BUILDING APPLICATIONS ON PEER-TO-PEER NETWORKS: A CASE STUDY ON MEDIA STREAMING

ABSTRACT

To overcome the limited bandwidth of streaming servers, Content Distribution Networks (CDNs) are deployed on the edge of the Internet. A large number of such servers have to be installed to make the whole system scalable, making CDN a very expensive way to distribute media. Building media streaming system on a peer-to-peer (P2P) network is found to be an efficient and inexpensive way to alleviate the traffic load of CDN servers. Unlike CDN servers, peers are heterogeneous in their resource contributions (storage capacity, network bandwidth, etc.) and duration of their commitments to the community. As a result, P2P streaming is often performed in a many-to-one manner and should be prepared to peer failures. In this research, we investigated the effects of resource limitations and *come-and-go* behaviors of peers on application development and performance. Specifically, we present the design and empirical evaluation of a media streaming protocol that takes those constraints into account.

Submission to track **Information Technology & Systems**

1. INTRODUCTION

Peer-to-peer (P2P) systems are distributed systems in which all nodes have identical functionalities [1, 2, 3]. There have been significant research interests on P2P systems during the past three to four years. The concept of P2P differs from conventional client/server paradigms in the absence of centralized entities that manage the synchronization and communication among end-point systems. Comparing to their client/server counterparts, P2P systems are more resistant to faults and imbalanced loads. What makes them more attractive is that they are more cost-effective due to their ability to harness the resources of a large number of nodes. File sharing has been the major application of P2P and thus the focus of most P2P research efforts. Attention to P2P was initially drawn by the success of two file sharing services – Napster and Gnutella. Following them are a number of other applications such as instant messaging [4], media streaming [5], content distribution [6], and online gaming [7]. In this paper, we focus on one specific application of P2P: media streaming.

Unlike traditional Internet file service, distribution of continuous media requires data to be delivered in a streaming manner. That is, each part of the media has to be transmitted and arrive at the client side before a deadline. This brings tremendous challenges to the design of media servers in best-effort delivery networks, such as the Internet. Generally, the untimely transmission of media fragments becomes worse when more requests are received and the communication channel gets congested. The current solution to this problem is to replicate media data on different sites on the Internet to avoid any individual locations being swamped by requests. One popular practice for media replication is to deploy Content Distribution Networks (CDN) on the edge of the Internet [8]. Each of these CDN servers holds a copy of all media files and is responsible for serving requests from its neighboring area (Figure 1A).

However, it is extremely costly to maintain a CDN as it requires massive CPU power, storage space and network bandwidth to serve a large community of users. Among the computing resources in a CDN-based media service, the output bandwidth was found to be the bottlenecking factor [9]. For example, a server with a T3 line

connected to it has a maximum bandwidth of 45Mbps. When the media streams being serviced are MPEG-1 videos with an average bitrate of 1.5Mbps, only 30 concurrent sessions can be supported. The peak number of sessions that need to be serviced simultaneously could be in the order of thousands or even higher. To make the whole system scalable in terms of number of concurrent requests it can handle without swamping the servers, a large number of such servers have to be deployed.

Recent research efforts [10, 11] have shown that it is promising to build a high-capacity and cost-effective media streaming system on the basis of a P2P network. The basic idea is to let client machines that acquired a (whole or partial) media object serve subsequent requests to that media from other clients (Figure 1B). In other words, the participating client nodes are used to help relieve the service load of servers in the CDN-based scheme by contributing their network bandwidth. The most important difference between P2P and centralized media distribution strategy is that the total capacity of a P2P system grows when the content it manages becomes more popular. Previous work [11] demonstrated that the capacity of a P2P media streaming system grows exponentially under reasonable assumptions and the servers can completely hand the streaming tasks over to the clients after some time.

With the attractive prospect of applying P2P in media streaming, there are also a lot of challenges we have to address. First of all, peers are heterogeneous in the storage capacity and out-bound bandwidth they can contribute. Any attempt to build a P2P streaming infrastructure has to take this into account. A particular problem is how to avoid swamping a streaming client (*supplying peer*) since for most peers the resource contribution is limited. For example, most of the peers in Napster and Gnutella are connected to the Internet by ADSL or dial-up modems [12]. This means more than one peers are needed to satisfy the bandwidth requirements of a single streaming session. The problems of synchronization and bandwidth assignment among multiple supplying peers are not trivial. Secondly, peers are heterogeneous in the duration of their commitment to the community. Unlike dedicated servers, a peer may fail or leave the service at any time. This come-and-go behavior makes a P2P system intrinsically dynamic. How to minimize the effects of this behavior on the Quality-of-Service (QoS) during media streaming is critical in the design of a P2P media streaming service as the latter is always the primary

concern in such services. To face this challenge, a streaming protocol that effectively assigns load to various peers and gracefully recovers from failure is a must. In this paper, we discuss the design, implementation, and evaluation of such a protocol.

The rest of this paper is organized as the following: Section 2 is composed as a sketch of our P2P streaming architecture; Section 3 presents the streaming protocol in detail; In Section 4, the results of extensive simulations on the properties of our streaming protocol are discussed; Section 5 concludes this paper with conclusions and future works.

