also use GridMedia traces of "CCTV1" channel to drive the experiment. The traces used are between 9:00AM and 10:00AM on 16th Nov. 2005 and the maximum concurrent online number in this period is 762. In our experiment, when a node in the traces joins, we select an available node on PlanetLab to join and once a client quits, it may correspond to another node in the traces and joins again. We let each node have 15 neighbors and set the request interval to 500ms. The source node upload capacity is 2Mbps. We performed multiple experiments for the same parameter settings. The results are very similar to each other. To avoid verbosity, we only plot the whole process in a single representative experiment in Fig. 9.

As shown in Fig. 9, the delivery ratio is above 99% in most of the duration, very close to 1. The playback delay is around 13 seconds. It is smaller than that in our simulation. since the group size in PlanetLab experiment (409) is much smaller than that in our simulation. But the playback delay is still considerable.

TABLE II
SPECIAL NODES USED IN PLANETLAB EXPERIMENT

Special node	URL
RP	planetlab1.csail.mit.edu
Source node	planetlab1.cs.cornell.edu
Log collecting server	planetlab6.millennium.berkeley.edu
Command nodes[a]	thu1.6planetlab.edu.cn

[a]Command nodes are used to control other nodes to join and leave

D. Analysis

From our simulation and real-world experiment, we have seen that the pull-based protocol is very powerful in bandwidth capacity utilization and throughput. In fact, with appropriate parameter settings, pull-based method can achieve nearly 100% delivery ratio if the total upload capacity among all peers can support the minimum bandwidth needed plus 15% header and control overhead. Actually, in theory, to achieve the maximum overlay multicast rate through a capacitated network is known as the "packing spanning/steiner trees" problem. To solve this problem, it is usually assumed that all the link and node capacity can be known in advance and most proposed distributed approximate optimal protocol also has this assumption. However, these parameters are very hard to obtain on Internet. Recall that, in the pull-based protocol, there is no proactive bandwidth measurement and what a receiver does is just to selfishly request the absent packets in the request window as many as possible, while what a sender does is just to send the requested packets with best effort. Moreover, we even do not use any intelligent packet scheduling for packet request (a purely random way is used). However, the traffic load is properly allocated among nodes with respect to their upload capacity and the nearly optimal deliverable rate is reached. A natural question is how such simple protocol can do this. In this section, we try to explain the power of pull-based method using a simple stochastic model. To our knowledge, we are the first to mathematically analyze *the near optimality in capacity utilization and throughput* of pull-based P2P streaming protocol.

The notations used in this section are summarized in Table III. To understand pull-based protocol better, we first analyze two simple cases as illustrated in Fig. 10 and Fig. 11. In Fig. 10, one receiver requests packets from n senders. We let u_i, $i = 1, \cdots, n$ denote the upload capacity of each sender. In Fig. 11, two receivers request packets from three senders and they have a request competition at sender 2. In our analysis, we assume that the download capacity of all nodes is greater than the streaming rate r. In these two simple cases, we assume each sender has already had a full copy of live streaming data. For simplicity, we also assume that in each interval the requested packets are evenly distributed to each sender, and the bottleneck is only at the last hop. We will discuss the case when the senders are internal nodes later. Here we study the steady number M of packets requested in one interval and the successful request probability in one interval. The successful request probability (SRP for short) is the probability that a packet can be successfully requested in one request attempt. Particularly, when all the senders have full copies of data, we call the successful request probability in stationary state *static SRP*. Then we have the following propositions:

TABLE III
NOTATIONS

Notation	Description
r	The packetized streaming rate, kbit/sec, including the IP, UDP and RTP header
l	The streaming packet size/length, 1290bytes (including 40-byte header)
u_i	The upload capacity of sender i, kbit/sec
b_i	The bandwidth consumed of sender i, kbit/sec
τ	The request interval
$T_w = \omega\tau$	The waiting timeout for requesting a packet again. T_w should be greater than τ. For simplicity, we assume T_w is integral times of τ, that is, ω is an integer.
W	The request window size, in seconds. Hence rW_r/l is the maximum packets in the window
B	The buffer size, in seconds, and $B > W$
M	The steady number of packets to request in a request window when stationary state is reached
k	The request interval index
h	Hop count
p_k ($p_k^{(h)}$)	SRP (successful request probability) in kth interval (of the h-hop node)
q_k ($q_k^{(h)}$)	Delivery ratio/quality within the first kth interval (of the h-hop node)
π	The static SRP(successful request probability), if all the senders had a full copy of data
$D_r = d\tau$	The average request delay one hop. For simplicity, we assume D_r is integral times of τ

Proposition 1: For the topology in Fig. 10, if $\sum_i u_i > r$ and the request window W is large, the number of packets requested in one interval within the request window will eventually reach its steady number M ($rW/l > M$), which satisfies

$$\sum_{i=1}^{n} \min\{u_i\tau/l, M/n\} = r\tau/l \qquad (1)$$

And the bandwidth consumed at sender i is $b_i = \min\{u_i, lM/(n\tau)\}$. The static SRP in each interval is

$$\pi = r\tau/lM \qquad (2)$$

Proof sketch: Without generality, Fig. 12 shows an example for requesting packets from three senders. The packets requested from each sender in one interval is M/n since packets are scheduled to be evenly distributed among senders. Meanwhile, the packets that sender i can be sent in one interval cannot exceed $\tau u_i/l$. So the packets sent by sender i is exactly $\min\{\tau u_i/l, M/n\}$ and thus the bandwidth consumed at sender i is $b_i = \min\{u_i, lM/(n\tau)\}$. And it is obvious that the request number will remain if and only if the incoming packets in current interval are as many as the fresh packets to request in the next interval. The fresh packets to request in each interval are $r\tau/l$. Therefore, if $\sum_{i=1}^{n} \min\{\tau u_i/l, M/n\} = r\tau/l$ (it can be satisfied because $\sum_i u_i > r$), then M is the steady request number. Furthermore, in each interval, only $r\tau/l$ packets can be successfully delivered among all M requested packets, so for a specific packet, its static SRP is $r\tau/(lM)$. Finally, we demonstrate the convergence of M. When the request number in a interval is smaller than M, there will be less packets arrived and these packets are requested again when timeout occurs, resulting in the increase of request number in the next interval. On the other hand, since the total upload capacity is larger than r, if the request number is larger than M, there will

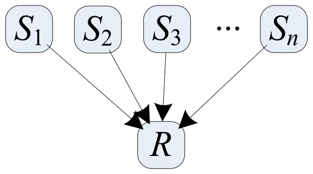


Fig. 10. Case I: 1 receiver requests packets from n senders

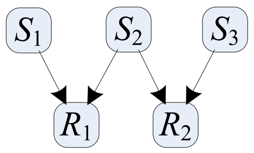


Fig. 11. Case II: 2 receivers competitively request packets from 3 senders

be more than $r\tau/l$ packets arrived leading to the decrease of request number in the next interval (in which only $r\tau/l$ fresh packets are inserted to the window). Therefore, no matter how many packets are requested in the first interval, the request number will always converge to M as long as $\sum_i u_i > r$.

Proposition 2: For the topology in Fig. 11, if $u_1 + u_2 > r$, $u_2 + u_3 > r$ and $\sum_{i=1}^{3} u_i > 2r$ and the request window W is large, the number of packets requested of R_1 and R_2 in one interval within the request window will respectively reach its steady number M_1 and M_2 ($rW/l > M_1, M_2$), which satisfies $\min\{u_1\tau/l, M_1/n\} + \min\{u_2\tau/l \cdot \frac{M_1}{M_1+M_2}, M_2/n\} = r\tau/l$ and $\min\{u_2\tau/l \cdot \frac{M_2}{M_1+M_2}, M_2/n\} + \min\{u_3\tau/l, M_2/n\} = r\tau/l$ (where $n = 2$). And the bandwidths consumed at sender 1, 2 and 3 are respectively $b_1 = \min\{u_1, lM_1/(n\tau)\}$, $b_2 = \min\{u_2, l(M_1 + M_2)/(n\tau)\}$ and $b_3 = \min\{u_3, lM_2/(n\tau)\}$. The receiver R_1 and R_2's static SRPs π_1 and π_2 are respectively $r\tau/lM_1$ and $r\tau/lM_2$.

