A request-driven swarming scheme for P2P data streaming

Jialing Xu and Victor O.K. Li
Department of Electrical and Electronic Engineering
University of Hong Kong
Pokfulam Road, Hong Kong, China
{jlxu, vli }@eee.hku.hk

ABSTRACT
Data streaming by swarming over peer-to-peer overlay networks has attracted much attention in recent years and initially the swarming solution is based on data-driven schemes. This paper presents a new request-driven swarming scheme. The scheme offers several advantages: high efficiency in data delivery, low control overhead, and flexibility in streaming control. Some key issues on the design of the relevant scheduling and forwarding mechanisms are discussed. We simulated our proposed request-driven scheme and compared it with an existing data-driven scheme. According to the simulation results, our request-driven scheme incurs much lower control overhead while providing comparable or better efficiency in data delivery.

Categories and Subject Descriptors
C.2.1 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Architecture and Design – *Distributed networks, Network communications.*

General Terms
Design, Performance, Reliability.

Keywords
peer-to-peer, streaming, swarming, request-driven.

1. INTRODUCTION
With advances in broadband access, the Internet is increasingly being used as a new medium to provide multimedia service to the public. Multimedia service requires efficient and scalable delivery techniques for streaming data. Presently, IP Multicast is probably the first choice for this type of data delivery service due to its sophisticated design [5][7][8][9] and high performance [6]. Its deployment, however, is limited due to high cost, excessive management workload, and redeployment difficulties when network topology and user distribution change. Thus researchers pursue application-level solutions, which attempt to build a logical overlay network among cooperative nodes on IP Unicast networks.

Peer-to-peer (P2P) technology has been suggested as the ideal platform to carry out application-level streaming. Numerous P2P-based streaming systems and theoretical studies have been completed. These can be broadly classified into two categories: tree-based and mesh-based. The former adopts similar design principles as the traditional IP Multicast which builds and maintains an explicit tree-like structure. This structure, however, is mismatched with the application-level overlay environment with dynamic nodes, and may offer poor streaming performance [1]. On the other hand, in the mesh-based approach, autonomous nodes exchange data with others spontaneously and there is no prescribed structure. This is the reason why it is also called swarming. It is simple and robust.

The initial swarming design is driven by the streaming data and involves heavy control overhead, rendering it inefficient and unscalable. We propose a lightweight, flexible, request-centric P2P streaming scheme named request-driven swarming. In this scheme, requests are utilized to construct the forwarding relationship before actual data delivery and to adjust the relationship during delivery. The simulation results and comparison with existing work demonstrate that the request-driven swarming scheme can greatly reduce the control overhead and delivery latency, while enjoying high efficiency at the same time.

In this paper, challenges in realizing the request-driven swarming scheme are identified. We discuss the issues of how to use requests to guide data streaming among nodes in the system, including streaming abstraction, request definition, and data forwarding. We propose an initial design of the request format and request-forwarding mechanism, which satisfy the basic requirements of our simple request-driven prototype.

QShine 2008, July 28-31, 2008, Hong Kong, Hong Kong.
 ISBN 978-963-9799-26-4
DOI 10.4108/ICST.QSHINE2008.3937

The rest of the paper is organized as follows. After a brief introduction of related work in multicast and in data-driven swarming over P2P in Section 2, Section 3 presents the detailed design of the request-driven swarming scheme, including the motivation of the design and the system architecture of the current prototype implementation. Evaluation based on simulation experiments and comparisons with existing work are included in Section 4. Finally conclusion is given in Section 5.

2. RELATED WORK

In the past, IP Multicast has been considered the most efficient vehicle for data streaming. Recently, however, systems implementing multimedia streaming applications on P2P overlay platforms have achieved much success. These systems can be generally classified into two categories according to their organizational characteristics: tree-based and mesh-based. The tree-based schemes organize the streaming relationship among all nodes following a prescribed tree-like structure. In the mesh-based systems, autonomous nodes exchange data with others spontaneously. Our work, the request-driven streaming scheme, belongs to the second category. In this section we give a brief review of the existing P2P streaming schemes under these two categories.

