Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Adaptive peer-to-peer streaming with MutualCast

HUANG Cheng, CHOU Philip A., LI Jin, ZHANG Cha
(*Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA*)
E-mail: {chengh; pachou; jinl; chazhang}@microsoft.com
Received Dec. 1, 2005; revision accepted Feb. 20, 2006

Abstract: Application Layer Multicast (ALM) can greatly reduce the load of a server by leveraging the outgoing bandwidth of the participating nodes. However, most proposed ALM schemes become quite complicated and lose bandwidth efficiency if they try to deal with networks that are significantly heterogeneous or time-varying. In earlier work, we proposed MutualCast, an ALM scheme with fully connected mesh that quickly adapts to the time-varying networks, while achieving provably optimal throughput performance. In this paper, we study how MutualCast can be paired with adaptive rate control for streaming media. Specifically, we combine Optimal Rate Control (ORC), our earlier control-theoretical framework for quality adaptation, with the MutualCast delivery scheme. Using multiple bit rate video content, we show that the proposed system can gracefully adjust the common quality received at all the nodes while maintaining a continuous streaming experience at each, even when the network undergoes severe, uncorrelated bandwidth fluctuations at different peer nodes.

Key words: Peer-to-peer streaming, Distribution capacity, Quality adaptation, MutualCast
doi:10.1631/jzus.2006.A0737 **Document code:** A **CLC number:** TN919.8

INTRODUCTION

Many applications, such as software distribution, Internet TV/video streaming, video conferencing, multiplayer gaming, personal media distribution and P2P web content duplication, distribute the same content from one source node to many destination nodes. For these applications, IP multicast is an ideal network layer solution. A distribution tree rooted at the source can be formed to reach an arbitrary number of receivers. During content distribution, every single piece of data from the source is duplicated at the intermediate tree nodes (routers) and the load on the source is kept minimal. Due to many reasons, however, IP multicast is not widely available and Application Layer Multicast (ALM) becomes a viable alternative. The general concept of ALM is to build a multicast overlay with all the participating nodes. Then, every node can duplicate and redistribute data to others connected through the overlay, and thus functions similarly as a router in IP multicast. Here, the nodes are peers to each other.

Compared to centralized solutions where all the receivers get data directly from the source, ALM leverages the upload bandwidths of all the peer nodes and in turn significantly reduces the load on the source. Intuitively, the more ALM exploits the peers' upload bandwidths, the higher overall throughput can be achieved. In fact, the maximum system wide throughput (or distribution capacity) is completely determined by the aggregate upload bandwidths of all the participating nodes (including the source). Note that the download bandwidths of the nodes are of less concern, as the penetration of broadband network virtually ensures that the whole network is asymmetric, as most residential nodes have a much narrower upload pipe than the download. Trying to achieve the distribution capacity is one of the key ALM design issues.

ALM can be used for asynchronous data distribution (e.g. file sharing). Many schemes have been proposed and studied, such as BitTorrent (Cohen, 2003), FastReplica (Cherkasova and Lee, 2003), Bullet (Kostic *et al*., 2003), just to name a few. As a matter of fact, some schemes utilize the peers' upload bandwidths very efficiently. For instance, the study of

BitTorrent (Bharambe *et al*., 2005) reveals that the nodes operate fairly close to their upload capacity under normal conditions. Recently developed Avalanche scheme (Gkantsidis and Rodriguez, 2005) can achieve even higher distribution throughput at the start and the end of distribution by using network coding.

ALM can also be used for synchronous data distribution, e.g., synchronous streaming media delivery. Many schemes have been proposed and studied in this space, such as End System Multicast (ESM) (Chu *et al*., 2000), Scattercast (Chawathe, 2000), Overcast (Jannotti *et al*., 2002), CoopNet (Padmanabhan *et al*., 2003), SplitStream (Castro *et al*., 2003), Scribe (Castro *et al*., 2002), Bayeux (Zhuang *et al*., 2001), CAN-multicast (Ratnasamy *et al*., 2001), and more recently CoolStreaming (Zhang *et al*., 2005), PPLive (http://www.pplive.com), Feidian (http://tv.net9.org), etc. Similar to asynchronous data distribution, these schemes also strive to maximally utilize the peers' upload bandwidths. For example, in ESM, all the peer nodes construct an overlay in a self-organizing and fully-distributed manner. Then, they gather information on network path characteristics using both passive monitoring and active measurements. The overlay structure is continuously refined as more network information becomes available. In ALMI (Pendarakis *et al*., 2001), each session has a session controller which constructs a shared tree. The tree is periodically recalculated based on the end-to-end measurements collected by the session members. CoopNet and SplitStream split the content into multiple stripes and distribute the stripes across separate multicast trees with disjoint interior nodes. Thus, any peer could be an interior node in one of the multicast trees, and contribute to forwarding the content. CoolStreaming dynamically adjusts data requests sent to the peers based on content availability and connection characteristics, in the hope of retrieving data from the peers at maximum rates. All these schemes, however, are yet to achieve the distribution capacity, despite of their complexities.

One unique aspect of streaming media, which makes it fundamentally different from file sharing, lies in the fact that users demand smooth playing experience. The users might be willing to wait reasonable amount of time initially, but once the playback starts, no unintentional interruption is ever desirable. In the Internet environment, however, network dynamics (e.g. delay, jitter and congestion) can and do occur. The available bandwidth, packet loss and packet delay all fluctuate. In an ALM overlay composed by many peer nodes, the variations of individual node will also affect the distribution capacity. To maintain continuous user experience, it is therefore crucial to consider these network dynamics when designing an ALM scheme. Obviously, one easy way to cope with the changes of network conditions is through over-provision. By choosing a streaming bit rate well within the distribution capacity, many schemes get around this problem. In fact, as pointed out by Wang *et al*.(2004), the network bandwidth, delay and jitter fluctuations can be well concealed when the available throughput is much higher than the streaming bit rate. Indeed, authors' limited experience with practical systems, such as CoolStreaming, PPLive and Feidian shows reasonable continuity. However, the biggest disadvantage is that the reception quality offered by these systems is quite limited due to the over-provision. Ideally, the streaming quality should adapt accordingly as the distribution capacity varies. The faster all the peers can upload, the better quality each node should receive. Yet, it is very difficult (if not impossible) for general ALM schemes to know the distribution capacity and adapt the streaming quality accordingly, especially when the network is significantly heterogeneous or time-varying.