Note that owing to the competitive requests, the successfully delivered packets of sender 2 to each receiver are proportional to the total request number from both receivers. The proof is similar to that of Proposition 1 and we omit it here.

In fact, for any given topology, we can compute the steady request number and static SRP of each node as long as the capacity supply is larger than the bandwidth demand. Due to page limitation, we do not further discuss the results for general topology here.

Then we can derive the delivery ratio of the receiver. We let p_k represent the SRP in the kth interval. Each packet will be requested at most W/T_w times when it is within the window. The probability that a packet fails to arrive in all intervals is $\prod_{i=1}^{\lfloor W/T_w \rfloor} (1 - p_{i\omega})$, (here $\omega = T_w/\tau$) and hence the probability that a packet is finally successfully requested until the last interval $\lfloor W/\tau \rfloor$ is $q_{\lfloor W/\tau \rfloor} = 1 - \prod_{i=1}^{\lfloor W/T_w \rfloor} (1 - p_{i\omega})$. Note that this probability is also the delivery ratio (recall the definition of delivery ratio). In the first case above, when it comes into stationary state, the SRP in each request interval is just the static SRP π of the topology, i.e., $p_i = \pi = r\tau/lM$, ($1 \le i \le \lfloor W/\tau \rfloor$). So the delivery ratio is $q_{\lfloor W/\tau \rfloor} = 1 - (1 - \pi)^{\lfloor W/T_w \rfloor}$, $\pi = \frac{r\tau}{lM}$. And similarly, for the second case, the delivery ratios of the two receivers are respectively $q_{\lfloor W/\tau \rfloor} = 1 - (1 - \pi_i)^{\lfloor W/T_w \rfloor}$, $\pi_i = \frac{r\tau}{lM_i}$, $i = 1, 2$.

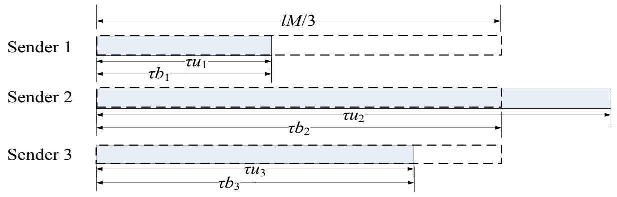


Fig. 12. Illustration for steady request number in one interval

TABLE IV
NUMERICAL EXAMPLES FOR THE TWO SIMPLE CASES

case	τ	T_w	r	W	u_1,u_2,u_3
I	500ms	1sec	300kbps i.e. 30packets/sec	20sec	50,100,150kbps
II	500ms	1sec	300kbps i.e. 30packets/sec	20sec	100,300,200kbps

case	M	$\pi = \frac{r\tau}{lM}$	$q = 1 - (1-\pi)^{\frac{W}{T_w}}$
I	$M = 22.5$	$\pi = \frac{15}{22.5}$	$q = 1 - 3 \cdot 10^{-10}$
II	$M_1 = 25$, $M_2 = 20$	$\pi_1 = \frac{15}{25}$, $\pi_2 = \frac{15}{20}$	$q_1 = 1 - 10^{-8}$, $q_1 = 1 - 10^{-12}$

To get an intuitive understanding, we give a numerical example for each case by computing delivery ratio from typical parameters. For the first case, the total upload capacity of 3 senders is equal to the streaming rate r. For the second case, the total upload capacity of 3 senders is equal to $2r$.

As shown in Table IV, in both cases, the delivery ratio q is almost one.

The considered cases above is one-hop scenario. In real scenario, packets are usually delayed, so it can not be assumed that each sender has full copy of data in every interval. The delay also causes the shift between the request windows/buffers of every two peers. A natural question is whether the delivery ratio keeps high after several hops. So we want to analyze the delivery ratio under different source-to-end delay and hops. Here, we define a term *h-hop node* as the node whose 99% packets are received within h hops. We then estimate the lower bound of delivery ratio of the h-hop node at different source-to-end delay.

To investigate the problem, we first clarify that the delay in each hop is made up of two parts - *retransmission delay* and *request delay*. The retransmission delay is caused by the request failure and request retry. Due to our analysis above we can easily compute the CDF of retransmission delay with the given SRP p_i in every request interval, that is, $P\{T < t\} = 1 - \prod_{i=1}^{\lfloor t/T_w \rfloor} (1 - p_{i\omega})$, $(\omega = T_w/\tau)$. Lower capacity supply results in smaller SRP and causes more retransmission delay. On the other hand, Fig. 13 shows the illustration of the request delay. When a packet arrives at the sender, its arrival is first notified to the receiver by buffer map, then it will wait for the request from the receiver and finally the packet is sent to the receiver. So the request delay for a packet in one hop includes three end-to-end network delays and two waiting delays (one for buffer map and the other for request). This delay is "hard" delay and is independent with capacity supply. We let $D_r = d\tau$ denote the average request delay, where d is an integer (for simplicity, we assume D_r is integral times of τ).

Since we want to estimate the lower bound of the delivery ratio of the h-hop node, we consider the worst case that all

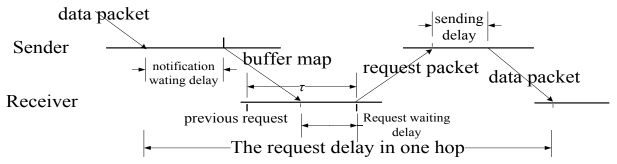


Fig. 13. Illustration for request delay in one hop

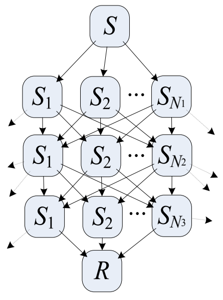


Fig. 14. A 4-hop node with maximum hops in each path

the paths from the source node to the receiver have h hops. Fig. 14 shows an example of a worst-case 4-hop node. So this node has the largest total retransmission and request delay and the hence poorest delivery ratio with the same source-to-end delay among all types of 4-hop nodes.

We let $p_k^{(h)}$ denote the SRP in the kth interval of the h-hop node. And let $q_k^{(h)}$ represent the delivery ratio till the kth request interval of the h-hop node. We assume that the source node can support the requests of all its neighbors (i.e. 1-hop node). So we have:

$$p_k^{(1)} = \begin{cases} 1 & 1 \le k \le \lfloor W/\tau \rfloor \\ 0 & \lfloor W/\tau \rfloor < k \le \lfloor B/\tau \rfloor \end{cases} \quad (3)$$

With given $p_k^{(h)}$, we can easily derive the delivery ratio $q_k^{(h)}$ within any k intervals.

$$q_k^{(h)} = \begin{cases} 0 & 1 \le k \le d \\ 1 - \prod_{i=1}^{\lfloor (k-d)\tau/T_w \rfloor} (1 - p_{i\omega}^{(h)}) & d < k \le d + \lfloor B/\tau \rfloor \\ q_{d+\lfloor B/\tau \rfloor}^{(h)} & k > d + \lfloor B/\tau \rfloor \end{cases} \quad (4)$$

In fact, since the average request delay is d intervals, the packet request in 1st interval will arrive in the $(d+1)$th interval. So in the first d intervals, there are no packets arrived.

