Streaming Live Media over BitTorrent

Qingchao Cai
School of Information
Yunnan University
Kunming, China, 650091
qingchaochoi@gmail.com

Xiaolu Zhang
School of Information
Yunnan University
Kunming, China, 650091
xiaolu62@hotmail.com

Xuejie Zhang
School of Information
Yunnan University
Kunming, China, 650091
xjzhang@ynu.edu.cn

Abstract

In this paper, we present an effective approach for live media streaming based on BitTorrent mechanism. In the approach, several modifications have been incorporated into BitTorrent to support on-time content delivery. First, pieces are selected based on their scheduler playback deadline. Second, source behaves in a push-based way instead of pull-based way. Finally, each peer periodically examines its active neighbors to improve the bandwidth utilization. Our simulations indicate that our proposals improve the bandwidth utilization and accommodate high scalability and resilience, and a good streaming quality can be achieved.

1 Introduction

Recently, as the demand for streaming application increases, P2P streaming has attracted the focus from academe and industry since it can delivery live media from a single source to many receivers without the support of network layer multicast while with little deployment cost, and thus overcomes the limitation of traditional approach such as CDN and IP multicast.

There are two major classes of P2P streaming: *(i)* tree-based approach [2][13][6][4][9][12], and *(ii)* mesh-based approach [14][10][7][11]. In tree-based approach, peers are organized to form multiple distribution trees where each sub-stream of live content is diffused. This approach has following disadvantages [7] : *(i)* The outgoing bandwidth of peers with deficient children can not be effectively utilized, thus limits the scalability. *(ii)* In the presence of churn, it costs much to maintain the overlay topology. *(iii)* The rate of content delivery to each peer through multiple trees is limited by the upstream connections with least throughput.

In mesh-based approach, participating peers form a randomly connected mesh and incorporate swarming data transfer mechanism, which leverages the availability of entire file at the source to distribute different pieces of file among participating peers, increasing the diversity of data and accelerating content delivery. However, incorporating this mechanism into P2P streaming is in challenging for following reasons: *(i)* Content must be delivered in-time, and *(ii)* the lifetime of content produced progressively by a live source is short, which limits the diversity of pieces among peers and thus degrades the utilization of their outgoing bandwidth.

Recently, some studies have been conducted to address some of the above problems. X. Zhang and et al [14] propose DONET, a data-driven overlay network for live media streaming. Each node in DONET periodically exchanges its buffer map which contains the information about what content it has with neighbors, then requests data it doesn't have from neighbors by checking the buffer map while responses requests from neighbors by uploading data to them. PRIME [7] presents an organized view of peers in a randomly connected mesh. All participating peers are grouped into several sub-trees rooted at the source based on their shortest paths from the source. In diffusion phase, a peer only acquire content from its parent in the sub-tree it belongs to, while in swarming phase, each peer can request other peers located in other sub-trees to exchange data. Chainsaw [10] is a bit like DONET [14] in terms of eliminating trees from overlay multicast, each peer in Chainsaw maintains a window of interest to decide packets to be requested, and keeps track of packets it has requested from each neighbor to avoid requesting same packets. Climber [11] forms a single tree for streaming, and adds some random edges to the tree for redundant connections.

BitTorrent [5] is a scalable file sharing protocol which incorporating swarming data transfer, and several researches have proposed that BitTorrent performs near-optimally in terms of uplink bandwidth utilization and download speed [3], thus, BitTorrent is a suitable candidate for P2P streaming which also demands high bandwidth utilization.

In this paper, we present an effective approach for live media streaming based on BitTorrent mechanism. In the approach, several modifications have been incorporated into

978-0-7695-3501-2/09 $25.00 © 2009 IEEE
DOI 10.1109/CMC.2009.254

BitTorrent to support on-time content delivery. First, pieces are selected based on their scheduler playback deadline. Second, source behaves in a push-based way instead of pull-based way. Finally, each peer periodically examines its active neighbors to improve the bandwidth utilization. Our simulations indicate that our proposals improve the bandwidth utilization and accommodate high scalability and resilience, and a good streaming quality can be achieved.