2.1 Tree-based Structure

Derived from IP Multicast, the tree structure is employed in many early overlay streaming schemes [2] [10] [14]. In this type of overlay streaming, efficient algorithms for multicast tree construction and maintenance are the keys to system efficiency and robustness.

SplitStream [14] is a representative example of this type. In SplitStream, the streaming content is striped across a forest of interior-node-disjoint multicast trees that distribute the forwarding load among all participating peers [4]. Such a multi-tree based forest brings some guarantee of efficiency and failure-tolerance, but is still vulnerable to churning and needs assistance from content coding to combat topology changes of the trees. Subsequent researchers [2] [10] have addressed the issue of reliability by more complicated algorithms. However, they are still not satisfactory.

Essentially, the tree structure is mismatched with the application-level overlay environment with dynamic nodes [3]. As the autonomous overlay nodes join and leave at will, the tree-like prescribed relationship is highly vulnerable, especially for streaming applications that require high bandwidth and time-stringent delivery.

2.2 Unstructured Swarming

Swarming schemes have recently become popular since they address the issue of robustness better than the tree-based schemes. Swarming design is also known as data-driven swarming, since it is the design principle of early swarming schemes.

Coolstreaming [14] is the first significant swarming system design based on the data-driven principle. In Coolstreaming, each node periodically advertises data availability information to a set of nodes called partners[1], retrieves locally unavailable data from them and supplies available data to them. Thus, data arrival drives the dissemination of availability information, then the information triggers the request for data and finally data is delivered. The data acts as the initiator of the procedure; hence the name data-driven. Swarming system is more resilient to high rates of churn, i.e. nodes joining and leaving; hence complicated algorithms for relationship maintenance are not necessary any longer.

Data-driven swarming, however, involves large control overhead leading to low efficiency and scalability. In multimedia applications, streaming must be timely to ensure playback quality. Control information, including data availability and data request, must be exchanged following the pace of data delivery. Therefore, the intensive data volume in multimedia applications leads to intensive volume of these control messages. As a result, large control overhead is incurred in data-driven swarming schemes. To address these problems, we propose a new swarming scheme for P2P streaming, called request-driven swarming, in which request is used instead of data to guide the streaming.

3. SYSTEM DESIGN AND OPTIMIZATION

3.1 Introduction of Request-driven Streaming

We propose the request-driven swarming scheme. In comparison with data-driven swarming, our design offers three advantages: efficiency in data delivery, low overhead traffic, and flexibility in streaming control.

Figure 1 compares the processing cycles of data-driven and request-driven swarming. The data-driven scheme contains four stages and two waiting periods. First, data arrives and then its availability is advertised periodically. When other nodes receive the advertised availability information, they then periodically send out requests for these data. Finally, these requests are served by the delivery of the required data. In this process, nodes wait for the periodical advertising of availability information and the periodical requests for data before data delivery. As a result, the four process steps and two waiting periods lead to long latency.
On the contrary, the processing in the request-driven scheme includes only three stages but no notable waiting. In Figure 1, requests for streaming are sent to those nodes

[1] A partner in Coolstreaming is a node that cooperates with the given node in swarming for the same stream.

which have attached to the stream already[2]. If these requests can be accepted according to current available bandwidth,

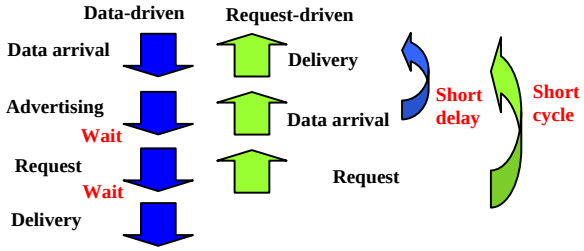


Figure 1. Comparison of data-driven and request-driven schemes

they will be recorded. When these nodes receive data, they can immediately forward them according to the previously accepted requests. Obviously, from data arrival to data delivery, only two steps and no notable waiting are required. The latency is thus reduced significantly.

Low control overhead is another merit of the request-driven scheme. In the data-driven scheme, high data volume leads to large control overhead. In this scheme, the data availability information is based on the description of data segments rather than the stream, and hard to compress. The stringent time constraint on data delivery requires that data availability information is advertised frequently, further intensifying the volume of control overhead. On the contrary, our request-driven scheme focuses on describing the stream. In most streaming application scenarios, a few attributes like rate, start time and stop time, can characterize the basic properties of a stream. Therefore describing a stream in a request by stream attributes reduces the message size and sending frequency, and thus greatly decreases the total control overhead.

