CONSTRUCTING ROBUST AND RESILIENT FRAMEWORK FOR COOPERATIVE VIDEO STREAMING

Shi Lu and Michael R. Lyu

View Lab, Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong SAR
{slu, lyu}@cse.cuhk.edu.hk

ABSTRACT

Peer-to-peer based streaming has been a promising solution for large-scale video broadcasting over the Internet. In a peer-to-peer video streaming framework, peers cooperate with each other for content distribution, so that the burden of the central server is greatly alleviated. Moreover, the peer-to-peer overlay is highly scalable to support a very large number of users. However, to support video streaming service, some strict performance issues need to be addressed, e.g., reliability, resilience and robustness to network dynamics. In this paper, we investigate several goals that a peer overlay should achieve in order to support good-quality video streaming. We then describe our work on organizing peers into such a robust and resilient framework. We have implemented a fully functional video broadcasting system based on the proposed peer-to-peer infrastructure. The prototype system has been successfully deployed and tested upon Planet-lab with encouraging experimental results.

1. INTRODUCTION

Videos have become pervasive nowadays. Consequently, video streaming service has been one of the most attractive applications on today's Internet. However, although there has been a rapid advancement of network bandwidth and high-capacity storage devices, distributing digital media contents simultaneously to a large number of users is still a challenge for the content providers. Traditional client-server infrastructure is not scalable and only a limited number of users can be supported within the server capacity. To overcome this problem, Content Delivery Network (CDN) deploys a large number of servers at the edge of the Internet, and the users' requests are redirected to the nearest server. However, CDN solution demands very high expense on purchasing and deploying the dedicated servers.

On the other hand, peer-to-peer based file sharing systems have been quite successful. Currently there are several popular file-sharing systems based on peer-to-peer infrastructure, like Napster [1], Gnutella [2], BitTorrent [3], DC++ [4], eMule [5], and each of them has attracted a sizable user group. In comparison with the traditional client-server infrastructure, peer-to-peer based content distribution systems transfer data in a de-centralized way. They are able to support a large group of users with very little, if any, dedicated resources. Moreover, since their system capacity grow dynamically with the number of participants, the more peers there are in the system, the better the overall performance will be. These scalable overlay networks show good potentials to support real-time video streaming services.

Enlightened by the peer-to-peer file sharing systems, several cooperative streaming schemes have been proposed. [6, 7] construct a tree-shaped overlay for content relay. Random mesh overlay are employed by [8, 9, 10]. Moreover, some systems have been publicly released [10, 11, 12].

Video streaming, in comparison with file sharing, demands much more in performance. We identify several issues the peer-to-peer framework should fulfill to support live video streaming. First, the framework should be resilient enough to quickly adapt to the changes of network conditions, e.g., latency and bandwidth, and still ensure a satisfying in-bound bandwidth on each peer. Second, since peers are free to leave and join, the framework should be robust in ensuring its performance and keeping the whole framework from being broken into several components by some peers' leaving. Third, for a specific peer, it should be able to assign the data transfer workload to its neighbors in a balanced and fair manner. In the literature, few work has investigated all the above issues in the context of peer-to-peer video streaming. In this paper, we describe our work in organizing the peers into such a resilient and robust framework to meet the performance requirements and support the video streaming service.

The paper is organized as follows: In Section 2 we give an overview of our peer-to-peer video streaming framework. In Section 3 we describe the scheduling algorithm and optimization in our framework. In Section 4 we show some

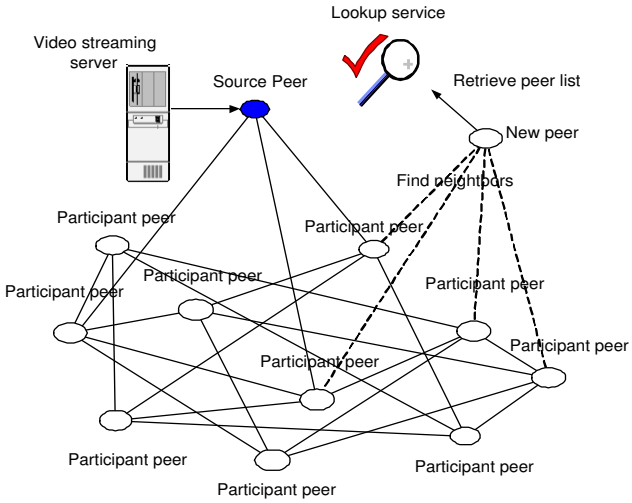

Fig. 1. Overview of the p2p streaming infrastructure

experiment results. Finally, in Section 5 we make conclusion and discuss our future work.