2 BitTorrent Protocol

BitTorrent [5] is a very scalable peer-to-peer file sharing protocol, it organizes participating peers into an unstructured overlay mesh to distribute a file, which is divided into disjoint and equal-sized pieces. Interested peers connect several other peers simultaneously and download different pieces of the file from these peers. The distribution of pieces is in a pull-based way that a peer must request a piece before downloading it.

In order to start a BitTorrent deployment, a file with the extension .torrent which contains the information about the file to be distributed is needed. Besides this .torrent file, there is another necessary component called tracker which keeps the trace of participating peers and helps peers find each other.

There are two types of peers in BitTorrent, namely seeds and leaches. Seeds are peers who have the entire file while leaches have a part (or none) of the file. Pieces can be downloaded directly from seeds or exchanged with other leaches. A new peer joins the swarm by contacting the trackers to obtain a random list of active peers, then it attempts to establish connections to these peers and finds out what pieces reside in each peer. Once it has received an UNCHOKE message from a neighbor, it then requests pieces it does not have from the neighbor and start downloading.

BitTorrent employs a rarest first policy for selecting pieces to download, each peer tries to download the least replicated pieces among its neighbors. Rarest first policy accelerates the distribution of the file among the peers, and reduces the risk of that no peer can finish downloading when there is no seed in the system.

While a leach can download from unlimited number of peers, it is allowed to upload only to a fixed number of at a given time even if it has received more requests. This process of temporary refusal to upload to some neighbors is called choking, and uploading is called unchoking. Which neighbors to be unchoked is based on a tit-for-tat policy, each peer uploads to peers who have provided it with the best download rates. Typically, the number of concurrent uploads of a peer is limited to four, and peers recalculate who they want to unchoke every ten seconds. The tit-for-tat policy in BitTorrent is effective in terms of improving the utilization of available resources and detering free-riding.

The pure tit-for-tat policy suffers from unable to discover if currently unused connections are better than ones being used. To fix it, each peer in BitTorrent at all times has a single optimistic unchoke, which is unchoked regardless the current download rate from it. Which neighbor is selected as the optimistic unchoke is rotated every 30 seconds.

Finally, a protocol called the endgame mode is introduced in BitTorrent to prevent peers which have the most of pieces from waiting a long time to finish download. A peer sends requests for all pieces which it doesn't have to all neighbors if these pieces were actively being requested, and cancels messages for pieces which have been received are sent to avoid wasting bandwidth on duplicate sends.

3 Incorporating BitTorrent Mechanism into P2P Streaming

While a BitTorrent swarm is scalable, efficient, robust, cost-effective, easy to deploy and with low control overhead, which also are the characteristics of a good P2P streaming system. BitTorrent is not inherently suitable to P2P streaming for the following reasons. First, BitTorrent doesn't support real-time applications. Second, peers do not download pieces in sequence. Finally, a peer should wait for a long time before it acquires the first piece because of the tit-for-tat policy.

In order to incorporating BitTorrent mechanism into P2P streaming, we propose following modifications to overcome above limitations.

3.1 Piece Request Policy

To smooth out the play of video data, a common approach is maintaining a buffer at the application level, and this approach is adopted in our system. Each peer buffers the pieces consumed in next playback interval in sequence, and drops pieces which arrived after their scheduled playback deadline.

In BitTorrent, peer requests the least replicated pieces among its neighbors regardless the position of these pieces, which may lead to the request of outdated pieces. To avoid this phenomena, We impose a restriction on the selection of pieces, which is that a piece with playback deadline close to current time would not be selected. However, a problem exists in the restriction is that it is hard to ascertain whether the playback deadline of a piece is close to current time since it depends on network environment. In our modification, we use a fix value to represent the threshold of the gap between the deadline and current time for simplicity.

Another problem of rarest first policy is that pieces with stricter deadline are equally treated with other pieces, hence these pieces may not be requested in time. To avoid this question, pieces with stricter deadline are assigned a higher

```
Input:
  $peer\_id$: id of peer which I send request to;
  $local\_set$: piece bitmap, 1 for having, otherwise 0;
  $deadline[i]$: deadline of piece i;
  $remote\_set[i]$: piece bitmap at peer i, same to
                    local_set
  $num\_neighbor$: number of neighbors;
  $num\_piece$: number of pieces;
  $priority[i]$: priority of piece i;
  $default\_priority$: 2 in our simulations;
  $low\_threshold$: decide which pieces to be
                    requested;
  $high\_threshold$: decide the priority of pieces;
begin
  for $i \leftarrow 0$ to num_piece do
    $occurrence[i] \leftarrow 0$
  end
  for $i \leftarrow 0$ to num_piece do
    if $priority[i] = default\_priority$ and
    $deadline[i] <$
    $current\_time + high\_threshold$ then
      $priority[i] \leftarrow priority[i] - 1$
    end
  end
  for $i \leftarrow 0$ to num_neighbor do
    for $j \leftarrow 0$ to num_piece do
      if $remote\_set[i][j] = 1$ then
        $occurrence[j] \leftarrow$
        $occurrence[j] + priority[j]$;
      end
    end
  end
  $min\_occurrence \leftarrow$
  $num\_neighbor * default\_priority + 1$
  for $i \leftarrow 0$ to num_piece do
    if $remote\_set[peer\_id] = 1$ and
    $local\_set[i] = 0$ and $deadline[i] >$
    $current\_time + low\_threshold$ then
      if $min\_occurrence > occurrence[i]$ then
        $min\_occurrence \leftarrow occurrence[i]$
        $candidate\_set \leftarrow null$
        $candidate\_set \leftarrow candidate\_set \cup i$
      end
      if $min\_occurrence = occurrence[i]$ then
        $candidate\_set \leftarrow candidate\_set \cup i$
      end
    end
  end
end
Output:
  piece_id: a random element in candidate_set.
```

Fig. 1: piece selection algorithm

priority to be chosen. The simulation result shows that this policy allows a large variety of pieces to be downloaded concurrently while increases the ratio of valid request. The piece selection algorithm is shown in figure 1.

Upon receiving a piece, a peer checks the request queue and cancels the requests for pieces which have not arrived after scheduled playback deadline. This can decrease the number of dirty pieces transferred in the system, and thus save the bandwidth.

3.2 Source Behavior

In a typical BitTorrent swarm, the seed might have to upload 150% to 200% of the total size of a torrent before other clients become seeds, this is because that it always receives the requests for pieces which have already been uploaded. To improve the seeding efficiency, we import super seeding mode [1] into our system. In super seeding mode, the seed pretends to be a normal leech with no data, as peer connect, the seed will inform this peer that it has received a new piece which has never been sent to any other peers, the seed then unchokes this peer and allows it to download the piece, after finish uploading this piece, the seed will not inform this peer of any other piece until it has receives the confirmation from other peers that the piece has been uploaded again.

As mentioned above, piece distribution is in a pull-based way that a peer uploads to one neighbor only after it has received request from this neighbor. But the source can deliver new pieces to its neighbors without requests since it knows these pieces are not available in its neighbors, thus, the source can behave in a push-based way, which will further increase the seeding efficiency and accelerate the distribution of new pieces.

At the end of each playback interval, the source produces several new pieces of content, then it continuously picks up a neighbor which received least from it and sends a new piece to this neighbor until all new data are sent. When all pieces are sent at least once, the source enters the normal mode. In this way, the source switches peers frequently, thus impact the utilization of its outgoing bandwidth, but pieces can be distributed more rapidly, which makes peers more useful to their neighbors and thus increases their upload speed and overall performance.

3.3 Choking Algorithm

The rate based tit-for-tat policy makes a lot of peers suffer from low start, since a new peer with no piece can acquire data only when it is selected as an optimistic unchoke which is performed every 30 seconds, thus, it should wait several 30-second's intervals to get the first piece, which is not tolerable in a streaming application.

After the deep study of BitTorrent protocol, we found that the reason to low start is that, although with abundant pieces, earliest joined peers could not unchoke enough peers until a period of choking interval passed by. To avoid this phenomenon, each peer examines the number of active neighbors periodically, if this number is less than upper limit, it will unchoke some choked neighbors to keep the number at a high level. In this way, early joined peers could choke enough neighbors as soon as possible. Selecting which peer to be unchoked is still depending on download rate from it to keep fairness. This modification gives new peers a higher probability of being unchoked, thus make new peers avoid low start.