On the other hand, given the delivery ratio of the h-hop node in every interval, i.e., $q_k^{(h)}$, we can derive the SRP of $(h+1)$-hop node. As the request delay is a "hard" delay, so the request window shifts behind by $D_r = d\tau$ in each hop. For a $(h+1)$-hop node which requests packets from n h-hop nodes, its request window is from $hd\tau$ to $hd\tau + W$. So the SRP outside the request window is zero:

$$p_k^{(h+1)} = 0 \quad (1 \le k < hd \ \text{ or } \ k > hd + \lfloor W/\tau \rfloor) \quad (5)$$

Then, for the SRP of a packet inside the request window (that is, $hd \le k \le hd + \lfloor W/\tau \rfloor$), we assume that an $(h+1)$-hop node requests from n h-hop node and the bandwidth consumed

$$
\begin{aligned}
&p_k^{(h+1)}\\
&=\sum_{i=1}^{n}\Bigg\{\underbrace{\Big(\frac{\tau b_1+\cdots+\tau b_i}{lM}+\frac{\tau b_1+\cdots+\tau b_{i-1}+\tau b_{i+1}}{lM}+\cdots+\frac{\tau b_{n-i+1}+\tau b_{n-i+2}+\tau b_n}{lM}\Big)}_{\text{For }\forall\ i\text{-combinations of }\{b_1,\cdots,b_n\}.\text{ Note that there are }\binom{n-1}{i-1}\text{ items containing }b_j}\\
&\quad\cdot(q_k^{(h)})^i(1-q_k^{(h)})^{(n-i)}\Bigg\}\\
&=\sum_{i=1}^{n}\binom{n-1}{i-1}\frac{\tau}{lM}\Big(\sum_{i=1}^{n}b_i\Big)(q_k^{(h)})^i(1-q_k^{(h)})^{(n-i)}\\
&\quad\text{(Note that, in stationary state, }\textstyle\sum_{i=1}^{n}b_i=r,\ \frac{\tau r}{lM}=\pi)\\
&=\sum_{i=1}^{n}\binom{n-1}{i-1}\pi(q_k^{(h)})^i(1-q_k^{(h)})^{(n-i)}\\
&=\pi q_k^{(h)}\sum_{i=1}^{n}\binom{n-1}{i-1}(q_k^{(h)})^{i-1}(1-q_k^{(h)})^{(n-1)-(i-1)}\\
&=\pi q_k^{(h)}
\end{aligned}
\tag{6}
$$

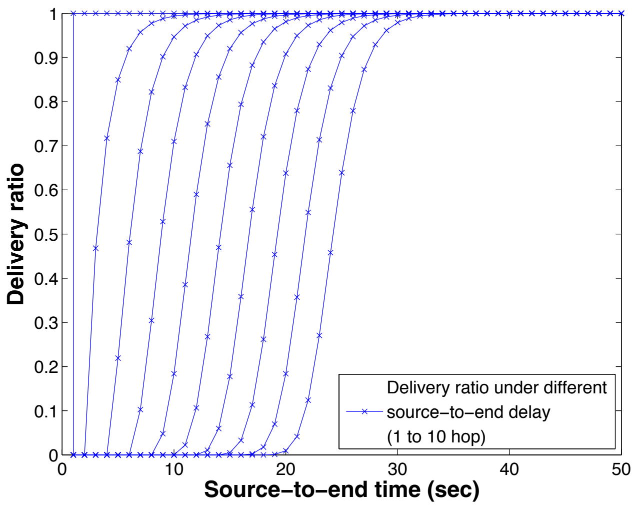


Fig. 15. Estimated lower bound of delivery ratio of 1-hop to 10-hop nodes by analysis

at each sender is b_i and the steady state is reached. When there is exactly i senders (denote by $j_1, j_2, \cdots, j_i$) hold the packet, the SRP can be easily computed as $(\tau b_{j_1} + \tau b_{j_2} + \cdots + \tau b_{j_i})/(lM)$, where M is the steady request number one interval. And the probability that exactly i senders hold the packet and that the rest do not is $(q_k^{(h)})^i(1-q_k^{(h)})^{(n-i)}$. So for any i-combinations $(1 \le i \le n)$ among the senders, we can compute its SRP. With the law of total probability, we have the SRP of the kth interval of $(h+1)$-hop node as shown in (6 above).

Finally, with Equation (3), (4), (5), (6), we can derive the delivery ratio of h-hop node under different source-to-end by iterations. We analyze the delivery ratio of an h-hop with typical parameters here. We set the capacity supply ratio to 1.2. And request interval τ is set to 500ms and request window and buffer windows are 20 sec and 60 sec. The request delay is estimated as 2τ. As discussed previously, the precise steady request number one interval of each node depends on the overlay topology. We here estimate the steady request number and static SRP in a simple way. We have three types of nodes whose upload capacities are 1Mbps, 384kbps, 128kbps respectively. Since each node has 15 neighbors, each receiver obtains 1/15 of its sender's upload capacity in average. According to the fraction of each type of nodes (as shown in Table I, it is $0.15 : 0.39 : 0.46$), without generality, we can assume, among the 15 neighbors, the number of the three upload capacity types are 2, 6 and 7. So the upload capacity of each sender that the receiver can utilize is $u_1, u_2 =$ 1M$/15 = 68.3$kbps, $u_3, \cdots, u_8 = 384\text{k}/15 = 25.6$kbps and $u_9, \cdots, u_{15} = 128\text{k}/15 = 8.5$kbps. Employing Equation (1), we can derive the approximate steady request number M one interval, that is, $M = 32.5$. And then according to Equation (2), we have the static SRP roughly $15/32.5 = 0.468$. Finally, we can compute the delivery ratio of each hop with Equation (3), (4), (5), (6) by iterations.

From our simulation, we observe that over 99% packets can be delivered within 10 hops under the group size of 10,000. So we plot the estimated lower-bound of delivery ratio as shown in Fig. 15. The curves from left to right indicate the delivery ratio under different source-to-end delay of 1-hop to 10-hop node respectively. The more hop count the node has, the worse delivery ratio it has under the same source-to-end delay. However, even the 10-hop node can achieve nearly 100% delivery ratio at source-to-end delay 30 sec. This demonstrates that although the SRP is only 0.468, the delivery ratio can achieve nearly optimal level even after 10 hops with the proper parameters (τ=500ms and W=20sec). Fig. 16 especially shows the estimated lower bound of delivery ratio of 10-hop node, meanwhile the delivery ratio of some randomly selected 10-hop node by simulation also plotted in the same figure. Note that all the delivery ratios of sampled nodes are above the lower-bound, which validates our analysis results. Furthermore, with our numerical analysis, the impact of some parameters can also be observed as in our simulation.

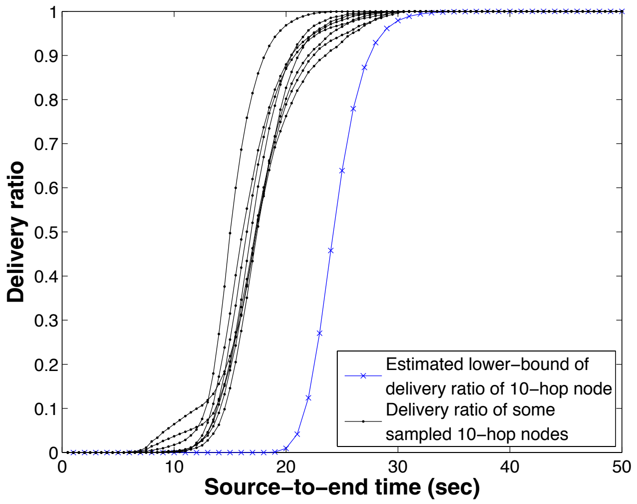


Fig. 16. Estimated lower bound of delivery ratio of 10-hop node and the delivery ratio of some randomly sampled 10-hop nodes

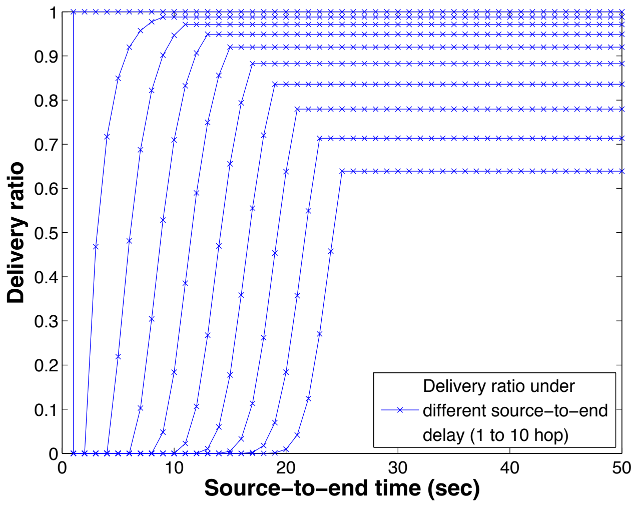


Fig. 17. Estimated lower bound of delivery ratio of 1-hop to 10-hop nodes. Request window size is set to 5 sec.

