# Network-Adaptive Video Coding and Transmission

Kay Sripanidkulchai and Tsuhan Chen
Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213
<kunwadee@andrew.cmu.edu> <tsuhan@ece.cmu.edu>

## ABSTRACT

In visual communication, the conditions of the network, such as the delay, delay jitter, and the packet loss rate, have strong impact to the video quality. It would be useful if there is a feedback channel from the client to the server to indicate the network conditions periodically, and a smart mechanism for coding and transmitting video that can adapt to these conditions. For example, when the network is congested, instead of sending all the packets that have a high probability of being lost in the network, we can selectively drop some packets at the server (such as dropping packets for bidirectional-predicted frames). While intuitive, it is difficult to illustrate the effectiveness of adaptation using a single video server-client pair. A practical simulation would require multiple video servers and clients, and only then the benefit of adaptation will show up as advantageous utilization of network resources to provide good video quality. In this paper, we will introduce our methods of adaptation and present experimental and simulation results.

Keywords: video on demand, network transmission, adaptive congestion control, MPEG encoding

## 1. INTRODUCTION

The technology for delivering ubiquitous high bandwidth multimedia services, such as voice with video, will soon become a reality. Many service applications are being developed to take advantage of the increasing bandwidth. One such application is Video on Demand (VoD). A major challenge in providing VoD in today's Internet is transmitting real time video streams across a heterogeneous best effort network while trying to achieve acceptable perceptual quality. Because the video stream is typically coded at a variable bit rate, the bandwidth requirement is variable. The interarrival time for consecutive frames must lie within specified delay bounds in order for the frames to be useful. Minimal transmission delays are tolerable, but long delays and delay jitters could cause jerky and discontinuous motion. Due to the uncertain nature of network load and routes, packets could have delays, or arrive out of order. When the network is congested, packets could even be dropped. Unpredictable packet losses due to congestion inside the network could seriously degrade video quality. Schemes to recover from packet losses via error correction and retransmission exist, but they often add on excessive overhead (e.g. Forward Error Correction) or more undesirable delays. In this paper, instead of using mechanisms to ensure reliable transmission of video packets and improve error resilience, we will focus on end to end adaptation to network congestion, which can be used in conjunction with error recovery.

The VoD server should respond to network congestion by adapting the rate at which it injects packets into the network. As the network becomes more congested, the server should reduce its transmission rate. If all applications ignored network conditions, or increased transmission rate upon congestion, the network would come to a halt, causing congestion collapse. Typically, video is transmitted using the User Datagram Protocol (UDP), which treats each video packet independent of each other, provides no service guarantees and no feedback to the sender.

In order to improve the received video quality, there are two possible approaches. The first one, an end to end approach, is to design and implement network aware VoD servers and clients. The server utilizes feedback information from the client about the network connection and the client's reception to adjust its transmission rate. There is a question of whether the feedback of previous performance and network conditions will be a sufficient predictor of the future and will be relevant enough to be used by the server. We also need to determine what kind of feedback would be useful.

The second approach is to have intermediate nodes in the network (i.e. active routers) that understand the semantics of the VoD application that will assist in adjusting transmission to provide acceptable quality of service. For example, for a VoD system using MPEG encoding, when the network is congested, the active router will try to avoid dropping I and P frames and will drop only B frames. This capability is not available in today's Internet.

In this paper, we will discuss the first approach. In Section 2, we will give an overview of a network aware adaptation protocol. In Section 3, we will present the implementation of the VoD system. Experimental and simulation results are presented in Section 4. We conclude our findings and discuss future work in Section 5.

## 2. NETWORK AWARE TRANSMISSION ALGORITHM

To cope with network congestion, it would be useful if the VoD server and client maintain a feedback channel. To inform the server of network status, the client can send its application level packet reception information back to the server. This information can include such information as loss rates, delay, delay jitter and perceived video quality. In this paper, we focus on VoD systems that serve MPEG-encoded video, but the results can be easily extended to other video coding standards.