The flexibility in streaming control is an extra attraction of the request-driven scheme. In the data-driven scheme, the receiver nodes request data passively based on the data availability information from partners. Therefore, the potential of streaming control improvement is quite limited. But in request-driven swarming the receiver nodes take an active role in streaming. The request mechanism endows nodes with the capability of manipulating various delivery behaviors. Moreover, feedback from these behaviors helps optimize the streaming performance.

3.2 Request Mechanism

The request mechanism is the core of our request-driven scheme and greatly impacts the scheme's performance and efficiency. A generic request for our initial scheme design and a simple scheduling algorithm are discussed here.

[2] It is assumed that these attached nodes can be discovered through existing P2P search and location mechanisms.

A request must first describe the stream. We simply describe a node's local view of a stream as a 3-tuple < *Rate*, *StartPosition*, *EndPosition*>. The variables *StartPosition* and *EndPosition* denote the starting point and the end point of the requested stream, respectively. Here, *Rate* denotes the streaming speed. These three attributes depict a basic view of a stream in almost all applications.

A time unit is not used in quantifying *StartPosition* and *EndPosition*, since it is unreasonable to assume the existence of universal time for all nodes in a distributed P2P network. Hence we suggest that the data stream be divided into sequential segments with increasing sequence IDs; and the sequence ID is used to denote the *StartPosition* and *EndPosition*.

Based on the stream description above, the request is defined as a 4-tuple <*Rate*, *StartPosition*, *EndPosition*, *Pattern*>. Here, the first three variables have the same meanings as above in depicting the requested stream. The last variable *Pattern* denotes the expected data forwarding behavior and is defined as an n-tuple $<A_1, A_2, \ldots A_n, B>$, representing modulo operation on segment sequence IDs. In modulo operation, sequence ID of data segment is the dividend, B denotes the divisor and $A_1, A_2, \ldots A_n$ are the candidate remainders for segment selection: given a sequence ID *X*, if the remainder $(X \bmod B) \in <A_1, A_2, \ldots A_n>$, this segment is requested to be forwarded.

Figure 2 presents an example of using the request definition in streaming control. First, node *B* sends a request, **<60, 101, 120, <1, 3, 5, 7, 8> >,** to node *A*, asking for the data segments with sequence ID in the range from 101 to 120. In addition, *Pattern* in the request specifies modulo operation: each segment's sequence ID is divided by 8, if the remainder is 1, 3, 5, and 7, then the segment is requested to be forwarded. Following the request, node *A* should forward segments with IDs 101, 103, 105, …, 119 to node *B*.

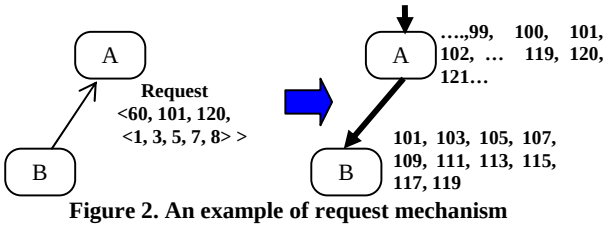


Figure 2. An example of request mechanism

In our implementation, a bitmap of a few bytes is used to represent *Pattern*. For example, the pattern in Figure 2 is implemented as a compound data structure consisting of a four-byte-length array as the bitmap, 10101010…. and a one-byte integer variable representing the divisor *B*=8, which is also the effectual bit length of the bitmap.

3.3 System Implementation

Figure 3 depicts the system diagram of our initial prototype. There are nine key modules in the architecture: four timers, three databases and two processors. Among these modules, the solid arrows indicate the data flows and the dashed ones are for control.