As shown in Fig. 17, we reduce the request window to 5 sec, and we can see that the lower bound of delivery ratio of 10-hop node dramatically diminishes to only about 63%.

Above all, we can see the secret of the pull-based protocol is that although some the request packets may not be satisfied in one request due to naive packet scheduling, however, after several request retries, the probability that a packet is always unsatisfied is very very low.

IV. Pull-Push Hybrid Method

In Section III-A, we have found that the pull-based protocol is nearly optimal in capacity utilization. Based on this observation, the basic idea of our pull-push hybrid method is to regard pull technique as a highly efficient bandwidth-aware routing protocol. Actually, in pull-based protocol, the streaming packets are delivered along near-optimally packed spanning trees. In a nutshell, our pull-push hybrid protocol is to push down the streaming packets along the trees formed by the pull technique.

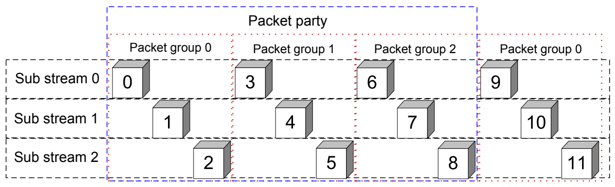


Fig. 18. An example that has 3 sub streams. Every packet group has 3 packets, and 3 packet groups make up a packet party

A. Protocol

The overlay construction used in the pull-push hybrid protocol is the same as that in the pull-based protocol. The peers self-organize into an unstructured random mesh. The streaming delivery in pull-push protocol is also simple just as pull-based protocol. Briefly, in pull-push protocol, the packets are first pulled from the neighbors and then pushed directly by the neighbors.

More detailedly, as illustrated in Fig. 18, we evenly partition the stream into n sub streams, and each sub stream is composed of the packets whose sequence numbers are congruent to the same value modulo n. And we group the every continuous n packets into a *packet group*, as shown in Fig. 18. Moreover, we cluster every continuous g packet groups into a *packet party*. Each packet group in a packet party is numbered from 0 to $g-1$ and hence the packet group number can be computed by $\lfloor s/n \rfloor \mod g$, where s is the packet sequence number. Fig. 18 shows an example that has 3 sub streams and $n = 3$, $g = 3$.

In the streaming delivery of pull-push protocol, when a peer joins, it first asks its neighbor to send it buffer maps periodically, and then pulls the required packets according to the packet availability in buffer map. Unlike in pull-based protocol, the buffer maps in pull-push are explicitly requested by the node and are not pushed from neighbor directly. Once a packet in packet group 0 in one packet party is requested successfully from a neighbor, the peer will send this neighbor a *sub stream subscription* to let it directly push the rest packets in the same sub stream. So there are two types of streaming packets - the *pulled packets* and the *pushed packets*. When over 95% packets are pushed directly from the neighbors, the node will stop requesting buffer maps. And once the delivery ratio drops below 95% or a neighbor who provides over 5% streaming data quits, the peer will start to request buffer maps and pull streaming packets from its neighbors.

A packet expected to be pushed directly may not arrive in time due to packet loss or bandwidth limitation. So the packets that do not arrive should be “pulled” from one neighbor after a waiting timeout. Usually, there are a little redundant packets due to the asynchrony between sender and receiver. We evaluate it in the simulation and experiment. In pull-push protocol, each node subscribes sub streams from its neighbors based on local information. A natural worry is whether there would be a subscription loop formed with the same sub stream just as the problem in tree-based protocol [5]. In fact, a node subscribes a sub stream only according to the packets in the packet group 0 of each packet party, and the sub stream is subscribed from a neighbor only if a packet is requested successfully from that neighbor. Moreover, in our pull-push

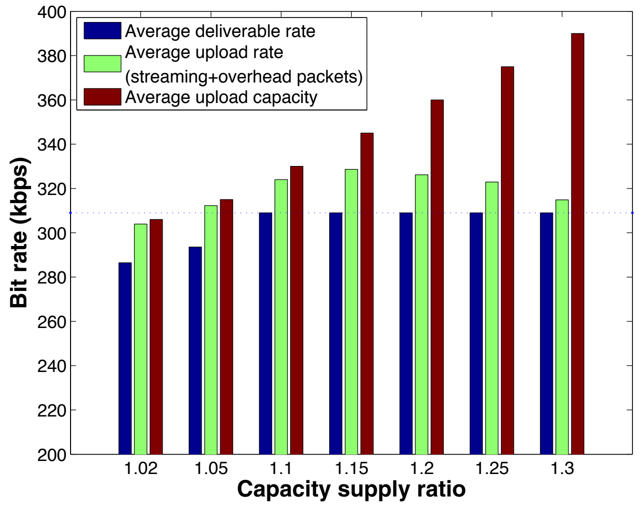


Fig. 19. Average delivery ratio with respect to different request intervals and capacity supply ratios in pull-push hybrid protocol

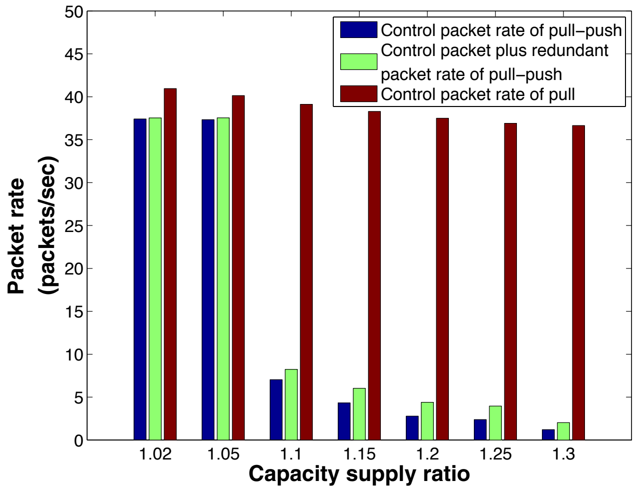


Fig. 21. Average control packet rate comparison between pull-based and pull-push hybrid protocol

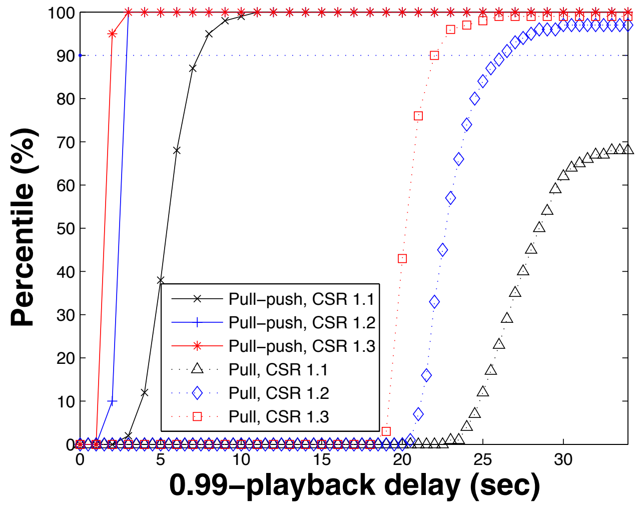


Fig. 20. CDF of 0.99-playback delay among all peers in pull-push hybrid protocol and pull-based protocol

hybrid protocol, the packets in one packet party are set to be greater than the request window size. As a packet is never requested twice, there are no chances to form subscription loops in the pull-push hybrid protocol.

B. Evaluation by Simulation

1) Metrics: Besides the metrics used in Section III-B2, we additionally use the following metrics to evaluate pull-push hybrid protocol.

