AN END-TO-END ADAPTATION ALGORITHM FOR BEST EFFORT VIDEO DELIVERY OVER INTERNET

BY

WALTER CHI-WOON FUNG

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF PHILOSOPHY

DIVISION OF INFORMATION ENGINEERING

THE CHINESE UNIVERSITY OF HONG KONG

JUNE 1998

Acknowledgement

I would like to dedicate this thesis to my parents. I would also like to thank my supervisor Dr Soung C. Liew for his enlightment and guidance, and the many opportunities he has given me. I also want to thank all my colleagues, including Patrick Wu, Alan Yeung, Soung Y. Liew, Li Ngai, Thomas Kwok and Hanford Chan (in random order) for their support over the years. They make the study here a joyful experience.

論文摘要

在互聯網上建基於 TCP, UDP 和 HTTP 的實時多媒體應用程式的數目正不斷增加。這些程式遇到的其中一個問題是不同頻道之頻寬之不一性以至同一頻道在不同時間頻寬之不一性。寬頻與窄頻之連結同時存在, 而且一個頻道上可用的頻寬在不同時間會由有不同需求的連結分享。

很多現存的系統並不會改變傳送率以配合網絡或系統的容量。這些系統會預先假設一個不變的頻道容量。對於不同容量的頻道, 這些系統會使用不同的媒體檔案.

部份會配合網絡容量的設計則對其所在之系統有不同的要求。例如要求伺服器的有相當強大的運算能力, 或者要求網絡的傳送層支援優先權的控制。然而, 這些都並不是現時互聯網或者大部份應用以太網技術的局域網(LAN)的性質。

在這份研究中, 我們提出一個客戶端影片圖片傳送率的配合系統。這個系統可以建立於可靠的運輸協定上, 例如 TCP, 或者在沒有傳送保證的運輸協定上, 例如 UDP。 可擴充的視像自選應用程式可以利用這個系統傳送 MPEG-1 影片而無需煩重的計算。我們指出這個系統對於在網絡上傳送自行調節的視訊是可行的。當視訊封包的來回延誤相近時, 不同的連結能公平地分享可用的頻寬。

總結來說, 這份論文描述了一個能在互聯網上傳送連續視訊而自動調節系統容量的方法。這份建議可以容易擴充, 而且可以在今日所用的網絡科技上應用。

Abstract

The number of real-time multimedia applications over the Internet built on top of TCP, UDP or HTTP is growing. One of the problems faced by these applications is the variation in connection bandwidth and the time varying nature of the Internet, where broadband (such as those via LAN) and narrowband (such as those via modem or ISDN) connections coexist, and the available bandwidth on each channel may be shared among different connections at different time.

Many existing systems do not adapt the transmission rate to the network or system capacity. They simply assume a fixed channel rate beforehand. For different channel capacity, different sets of video files are used.

Some system that adapt dynamically to the system capacity impose certain stringent assumptions. They either require heavy computational efforts on the server, or they require the underlying network to have some kind of support, such as priority control, which is not the case for the current Internet or most of the LAN that use Ethernet technology.

In this study, we propose a client-pull frame-rate adaptation scheme for best-effort video transmission. It can be built on top of a reliable transport protocol such as TCP, or on transport protocol with no delivery guarantee, such as UDP. Scalable video-on-demand applications can be built with this scheme to stream

MPEG-1 video without heavy computational efforts. We show that this scheme is a feasible means to adapt video traffic over networks which provide best effort performance guarantee. When round trip times of the packets of the network video connections are similar, these connections share the available bandwidth in a fair manner. Applications using this algorithm on top of UDP interact well with other network applications that uses TCP in that the algorithm will not be deprived of its share of bandwidth.

In summary, in this thesis describes a scheme for continuous video-transmission over the Internet that adapts to the system capacity without assuming extra support from underlying network layer such as QoS guarantee or priority control. The proposed scheme is scalable and is ready to be deployed with the network technologies in use today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Multimedia delivery systems on the Internet are an area of interest with the growing popularity of graphical user interfaced network applications such as the the world wide web. Publishing on the web, or using web as an interface for Intranet applications become very common. To enrich the contents, many webpages are embedded with multimedia clips. For realtime streaming of multimedia contents, such as audio and video, various multimedia server/client systems have been proposed. A problem faced by these systems is the heterogeneous nature of the Internet, in which some clients are connected to the server through a broadband channel, while others are connected through dial-up modems. Within a network, the sustainable bandwidth of a particular path also changes throughout the course of the day [1]. To dynamically adjust the data rate to the available bandwidth of the connection [9], different schemes have been proposed.

One class of schemes adjusts the source encoding rate. Using feedforward or feedback information from the client, the server dynamically adjusts the encoding threshold of the source to achieve a data rate that can be supported by the network [10, 13, 14]. The effect of changing the encoding threshold is that picture quality will vary in accordance with the data rate. The frame rate is kept constant to maintain the smoothness, but each frame may look blurry or blocky.

Another class of schemes uses multi-layered coding [12, 17]. Video can be encoded with "basic data" on a "primary" layer and "enhancement" data on a "secondary" layer. Primary data is streamed to the client via a reliable channel and enhancement data via a lossy channel. Alternatively, they could be sent on the same channel with enhancement data being given a lower priority. When the network is congested, enhancement data can be dropped to reduced the data rate. Video frames can still be decoded as long as the data on the primary channel arrives, with better quality if enhancement data loss is small. These schemes require the network to support some kind of priority control, whereby packets tagged with low priority are dropped first.

Instead of relying on the priority for packets, some schemes that use multi-layered coding stripe one session into different subsessions, with each layer being one of the subsession. Depending on the connection speed, different clients can subscribes different number of subsessions. For example, for clients connected with broadband channels, they can subscribe all the subsessions. For clients connected with ISDN or other narrowband channels, only the basic subsession is subscribed. Those in between can subscribe the basic subsession plus one or two subsessions with enhanced layer, yielding a better visual quality.

1.2 Limitation of Existing Research

Many of the current light weight video delivery system have one or more of the following limitations:

1. They simply assume a fixed channel rate. To accommodate different quality for different type of channels, different sets of video files are stored in the server.

2. They use proprietary video format. Regardless the original format, video clips must first be re-encoded to the supported format before they can be delivered.

3. For schemes with priority in which less important data may be dropped within the network, the network has to have support of priority control. This is not the case for the Internet or most of the LAN that uses Ethernet.

4. For those that re-encode the video in realtime, dedicated hardware is needed for such computationally expensively operation. The heavy computational load on the server is also an obstacle to building scalable server/client systems.

1.3 Contributions of This Thesis

Here, we propose a client-pulled frame-rate adaptive video retrieval scheme. In our system, each client makes explicit request for each frame of video data. By measuring the application elastic buffer occupancy and the incoming frame rate, the client can adjust the request rate to accommodate the network bandwidth and server capacity. If the transport layer does not provide reliable delivery, loss rate is also monitored. By reducing the request rate and thus reducing the congestion at the bottleneck, loss rate can be reduced. Data rate are reduced by skipping frames, and the skipped frames are replaced at the client by duplicating the received I or P frames, obviating the need for adjusting the display rate. Our studies are based on standard pre-recorded MPEG-1 video files that can be played back on most of the computer systems, including workstations and desktop PCs today.

The goal of this work is to explore a light-weight, best effort video transport system that can readily be deployed on the network environment today.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 summarizes the current development of Realtime Transport Protocol. Chapter 3 presents the algorithm for adapting to the current network condition. In chapter 4 we discuss the measurement of statistics in detail. Chapter 5 presents the frame skipping and stuffing technique in detail. Experiment results are presented in Chapter 6. Chapter 7 concludes this work.

Chapter 2

Related Work

2.1 Ongoing Efforts For The Support of Real Time Applications on the Internet - RTP

As realtime applications gain popularity on the Internet, efforts are made to provide support for such traffic. Among them, the Realtime Transport Protocol, or RTP, now being an Internet-Draft by the Internet Engineering Task Force (IETF), draws much of the attention.

RTP is a thin protocol intended to provide end-to-end delivery service for data with realtime characteristics. Applications typically run RTP on top of UDP, but the protocol may be used with other underlying network or transport layer protocol. RTP itself does not provide any mechanism to ensure timely delivery or provide any QoS guarantees. It relies on lower-layer service to do so. For example, if the lower layer is UDP, these properties are not provided. On the other hand, if the RTP is run on top of ATM, it can have some QoS guarantee. Neither does it guarantee delivery or prevent out-of-order delivery, nor does it assume so for the underlying network.

RTP consists of two closely related protocols: the Realtime Transport Protocol (RTP) and the Realtime Tranport Control Protocol (RTCP). The Realtime Transport Protocol carries the Data with realtime properties. Timing reconstruction of the data and loss detection of packets are achieved by means of timestamps and sequence numbers in the RTP protocol header. RTCP carries sender and receiver reports with information including the number of packets sent, number of packets lost, interarrival jitter, and so on. RTCP packets are multicasted to the participant of the session periodically, using up to 5% of the session bandwidth as suggested in the RTP document. Applications may use these information to detect or predict persistent congestion and react accordingly to avoid further congestion.

Congestion can be relieved by reducing the transmission rate. Transmission rate can be adjusted by various methods. RTP provides the concept of *mixers.* A mixer is an intermediate system that receives RTP packets from one or more sources, combine and reconstruct the stream in some manner, change the format optionally, and then forwards a new RTP packet. By reducing the encoding rate of the data, the forwarded packets can have a lower bandwidth consumption. An example where mixer is useful in dealing with congestion is that in audio multicast sessions where most of the participants are connected through broadband network while some of the remaining participants experience congestion with their low speed link. Instead of encoding at the lowest supported bandwidth among all participants, mixers can be used to reduce bandwidth requirement of the audio data by changing the compression format and possibly reducing the quality of the data.

Another method of adjusting the encoding rate suggested in the RTP Internet Draft is the use of layered coding. Instead of having one high quality stream, the source is encoded into a primary layer with lower quality (and thus lower bandwidth requirement) and one or more enhancement layers. Data can be decoded by receiving the primary layer, with better quality for each enhancement layer received. To facilitate the variation of the available bandwidth for different participant in the multicast scenario, the source is striped into sessions, with one layer in one session. Participant with low bandwidth connections can subscript just the session with the primary layer. Participant with more bandwidth to spare can subscript sessions containing the enhancement layers.

2.2 Using the Algorithm on top of RTP

RTP is used to carry the realtime data. RTCP is used to monitor the quality of service and to convey information of the participants in an on-going session. The two together provided a framework on which realtime applications can be built.

The proposed algorithm is an end-to-end application protocol aimed at adapting frame based video traffic to the current network condition. The algorithm includes the measurement and estimation of network condition as well as the procedures to adapt the video traffic to the network condition. Applications that use this algorithm as an server/client application protocol may choose to use UDP or TCP as the transport protocol. These applications may also use RTP to carry the video data.

Since this algorithm aims at adapting video traffic for point-to-point connections (e.g. web-based video-on-demand applications), each session consists of only the server and the client. The RTCP packets is sent from server to client periodically. In the current version, the information from the server is not used. All the statistics used are measured locally at the client side. If applications choose to use RTP with this algorithm, the information carried by RTCP packets are optional.

The relationship of RTP with the proposed algorithm is that RTP outlines the framework for carrying realtime data over the Internet, with protocols that aim at providing informations about the on-going session. How these information are used, however, is out of the scope of RTP. The proposed algorithm in this thesis takes into account how the information measured by the client can be used to adapt to the network conditions. The algorithm can be used with RTP and benefits from the established structure of the RTP protocol. For a more light-weight system structure, applications may run on UDP instead to have more compact header format.

Chapter 3

An Adaptive Video Retrieval Algorithm

In this section we investigate an algorithm that adapts the request rate (hence the video quality) according to the data rate supportable by the system and dynamically adjusts the request-display window size to absorb the variations in the arrival rate. By these means we want to provide a continuous flow of video display on the client side with the best possible quality.

3.1 Lossless Environment

We first assume a reliable transport layer on which these applications are built, such as TCP. With the reliable channel, all the data sent by the server will arrive at the client side. Later in this section we will consider the lossy model and the effect of it on the algorithm.

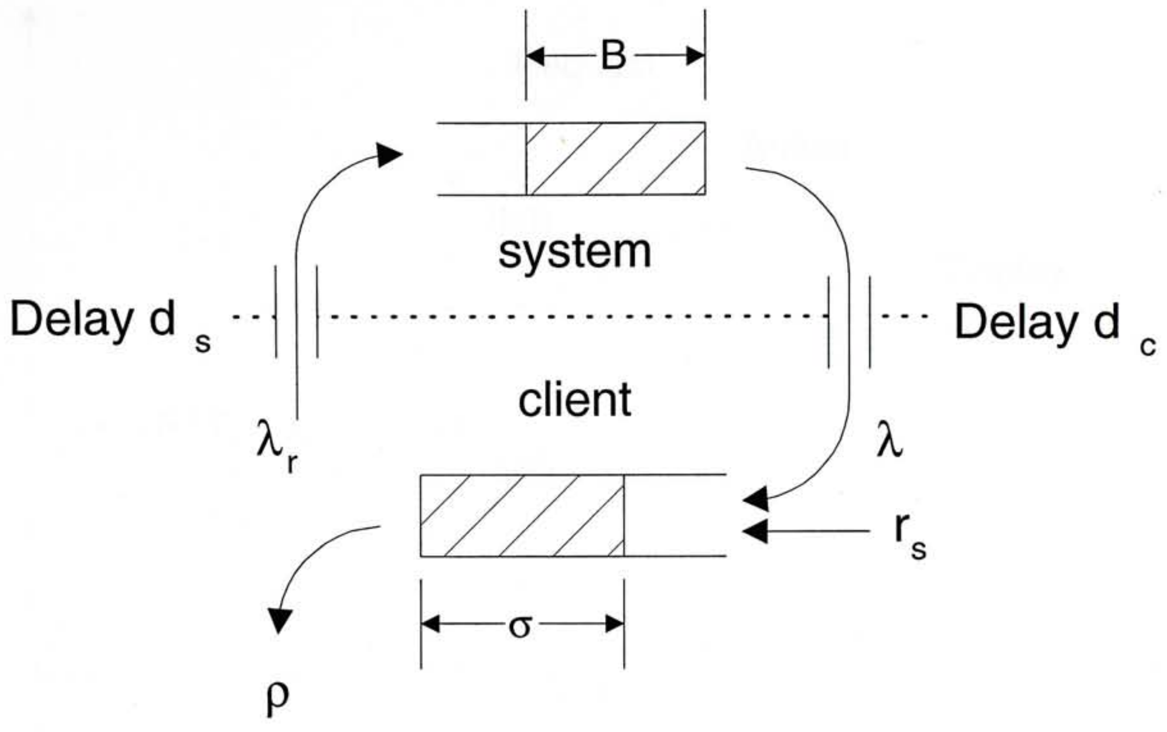


Figure 3.1: System Model for Lossless Environment

Define λ_r as the frame request rate, λ_{upper} as the maximum frame rate supportable by the system and λ as the measured frame arrival rate. Define also Λ_r as the request data rate, Λ_{upper} as the maximum data rate supportable by the system, and Λ be the measured data arrival rate. In this context, λ_r refers to the number of frames per second requested from the server. The frame number (which increments sequentially with each displayed frames) of the requests, however, advances according to the total rate λ_{total} which equals $\lambda_r + r_s$, where r_s is the frame-skipping rate and is normally set $\rho - \lambda_r$, where ρ is the display rate. Dummy frames are stuffed into the received stream at rate r_s on the client side, and this rate is determined when the requests are made.

The mapping between Λ_r and λ_r depends on the video source. This will be discussed in detail in later sections.