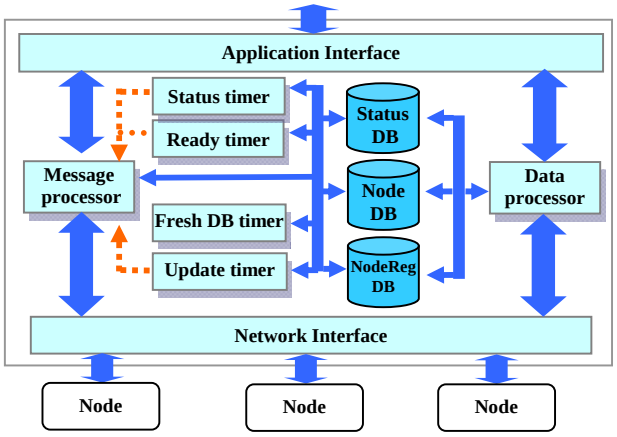


Figure 3. A system diagram for a node

The three database modules maintain the information for the normal operation of nodes. The *NodeDB* keeps statistics of upstream nodes and the connections with them. *NodeRegDB* contains information about the registered downstream nodes and the requests from them. The *StatusDB* keeps all the local information about the node itself. All the maintained information is collected from and shared with other functional modules.

The four timers are in charge of periodical routines: the *ready timer* checks the node's status and informs the upper application about the current operation condition of the overlay; the *status timer* calculates the current overly status information based on the data in *statusDB* and updates the relative values for the two processors' decision making; the *freshDB timer* periodically removes from databases the obsolete data including the records of non-responding upstream nodes and out-of-date requests; the *update timer* collects information of available nodes on the network and notifies the *message processor* to communicate with them through requests.

The *data processor* and *message processor* take charge of real-time data and message processing, respectively. In particular, the *data processor* forwards received data to other nodes and to upper application such as media player. The pattern-based forwarding function in Section 3.2 is utilized in the *data processor*. The job of the *message processor* is to guide streaming by requests. Measuring current transmission condition, making scheduling decision, and controlling streaming with requests are the core functions of the message processor.

In our initial implementation, *message processor* sends out requests at startup to the nodes which have already attached to the stream. When such a request is received, the *message processor* checks if it is possible to accept the request based on the node's current available upload bandwidth and the specified data rate in the request. If this request is acceptable, *message processor* reduces the available bandwidth by the data rate specified in the request, records the request for data forwarding and sends a reply message to the request sender about the acceptance; otherwise, if the request is unacceptable, the request sender is also informed about the failure by a reply message.

Input:
totalSegSum: total number of received segments
effecualSegSum: number of effectual segments
newAvailNodeSet[ ]: set of new available nodes on the network
upstreamNodeSet[ ]: set of upstream nodes to which requests have been sent

Parameters:
exptSegSum: expected number of received effectual segments
exptRedRatio: expected threshold of totalSegSum /effecualSegSum

Scheduling:

```
if effecualSegSum< exptSegSum then
    // not all data segments are received
    for i=0 to min(maxRequest, newAvailNodeSet.size() ) do
      // send request to avail nodes for all data segment
       requestSet.insert(newAvailNodeSet[i],
pattern(11111111,8));
    end for i;
else
   // all data segment have been received
    if (totalSegSum/ effecualSegSum> exptRedRatio) then
       // schedule to reduce the redundancy ratio
          //randomly select two different nodes
          j1=random(0, upstreamNodeSet.size( )/2 );
          j2=random(upstreamNodeSet.size()/2,
                          upstreamNodeSet.size( ) );
          // clear some set bits in j1's bitmap which are also set  in j2's
          pattern1= upstreamNodeSet[j1].pattern()&
                      ~(upstreamNodeSet[j1].pattern()&
                        upstreamNodeSet[j2].pattern() )
           //set request with new pattern to node j1
           requestSet.insert(upstreamNodeSet[j1], pattern1)
   end if
end if;
```

Output:
requestSet[ ]: set of requests to be sent out

Figure 4. Streaming scheduling algorithm

To handle the streaming scheduling task, a simple algorithm is embedded in the *message processor*. Its basic logic is shown in Figure 4. The main idea of the algorithm is to adjust the pattern to reduce duplicated data delivery to below a certain level, denoted by the parameter *exptRedRatio*. When all necessary segments are received, if the ratio of the number of all received segments to the number of effectual segments is greater than *exptRedRatio*, then the node stops sending out requests for more segments and starts to reduce the duplication by sending out requests with adjusted *pattern*s to its current upstream nodes. In order to reduce the impact of different optimized scheduling algorithms on streaming, this simple algorithm merely fulfills the basic demands on redundancy reduction.