- *Redundancy* and *redundant packet rate*: redundancy is the ratio of the duplicated streaming packets to the total traffic and redundant packet rate is the rate of duplicated streaming packets.
- *Push fraction*: The fraction of the pushed streaming packets among all streaming packets. Larger push fraction implies a lower delay.

2) Simulation results: Fig. 19 shows the average deliverable rate, the average upload rate and the average capacity. The dotted horizontal line indicates the packetized streaming rate (i.e. the maximum deliverable rate). We see that pull-push protocol is also nearly optimal in bandwidth utilization and even more effective in throughput than the pull-based protocol compared with Fig. 1, that is, when the capacity ratio is higher by only 10% than the minimum bandwidth demand, the deliverable rate can achieve the optimal rate.

Fig. 20 shows the CDF of 0.99-playback delay of the two protocols with different capacity supply ratios. Note that in pull-push hybrid protocol, when the capacity supply ratios are 1.3, 1.2 and 1.1, 90% users can respectively play back the video within 2.2, 3, 7 seconds. However, the playback delays of pull-based protocol are respectively 23, 26.5 and ∞ (pull-based protocol can not achieve maximum deliverable rate when the capacity supply ratio is 1.1), much larger than the delay of pull-push protocol.

Fig. 21 shows the comparison of the control overhead and redundancy between pull-push hybrid protocol and pull-based protocol. The control packet rate of pull-push hybrid protocol is far smaller than that in pull-based protocol when the capacity supply ratio is above 1.1. We notice that the aggregated rate of control packets and redundant packets in pull-push protocol (only about 1$\sim$5 packets/sec) is also much smaller than that in the pull-based protocol which is over 36 packet/sec, even more than the streaming packet rate.

Fig. 22 shows the impact of request interval and request window size on the delivery ratio of pull-push protocol. We also plot the results of pull-based protocol for comparison. The capacity supply ratio used is 1.2. Interestingly, rather than the significant impact on pull-based protocol, the request interval and request window size have little impact on pull-push protocol in static environment. We see that the delivery ratio of pull-push hybrid protocol remains nearly 1 with a wide range of request intervals and window sizes. This is because in pull-push hybrid protocol, once a packet is successfully requested, all the following packets in the sub stream will be relayed, and no more explicit request is needed. And there is not so many competitive and unordered packet requests as in pull-based protocol leading to a much larger successful request probability.

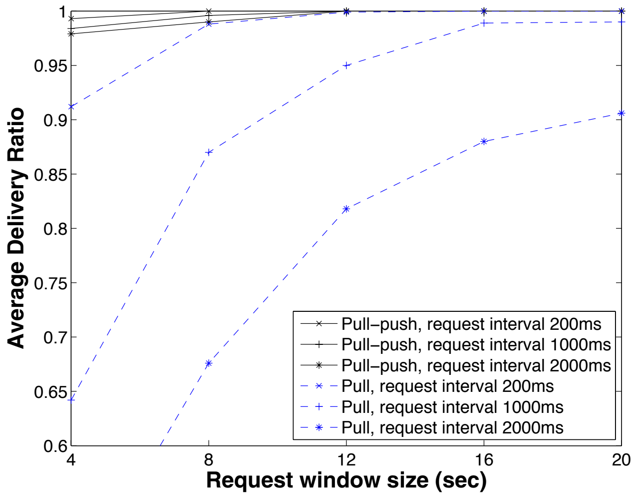


Fig. 22. Delivery ratio with respect to the request interval and window size in both pull-push hybrid protocol and pull-based protocol

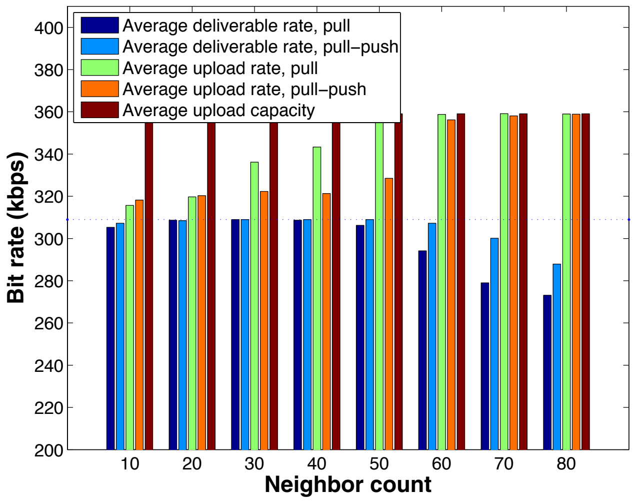


Fig. 23. Delivery ratio with respect to neighbor count in both pull-push hybrid protocol and pull-based protocol

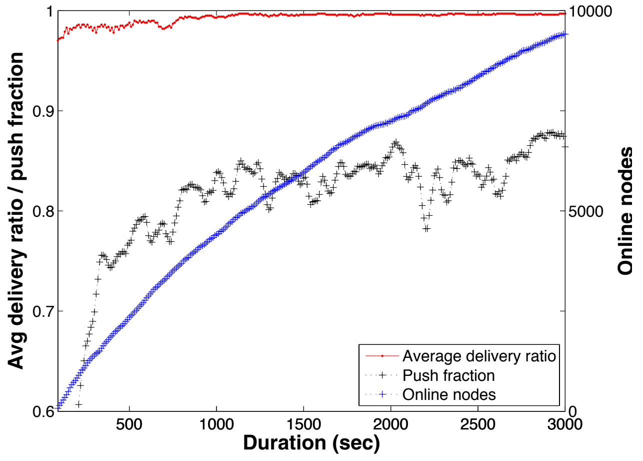


Fig. 24. Average delivery ratio and push fraction of pull-push protocol driven by real P2P streaming system user traces

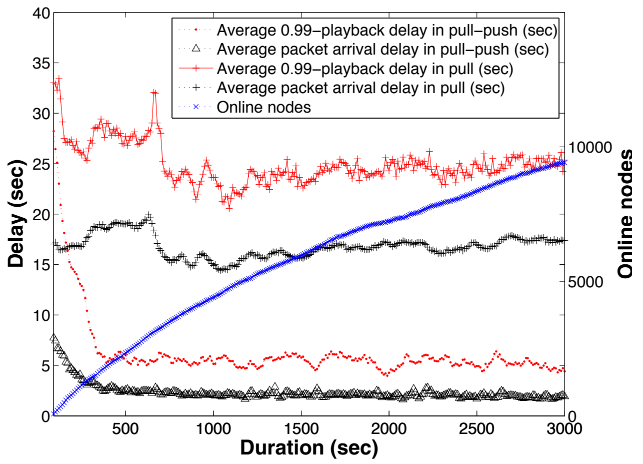


Fig. 25. Average playback delay and packet arrival delay of pull-push protocol and pull-based protocol driven by real P2P streaming system user trace

Fig. 23 shows the impact of the neighbor count in pull-push hybrid protocol and the result of pull-based protocol is also illustrated for comparison. The capacity supply ratio used is 1.2. As shown, the deliverable rate keeps nearly one even if the neighbor count is 60. And in most cases, the average upload rate of pull-push hybrid protocol is much lower than the pull-based protocol, which implies the smaller overhead of pull-push hybrid protocol.

Fig. 26 shows the impact of server bandwidth and group size. With the purpose of comparison, we also plot the result of pull-based protocol here. Capacity supply ratio is set to 1.2. We note that pull-push hybrid protocol can get a higher delivery ratio under the same server bandwidth compared to the pull-based protocol. The pull-push hybrid protocol can achieve nearly 100% quality when the server bandwidth is 500kbps, i.e., 1.6 times streaming rate. So we see that pull-push hybrid protocol is superior to pull-based protocol in reducing server bandwidth. This is because, firstly, pushing packets from server is more stable than competitive packets pulling, which increases the probability that a unique streaming packet is sent out from the server; secondly, most of the packets are pushed in a very low delay which depresses the possibility that a packet gets beyond the window after several hops.