2. THE PEER TO PEER STREAMING FRAMEWORK

Here we give an overview of our peer-to-peer video streaming framework, shown in Fig. 1. The system is composed of a video streaming server, a source peer, a lookup service and the participant peers. The source peer takes the streaming content from the video streaming server and feeds the content into the peer overlay. The lookup service helps new peers to find other peers; it can be either centralized or distributed. To join the video streaming framework, a new peer first contacts the lookup service, obtains a list of online peers, then establishes connections with other peers and starts receiving the content. Each of the participant peers maintains a local media buffer, in which some media data is stored for supporting the local media player and further shared to other peers in the broadcast framework.

3. SCHEDULING AND OPTIMIZATION IN THE FRAMEWORK

3.1. Receiver-driven data transfer

To adapt to the network dynamics, especially the changes in network bandwidth and delay, data content transfer in our framework is in a receiver-driven mode; in other words, it is the receiver to monitor the data distribution and the performance of network links then determine the amount and direction of the data flow.

The data content transferring process is driven by data distribution information, which is similar to [10]. For a data packet, its availability state can be available or unavailable. Then the data status of a peer can be represented by a data status bit-map (DSB), where one bit corresponds to one data packet. The size of the DSB is quite small, so every exchanged message in the framework is attached with the most current DSB of the sender peer. Consequently, each peer is always keeping up with the most up-to-date DSB of its neighbor peers. The data flow direction of one connection is two-way, determined by data distribution. Consider peer A and peer B, for example. When peer A finds that peer B has data that it does not have, it may send a request to peer B for that data packet; after some time maybe peer B finds that peer A has some data that it does not have, then their sender-receiver roles will be changed.

3.2. Bandwidth/delay monitoring

A peer maintains several network links $\{l_1...l_n\}$ to its neighbors $NB = \{nb_1...nb_n\}$. For each link l_i, the peer monitors three performance parameters: current data in rate r^i_{in}, current data out rate r^i_{out} and the current request delay d_i. r^i_{in} and r^i_{out} are measured as the mean data rate over a time interval. d_i is the time interval between the last sent data request and the arrival of the corresponding data block.

3.3. Data transfer scheduling

Consider the mesh formed by the participant peers as a graph, the optimal way to distribute the data packets is of course by sending packets along the minimum-spanning tree of the peer overlay. However, the topology of the graph is always changing for peer join or leave. Moreover, the network link bandwidth and delay are volatile too. In such a volatile circumstance, obtaining a minimum spanning tree of the peer overlay at every moment is infeasible. In our implementation, we adopt an efficient real-time content scheduling algorithm running distributively on each peer to determine the transfer path of the data packets. The algorithm quickly generates a semi-optimal solution based on the peer's current local network status.

For each network link, when a data request is sent, before the corresponding data packet arrives, the network link will be marked as "busy" and no more data request will be sent along the network link. When a data packet arrives, the network link state becomes "ready", and then further data request can be issued to the network link. The scheduling step takes place when a data packet arrives at the peer and the corresponding network link becomes "ready".

In our framework, each data packet can be represented as $P_x = \{D_x, dsb_x, req_x\}$, where D_x is the video data chunk, dsb_x is the current data status bitmap, and req_x is the data request. We put the three fields together as one block to

reduce very small packets transfer so that the system can be more efficient. To support smooth playback without jittering, all data packets should be fetched in the peer's buffer before it is supposed to be played. The target of the content scheduling step is to ensure that as many data packets as possible can be retrieved from the neighbor peers before their playback deadline. Also, the data transfer load should be assigned to multiple links in a balanced manner.