4 Performance Evaluation

We implement a discrete-time simulator to evaluate the performance of our system, and run several different experiments on a random network to examine the performance increase brought by the modification proposed in this paper

4.1 Experimental Environment

The physical topology used in our simulation is generated using BRITE [8] with the following parameters: 20 AS with 10 routers each AS in top-down mode and RED queue management at all routers. All core links are symmetrical with high bandwidth ranging from 4Gbps to 10Gbps. We modeled the following two scenarios in our evaluations: 200 homogeneous peers with *1)* 1Mbps and *2)* 1.5Mbps link bandwidth. The edge links are also symmetrical with a delay randomly selected between [5ms, 25ms], thus bandwidth bottleneck would only occur at edges.

A video with the duration of about six minutes is used in our evaluation, it has a constant bit rate of 800 kbps. The source generates a new part of the video which consists of a group of packets with consecutive timestamps every one second, and encodes them with Multiple Description Coding (MDC).

All peers join the overlay at the beginning of the deployment of swarm, and begin to consume content after a playback delay of 60s. Piece size is set to 16KB to shorten the time spending on transferring a completed piece, and each peer serves at most 15 peers. Other parameters are the same as the original BitTorrent protocol. Each simulation will last 420s which is the sum of the duration of the video and playback delay.

4.2 Simulation Result

The main factors on performance we investigate include: *(i)* piece request policy, *(ii)* upload bandwidth utilization, *(iii)* delivered quality, and *(iv)* resilience and scalability.

4.2.1 Piece Request Policy

We investigate the impact of the new piece request policy on the efficiency of the system. Figure 2 shows the percentage of valid request in different scenarios and the ratio of valid pieces, here a valid request means that the sender has received the requested piece before its scheduled playback deadline, while a valid piece indicates that it arrived in time.

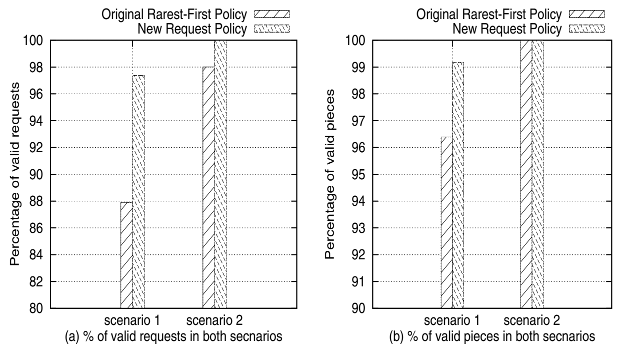

Fig. 2: Effect of new request algorithm

Since the new piece request policy takes consideration of the priority and playback deadline of pieces, the performance advantage of our system against original BitTorrent protocol will become larger as the link bandwidth of peers in the swarm decrease, which can be directly deduced from figure 2.

As is shown in figure 2, the new piece request policy behaves better than the original policy in terms of the ratio of valid requests and pieces in both scenarios. In scenario 1, the percentage of valid requests of peers implemented with new request policy is about 10% higher than that of peers implemented with original policy. In addition, In our system, more than 99% of data arrive in time, while 4% of content arrive late in original BitTorrent swarm, which leads to the waste of network resource. As the bandwidth capacity of peers increase to 1.5Mbps, both of the ratio of valid requests and pieces increase to 100% in our system, and original BitTorrent protocol also perform much better. Since our system performs near-optimally on the presence of low bandwidth, there is little space for the improvement of performance, thus the advantage of the new piece request policy becomes less obvious.

4.2.2 High Utilization of link Bandwidth

The utilization of outgoing bandwidth is one of the most important performance factors, it determines the overall performance and delivered content quality. Several researches have proved that BitTorrent performs near-optimally in terms of uplink bandwidth utilization and download speed [3]. Though we have imposed some restrictions on BitTorrent to support streaming, we can still expect a system with high link bandwidth utilization.