Therefore, peer-to-peer streaming systems usually face the following dilemma. On one hand, the systems try to maximally exploit the peer nodes' upload capacity, so as to increase the overall throughput. While on the other hand, the systems tend to over-provide and choose relatively low streaming bit rates, thus resulting in inefficiency and less satisfactory experience.

In this paper, we address this dilemma with an adaptive peer-to-peer streaming solution using a MutualCast framework. MutualCast arranges the nodes in a fixed topology. It adapts efficiently to bandwidth variations of individual nodes and achieves provably optimal throughput performance. Moreover, MutualCast can be effectively implemented using redistribution queues, which naturally provide information needed for quality adaptation. Indeed, we combine Optimal Rate Control (ORC), a

control-theoretical framework for quality adaptation, with the MutualCast delivery scheme. Using multiple bit rate video content, we demonstrate that the proposed system can gracefully adjust the common quality received at all the nodes while maintaining a continuous streaming experience at each, even when network dynamics cause independent, severe bandwidth fluctuations.

We present the major ideas in our paper as follows. In Section 2, we describe the MutualCast framework, including distribution routes, bandwidth allocation and distribution queues. In Section 3, we formulate the quality adaptation as a feedback control problem and describes an optimal linear quadratic control solution. We then describe how to combine MutualCast with ORC through the redistribution queues. In Section 4, we present experimental results from a real system using multi bit rate encoded video content. Finally, we conclude the article in Section 5.

MUTUALCAST DISTRIBUTION

Framework

MutualCast (Li *et al*., 2005) differs from many peer-to-peer delivery schemes in that it uses a fixed network topology, but adapts by letting peer nodes with different capabilities distribute different amount of content. MutualCast splits the content into many small blocks. Each block is assigned to one single node for redistribution, which could be a peer node requesting the content, a peer node not requesting the content, or even the source node itself. The node in charge of redistribution is then responsible for duplicating the block to all other peer nodes requesting the content. Intuitively, the number of blocks assigned to each node should be proportional to its capability. A more capable node may redistribute more blocks, and a less capable node may redistribute fewer. Then, even though the nodes' capability to distribute blocks may vary due to fluctuations in their upload bandwidths, packet loss and packet jitter, MutualCast can easily cope with these dynamics on an ongoing basis.

The MutualCast framework can be illustrated using an example shown in Fig.1. The source node s has content which is chopped into 8 blocks. Among the total 4 peer nodes, 3 nodes (t_1, t_2 and t_3) want a copy of the content, and the other node (t_4) does not want a copy but merely assists in the distribution. Some blocks (e.g. 1, 2, 3 and 4) are assigned to nodes t_1, t_2 and t_3 for redistribution. Taking block 1 as an example, it is first sent by the source node to t_1, which then forwards it to t_2 and t_3. Some blocks (e.g. 5, 6 and 7) are assigned to nodes t_4. Taking block 5 as an example, it is first sent by the source node to t_4, which then forwards it to all t_1, t_2 and t_3. Other blocks (e.g. 8) are assigned to the source node itself, which directly distributes these blocks to all peer nodes requesting the content (i.e. t_1, t_2 and t_3 in here). Note that the size of the block is a compromise between the granularity of distribution and the overhead required for identifying the block. We choose the block size to be 1 kB in our implementation. It should be clear from this example that the network topology of MutualCast is in fact generic and fixed. Indeed, MutualCast relies on bandwidth reallocation to adapt to the network condition variations, instead of less preferred approaches of adapting the network topology itself. Of course, the fixed and fully connected topology limits the size of the MutualCast network.

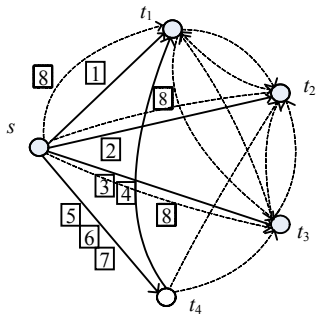

Fig.1 Example of MutualCast delivery

Distribution routes and bandwidth allocation

The above example shows that the MutualCast network distributes the content blocks through three routes: (1) through the content-requesting peer nodes; (2) through the non-content-requesting peer nodes; and (3) directly from the source node. In general, if the MutualCast network includes a source node, N_1 content-requesting peer nodes (N_1>1 as otherwise the problem is trivial) and N_2 non-content-requesting (but willing to assist in distribution) peer nodes, the content blocks will be distributed through N_1 height-2 trees with intermediate degree N_1-1 (with the inter-

mediate node being one of the content-requesting nodes), N_2 height-2 trees with intermediate degree N_1 (with the intermediate node being one of the non-content-requesting nodes), and one height-1 tree with degree N_1, all rooted at the source node.

Now, the bandwidth allocation question is which block should be assigned to which tree. Using a simple example, we can gain useful insights. Assume a portion of the content to be distributed needs bandwidth B. If it is assigned to distribute through a content-requesting peer node, then the upload bandwidth is $(N_1-1)B$ at the intermediate node and B at the source node. Similarly, if it is distributed through a non-content-requesting peer node, then the upload bandwidth is N_2B at the node and B at the source. However, if it is directly distributed by the source node, then the source bandwidth is N_1B and much higher. Since it is desirable to use source bandwidth effectively, we should assign as many blocks as possible to the peer nodes (both content-requesting and non-content-requesting) and only assign blocks directly to the source node as the last resort.