For each peer, we design the data transfer scheduling algorithm as follows:

Algorithm 1 Data scheduling algorithm on each peer

Input: the network links $\{l_1...l_n\}$, the corresponding delay time $\{d_1...d_n\}$, the DSB set $\{dsb_1...dsb_n\}$, local DSB dsb_{local}
while true **do**
  When packet $P_x = \{D_x, dsb'_i, req_x\}$ arrives from link l_i:
  Mark l_i as "ready";
  $dsb_i = dsb'_i$;
  Save D_x, update dsb_{local};
  From dsb_{local}, find the packet p_{req} whose playback waiting time $t_{req} > d_j$;
  If req_x is valid, find data block D_{req_x};
  Assemble the reply packet $P_{resp} = \{D_{req_x}, dsb_{local}, req\}$;
  Send p_{resp} via link l_i; Mark l_i as "busy"
  for all l_j such that l_j state is "ready" **do**
    Calculate the weighted recent delay d_j;
    From dsb_{local}, find the unavailable packet P_{req} whose playback waiting time $t_{req} > d_j$, and is available from l_j;
    Send $P_{req} = \{null, dsb_{local}, req\}$ to link l_j;
    Mark l_j as "busy";
  end for
end while

3.4. Resilience and robustness

In order to adapt to the changes of the network conditions, we set a minimal connection number nc_{min}, which is the minimal number of network link that a peer should maintain. When nc_{min} is larger, the peer becomes more robust to neighbors' leaving. In our implementation, nc_{min} is set to four. Since the receiver-driven content transfer scheduling algorithm schedules the data request according to the recent state of the network link, the performance can well adapt to the changes of the connection parameters. Moreover, the data transfer load can be assigned to the connections according to their status, thus making a good load balancing among multiple connections. Normally the network links do not need to work at the maximum rate, and high stress can be avoided.

In real Internet environment, the probability that a lot of peers simultaneously leave the overlay is very small. When a peer leaves, its neighbors will be notified. Since each affected peer has maintained at least nc_{min} network links, it can still get content from the remaining network links. If the network link number is less than nc_{min}, the peer will try to establish new network links with others until nc_{min} is reached again. This behavior ensures the system robustness against peer leaving.

3.5. Overlay integrity

When there are articulation point in that graph, there is a possible threat for the whole overlay be broken into pieces if the peer on that articulation point exits the streaming framework, as shown in Fig. 2.

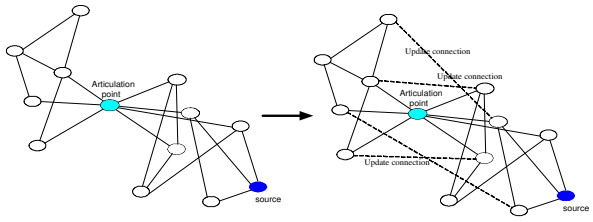

Fig. 2. Articulation point elimination

Each peer p_i keeps measuring its distance dis^i to the source peer. The distance is defined as the minimum number of hops between itself and the source peer. To calculate its own dis_i, for peer p_i, we have

$$dis^i = \min_{nb_j \in NB_i} dis^j + 1,$$

where NB_i is the set of neighbor peers of p_i. The dis value of the source peer is set to 0. We also note that the overhead to calculate dis_i is well distributed to all participant peers. Each packet between peers will be attached with the distance value of the sender peer. We can see that the computation overhead for each peer is very small.

From time to time, each peer contacts the lookup service for a new peer list, and then tries to establish new network links with those peers who have capacity for more network links and whose dis value is small. This behavior brings in two merits. First, the radius of the overlay will be within control since all peers are trying to minimize its distance to the source peer. Second, from a global point of view, the whole framework keeps updating and repairing itself so that the situation that the framework may break into several disconnected component can be avoided.