### 2.1 NETWORK AWARE SERVER

In order to adapt to changing network conditions, the VoD server should be capable of changing its transmission rate. For server scalability, the amount of processing at the server should be minimal. Because a client knows its reception quality and the network congestion status of the route from the server, the client is in the best position to control the quality of video to be transmitted to it. This is accomplished by having the client send feedback in the form of control packets. These control packets determine which MPEG frames the server should transmit. The client specifies which mode the server should be in. For an MPEG video, we propose 4 modes as follows:

1. SEND_ALL_PACKETS
2. SEND_ALL_I_AND_P_AND_ONLY_X_PERCENT_OF_B_FRAMES
3. SEND_ONLY_I_AND_P_FRAMES
4. SEND_ONLY_I_FRAMES

These four modes provide sufficient flexibility and extensibility to control the degradation of the video stream. Although this scheme does not take into account the server load, it can be modified such that it does. That is, if all the clients requested the server to SEND_ALL_PACKETS and the server is overloaded, the server can deny some of the SEND_ALL_PACKETS request. The exact policy decision does not need to rely on the feedback scheme and can be specified independently.

### 2.2 NETWORK AWARE CLIENT

The feedback mechanism should be lightweight and not require excessive processing at both the client and the server. As mentioned above, we choose to let the client make all the decisions and inform the server what to transmit. This allows for server scalability. These are two sub-components to the feedback mechanism:

- Network Congestion Estimation Algorithm: The client determines the status of the network by keeping track of the number of lost packets within a certain interval. Based on the estimated network congestion, the client decides the server's transmission mode.
- Feedback Control Protocol: At the end of each time interval, the client sends out a control packet to the server stating which mode packets should be transmitted.

## 3. IMPLEMENTATION

In order to evaluate the proposed network aware transmission algorithm proposed in Section 2, we implemented a VoD system using RTP$^2$/UDP as the transmission protocol. RTP is appropriate for carrying real time information. In addition, it does not provide any guarantees and leaves such issues to the application level to choose suitable mechanisms. The client is capable of real time decoding of MPEG video. The issues of error resiliency and adapting to packet losses will be investigated on the client side. The VoD system, consisting of a server and client, is depicted in Figure 1. We will proceed to discuss the implementation of each depicted component.