In Fig. 24 and Fig. 25, we investigate the performance of pull-push hybrid protocol under dynamic environment. As in Section III-B3, the simulation is driven by real deployed P2P streaming system traces. We use request interval 1000ms and request window size 10 seconds. And the capacity supply ratio is 1.2. In Fig. 24, we see that the delivery ratio remains perfectly above 0.97 all the time and nearly 1 in most time. The fraction of pushed packets is above 80% in most time. Fig. 25 shows the delay performance of the two protocols. The average packet arrival delay of pull-push hybrid protocol is around 2.5 seconds much smaller than that of pull-based protocol (i.e. around 17 seconds). Meanwhile, the 0.99-playback delay of pull-push hybrid protocol rapidly decreases from 30 seconds to below 10 seconds within the first 200 seconds and then keeps around 6.5 seconds in most time. This is because the repeated request and retransmission is eliminated and most packets are relayed directly in our pull-push hybrid protocol. However, the pull-based protocol has a playback delay of 24

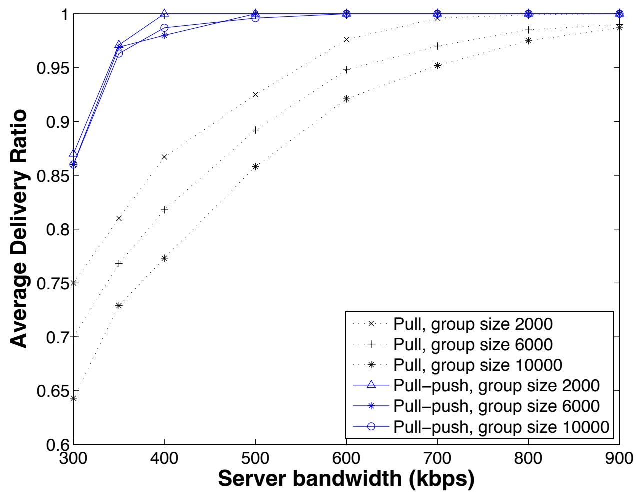


Fig. 26. Average delivery ratio with respect to the server bandwidth, group size in pull-push hybrid protocol

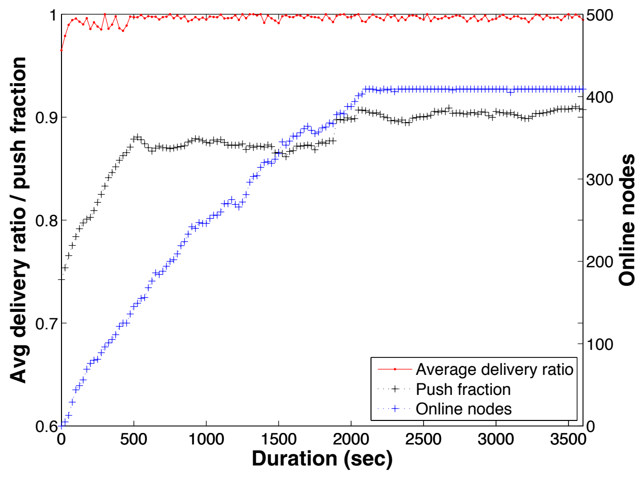


Fig. 27. PlanetLab experiment with 409 nodes. Average delivery ratio and push fraction of pull-push protocol

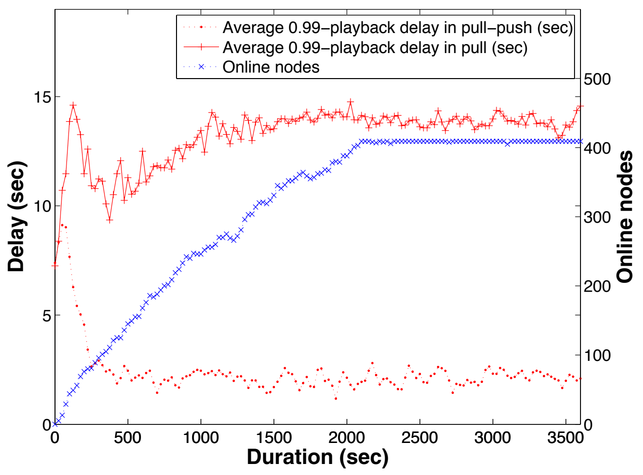


Fig. 28. PlanetLab experiment with 409 nodes. Comparison of average playback delay and packet arrival delay

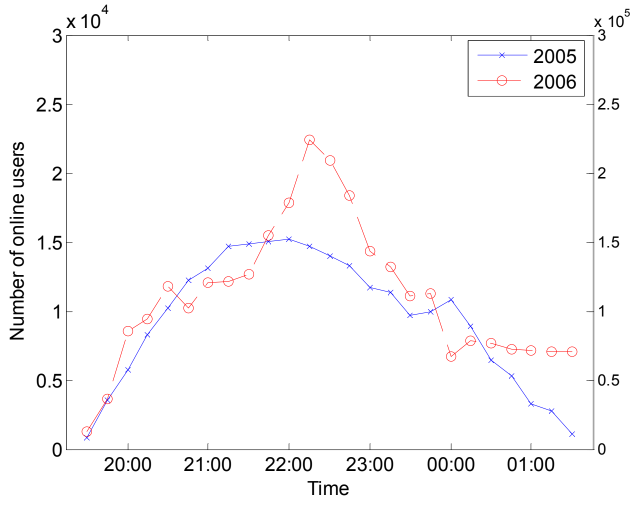


Fig. 29. Concurrent online users on the Chinese New Year's Eve in 2005 and 2006

seconds. The results show that the pull-push hybrid protocol also has excellent performance under dynamic environment.

C. Evaluation on PlanetLab

We also conduct the experiment on PlanetLab for pull-push hybrid protocol. The main configuration of the experiment is the same as in Subsection III-C. We also use the trace from real deployed P2P streaming system to drive the node behavior as described in Subsection III-C.

As shown in Fig. 27, the delivery ratio of pull-push hybrid protocol is very close to 1 all the time. And push fraction is near 90% which is a little larger than that in simulation because the group size in PlanetLab experiment is much smaller. In Fig. 28, we see that the delay of pull-push hybrid protocol is around 2 seconds much smaller than 13 seconds in pull-based protocol.

D. Deployment on Internet

The pull-push hybrid protocol is fully implemented in our real-deployed system - GridMedia. This system was adopted by CCTV to live broadcast TV program since Jan. 2005. And particularly, GridMedia system helped CCTV to broadcast Spring Festival (Chinese New Year) Gala Show of Channel CCTV1 in 2005 and 2006 through global Internet, serving more than 500,000 and 1,800,000 person times from tens of countries all over the world respectively. More excitingly, GridMedia supported 15,239 and ***224,453*** concurrent users in a 300kbps streaming rate by only one server in the two years, both of which are the records of the number of concurrent users supported by one server till then. The concurrent online users in 2005 and 2006 are shown in Fig. 29. Furthermore, Fig. 30 shows the average delivery ratio captured by the server (since the population is very large, only part of the users are in the statistic of quality).