This observation leads to the following general bandwidth allocation scheme. Among the three distribution routes outlined above, the most preferred route is route 1, followed by the route 2. Only when the source node still has upload bandwidth left after exhausting the routes 1 and 2, it may choose route 3 to distribute content directly to the peer nodes. Assume that in the MutualCast network the source node has an upload bandwidth B_s, the N_1 content-requesting peer nodes have an average bandwidth B_1, and N_2 non-content-requesting peer nodes have an average bandwidth B_2. Applying the distribution route selection strategy, the distribution capacity of the MutualCast network, defined as the maximum amount of content sent to the peer nodes, is obtained as:

$$\theta=\begin{cases} B_s, & B_s \le B_{s1}+B_{s2}, \\ (B_{s1}+B_{s2})+\dfrac{B_s-(B_{s1}+B_{s2})}{N_1}, & B_s > B_{s1}+B_{s2}, \end{cases} \quad (1)$$

where

$$B_{s1}=\frac{N_1}{N_1-1}B_1 \text{ and } B_{s2}=\frac{N_2}{N_1}B_2. \quad (2)$$

This shows that before the upload bandwidths of all the peer nodes have been exhausted, the distribution throughput is limited only by the upload bandwidth of the source node. All the N_1 content-requesting peer nodes receive content at the rate of the upload bandwidth of the source node. After the upload bandwidths of all the peer nodes have been exhausted, the distribution throughput becomes $(1/N_1)$th of the sum of the upload bandwidths of the network ($N_1B_1+N_2B_2+B_s$) minus a small portion (N_2B_2/N_1) wasted in the distribution through non-content-requesting peer nodes.

Distribution route selection through redistribution queue

With the priority outlined in the previous subsection, if the available upload bandwidths of the source and all the peer nodes are known, the bandwidth allocated between any two nodes may be calculated, and content blocks be distributed accordingly. However, there is an even simpler method that works in a distributed fashion. Simple queues can be used to estimate the bandwidth on any connections, and in turn govern the selection of the distribution routes based on the status of the queues. In this way, implicit bandwidth allocation can be achieved without knowing of the bandwidths.

The key idea is to establish a queue to buffer content being sent from one node to another, and to use the queue to control the speed of distribution between any two nodes. In our implementation of MutualCast, the connections between nodes are established via TCP protocol. The redistribution queues are thus simply the TCP send and receive buffers. An additional advantage of using TCP is that the flow control, reliable data delivery and node leave events are all automatically handled by TCP. Reliable data delivery in MutualCast is inherited through these reliable TCP connections. Congestion control in MutualCast is likewise inherited.

Let forward link represent the TCP connection carrying blocks to be redistributed, and delivery link the TCP connection carrying blocks not to be further redistributed. Then, each peer node establishes one delivery link with every other content-requesting peer node. The source node establishes one forward link with every non-content-requesting peer node, one forward and one delivery link with every content-requesting peer node. The selection of the distribution

routes then boils down to finding available slots in these links.

We now examine the workflow of the source and peer nodes. Each content-requesting peer node consists of two threads. One thread receives blocks from the delivery links. The other thread receives blocks from the forward link and redistributes them to all the other content-requesting peer nodes through the delivery links. For the non-content-requesting peer nodes, only the forward link thread is in operation.

The operational flow of the forward link thread of a peer node (both content-requesting and non-content-requesting) is shown in Fig.2. In each of the loop iteration, the peer node removes one content block from the incoming forward link, and copies the block onto the outgoing delivery links of all the other content-requesting peer nodes. The thread does not remove another content block from the incoming forward link until it has successfully copied the last content block onto all the delivery links. That way, if the outgoing delivery links are blocked, possibly resulted from reaching the limit on the upload bandwidth, the peer node will stop removing the content blocks from the incoming forward link. The receiving rate of the forward link thus is effectively regulated to be $(1/M)$th of the upload bandwidth, where M is the number of nodes that the content block is redistributed to. Clearly, M equals N_1-1 for the content-requesting peer node and N_1 for the non-content-requesting peer node.

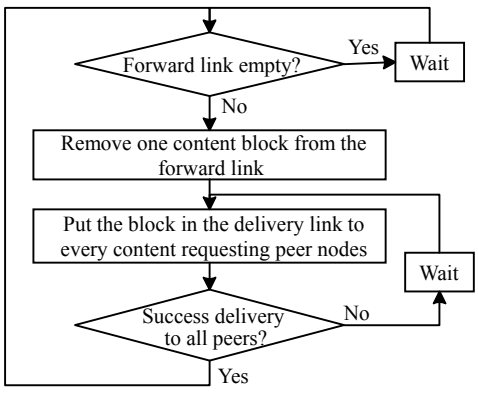

Fig.2 Forward link thread peer node

The operational flow of the delivery link thread (only in the content-receiving peer nodes) is shown in Fig.3. For content blocks arriving on the delivery links from nodes other than the source node, the operation is simply to remove them from the link as soon as they arrive. For content blocks arriving on the delivery link from the source node, we put an additional constraint that the blocks are removed only when the receiving buffer length of the forward link from the source node is above a certain threshold. The rationale is that the delivery link and the forward link are two competing TCP connections sharing the same network path from the source to the peer node. The content blocks sent through the forward link have higher priority, as they are to be redistributed to the other content-requesting peer nodes. The receiving buffer length policy guarantees that the bandwidth of the forward link is at least $(1/M)$th of the upload bandwidth before the delivery link is activated.

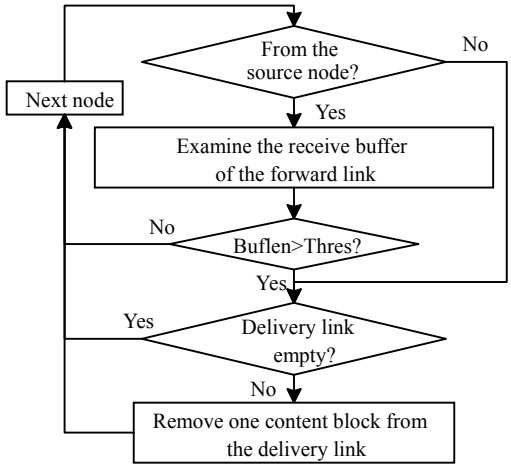

Fig.3 Delivery link thread peer node

The operational flow of the source node is shown in Fig.4. For each content block, the source node selects one of the distribution routes based on the status of the redistribution queues. The route selection is based on the following order of priorities: the highest priority to redistribute through the content-requesting peer nodes, followed through non-content-requesting peer nodes, and the lowest priority to distribute directly from the source node.