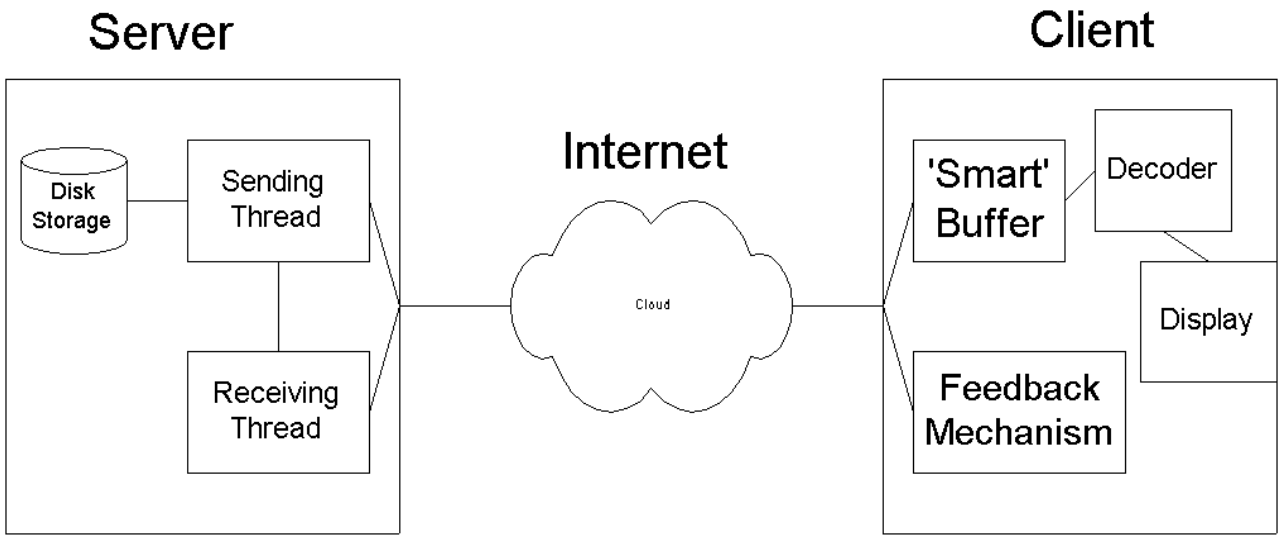


Figure 1: Video on Demand system

### 3.1 SERVER

Because we are only interested in the feedback mechanism, a simple server was implemented. The server is composed of two threads simultaneously executing:

#### 3.1.1 SENDING THREAD

The sending thread performs three key functions:

- Read in the MPEG bitstream from disk.
- Parse the bitstream, packetize, and add RTP MPEG-specific headers according to the specifications in RFC 2250[1]
- Send the bitstream in real time.

The encoded MPEG bitstream is assumed to be pre-coded and stored on disk. The sending thread reads the bitstream into memory, in chunks. It then, parses the bitstream and packetizes it. Although we use the same headers as the specification in RFC 2250, the meaning of the headers are different. We also packetize the bitstream such that no two frames are transmitted over the network in the same packet. This is to add extra error resilience so that one packet loss results in at most one frame lost. To avoid further packet fragmentation at the IP level, we choose the maximum packet size (including RTP/UDP headers) to be at most the size of an Ethernet packet, which is 1500 kB. Therefore, a frame can be split over several packets. [Kay, 1500 KB is much larger than the size of one frame, so why can one frame span over multiple packets?]

In RFC 2250, these four fields are used to specify the prediction mode of each frame. In our implementation, we used the fields to specify the fragment number or the position of this packet with respect to other packets that belong to the same frame. This modified version is useful for determining packet losses: where the last packet from the previous frame ended and where the first frame of the next packet started. Thus, if we have lost a packet that belongs to a frame, we know that we can throw away successive incoming packets belonging to the same frame and discard packets that have previously arrived from the client's buffer. The buffer implementation at the client will be discussed later.

Once we have parsed a frame and packetized it, we send it out to the client. The transmission is spaced out and smoothed as much as possible to avoid bursts. Bursty transmission is generally not a desired behavior because it may cause transient congestion states in the network.

The sending thread algorithm in pseudocode is shown in Figure 2.

```
While (not_done) {
      Read_a_frame_of_bits;
      While (bits_left) {
            get_packet_worth_of_data;
            add_MPEG_specific_headers;
            add_RTP_headers;
            /* to smooth out transmission */
            wait_till_time_to_send;
            send_packet;
      }
}
```

Figure 2: Sending thread algorithm in pseudocode

### 3.1.2. RECEIVING THREAD

The receiving thread reads incoming control packets sent from the client. These control packets are feedback packets that inform the server of the client's reception state since the last received control packet. Once a control packet is received, the server changes its sending 'mode' accordingly. To allow for server scalability (one server accommodating multiple clients simultaneously), we move the burden of choosing which mode to change to from the server to the client. Pseudocode for the receiving thread is depicted in Figure 3. The feedback information changes the sending thread's operation as depicted in the pseudocode in Figure 4.

```
While (1) {
      receive_control_packet();
      change_mode;
}
```

Figure 3: Receiving thread algorithm in pseudocode

```
while (not_done) {
      read_a_frame_of_bits;
      if (frame_type_can_be_sent_in_this_mode) {
            while (bits_left) {
                  get_packet_worth_of_data;
                  add_MPEG_specific_headers;
                  add_RTP_headers;
                  /* to smooth out transmission */
                  wait_till_time_to_send;
                  send_packet;
            }
      }
}
```

Figure 4: Sending thread algorithm in pseudocode with modifications to allow for feedback control.