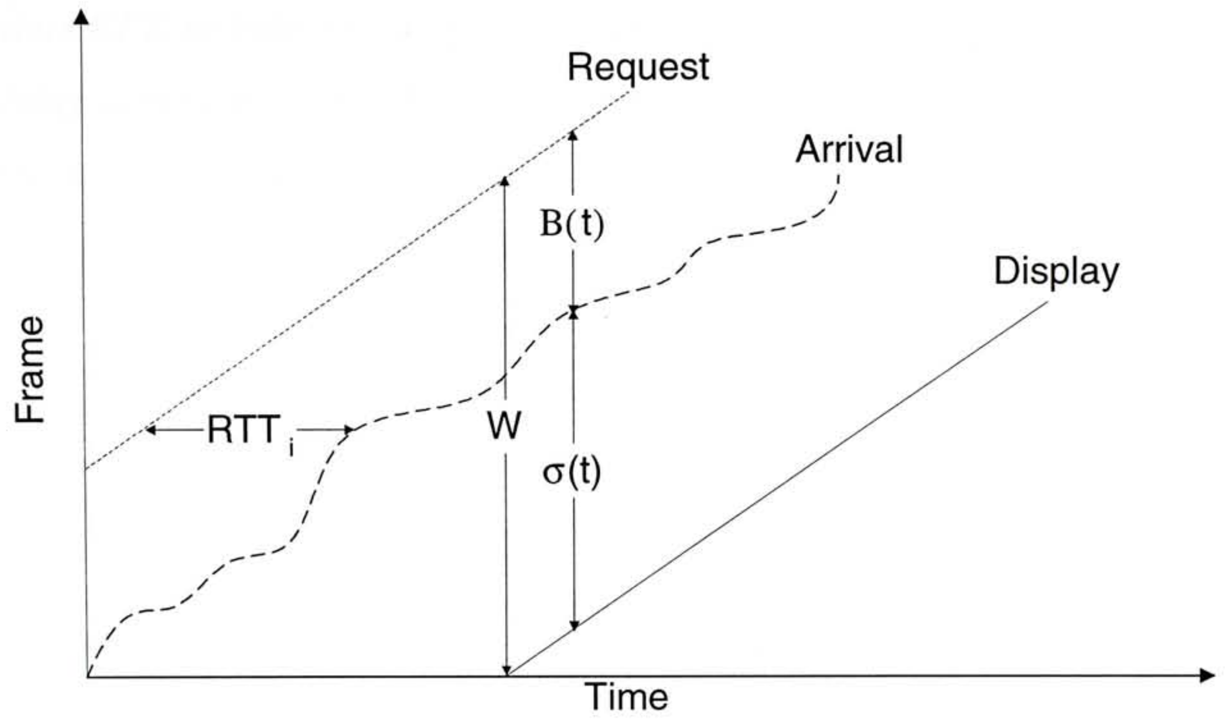


Figure 3.2: Request, Arrival and Display curves

As illustrated in figure 3.1, requests from the client go through some propagation delay d_s and arrive at the system. at rate λ_r. As far as the client is concerned, it can treat the server and the network as a system which responds to its requests after some delay. A number of B requests are queued up, and they are served at rate λ. Frames arrives at the client's buffer after some propagation delay d_c. These frames are put into the client's buffer together with the stuffed frames to yield buffer occupancy of σ frames. The frames are displayed at a fixed rate ρ.

There are variations in the service time for each frame because of the fluctuations in system capacity and difference in frame size. As a result, various system parameters will vary accordingly, as shown in the example in figure 3.2. The Round trip time (RTT) is the delay between the issuing of a request and the complete reception of the corresponding frame data. For frame i, its round trip time RTT_i includes the propagation delay to and from the server, the queuing delay in the system and the service time (including the transmission time of the frame). B_{total} is the total number of outstanding requests, which includes all the requests in transmission to the server, requests backlogged in the server, frames that are being sent to the client, plus the number of stuffed frames associated with these requests. The actual number of requests backlogged in the network and server B_{actual} is equal to B_{total} minus the number of stuffed frames. In particular, we indicate the number of backlogs waiting in the bottle neck of the system by B, as shown in figure 3.1.

3.1.1 Adapting the Request Rate to the Available Bandwidth

This section describes how the client detects and matches its frame request rate λ_r so that the resultant request data rate Λ_r matches the supportable data rate Λ_{upper}. When $\Lambda_r > \Lambda_{upper}$, the system cannot support the service. In this case, data and requests are backlogged in the system, and the client buffer level starts to drain. When the condition is identified, the algorithm would try 1, to clear the backlogs by slowing down the request rate to a value smaller than Λ_{upper} and 2, to adjust Λ_r to an appropriate value after the backlogs are cleared. When $\Lambda_r < \Lambda_{upper}$ (and $\lambda_r < \rho$), the system is under-utilized. The algorithm would try to maximum the utility by increasing the request rate slowly. We will examine each case in detail.

Case 1: $\Lambda_r > \Lambda_{upper}$

Under this condition, the arrival rate Λ reaches Λ_{upper}. Since Λ is smaller than Λ_r (or equivalently $\lambda < \lambda_r$), the buffer level σ will keep decreasing. The rate at which the buffer drains is $\rho - (\lambda + r_s)$, or $\lambda_r - \lambda$. When $\sigma < \sigma_{min}$, the buffer is said to be below a critical level. It implies that backlogs of outstanding requests are building up in the system. In particular, B is increasing. At this point, the client enters a *three-stage probing phase.*

In the first stage of the *probing phase*, the client set Λ_r to a value such that the resultant Λ_r is smaller than Λ. With the input rate smaller than the output rate at the bottleneck link, the number of backlog B would gradually decrease. Since there are still backlog in the system, the arrival rate would still be running at about Λ_{upper}. The client can therefore measure the supportable arrival rate in this period.

We mark the first requested frame after lowering λ_r as f_1. Referring back to figure 3.1, by requesting at a rate lower than the supportable rate, the input rate to the system bottle neck (Λ_r) will be smaller than the output rate (Λ_{upper}), and so the system backlog B will be cleared up and at the same time σ will be built up gradually.

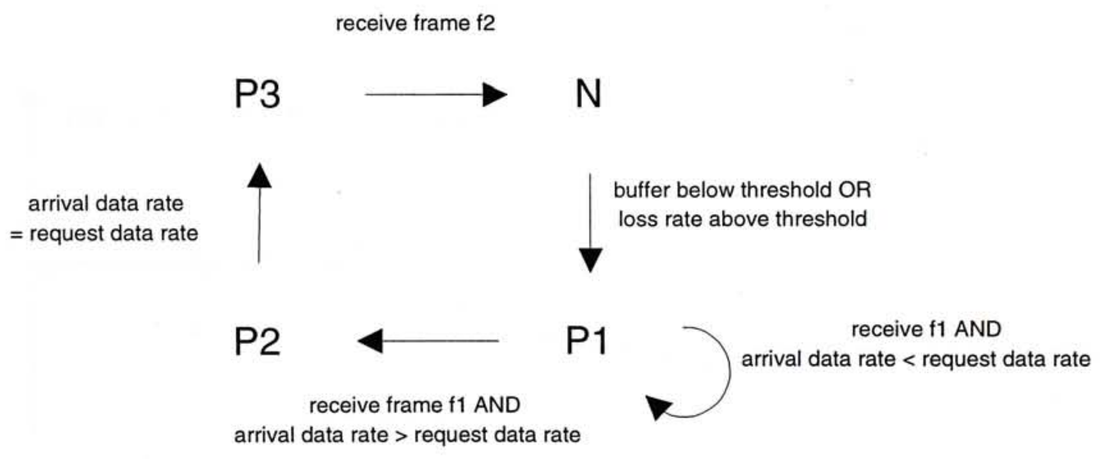
Meanwhile the client should measure the rate difference $\delta = \Lambda - \Lambda_r$. The measurement continues until f_1 is received. This δ is stored for later use.

The second stage starts when f_1 is received. It indicates that the previous decision on decreasing the request rate should now be effective, i.e. the frames arrive from this point onward are requested at the new request rate. Also, the backlog in the system would have been cleared up. The arrival rate would gradually decrease to the current arrival rate.

If at this point $\Lambda < \Lambda_r$, the request rate is still not properly matched yet. Therefore the first stage of the *probing phase* should be run again. i.e. set Λ_r to a value smaller than the current Λ. Otherwise (i.e. $\Lambda \geq \Lambda_r$), the client should continue to measure Λ. As the backlog in the system clears up, Λ will gradually decrease to the current Λ_r. This marks the end of the second stage, and now the client enters the third stage of the *probing phase*

In the third stage, the client increase the request to the value measured previously in the first stage. At this point, the client can set λ_r by mapping the data rate $\Lambda_r + \delta$ to the frame rate, where δ is the rate difference measured before frame f_1 arrived. We mark the first request frame after the setting of λ_r as f_2. The *three-stage probing phase* ends when f_2 is received.

A summary of the transition of stage is shown in figure 3.3


receive frame f2
P3
N
arrival data rate
= request data rate
buffer below threshold OR
loss rate above threshold
P2
P1
receive f1 AND
arrival data rate < request data rate
receive frame f1 AND
arrival data rate > request data rate

N: Neutral
P1: Over-compensated probe down
P2: Equilibrium sensing
P3: Recompensation probe up

Figure 3.3: the three-stage probing phase

Some examples are shown in figure 3.4, 3.5 and 3.6. Example 1 shows the case where λ_{upper} stays constant throughout the *probing phase.* Examples 2 and 3 show the cases where λ_{upper} changes during the *probing phase* and the result of these changes.

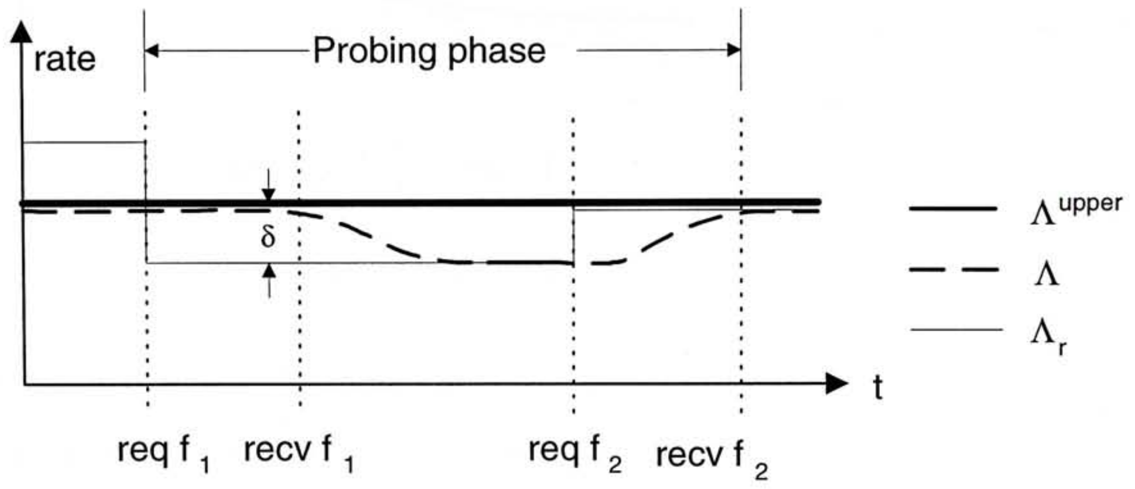


Figure 3.4: Probing Phase Example 1

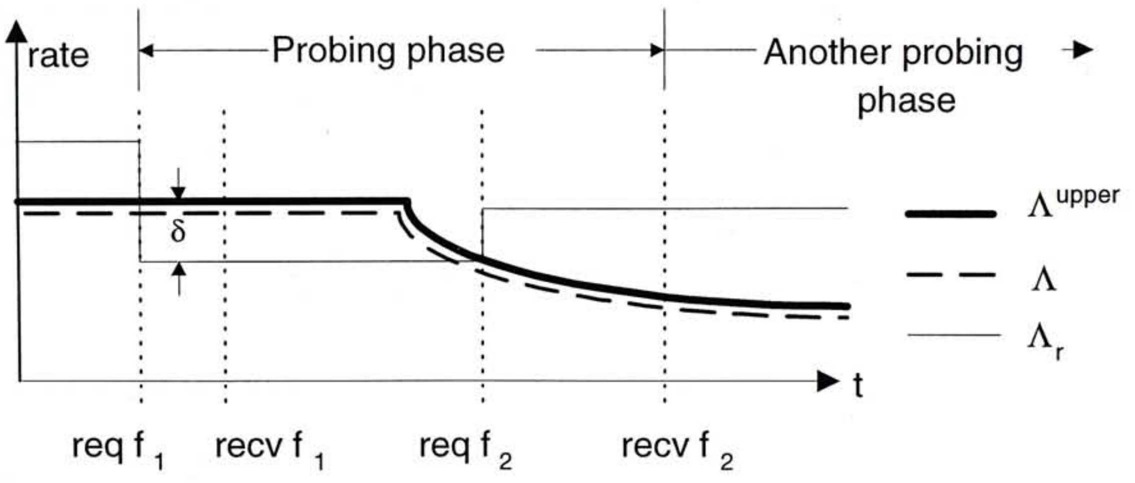


Figure 3.5: Probing Phase Example 2

We mentioned σ_{min} earlier in this section. The client should make decision on changing the request rate when the buffer drifts down to this value, and it takes some time for the decision to take effect on the buffer level. If the decision

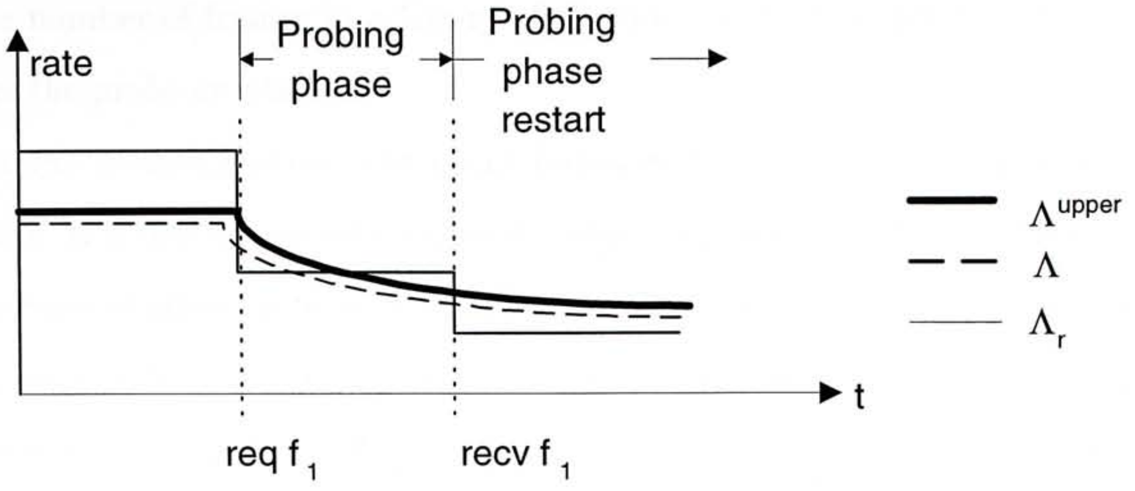


Figure 3.6: Probing Phase Example 3

is made too late, all the remaining data in the buffer could have been used up, the the display would have to be stopped. To avoid buffer starvation, we set a minimum buffer threshold σ_m (say, $\sigma_m = 10$ frames of video data), a safety margin below which we do not want the buffer level to drop. Since the buffer is draining at a rate of $\lambda_r - \lambda$, and the time it takes for the new request rate to take effect is B_{actual}/λ, we have,

$$\begin{aligned} \frac{B \cdot \hat{s}}{\Lambda} &< \frac{\sigma_{min} - \sigma_m}{\lambda_r - \lambda} \\ \text{or,} \quad \sigma_{min} &> B \cdot \hat{s} \left(\frac{\lambda_r - \lambda}{\Lambda} \right) + \sigma_m \end{aligned}$$

where $\hat{s}$ is the estimated frame size.

Case 2: $\lambda_r < \lambda_{upper}$

Assume that $\Lambda_r = \Lambda = \Lambda_{upper}$. Suppose now Λ_{upper} increases (due to the freeing up of system resources, for example). The observable effect is that the subsequent RTT will be smaller. Define a window of L frames (preferably a multiple of the number of frames in a Group of Picture). If $RTT_i < RTT_{i-L}$, the client enters the *probe-up phase*.

In the *probe-up phase*, the client increases the request rate linearly. The value of λ_r is first increased by a small value (say, by one). This λ_r is kept until the measured arrival rate Λ matches Λ_r, the data rate corresponding to λ_r, then λ_r is increased again. This process should be repeated until a point when λ_r reaches ρ or when $\Lambda_r > \Lambda_{upper}$. For the latter case, the client will enter the *probing phase* as described in the previous section.