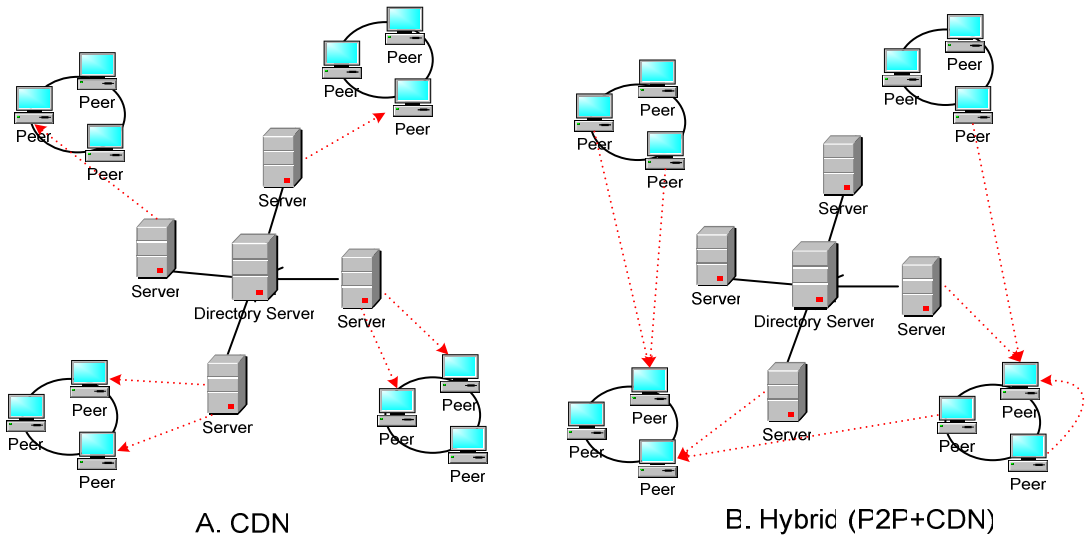

Figure 1. *A comparison between the system architectures of the CDN (A) and hybrid (B) media streaming service. Red lines represent the direction of streaming data transmission.*

2. THE P2P MEDIA STREAMING MODEL

A hybrid media streaming system that combines CDN servers and a P2P network is proposed and analyzed in [13] and [11]. Our streaming architecture is based on this hybrid system (Figure 1B). There are two reasons why we do not adopt to a pure P2P architecture for media streaming: 1. We need servers to place the original media files and boot up the service; 2. As streaming bandwidth is always the bottleneck resource, we could achieve high system capacity by shifting the streaming load to the peers. In this system, each participating peer announces the bandwidth and storage it can provide and we assume these announcements are accurate.

2.1. System Components

There are three major components in the system: Directory Server, CDN servers, and the community of participating peers.

Peers. All client machines that participate in the service. In a streaming session, the peer that receives media data from others is called a *requesting peer* and the one that sends out data a *supplying peer*. We may have multiple supplying peers in one session due to the limited bandwidth contribution of peers.

Directory server. The first step for a user request is to locate media objects of interest. This typically involves keyword or content-based search that can be accomplished by the database server in a very efficient way. The role of the directory server is to maintain status information of media objects and peers to support update and search. Specifically, our directory server has to keep the paired information in the format of (*Peer*, *Object*) where *Peer* is a client that's willing to serve as a supplying peer and *Object* is the ID of a media file *Peer* holds.

CDN Servers. It holds a copy of all media files and is responsible for streaming during the initial stage of the service startup and afterwards when the requested media cannot be serviced through the P2P network.

2.2. System Operation

When a peer requests a media file, it follows the steps listed below:

1. Check in local cache, if it is there, then we are done, otherwise go to step 2;
2. Send query to the directory server. The latter searches its directory (database) for peers that have the file and return those peers as possible supply peers to the requesting peer;
3. The requesting peer then probe all those possible supply peers in parallel about the requested file and get the delay and bandwidth of all those peers. Among those peers, choose the peers with the highest bandwidth. If no enough bandwidth is available, then check whether CDN server has enough spare bandwidth. If the total bandwidth available is enough, then start streaming. Otherwise the request is rejected;

4. If the request is successful, then send a message containing the updated status of the peers to the directory service. The directory server then updates its directory based on these messages.

3. PEER-TO-PEER STREAMING PROTOCOL

In this section, we describe a P2P video streaming protocol, named *Redundant Multi-Channel Streaming Protocol* (RMCSP), which realizes seamless media transmission under variations of link status (e.g. bandwidth, delay, and loss rate). The failure or leave of a subset of supplying peers can be viewed as an extreme case of such variations. The main idea is to use receiver-driven control messages to synchronize and adjust the packet delivery from all supplying peers. The coordination-related decisions are made exclusively by the data receiver based on the information received from all senders. Unlike other receiver-driven protocols such as RTP/RTCP, there are more than one supplying peers in a streaming session. Furthermore, we can only make very loose assumptions on the reliability and commitment duration of the data senders. An important assumption in RMCSP design is that spare channels (peers) can generally be found in addition to the supplying peers. This assumption is proved reasonable by previous studies on media streaming [11]. Taking advantage of this redundancy, RMCSP uses these spare channels to complement the temporary bandwidth loss and choose replacements for failed peers. The RMCSP consists of three components: a transport protocol, a dynamic connection migration algorithm, and a packet distribution algorithm.

3.1. Session Initiation

By querying the index server, a requesting peer gets a list L of all peers that are able to deliver the media requested. In addition, every client knows the set C of available CDN servers when they boot up the media service. To initiate the streaming session, the requesting peer probes all peers in L and sorts them by increasing round-trip time (RTT) of the probe messages. The first n peers in the sorted list that satisfies the following condition will be used as supplying peers for the streaming:

$$\sum_{i=1}^{n} B(Li) \geq R$$

where $B(L_i)$ is the posted bandwidth contribution of peer L_i and R is the playback rate of the video. If the sum of all $B(L_i)$ in L is smaller than R, we need to find available CDN servers from C to serve the video such that the total bandwidth is no less than R. If all the CDN servers are not available at this moment, the request is rejected.