## 3.2 CLIENT

For a network aware client, we modified Berkeley's mpeg_play to support RTP, and our network aware transmission algorithm. The client is composed of 4 components.

1. Display/Renderer: modified from UC Berkeley's mpeg_play unix utility.
2. Decoder: modified from UC Berkeley's mpeg_play unix utility to support real time decoding. The interface between the decoder and the client's 'smart' buffer is similar to that of a filesystem. That is the decoder can issue a read command that will take out n bytes from the 'smart' buffer. We choose this approach because the Berkeley mpeg_play implementation assumes filesystem-like operations. We also modified the decoder so that it does not need to read the whole MPEG bitstream into memory before starting to decode. This allows for piece-meal real time decoding (frame by frame).
3. 'Smart' Buffer: Initial buffering of the stream (before any decoding) is needed so that delays can be absorbed. This amount of buffering should be enough so that it can absorb jitters without taking up too much memory resources. Packets arriving out of order will be ordered at the buffer and, then, passed on to the decoder. Also, the buffer management scheme should be efficient in that if a packet is loss, subsequent packets that rely on the lost one should not be stored. For example, if a P frame is lost, newly arrived B frames that rely on that P frame should be dropped. It is also possible to have the buffer do some form of interpolation for the lost frame for better video quality. Although the last two features are in conflict with each other, both have positive impact. Finding a balance between the two is important for an error resilient client. This buffer is located at the interface between the socket buffer and the decoder. It receives incoming MPEG bitstream packets from the network and temporarily stores them and delivers them to the decoder upon request. The 'smart' buffer is implemented with the following efficient buffer management algorithm.
   - If a packet belonging to a frame is lost, it will throw away all packets belonging to the same frame.
   - If a packet belonging to an I-frame is lost, all packets are discarded until the buffer sees a new I-frame.
   - If a packet belonging to an I-frame is lost, all packets are discarded until the buffer sees a new I-frame. [Kay, the last two are the same?]

   Thus, the buffer is not wasting any of its capacity storing bits that cannot be decoded. The size of the buffer should be set such that it is large enough to smooth out jitters that can occur during transmission, but not too large to cause delayed start-up of decoding and displaying. An optimal size is a little larger than the bandwidth-delay product in the system, where bandwidth is the bandwidth of the link and delay is the time it takes for a packet to leave the server and arrive at the client.
4. Feedback Mechanism: The frequency at which these control packets are sent also determines the success of adaptation. We cannot send these packets too frequently or else we would be using up a significant amount of bandwidth. However, if we do not send these packets frequent enough, we are risking the possibility of out-of-date feedback. That is, if these control packets are send every 5 minutes, then the server is adapting based on old and, most likely, irrelevant network status. We find that control packets should be sent at least twice every second in order to achieve reasonable adaptation. We will discuss in detail the implementation of one of the two sub-components of the feedback mechanism: network congestion estimation algorithm.

### 3.2.1 NETWORK CONGESTION ESTIMATION ALGORITHM

The client estimates the state of congestion in the network based on packet losses. It keeps a penalty box, which is cleared after n packets have been received since the last control packet was sent. Based on the assumption that all packets take the exact same route from the server to the client, there is no packet reordering. Thus, if we see a jump in sequence numbers, we can assume that the packets with missing sequence numbers were lost. For each packet lost, we increment the penalty value by 1. For each frame lost, we increment the penalty value in a weighted manner. That is for each I-frame lost, we increment penalty by 4. For each P-frame lost, we increment the penalty by 3. And, for each B-frame lost, we increment the penalty by 1. Weights should be determined by the typical size of each frame type. Once we have received enough packets to send out a feedback packet, we decide which mode we would like the server to be in according to the penalty value. Table 1 shows the values that were used in this implementation. The feedback packet is then formatted according to the control protocol (which will be discussed later) and sent to the server. If the decision is to have the server send fewer packets, the feedback interval is also reduced to half. The penalty box in now cleared and we start the process all over again. These are many possible optimizations and modifications to the algorithm, for example,

- Have a more precise way of defining the feedback interval so that it actually relies on time instead of the number of packets received. For the latter, if a large number of packets were lost around the same time, the feedback interval (based on number of received packets) would be stretched out and any feedback to the server would become obsolete.