Figure 3 shows the average download rate of leaches in different scenarios. As mentioned above, new peers in BitTorrent may suffer from low start because of the tit-for-tat policy, which is confirmed in figure 3 where peers in original protocol suffer serious low start until a very long period of time passed in both scenarios. However, this situation does not exist in our system, as can be seen from figure 3, peers in our system start downloading as soon as they joining swarm, and quickly saturate link bandwidth, then reach a stable status where the upload rate is nearly equal to the download speed and stay in this status until the end of streaming.

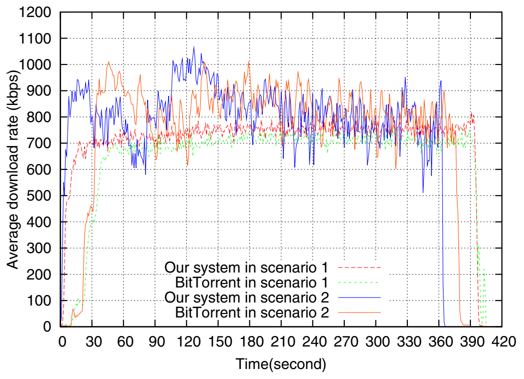

Fig. 3: Link bandwidth utilization

Besides avoiding low start, our system also behaves much better than original protocol in terms of average bandwidth utilization. In scenario 1, peers in our system download at an average speed of about 750kbps, which is nearly 50kbps faster than peers in original protocol, this is due to the rapid distribution of new pieces and high piece variety in our system.

As link capacity of participating peers increase, the average download rate of both scenarios have increased obviously. However, due to quick start, all peers in our system finish download only seven seconds after the end of video, while peers in BitTorrent swarm should experience a delay of 30 seconds.

4.2.3 Delivered Quality

We examine delivered quality which is an another important performance factor by computing how much content they have downloaded. Figure 4 depicts the distribution of population of peers with different perceived quality. As shown in the figure, in our system, most peers with link bandwidth of 1Mbps acquired more than 94% of the content produced by the source, and all peers with link bandwidth of 1.5Mbps have downloaded all data. The average downloaded content among peers in our system is about 10 percent more than in original BitTorrent swarm in scenario 1, which is a huge improvement in streaming application. While in scenario 2, peers acquire almost the same quality in both implementations.

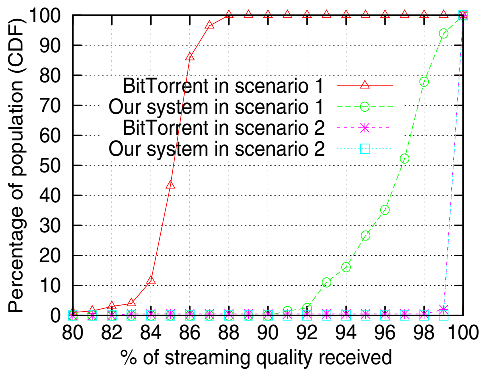

Fig. 4: Distributions of peers received different quality

Our system has avoided another problem existed in BitTorrent, which is that peers may be snubbed by neighbors, thus get a poor download rate. As can be seen from figure 4, in scenario 2, although with high link bandwidth, there is still one peer with perceived quality less than 82% in BitTorrent swarm, while all peers in our system have received 100% content.

4.2.4 Resilience and Scalability

To investigate the resilience of our system, a catastrophic event is simulated by killing half of peers at the time of 200s in scenario 1, thus, on average, the survived peers will be left with half the neighbors they had before the event.

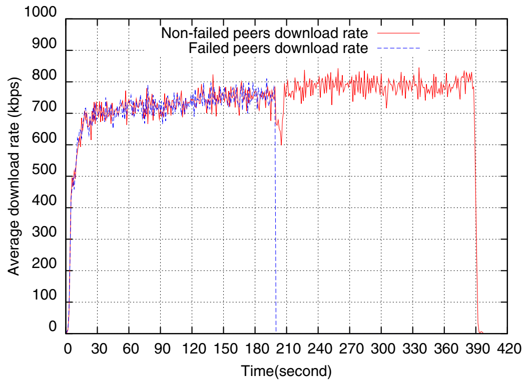

Fig. 5: Observed download rate with the failure of half of peers at 200s

Figure 5 shows the impaction of the simultaneous failure of half of peers at 200s. Due to the departure of half of peers, pieces transferred in the connections of these peers lost, and the active neighbors of survived peers reduced. Hence the average download rate of survived peers experienced a relatively sharp decrease which can be seen from the figure. However, this process only lasted for a short time less than 5 seconds. After reestablishing neighbor relation, survived peers recovered from the failure and once again acquired a high bandwidth utilization.