As shown in Fig.4, the source node first checks if there is space available in the forward links to the content-requesting peer nodes. If the send buffer of one of the forward link is not full, the content block is put into the buffer to be sent to the corresponding peer node, which then redistributes the block to all the

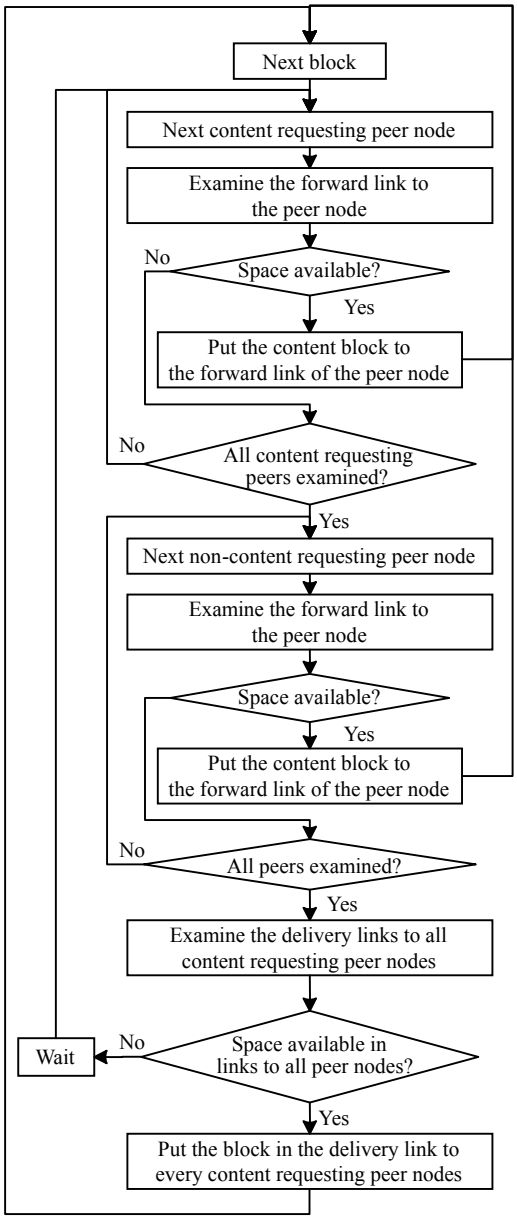

Fig.4 Operation flow of source node

other content-requesting peer nodes. If no available space can be found, the source node checks the forward links to the non-content-requesting peer nodes. If there is space available, then the content block is put into the buffer of the corresponding forward link. If there is still no space available even on these links, the source node pursues the final distribution route. It checks if there is space for one block available in all the delivery links to all the content-requesting peer nodes. Combined with the receiving buffer length policy in Fig.3, this ensures that the bandwidth of the forward link does not get squeezed by the traffic of the forward link. If space is found, the content block is replicated and put into the delivery link to each content-requesting peer node. If there is no space on any of the distribution routes, the source node will wait for a short time before it retries again.

Using redistribution queues and the above operational strategy for the peer and source nodes, MutualCast handles anomalies such as packet loss and network congestion by adjusting the upload bandwidths of the nodes. It also achieves the distribution capacity by fully utilizing the upload bandwidths of the source and all the peer nodes (refer to (Li *et al*., 2005) for theoretical analysis and optimality proof).

STREAMING QUALITY ADAPTATION

It should be clear now that the MutualCast network achieves optimal throughput performance. As the available upload bandwidth of individual peers varies, so does the throughput. When media content is streamed through the MutualCast network, it certainly faces time varying network conditions. However, users still expect that the startup delay will be low, playback will be continuous, and quality will be as high as possible given the average throughput.

Buffering at the client is the key to meeting these user expectations. Technically, buffering serves several distinct but simultaneous purposes. First, it allows the client to compensate for short-term variations in packet transmission delay (i.e., "jitter"). Second, it gives the client time to perform packet loss recovery if needed. Third, it allows the client to continue playing back the content during lapses in network bandwidth. And finally, it allows the content to be coded with variable instantaneous bit rate, thus improving overall quality.

By controlling the size of the client buffer over time it is possible for the client to meet the above mentioned user expectations. If the buffer is initially small, it allows a low startup delay. If the buffer never underflows, it allows continuous playback. If the buffer is eventually large, it asymptotically allows high robustness as well as high, nearly constant quality. Thus, client buffer management is a key element affecting the performance of streaming media systems.

The buffer duration (i.e., the number of seconds of content in the buffer) tends to increase or decrease depending on the ratio between the arrival rate r_a (the number of bits per second of real time that arrive at the client) and the coding rate r_c (the number of bits per second of encoded content, on average). If r_a/r_c is greater than the playback speed v, then the buffer duration increases; otherwise it decreases. The arrival rate r_a is essentially determined by the network capacity, so if the network capacity drops dramatically for a sustained period, reducing the coding rate r_c is the only appropriate way to maintain the buffer duration and prevent an underflow leading to a rebuffering event. Adjusting the coding rate in the face of time varying network conditions is the problem of coding rate control.

Today's commercial streaming media systems (Conklin *et al*., 2001; Birney, 2003) rely on multi bit rate (MBR) coding to perform coding rate control. In MBR coding, semantically identical content is encoded into alternating bit streams at different coding rates and stored in the same media file at the server, allowing the content to be streamed at different levels of quality corresponding to a set of coding rates $\{r_c\}$, typically using bit stream switching.

Here we first summarize an Optimal Rate Control (ORC) framework (Huang *et al*., 2004a; 2004b; 2005) based on linear quadratic optimal control theory and designed for both scalable and MBR streaming media. We then describe how to naturally combine ORC with the MutualCast delivery scheme and gradually adjust the common quality at all peers.