- The mapping between penalty points and server sending mode should be systematically defined based on experimental data.
- Penalty points are solely based on the transient state of the network (the number of packets lost during the feedback interval). It may be an advantage to factor in previous intervals using a form of exponential averaging, or to give different weights to the penalty points depending on the time relevance. For example, a packet lost during the first millisecond of the feedback interval should carry less weight than a packet lost during the last millisecond of the feedback interval.

| Sending Mode | Penalty Points p (as a percentage of the number of packets in a feedback interval) |
|---|---|
| SEND_ALL_PACKETS | $p \leq 10\%$ |
| SEND_ALL_I_AND_P_AND_ONLY_X_PERCENT_OF_B_FRAMES | $10\% < p \leq 50\%$ |
| SEND_ONLY_I_AND_P_FRAMES | $50\% < p \leq 70\%$ |
| SEND_ONLY_I_FRAMES | $p > 70\%$ |

Table 1: Penalty points at client determine sending mode at the server.

## 4. EVALUATION

We will now describe our experimental setup and results. We evaluated a network aware system with a non-network aware system. The non-adaptive system was composed of a similar client and server, but the client was not running a feedback algorithm and did not send any feedback information to the server. Two methods of evaluation were carried out: experiments over an actual network, and simulation experiments.

The key factor in determining perceptual quality is the distribution of frames that were decodable on the client side. The number of I-frames that were decodable (i.e. useful) was given the most weight. Subsequently, the number of P-frames and B-frames that were decodable were given less weight. Also in terms of network performance, the ratio of decodable data over all data sent over the network for both systems were compared.

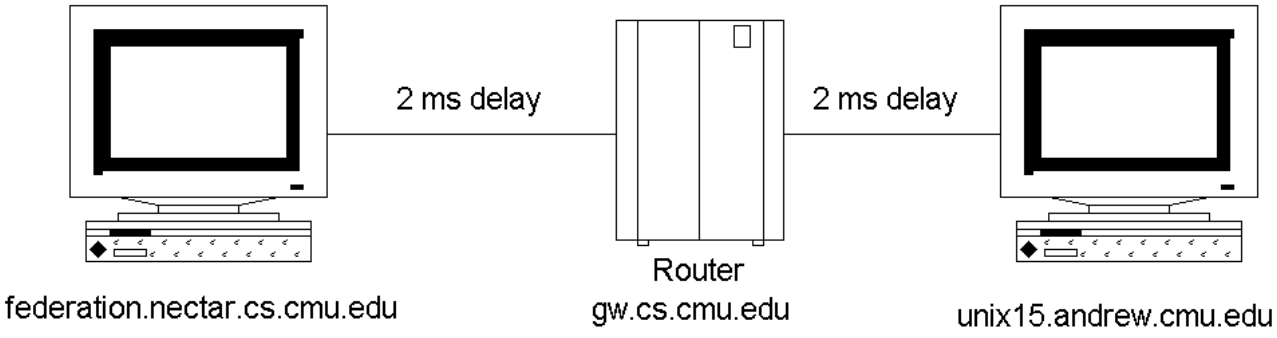


Figure 5: Experimental testbed configuration

### 4.1 EXPERIMENTAL RESULTS

The experimental testbed is shown in Figure 2. We transmitted our test bitstreams between two subdomains within the Carnegie Mellon University's network. The route between the testbed machines was not a dedicated path. It is shared with other users. Therefore, background traffic is actual network traffic generated by users. Packet losses and frame losses were analyzed.