We briefly introduce some deployment details here. For the source of the streaming session, we deployed a Darwin Streaming Server [30] as the source node. The streaming server was equipped with Dual Core Xeon Server, 1 GB RAM and RedHat Enterprise Server 3.1. It was located in the China Telecom Internet Data Center with 1Gbps upload capacity. The TV signals were compressed and encoded in CCTV station and real-time pushed to the streaming server through a 2Mbps fiber connection. The encoding and streaming rate

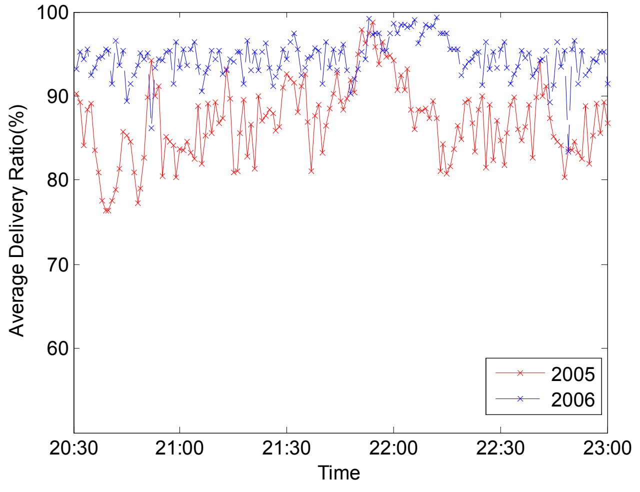


Fig. 30. Quality of streaming service in terms of average delivery ratio in 2005 and 2006

was 300Kbps. In the first deployment in 2005, we set the maximum connections to the streaming server up to 200, indicating that only 200 peers could obtain contents directly from the server. The workload on the streaming server was thus roughly 60Mbps. In the second deployment in 2006, in face of the fact that such kind of applications got more popular in Internet, we increased the maximum connections up to 800, that is, in total at most 240Mbps outgoing bandwidth at server side. Considering that there were 224,453 users online at most and the streaming rate was 300kbps, the total peak server bandwidth demand would be over 64Gbps if traditional Client/Server or CDN-based systems were used. And in fact, in other words, our system supported $224,453/800 \approx 270$ times of users more than Client/Server-based system.

V. Discussion

A. Insights obtained

In fact, much effort has been made on maximizing throughput of peer-to-peer streaming in both systematical and theoretical work [10], [7], [6]. A lot of approaches leverage optimization theory and network coding to increase the network throughput in P2P streaming. In this paper, we found that, in peer-to-peer streaming, optimal throughput can be nearly achieved by a simple protocol in most typical scenarios. In fact, in our proposed pull-push hybrid protocol, the delivery ratio can achieve 1, as long as the capacity supply is higher by only 10% than the minimum bandwidth demand. Considering 3% header overhead and 2%$\sim$3% control overhead, the deliverable rate actually achieves nearly optimal throughput. So making full utilization of the upload capacities of all peers in P2P streaming are not difficult as expected before.

B. Limitations of simple pull-based and pull-push hybrid protocol

Although pull-based and pull-push hybrid protocol have strong capability to nearly fully utilize the peer upload capacity, they have their own limitations.

First, we imagine an ideal peer-to-peer streaming protocol here, which have the following features: a) it can satisfy all the peers watching 100% quality video even if the total upload capacity supply is equal to the minimum total bandwidth demand; b) the server bandwidth it consumes is only one time streaming rate so that any home user can broadcast their video with quality as high as they can; c) when the total upload capacity of all peers are more abundant (for instance, there are a fraction of Ethernet users), it can minimize the ISP core network and cross ISP traffic without affecting the peers watching 100% quality video, d) can achieve minimum playback delay under peer churn, with optimized peer selection and streaming scheduling and e) when the capacity is insufficient, it can make a fair traffic allocation among peers with respect to the contribution of each peer.

In this paper, we see that simple pull-based and pull-push hybrid protocol can nearly fulfill Feature a). However, for Feature b), although simple pull-push hybrid protocol can support a group of 10,000 peers to get nearly 100% delivery ratio consuming server bandwidth of 1.6 times the streaming rate without peer churn, how to only use server bandwidth of one time streaming rate is a very challenging topic. For Feature c), due to the random peer selection and random packet scheduling, both the simple pull-based and pull-push hybrid protocol can not deal with this. In fact, "owing to" the powerful capability of pull-based protocol to utilize available bandwidth, the simple peer selection and random scheduling usually cause tremendous core network and cross-ISP traffic. Furthermore, the cost of most of this traffic is paid by ISP itself. This will result in that ISPs do not have incentives to update their infrastructures to promote their network capacity, since once they do this, their core network will be congested by more unprofitable traffic. Thus, how to reduce the peer-to-peer traffic in core network and iter-ISP links to achieve a win-win model is a big challenging topic. For Feature d), although pull-push hybrid protocol has much lower playback delay than pull-based protocol, we can still further reduce it. For instance, we can do optimized peer selection with considering the peer online time and peer upload capacity comprehensively. Some work has been done towards optimal peer selection (such as [31]), and we believe that it can be combined with our pull-push hybrid protocol in our future study. For Feature e), these two simple protocols, of course, can not handle this issue. We note that [24] designs elegant contribution-aware mechanism for multiple-tree based protocol. However, designing contribution-aware pull-based and pull-push hybrid protocol should be further studied.

C. Future research directions

- Minimizing ISP core network and cross-ISP traffic: in our paper, we do not discuss the link stress of such protocols due to topology mismatch between overlay and underlay. Since ISPs worry much about the huge P2P traffic, measuring and evaluating the impact pull and pull-push protocols on link stress and try to use existing proxy cache and locality-aware peer selection technique to relieve the link stress are challenging topics for future study. Recent work such as [32] has started to practically solve this problem in general P2P applications.
- Server bandwidth reduction: in typical parameter settings, pull-based and pull-push hybrid protocol need server bandwidth of 3 and 1.6 times streaming rate respectively. How to let home users broadcast video with quality as

high as they can is a very challenging. Can we not only use only one-time-streaming-rate server bandwidth but also deliver high quality video to all users? Can network coding be leveraged to handle this?

- Network coding: in peer-to-peer live streaming, leveraging network coding may not help much to improve the the throughput. So other benefits of network coding should be paid more attention, such as data randomization and persistence, resistance to high churn rate and diminishing startup waiting delay. Recent work Echelon [33] and Lava [34] has made this wise step.
- Real Internet environment: in this paper, we have the assumptions that the bandwidth bottleneck is only at the end-host and all the nodes can connect with each other, as most previous work is based on. However, on the real Internet, the connections across the peer link bridge between ISPs have low rate, and NAT/firewall usually prevent end-host from connecting with each other. Does the near optimality of pull-based and pull-push hybrid protocol still hold under real Internet environment?

VI. Conclusion

In this paper, we first give our observation that, besides simplicity and robustness as recognized before,the pull-based protocol is nearly optimal in terms of bandwidth utilization and system throughput when the server bandwidth is above several times of the streaming rate. We also give mathematical analysis to give an insightful understanding of the pull-based protocol and explain why pull-based protocol has this characteristic. Moreover, we point out that high overhead and large delay of pull-based protocol can be easily overcome by our proposed pull-push hybrid protocol. We highlight the proposed protocol which can not only approach the nearly optimal throughput but also can reach far lower playback delay and much smaller overhead than the pull-based protocol. Especially in static environment, the near optimality in peer bandwidth utilization can be achieved when the server bandwidth is 1.6 times of the streaming rate. Based on our findings, we further give some discussions for pull/pull-push hybrid protocol and the future work in P2P streaming.

References

[1] "Pplive," 2007. [Online]. Available: http://www.pplive.com/

[2] "Zattoo," 2007. [Online]. Available: http://zattoo.com/

[3] "Gridmedia: http://www.gridmedia.com.cn/," 2006.

[4] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "Coolstreaming/donet: A data-driven overlay network for efficent media streaming," in *IEEE INFOCOM 2005*, Miami, US, Mar. 2005.

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

[6] N. Garg, R. Khandekar, K. Kunal, and V. Pandit, "Bandwidth maximization in multicasting," in *Annual European Symposium on Algorithms*, Budapest, Hungary.

[7] Y. Cui, B. Li, and K. Nahrstedt, "On achieving optimized capacity utilization in application overlahy networks with multiple competing sessions," in *ACM Symposium on Parallel Algorithms and Architectures 2004*, Barcelona, Spain, June 2004.

[8] J. Li, P. A. Chou, and C. Zhang, "Mutualcast: An efficient mechanism for one-to-many content distribution," in *ACM SIGCOMM Asia Workshop 2004*, Beijing, China, 2004.

[9] T. Small, B. Li, and B. Liang, "Outreach: Peer-to-peer topology construction towards minimized server bandwidth costs," *IEEE JSAC Special Issue on Peer-to-Peer Communications and Applications*, Jan. 2007.

