Effectiveness of TCP for Video Transport

1. Introduction

In this project, we evaluate the effectiveness of TCP for transferring video applications that require real-time guarantees. Today's video applications usually run on enhanced UDP protocols rather than TCP. The reason for this is the belief that TCP's retransmission-based reliable delivery and source-based congestion control algorithms make it unsuitable for real-time video transport. In this project, we have examined these properties of TCP in terms of video transport using NS-2 network simulator.

We first identify the problems with TCP that might lead to unacceptable delays in stored or real-time video transport. After a thorough discussion of these problems, we present our simulation results showing how crucial these problems are. Then we describe how these problems can be alleviated, if not removed totally. We propose enhancements to TCP, which can make it work better for video transport. We evaluate the effectiveness of our enhancements.

2. Properties of Streaming Video – Why Do Problems Arise?

There are several distinguishing characteristics of streaming media that might make it unsuitable for today's network protocols and networks. Perhaps the most important of these is the variable bit-rate (VBR) nature of streaming video. Due to the properties of encoding, which we have examined in the beginning of this semester, each frame can be of different sizes. Especially I frames of MPEG video are much larger than B and P frames. This means that I frames will span many more TCP packets than B or P frames. Therefore, the transmission time for I frames is greater than the transmission time of B or P frames. One important consequence of this is that the probability with which a packet of an I frame will be dropped is higher than the probability with which a packet of a B or P frame will be dropped. This is not desirable because B and P frames are dependent on I frames. The nature of encoding can thus affect the quality of video that is transported over today's networks which drop packets. A protocol that would provide guarantees for video transport should exploit the encoding of the video.

Another requirement of streaming video is the frame rate. If the transported video stream is not displayed at a determined rate, there will be discontinuities in the stream and the visual quality of the video will be degraded. To guarantee such a frame rate, the underlying network needs to provide some end-to-end delay guarantees to the video stream. Unfortunately, current networks are not capable of providing such guarantees.

One property of streaming video that might make the implementation of a protocol targeted for video transport easier is its loss resilience. Many video applications are tolerant to packet losses as long as packet losses are not bursty and do not occur at critical points. We can exploit this loss-tolerance property of soft-real time video applications to relax the reliable transmission criteria.

3. Data Transport Protocols – Fairness Issues

A very important function of data transport protocols that operate over best-effort networks is congestion control. Congestion control mechanisms are thought to be critical to the stable functioning of the internet. Most current internet traffic (90-95%) uses the TCP protocol which implements congestion control.

Due to the growing popularity of streaming media applications, a number of applications are being implemented using UDP. The reason for this is the ineffectiveness of TCP for streaming video applications. As UDP does not implement congestion control, protocols or applications that are implemented using UDP should detect and react to congestion in the network. Ideally, they should implement congestion control in a fashion that ensures fairness when competing with the existing TCP traffic in the internet. In other words, these applications should be TCP-friendly. Otherwise, these applications may obtain larger portions of the available bandwidth than TCP-based applications.

Most protocols on the internet currently do not provide mechanisms suitable for the delivery of time-sensitive streaming video data. Somewhat orthogonal to this is the concern that media applications can use improper congestion control or may not implement congestion control at all, which may lead to a collapse of the internet or unfairness among internet protocols. Therefore, we need a protocol that implements congestion control and yet still provides the guarantees required by video transport.

4. Overview of TCP and Its Limitations for Video Transport

4.1. General Overview of TCP

TCP is a sophisticated transport protocol that offers reliable, connection-oriented, byte-stream service. This kind of service is useful for many best-effort applications because it frees the application from having to worry about missing or reordered data.

TCP guarantees the reliable, in-order delivery of a stream of bytes. It includes a flow control mechanism that allows the receiver to limit how much data the sender can transmit at a given time. TCP also implements a highly-tuned congestion control mechanism. The reason for such a mechanism is to keep the sender from overloading the network.

4.2. Connection Establishment and Teardown

TCP is a transport level protocol that requires a connection between two peers to be established before any data packets are exchanged. To support connection establishment and teardown, TCP specifies a state-transition diagram to be implemented by each peer. Connection establishment is actually a feature of TCP that is useful for video transport.

4.3. Sliding Window Algorithm

TCP employs a sliding window algorithm which guarantees reliable and ordered delivery and provides support for flow control. On the sending side, TCP maintains a send buffer. This buffer is used to store data that has been sent but not yet acknowledged. This buffer also stores data that has been written by the sending application but not yet transmitted. On the receiving side, TCP maintains a receive buffer. This buffer holds the data received out of order. Also it holds the data that is in correct order. Application process may not have had a chance to read the data that is in the receive buffer. These two buffers have maximum sizes. If the sender is sending at a faster rate than the receiver can consume, the receive buffer will get full. TCP does not drop packets in that case. However, each time a packet is received in the receive buffer, TCP sends how many more bytes the receive buffer can hold to the sender. Hence, the protocol makes the

sender aware of the buffer kept on the receiver side. Using this information the sender can regulate its rate so that the receive buffer does not overflow. This mechanism is called the flow control mechanism and is different from the congestion control mechanism we will describe later.

4.4. Reliable Transmission

This is one of the features of TCP that is perhaps most destructive for video transport. TCP regards each packet very important and does not drop any packets. To achieve this, after the sender sends a packet to the receiver, it waits for an ACK (acknowledgement that the receiver has received the packet) signal (packet) from the receiver. Each TCP packet of a connection is assigned a sequence number. If the sender does not receive an ACK for a particular sequence number within a specified amount of time (timeout), it assumes that the packet was dropped by the network and retransmits the packet. Note that the sliding window algorithm cannot make progress if the ACK of an earlier packet is not received by the receiver. Hence, receiver side may wait no packets may be transmitted only if a single packet arrives very late on the receiver side. This is a serious drawback in case of video transport.

In case of video transport, as mentioned above, not all packets have equal value. Some packets are more important than others due to the nature of video encoding. In MPEG encoding, packets of I-frames are more important than packets of P-frames because losing an I frame means losing all the P-frames that are dependent on the lost I-frame. This is one property of video that can be exploited by a transport layer protocol.

Another drawback of reliable transmission is that it is not really required for video transport. Video transport is quite tolerant to data loss as long as packet losses are not too frequent or packets dropped are not very crucial for decoding. Therefore it is ok to lose packets once in a while (Of course bursty packet losses should be avoided). TCP reliable transmission does not take into account this factor. Worse than this, reliable transmission exacerbates the problem by unnecessarily delaying packets that are sequenced after a lost packet. This leads to large amount of decoding delays and results in unacceptable video quality on the receiver application.

One more drawback of TCP's retransmission-based reliability is the latency of retransmissions. Each retransmission that is due to a packet loss adds at least one round trip latency to the receiver application. For low-latency applications, such as video and audio, the addition of such latency can lead to missed deadlines. Hence, we would like to keep the number of retransmissions as small as possible for low-latency applications. Only those packets that are crucial, if any, need to be retransmitted.

4.5. Congestion Control

Another feature of TCP that is detrimental for video transport is congestion control. TCP employs end-to-end congestion control by which the sender predicts or infers whether congestion has occurred or will occur in the network and adjusts its sending rate based on this determination. The purpose of congestion control is not to overload the network. There are several different congestion control algorithms employed by TCP. We will not describe them in detail in this report. But we will give the basic idea of implementation of TCP congestion control algorithms.

The basic idea of TCP congestion control is for each source to determine how much capacity is available in the network, so that it knows how many packets it can safely have in transit [1]. Once a given source has this many packets in transit, it uses the arrival of an ACK as a signal that one of its packets has left the network. It therefore infers that it is safe to insert a new packet into the network without adding to the level of congestion. This feature of TCP is said to be self-clocking, because TCP uses ACKs to pace the transmission of packets.