3.2 Lossy Environment

Although TCP provides reliable transportation, it is not designed for multimedia streaming data. While it may be used for real time multimedia applications in local area networks where RTT is small, it becomes inadequate for real time data with the slow evolution of its transmission window over long distance networks where RTT goes up to 300ms. For this reason many applications use UDP for their transport layer protocol, or build their own protocol on top of UDP. Unlike TCP, UDP does not provide reordering for out of order packets or retransmission for loss packets. Under such environment, the assumption of conservation of data, i.e. all frames sent by the server will be received by the client, is not valid anymore. In case of network congestion where buffer overflows occur in intermediate nodes, packets will be dropped silently.

We consider the case of congestion here. Congestions may occur in the network or in the server. These two cases are not mutually exclusive, meaning that both might happen at the same time. When any of the buffer along the path (including the buffer of the network adaptors, buffers in the switches, or the application buffers of server) overflows, packets of requests or data are dropped. Since the system has no preference on which kind of video packets (whether they are packets for I, P or B frames) to drop first, any one of them might be discarded. From the client's point of view, video packet loss means degradation to the visual quality of the video, especially when packets for I/P frames are lost. If the client can reduce the loss rate by skipping the request for less important data (i.e. B frames) first, the impact towards visual quality will be less severe. Therefore it is better for the client to reduce the load to the server or network by reducing the request rate.

As mentioned before, instead of building up backlog, unsupported requests are dropped by the system. This has two implications for the client. 1, the client has to detect loss and try to compensate for the loss and 2, the client should determine if the data loss is caused by the fact that the request rate is too high.

Loss can be detected using similar methods TCP has been using. A time out value is set using the round trip time value and the RTT deviation for each packet. A packet is considered lost when it does not arrive before the associated time out. Alternatively, in local area network where the occurrence of out-of-order packets is rare, loss can be detected simply by using sequence number in the header. When a packet of sequence number n (or packet n) is received before packet k, where $n > k$, it can be concluded that packet k is lost. When a frame is lost, dummy frames should be padded to the incoming stream of data to preserve the timing with other streams.

Losses are rather random when the streams are using a small fraction of the available bandwidth. When this is the case, the loss probability depends on the configuration of the path along which the data are sent. This is not a case for congestion. Rather, it is the nature of the lossy environment. In this case, skipping requests will not help reducing the loss probability and therefore the request rate should be maintained. Studies show that a random loss rate up to 10% is possible in some network environment [4]. To absorb the effect of these random loss, a safty margin of loss rate up to ϵ of the request rate is used. When loss is detected with loss rate below the safty margin, it is assumed that the loss is not caused by system congestion and therefore request rate will not be changed.

When the streams constitute the major portion of the available bandwidth, losses become correlated with the request rate of each stream. If the request rates are indeed too high, reducing the rate helps to reduce the loss rate.

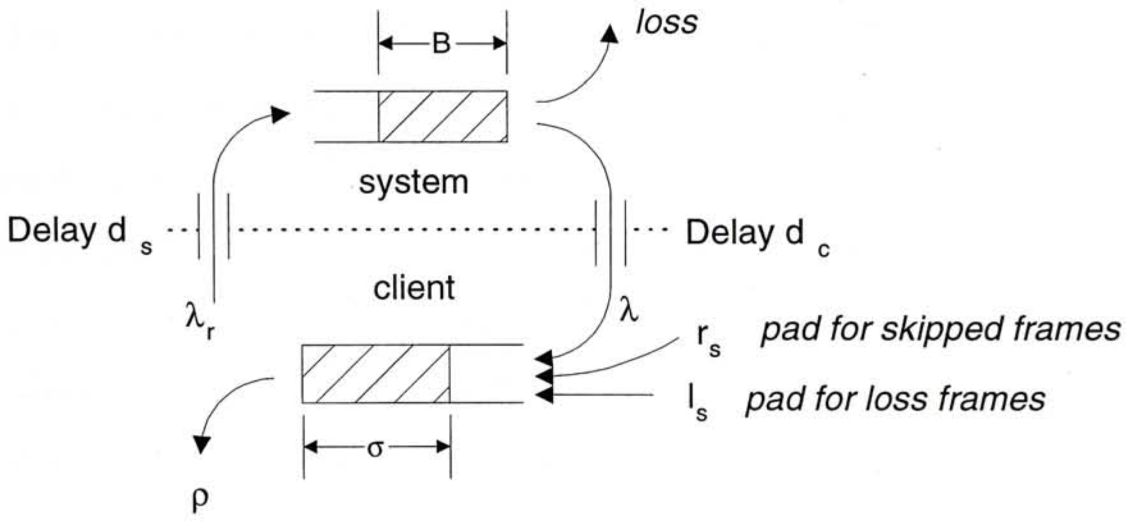


Figure 3.7: System Model for Lossy Environment

Figure 3.7 shows the model for a lossy system. The notation used is the same as that in the lossless system, except that a certain portion of data is lost. When losses are detected, dummy frames are inserted into the stream in place of the original frame.

3.2.1 Adapting Λ_r in Lossy Environment

Case 1: Buffer Underflow on Client Side

In some architecture where the server has a large application buffer, the requests from clients may be backlogged inside that buffer. This will happen when the server has reached its capacity and can no longer handle all the requests on schedule. These backlogs in the server's buffer means the number outstanding request for the client is increasing, or equivalently, the client buffer level is decreasing. In these cases, the adaptation procedure is the same as that in the lossless case. By reducing the requests, the server is relieved from the load and thus the backlogs can be clear up.

Let Λ be the measured arrival data rate. To be precise, only the packets with actual video data are counted. Dummy frames compensating the skipped requests or loss are not included in the measurement. Using the same calculation as in the lossless case, when the client buffer level is dropped below the threshold σ_{min}, the client will enter the *three-stage probing phase.*

Since the environment is now lossy, not all the requests sent are backlogged in the system. Some of them might have been lost. In this case, the number of outstanding requests for the calculation of σ_{min} must be updated accordingly: when loss is detected, their corresponding requests are not considered outstanding anymore.

Case 2: High Loss Rate

When the bottleneck links reach their capacity, the buffer of the nodes (e.g. a switch or a router) before the link may experience buffer overflow and cause packet loss. On the client side, loss of video data are detected and padded by dummy frames. By means of these dummy frames the timing of the video stream is preserved (some of the video decoder would have problem preserving the synchronization between two MPEG streams. For each frame skipped, the video will go one frame faster than, say, the audio stream). These dummy frames, however, cannot reproduce the visual information the original packets are carrying. If other frames are encoded with reference to these frames, the decoder will produce blurred or incorrectly decoded frames. To reduce the loss, request rate should be reduced to relieve the load at the bottleneck.

As shown in figure 3.8, video stream I shares the bottleneck link with connections I_1 to I_k. When the aggregated input rate of I, $I_1, I_2, ..I_k$ exceeds the bottleneck service rate λ_i, the link buffer is likely to overflow. Packets from these connections will be dropped when the link buffer overflows. The resultant rate for the video stream is constrainted by this link, i.e. the arrival data rate can at most be as high as the service rate of this link.

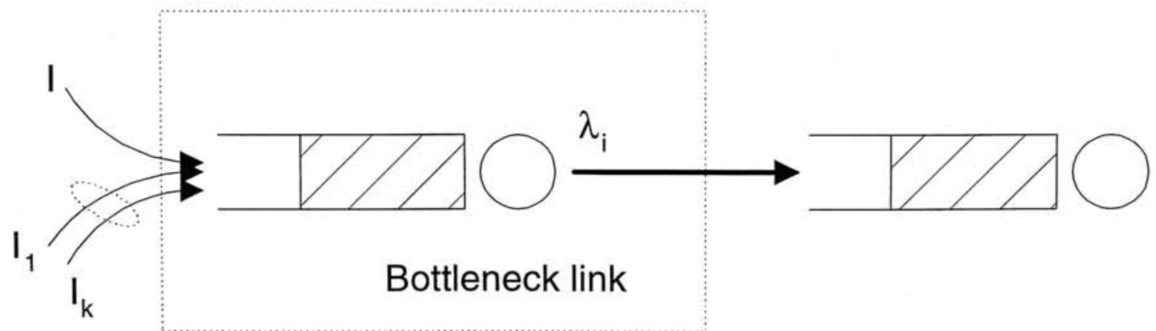


Figure 3.8: Bottleneck

Loss is readily detected on the client side. Whenever loss is detected, the current request rate is compared with the measured arrival rate. From time to time, the client may find that the arrival rate is a bit lower than the request rate because of random loss. For these losses, the client need not change the request rate. A rate of up to 10% is reported for random loss in the Internet in various other works [4, 3]. When the request rate reaches the capacity, the loss rate would increase substantially. By monitoring the loss rate at the client side, the client can reduce the request rate when the loss rate exceed a certain threshold which indicates congestion at bottle neck link.

Denote Λ as the measured arrival data rate. Suppose there is a bottleneck link along the path where the service rate is Λ_i. The traffic along the path will then be constrained by Λ_i. Λ can therefore be at most Λ_i. Suppose the stream carrying the video is the dominant traffic at the bottleneck link. By reducing its rate I to Λ, the aggregated input rate to the buffer at the bottleneck link will be smaller than Λ_i and therefore supportable by its service rate. Buffer overflow will be relieved and thus the loss rate will be reduced.

The situation may be different when the bottleneck link is used largely by traffics without incorporating the adaptation algorithm or without rate control. As mentioned above, loss rate is monitored at the client side and adaptation is triggered when the client loss rate estimation exceed a threshold value. The client then try to reduce the request rate to the currently measured arrival rate to relieve the load at the bottleneck link. Instead of reducing the aggregrated input rate at the bottleneck link, bandwidth spared by the client is used up by other traffic sources. There is a chance that the loss rate of the video stream that reduced the request rate will stay high, forcing it to further reduce its request rate, until the other traffic sources stop using up the bandwidth it spared.

The latter situation is an inherit problem with networks where there is no reservation of resources. The Internet is comprised largely of this kind of networks. The service discipline at the nodes are simply FIFO. When there are sessions using UDP as transport protocol with aggregrated rate higher than the link capacity, they will share the loss among the sessions. Consider an example where there are n sources each sending at rate λ on a link with capacity C. The total input rate to the link is $n\lambda$. When $n\lambda > C$, the loss rate r_{loss} for each source will be roughly

$$r_{loss} = \frac{n\lambda - C}{n}$$

Under such situation, when a client drops its request rate to the arrival rate of Λ, its traffic may still be served at a rate smaller than Λ. Dropping the request does relieved the load at the bottleneck link, but the spared resources is distributed evenly to all the sessions on that link. As a result, the loss rate for everyone (if they do not increase their request rate further) will decrease, but the link may still be overloaded.

These kind of connections that jam the network by using up all the available bandwidth cannot be avoided on networks without any policing function on the source. In case of Internet, there are also no way to avoid such kind of connections. But for most applications developers, a sense of fair use of bandwidth is kept in mind during design. Added to the fact that most of the application use TCP as transport layer protocol, rate control is already incorporated. When the aggregrated bandwidth at a link exceed the capacity and loss occur on connections, TCP would reduce the transmission window size and effectively reduce the transmission rate. Application that uses the algorithm on top of UDP would also reduce the transmission rate. In this case the load at the congested link would be relieved.

Case 3: Going Up

The RTT of the packets may decrease when the bottleneck link is no longer congested. The reason is that the service rate of that link increased with respect to the session. When a decrease of RTT is detected, there is a chance that the link has more bandwidth to spare. To exploit the current capacity of the link, the client tries a higher request rate and see if the system can support that new rate. It is done by setting the the client to *probe-up* stage when the loss rate is below a preset level and a drop in RTT is detected. In the *probe-up* stage, the request rate is increased linearly to see if it is supportable. Request rate is increased. And when the newly requested frame returns, the client will further increase the rate until the detected loss rate exceed the preset threshold. When that happens, request rate falls back to the measured arrival data rate.

3.3 Adjusting the Window Size

This section describes how the clients determines an appropriate request-display window W and adapt to that target window size. Recall that in TCP, the round-trip time (RTT) and its variation are used to determine the retransmission time-out (RTO) value. When the acknowledgement of a packet does not arrive after the RTO value, it is assumed to be lost. In our system, the request-display window W is analogous to RTO in TCP in the sense that W divided by the constant display rate ρ gives us the time allowance we have before the request frame must arrive at the receiver. As in TCP, W/ρ should be set to the expected RTT plus some allowance corresponding to the deviation of RTT.

Assume each request corresponds to one real frame (as opposed to stuffed frames). For the arrival of frame i, we can measure the corresponding RTT_i. Using similar technique as in TCP [19, 20], we calculate the smoothed RTT A and the RTT mean variation D with the following equations for each measured RTT_i

$$Err = RTT_i - A$$

$$A \leftarrow A + gErr$$

$$D \leftarrow D + h(|Err| - D)$$

where g and h are smoothing factors. Smaller factors yield "smoother" estimated values at the expense of a slower response to changes. In TCP, $g = 1/8$ and $h = 1/4$. Lacking compelling reasons to do otherwise, we adopt the same values. Using A and D, we estimate the request-display window $\hat{W}$ by:

$$\hat{W} \leftarrow \rho(A + 4D) \tag{3.1}$$

By setting a larger W, the probability of buffer starvation due to network delay jitter decreases at the expense of using up more memory for buffering frames. In addition, in the case of interactive video applications, more frames need to be flushed upon interactive commands such as jumping to a different display point, fast forwarding, etc. This will give a slower response time.

If the current window size W is found to be "much smaller" than the estimated window size $\hat{W}$, W is set to $\hat{W}$. W can be increased by increasing λ_{total}.

Since we do not want to add any extra work load to the network, we increase λ_{total} by increasing r_s while maintaining the current λ_r. That is, we skip more frames while keeping the same request rate. The effect is shown in figure 3.9.

Keeping a large window is not always necessary. If the server-client connection is known to be stable, the large window will only hinder the response time for interactive applications. For stable network conditions, the variation in RTT is small, and hence a smaller $\hat{W}$. When the current window size is a lot larger than $\hat{W}$, the client should set it to $\hat{W}$. This can be done by pausing the requests until the window size matches. The effect is shown in figure 3.10.