[10] Z. Li, B. Li, D. Jiang, and L. C. Lau, "On achieving optimal throughput with network coding," in *IEEE INFOCOM 2005*, 2005.

[11] Y. Chu, S. Rao, and H. Zhang, "A case for end system multicast," in *ACM Sigmetrics 2000*, Santa Clara, California, USA, June 2000.

[12] B. Cohen, "Bittorrent: http://bitconjuer.com."

[13] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *IPTPS 2005*, Conell, US, Feb. 2005.

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

[15] S. Annapureddy, S. Guha, and et al, "Exploring vod in p2p swarming systems," in *IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 2007.

[16] "Qqlive," 2006. [Online]. Available: http://tv.qq.com/

[17] R. Kumar, Y. Liu, and K. Ross, "Stochastic fluid theory for p2p streaming systems," in *IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 2007.

[18] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *IEEE INFOCOM 2006*, Barcelona, Spain, Apr. 2006.

[19] A. Legout, G. UrvoyKeller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *Internet Measurement Conference 2006*, Brazil, 2006.

[20] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *ACM SIGCOMM 2004*, Portland, OR, USA, Aug. 2004.

[21] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 2007.

[22] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of p2p live streaming services," in *IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 2007.

[23] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singhr, "Splitstream: high-bandwidth multicast in cooperative environments," in *Symposium on Operating Systems Principles (SOSP) 2003*, New York, USA, Oct. 2003.

[24] Y.-W. Sung, M. Bishop, and S. Rao, "Enabling contribution awareness in an overlay broadcasting system," in *ACM SIGCOMM 2006*, Pisa, Italy, 2006.

[25] V. Agarwal and R. Rejaie, "Adaptive multi-source streaming in heterogeneous peer-to-peer networks," in *Multimedia Computing and Networking 2005 (MMCN)*, San Jose, CA, USA, Jan. 2005.

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

[27] Y. Chu, A. Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "Early experience with an internet broadcast system based on overlay multicast," in *USENIX 2004*, Anchorage, Alaska, USA, May 2007.

[28] M. Zhang, L. Zhao, Y. Tang, J. Luo, and S. Yang, "Large-scale live media streaming over peer-to-peer networks through global internet," in *ACM workshop on Advances in peer-to-peer multimedia streaming (P2PMMS)*, Singapore, Nov. 2005, pp. 21–28.

[29] "Planetlab," 2007. [Online]. Available: http://www.planet-lab.org/

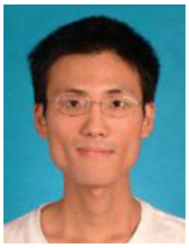
[30] "Darwin streaming server," 2007. [Online]. Available: http://developer.apple.com/opensource/server/streaming/

[31] J. Jiang and K. Nahrstedt, "Randpeer: Membership management for qos sensitive peer-to-peer applications," in *IEEE INFOCOM 2006*, Barcelona, Spain, Apr. 2006.

[32] G. Shen, Y. Wang, Y. Xiong, B. Y. Zhao, and Z.-L. Zhang, "Hptp: Relieving the tension between isps and p2p," in *IPTPS 2007*, Bellevue, Washington, USA, Feb. 2007.

[33] C. Wu and B. Li, "Echelon: Peer-to-peer network diagnosis with network coding," in *IEEE IWQoS 2006*, New Haven, CT, USA, 2006.

[34] M. Wang and B. Li, "Lava: A reality check of network coding in peer-to-peer live streaming," in *IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 2007.



Meng ZHANG (M'05) received the BEng degree in Computer Science and Technology from Tsinghua University, Beijing, China, in 2004. He is currently a PhD candidate in the Department of Computer Science and Technology in Tsinghua University. His research interests are in all areas in multimedia networking, particularly in QoS issue of peer-to-peer streaming, peer-to-peer Video-on-Demand, multimedia streaming on overlay networks, etc. And he has published over 10 papers in multimedia networking. Besides, he is the key designer and developer of GridMedia - one of the earliest very large scale peer-to-peer live streaming system in the world.

Qian ZHANG (M'00-SM'04) received the B.S., M.S., and Ph.D. degrees from Wuhan University, China, in 1994, 1996, and 1999, respectively, all in computer science.

Dr. Zhang joined Hong Kong University of Science and Technology in Sept. 2005 as an Associate Professor. Before that, she was in Microsoft Research, Asia, Beijing, China, from July 1999, where she was the research manager of the Wireless and Networking Group. Dr. Zhang has published about 150 refereed papers in international leading journals and key conferences in the areas of wireless/Internet multimedia networking, wireless communications and networking, and overlay networking. She is the inventor of about 30 pending patents. Her current research interests are in the areas of wireless communications, IP networking, multimedia, P2P overlay, and wireless security. She also participated many activities in the IETF ROHC (Robust Header Compression) WG group for TCP/IP header compression.

Dr. Zhang is the Associate Editor for IEEE Transactions on Wireless Communications, IEEE Transactions on Multimedia, IEEE Transactions on Vehicular Technologies, Computer Networks and Computer Communications. She also served as Guest Editor for IEEE JSAC, IEEE Wireless Communications, Computer Networks, and ACM/Springer MONET. Dr. Zhang has been involved in organization committee for many important IEEE conferences, including ICC, Globecom, WCNC, Infocom, etc. Dr. Zhang has received TR 100 (MIT Technology Review) world's top young innovator award. She also received the Best Asia Pacific (AP) Young Researcher Award elected by IEEE Communication Society in year 2004. She received the Best Paper Award in Multimedia Technical Committee (MMTC) of IEEE Communication Society and Best Paper Award in QShine 2006. She received the Oversea Young Investigator Award from the National Natural Science Foundation of China (NSFC) in 2006. Dr. Zhang is the vice-chair of Multimedia Communication Technical Committee of the IEEE Communications Society.

Lifeng SUN (M'05). Dr. Lifeng Sun was born in Xi'an, China, 1972. He received his B.S. Degree and Ph.D. Degrees in System Engineering in 1995 and 2000 separately from National University of Defense Technology, Changsha, Hunan, China. Dr. Sun's professional interests lie in the broad area of ad hoc network streaming, peer-to-peer video streaming, interactive multi-view video, distributed video coding, he has published over 50 papers in the above domain. He was on the Post Doctor research of the Department of Computer Science and Technology at Tsinghua University from 2001 to2003; He is now on the Faculty of the Department of Computer Science and Technology at Tsinghua University. Dr. Sun is the secretary-general of Multimedia Committee of China Computer Federation, Vice Director of Tsinghua-Microsoft Media and Network Key Lab of Ministry of Education (MOE). Dr. Sun serviced as TPC member of Pacific-Rim Conference on Multimedia (PCM) 2005, Pacific-Rim Conference on Multimedia (PCM) 2006, IEEE International MultiMedia Modelling Conference (MMM) 2007, IEEE International MultiMedia Modelling Conference (MMM) 2006, and reviewed papers for IEEE ICME 2006, IEEE CSVT special issue on multiview video coding.

Shiqiang YANG is a chief professor at Tsinghua University with department of Computer Science and Technology. He r eceived the B.S, M.S in Computer Science from Tsinghua University , Beijing , China in 1977 and 1983, respectively.

From 1980 to 1992, he worked as an assistant professor at Tsinghua University, Beijing, China. From 1992 to 1994, he visited City University of Hong Kong and was a research assistant at the Department of Manufacture Engineering. As the associate header of Department of Computer Science and Technology at Tsinghua University since 1994, he served as the associate professor from 1994 to 1999 and then as the professor since 1999.

He is currently the President of Multimedia Committee of China Computer Federation, Beijing, China and the co-director of Microsoft-Tsinghua Multimedia Joint Lab, Tsinghua University, Beijing, China. His research interests mainly include multimedia application, video procession, streaming media and embedded multimedia. In recent three years, he has published more than 60 papers in international conferences and journals, as well as more than 10 proposals within MPEG and AVS standardization.