When the initial set of supplying peers (and/or CDN servers) is determined, streaming begins. Meanwhile, the requesting peer sends a message containing the set of supplying peers (as an ordered list) to the index server, which, upon receiving the message, changes the bandwidth contribution of these peers in the index. The bandwidth contribution of all but the last peer/CDN server in the set now becomes 0. An observation here is that we always try to minimize the size of the set of supplying peers by taking as much bandwidth as we can from each peer. By this, a supplying peer only serves very few (1 or 2) requests and a requesting peer receives data from a small number of peers. The advantage of having a small set of peers in a streaming session is that the overhead of streaming control is low. After the initiation, the original peer set L is divided into two disjoint subsets: supplying peers (denoted as *LS*), and the rest of the peers (denoted as *LB*). During the streaming period, the requesting peer keeps a record of these sets as well as the CDN server set C. In case of channel failure or service degradation of any element in *LS*, replacement(s) will be chosen from the other 2 sets, among which *LB* has higher priority than C.

3.2. Transport Protocol

As the dominant transport protocol for a majority of Internet activities, TCP, however, is ill-fitted for real-time applications. To deal with the best-effort delivery of the IP Internet, TCP rapidly (exponentially) decreases the sending rate in response to individual packet loss, which results in dramatic changes of the latency of packets. The sending rate is doubled by TCP when the loss rate is found to be increasing. In contrast to the fuzzy back-off and speed-up mechanisms in TCP, the TCP-Friendly Rate Control (TFRC) algorithm proposed in [14] used a strategy called *Equation-based Congestion Control*, where the sending rate changes only as response to a group of loss events. Thus, TFRC

achieves smoother change of sending rates than TCP. However, the sending rates in TFRC may still fluctuate due to the "slow-start" recovery mechanism. In RMCSP, we use a modified TFRC that minimizes the change of sending rates by using the maximum available bandwidth. This requires, however, the sender to estimate the current bandwidth and adjust its sending rate. Our protocol will follow the idea of slow back-off and bandwidth estimation in TFRC to avoid dramatic fluctuations in the sending frequency. One of the well-known equations [15] for bandwidth estimation is given as the following equation

$$T = \frac{s}{r\sqrt{\frac{2p}{3}} + t_{RTO}\left(3\sqrt{\frac{3p}{8}}\right)p(1+32p^2)}$$

where r is estimated round-trip time, p the loss event rate, T the estimated bandwidth, t_{RTO} the TCP timeout, and s the packet size. This gives an upper bound of allocated sending rate. Therefore, in our model, the maximum sending rate of a specific sender i at any time point t is $S(i, t) = \min(T(i, t), B(i))$ considering the bandwidth contribution B of each participating peer should not be exceeded.

3.3. Connection Migration

As described in 3.2, the sending rate of each supplying peer is determined on the receiver side by calculating useable bandwidth using the aforementioned equation. When the measured bandwidth T of a link changes, RMCSP will respond by changing the sending rate according to the newly detected parameter. Instead of responding to any bandwidth change, we only record changes that are greater than a threshold value w, which can be a small fraction (5-10%) of $S(i, t)$. When the measured bandwidth is w or more lower than the current sending rate ($S(i, t) - T(i, t) > w$) for consecutive γ times, the sending rate has to be changed to $T(i, t)$. The same thing happens when RMCSP detects bandwidth that is higher than the sending rate by w for γ times except that it doesn't allow the sending rate to go beyond the publicly posted bandwidth contribution $B(i)$.

The decrease of bandwidth of one or more streaming channels may end up with a total sending rate lower than R. Under this situation, we need to find more bandwidth from other peers to keep the streaming session sustainable. If the lost bandwidth is small,

we may increase the sending rate of some well-connected (to the receiver) supplying peers beyond their bandwidth contribution. This kind of 'stealing' is not very harmful to a peer as long as the stolen bandwidth if a small fraction of its bandwidth contribution. An alternative is to find more bandwidth from peers other than the ones already in service (e.g. those in set *LB* or set *C*). In RMCSP, instead of putting all the workload on the server side, the requesting peer will find a new supplying peer, resynchronize the sending rates and initialize a new connection for data transferring. A basic algorithm (*spare channel replacement*) for service migration would contain the following steps:

1. Send a new request to the directory server and obtain a list of potential supplying peers in return;
2. Probe every peer in the list and sort them by response time;
3. Choose peer(s) that are able to provide at least the bandwidth lost from the original session and initialize connection between them and the requesting peer;
4. Recalculate the sending rates and streaming recovers.

Suppose the RTTs of messages passing in step 1 and step 2 are ρ_1 and ρ_2, and TFRC initialization costs ρ_3, the recovery time for a connection migration is $\rho_1+\rho_2+\rho_3$. During this time, the streaming application only receives partial data thus the media playback will suffer from degraded QoS.

As an improvement to the above algorithm, the requesting peer can keep the state information of redundant peers (*spare channels*) when normal streaming is in progress. Note the list *LB* (Section 3.1.) is exactly the set of *spare channels*. When the streaming session is initiated, the requesting peer keeps probing not only all supplying peers (set *LS*) but k peers in *LB* as well. The number k is chosen such that $k \leq |LB|$ and it is of high probability for the k peers being probed to provide enough capacity in case of bandwidth loss. If *LB* is an empty set, at least one member in *C* (CDN servers) is probed instead. To reduce the overhead of probing spare channels, we can set the probing frequency to be lower than that for monitoring supplying peers. In particular, the probing cycle for spare channels is set to $k\varphi$ where φ is the probing cycle for supplying peers. Now the improved algorithm can start directly from step 4 in case of recovery since it has all information needed. This leads to zero recovery time, which means the switch over to new data senders is achieved transparently to the users. We will show (Section 4) that the overhead

of spare channel monitoring is small. One thing to point out is that the status for peers in the initial *LB* set may be outdated. Therefore, the spare channel monitoring mechanism in the requesting peer may go over step 1 through step 3 to update *LB* while streaming is underway.