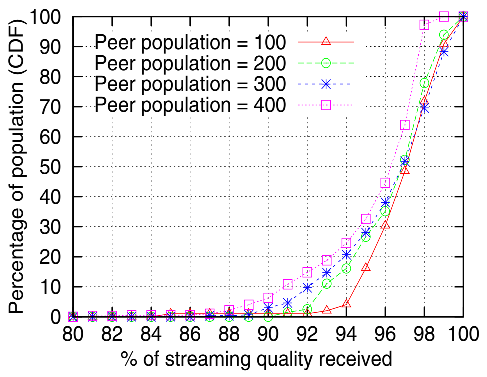

Fig. 6: Distributions of peers received different quality with different population

Finally, we adjust the peer population in scenario 1 to examine the scalability of our system. Since our system is based on BitTorrent which is a very scalable file sharing protocol, a good scalability can be achieved in our system, and figure 6 provides a nice evidence.

It is obvious that most of peers have received more than 94% of original video file no matter how much the peer population is, thus, the increase of peer population has little impaction on the delivered quality. With large peer population, the performance of our system may experience a tiny degradation, however, it can be effectively solved with a little increase of playback delay and buffer size.

5 Conclusion

In this paper, we propose an effective approach for live media streaming based on BitTorrent mechanism. We first give a simple description of BitTorrent protocol and analysis the problems encountered while importing it into P2P streaming. To address these problems, we make some modifications to BitTorrent protocol, these modifications include *(i)* prioritizing the pieces based on their deadline, *(ii)* redefining source behavior, and *(iii)* periodically examination of active neighbors. Through extensive NS simulations, we give the performance analysis of our system from several aspects, and the simulation result indicates that our system can effectively utilize bandwidth of peers and has the advantage of high resilience and scalability.

6 Acknowledgements

This work was supported by National Natural Science Foundation of China under Grant No. 60573104 and Innovation Group Project of Yunnan University.

References

[1] Super-seeding. Available: `http://en.wikipedia.org/wiki/Super-seeding`. [online].

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proc. ACM SIGCOMM*, Pennsylvania, USA, Aug. 2002.

[3] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving a BitTorrent Network Performance Mechanisms. In *Proc. Infocom*, Barcelona, SPAIN, Apr. 2006.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Content Distribution in a Cooperative Environment. In *Proc. SOSP*, New York, USA, Oct. 2003.

[5] B. Cohen. Bittorrent. Available: `http://www.bittorrent.com`. [online].

[6] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming live media over peer-to-peer network. Technical report, Stanford University, 2001.

[7] N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-Driven Mesh-based Streaming. In *Proc. INFOCOM*, Alaska, USA, May. 2007.

[8] A. Medina, A. lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. In *Proc. MASCOTS*, Ohio, USA, Jan. 2001.

[9] V. N. Padmanabhan, H. J. Wang, and P. A. Chou. Resilient Peer-to-Peer Streaming. In *Proc. ICNP*, Georgia, USA, Nov. 2003.

[10] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proc. IPTPS*, New York, USA, Feb. 2005.

[11] K. Park, S. Pack, and T. Kwon. Climber: An Incentive-based Resilient Peer-to-Peer System for Live Streaming Services. In *Proc. IPTPS*, Florida, USA, Feb. 2008.

[12] D. A. Tran, K. A. Hua, and T. T. Do. A peer-to peer architecture for media streaming. *IEEE J. Select. Areas in Comm.*, 22, Jan. 2004.

[13] D. Xu, M. Hefeeda, S. Hanbrusch, and B. Bhargava. On Peer-to-Peer Media Streaming. In *Proc. of ICDCS*, Wien, Austria, Jul. 2002.

[14] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In *Proc. INFOCOM*, Miami, USA, Mar. 2005.