Optimal coding rate control

The elements of our control model are illustrated in Fig.5. Media time refers to the clock running on the device used to capture and timestamp the original content, while client time refers to the clock running time the client used to play back the content. The playback deadline indicating the time at which frames are instantaneously decoded and rendered, increases linearly at a rate of $1/v$ seconds of client time per second of media time, where v is the playback speed. (For instance, slow motion sets v=1/2, then each second of media will be consumed in 2 s of client time) The arrival schedule indicating the times at which encoded frames arrive at the client, increases in steps of size $b(n)/r_a$, where $b(n)$ is the size in bits of frame n.

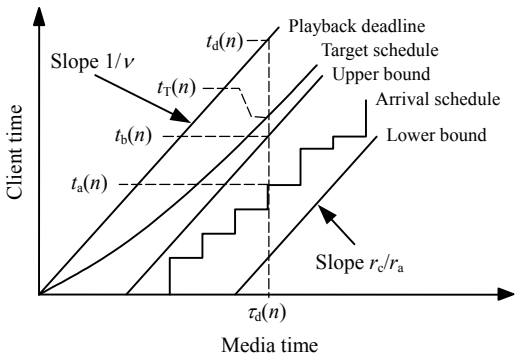

Fig.5 Coding rate control model

The arrival schedule is bounded between a lower and an upper bound representing a leaky bucket (Ribas-Corbera *et al*., 2003) containing the encoded stream. The upper bound increases linearly at slope r_c/r_a. The goal of our control system is to control the upper bound so that it tracks a target schedule sufficiently in advance of the playback deadline. It should be clear that the direction of the upper bound can be changed by changing the coding rate r_c, possibly to compensate for a change in the arrival rate r_a (affected by available network bandwidth). It turns out that the upper bound $t_b(n)$ at frame n evolves according to the linear dynamical system

$$t_b(n+1) = t_b(n) + \frac{r_c(n+1)}{f\tilde{r}_a} + w(n),$$

where $r_c(n+1)$ is the coding rate of frame n+1, $\tilde{r}_a$ is a smoothed estimate of the arrival rate, and f the frame rate. Any deviation caused by using $\tilde{r}_a$ instead of the instantaneous arrival rate r_a in the above equation is captured by the noise term $w(n)$. Thus we can use linear feedback to make the upper bound $t_b(n)$ track a target schedule $t_T(n)$ by controlling the coding rate of a future frame. We also wish to minimize quality variations due to large or frequent changes in the coding rate. This is achieved by designing the feedback gain to minimize the quadratic cost function

$$I = \sum_{n=0}^{N}\left((t_b(n) - t_T(n))^2 + \sigma\left(\frac{r_c(n+1) - r_c(n)}{\tilde{r}_a}\right)^2\right),$$

where the first term penalizes the deviation of the buffer tube upper bound from the target schedule and

the second term penalizes the relative coding rate difference between successive frames. Here, N is the control window size and σ is a Lagrange multiplier or weighting parameter balancing the two terms.

By defining the error state $e(n)=t_b(n)-t_T(n)$ and the control input $u(n)=(r_c(n+2)-r_c(n+1))/\tilde{r}_a$, the linear dynamical system can be expressed in error space as:

$$\boldsymbol{e}(n+1)=\begin{bmatrix}2 & -1 & 1/f\\ 1 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\boldsymbol{e}(n)+\begin{bmatrix}0\\ 0\\ 1\end{bmatrix}u(n)+\begin{bmatrix}1\\ 0\\ 0\end{bmatrix}d(n), \tag{3}$$

where $\boldsymbol{e}(n)=[\boldsymbol{e}(n)\boldsymbol{e}(n-1)u(n-1)]^T$ is a state vector in the error space and $d(n)=w(n)-w(n-1)$. Correspondingly, the cost function becomes:

$$I=\sum_{n=0}^{N}\{\boldsymbol{e}(n)^T\boldsymbol{Q}\boldsymbol{e}(n)+u(n-1)^T Ru(n-1)\}, \tag{4}$$

where $\boldsymbol{Q}=\boldsymbol{C}^T\boldsymbol{C}$ (with $\boldsymbol{C}=[1\ 0\ 0]$) and $R=\sigma$. Thus, the original coding rate control problem can be converted to a standard regulator problem in error space. After an optimal control feedback gain $\boldsymbol{G}^*$ is obtained by solving the corresponding Discrete Algebraic Riccati equation (DARE) (Anderson and Moore, 1990) (letting $N\to\infty$), the ideal coding rate for frame $n+2$ can be computed as (refer to (Huang *et al*., 2005) for detailed derivation and analysis of stability and robustness)

$$r_c(n+2)=r_c(n+1)-\boldsymbol{G}^*\boldsymbol{e}(n)\tilde{r}_a.$$

The term $t_b(n)$ within the error vector $\boldsymbol{e}(n)$ can be estimated as $t_a(n)+g(n)/\tilde{r}_a$, where $g(n)$ is the amount of space left in the leaky bucket after frame n is inserted.

Multiple bit rate streaming

The above coding rate control framework works perfectly for scalable streaming media. But to apply it to MBR streaming, a couple of additional issues need to be carefully addressed.

First, in MBR streaming there are only a limited number of coding rates (usually 5~7) available. This coarse quantization of the desired coding rate introduces a significant nonlinearity into the closed loop system. In fact, the large gaps between the available coding rates introduce oscillations. For example, if two neighboring coding rates straddle a constant arrival rate, the controller will oscillate between the two coding rates in an attempt to keep the client buffer at a target level.

Second, in MBR streaming the coding rate cannot be switched at an arbitrary time. In fact, before the server can switch to a new stream, it must wait for the next switch point (e.g., an I frame) in the new stream, which could be five or ten seconds away. Thus, the old coding rate may continue for quite a while before it changes to the new coding rate. From the controller's perspective, this long random extra delay tends to destabilize the closed-loop system.

Here, we describe a technique to help stabilize the control system and reduce steady state oscillations to a period of at least a minute. With this technique, rapid down-switching is permitted. In fact, we choose a small value of σ, changing the balance between responsiveness and smoothness of the coding rate in favor of a rapid switching response. However, only conservative up-switching is permitted. Conservative up-switching ensures that spurious changes in coding rate do not occur, and that oscillations in the coding rate have a low frequency. In particular, conservative up-switching reduces the oscillations between two adjacent but widely spaced MBR coding rates, one above the arrival rate and one below the arrival rate.