The departure of supplying peer(s) during a streaming session imposes no more difficulties than bandwidth loss. There are two types of departure: graceful leave and site failure. In a graceful leave, the peer explicitly sends out a *LEAVE* message to the index server as well as all peers that have any connection (streaming or spare channel probing) with it. Upon receiving the *LEAVE* message from a supplying peer, the requesting peer simply switches to other peer(s). Meanwhile, the directory server will also update the status of the leaving peer. In case of peer failure, no *LEAVE* messages will be sent thus the failure has to be detected by a health monitoring emchanism. For a peer i at time t, when the following condition holds true for consecutive γ times, we believe peer i has failed.

$$S(i, t) - T(i, t) > w'$$

where w' is a significant fraction of $S(i, t)$. Note it may take some time for the bandwidth estimation T to decrease to zero when failure happens. A remedy to this is to declare a failure when we found the instant bandwidth estimation is decreased to some dramatic low level for γ times. After detecting a failure, the requesting peer does the same thing as if it was a graceful leave. Besides, it is the responsibility of the requesting peer to inform the directory server of the failure.

To make the server migration algorithm more efficient and less complex, we may consider CDN servers as the only candidates for spare channels. When any bandwidth compensation is needed by a streaming session, the requesting peer can directly go to the CDN server(s). The regular bandwidth probing activities between the requesting peer and the backup peers can be spared assuming high availability of CDN servers. If all the CDN servers are fully loaded under such circumstances, the streaming task has to be terminated. In practice, the probability of such occurrences is small.

3.4. Packet Distribution

The problem of packet distribution is how to allocate senders for the group of packets that should be delivered within a synchronization period knowing the sending rate S and latency r of each supplying peer. For RMCSP, we will use a solution that is very close to the packet partition algorithm proposed in [16]. The basic idea of this packet partition algorithm is: given the S and r values of all supplying peers as well as the set of media data packets within a synchronization period, each supplying peer locally determines the sender of each packet and sends those it is responsible for. The control message contains the S and r values of all peers calculated by the requesting peer and a synchronization serial number. Upon receiving the control message, each supplying peer computes the time difference A between a packet's estimated sending time and decoding time for every packet and every peer. The peer that maximizes A is chosen to be the sender of that packet. The calculation of A is done using the following equation:

$$A(j, p) = T_j - P_{j, p}\, \sigma(p) + 2r(p)$$

In the above formula, $A(j, p)$ is the time difference we are trying to calculate for packet j if sent by peer p, T_j is the time packet j should arrive at the client side for decoding, $P_{j, p}$ is the number of packets sent so far (within the same synchronization period) by p, $\sigma(p)$ is the sending interval of p, and $r(p)$ is the estimated delay between the receiver and p. As we can see, everything needed for calculating $A(j, p)$ is known to the supplying peers: $r(p)$ and $\sigma(p)$, which is the inverse of sending rate S, are explicitly included in the control packet; T_j can be obtained from the media stream itself and $P_{j, p}$ is generated on the fly. There is no ambiguity in calculating $A(j, p)$ so that for each packet, a unanimous decision can be made among all sending peers.

The above packet allocation algorithm consumes excessive CPU cycles at the supplying peers when the synchronization frequency becomes high. Each peer has to do the same floating point computations within every synchronization period. As an improvement, we move the burden of packet distribution to the receiver side in RMCSP. When the receiver computes the packet distribution locally, it packs in the control message to each supplying peer a list of sequence numbers of the packets needed to be sent from that peer. Then the above equation can be used to calculate $A(j, p)$ for packet j at peer p. However, there is one problem if we compute $A(j, p)$ on the receiver side: for a non-CBR media stream, the deadline of each packet (T_j) is unknown to the receiver

because copies of the media object are only kept in the data senders. One solution would be to download from the CDN server a digest of the media file containing the deadlines of all packets before streaming starts. The size of the digest can be very small. For video streams, packets within a frame share the same deadline thus the digest size is only related to the frame rate of the media. Consider a typical MPEG-1 video with average bitrate of 800Kbps and frame rate of 30fps, the overhead for transmitting the digest file is only 30×24/800K = 0.088% assuming 24 bits are used to represent a packet deadline entry with a time instance, a starting packet number, and an ending packet number.

4. EXPERIMENTAL RESULTS

We implemented RMCSP in the *NS2* network simulator [17]. The major purpose of the simulation experiments is to test the feasibility of RMCSP, especially its robustness under peer failures and makes suggestions on how to tune the parameters.

The network configuration of the RMCSP session simulated is shown in Figure 2. In this simple topology, the requesting peer is connected to its local router R via a link with total bandwidth of 1.5Mbps, symbolizing a typical DSL user. Three supplying peers, Peer 1, 2, and 3 are shown to be linked to R by 3 different network connections that share no common congestion link among them. In the graph, these 3 connections are abstracted into virtual links with parameters different than the one between R and the receiver (Link 1). The bandwidth of the 3 virtual links is also set to 1.5Mbps each. The delays of Link 2, 3, and 4 are configured to be one order of magnitude higher than Link 1. The arrows in Figure 2 represent the direction of media data transmission.

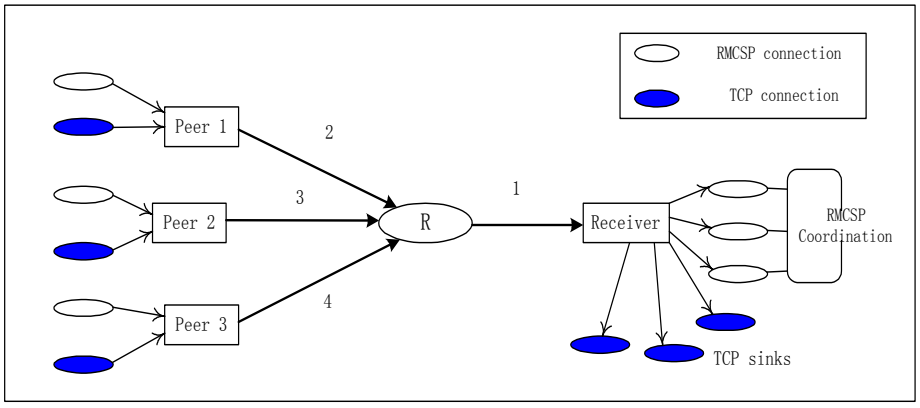

Figure 2. *Simulation setup for peer-to-peer streaming*

We consider a 300-second CBR video with bitrate 720Kbps. Peer 1 and Peer 2 are selected as the original supplying peers for the streaming of this video with Peer 1 contributing 480Kbps and Peer 2 providing 240Kbps. Peer 3 is a backup peer with an initial sending rate of 8Kbps, reflecting the basic probing activities in a 1-second interval. Arbitrary TCP connections are also added to simulate cross traffic.