4. SIMULATION AND EVALUATION

To evaluate the control overhead and efficiency of our request-driven scheme, we implemented a prototype under a simulation environment and conducted experiments on it. In this section, we first introduce the simulation platform. Then the prototype implementation and experimental parameter settings are presented. Finally, simulation results are presented and analyzed in a comparison with Coolstreaming, the representative system of data-driven swarming system.

4.1 Simulation Environment

The simulation platform consists of OMNeT++ [11] with its extension modules INET [12] and OverSim [3]. OMNeT is a component-based and open architecture discrete time event simulator. INET is a framework built on OMNeT++ and contains IPv4, IPv6, TCP, UDP and many other protocol implementations, and several application models. OverSim is an open-source overlay network simulation framework residing on top of the INET framework. The simulation suite provides an integrated P2P framework as shown in Figure 5(a).

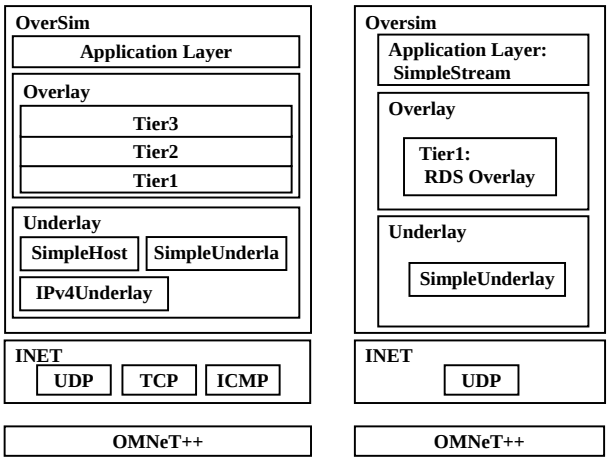
Figure 5(b) depicts the architecture of our prototype implementation according to the framework in Figure 5(a). In the implementation, OMNeT++ takes charge of message delivery and INET simulates the network communication scenario. On top of INET, the RDS (request-driven swarming) overlay realizes the system design presented in Figure 3. Above the overlay, the application layer, named SimpleStream, acts as streaming provider or consumer, sending or receiving data from the RDS overlay.

4.2 Evaluation

Under this simulation environment, we evaluated overhead cost and efficiency of the prototype design. The request volume and the in-time delivery ratio are chosen as the metrics to compare with published data of Coolstreaming. The two metrics under different swarming scales are analyzed and discussed here.

As the source code of Coolstreaming is not publicly available, it is impossible to transplant its implementation to the same simulation environment for comparison. So we choose similar simulation metrics as in [14]. To evaluate overhead traffic volume, we choose the number of request messages and the replies as the metric. With regard to evaluating the streaming efficiency, an index named in-time delivery is used. The index has the same meaning as the metric continuity-index in Coolstreaming.

In the evaluation, all the simulation runs share the same parameters. The duration time of all simulation run is 5000 seconds, which is close to the playback time of a typical movie. The nodes' creation and death follow the Weibull distribution [13], which is commonly used in life data analysis and P2P network simulation. The node's bandwidth and delay follow uniform distributions over the intervals (128Kbps, 10Mpbs) and (50ms, 10ms), respectively. The end points of these intervals correspond to



(a) OMNet++, INET, and OverSim **(b) RDS Implementation**

Figure 5. Simulation framework and prototype implementation

typical DSL (Digital Subscriber Line) and Ethernet parameters. Only one stream from one source node called the origin node is involved, and the origin node is set to generate one segment per second. Segment delivery uses UDP (User Datagram Protocol) and the segment length is set to 1000 bytes. Therefore a segment can be encapsulated in an individual packet. *exptRedRatio* in the scheduling algorithm is 1.5 for all nodes. All experiments for different metrics are repeated with various node numbers from 50 to 1000.

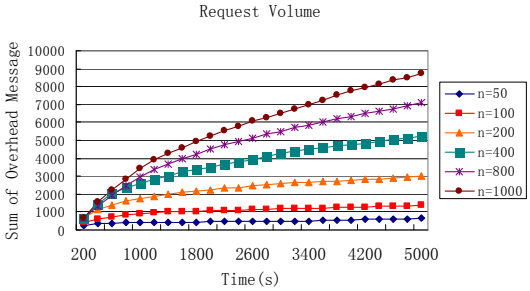


Figure 6. Cumulative request volume as a function of time under different swarming scales

Figure 6 depicts the cumulative volume of messages as a function of time in different swarming scales. As shown in the figure, the cumulative message traffic volume increases with time and with the scale; and the increasing rates become stable. Figure 7 shows the volume in every 200-second interval and gives an obvious view of the trend. For

all simulation runs with different swarming scales, the request volume in each interval is high at the beginning because of the initial construction of swarming relationships. Then when most of the nodes have attached to the stream the volume decreases and finally stays at stable low levels.

Table 1 presents detailed statistics of the average message traffic volume sent per node in different swarming scales.

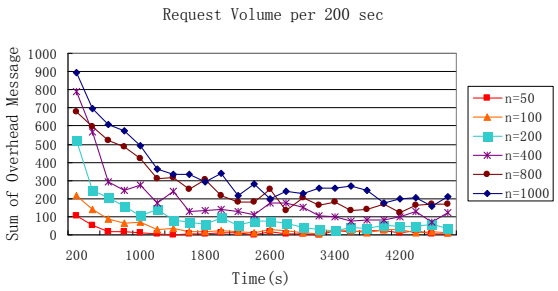


Figure 7. Request volume per 200 sec as a function of time under different swarming scales

The data indicates that the number of requests sent per node is limited and independent of the scale of the swarming.

Table 1. The average number of requests sent per node

Swarming scale (node number)	50	100	200	400	800	1000
Requests sent per node	0.34	0.403	0.488	0.478	0.337	0.336

In our prototype the request packet size is 80 bytes, and in 200 seconds the number of requests sent by every node is less than 0.5 on the average. So the overhead cost of each node in 200 seconds is only 40 bytes. According to the data published in [14], the control overhead of Coolstreaming, which is defined as the ratio of message volume to the video data by byte, is about 0.006 when the number of partners is 2. This corresponds to *exptRedRatio* =1.5 in our scheme. Given the video data rate is 500Kbps, the number of nodes 100, the overhead cost of Coolstreaming in 200 sec should be about 500kbps×200sec×0.006/100nodes/8 = 768bytes/node, which is more than 19 times the overhead in our prototype. Therefore, we believe the request-driven swarming scheme can greatly reduce the volume of overhead.

The efficiency of data delivery is another feature of the request-driven scheme. In the evaluation, we use the metric in-time delivery ratio to estimate the delivery efficiency. This metric is defined as the ratio of the number of received effectual segments to the segment numbers expected in a given interval. The metric has the same meaning as the metric of continuity index in the evaluation of Coolstreaming [14]. In the experiments, nodes are expected to receive segments at the same rate as the origin node, which is 60 segments per minute. Figure 8 shows the result under different swarming scales. When streaming starts up, the ratio is low since only a few nodes join the streaming and they have not received many effectual segments. Then the ratio increases quickly since more and more nodes participate in streaming. Finally, when almost all nodes have attached and are receiving data from streaming, the ratio reaches its upper bound, which is about 0.983 in all simulation runs under different swarming scales. Compared with the upper bound of continuity in [14] which is 0.97, the request-driven scheme achieves high efficiency similar to Coolstreaming.

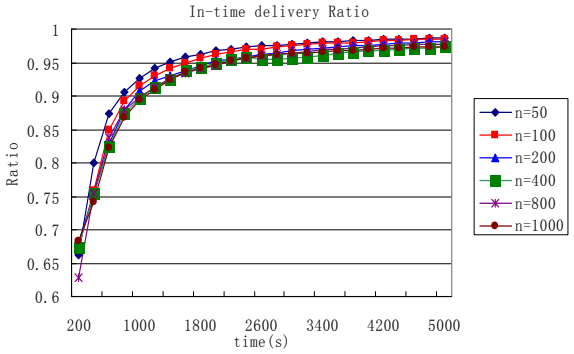


Figure 8. In-time delivery ratio as a function of time under different swarming scales

5. CONCLUSION

In this paper, a request-driven swarming scheme for P2P overlay streaming is proposed to address the issues of latency and control overhead reduction. This scheme does not maintain an explicit overlay structure but uses requests to guide the streaming adaptively. Experimental results show that the request-driven swarming scheme has low control overhead but does not sacrifice efficiency. In comparison with Coolstreaming, a representative system of data-driven swarming, the request-driven scheme promises more flexible control over streaming, much lower control overhead and similar efficiency in data delivery. To achieve better balance between performance and bandwidth expense, future work includes finding more sophisticated algorithms on upstream node selection and bandwidth estimation.