The idea behind conservative up-switching is to establish a conservative limit on how high the coding rate can be raised above the arrival rate. If the current coding rate is below the arrival rate, and the client buffer duration begins to increase above its target level, then the coding rate can be switched up to a new coding rate above the arrival rate only if the new coding rate is below the conservative limit. Given the current client buffer duration, the conservative limit is set to a value such that if the coding rate is switched up to a new coding rate at this value, the client buffer would take at least Δt seconds of client time to drain back to the target level. Thus, the mechanism ensures that the period of oscillation will be at least Δt seconds. In our experiments, we set Δt to be 60 s.

Fig.6 shows how we compute the conservative limit. Let $\Delta\tau_1$ be the client buffer duration (in media time) at the moment that the coding rate is switched up

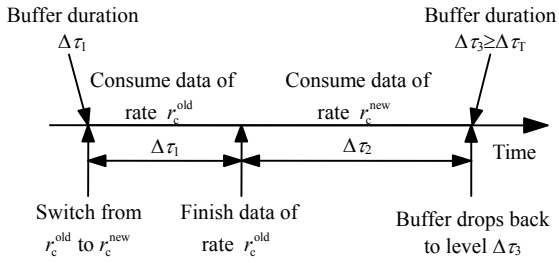

Fig.6 Conservative rate up-switching

from r_c^{old} to r_c^{new}. Thus $\Delta\tau_1$ is the number of seconds of content that will be consumed at the old coding rate r_c^{old} before content at the new coding rate begins to be consumed (For simplicity we assume that all of the content in the client buffer at the time of the switch is coded at rate r_c^{old}). Let $\Delta\tau_2$ be the number of seconds of content that is consumed at the new coding rate r_c^{new} before the client buffer duration drops to some level $\Delta\tau_3$ seconds (in media time), greater than the target level $\Delta\tau_T$. The duration of this phase is determined such that the total time since the switch is exactly $\Delta t=(\Delta\tau_1+\Delta\tau_2)/v$ seconds (in client time). Now, the number of bits that arrive at this time

$$\begin{aligned} r_a\Delta t &= r_c^{new}(\Delta\tau_2+\Delta\tau_3) \\ &\geq r_c^{new}(\Delta\tau_2+\Delta\tau_T) \\ &= r_c^{new}(v\Delta t-\Delta\tau_1+\Delta\tau_T), \end{aligned}$$

or

$$r_c^{new} \geq \frac{r_a\Delta t}{v\Delta t-\Delta\tau_1+v\Delta t_T}, \tag{5}$$

where Δt_T is the target buffer duration in client time. The parameter Δt can be tuned to yield the desired behavior. A large Δt indicates that up-switching is more conservative, while a smaller Δt indicates that up-switching is more prompt. In our implementation, Δt is set to 60 s while the target Δt_T is typically about 10 s. This improves controller stability for MBR streaming.

Combining ORC with MutualCast

ORC can be easily implemented as a black-box at the client, which exposes mainly two input interfaces and one output interface. One interface gets input from the client network component and is triggered whenever a new network packet is received. ORC uses this information to compute the average arrival rate $\tilde{r}_a$. The other interface gets input from the client application and is triggered whenever a complete new frame is received. Then, ORC runs the aforementioned algorithm and outputs a desirable MBR rate. The MBR rate is feedback to the server, which in turn switches to the corresponding bit stream at the next switching point.

From the above description, it is clear that ORC can be combined easily with the MutualCast delivery scheme. From ORC's perspective, the source node has the media content and certainly functions as the server. The rest of the MutualCast network then can be visualized as a giant client, and the redistribution queue at the source node can be imagined as the time varying network connection between the server and the client. When a new packet is inserted into the redistribution queue, it triggers ORC. And when a complete new frame is inserted into the redistribution queue, it also triggers ORC, which then outputs a desirable MBR rate. At the next switching point, a stream of corresponding bit rate will be selected by the source node and the quality to all the peer nodes is affected afterwards. Note that in this case, ORC is physically running at the source node, thus the quality adaptation will only be affected by the system wide throughput, not directly by the number of the peer nodes.

Additional startup delay

Using redistribution queues can cause additional startup delay, which is equivalent to the time taken to fill the buffer in the queues. Imagine one node is sending data to another with a limited upload bandwidth, then the additional startup delay equals the amount of time taken to fill the sender buffer. Note that the receiver buffer will remain empty assuming unlimited download bandwidth. Let the sender buffer size be K and the upload bandwidth B, then the additional startup delay is

$$t_d = K/B. \tag{6}$$

In the MutualCast network, some data blocks are duplicated and delivered to all peer nodes directly by the source node (when $B_s>B_{s1}+B_{s2}$ in Eq.(1)). The additional startup delay inccurred by these data blocks can be computed using Eq.(6). Other data blocks, however, are redistributed by the peer nodes

and reach their destination through the two-tier trees. The intermediate nodes in these trees have limited upload bandwidths. For those data blocks, the additional startup delay equals the time taken to fill the source sender buffer, the intermediate node's receiver buffer, and its sender buffer to the destination node as well. Assuming all buffers are the same size, then t_d=3K/B. Considering all possible routes for the data blocks: (1) directly from the source; (2) from the other content-requesting peer nodes; and (3) from the non-content-requesting peer nodes, we can get the additional startup delay as

$$t_d=\begin{cases}\max\left(\dfrac{3K}{B_1}(N_1-1),\dfrac{3K}{B_2}N_1\right), & B_s\le B_{s1}+B_{s2},\\ \max\left(\dfrac{3K}{B_1}(N_1-1),\dfrac{3K}{B_2}N_1,\right. & \\ \left.\dfrac{K}{B_s-(B_{s1}+B_{s2})}N_1\right), & B_s>B_{s1}+B_{s2}.\end{cases} \quad (7)$$

This really is an interesting and unintuitive result showing that although we can limit the buffer in the redistribution queues, the additional startup delay still increases linearly in terms of the peer nodes number. This indeed is another limiting factor constraining the size of the MutualCast network.