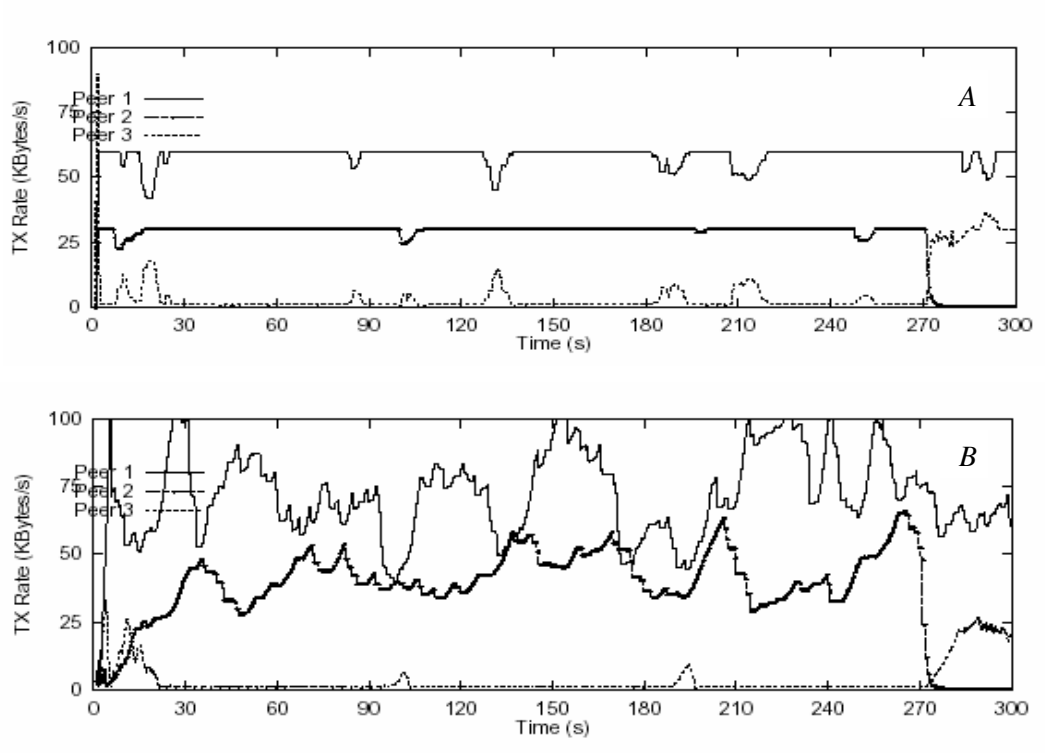

Figure 3. *Transmission rate of peers over time using RMCSP(A) and TFRC(B)*

At time 0s, the TCP agents started transferring data. At time 2s, the streaming of the video started with initial sending rates 480Kbps and 240Kbps, respectively. These numbers are also the upper bound of sending rates for these peers. For the backup peer Peer 3, no data was transmitted besides the packets for probing bandwidth availability. The requesting peer sends out control packets for synchronization every 500 milliseconds. The packet size for RMCSP is 1000 bytes. Algorithm used for computing loss rate is Weighted Average Loss Interval (WALI) method mentioned in [14]. Both Link 1 and Link 2 were applied a uniform random loss model with loss rate set to 0.01. Therefore, the loss pattern of the RMCSP connection between Peer 1 and Receiver is different from that between Peer2 and the Receiver, with the latter less prone to packet drop. Peer

failure happens at time 270s, when Peer 2 was abruptly turned down. As a control experiment, we also tested the same scenario using TFRC. The metrics we used for evaluation are the smoothness of the sending rates and data loss ratios.

4.1. Overall Evaluation

Figure 3A demonstrates the measured sending rates of the above streaming task using RMCSP. For most of the time, the sending rates of both supplying peers are on a smooth line that represents their predefined bandwidth contribution. Under such circumstances, streaming is going perfectly with the highest level of QoS guarantee. There are a few valleys for the plotted lines of both Peer 1 and Peer 2. Such decreases of sending rates are mostly caused by consecutive packet drops. In turn, the smaller loss interval was translated into bandwidth availability that is lower than the target sending frequency. Another observation is that the valleys in the curve for Peer 1 are generally deeper than those for Peer 2. This can be explained by higher link stress (480Kbps vs. 240Kbps) and loss frequency in Peer 1. According to Figure 3, our protocol recovers from dramatic changes of sending rates caused by degraded network conditions in the order of seconds. Most of the valleys are less than 5 seconds wide and none is wider than 10 seconds.

RMCSP was also found to be sensitive to bandwidth underflow and peer failures. Almost all the lost bandwidth due to lowered sending rates is compensated for by the contribution of the backup server, Peer 3. The latency for the bandwidth compensation is also low. Basically, the time between the detection and compensation of bandwidth underflow is directly related to the synchronization frequency. Actually, the latency for bandwidth compensation is exactly one synchronization cycle. The extreme case of bandwidth loss, peer failure, is also very well handled by our protocol. In less than 3 seconds, the estimated bandwidth of the channel where failure (Peer 2) occurred goes down to near zero. Within the same time scale, Peer 3 started to take over the streaming load of Peer 2. Fluctuations can be observed on the transmission rate of Peer 3 when it first takes over.