Some of the congestion control algorithms employed by TCP are additive increase/multiplicative decrease, slow start, and fast retransmit/fast recovery. All of these mechanisms rely on ACK signals.

The implications of source-based congestion control for video transport are very extensive and crucial. This means that the rate of a flow can change arbitrarily in the source of the flow due to the inferences source makes about the bandwidth availability in the network. If the video needs to support a certain rate (which all video streams do) then fluctuations in the rate allocated to the source may cause many delays. The receiver will receive the video frames in a delayed fashion and this will degrade visual quality. However, on the other hand, congestion control has to be employed in a network that does not provide any guarantees so that the network remains stable. Therefore, even though congestion control is bad for video transport, it needs to be there as long as applications are not given quality of service guarantees by the network and the network does not employ some kind of congestion control.

4.5.1. More on TCP's Congestion Control Algorithms

TCP's congestion control behavior is primarily dictated by how the size of its sliding window is controlled. The sender's window keeps track of all the unacknowledged data and this is an efficient way of ensuring reliability. The size of the sender's window represents the amount of data in transit. A sender that is maintaining a full window of packets releases a packet into the network only when it receives an acknowledgement for the receipt of a packet within the window. This rule ensures that the number of outstanding packets is kept constant. The size of the window is determined by the congestion control algorithm.

TCP's slow start mechanism allows it to begin utilizing all the available bandwidth quickly by doubling its window size every round trip time. This provides good throughput without a large wait to discover the available bandwidth. After discovering the available bandwidth, TCP attempts to avoid congestion by additively increasing its window when there are no packet losses and by multiplicatively reducing its window size when a packet loss is detected. Fast retransmit and fast recovery allow TCP to recover from single packet losses by waiting for three duplicate ACKs, retransmitting the lost packet and then resuming the normal sends with the half window size.

5. Limitations of TCP for Video Transport Summarized

Here are the main problems with TCP that will hinder video transport:

1. TCP is too reliable, whereas many video applications are error-resilient. Enforcing strict reliable delivery is not required for video data. Such strict reliability constraints lead to many consecutive deadline violations under congestion.

2. TCP does not have any notion of deadlines of a packet. Every packet is equal in the eyes of TCP. This is good for best effort delivery. But for video data, a packet whose deadline has already passed is totally useless.

3. These properties of TCP are inherently embedded in the sliding window algorithm which ensures reliable and ordered delivery of data packets.

6. Evaluation of The Shortcomings of TCP for Video Transport

In this section, we evaluate the shortcomings of TCP using our simulations on NS network simulator. We have performed several experiments to show that TCP is not suitable for video transport. All of these experiments measure the performance of the transport protocol based on the following two metrics:

1. The initiation latency required to maintain acceptable quality of the video stream. (By acceptable quality we mean no frames are skipped and no deadlines are violated. This might be too strict a criterion, but it provides an upper bound for us.)

2. The percentage of deadline violations observed for shorter initiation latencies.

6.1. Simulator Parameters and Experimental Setup

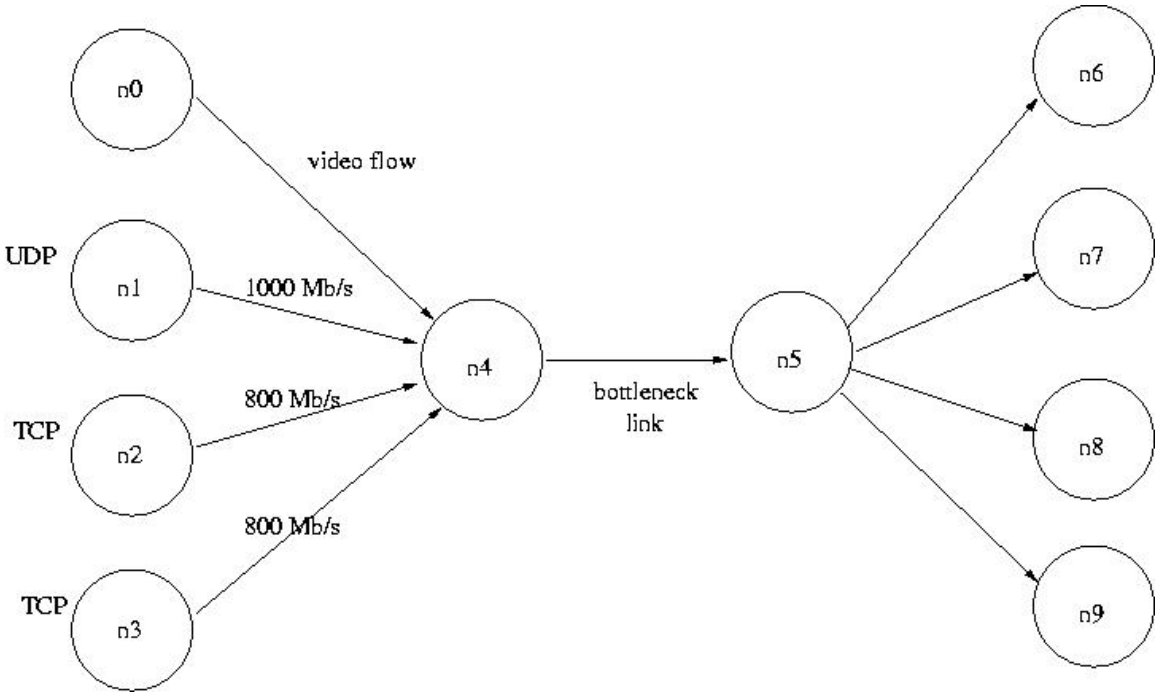


Figure 6.1. Structure of the network topology simulated.

The network simulated is shown in Figure 6.1. The rates of the senders are denoted on the corresponding links. Different experiments were carried out as shown in the graphs in the following sections. Only one of the nodes generated video traffic and the other nodes generated varying kind of traffic like CBR over UDP, FTP over TCP, Poisson VBR traffic over UDP. The bandwidth allocated to these links varied with the experiments. Some of the sources were made bursty and some sent continuous traffic. The link between n4 and n5 served as the bottleneck link for the experiments. The goal of

the experiments was to study how was the video traffic affected in the presence of other traffic sources.

The following table gives the values of the default simulation parameters used.

Parameter	Value
Packet Size	1500 bytes
ACK Size	40 bytes
Bottleneck Link BW	Varies
Bottleneck Link Delay	20 ms
Source/Dest Link BW	Varies
Source/Dest Link Delay	10 ms
Simulated Time	10 seconds
Total number of clients supported	Varies (from 3 – 10)
Bursty clients supported	Yes
Traffic pattern of the clients	FTP over TCP, CBR over UDP, Poisson VBR traffic over UDP.

Table 6.1. Default Simulation Parameters

We were mostly interested in measuring the throughput which was allocated to the flow sending the video traffic in most of the experiments. Trace files to generate CBR/VBR traffic were used. Since NS-2 doesn't support VBR traffic, we emulated the VBR traffic by breaking the VBR packets into multiple packets of fixed size (we chose 1500 bytes – TCP packet size). Also assuming that we need to transmit 30 frames/sec, the time for transmitting each of these packets was set accordingly. A script was written to do this which converted the ASCII trace file (vbr.trace1) into a format which can be understood by the ns.

All traces were enabled for the output in NS, since we didn't have much idea on how to set it for individual flows. We just needed to analyze the output for video traffic in all the experiments. A script was also written to parse the output file and calculate the throughput which the video flow received for a given set-up.