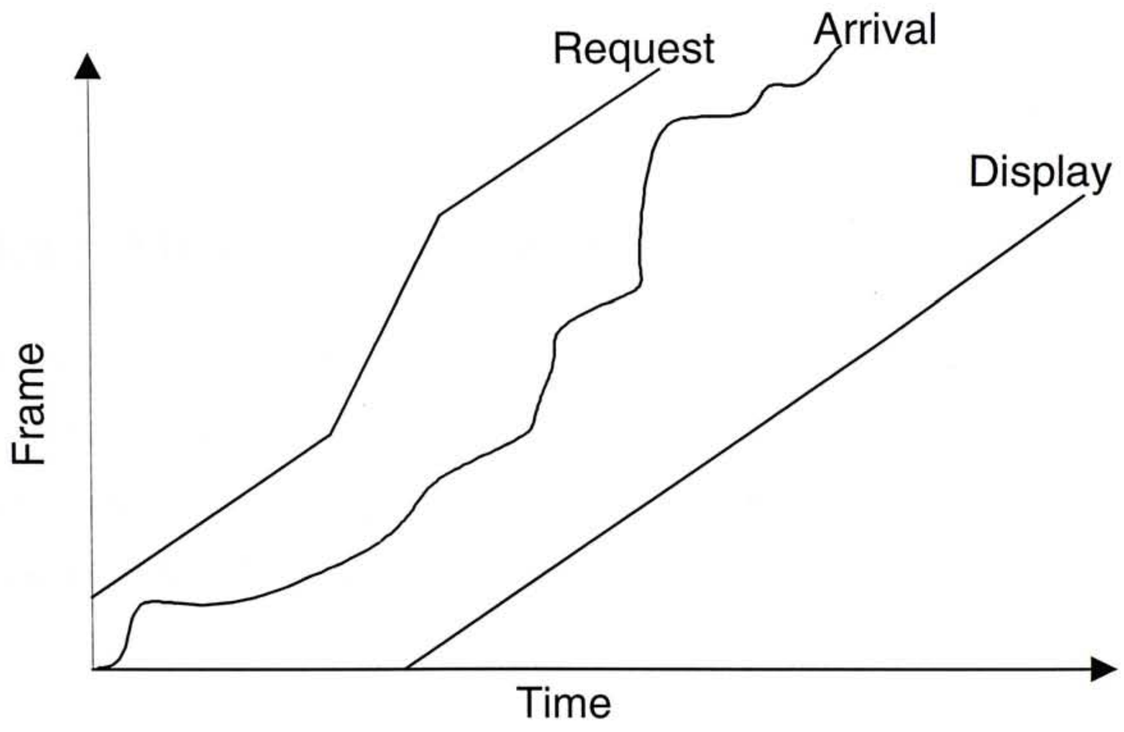


Figure 3.9: Increase Window

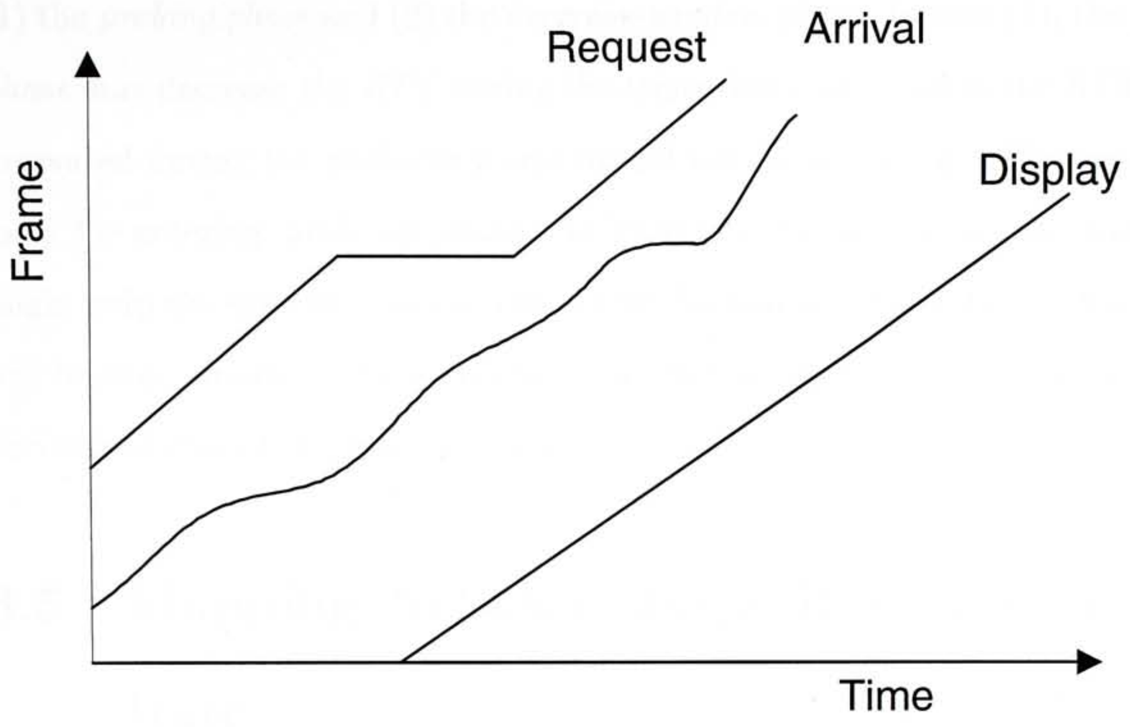


Figure 3.10: Decrease Window

3.4 Measurement Issues

In the previous sections we mentioned that the client needs to estimate parameters such as the arrival rate λ and the round trip time RTT. The operation of some phases changes these parameters, and thus the statistics gathered during that time should not be used for the estimation performed during other phases.

Phases	Param Used	Param Altered	Phases Affected
probe-up	RTT_i		
probing	λ	RTT	*probe-up* (1)
increase W	RTT		
decrease W	RTT	RTT	*probe-up* (2)

Table 3.1: Relation Between Phases

Referring to table 3.1, the decision for the *probe-up phase* will be affected by

(1) the *probing phase* and (2) the *decrease-window phase.* In case (1), the *probing phase* may decrease the *RTT* during the transition stage, and so the RTT values measured during the *probe-up phase* should not be stored for *RTT* comparison used for entering *probe-up phase.* In case (2), by decreasing the window it might help the system clear up some of the backlog and thus the resultant *RTT* will become smaller. Client should avoid comparing these *RTT*'s to make the decision to enter the *probe-up phase.*

3.5 Mapping between Data Rate and Frame Rate

Recall that in the algorithm the decision on the request frame rate based on the measurement of the current data rate. For MPEG-1 video streams, the size of a picture (or a frame) is not fixed. *I* frames are often larger than *P* frames, and *P* frames are often larger than *B* frames. In the stream, there are more B frames than P frames or I frames. Frames of the same type do not necessarily have the same size either, though the difference is often smaller than that between different type of frames. Since the frame sizes are variable, the relationship between data rate and frame rate is not linear.

Suppose the GOP pattern of a video stream is $IBBPBBPBBPBBPBB$, with the average size of 10000 bytes, 3000 bytes and 1000 bytes for I, P and B frames respectively. For the frame skipping scheme mentioned earlier, B frames are skipped first, and then P frames and finally I frames. The relationship of the available data rate and the supported frame rate is shown in figure 3.11

For example, when the available bandwidth is at 64KBytes/sec, the system

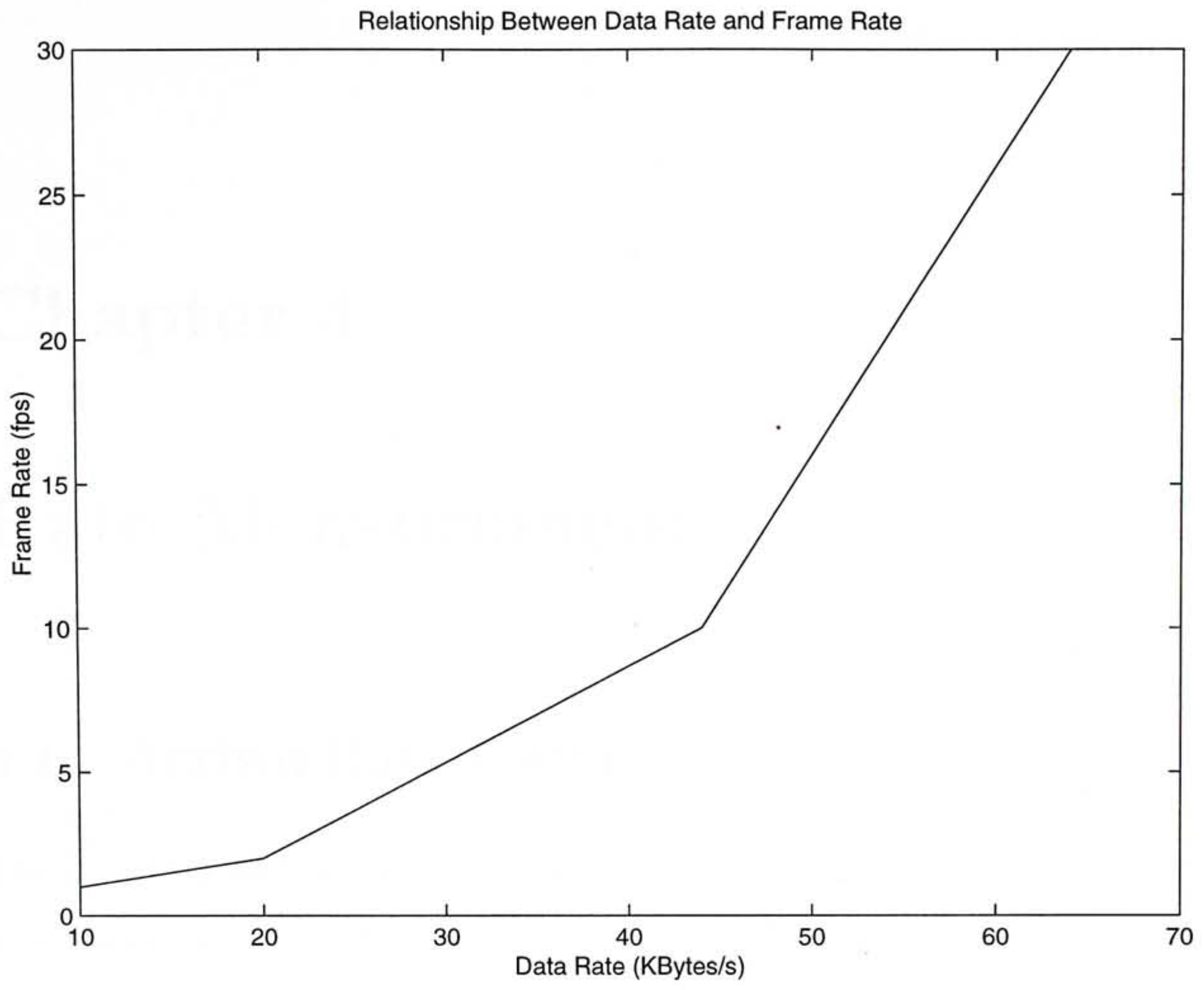


Figure 3.11: An Example Stream

can support video at 30fps. When the available bandwidth drops by half, i.e. to 32KBytes/sec, only 6fps can be supported.

Frame sizes are not fixed. An estimation of the frame size for the corresponding type of frame (I, P or B) is updated when frames of that type arrives.

Chapter 4

Rate Measurement

4.1 Arrival Rate Estimation

The algorithm requires an estimation of the current arrival data rate. Assume a incoming flow of regularly spaced packets and let the arrival time of packet i be t_i with size s_i, the instantaneous data rate r_i is then measured by

$$r_i = \frac{s_i}{t_i - t_{i-1}}$$

In practice, however, the interarrival time of packets may not be regular. Because of traffic multiplexing and queueing in the nodes within the route, packets could come in bursts even when they are sent with regularly spaced intervals. This leads to a fluctuation in the instantaneous data rate. By averaging the instantaneous measurements, a better estimation of the current data rate can be obtained.

Since both the size and the interarrival time of the packets are not fixed, insteading of using the instantaneous rates in the averaging, λ_i, the smoothed arrival data rate upon receiving packet i, is obtained by

$$\lambda_i = \frac{\text{smoothed arrival packet size}}{\text{smoothed interarrival time}}$$

And both packet size and interarrival time is smoothed using

$$S_i = (1-\alpha)S_{i-1} + \alpha s_i$$

where S_i is the smoothed value of the instantaneous measurement s_i, and α is a smoothing factor between 0 and 1.

The choice of α determines the smoothness of the run-length average and the sensitivity to changes. A smaller α gives a smoother average but the response of the average to changes is slow.

The burstiness of incoming packets are measured by experiments. The client regularly sends requests to the server and the server echos back fix sized packets. The interarrival times of these packets are measured by the client forms a discrete signal, with each sample being one of the interarrival time. This signal of interarrival time is frequency analyzed by performing Fourier Transformer, and figure 4.2 shows the results of the server/client in a LAN environment and figure 4.3 shows the result of transpacific environment. The route of the transpacific connection is shown on table 4.1 and 4.2. Both indicates the effect of bursty arrival. For LAN, a peak occur at around 0.5 and for transpacific, two peaks at 0.25 and 0.5 of the Nyquist frequency respectively. Comparing them to figure 4.1 which shows the frequency response of averaging filter with different α, a choice of $\alpha = 0.05$ is reasonable for minimizing the effect of burstiness.

```
Tracing route to sparc5.cs.uiuc.edu [128.174.242.193]
over a maximum of 30 hops:
1    2 ms     1 ms     1 ms     router99.ie.cuhk.edu.hk [137.189.99.254]
2    2 ms     3 ms     3 ms     137.189.200.250
3    1 ms     1 ms     2 ms     csc0g04brb.net.cuhk.edu.hk [137.189.192.254]
4    2 ms     2 ms     2 ms     192.245.196.9
5    218 ms   223 ms   218 ms   unknown-hssi0-1-0.Sacramento.mci.net [166.49.28.9]
6    220 ms   223 ms   252 ms   core2.Sacramento.mci.net [204.70.4.49]
7    275 ms   584 ms   661 ms   core1-hssi-3.Bloomington.mci.net [204.70.1.137]
8    280 ms   295 ms   283 ms   bordercore3.WillowSprings.mci.net [166.48.32.1]
9    276 ms   285 ms   302 ms   cicnet.WillowSprings.mci.net [166.48.33.250]
10   277 ms   278 ms   298 ms   border1-hssi2-0.cmi.qual.net [131.103.24.26]
11   278 ms   274 ms   294 ms   dmz.gw.uiuc.edu [192.17.8.32]
12   286 ms   274 ms   276 ms   world.gw.uiuc.edu [128.174.0.253]
13   273 ms   273 ms   283 ms   terra.gw.uiuc.edu [128.174.1.226]
14   284 ms   273 ms   313 ms   dcs2.gw.uiuc.edu [128.174.252.242]
15   288 ms   281 ms   276 ms   sparc5.cs.uiuc.edu [128.174.242.193]
```

Table 4.1: route of the transpacific link, from banpc13.ie.cuhk.edu.hk to sparc5.cs.uiuc.edu

4.2 Loss Rate Estimation

For lossy environment, a loss rate estimation is required to decide if the request rate is too high for the channel and should be reduced.

Loss data rate can be estimated using the request rate and arrival rate. Using the smoothed arrival rate and request rate calculated previously, we have

$$\epsilon = \Lambda_r - \Lambda$$

That is, when loss is detected, the difference between the smoothed arrival rate and the request rate indicates an loss rate estimation. Using the same smoothing factor α for both loss rate and arrival rate estimation, the latter method of estimating the loss rate is effectively the same as the first method.

Tracing route to banpc13.ie.cuhk.edu.hk [137.189.97.207] over a maximum of 30 hops:				
1	dcs2.gw.uiuc.edu (128.174.242.1)	1.515 ms	1.123 ms	1.065 ms
2	node0-1.gw.uiuc.edu (128.174.252.240)	1.663 ms	1.379 ms	1.277 ms
3	dmz.gw.uiuc.edu (128.174.0.250)	1.562 ms	1.509 ms	1.851 ms
4	cmi.champaign.iagnet.net (192.17.8.35)	2.414 ms	2.547 ms	2.695 ms
5	bordercore1-hssi3-0-1.ord.qual.net (131.103.24.25)	5.184 ms	5.134 ms	5.122 ms
6	bordercore3-hssi0-0.WillowSprings.mci.net (166.48.33.249)	52.443 ms	34.160 ms	5.802 ms
7	core4.SanFrancisco.mci.net (204.70.4.81)	53.311 ms	53.434 ms	53.735 ms
8	core2-hssi-3.Sacramento.mci.net (204.70.1.234)	55.512 ms	61.869 ms	56.457 ms
9	bc30-loopback.Sacramento.mci.net (166.49.28.1)	58.527 ms	58.593 ms	58.142 ms
10	jucc-netplus.Sacramento.mci.net (166.49.28.10)	285.484 ms	275.040 ms	274.019 ms
11	192.245.196.10 (192.245.196.10)	274.379 ms	281.623 ms	279.317 ms
12	csc0g07brb.net.cuhk.edu.hk (137.189.192.238)	276.711 ms	294.684 ms	274.746 ms
13	137.189.200.249 (137.189.200.249)	331.166 ms	319.090 ms	355.015 ms
14	banpc13.ie.cuhk.edu.hk (137.189.97.207)	321.897 ms	353.469 ms	317.106 ms

Table 4.2: route of the transpacific link, from sparc5.cs.uiuc.edu to banpc13.ie.cuhk.edu.hk