PERFORMANCE

We evaluate the performance of adaptive peer-to-peer streaming in the MutualCast network composed of 1 source and 4 peer nodes. The media file distributed is an MBR file containing a 20-minute clip of the Matrix coded at five different combinations of audio and video bit rates, as listed in Table 1, using a 5-second leaky bucket for each coding rate. The upload bandwidths of all peer nodes are shown in Fig.7, which vary randomly every second but also follow a general trend of decreasing, increasing and then stabilizing. Using Eq.(1), we compute the distribution capacity of the MutualCast network shown in Fig.8. Correspondingly, the measured throughput at the source node is also shown in Fig.8, which matches the optimal throughput quite well. This again confirms that the MutualCast network achieves the distribution capacity. Note that the measured throughputs are a bit more fluctuating, which is reasonable considering the much more severe bandwidth fluctuations of all the peer nodes.

Fig.9 shows the receiving rate, reception quality and buffer status at one of the peer nodes, which is quite typical among all the peer nodes. It is clear that users' expectations are satisfied with no rebuffering, maximal quality and smoothness over the entire session. Indeed, Fig.9 shows that the coding rate (and hence the quality) is as high as possible given the average arrival rate, except during the first 15 s or so, in which the coding rate is lower than the arrival rate

Table 1 Bit rates in MBR file

Audio (kbps)	Video (kbps)	Audio+Video (kbps)
32	32	64
32	64	96
32	189	221
32	314	346
32	464	496

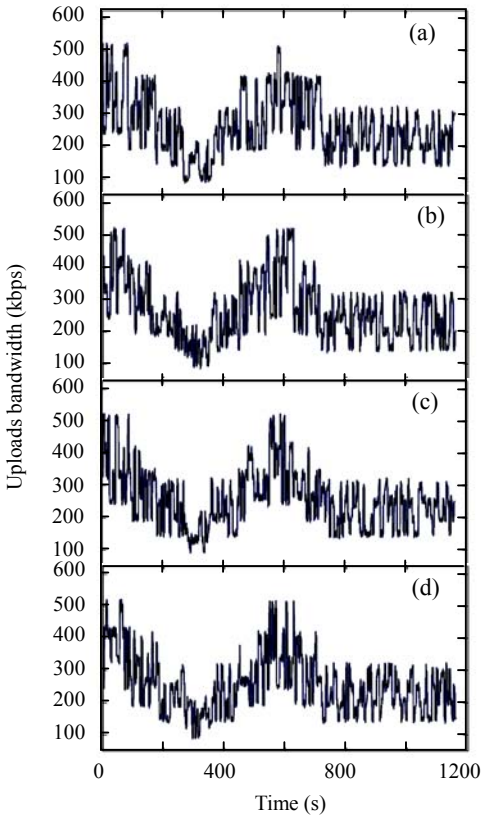

Fig.7 Upload bandwidth at receiver nodes. (a) Node t_1; (b) Node t_2; (c) Node t_3; (d) Node t_4

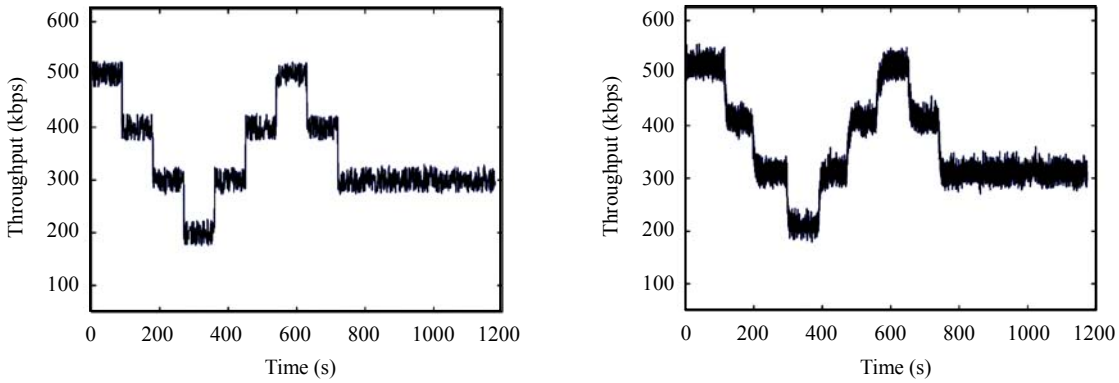

Fig.8 MutualCast system wide throughput. (a) Theoretical source throughput; (b) Measured source throughput

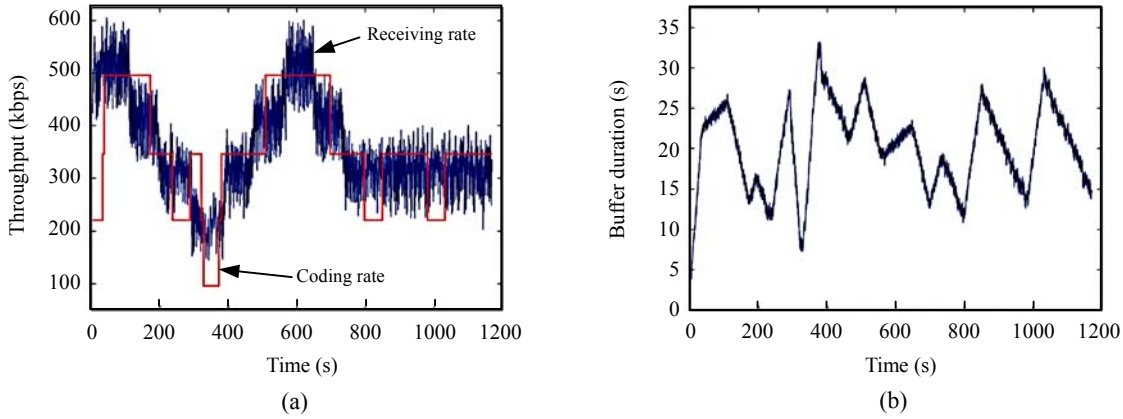

Fig.9 Throughput (a) and buffer duration (b) at receiver nodes

to build up the client buffer without incurring a large startup delay. Smoothness is also achieved, since the coding rate does not change spuriously, dropping only when the client buffer falls below its target and rising only when it can sustain the higher bit rate for at least 60 s in steady state. Correspondingly, Fig.9 shows that after the initial 15 s, the buffer duration hovers between 5 and 35 s, and does not underflow. Note that the receiving rate at the peer node is even more fluctuating than the source node throughput, although their long term average is the same. Also note that these fluctuations at the peer node do not affect the reception quality determined at the source node. As long as the peer node buffer does not underflow, continuous playback is guaranteed.