Calculation of initiation latency: As discussed earlier, we converted the frames in the video file into multiple packets. The frames were then transmitted from the node 0 to node 7 using TCP-Reno. Since the video files were huge, we just transmitted a sub-sample of the entire video file in the simulation time of 10 seconds. This sub-sample and throughput which the flow received was used to calculate the actual time it would have taken to transmit the entire video file. We assumed a frame rate of 30 frames/second for playback. The initiation latency was calculated as follows:

Throughput = (Number of packets received * 1000) / (Simulation time)

Time taken to transmit the entire file (T) = (Total size of the file) / (Throughput)

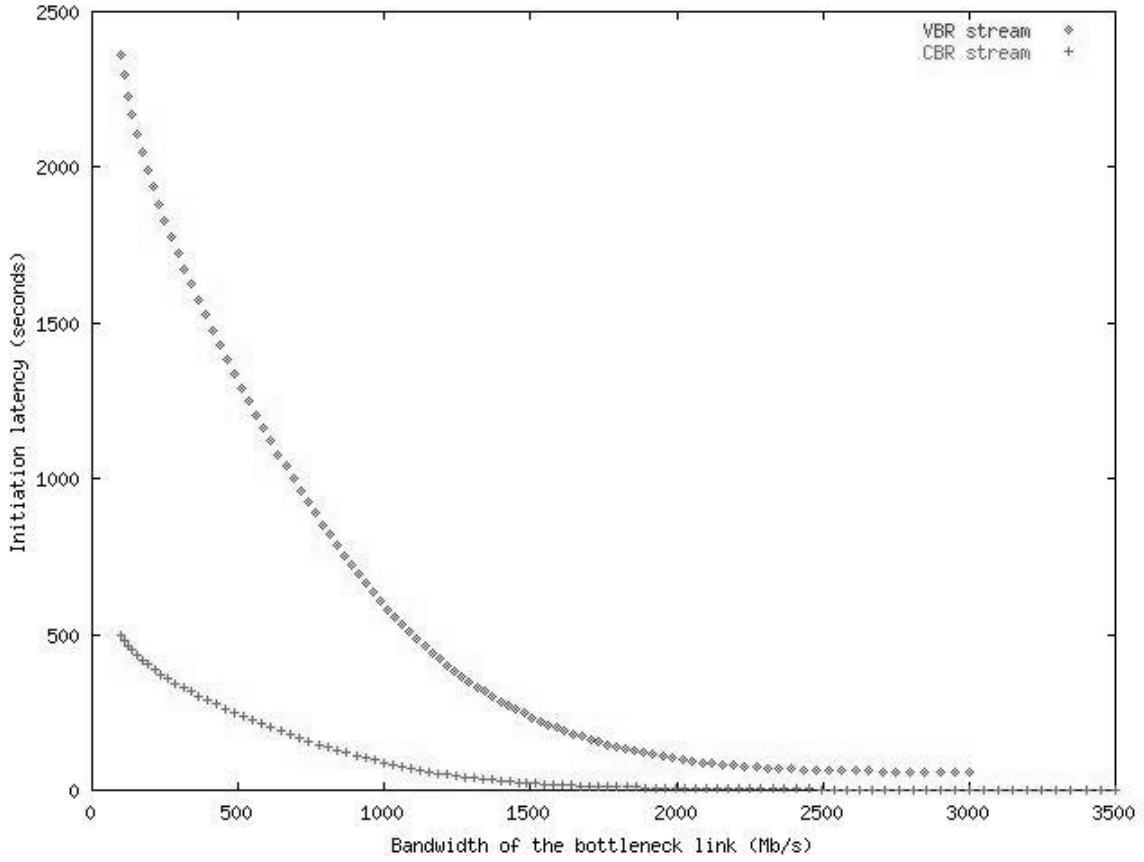
Playback Time (D) = (Total number of frames) / 30

Initiation Latency (I) = (T – D)

6.2. Performance Results for TCP

In this section we show that TCP results in unacceptable performance for video transport (especially VBR) when the network is congested.

6.2.1. Performance Metric: Initiation Latency



Graph 6.1. Initiation latency required for 0% deadline violations vs. Bandwidth of the bottleneck link.

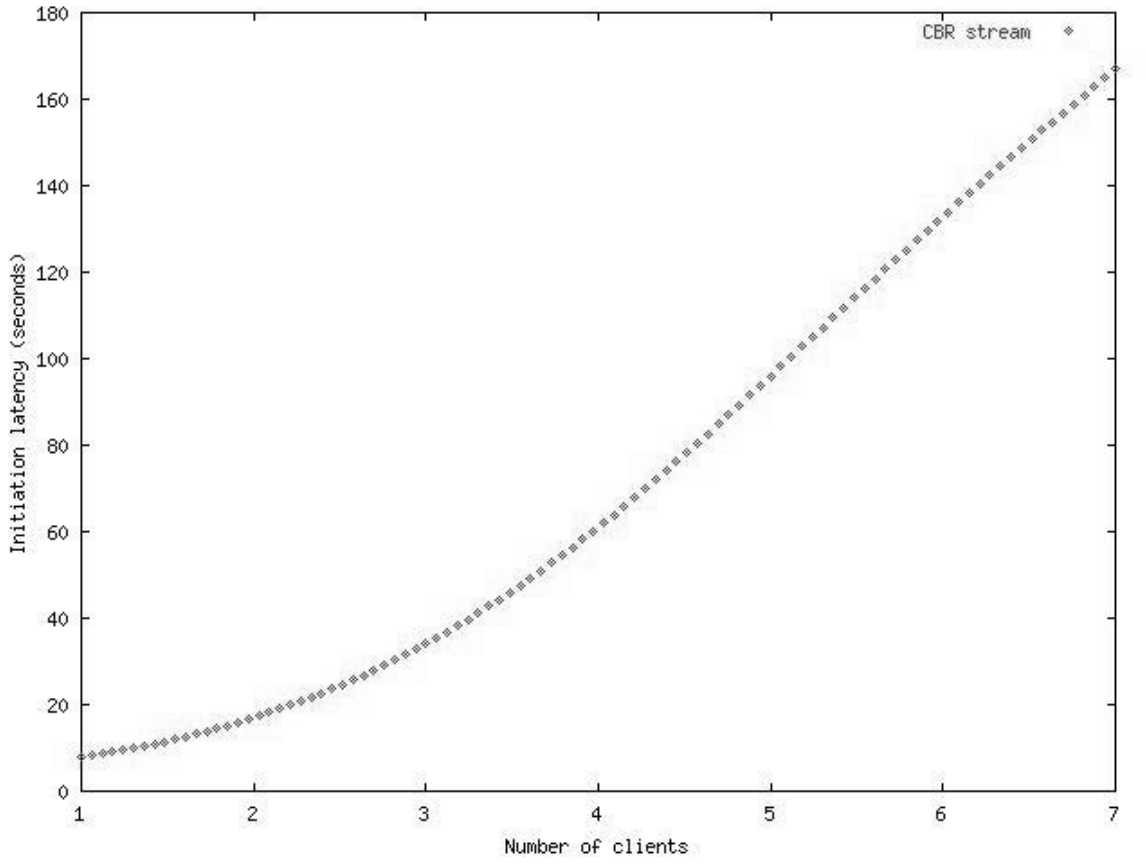
Graph 6.1 shows how the initiation latency required for no deadline violations varies as the bandwidth of the bottleneck link is varied. There are several important conclusions we can draw from this figure:

1. As expected, as the bandwidth of the bottleneck link increases, the initiation latency required for no deadline violations decreases. In fact, this decrease appears to be an exponential decrease. This result is expected because as we increase the bandwidth of the link more packets will get a chance to go faster, fewer packets will be dropped. How does the dropping of packets affect the performance of the TCP transport protocol? When a packet is dropped, the sender should first recognize that a packet is dropped and send another copy of the packet to the receiver. This is called the retransmission of the packet.