The same simulation was run using TFRC as the underlying transport protocol (Fig 3B). The measured sending rates show extremely high variances. This is undesirable for our streaming architecture in 2 aspects: first of all, TFRC frequently uses more

bandwidth than the contribution value posted by supplying peers. For example, both Peer 1 and Peer are using more bandwidth than it is willing to contribute for a majority of the total time. On the other hand, frequent change of bandwidth allocation among supplying peers increased the complexity of media synchronization. If we compare the estimated

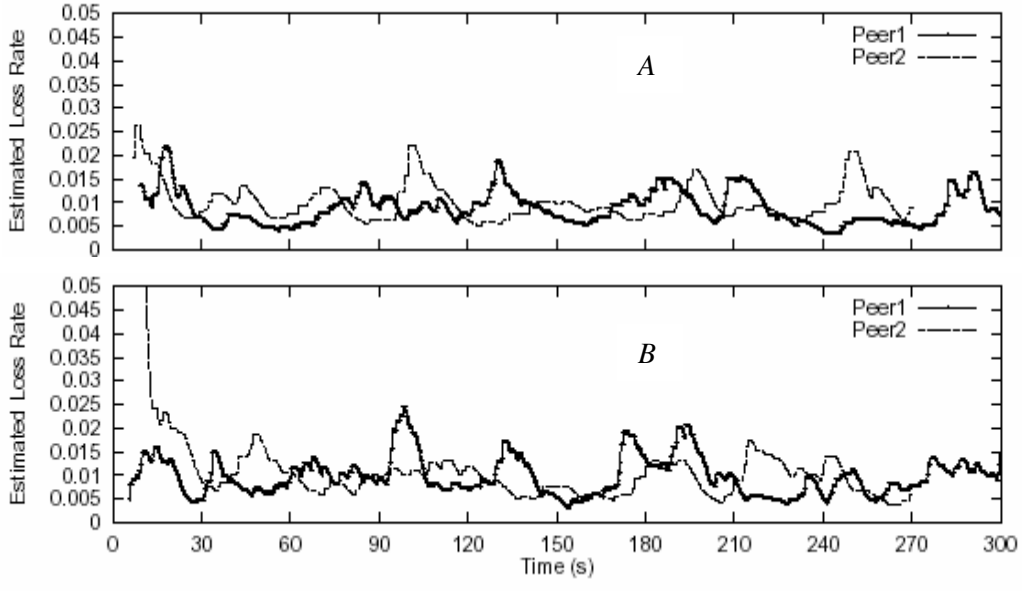

Figure 4. *Estimated loss rate for RMCSP (A) and TFRC (B)*

loss event rate of both protocols, little difference (Figure 4) can be found. The only exception is at the early stage of the streaming, when TFRC obtained an excessively high loss rate value for Peer 2. This could be an unfortunate drop of a number of packets in a small time interval. The number of consecutive drops doesn't have to be big to make this happen because no historical data can be used to lower the averaged loss event rate at that moment. However, this abnormality recovers to a health level very quickly. The similarity of estimated loss rate between RMCSP and TFRC is as expected since RMCSP only modifies TFRC by capping the sending rate and getting rid of "slow start". For the same reason, the estimated bandwidth for both protocols are also similar to each other (data not shown). It is just RMCSP makes better use of bandwidth than TFRC. On the other hand, RMCSP's aggressive use of channel capacity is constrained within a boundary thus the competitive TCP connections are not starved. From this point of view, we may read RMCSP as a loose bandwidth reservation mechanism on the outbound link

of the data senders (complete bandwidth reservation requires collaboration of intermediate routers). Of course, the assumption here is that the sender keeps no other connections that are more aggressive than RMCSP in consuming its outbound bandwidth, such as UDP connections. In a modern computer for general use, this assumption is realistic.

4.2. Effects of Link Load

Link stress also has significant effects on the performance of our streaming protocol. The sending rates for streaming videos with different bitrates are plotted in Figure 5. Figure 10A shows the results for a video with bitrate only 480Kbps with Peer 1 contributing 320Kbps and Peer 2 160Kbps. The sending rates are almost straight lines for all 3 peers. This means there is always enough bandwidth for video streaming when the streaming job requires only less than 1/3 of the total bandwidth of a DSL connection. In this case, the link stress of streaming for Peer 1 is 320K/1.5M ≈ 20% and 160K/1.5M ≈ 10% for Peer 2. Figure 5B demonstrated another extreme scenario using a video of bitrate 960Kbps. Again, the contribution from Peer 1 is twice the contribution from Peer 2 (640Kbps vs. 320Kbps). Under such circumstance, all the peers showed degraded performance in terms of smoothness of sending rates. For Peer 1, which has the highest link stress, the sending rate is more or less oscillating around the 400Kbps (50KBps). This means such high bandwidth contribution (640K/1.5M ≈ 43%) for participating peers can hardly be achieved in media streaming. For Peer 2, the base line for fluctuations is about 240kbps (30KBps). Without surprise, the backup peer (Peer 3) had to compensate for the insufficient bandwidth for most of the time. The level of contribution from Peer 3 was close to that from Peer 2. In Figure 7, the bandwidth contribution for Peer 1 was 480Kbps (60KBps), or 1/3 of the total bandwidth of the link, and the curve for Peer 1 also showed some fluctuations. This implies that 480Kbps is probably the upper bound for a meaningful bandwidth contribution in a 1.5Mbps DSL line. Even when the contribution goes up to 520Kbps (65KBps), dramatic changes of sending rates can be observed (data not shown here). Therefore, we regard 1/3 of total bandwidth as a rule of thumb for peer contribution.

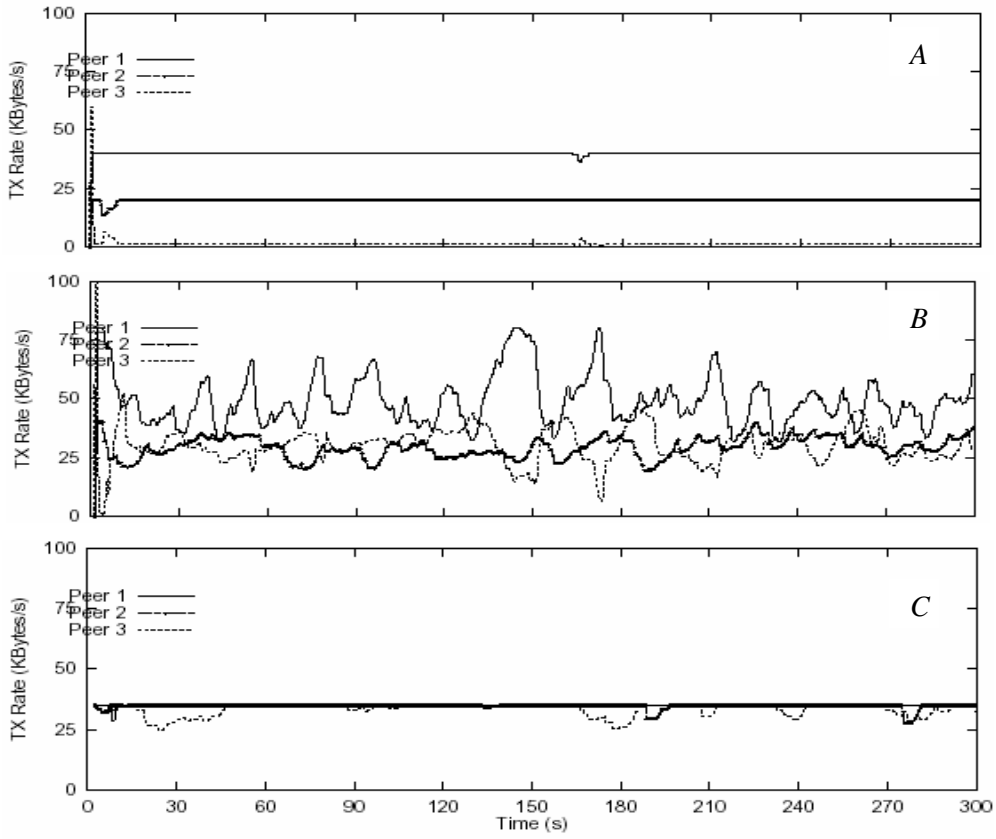

Figure 5. *Transmission rate under different total streaming bitrate. A.Total bitrate of 480Kbps with 320Kbps from Peer1and 160Kbps from Peer2; B. Total bitrate of 960Kbps with 640Kbps from Peer 1 and 320Kbps from Peer 2. C. Total bitrate of 1120Kbps with 4 supplying peers each contributing 280Kbps.*

We also found that it is the load of individual links that really matters. In Figure 5C, we showed the measurements of a video with 1.12Mbps bitrate. The difference from other scenarios is that 4 supplying peers are used (only 3 are plotted in the graph), each of them contributing the same amount of bandwidth (280Kbps). It is easy to see that the streaming performance is much better than the previous scenarios where there are certain links in heavy stress. We can conclude that to some extent, seeking bandwidth from more peers is better than aggressively using more bandwidth from fewer peers. However, more simulations indicate that when the bitrate reaches 1.2Mbps the performance is inevitably

inferior to lower bitrate streaming no matter how many supplying peers to use. Another factor to consider is the overhead of maintaining all concurrent RMCSP connections.

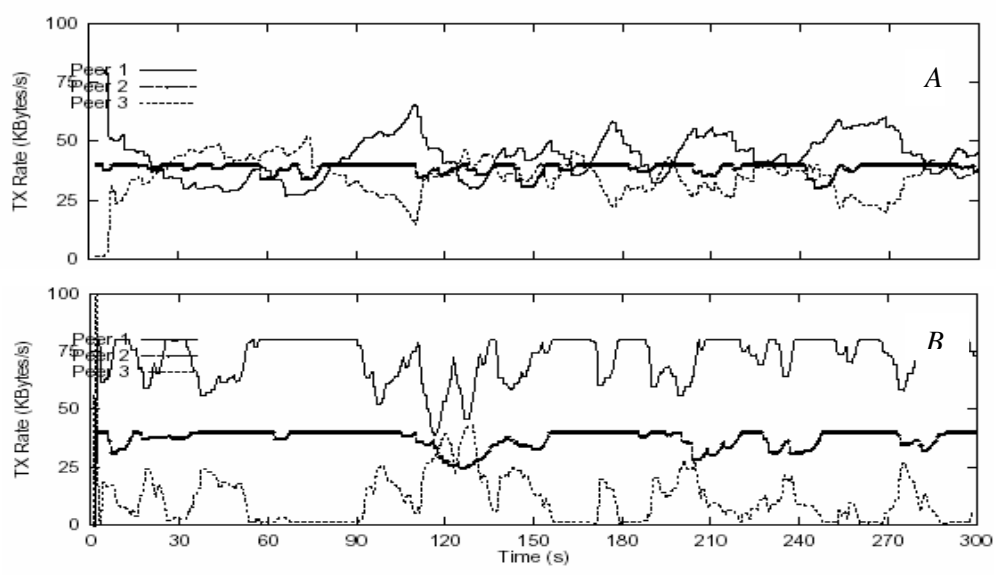

Figure 6. *Transmission rate under different situations. Same scenario as in Figure 5B with total bitrate of 960Kbps. A. Without link loss. B. Without TCP competition.*

In analyzing the impact of link stress on sending rates, we considered two possible factors: the competition with TCP and the inability of DMCSP to accurately estimate bandwidth over a lossy link. In order to know which factor has a bigger impact, we simulated unrealistic situations where the above two factors can be isolated (Figure 6). We follow the same scenario of that studied in Figure 5B where Peer 1 has very high link stress. The metric we used is again the sending rates of peers. In Figure 6A, we simulated the situation assuming no losses on all four links. As we may see, the bandwidth usage for Peer 1 was still under heavy fluctuations and the baseline of sending rates for all three peers are very close. As compared to that of Peer 1, the sending rate for Peer 2 is more stable. In another experiment (Figure 6B), all TCP connections are taken away when the streaming is underway over lossy links. Without competition of TCPs, the bandwidth usage of RMCSP is closer to the expected style: Peer 1 can reach its bandwidth contribution for at least half of the streaming time. And the valleys of Peer 1's sending rate curve are properly filled by Peer 3. We can clearly conclude, by comparing Figure 6A and Figure 6B, that co-existing TCP connections is the controlling factor that inhibits

bandwidth over-provisioning of supplying peers. Of course, this doesn't mean RMCSP is perfect in modeling the network status. Actually, as an ongoing work, we are rethinking the bandwidth estimation algorithm of RMCSP.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduce a new application built upon P2P networks and CDN servers: a hybrid peer-to-peer streaming system. We discuss the difficulties in application design on P2P networks as peers' resource contribution is limited and peers may leave/fail without prior notice. A peer-to-peer streaming protocol, RMCSP, is presented to address the above challenges. The design of RMCSP focuses on the capability of the receiver to coordinate concurrent data transport links to keep smooth streaming, as well as a redundancy-aware bandwidth compensation mechanism to handle dynamic network status including peer failures. The idea of achieving smooth sending rates on the transport level of RMCSP was adapted from the TFRC protocol, which is modified to satisfy more strict requirements for data streaming. Simulation results show that RMCSP performs fairly well under realistic network conditions. Various factors related to the performance of RMCSP are also analyzed and discussed. Based on the analysis, we also present some guidelines on the selection of peer contributions and streaming organization for the purpose of achieving high-quality streaming by RMCSP.

Future work involves more in-depth study of the dynamics of the RMCSP protocol, especially revisiting the estimation algorithms for loss event and bandwidth. The simulation for the streaming protocol made some assumptions that may not be realistic in a real-world system. For example, most of the popular video compression formats ended up with VBR video streams. The conversion of VBR into near-CBR streams via smoothing algorithms has to be considered. Other issues on the video streaming applications such as video coding, error correction, and QoS control are also important.

REFERENCE

[1] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. *HP Labs Technical Report HPL-2002-57*.

[2] A. Rowstron and P. Druschel, Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In Proceedings of *ACM/IFIP Conference on Distributed Systems Platforms (Middleware)*, November 2001, pp.329--350.

[3] J. Crowcroft and I. Pratt. Peer to Peer: Peering Into the Future. LNCS 2345, *Proceedings of the IFIP-TC6 Networks 2002 Conference*, Pisa, May 2002.

[4] Luis Felipe Cabrera, Michael B. Jones, and Marvin Theimer. Herald: Achieving a Global Event Notification Service. In Proceedings of *Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII),* Elmau, Germany., May 2001.

[5] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava. On Peer-to-Peer Media Streaming. In *Proceedings of IEEE International Conference on Distributed Computing Systems (ICDCS),* July 2002. pp. 363-371.

[6] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh. SplitStream: High-Bandwidth Multicast in A Cooperative Environment. In Proceedings of *International Workshop on Peer-to-Peer Systems (IPTPS),* Berkeley, CA, February 2003.

[7] http://www.nrg.cs.uoregon.edu/p2pGaming/index.html

[8] A. Biliris, C. Cranor, F. Douglis, M. Rabinovich, S. Sibal, O, Spastcheck, and W. Sturm. CDN Brokering. In *Proceedings of International Workshop on Web Caching and Content Distribution (WCW)*, June 2001.

[9] V. N. Padmanabhan and K. Sripanidkulchai. The Case for Cooperative Networking. *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002. pp.178-190.

[10] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, Promise: Peer-to-Peer Media Streaming Using Collectcast, In Proceedings of *ACM Multimedia 2003 (ACM SIGMM)*, Berkeley, CA, November 2003, pp. 45-54.

[11] Y. Tu, J. Sun, and S. Prabhakar. Performance Analysis of a Hybrid Media Streaming System. In *Proceedings of SPIE/ACM Conf. on Multimedia Computing and Networking (MMCN)*, San Jose, CA, January 2004, pp. 69-82.

[12] S. Saroiu, P. Gummadi, S. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *ACM Multimedia Systems Journal*. 9(2):170-184, 2003.

[13] D. Xu, H-K. Chai, C. Rosenberg, and S. Kulkarni. Analysis of a Hybrid Architecture for Cost-Effective Streaming Media Distribution. In *Proceedings of SPIE/ACM Conf. on Multimedia Computing and Networking (MMCN)*, Santa Clara, CA, January 2003.

[14] Sally Floyd, Mark Handley, Jitendra Padhye, and Joerg Widmer. Equation-Based Congestion Control for Unicast Applications**.** In Proceedings of *ACM SIGCOMM*. August 2000. pp. 57-69.

[15] J. Padhye, V. Firoiu, D. Towsley, J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In Proceedings of *ACM SIGCOMM*, Vancouver, CA, September 1998. pp. 303-314.

[16] T. Nguyen and A. Zakhor, Multiple Sender Distributed Video Streaming, In *IEEE Transactions on Multimedia* 6(2):315-326, 2004.

[17] S. McCanne and S. Floyd. ns--Network Simulator. http://www-mash.cs.berkeley.edu/ns/.