6. REFERENCES

[1] ANNAPUREDDY, S., GUHUA, S., GKANTSIDIS, C., GUNAWARDENA, D., AND RODRIGUEZ, P. 2007. Exploring VoD in P2P Swarming Systems. In Proceedings of IEEE INFOCOM 2007, Anchorage, Alaska, USA, MAY 2007, 2571- 2575.

[2] BANERJEE, S., KOMMAREDDY, C., KAR, K., AND BHATTACHARJEE, B. 2003. Construction of an efficient

overlay multicast infrastructure for real-time applications. In: Proceedings of IEEE INFOCOM 2003, San Francisco, California, USA, APRIL 2003, vol. 2 1521- 1531.

[3] BAUMGART, I., HEEP, B., AND KRAUSE, S. 2007. OverSim: A Flexible Overlay Network Simulation Framework. In Proceedings of 10th IEEE Global Internet Symposium, Anchorage, AK, May, 2007, 79 - 84.

[4] CASTRO, M., DRUSCHEL, P., KERMARREC, A., NANDI, A., ROWSTRON, A., AND SINGH, A. 2003. SplitStream: high-bandwidth multicast in cooperative environments. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. Bolton Landing, NY, USA, OCTOBER, 2003, SOSP '03. ACM, New York, NY, 298-313. DOI= http://doi.acm.org/10.1145/945445.945474.

[5] HIM, K. A., CAI, Y., AND SHEO, S. 1998. Patching: A multicast technique for true video-on-demand. in Proceedings of ACM Multimedia, Bristol, England, SEP 1998, 191-200.

[6] LIU, J., LI, B., AND ZHANG, Y.-Q. 2003. Adaptive video multicast over the Internet. IEEE Multimedia 2003, vol. 10, no. 1, 22-31.

[7] POON, W.-F., LO, K.-T., AND FENG, J. 2001. Adaptive batching scheme for multicast video-on-demand systems. IEEE Transactions on Broadcasting, vol. 47 no. 1 66 -70.

[8] RAMESH, S., RHEE, L., AND GUO, K. 2001. Multicast with cache (Mcache): an adaptive zero-delay video-on-demand service IEEE Transactions on Circuits and Systems for Video Technology, vol. 11 no. 3 440-456.

[9] SEN, S., GAO, L., REXFORD, J., AND TOWSLEY, D. 1999. Optimal patching scheme for efficient multimedia streaming. In Proceedings of. NOSSDAV.1999, AT&T Learning Center, Basking Ridge NJ, JUNE, 1999.

[10] TIAN, R., ZHANG, Q., XIANG, Z., XIONG, Y. LI, X., AND ZHU, W. 2005. Robust and efficient path diversity in application-layer multicast for video streaming Circuits and Systems for Video Technology, IEEE Transactions. vol 15, issue 8, 961- 972.

[11] VARAGA, A. 2001. The OMNeT++ discrete event simulation system. In Proceedings of European Simulation Multiconference (ESM'2001), Berlin, Germany, Prague, Czech Republic, http://www.omnetpp.org/portal.php?what=link&item=20030407013804784.

[12] VARGA, A. 2007. INET Framework for OMNeT++/OMNEST. http://www.omnetpp.org/doc/INET/

[13] WEIBULL, W. 1951. A statistical distribution function of wide applicability. J. Appl. Mech.-Trans. ASME 18(3), 293-297.

[14] ZHANG, X., LIU, J., LI, B., AND YUM, T.-S. P.2005. CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In Proceedings of IEEE INFOCOM 2005 24th Annual Joint Conference of the IEEE Computer and Communications Societies. vol. 3 2102-2111.