4. EVALUATION

We have implemented a peer-to-peer streaming prototype system based on the paradigm described in the previous section. The system is deployed on Planet-lab [13] for performance analysis. To evaluate the static performance of our

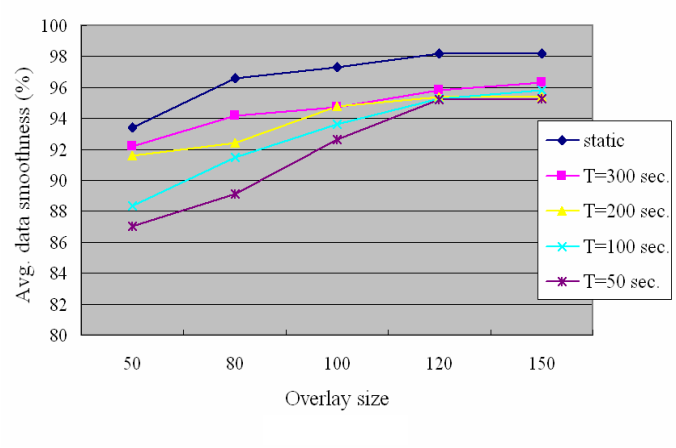

Fig. 3. Average data smoothness in different situations

framework, we measure the average data smoothness of all peers. When peers join in the overlay, they would not leave intentionally. Thus the peers form a relatively static overlay. The data smoothness a peer experiences during a time period is defined as the ratio of the video data packets successfully received before its playback time during the time period. We run the experiment with different overlay sizes of (50, 80, 100, 120, 150 peers) to observe the effectiveness of our framework. The bit rate of the video stream broadcasted to the peers is 450kb/s.

To test the resilience and robustness of the system, we need to create a dynamic overlay. We perform a series of experiments with the overlay sizes employed in the previous experiment, but we let all the peers keep joining and departing the overlay from time to time. The mean sojourn time a peer stays in the overlay is exponentially distributed, and the peer will leave the broadcast and join again as a newcomer. The mean of the sojourn time T is set to 50, 100, 200 and 300 seconds respectively. Such frequent peer joining and leaving conditions result in a very dynamic overlay. We then carry out a series of experiments with the same overlay sizes as in the static performance test and obtain the average data smoothness upon this dynamic peer overlay.

Fig. 3 shows the average data smoothness under different overlay sizes and different T values. First, we can see that the average data smoothness increases with the size increase of overlay as expected. The more peers in the broadcast overlay, the better performance the peers will have. Moreover, for all T values, the average data smoothness increases with the overlay size. Second, our framework works well under dynamic network condition. Even when T is quite small, the average data smoothness is still satisfactory. When T is smaller, the network can be regarded as more volatile, which results in less data smoothness. Due to the space limit, we only show the data smoothness result. More detailed experiment results will be shown in our future publications.

5. CONCLUSION AND FUTURE WORK

In this paper, we investigate several performance issues in peer-to-peer video streaming, and specify several requirements for peer-to-peer networks in supporting continuous video streaming service. Then we propose a receiver-driven data transfer scheme based on data availability information and an efficient data transfer scheduling algorithm, which optimizes each peer's local performance and makes the whole system resilient and robust. We implement and deploy the prototype system on Planet-lab and obtain promising experimental results.

In the future, we would further investigate the possible video-on-demand applications based on the current system. Efficient distributed lookup service is another possible topic for future research.

6. ACKNOWLEDGEMENT

The work described in this paper was fully supported by the following two grants of the Hong Kong Special Administrative Region, China.(RGC Project No. CUHK 4205/04E and UGC project No. AOE/E-01/99)

7. REFERENCES

[1] Naspter. http://www.napster.com.

[2] Gnutella. http://www.gnutella.com.

[3] Bittorrent. http://www.bittorrent.com.

[4] dc++. http://dcplusplus.sourceforge.net.

[5] emule. http://www.emule-project.net.

[6] D. A. Tran, K. A. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proceedings of the IEEE INFOCOM'03*, pages 1283–1292, 2003.

[7] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of NOSSDAV'02*, pages 177–186, 2002.

[8] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end systems multicast. In *Proceedings of the ACM SIGMETRICS*, pages 1–12, 2000.

[9] M. Hefeeda, A. Habib, B. Botev, D. Y. Xu, and B. Bhargava. Promise: peer-to-peer media streaming using collectcast. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 45–54, 2003.

[10] X. Y. Zhang, J. C. Liu, and B. Li. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *Proceedings of the IEEE INFOCOM'05*, 2005.

[11] pplive. http://www.pplive.com.

[12] ppstream. http://www.ppstream.com.

[13] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: An overlay testbed for broad-coverage services. In *ACM SIGCOMM Computer Communication Review*, volume 33, pages 3–12, 2003.