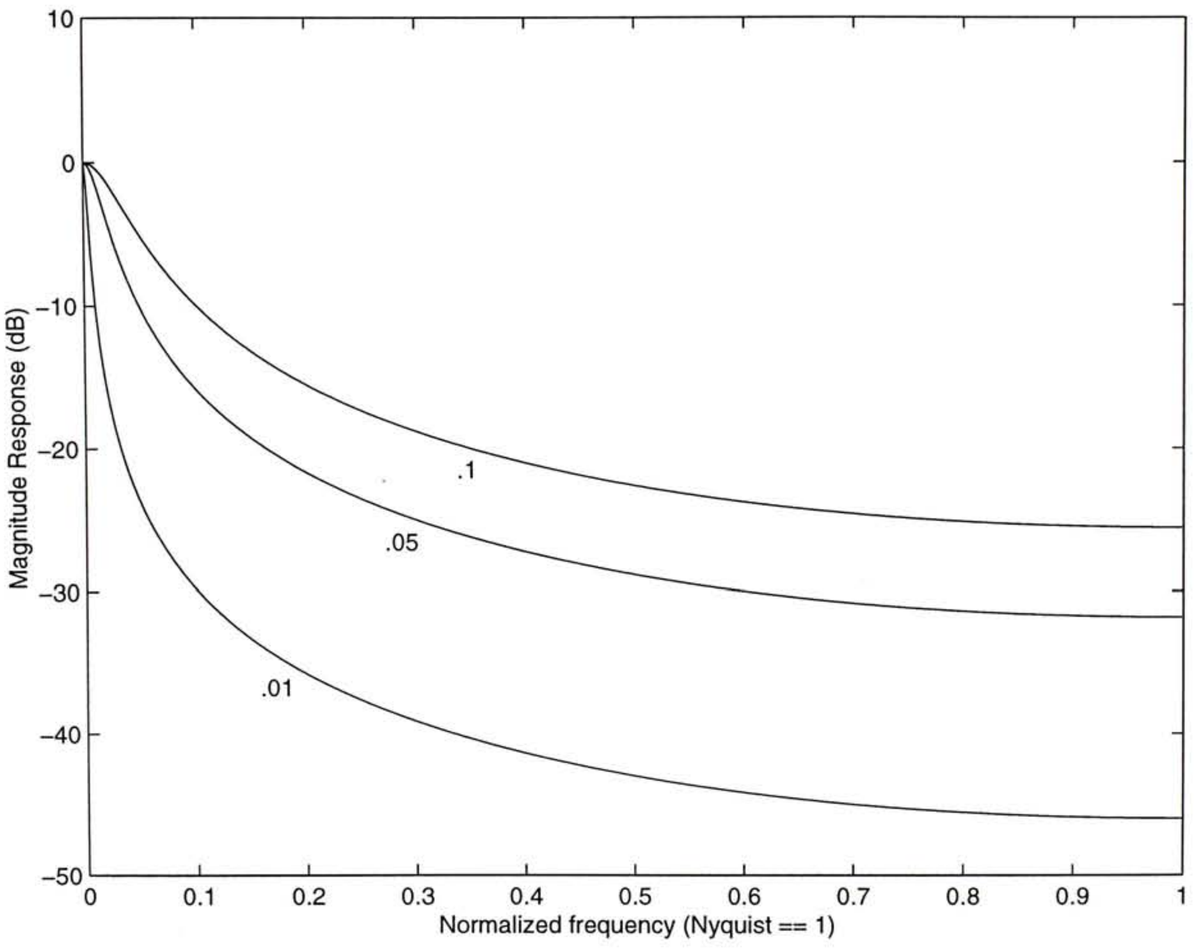


Figure 4.1: Low Pass Filter Response, α =.1, .05 and .1

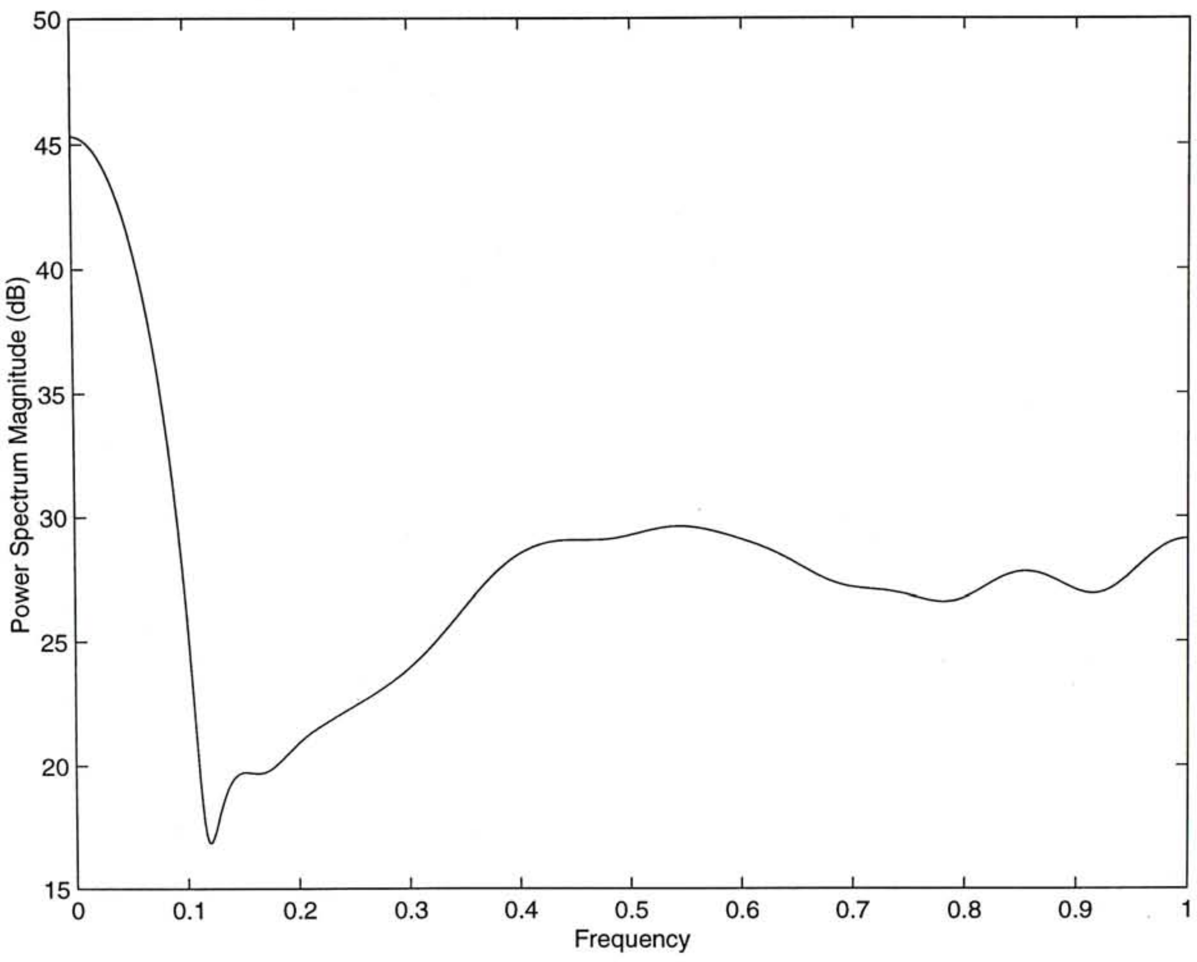


Figure 4.2: Frequency Response of Packets on Local Area Network

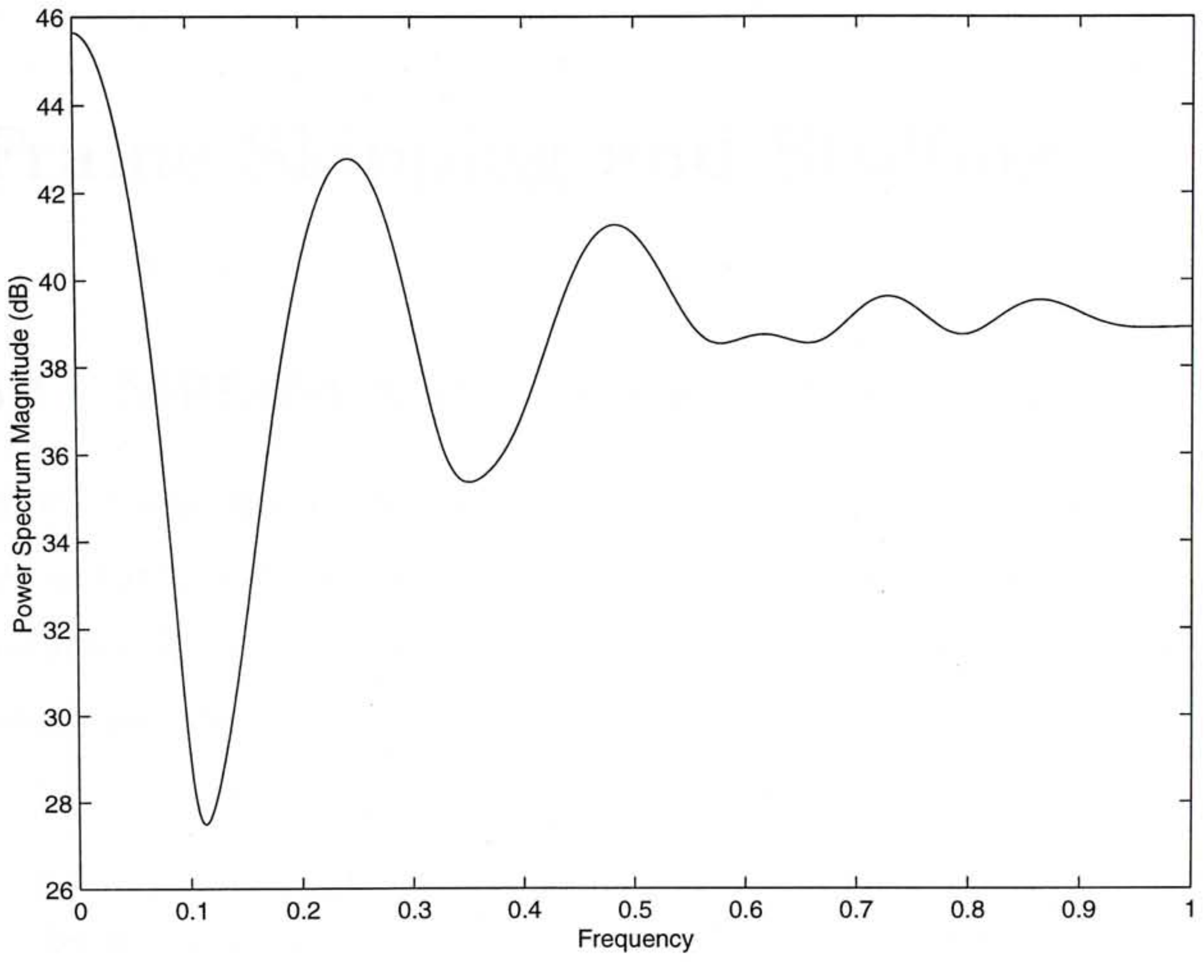


Figure 4.3: Frequency Response of Packets on Transpacific Link

Chapter 5

Frame Skipping and Stuffing

5.1 MPEG-1 Video Stream Basics

MPEG-1 video files consist of intra-coded (I), inter-coded (P) and bi-directional (B) pictures, or frames. These frames are grouped together to form groups of pictures (GOP). For example, a GOP with nine pictures may look like this in bit-stream order (subscript indicates the display order):

$$I_2\ B_0\ B_1\ P_5\ B_3\ B_4\ P_8\ B_6\ B_7$$

Let us denote the number of frames in a GOP by n_g, and the number of I, P, B frames in an GOP by n_i, n_p and n_b respectively. In the above example, we have $n_g = 9$, $n_i = 1$, $n_p = 2$ and $n_b = 6$.

An I frame can be decoded independently of other frames. A P frame may contain forward motion vectors which refer to the decoded pictures of the closest preceding I or P frame for decoding. A B frame may contain both forward and backward motion vectors which respectively refer to the closest preceding and following I or P frames for decoding. Since B frames themselves will not be referred to by other frames for decoding, the detrimental effect of dropping B frames is less severe than that of dropping I or P frames.

5.2 Frame Skipping

In our scheme, the client will first skip the requests for some of the B frames when it is found that the network or the server is congested, thus relieving them of the heavy load. The dropped B frames are replaced by dummy B frames with forward motion vectors indicating that there is "no change from the previous picture". More specifically, the motion vectors referring to the predicting picture are set to zero and all macroblocks (except for the first and last) are skipped. The visual effect is that the referred I or P frame will be displayed again. If the stuffed frame is positioned immediately after the referred I or P frame, the display is effectively frozen by one frame. If all such B frames are dropped and the system still cannot support the frame rate, the B frames following the stuffed B frames are also dropped.

When all the B frames are dropped, P frames are the next candidate. Since a decoded P frames will be used to decode subsequence P frames, the one followed by no other dependent P frames is dropped first. The dropped P frame is again replaced by a dummy B frame which has forward motion vectors indicating that there is "no change".

I frames are dropped only when all the P frames and B frames within the GOP are dropped. Dropped I frames are replaced with the dummy B frame. If such I frame is the first frame of the GOP, the "closed gop" flag in the GOP header needs to be changed to false so that the stuffed B frame can refer to decoded I/P frame in the previous GOP.

For the GOP example above, the dropping priority is

$$I(5)\ B(1)\ B(2)\ P(4)\ B(1)\ B(2)\ P(3)\ B(1)\ B(2)$$

Frames at the same priority can be dropped at random. To maintain smoothness of the video, it would be better to drop the frames with the same priority in an even manner.

The GOP pattern is sent to the client before the playback starts. It can be found by simple pre-processing of the MPEG file. Using 2 bits to represent an entry, the GOP information a one hour 30-frame/sec MPEG file is about 27000 bytes, and it can be further compressed. For fixed GOP pattern, the client only need to know the repeating GOP pattern.

Here we assume that the GOP pattern is fixed throughout the video sequence, which is the case for most video files. If the frame rate supported by the network is λ and the display frame rate is ρ, then the number of frames to be dropped in a GOP, x_g, is

$$\frac{x}{n_g} \doteq \frac{\rho - \lambda_n}{\rho}$$

When the adaptation resolution, $1/n_g$, is large, x can only take on a few values and this will limit the smoothness of the adaptation. In this case, several GOP can be grouped together as a frame-skipped picture group (FPG). The desired number of frames in a FPG depends on the range of adaptable data rate.

For example, if the bit-rate of a video stream is 1.5Mbps and we want to be able to support arrival rate of 150kbps to 1.5Mbps, the adaptation resolution must be smaller than 1/10. If we set it to 1/20, then n_f should be larger than 20.

Denote the number of frames dropped and the total number of frames in an FGP by x_f and n_f, we have

$$x_f = \left\lceil n_f \times \left(1 - \frac{\lambda}{\rho}\right)\right\rceil \tag{5.1}$$

Using $\hat{n_i}$, $\hat{n_p}$ and $\hat{n_b}$ to denote the number of I, P and B frames in the FPG respectively. If $x_f \leq \hat{n_b}$, only B frames will be dropped. The dropping priority should be followed according to the aforesaid manner. Frames should be dropped evenly to maintain the smoothness of the video.

if $\hat{n_b} < x_f \leq \hat{n_b} + \hat{n_p}$, all the B frames within an FGP are dropped. After that, P frames are dropped from the end of the GOP or from those preceding an I frame, as according to the other mentioned previously.

if $\hat{n_b} + \hat{n_p} < x_f < n_f$, all the B and P frames within a GOP are dropped. I frames can be dropped evenly to maintain the smoothness of video.

if $x_f = n_f$, multiple FPG's are grouped together to form a larger FPG. Equation (1) is reapplied to obtain the number of frames to be dropped.

5.3 Frame Stuffing In Lossy Environment

When video data are transported with no delivery guarantee, there is a chance that these data are dropped in the network. For MPEG-1 video, frame loss degrades the visual quality, and the lost data cannot be recovered. Dummy B frame indicating "no change with respect to the previously referred frame " is stuffed into the stream when loss is detected.

The degree of visual degradation depends on the type of frames lost. First assume no previous frame loss. When B frame is lost, there are two possibilities. One is that the B frame is preceded by an I or P frame in transmission order. In this case the lost frame is stuffed with the dummy B frame, the effect is the same as when we skipped the request of that B frame, that is, the previous frame would appeared frozen for another one frame duration. Subsequence frame decoding would not be affected. The other possibility is that the B frame is preceded by another B frame. In this case the lost frame is stuffed with the previous B frame, i.e. that B frame would appear in the stream twice. By doing so these two frame will be identical after decode, and the visual effect is that this frame would be frozen for one extra frame duration. These two cases pose the least adverse effect to the playback.

When an I frame or a P frame is lost and is stuffed with the dummy B frame, all the subsequence (in transmission order) B or P frames in that GOP that refer this P frame for decoding will be affected. If these subsequence frames are fed to the decoder as if there were never a frame loss, they will be decoded with reference to the wrong frame (depends on the implementation of the decoder, most likely this "wrong" frame is the previous I or P frame successfully received and decoded). The resultant decoded frames will look strange. They would have erratic blocky effects, they would appeared blurred, and in most cases they won't fit in the original visual flow.

One way to avoid such disruption is to replace all these subsequence affected frames with the dummy B frame even when they are received successfully. With this replacement, the screen would be frozen for a longer period of time, or more precisely, for n frame duration where n is the number of frames replaced, including the lost P frame itself.

A P or B frames actually do not necessary refer to the previous I or P frame for decoding. In some cases, these frames are entirely encoded independently. If these independently encoded P or B frames are received successfully, replacing them with the dummy B frame because of the loss of the preceding P frame would be wasteful. For more sophisticated systems, the actual encoding method of that frame can be checked before making the decision of stuffing,

Chapter 6

Experiment Result and Analysis

In this section we will describe the detail of the methods used to evaluate the algorithm, including the experiment setup and the system configuration. We would also give analysis to the results obtained in the experiments. Through the analysis and the experiments we would like to investigate the performance of the algorithm, the interaction between the algorithm and other transport layer protocols such as TCP and UDP, and the output video quality.

6.1 Experiment

In this part of the experiment, we test the system on the local area network in the Department of Information Engineering, CUHK. Figure 6.1 shows the configuration of the experiment. The clients run on Pentium Pro PC with Microsoft Windows 95. The servers run on a Sun Sparc 20 workstation with SunOS 5.4. These machines are interconnected with a 10Mb Ethernet. To simulate the effect of system congestion, dummy clients which send extra requests are also connected to the server. These dummy clients control the system resources available to the client under test. Each dummy client makes use of UDP to inject data into the network with rate ranging from 2Mbps to 3Mbps. Depends on the need of each experiment, the sum of data rate used by all dummy clients can be up to 8Mbps. None of these dummy clients adapts to the network traffic condition, i.e. they transmit at a fixed rate. In case their requested rate is not supportable by the network, packets are dropped silently in the network.

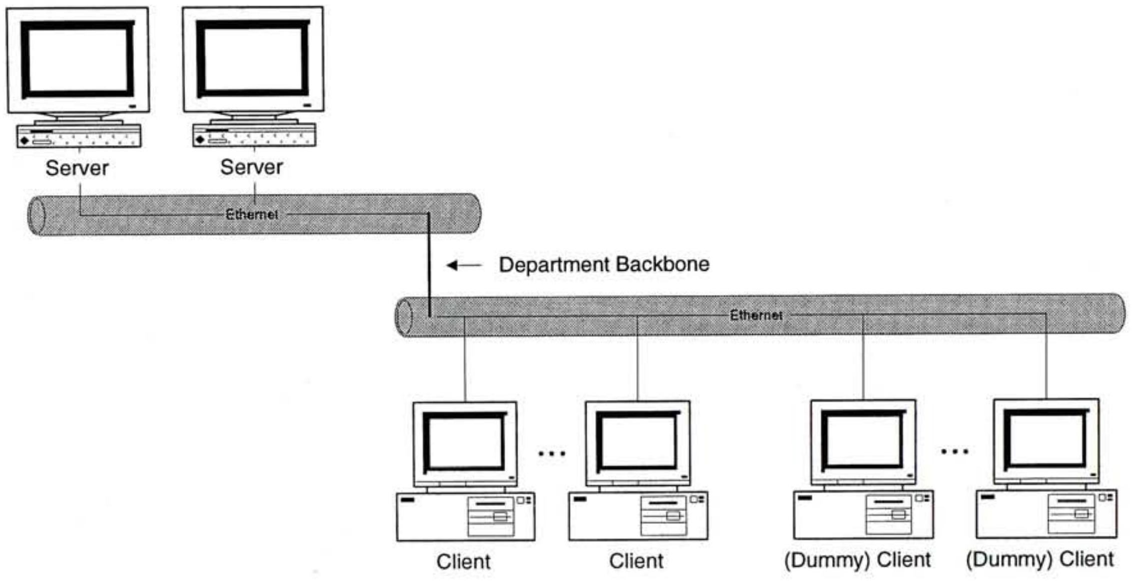


Figure 6.1: Experiment Setup

Using TCP as transport layer protocol

We first consider running our adaptive algorithm on top of TCP. Figure 6.2 shows the buffer occupancy, request and arrival data rate for a particular session with smoothing factor α set to 0.01. The link capacity is about 8.2Mbps. Dummy clients send interferring traffic at rate 7.6Mbps starting at time 40s. The buffer level goes down at the same time, system backlog builds up) and reaches the threshold at time 45s. Request rate is dropped from 140kBytes/s down to about

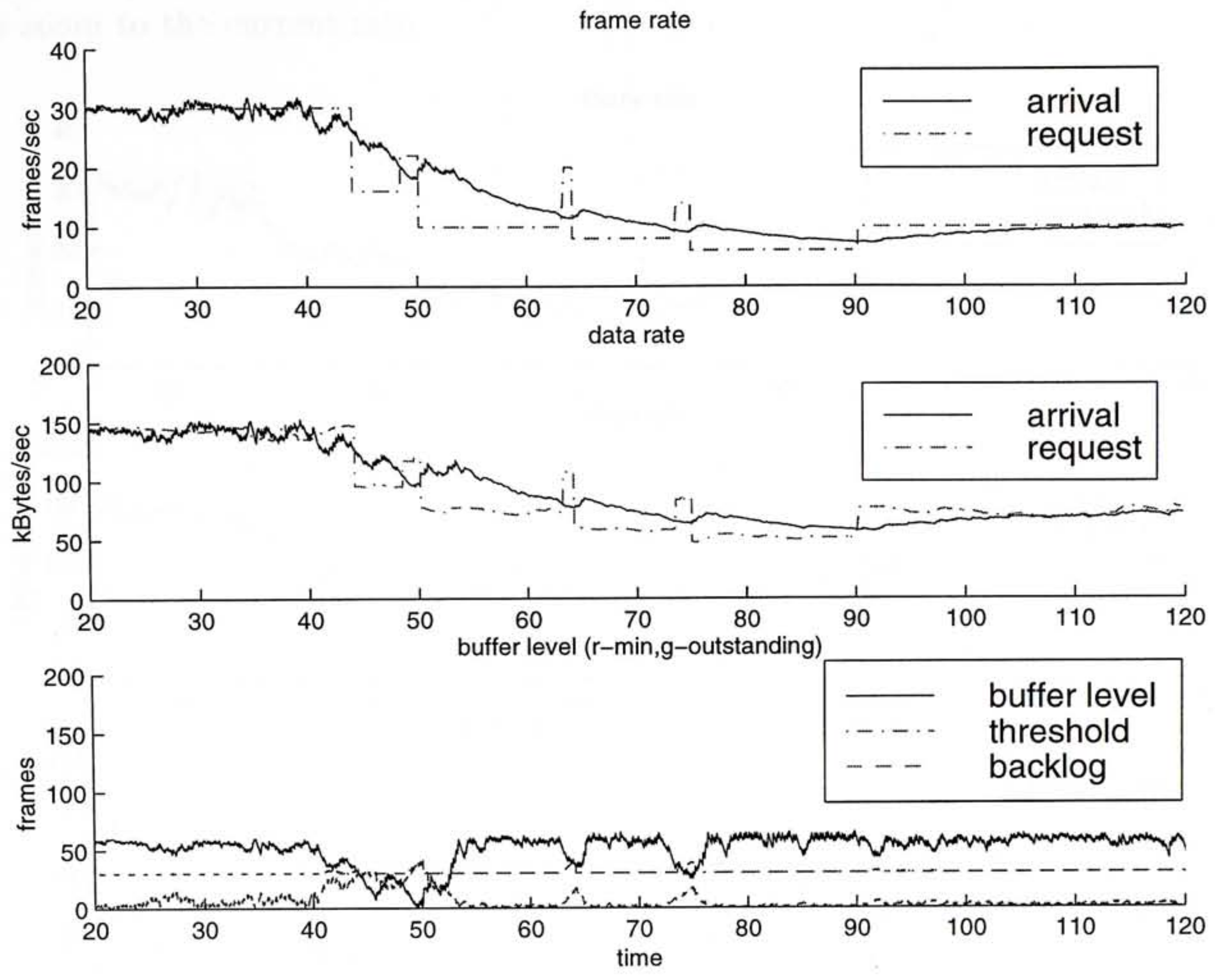


Figure 6.2: Snap Shot of the Adaptation Process with $\alpha = 0.01$

100kBytes/s to relief the load. But the first drop is not sufficient. A second drop in request rate starts at time 50s. The process repeats four times until it reaches the 75kBytes/s (or 600kbps) at time 90s, which is the maximum rate the client under test can get.

The results shows a very smooth estimation for the arrival rates, but the response of the estimation is rather slow: when the actual arrival rate drops, it takes a rather long time for the estimation to converge to the new value. The slow response time actually causes an inaccurate estimation of the current arrival rate, and therefore the algorithm needs several rounds of probing phase to zoom to the current rate.

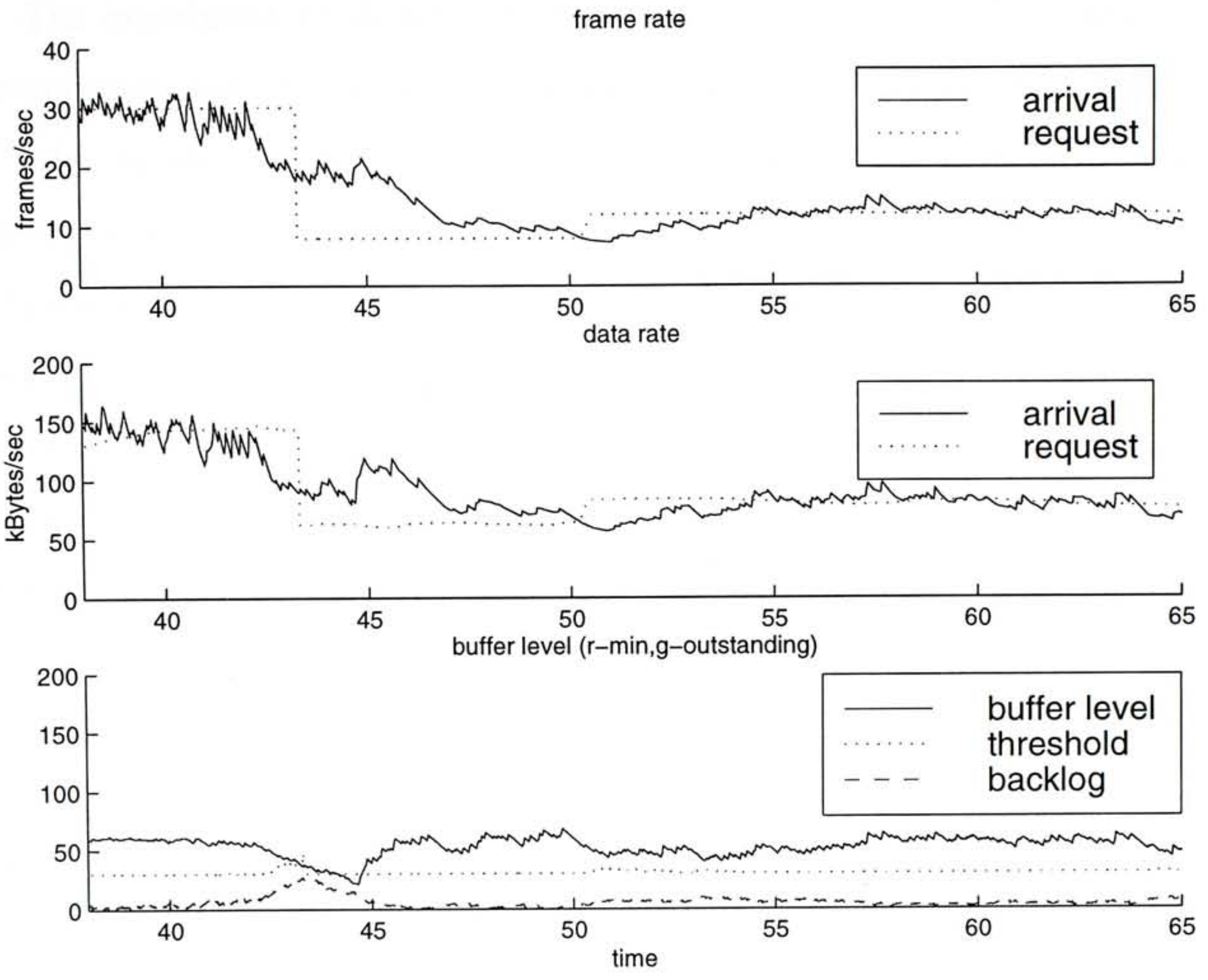


Figure 6.3: Snap Shot of the Adaptation Process with $\alpha = 0.05$

The same experiment is done again with $\alpha = 0.05$. Same set of dummy clients are used to occupied the system resources starting at time 40s. After one probing cycle the system requests at about 75kBytes/s at time 50s, which is the maximum it can get. Comparing to previous experiment with $\alpha = 0.01$ where the probing cycles take a total of 50 second to reach the current maximum, this one takes only about 10 second.

The above results confirm that $\alpha = 0.05$ is a good compromise for the smoothing factor. Simular result is obtained for UDP traffic. This value will be used in the following experiments.

The experiment result for the adaptive algorithm over TCP on local area network indicates that the algorithm works well to provide a continuous stream of video by avoiding buffer underflow at the client. In the LAN environment where the round trip time is small, the evolution of the TCP congestion window is fast enough for the connection to reach the full rate requested by application itself. Using TCP eliminates the need for loss detection and retransmission at the application layer, and thus can be handled more easily by the applications. However, the WAN environment where the round trip time of a packet can be as large as 300ms, the transmission rate of TCP evolves too slowly to reflect the actual available transmission rate. The arrival data rate measured by the application on top of TCP is inaccurate to be used in the calculation of the algorithm. As a result, we only recommend using the adaptation algorithm over TCP on local area network environment. For connections with long RTT, UDP is a better choice for transport layer protocol.

Using UDP as transport layer protocol

Next, we run the adaptation algorithm on top of UDP instead of TCP in the local LAN environment. Loss is rare until the traffic reaches the network capacity at around 8.2Mbps. Three non-adaptive dummy clients generating extra UDP traffics are used to create congestion. The total data rate used up by these dummy clients are 7.4Mbps. The snap shot on figure 6.4 shows that when the dummy clients begin their connection at time 7s, the arrival data rate for the client starts to have more fluctuation. Loss are detected at time 13s and 14.5s. By reducing the request data rate to the measured value, the network usage falls

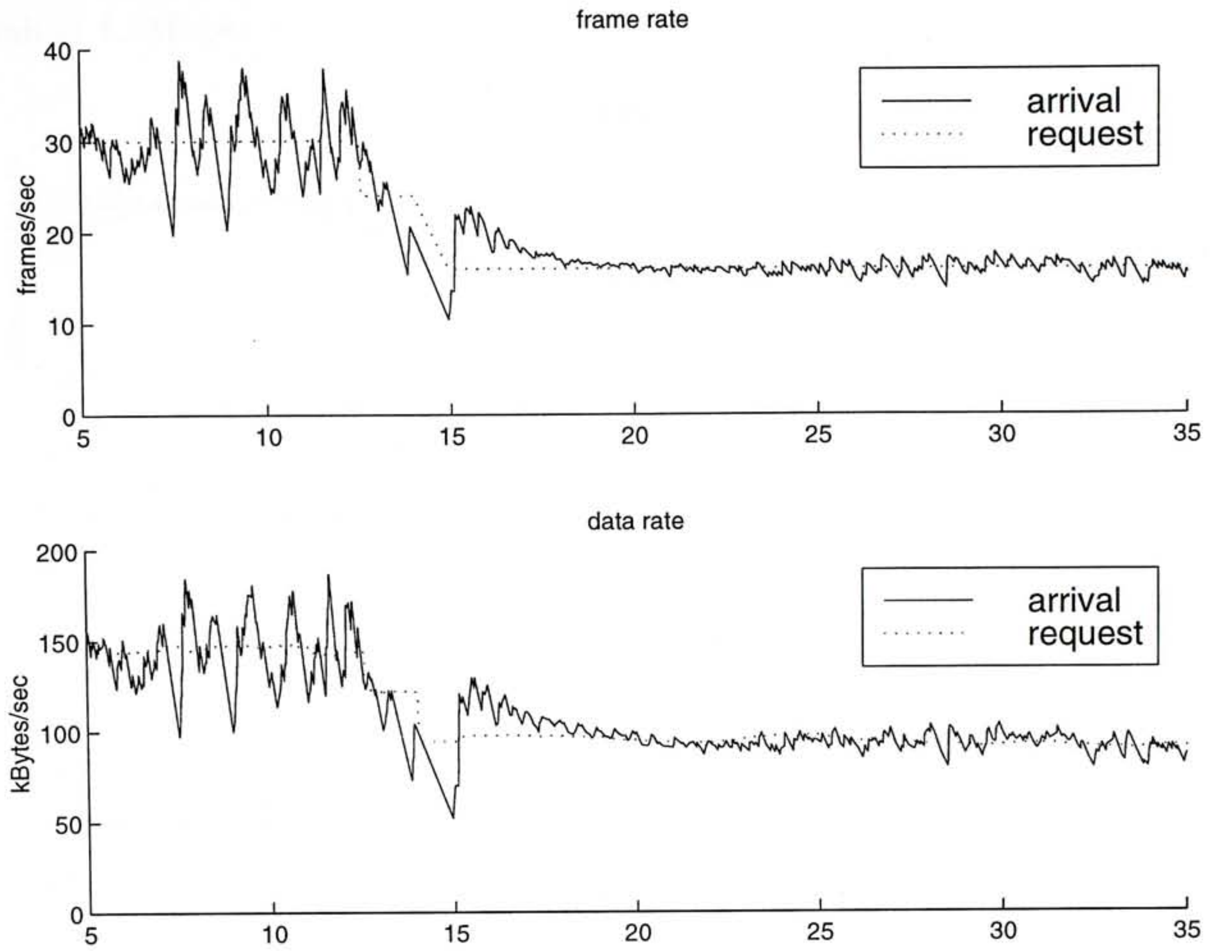


Figure 6.4: Snap Shot of the Adaptation Process in a Lossy Environment

below the capacity margin with a stablized data arrival rate.

Note from the graph that the average arrival rate before reducing the request rate is higher than that after. From this observation it appears that the algorithm is reducing the request rate too much. This is unavoidable because we are use clients to create congestion on the LAN and unlike our "polite" adaptive clients, these dummy clients do not adapt to the network traffic condition. When our adaptive client reduces the data rate, the dummy client uses up the spared data rate. The resulting data rate of the adaptive client is about 100kBytes/s or 800kbps, which together with the total rate of the dummy clients sum up the limit of 8.2Mbps.

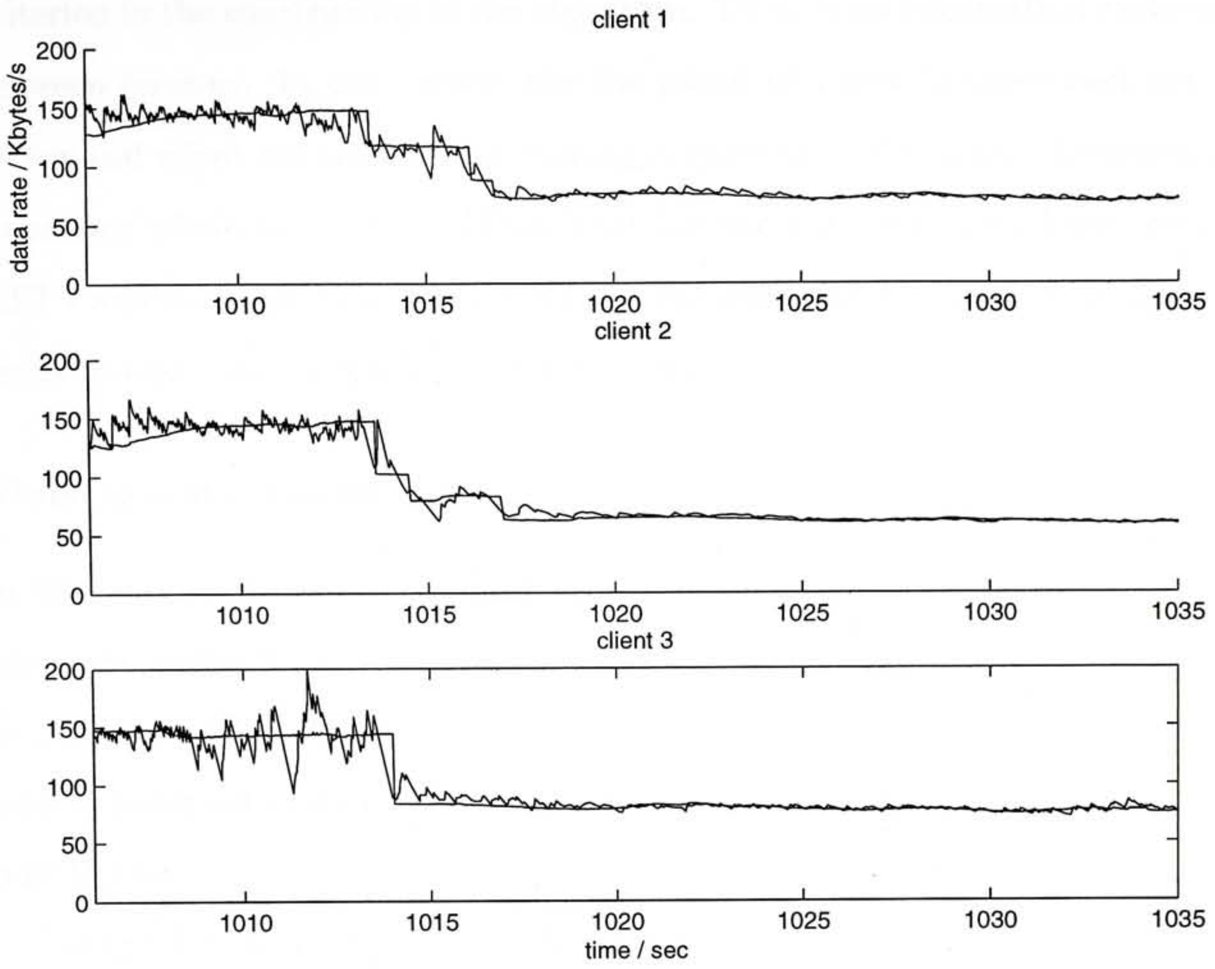


Figure 6.5: Multiple Clients Over the Same Bottleneck Link

In the next experiment, we investigate the situation where multiple clients are adaptive. When multiple adaptive sessions use the same bottleneck link, the clients have to share the bandwidth among themselves. Figure 6.5 shows the result of such scenerio. Three clients are connected to the server with a dummy client connected to the server at time 1013s, using up 6.4Mbps of the available 8.2Mbps bandwdith, leaving about 1.8Mbps for the adaptive clients. After adaptation, each client should receive about 75kBytes/s or 600kbps.

The result shows that bandwidth sharing among each client is not exactly equal but close. This is so even though fairness is not explicitly considered as a criterion in the construction of the algorithm. There is no information exchange between sessions. In cases where the the round trip time between each set of server and client are similar, the sharing is quite fair. We expect, however, in situations where some of the clients that use the bottleneck link have shorter RTT's and the other have a longer RTT's, the probe-down part of the algorithm tends to bias towards the one with longer RTT.

Video Quality Assessment

In this part we focus on using the algorithm over UDP. For the case of TCP, when the network cannot support the data rate of the video stream, buffer underflow would occur. Without using the adaptive algorithm, the client would keep running out of data. The visual effect is that video will keep pausing from time to time.

Using UDP, however, when the network cannot support the data rate of the video stream, packets will be skipped by network silently. Missing frames are stuffed at the clients side and thus buffer underflow may not occur as in the case of TCP.

For MPEG video, packet loss results in video quality degradation. When data from I or P frames are lost, subsequent P and B frames in that GOP have to be discarded. In this part we will examine the quality of the video sequences that are transmitted with and without using the algorithm in congested networks.

The same set of experiment are run twice. In the first trial, a video client that runs the adaptation algorithm competes with dummy clients that occupy the channel with fix-rated UDP traffic. The video received by this adaptive video client is recorded. In the second trial, the adaptation algorithm of the video client is turned off. It then competes with the same set of dummy clients. The video received by this non-adaptive video client is recorded. We will compare the quality of these two sets of video.

Figure 6.6 shows the frame rates of the output video of the two type of clients. The frame rate here means how many frames with actual data are played per second. Dummy frames stuffed because of request skipping, data loss from network and those extra loss e of the loss of data from an I or P frames are not counted in the frame rate.

The frame rate of the non-adaptive client fluctuates rather drastically. This means that although the average frame rate of the non-adaptive client is higher than that of the adaptive one, the non-adaptive video can have many sudden pauses. By contrast, although the adaptive one has a lower overall frame rate, its video quality is smoother in comparsion. This overall smoother video quality has been observed subjectively in our experiment.

We have shown the data rate dropped by the adaptive clients in a congested network earlier in the chapter. The result shows that the data rate dropped after applying the algorithm is more than that dropped by the network before the algorithm is used. A similar result is observed here: the data rate dropped by the adaptive client is more than that by the non-adaptive client.

Three reasons account for the better video quality of the adaptive client despite the lower rate: 1) The adaptive client selectively drops less important data (i.e. B frames) outside the network where as the non-adaptive client has no choice but to let the network drops its data randomly. If I or P frames are dropped, subsequent P and B frames in the GOP cannot be decoded and thus

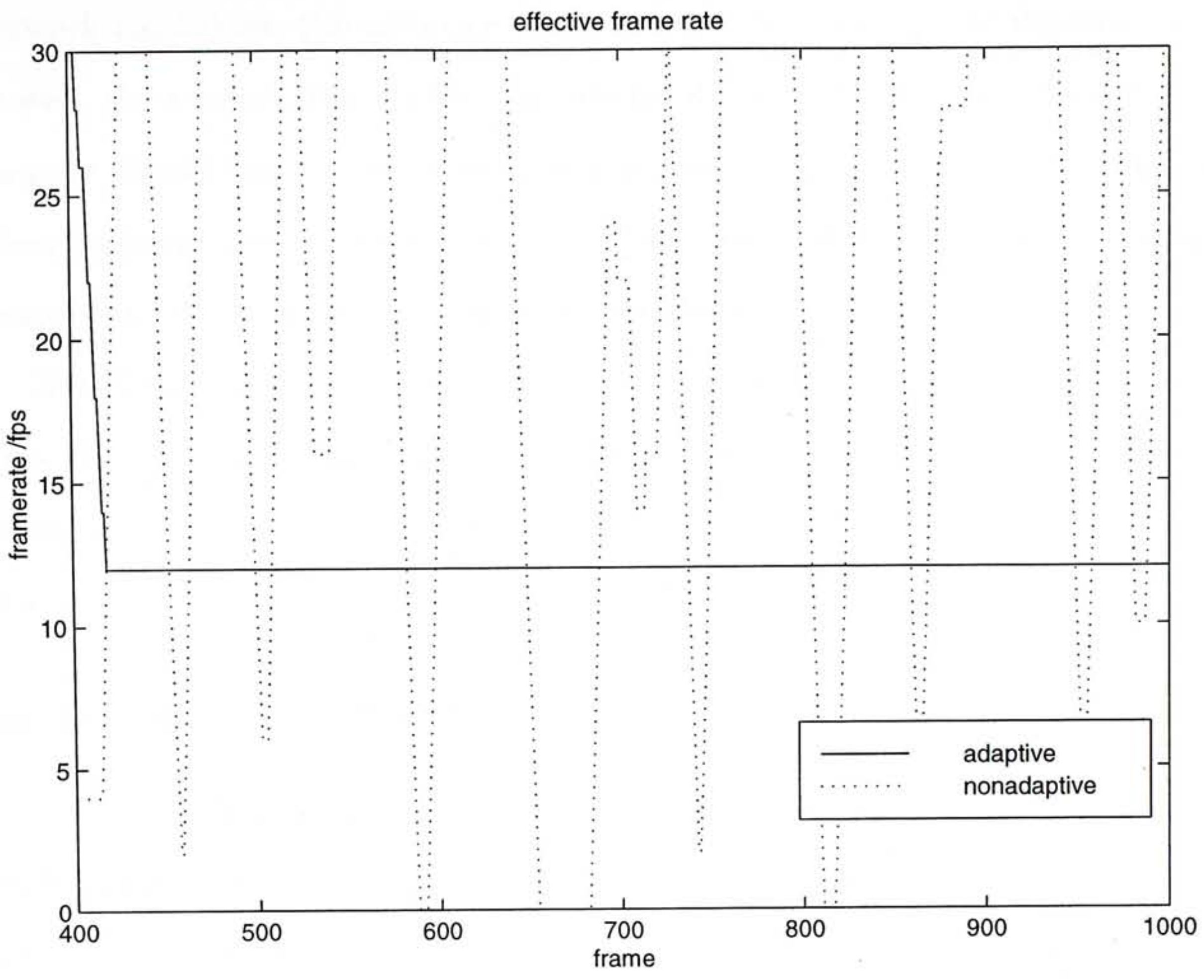


Figure 6.6: Frame Rate of the Resultant Video Sequences

results in a sudden pause. 2) The adaptive client drops a complete frame whereas the non-adaptive client may drop part of a frame in the network, which results in the drop of a complete frame at the UDP receiver anyway. The bandwidth used to transmitted that frame is wasted. 3) The spacing between dropped frames of adaptive clients are more even, resulting in smooth quality as indicated in fig 6.6.

Another observation is that the adaptive client controls the bandwidth consumption so that the total requested rate (by the adaptive client plus the dummy clients) falls below the channel capacity. Since dummy clients do not adapt to network conditions, the adaptive client reduces its share for the dummy client as well. As a result, the packet loss rate for dummy clients is lower than those dummy clients that compete with non-adaptive clients. It is interesting that when adaptive client is used, not only is its video quality better, the other non-adaptive UDP clients also receive more bandwidths.

Noted that although the bandwidth received by the adaptive client is lower than that of the non-adaptive clients, the rate of valid data is not necessary lower. Also, as shown in the results above, video quality of the adaptive client can actually be better with a smoother display frame rate.

Interaction with TCP traffic

We run the adaptive algorithm on top of UDP and let it competes with other TCP traffic. The TCP traffic is generated by dummy clients requesting at a fixed rate. When the algorithm is run on top of UDP, the protocol that first interact with TCP traffic is actually UDP. TCP uses $SlowStart$ and $CongestionAvoidance$ algorithm for flow control. The variable *cwnd*, or the TCP congestion window, controls how many bytes of data the sender can transmit. When congestion is detected, *cwnd* is set to half. This would reduce the transmission rate of the TCP connection.

There are two cases when the network become congested. The first case is that the adaptive clients are first connected and then other TCP connections join in. When the total rate exceeds the channel capacity, some packets will be dropped by the network. At this point the total rate only exceed the channel capacity by a very small amount, the loss rate is low. If the loss is on TCP connections, its congestion windows will be reduced and thus its rate is also reduced. For adaptive clients, since there is a 10% loss margin before the algorithm would skip request, it would ignore the loss. As a results, TCP connections cannot expand their congestion window (and thus the transmission rate) to a higher value.

The second case is that the network already has a handful of TCP connections while a new adaptive client makes the connections. Loss would occur as the channel exceed its capacity. Experiment results show that TCP always react faster than the adaptive clients in the sense that at least one of the TCP connection would reduce its transmission rate before significant loss occurs at the adaptive client.

In most cases, we found that TCP traffic is not as agressive as realtime UDP traffic. When UDP traffic with high data rate arrives, TCP connections cannot increase their congestion window to a larger value. Effectively, the transmission rate of the TCP connections are reduced. The result is that TCP traffic becomes transparent to the adaptive UDP traffic and therefore the algorithm would not skip any request.

6.2 Analysis

In this section, we will analyse the effectiveness of the algorithm in situations where 1) other streams are UDP streams with no rate control, and 2) other streams are UDP video streams using the same algorithm. Through the analysis, we want to evaluate the conditions under which it is better to use adaptive clients than non-adaptive clients.

This analysis only examines the amount of valid date received and it does not take into account the actual video quality. Although in general, the video quality depends on the amount of valid data received, there are cases when one receiving less could outperform one receiving more. An example is that if the display frame rate is very irregular, the video quality could become worse than one that has fewer valid frames but a more regular frame rate.

The adaptation algorithm aims at skipping video frames in a regular manner. In terms of frame rate regularity, it always performs better than letting the network drop frames randomly. In the analysis below, we would show the conditions for adaptive clients to receive more valid data than non-adaptive clients. These conditions are the sufficient conditions for adaptive clients to have better video quality than non-adaptive clients. Adaptive clients could have better video quality even when these conditions are not met, but this is case dependent and cannot be analysed in general.

In the analysis, we assume that the network nodes follow the FCFS discipline. Packet dropping occurs when the buffer at the node overflows.

We also assumed that the algorithm does not skip request more than necessary. In another word, the sum of the resulting data rate at the bottleneck link should be equal to the capacity of the bottleneck link. The experiment results show that this assumption is valid in the settings for the algorithm running on top of UDP.

We show the conditions when dropping the data rate by the algorithm would result in more valid data being received than letting the network randomly drops packets among the streams. In many situations, although adopting the adaptation algorithm may result in the application receiving less bandwidth than otherwise, the video quality is actually better. This is because with the adaptation algorithm, least important data are skipped outside the network and relatively little data are dropped randomly in the network. Whereas, if there is no adaptation, much data, including the important I frames, are randomly skipped in the network. When these frames are lost, subsequent P or B frames in the GOP cannot be decoded properly. These P and B frames, although they are received successfully, is not valid to the client.

Since these P or B frames cannot be decoded, they are considered lost. Define p as the factor accounting for such kind of loss. We call p the extra loss factor in the subsequent analysis. If the loss probability of the network is ρ, the total loss probability for the video stream would become $\rho \cdot (1+p)$. For a video stream with data rate λ_v, the total loss rate is $\lambda_v \cdot \rho \cdot (1+p)$.

The value of p is always greater than 0 for MPEG videos. Later in the section, we would give some examples of the typical values of p.

6.2.1 Interacting With Streams With No Rate Control

Define the loss probability of the link at input rate r as $\rho(r)$, the extra loss factor for the video stream given a packet loss as p, the maximum video rate as λ_v, channel capacity as λ_c and the total rate of other UDP traffic as λ_o.

When the aggregrated input rate exceeded the channel capacity, the loss probability $\rho(r)$ is approximately

$$\rho(r) = \frac{(\lambda_o + \lambda_v) - \lambda_c}{\lambda_o + \lambda_v}, \quad \text{when } \lambda_o + \lambda_v > \lambda_c$$

For non-adaptive clients, the data rate dropped D_N is

$$D_N = \rho(r) \cdot \lambda_v \cdot (1+p)$$

Let us now consider adaptive clients. Define the drop in data rate using the adaptive algorithm as d_{apt}. In the optimal case, d_{apt} should be the amount of data rate that exceeds the channel capacity, i.e. $d_{apt} = \lambda_v + \lambda_o - \lambda_c$. Further assume that there is no loss in the network (or $\rho(\lambda) = 0$) when $\lambda \leq \lambda_c$. This assumption is valid in most of the network configuations nowadays where loss is mostly due to congestion. Loss due to damage is extremely rare. Therefore, for adaptive clients, the data rate dropped D_A is

$$D_A = \lambda_v + \lambda_o - \lambda_c$$

For adaptive clients to receive more valid data than non-adaptive clients, the data dropped by the non-adaptive clients should be more than that of the adaptive clients. We have

$$\begin{aligned}
D_A &< D_N \\
(\lambda_o + \lambda_v) - \lambda_c &< \frac{\lambda_o + \lambda_v - \lambda_c}{\lambda_o + \lambda_v} \cdot \lambda_v \cdot (1 + p) \\
\lambda_o + \lambda_v &< \lambda_v + \lambda_v \cdot p \\
\Rightarrow \ p &> \frac{\lambda_o}{\lambda_v}
\end{aligned}$$

Since $\rho(r) \cdot (1 + p)$ is the overall loss probability, it must be less than or equal to 1, or

$$\begin{aligned}
1 + p &\leq \frac{1}{\rho(r)} \\
\Rightarrow \ p &\leq \frac{1 - \rho(r)}{\rho(r)}
\end{aligned}$$

Since $\rho(r)$ is approximately $(\lambda_o + \lambda_v - \lambda_c)/(\lambda_o + \lambda_v)$, and,

$$\begin{aligned} \frac{\lambda_o}{\lambda_v} &< p \leq \frac{1-\rho(r)}{\rho(r)} \\ \Rightarrow \frac{\lambda_o}{\lambda_v} &< p \leq \frac{\lambda_c}{\lambda_o + \lambda_v - \lambda_c} \end{aligned}$$

The smaller the value of λ_o/λ_v, the easier it is for the above condition to hold true. It implies that when all other traffic has no rate control and if the video stream constitutes only a very small part of the total usage, the adaptation may not be able to compete with other traffic inside the network. On the other hand, if the video itself constitutes a larger part of the overall traffic, adaptive clients could receive more valid data than non-adaptive clients.

6.2.2 Multiple Streams Running The Algorithm

Define the data rate of the streams as $\lambda_1, \ldots, \lambda_n$. To simply the analysis, let $\lambda_1 = \lambda_2 = \ldots = \lambda_n = \lambda_v$.

$$\begin{aligned} \rho(r) &= \frac{\lambda_1 + \lambda_2 + \ldots + \lambda_n - \lambda_c}{\lambda_1 + \ldots + \lambda_n} \\ &= \frac{n\lambda_v - \lambda_c}{n\lambda_v} \end{aligned}$$

For adaptive clients, assume all loss occurs outside network and are due to adaptation. This assumption is true when the total request rate falls below the channel capacity of the network. Data rate dropped by adaptive client, D_A, is,

$$\begin{aligned} D_A &= \frac{\lambda_1 + \lambda_2 + \ldots + \lambda_n - \lambda_c}{n} \\ &= \frac{n\lambda_v - \lambda_c}{n} \end{aligned}$$

For non-adaptive clients, assume loss are evenly distributed among the streams. The data rate dropped by non-adaptive client, D_N, is,

$$\begin{aligned} D_N &= \rho(r) \cdot \lambda_v \cdot (1+p) \\ &= \frac{n\lambda_v - \lambda_c}{n\lambda_v} \cdot \lambda_v \cdot (1+p) \end{aligned}$$

For the clients running the algorithm to have better video quality, $D_A < D_N$

$$\begin{aligned} D_A &< D_N \\ \frac{n\lambda_v - \lambda_c}{n} &< \frac{n\lambda_v - \lambda_c}{n\lambda_v} \cdot \lambda_v \cdot (1+p) \\ p+1 &> 1 \\ p &> 0 \end{aligned}$$

For MPEG video, p is always larger than 0. Therefore, in the optimal case, the algorithm always perform better.

6.2.3 Calculation of p

The extra loss factor p measures the additional amount of discarded data when an I or P frame is lost. Assume the original loss probability is low, such that the probability of losing more than one frame in an GOP is negligible. Then p equals the average number of frame loss in an GOP minus 1. Consider a particular MPEG stream with GOP pattern $IPBBPBBPBB$ (and assume that the loss probability of losing I, P and B frame is proportional to its size, of which the ratio is 10:5:1). Since losing an I or P frame would render the subsequent P or B useless, losing an I frame would mean a total loss of 10 frames (all frames in the GOP). Similarly, losing the first P frame would mean a loss of 9 frames (all frames except the I frame), and so on. We have,

$$\begin{aligned} p &= \frac{10(10)+9(5)+1+1+6(5)+1+1+3(5)+1+1}{10+5(3)+1(6)} - 1 \\ &= 6.3 \end{aligned}$$

From the previous analysis, p must be greater than λ_o/λ_v such that $D_A < D_N$. A typical example on local area network where $\lambda_c = 9Mbps$, $\lambda_o = 8Mbps$ and $\lambda_v = 1.5Mbps$ (assuming an MPEG-1 stream). The loss probability would then be $\rho = 0.5/9.5 = 0.05$. To have $D_A < D_N$, p must be greater than $8/1.5 = 5.3$. And it is so in the previous example.

Since there are no restriction on the GOP pattern or IPB frame size ratio in the MPEG video specification, different encoders may give very different values which result in the variation on the value of p. Typical GOP patterns have 9 to 15 frames, interleaving 2 to 3 B frames between two I frames or P frames. Frame size ratio $I : P : B$ could range from 4:2:1 to 40:10:1. With such variations, the value of p could range from 3.2 ($IBBPBBPBB$, 4:2:1) to 9 ($IBBPBBPBBPBBPBB$, 40:10:1). In general, the larger the difference between the frame size of I, P and B frames, the larger the value of p; the larger the GOP, the larger the value of p.

In summary, this analysis shows the condition for adaptive clients to receive more valid data than non-adaptive clients. In situation where the video stream with a large extra loss factor p and when it constitutes a significant part of the total bandwidth usage, adaptive clients perform better than non-adaptive ones in terms of the amount of valid data received.

Chapter 7

Conclusions

We have proposed a client-pulled end-to-end application level frame-rate adaptation scheme for streaming video that dynamically adjusts the request rate and window size based on the measurement of arrival rate buffer occupancy and round trip time on the client side to accommodate the current network condition. Using the statistics at the client side, the client controls the data rate of the session by adjusting the request rate to retrieve a continuous flow of video from the server over the network.

Since most of the adaptation process, including statistic gathering and decision making, is done on the client side, the server is left with the job of reading and sending video data only. By off-loading the server with the adaptation process, this video delivery system can be more easily scaled up to support many clients.

Results show that using this algorithm, applications quickly react to congestion and can zoom to a suitable request rate within tens of seconds. Video continuity is preserved in most cases.

The video delivery adaptation algorithm is distributed. Each client adapts to the traffic condition based on their own measurement of the network traffic. There is no information exchange between clients. While the algorithm does not explicitly impose rules about fairness among different clients, multiple sessions using the same algorithm along the same path show similar performance. Fairness among each client along the path is not absolute but is acceptable.

A potential drawback of this algorithm is that the application that makes use of it cannot compete with other sessions that do not run any rate control algorithm, e.g. raw UDP sessions. While this algorithm tries to reduce the bandwidth usage in case of congestion, other applications that do not have any rate adaptation control might use up the spared bandwidth, thus the effort of this algorithm is wasted.

This phenomenon is not unique to this algorithm. In fact, in the network where no policing is done on the connections, bandwidth spared by connections with rate control is easily used up by those without rate control. TCP, the most wildly used transport protocol, also suffers from this problem. If there are indeed greedy applications trying to use up the available bandwidth by UDP connections, other applications that use TCP will not be able to get any chance to transmit their data. Given that most of the application on the Internet is of TCP nature (for example, HTTP and FTP traffics), this algorithm works to obtain a fair share of bandwidth.

For many video formats (such as MPEG), the decoding of video frames depends on frames received previously. Take MPEG as an example, if an I frame is dropped, subsequent P or B frames in the GOP cannot be decoded. An undesired dropped frame renders another cluster of received frames useless.

This is a waste of network resources. It would also result in inregularity in display rate, which is disturbing to the eyes. Using the algorithm proposed in the thesis, we selectively drop the frames in the video stream that minimize the effect to the decoding of other frames. The resulting video has a more regular display rate and thus is more pleasant to the viewers. This is so even though sometimes more frames are skipped with the use of adaptive algorithm. In this case we achieve a better video quality using less bandwidth.

We have shown that using the adaptation algorithm to skip frames outside the network results in better video quality than letting the network skip frames randomly. We have also shown that when the video stream has a high extra loss factor p, it is more likely for adaptive clients to work better than non-adaptive ones.

In conclusion, this algorithm is rather effective and is suitable for best effort point-to-point video-on-demand applications over channels where bandwidth is not abundant.

Bibliography

[1] A. Mukherjee, "On the Dynamics and Significance of Low Frequency Components of Internet Load", *Internetworking: Research and Experience*, 1994, Vol.5, pp 163–205

[2] D. Sanghi, A. K. Agrawala, Gudmundsson and B. N. Jain, "Experimental Assessment of End-to-end Behavior on Internet", *Proceeding of IEEE INFOCOM '93*, March 1993, pp. 867-874.

[3] V. Paxson, "End-to-End Internet Packet Dynamics", *Proceeding of SIGCOMM 97*, 1997.

[4] Jean-Chrysostome Bolot, "Characterizing End-to-End Packet Delay and Loss in the Internet", *Journal of High-Speed networks*, September 1993, vol. 2, no. 3, pp. 305-323.

[5] J-C. Bolot, T. Turletti, I. Wakeman, "Scalable feedback control for multicast video distribution in the Internet", *Proc. ACM Sigcomm '94*, London, UK, Sept. 1994. pp. 58-67.

[6] J-C. Bolot, T. Turlett, "A rate control scheme for packet video in the Internet", *Proc. IEEE Infocom '94* pp. 1216-1223, Toronto, Canada, June 1994.

[7] R. Carter and M. Crovella, "Measuring Bottleneck Link Speed In Packet-Switched Networks", Tech. Report BU-CS-96-006. Computer Science Department, Boston University, March 1996

[8] Wu-chang Feng, D. Kandlur, D. Saha and G. Shin, "Understanding TCP Dynamics in an Intergrated Services Internet", *Proceeding of the IEEE 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, 1997, pp. 279-290.

[9] J. Gecsei, "Adaptation in Distributed Multimedia Systems," *IEEE Multimedia*, April-June 1997, pp. 58–66.

[10] S. Chakrabarti and R Wang, "Adaptive Control for Packet Video", *Proceedings of the International Conference on Multimedia Computing and Systems '94*, 1994, pp 56–62.

[11] T. V. Lakshman, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss", *IEEE/ACM Transactions On Networking*, June 1997, vol. 5, no. 3, pp. 336-350

[12] S.L. Blake, S. A. Rajala, F. Gong and T. L. Mitchell, "Efficient Techniques For Two-Layer Coding of Video Sequences," *Proceedings of ICIP '94*, 1994, pp. 253–257.

[13] S.C. Liew and C.Y. Tse, "Video Aggregation: Adaptive Video Traffic for Transport over Broadband Networks by Integrating Data Compression and Statistical Multiplexing", *IEEE J. on Selected Areas in Commun.*, Vol. 14, no.6, pp 1123-1137, Aug. 1996.

[14] S. C. Liew and C. Y. Tse, "A Control-Theoretic Framework for Adaptation of VBR Compressed Video for Transport over a CBR Communications Channel", Proc. ICIP '96 (A more complete version submitted to Transactions on Networking).

[15] S. C. Liew and A. Yeung, "Multiplexing Video Traffic using Frame-Skipping Aggregation Technique", *IEEE ICIP 97*, pp. 334-337

[16] P. Goyal, H. M. Vin, C. Shen and P. J. Shenoy, "A Reliable, Adaptive Network Protocol for Video Transpor", *Proceeding of IEEE INFOCOMM '96*, 1996, pp. 1080-1090.

[17] P. Pancha and M. El. Zarki, "Prioritized Transmission of Variable Bit Rate MPEG Video", *IEEE GLOBECOM '92*, 1992, pp 1135–1139

[18] H. Kanakia, P. Mishra and A. R. Reibman, "An Adaptive Congestion Control Scheme for Real Time Packet Video Transport", *IEEE/ACM Transactions on Networking*, vol. 3, no. 6. December 1995. pp 671-681

[19] V. Jacobson, "Congestion Avoidance and Control," *Computer Communication Review*, Vol. 18, No. 4, 1988, pp. 314–329.

[20] V. Jacobson, "Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno", *Proceedings of the 18th Internet Engineering Task Force,*, 1990, pp. 365.

[21] P. Pancha and Magda El Zarki, "Prioritized Transmission of Variable Bit Rate MPEG Video",

[22] M. Krunz and H. Hughes, "A Performance Study of Loss Priorities for MPEG Video Traffic", *Proceeding of IEEE ICC 95*, June, 1995.

[23] M. Krunz and H. Hughes, "A Traffic Model for MPEG-Coded VBR Streams", *Performance Evaluation Review (Proceedings of the ACM SIGMETRICS '95)*, pp. 47-55, 1995.

[24] M. Krunz and S. K. Tripathi, "On the Characterization of Bit Rate Variations in MPEG Sources", *Proceedings of ACM SIGMETRICS'97 Conference*, Vol. 25, No. 1, pp. 192-202, June 1997.

[25] M. Krunz and S. K. Tripathi, "Exploiting the Temporal Structure of MPEG Video for the Reduction of Bandwidth Requirements", *Proceedings of IEEE INFOCOM'97 Conference*, pp. 67-74, April 1997, Japan.

[26] R. T. Apteker, J. A. Fisher, V. S. Kisimov and H. Neishlos, "Video Acceptability and Frame Rate", *IEEE Multimedia* vol2. issue 3, 1995, pp. 32-40.

[27] S. Loeb, "Delivering Interactive Multimedia Documents over Networks", *IEEE Communications Magazine*, May 1992, vol. 30, issue 5, pp. 52-59.

[28] G. C. Goodwin and K. S. Sin, "Adaptive Filtering Prediction and Control", *Englewood Cliffs, NJ: Prentice-Hall* 1984.