There was no significant difference in the perceived quality of the transmitted video in both systems. Although, intuitively, the adaptive system should have higher video quality, in some cases it did worst than the non-adaptive system. This could be because the adaptive system performs well only under severe congestion. Since the testbed is from the andrew domain to the cs domain within the local area network of cmu, we did not see the severe congestion and network usage behavior that is present in wide area networks, such as the Internet. Also, we only evaluated one end-to-end adaptive system. In order to see the overall improvement of the adaptation, we need to take into account other traffic that is sharing the same network links.

There were, however, significant improvements in network resource utilization. As we can see from Table 2, for the P and B frames, we see that a higher percentage of usable packets and/or frames from the adaptive system. This means that although we are getting similar visual quality, we would be reducing the actual bandwidth used to transmit the video.

The reason the adaptive system had higher loss with the I-frames is that it just happened that in this experiment, 3 more packets belonging to 3 different I-frames were dropped.

| | Network Aware System | Non-network Aware System |
|---|---|---|
| Decodable % of transmitted I frames | 87.80% | 92.70% |
| Decodable % of transmitted P frames | 70.89% | 67.90% |
| Decodable % of transmitted B frames | 76.02% | 66.33% |
| Decodable % of transmitted I packets | 80.85% | 97.07% |
| Decodable % of transmitted P packets | 71.99% | 60.29% |
| Decodable % of transmitted B packets | 80.50% | 57.70% |

Table 2: Results from real network experiments

### 4.2 SIMULATION RESULTS

Two loss patterns due to congestion were simulated. The two patterns are uniformly distributed losses of 10% of packets, and step function (the lost rate is 0%, and then jumps suddenly to10%).

#### 4.2.1 UNIFORMLY DISTRIBUTED LOSS OF 10%

The visual quality of the video in the adaptive system, surprisingly, is lower than the non-adaptive system. This is because our network model of uniformly distributed loss is not realistic. For example, if the server adapts and sends out only I and P packets, those packets have an equal likelihood of being dropped. However, if the server is sending B packets, then all three packet types I, P, and B have equal likelihood of being dropped. So in the non-adaptive case, if a B packet is dropped the quality degradation is not severe. In the adaptive case, if an I or P packet is dropped (because the server adapts and sends no B packets), then the visual quality could dramatically decrease. In a real network, it is expected that if the server reduces its transmission rate, the network should not be as congested and the packet loss rates should go down. In our simulated run, our adaptive system had such poor performance that it could not decode any of the last 200 frames (from 1210 frames).

#### 4.2.2 STEP FUNCTION LOSS OF 10%

In this case, we can see that the adaptive system outperforms the non-adaptive system. We transmit fewer packets in the adaptive system and see fewer losses. The visual quality, however, was worse for the adaptive case because there were fewer B-frames to decode. With fewer B-frames, we get jerky discontinuous motion.

| | Network Aware System | Non-network Aware System |
|---|---|---|
| Decodable % of transmitted I frames | 65.90% | 60.10% |
| Decodable % of transmitted P frames | 50.00% | 46.90% |
| Decodable % of transmitted B frames | 71.26% | 47.10% |
| Decodable % of transmitted I packets | 62.23% | 55.10% |
| Decodable % of transmitted P packets | 43.42% | 39.10% |
| Decodable % of transmitted B packets | 64.89% | 39.51% |

Table 3: Results from simulate network congestion (step function)

## 5. CONCLUSION AND FUTURE WORK

We have shown how a network aware adaptation algorithm improves the percentage of decodable packets received at a client. We successfully avoid sending packets that would have otherwise been dropped in the network due to congestion. The server and client pair running our network aware algorithm is capable of reacting to changing network conditions. Using lower transmission bandwidth, we can effectively achieve better perceptual quality. To continue with further evaluation of our network aware adaptation approach, we need to look at the interaction between our adaptive flow and other network flows sharing the same routers.

## REFERENCES


1. D. Hoffman, G. Fernando, V. Goyal, M. Civanlar, "RTP Payload Format for MPEG1/MPEG2 Video", STD 1, RFC 2250, January 1998.
2. H. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 1, RFC 1889, January 1996