The packet finally becomes out of consideration by the sender when the sender receives the acknowledgement for that packet. Hence, a single packet drop increases the latency by one round-trip delay in TCP.

As the link bandwidth is decreased, more and more packets get dropped in the bottleneck router. This means that the latency to transmit each of these packets will increase by at least one round-trip delay between the sender and the receiver. Therefore, we get a curve as shown in Graph 6.1.

2. We see that the difference between VBR and CBR video are drastic. VBR video requires much higher initiation latencies as compared to CBR video. For a link bandwidth of 1000 Mb/s, the initiation latency required by the VBR stream is 453 seconds (around 9 minutes of buffering), whereas the initiation latency required by the CBR stream is only 11 seconds. This means that the CBR stream performs quite well compared to the VBR stream. Therefore, TCP is much less suitable for variable bit rate streams which usually have bursty behavior. Based on this result, we will argue that bandwidth smoothing at the sender might be a viable option to mask the inadequacies of TCP.

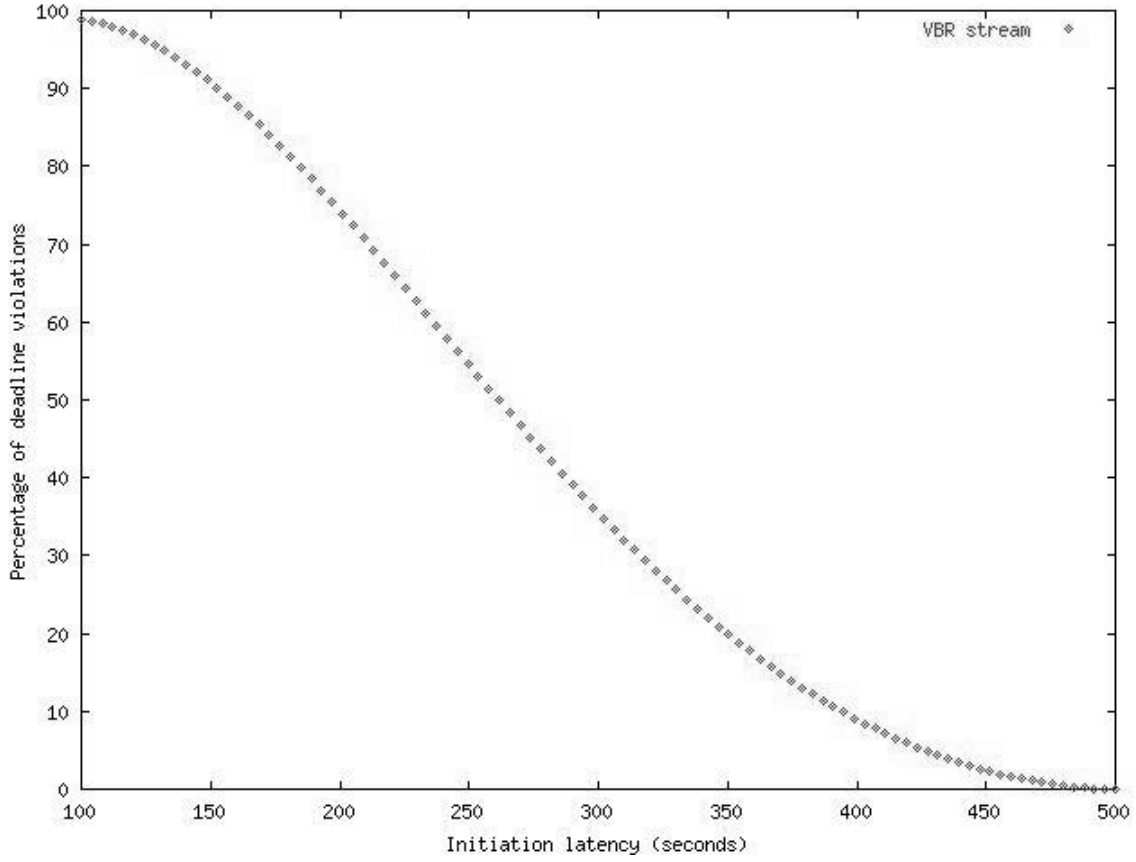


Graph 6.2. Initiation latency required for 0% deadline violations vs. Number of clients (Bandwidth of bottleneck link set to 1000 Mb/s)

Graph 6.2 shows how the initiation latency varies with the number of clients in the network. We use the same network topology shown in Figure 6.1 but vary the number

of end nodes on both right and left sides. The bottleneck link bandwidth is set to 1000 Mb/s. The network contains one client streaming CBR video (node n0). The rest of the clients send different kinds of traffic at the rate of 800 Mb/s. We see that initiation latency increases almost exponentially with the addition of more nodes when the bottleneck link is congested. If the video server is the only node present in the system then the initiation latency required is only 8 seconds. However, as the number of nodes is increased, the initiation latency increases almost exponentially. This shows that under congestion, TCP is not very scalable for video applications. Besides, this is bad news for applications that run over TCP and assume a fixed initiation latency, because when network traffic conditions change the fixed initiation latency may not mask the delays introduced by the network.

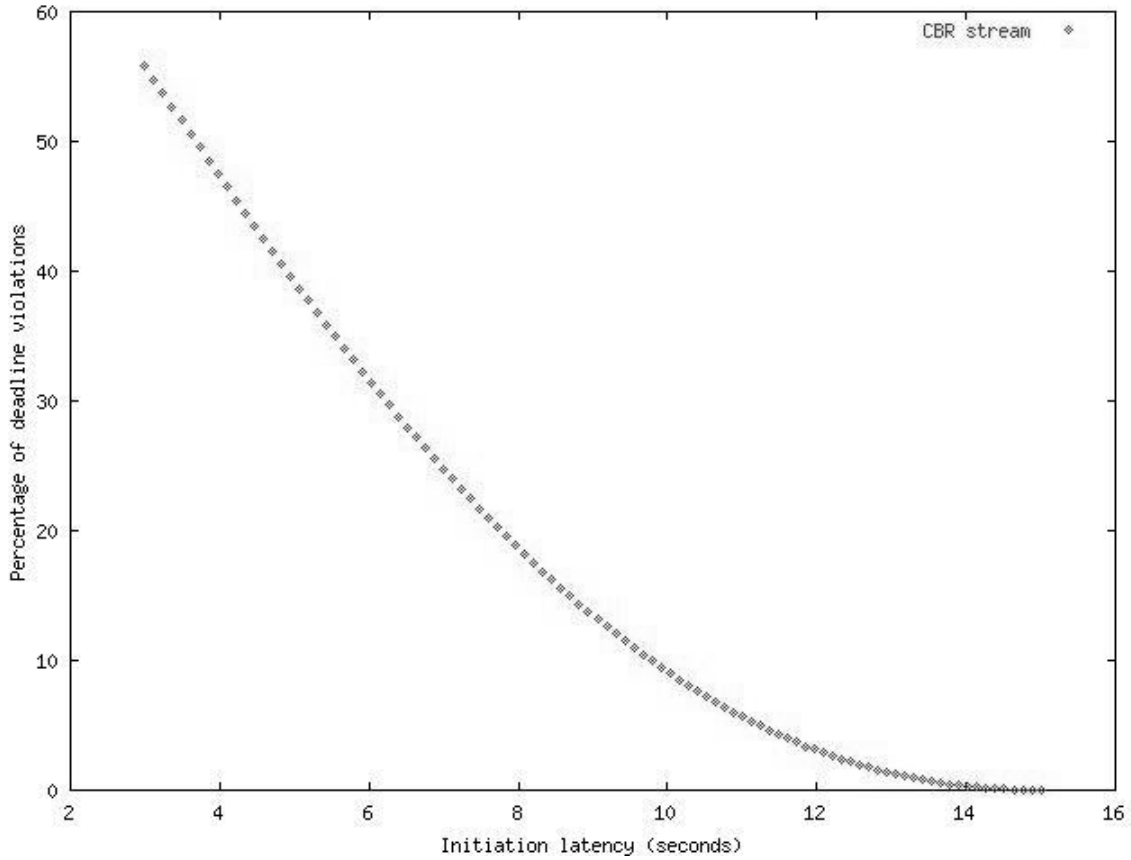
6.2.2. Performance Metric: Percentage of Deadline Violations Observed



Graph 6.3. Percentage of deadlines violated vs. Initiation latency (VBR)

Graph 6.3 shows how the percentage of deadlines violated varies with the initiation latency of the video playback. Network topology is the same as shown in Figure 6.1. The bandwidth of the bottleneck link is set to 1000 Mb/s. We see that the deadline violations start becoming unacceptable with a slight decrease in the optimal initiation latency. Around 10% of deadlines are violated (which probably will result in very poor quality video) when the initiation latency is 400 seconds. This graph shows that TCP is not at all suitable for live video transport. Also, as initiation latency is highly dependent

on network configuration (as shown in Graph 6.2), applications cannot easily estimate this latency. Therefore, even in the case of stored video, TCP performance is unacceptable.



Graph 6.4. Percentage of deadlines violated vs. Initiation latency (VBR)

Graph 6.4 proves the same point in case of CBR video using the same experimental setup. The only difference is that percentage of deadline violations are not that dramatic.

7. How Can We Make TCP Perform Better?

In this section we discuss how the performance of TCP for live video transport can be improved using two different approaches. One approach is augmenting end-to-end applications so that they mask the inadequacies of TCP. Another approach is to modify the current implementation of TCP to address its limitations for video transport. We discuss the implications of both approaches.

We note that, although they alleviate some of the problems, none of the approaches will make TCP suitable for video transport (especially live video transport).

7.1. Augmenting End-to-end Applications

One way to mask the problems associated with video transport over TCP is augmenting the end-to-end applications. This approach is taken by Real Networks' Real Player [2]. (However, Real Player uses UDP.)

7.1.1. Buffering

A simple approach to augmenting end-to-end applications is to add buffering to mask the variable end-to-end delay and congestion through the network. As we can see from the simulation results presented in Section 6, if we buffer enough of the sent packets on the receiver side, the receiver will be able to mask the delays in the network. However, there are several problems with this approach:

1. Buffer space required will probably be quite large. This is due to the nature of the data we are dealing with. Video files, even when they are compressed, are quite large and can take up a lot of space in memory. Another reason for buffer space being large is the fact that we would like to mask the delays in the network. Therefore, we need to be conservative about what we think the maximum end-to-end delay will be. Using this conservative estimate, we would like the application to refrain from initiating the video playback until enough frames to mask the cumulative end-to-end latencies have arrived on the receiver side. The conservative estimate on the end-to-end delay blows up the memory requirements of the buffer.

2. As a result of high buffer space requirements, the data that is buffered may not fit in the physical memory available on the receiver side. This has serious implications for decoding or playing the video stream. The data that is buffered may be swapped out onto the hard disk of the receiver. When the receiver application tries to start the playback, it may need to do Disk I/O accesses which will increase the initiation latency. Besides, periodic disk I/O will be needed to swap the required video stream into the physical memory from the disk. This may result in increased decode and playback latency and may degrade the visual quality of the video.

3. This approach increases the initiation latency of video playback on the client side. Such an increase is not good from a user viewpoint. There is another very important problem with increasing the initiation latency. High initiation latencies are not acceptable for live video transport. If the initiation latency is too high, the delay between the sender and receiver will be too high, which will prevent the video from being considered "live". For this reason, we need low initiation latencies for live video.

4. Determination of the buffering period is not very easy. This interval is dependent on the network conditions and the bitrate of the video. The receiver somehow needs to be aware of the network conditions to determine the buffering interval. This information can be sent to the receiver (approximately of course) using packets. However, it may not be always accurate because server does not have the capacity to determine this very accurately. Besides, during transmission, network conditions are very likely to change and the initial buffering may not be enough to mask the delays caused by congestion in the network. Therefore, conservative bounds on the buffering interval should be used, which is undesirable. In the extreme case, the whole video stream can be buffered before starting playback.

7.1.2. Bandwidth Smoothing

One other end-to-end application based approach to alleviate the problems with TCP is to use bandwidth smoothing on the server side to reduce the variability due to VBR nature of video. As we have shown in Section 6, TCP works a little better for CBR video, although not perfectly. Applying bandwidth smoothing can be done in several ways.

One way, which is the easier way, is to estimate the bandwidth available in the network. Based on that information, the server can convert the VBR stream into CBR by buffering. After this conversion the server can start streaming the CBR video stream. Most important problem with this approach is the buffer space requirements of such a task. Converting a whole video file requires a big amount of buffer memory on the server side. Another problem is the increase in initiation latency. Now, before the video is transmitted, it needs to incur additional conversion delay on the server side. Again, as discussed above this is unacceptable for live video and disturbing for stored video.

We can optimize the above approach by noting the following:

1. The more smoothing done on the sender side, the higher the buffering requirements on the sender side.

2. The more smoothing done on the sender side, the higher the initiation latency for video playback on the receiver side.

We can reduce both the buffering requirements and the initiation latency by reducing the smoothing interval. Hence, we can smooth a VBR file based on intervals. This approach is called approximating the VBR file in a piecewise CBR fashion. A piecewise CBR schedule can be determined by the server based on the bandwidth availability in the network. Ideally, we would like the network to provide information to the server about the bandwidth availability. However, this is not the case in today's networks. Therefore, the server needs to estimate the bandwidth availability in the network in order to perform bandwidth smoothing. This estimation can be done using a mechanism that is very similar to TCP congestion control mechanism. Or different mechanisms that estimate bandwidth such as congestion avoidance algorithms can be employed.

Bandwidth smoothing, as noted earlier, does not remove the problems with TCP. It somewhat alleviates the problems but it is only useful when combined with buffering at the receiver side. In case of live video it is mostly useless due to increased initiation latency. For example, if we transmit all the frames in a group of pictures of MPEG video at the same rate, this will cause increased delay, which is unacceptable for live video.

7.1.3. Why Don't Applications Do This?

Many streaming video applications such as Real Player work on UDP. These applications employ buffering but do not employ this over TCP. The main reasons for not using TCP are the following:

1. TCP's reliable transmission is too reliable. Video streams are usually tolerant to infrequent packet losses. Such strict reliable transmission requirements as employed by TCP increase the end-to-end delays for a packet very significantly.

2. TCP's congestion control mechanism is unstable especially when the network is congested. Also, it can be very negatively affected by other flows that do not use

congestion control. Such instability introduces unpredictable and variable delay for each video frame.

UDP lacks both of these disadvantages. UDP uses best-effort delivery. It does not guarantee reliable and ordered transmission. This is acceptable for video applications. Ordering of packets and hence frames can be done at the application level. And packet losses can be compensated for at the application level. The applications that are using UDP also are not bound by the congestion control mechanism of TCP. They can implement congestion control mechanisms of their own. However, care must be taken when doing this so that applications do not consume network bandwidth unfairly.

Having said this, we note that UDP is still not suitable for video transport. The main reason is, it does not guarantee an end-to-end delay bound or stable bandwidth allocation in the network. Without such a guarantee, video transport is almost impossible. Therefore, we argue that some quality of service guarantees must be made by the network and the transport protocol for the satisfactory transmission of video. This is even more true in case of live video.

7.2. Changing the Implementation of TCP

Another way of approaching the problem is by changing the implementation of TCP. It might be argued that changing the implementation of TCP creates another protocol. In any case, the point of changing the implementation of TCP is to be able to support video transport without changing the underlying network and implementation of protocols by too much.

We have pointed out the most important shortcomings of TCP in terms of live video transport in sections 4 and 7.1.3. Now we will suggest some solutions which may alleviate these shortcomings. We reiterate the main problems with TCP that lead to its poor performance for video transport:

1. TCP is too reliable, does not distinguish between important and non-important packets.
2. Related to this, TCP does not have a notion of “deadline” which is critical for video transport.
3. Sliding window algorithms of TCP make the available bandwidth fluctuate and are tightly coupled with strict reliable delivery.

Our aim in the design of a modified TCP is to attack these problems and try to make the modified TCP a useful protocol for video transport. Our design principles for the modified TCP algorithm considers the following issues:

1. Congestion Control: The current structure of the internet depends on end-to-end congestion control mechanisms. There is no quality of service or reliability support from the network itself. The network does not provide any guarantees to the data flowing on the network and hosts that are generating the traffic. To keep such a network stable it is imperative to employ end-to-end congestion control mechanisms. Therefore, our modified TCP protocol also implements congestion control. This mechanism is window-based and ACK-clocked as described in section 4.5. The congestion control protocol of TCP is robust and adapts fairly quickly to the available bandwidth. Using the same

congestion control scheme as TCP does, our modified TCP algorithm does not become greedy and competes fairly with TCP flows in the network. Besides, it does not threaten the network by overloading it.

2. Fairness: Current structure of the internet does not prevent a flow from sending packets more aggressively than other flows and hence obtain more than its fair share of the bandwidth. In fact, without any quality of service guarantees, a fair network should guarantee equal share of the network bandwidth for each of the flows. When designing a new transport protocol on the internet, we should take into account whether or not the protocol is fair to other protocols. This fairness can be somewhat provided by employing some kind of rate-based scheduling algorithm in the end nodes (We are not concerned with the infrastructure of the network here, just the end nodes. If we change the whole infrastructure we might as well incorporate the quality of service notion of the integrated services networks). These rate-based scheduling algorithm can ensure that the application does not send data into the network at a rate that is greater than what it has specified. Such a scheme would provide some fairness among flows competing in the network. However, the realization of such a scheme requires the application to request some amount of bandwidth guarantee from the transport layer. Unfortunately, this scheme is not currently employed on the internet. It also requires significant changes to the structure and requires an end-to-end admission control algorithm which is beyond the complexity we are aiming for in this report.

Therefore we resort to TCP's congestion control algorithms to provide some amount of fairness in the network.

3. Deadline-Driven Transmission: Our modified algorithm takes into account the fact that video transport inherently requires deadline-driven transmission. Each frame has a deadline by which it needs to be displayed on the receiver side to ensure acceptable visual quality of the video. This means that some frames whose deadlines are missed can be discarded and do not need to be retransmitted. Also, it means that we need to incorporate some notion of deadlines into the modified TCP protocol. These observations come from two features of video streams:

1. They are time-sensitive.
2. They are fairly error-resilient.

7.2.1. Reliability of the Modified Algorithm

With these two requirements in mind, we design a modified TCP algorithm which excludes the strict reliability requirement of TCP, which is not required for video transport.

In the modified TCP, each section of time-sensitive data is associated with a deadline. The sender sends a section of data and performs retransmissions on it until the deadline of the data. If the deadline of the data has passed, then no retransmissions are performed. The data is simply discarded.

To keep track of the deadlines associated with data sections, we add a timer to the sender side. When the unsent data on the sender side becomes useless due to deadline expiry, the modified TCP replaces the data with new data whose deadline has not expired yet.

Deadlines of the data sections are determined relative to the sender. It is also possible to define the deadlines based on the receiver but then the receiver needs to use the estimates of round trip time to determine whether the deadline of a section of data is expired.

7.2.2. Implementation of the Modified TCP Algorithm

The sender side application associates each packet of a frame with a deadline. This deadline is determined by the application and can be determined based on the frame rate of the video and an estimate of end-to-end round trip time. The sender side of the transport protocol maintains a list which holds the deadlines for each data section (frame). The sender behaves just like the original TCP until the expiry of a timer. Once the timer associated with the current oldest data section expires, the sender discards the packets associated with the current data section. It sets the timer to the deadline for the next section of packets (if this deadline also has not expired yet) and starts transmitting the new data section. Hence, the sender side effectively advances the sliding window

The timer that is used to determine whether a deadline is violated is very similar to the timer used by the original TCP algorithm to perform adaptive retransmission based on the timeout values. We need only one timer because the timer only keeps track of the deadline associated with the oldest data section that is currently in sender's window.

We modify the receiver side of the TCP protocol such that it also advances the sliding window if the sender discards packets because their deadlines are already violated. To achieve this, the sender needs to inform the receiver that a section of data is discarded due to deadline violations. If we do not provide a mechanism to achieve this, the system will deadlock because reliable transmission of TCP requires that the receiver wait for the next sequential packet. The sliding window algorithm uses sequence numbers for packets to synchronize between the sender and the receiver. the receiver always keep track of an expected sequence number. This denotes the packet that will be submitted to the application next. Such a sequence numbering mechanism guarantees the reliable and ordered delivery of data.

In our algorithm the problem arises when the sender discards packets without sending them and the receiver still expects those packets to be sent. The solution is to prevent the receiver from expecting the packets that are already discarded by the sender. To solve this problem, the sender explicitly notifies the receiver of the next expected sequence number. This is done by utilizing the TCP options in the TCP packet format. The receiver updates its next expected sequence number to the value specified by the special option included in every TCP packet only if the special option has a value greater than the current next expected sequence number. Under normal conditions when the sender does not discard any packets, the value of this option is set to the previous next expected sequence number. When a packet is discarded, this value is set to the sequence number of the oldest packet whose deadline has not been violated.

7.2.3. Details of the Algorithm

When the sender discards packets that belong to a frame due to a deadline violation, some of these packets may still be in transit. The sender determines this and accepts the acknowledgements of these packets. This is not a very important part of the algorithm but it needs to be correctly handled by the sender.

7.2.4. An Example

The following diagram shows an example of how our modified algorithm works. As a comparison, we provide an example of how original TCP works.

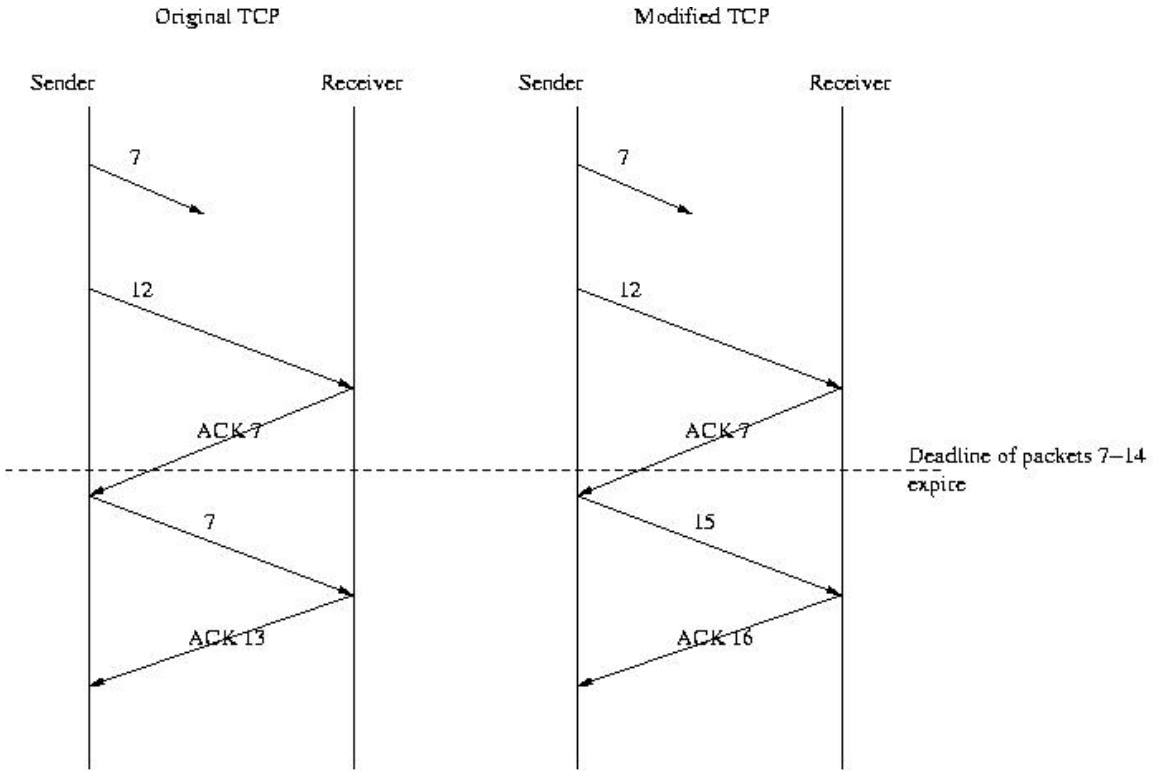


Figure 7.1. Illustration of the difference between the original TCP and our modified version of TCP.

The example above shows a sequence of packets starting with sequence number 7. We assume that packets 8-11 are transmitted successfully by the sender and ACKs for those packets are successfully received. Packet 7 is dropped by the network. We also assume that packets 7-14 belong to the same frame and therefore have the same deadline. This deadline expires as denoted by the broken line. In the TCP diagram, this broken line has no importance, because TCP has no notion of deadlines. Therefore when TCP sender receives the duplicate ACK for 7 once again, it sends packet 7 again.

On the other hand, with our modified algorithm, the sender detects that the deadline for 7-14 has expired when it receives the duplicate ACK for 7. It discards these packets from its buffer and sends packet 15 instead. Packet 15 contains 16 as the next expected number in its options field. When the receiver receives packet 15, it sees that the next expected sequence number provided by 15 is larger than what it is expecting (7) and readjusts. It sends an ACK for 16 which it now is waiting.

8. Evaluation of The Modified TCP Algorithm

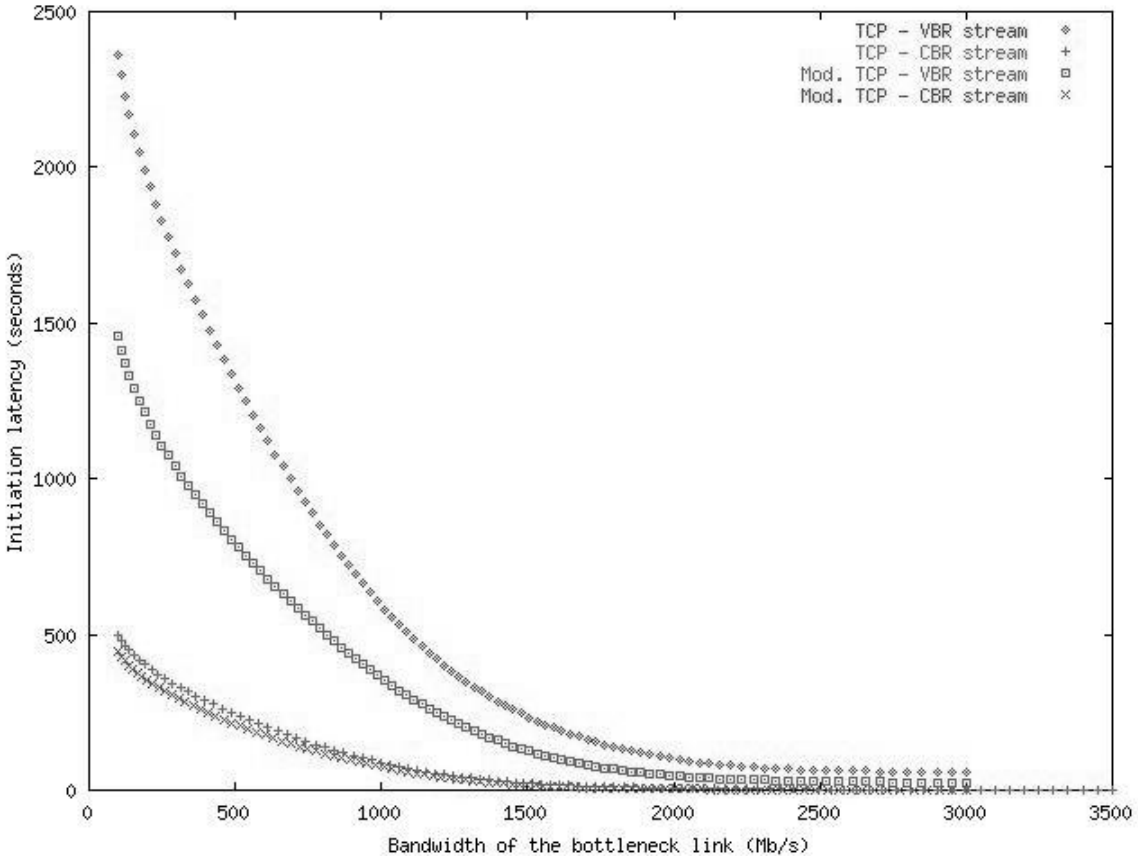
In some sense, the proposed TCP modification doesn't let the TCP to slow down and lag behind the receiver when deadlines are violated. The modified algorithm does not guarantee that deadlines will not be violated. This task is impossible using the current underlying network implementation. However, it stops TCP from missing more and more deadlines when the network gets congested.

We evaluate our modified protocol using the same metrics we used to evaluate TCP performance in section 6. We compare the performance of this protocol to TCP performance and show that a deadline-driven and intelligently-unreliable TCP-based protocol is much better for video than plain TCP. All of the experiments shown in this section are performed on the network topology described in section 6.

8.1. Performance Metric: Initiation Latency

Graph 8.1 shows the initiation latency required to achieve 0% deadline violations for all possible combinations of (CBR, VBR) x (TCP, modified TCP). It is clear that our modified TCP algorithm performs much better than the original TCP algorithm for VBR flows. Initiation latency is decreased quite a bit for the VBR video. For example, at 1000 Mb/s bottleneck link bandwidth, the initiation latency for VBR video over TCP is 453 seconds, whereas the initiation latency for VBR video over modified TCP is 277 seconds. At 2500 Mb/s bottleneck link bandwidth, the initiation latency for VBR video over TCP is 59 seconds, whereas the initiation latency for VBR video over modified TCP is 26 seconds. Our modified TCP protocol effectively decreases the required initiation latency.

For the CBR video, our scheme also decreases the initiation latency. However, the effects are not as dramatic as they are for the VBR flow. We conclude that our algorithm overcomes some of the shortcomings of TCP very well. However, it is not as suitable for VBR video as changing the underlying network structure so that it provides bandwidth and deadline guarantees to each flow. No such guarantees exist in our algorithm.

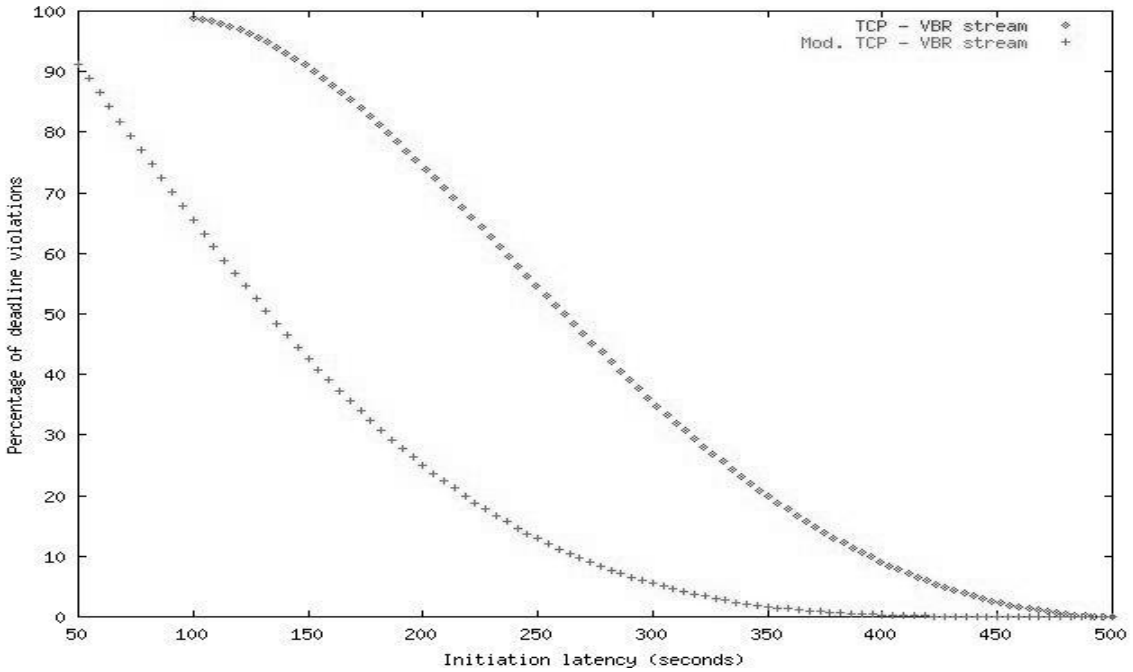


Graph 8.1. Initiation latency required for 0% deadline violations vs. Bandwidth of the bottleneck link for the original and modified TCP protocols.

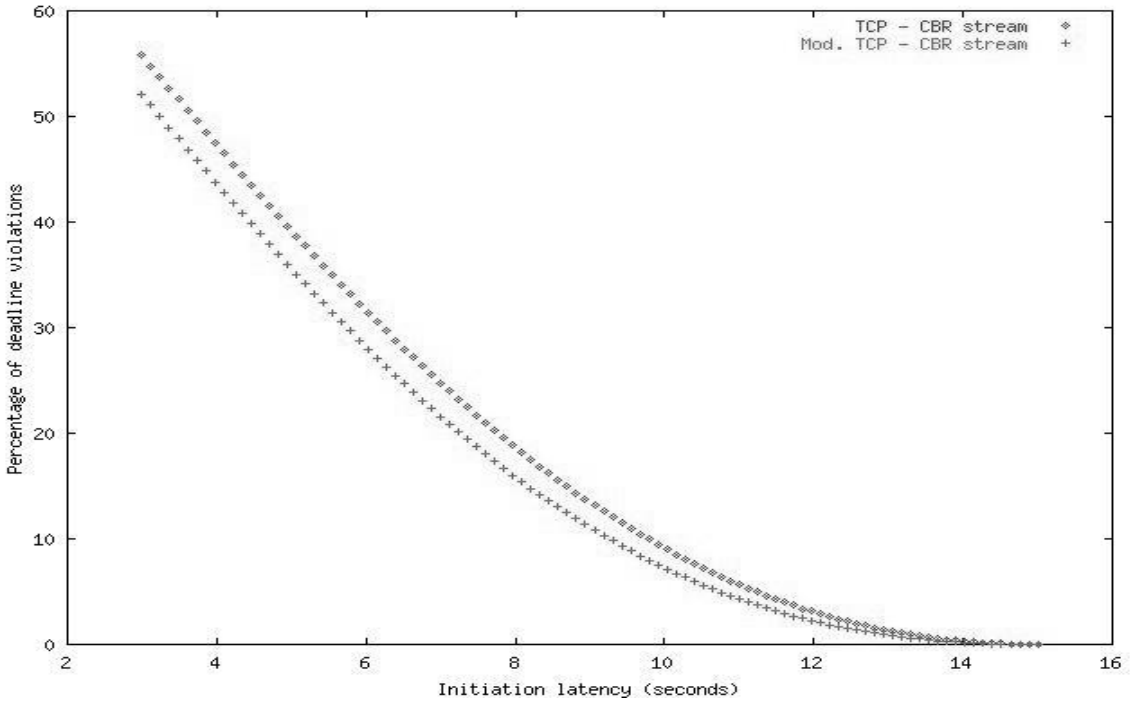
8.2. Performance Metric: Percentage of Deadlines Violated

Figure 8.2 shows how the percentage of deadlines violated is reduced by our modified TCP protocol for the VBR stream. The experimental conditions are the same as outlined in section 6.2.2. We see that no packet losses occur until around 370 seconds with our modified TCP protocol. On the other hand the initiation latency has to be at least around 500 seconds with TCP. The modified TCP algorithm still cannot support low initiation latencies. Therefore, for live video it will still perform very badly. However, it is an improvement over TCP and with better end-to-end application support and enhancements it may perform even better.

Figure 8.3 shows the results of the same experiments using CBR flows. The positive effects of our modified TCP algorithm are not as dramatic for CBR video, but it still improves the performance of CBR video.



Graph 8.2. Percentage of deadlines violated vs. Initiation latency for the original and modified TCP protocols (VBR).



Graph 8.3. Percentage of deadlines violated vs. Initiation latency for the original and modified TCP protocols (CBR).

9. Conclusion

In this project we evaluated the effectiveness of video transport using the TCP protocol. Our experimental results confirm our hypothesis that TCP is not suitable for video traffic over the internet. This is especially true for VBR video which injects bursty traffic into the network. We conclude that the main reason for TCP's ineffectiveness is due to its blindness to the requirements of video data. TCP enforces a strict reliable transmission scheme and does not at all exploit the loss-resilience of streaming video data. Based on these conclusions we proposed a modified TCP protocol which exploits the characteristics of video and implements a deadline-driven and intelligently unreliable protocol. our experimental results show that the modified TCP algorithm performs much better than the original TCP, especially for VBR flows. However, this performance is still not enough for live video and more importantly is very much dependent on the network topology, traffic, and characteristics. This is mainly due to the fact that the underlying network does not support any delay or bandwidth guarantees to real-time data like video. Our main conclusion is that real-time video data (and in particular live video) cannot be supported fully without significant changes to the underlying network infrastructure which will incorporate guaranteed quality of service to real-time applications.

References

[1] L.L. Peterson and B.S. Davie. Computer Networks: A Systems Approach, Morgan Kauffman, 2000.

[2] Real Networks. Real networks press release, December 6, 1999.