SUMMARY

In this paper, we describe the MutualCast framework, a simple multi-tree ALM scheme that quickly adapts to heterogeneous and time-varying networks, while achieving provably optimal throughput performance. Combining it with ORC, a control-theoretical framework for quality adaptation, we study how the adaptivity of MutualCast can be paired with the adaptive rate control for streaming media. Using multiple bit rate video content, we show that the proposed system can smoothly adjust the common quality received at all the nodes while maintaining a continuous streaming experience at each, even when the network undergoes independent and severe bandwidth fluctuations.

References

Anderson, B.D.O., Moore, J.B., 1990. Optimal Control: Linear Quadratic Methods. Prentice Hall.

Bharambe, A., Herley, C., Padmanabhan, V., 2005. Understanding and Deconstructing Bittorrent Performance. Microsoft Research Technical Report, MSR-TR-2005-05.

Birney, W., 2003. Intelligent Streaming. Available at http://www.microsoft.com/windows/windowsmedia/howto/

articles/intstreaming.aspx.

Castro, M., Druschel, P., Kermarrec, A.M., Rowstron, A., 2002. SCRIBE: a large-scale and decentralized application-level multicast infrastructure. *IEEE J. Selected Areas in Communications*, **20**(8):1489-1499. [doi:10.1109/JSAC.2002.803069]

Castro, M., Druschel, P., Kermarrec, A.M., Nandi, A., Rowstron, A., Singh, A., 2003. Splitstream: High-Bandwidth Content Distribution in a Cooperative Environment. Proc. the International Workshop on Peer-to-Peer Systems. Berkeley, CA.

Chawathe, Y., 2000. Scattercast: An Architecture for Internet Broad-Cast Distribution as an Infrastructure Service. Ph.D Thesis, University of California, Berkeley.

Cherkasova, L., Lee, J., 2003. FastReplica: Efficient Large File Distribution within Content Delivery Networks. Proc. the 4th USENIX Symposium on Internet Technologies and Systems. Seattle, WA.

Chu, Y.H., Rao, S., Zhang, H., 2000. A Case for End System Multicast. Proc. ACM Sigmetrics. Santa Clara, CA.

Cohen, B., 2003. Incentives Build Robustness in Bittorrent. Proc. Workshop on Economics of Peer-to-Peer Systems. Berkeley, CA.

Conklin, G., Greenbaum, G., Lillevold, K., Lippman, A., Reznik, Y., 2001. Video coding for streaming media delivery on the Internet. *IEEE Trans. Circuits and Systems for Video Technology*, **11**(3):269-281. [doi:10.1109/76.911155]

Gkantsidis, C., Rodriguez, P., 2005. Network Coding for Large Scale Content Distribution. Proc. Conf. Computer Communications (INFOCOM). Miami, FL.

Huang, C., Chou, P.A., Klemets, A., 2004a. Optimal Coding Rate Control for Scalable Streaming Media. Proc. Int'l Packet Video Workshop. Irvine, CA.

Huang, C., Chou, P.A., Klemets, A., 2004b. Optimal Control of Multiple Bit Rates for Streaming Media. Proc. Picture Coding Symposium. San Francisco, CA.

Huang, C., Chou, P.A., Klemets, A., 2005. Optimal Coding Rate Control for Scalable and Multi Bit Rate Streaming media. Microsoft Research, Redmond, WA, Tech. Rep. MSR-TR-2005-47.

Jannotti, J., Gifford, D.K., Johnson, K.L., Kaashoek, M.F., O'Toole, J.W.Jr, 2002. Overcast: Reliable Multicasting with an Overlay Network. Proc. the 4th Symposium on Operating System Design and Implementation (OSDI). San Diego, CA.

Kostic, D., Rodriguez, A., Albrecht, J., Vahdat, A., 2003. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. Proc. 19th ACM Symposium on Operating Systems Principles. Bolton Landing, New York.

Li, J., Chou, P.A., Zhang, C., 2005. MutualCast: An Efficient Mechanism for One-to-Many Content Distribution. Proc. ACM SIGCOMM ASIA Workshop.

Padmanabhan, V.N., Wang, H.J., Chou, P.A., 2003. Resilient Peer-to-Peer Streaming. Proc. Int'l Conf. Network Protocols. Atlanta, GA.

Pendarakis, D., Shi, D.V.S., Waldvogel, M., 2001. ALMI: An Application level Multicast Infrastructure. Third USENIX Symposium on Internet Technologies and Systems (USITS).

Ratnasamy, S., Handley, M., Karp, R., Shenker, S., 2001. Application-Level Multicast Using Content-Addressable Networks. Proc. the Third International COST264 Workshop (NGC 2001). London, UK.

Ribas-Corbera, J., Chou, P.A., Regunathan, S.L., 2003. A generalized hypothetical reference decoder for H.264/AVC. *IEEE Trans. Circuits and Systems for Video Technology*, **13**(7):674-687. [doi:10.1109/TCSVT.2003.814965]

Wang, B., Kurose, J., Shenoy, P., Towsley, D., 2004. Multimedia Streaming via TCP: An Analytic Performance Study. Proc. Int'l Conf. Multimedia. New York City.

Zhang, X., Liu, J., Li, B., Yum, T.S.P., 2005. DONet/CoolStreaming: A Data-Driven Overlay Network for Live Media Streaming. Proc. Conf. Computer Communications (INFOCOM). Miami, FL.

Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R.H., Kubiatowicz, J., 2001. Bayeux: An Architecture for Scalable and Fault-Tolerant Wide-Area Data Dissemination. Proc. Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV).