

# Video Delivery over TCP

Lecture 11  
Subir Varma

# Video Traffic

- ▶ Video traffic comes in two flavors, video on demand (VoD) and live-video streaming. VoD traffic is from stored media and is streamed from servers, and it constitutes the majority of the video traffic.
- ▶ There are real-time constraints in the delivery of video traffic to the client player. This constraint arises because the client video device expects data to be constantly available so that it can keep updating the screen at a constant rate (which is usually 30 frames/s). If there is a hiccup in this process and no data is available, then this results in a temporarily frozen screen while the network catches up.
- ▶ Most of the early work on packet video transmission focused on providing real-time transmission by means of new techniques that supported resource reservations and QoS (quality of service) provisioning in the network. Most operators balked at supporting these protocols for consumer video transmission because of the extra complexity and cost involved at both the servers and in the network infrastructure.
- ▶ During the early years of the Web, the conventional wisdom was that video streaming would have to be done over the User Datagram Protocol (UDP) because video did not require the absolute reliability that TCP provided, and furthermore, TCP retransmissions are not compatible with real time delivery that video requires.
- ▶ TFRC: TCP Friendly Rate Control – A way to add congestion control to UDP

# Benefits of Using TCP for Video

- ▶ TCP's rate fluctuations, which were thought to be bad for video, could be overcome by using a large receive buffer to dampen them out.
- ▶ Since most video transmissions were happening over the Web, using the HyperText Transfer Protocol (HTTP) for video was also very convenient. The combination of HTTP/TCP for video delivery had several benefits, including:
  - TCP and HTTP are ubiquitous, and most video is accessed over the Web.
  - A video server built on top of TCP/HTTP uses commodity HTTP servers and requires no special (and expensive) hardware or software pieces.
  - HTTP has built-in Network Address Translation (NAT) traversal capabilities, which provide more ubiquitous reach.
  - The use of HTTP means that caches can be used to improve performance. A client can keep playback state and download video segments independently from multiple servers while the servers remain stateless.
  - The use of TCP congestion control guarantees that the network will remain stable in the presence of high bit rate video streams.

# The DASH Protocol: Dynamic Adaptive Streaming over HTTP

- ▶ DASH enables the video receiver is able to adaptively change the video rate so that it matches the bandwidth that the network can currently support.
- ▶ DASH can be considered to be a flow control rather than a congestion control algorithm because its objective is to keep the video receive buffer from getting depleted rather than to keep network queues from getting congested.
- ▶ DASH operates on top of TCP congestion control, albeit over longer time scales, and the interaction between the two is rich source of research problems.
- ▶ DASH enables each Video Service Provider to implement their own rate adaptation algorithm at the client, while maintaining inter-operability with servers belonging to other Service Providers.

# MPEG Video Compression

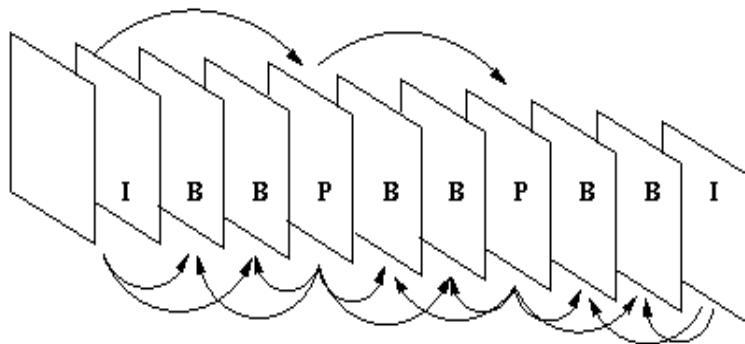
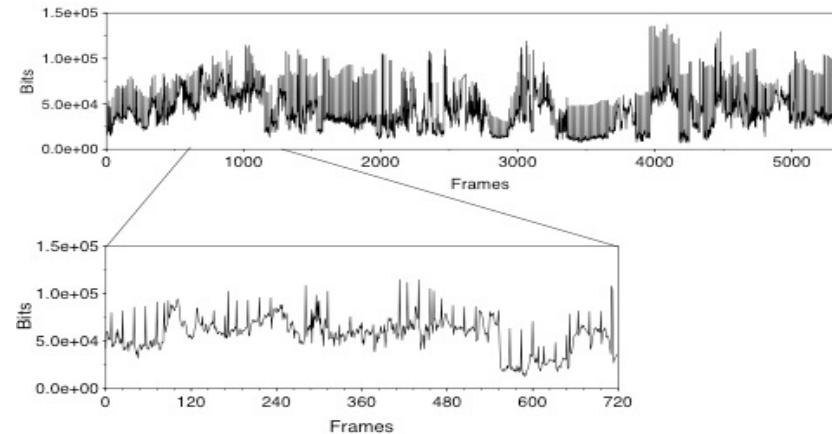


Figure 1: Prediction between MPEG-2 Frames

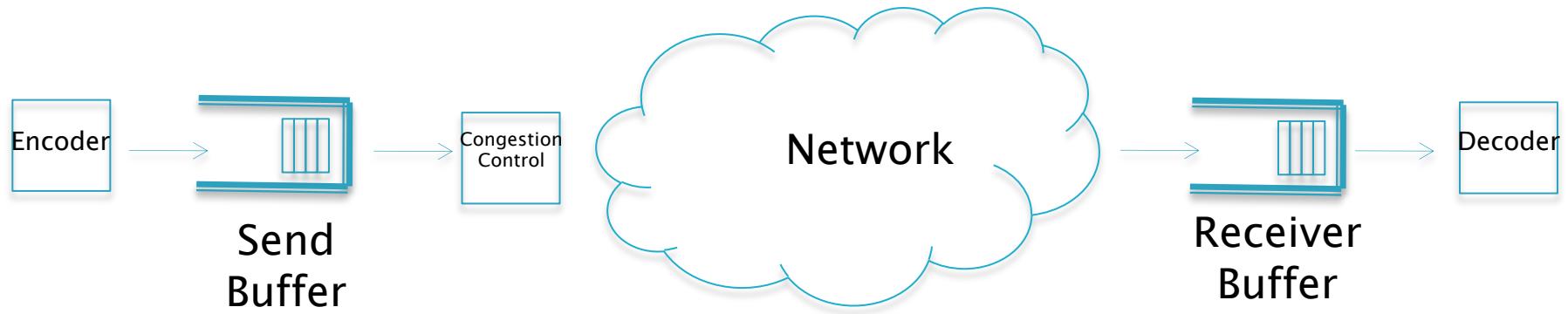


- Video compression is done by using the Discrete Cosine Transform (DCT) on the quantized grey scale and color components of a picture frame, and then transmitting the truncated DCT coefficients instead of the original picture.
- In addition to the intraframe compression, all compression algorithms also carry out interframe compression, which takes advantage of temporal picture redundancy in coding a frame by taking its delta with respect to a previous frame.
- There are three types of frames shown: I frames are largest because they only use intraframe compression; B and P frames are smaller because they use previous I frames to further reduce their size. This results in a situation in which the encoded bits per frame is a variable quantity, thus leading to Variable Bit Rate (VBR) video.
- As a result of compression, it is possible to send an HD-TV 1080p video using a bit rate of just 2 mbps.

# Video Delivery over Packet Networks

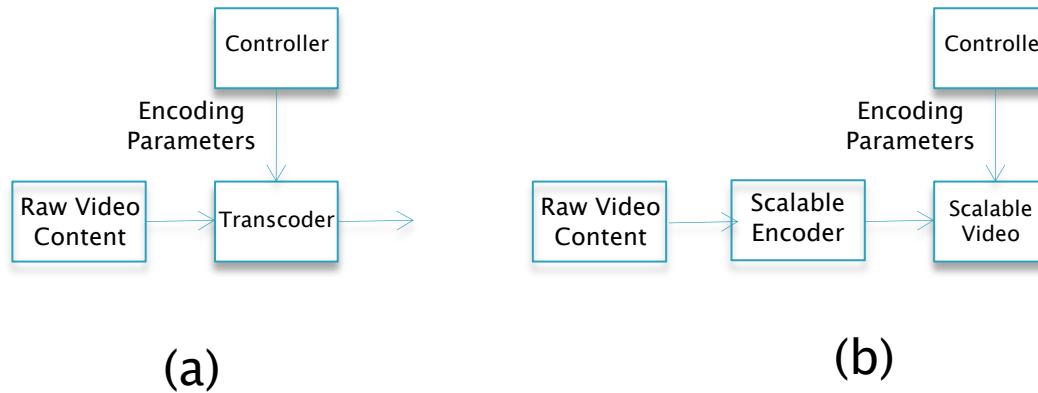
- ▶ Because we are not depending on the network to provide a guaranteed bandwidth for the video stream, there arises the problem of **matching the video bit rate with the bandwidth that the network can currently provide on a best-effort basis.**
- ▶ If the network bandwidth is not sufficient to support the video bit rate, then the decoder at the receiving end starts to consume the video data at rate that is greater than the rate at which new data is being received from the network.
- ▶ As a result, the decoder ultimately runs out of video data to decode, which results in a screen freeze and the familiar “buffer loading” message that we often see.
- ▶ How can we avoid this without doing guaranteed bandwidth management?

# End-to-End Block Diagram of a Video Transmission System



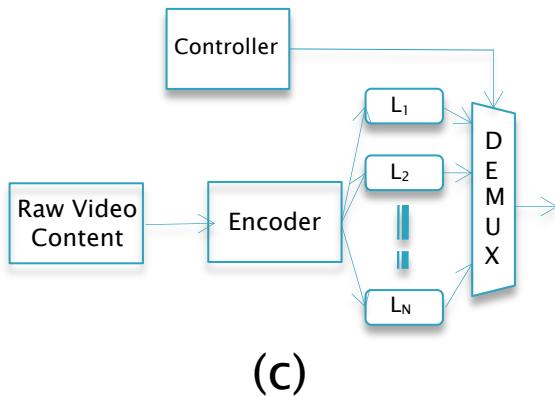
- Use of a large receive buffer: As shown in Figure 6.2, the system can smooth out the variations in network throughput by keeping a large receive buffer. As a result, temporary reductions in throughput can be overcome by used the video stored in the receive buffer.

# Adaptive Streaming Techniques



- a) **Transcoding-based solutions** (Figure 6.3A): These algorithms change one or more parameters of the compression algorithm that operates on the raw video data to vary the resulting bit rate. Examples include varying the video resolution, compression ratio, or frame rate. Transcoding is very CPU intensive and requires hardware support to be done at scale, which makes them difficult to deploy in Content Delivery Networks (CDN).
- b) **Scalable encoding solutions** (Figure 6.3B): These can be implemented by processing the encoded video data rather than the raw data. Hence, the raw video can be encoded once and then adapted on the fly by using the scalability features of the encoder. Examples of scalable encoding solutions include adapting the picture resolution or frame rate by exploiting the spatial or temporal scalability in the data. However, even scalable encoding is difficult to implement in CDNs because specialized servers are needed for this.

# Adaptive Streaming Techniques

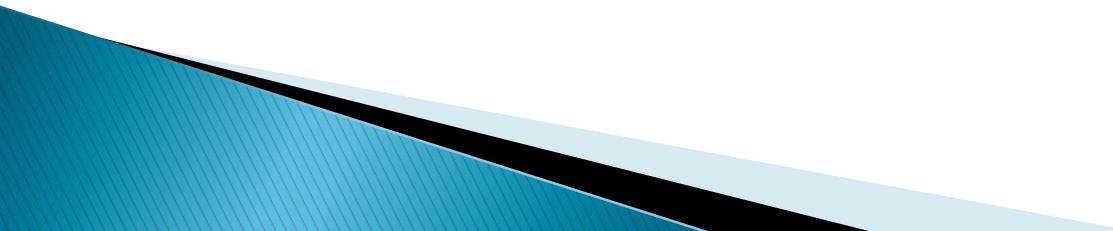


- c) Stream switching solutions (Figure 6.3C): This technique is the simplest to implement and can also be used by CDNs. It consists of preprocessing the raw video data to produce multiple encoded streams, each at a different bit rate, resulting in N versions. An algorithm is used at the time of transmission to choose the most appropriate rate given the network conditions. Stream switching algorithms use the least processing power because after the video is encoded, no further operations are needed. The disadvantages of this approach include the fact that more storage is needed and the coarser granularity of the encoded bit rates.

# Video Rate Control

- ▶ The industry has settled on using a large receive buffer and stream switching as the preferred solution for video transmission.
- ▶ Before the coding rate at the source can be changed, the video server has to be informed about the appropriate rate to use. Clearly, this is not a function that a congestion control protocol such as TCP provides; hence, all video transmissions systems use a rate control protocol operating on top of TCP.  
**But who should do the rate control, the server or the client?**
- ▶ Early video streaming protocols such as RAP and TFRC used server side rate control based on feedback being received from either the network or the receiver; hence, they were doing a combination of congestion control and flow control.

# DASH Protocol for Adaptive Video Tx

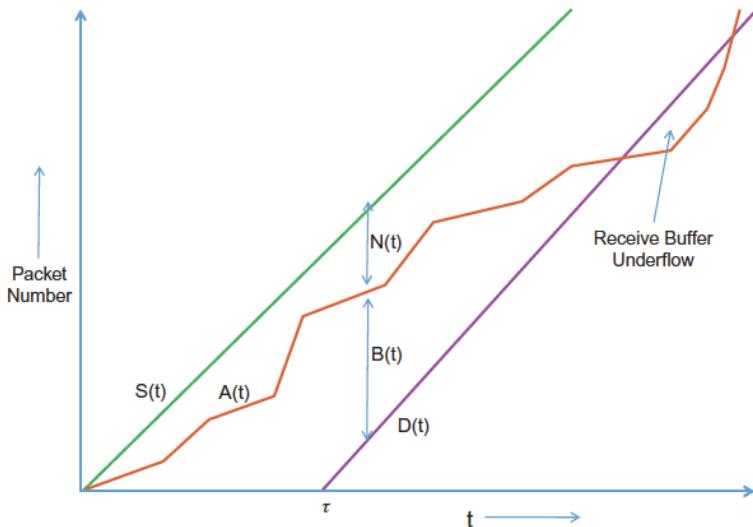


# The DASH Protocol

The DASH protocol, which dominates video transport today, uses a scheme that differs from these early algorithms in the following ways:

- ▶ DASH is built on top of TCP transport, unlike the earlier schemes, which were based on UDP.
- ▶ Instead of the transmitter, the receiver in HAS drives the algorithm. It keeps track of the TCP rate of the video stream as well as the receive buffer occupancy level, and then using the HTTP protocol, it informs the transmitter about the appropriate video bit rate to use next.
- ▶ Instead of sending the video packets in a continuous stream, DASH breaks up the video into chunks of a few seconds each, each of which is requested by the receiver by means of an HTTP request.
- ▶ DASH adapts the sending rate, and consequently the video quality, by taking longer term averages of the TCP transmit rate and variations in the receive buffer size. This results in a slower variation in the sending rate, as opposed to TCP congestion control, which varies the sending rate rapidly in reaction to network congestion or packet drops.

# CBR Video Transmission



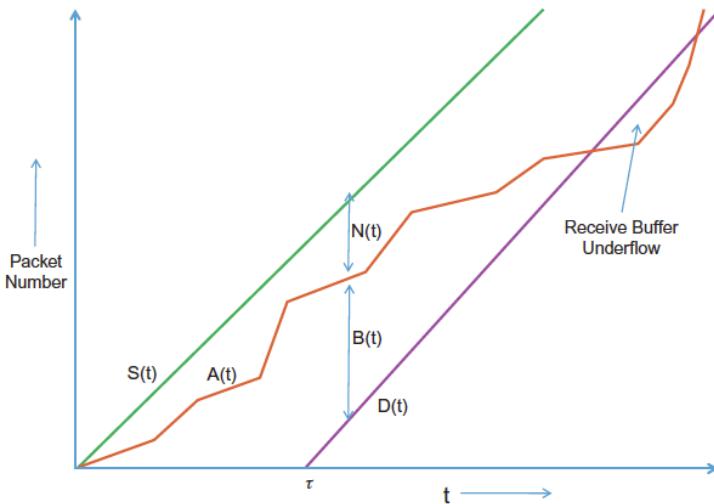
$A(t)$ : The number of bits that have arrived at the receive buffer  
 $N(t)$ : The number of bits in transit through the network  
 $B(t)$ : The number of bits waiting to be decoded in the receive buffer

$S(t) = Kt$  is the number of bits the source encoder has transmitted into the network by time  $t$   
 $D(t)$  is the number of bits the receiving decoder has pulled from the receive buffer by time  $t$

$$D(t) = \begin{cases} 0 & 0 \leq t \leq \tau \\ K(t - \tau) & t \geq \tau \end{cases}$$

$\tau$  is the delay before the decoder starts pulling data from the receive buffer

# CBR Video Transmission



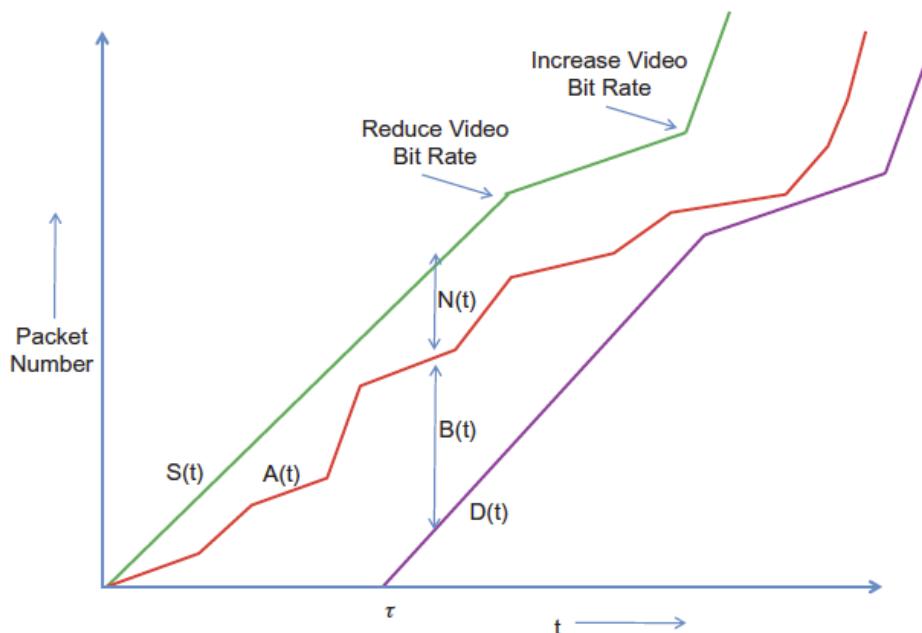
$$N(t) = S(t) - A(t) = Kt - A(t), \quad t \geq 0$$

$$B(t) = \begin{cases} A(t) & 0 \leq t \leq \tau \\ A(t) - D(t) = A(t) - K(t - \tau) & t \geq \tau \end{cases}$$

Note that  $A(t)$  is dependent on the congestion state of the network, hence

- If the network is congestion free, then  $A(t)$  is close to  $S(t)$ , so that the amount of traffic in the network  $N(t) \approx 0$  by [equation 2](#), and the amount of bits in the receiver is approximately given by  $B(t) \approx K\tau$  by [equation 3](#), so that the receive buffer is full.
- If the network is congested, then  $A(t)$  begins to approach  $D(t)$ , so that  $B(t) \approx 0$  and  $N(t) \approx K\tau$  by [equations 3 and 4](#). In this situation, the decoder may run out of bits to decode, resulting in a frame freeze.

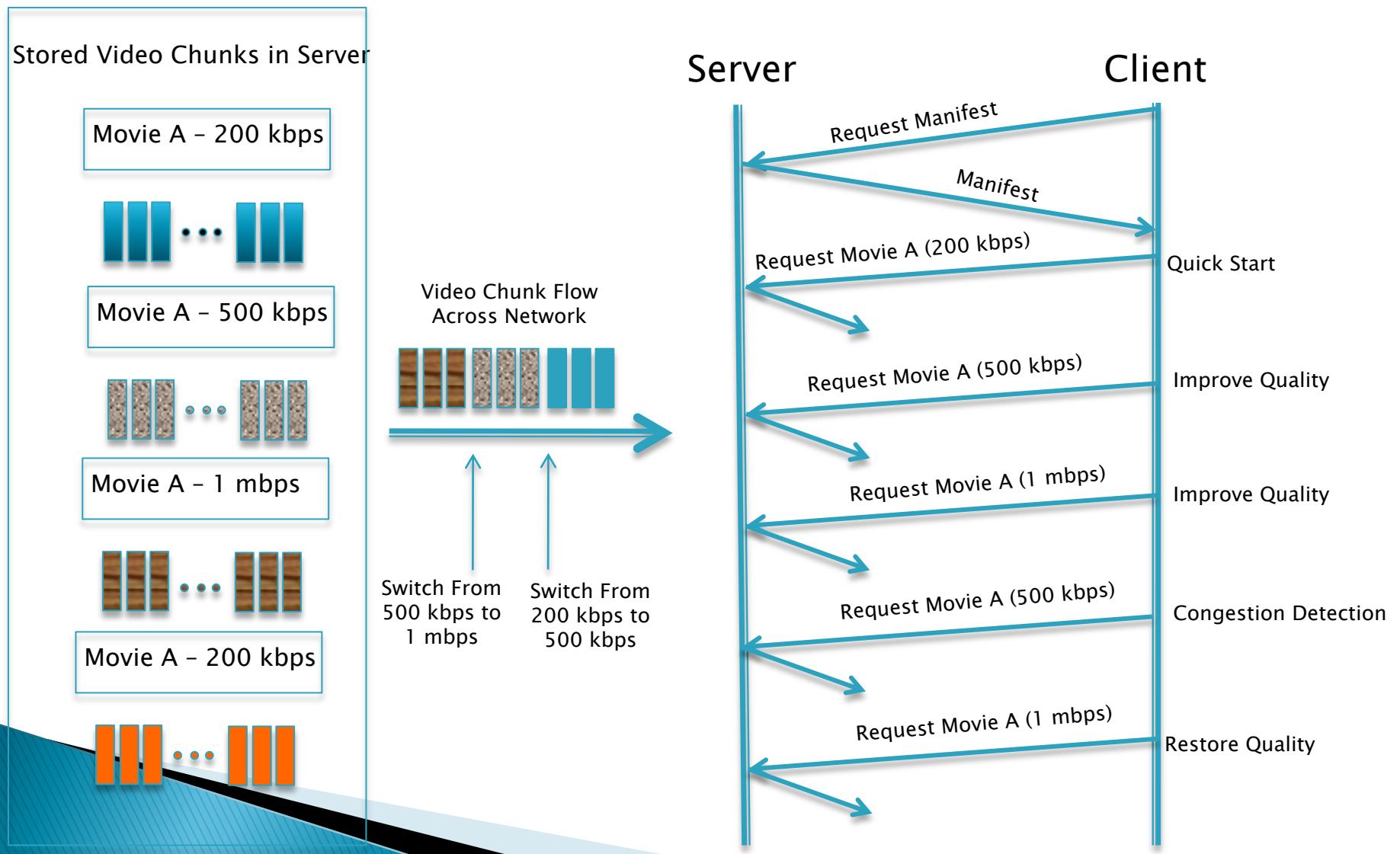
# VBR Video Transmission using DASH



The video is encoded at multiple rates, which can be adaptively changed depending on the network conditions.

- DASH adds another layer of rate control on top of TCP.
- DASH is video aware and is able to interact with the video application at the sender to adaptively change its sending rate.
- DASH decreases the video rate if the network congestion increases, and conversely increases it when the congestion reduces.

# Illustration of HTTP Requests and Bitrate Adaptation in DASH



# DASH Operation

- ▶ A video stream is divided into short segments of a few seconds each, referred to as chunks or fragments.
- ▶ Each chunk is encoded and stored in the server at number of versions, each with a different bit rate.
- ▶ At the start of the video session, a client downloads a manifest file that lists all the relevant information regarding the video streams.
- ▶ The client then proceeds to download the chunks sequentially using HTTP GETs.
- ▶ By observing **the rate at which data is arriving to the client and the occupancy of the video decoding buffer**, the client chooses the video bit rate of the next chunk.
- ▶ The precise algorithm for doing so is known as the ABR (Adaptive Bit Rate) algorithm

Chunk of  
duration  
 $\tau$  sec

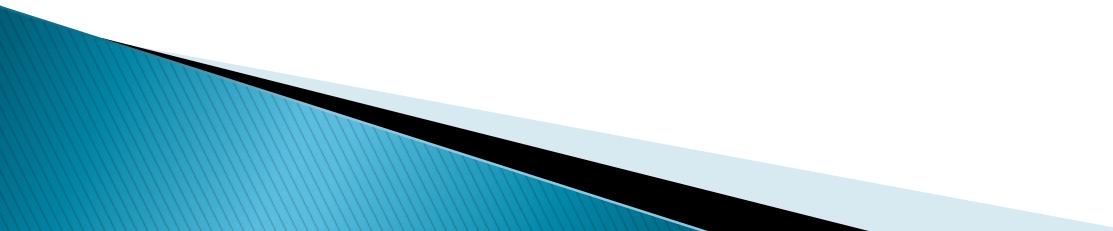
Size of the chunk = (Rate at which the chunk was coded \*  $\tau$ ) bits

Hence higher bit rate chunks take longer to transmit

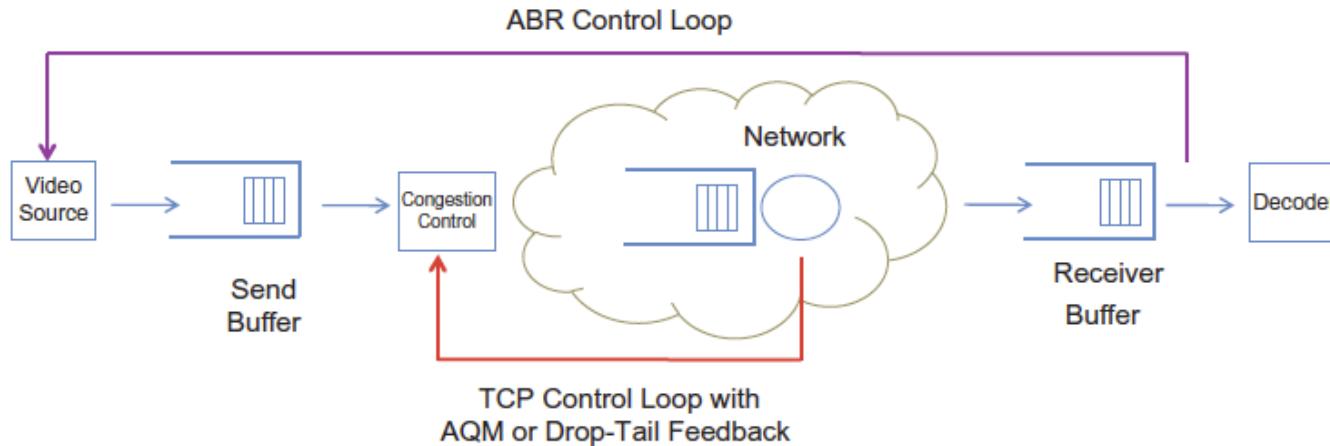
# Some Examples of DASH Systems

- Microsoft HTTP Smooth Streaming (HSS) [5]: Uses a chunk size of 2 sec and a 30-sec receive buffer at the receiver
- Apple HTTP Live Streaming (HLS)
- Adobe HTTP Dynamic Streaming (HDS) [6]: Uses a receive buffer of size less than 10 sec
- Netflix [7]: Uses a chunk size of 10 sec and a 300-sec receive buffer

# Adaptive Bit Rate (ABR) Algorithms



# The Adaptive Bit Rate (ABR) Algorithm



- The TCP inner control loop reacts to network congestion and **tries to match the TCP send rate with the rate that the network can support**,
- The ABR outer loop reacts to the rates that TCP decides to use and **tries to match the rate of the video stream to the average TCP rate**.
- The TCP control loop operates in order of a time period, which is approximately equal to the round trip delay (i.e., tens of milliseconds in most cases)
- the ABR control loop operates over a much larger time period, ranging from a few seconds to tens of seconds depending on the ABR algorithm
- This leads to a natural averaging effect because ABR does not attempt to match the fluctuating short-term TCP throughput but rather a smoothed throughput that is averaged over several seconds.

# The Adaptive Bit Rate (ABR) Algorithm

The ABR algorithm controls the following quantities:

- The rate at which the receiver sends HTTP requests back to the video source
- The appropriate video stream to transmit, whose rate most closely matches the rate that can be supported by the network

The objectives of an ideal ABR algorithm include the following:

1. Avoid interruptions of playback caused by buffer underruns.
2. Maximize the quality of the video being transmitted: Fulfilling this objective constitutes a trade-off with objective 1 because it is always possible to minimize the number of interruptions by always transmitting at the lowest rate.
3. Minimize the number of video quality shifts to improve user experience: This leads to a trade-off with objective 2 because the algorithm can maximize the video quality by reacting to the smallest changes in the network bandwidth, which, however, increases the number of quality shifts.
4. Minimize the time between the user making a request for a new video and the video actually starting to play: This objective can also be achieved at the cost of objective 2 by using the lowest bit rate at the start.

$$\tau = \frac{B(n)}{V(n)} = \frac{B(m)}{V(m)}$$

If  $V(n) > V(m)$  then  $B(n) > B(m)$

Hence a chunk encoded at a larger video bitrate takes longer to traverse the network

# ABR Algorithm

Define the following:

$V = \{V(1), \dots, V(L)\}$  Set of available bit rates for the video stream that are available at the transmitter, with  $0 < V(n) < V(m)$  for  $n < m$

$U_n$ : The video rate from the set  $V$  which is used for the  $n^{\text{th}}$  chunk, which is an output of the ABR algorithm

$\tau$ : Size of a chunk in seconds, in terms of actual time spent in playing the video contained in it

$R_n$ : Throughput for the  $n^{\text{th}}$  video chunk as measured at the receiver

$\hat{T}_n$ : HTPP inter-request interval, between the  $n^{\text{th}}$  and  $(n + 1)^{\text{rst}}$  requests. This is also an output of the ABR algorithm.

$\tilde{T}_n$ : Download duration for the  $n^{\text{th}}$  chunk

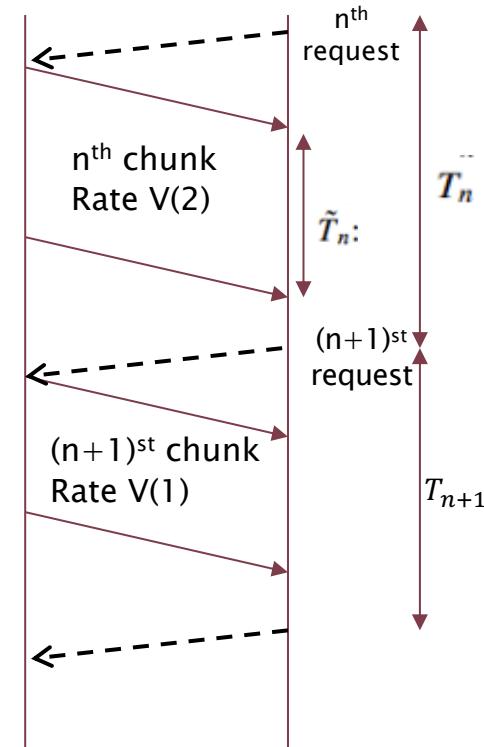
$T_n$ : Actual inter-request time between HTTP requests from the receiver, between the  $n^{\text{th}}$  and  $(n + 1)^{\text{rst}}$  request, so that

$$T_n = \max(\hat{T}_n, \tilde{T}_n) \quad (6)$$

This is also referred to as the  $n^{\text{th}}$  download period.

$B_n$ : Duration of the video stored in the video buffer at the end of the  $n^{\text{th}}$  period

Because the size of the receive buffer is in terms of the duration of video data stored in it, the download of a chunk causes the buffer size to increase by  $\tau$  seconds irrespective of the rates  $U_n$  or  $R_n$ . Similarly, it always takes  $\tau$  seconds to play the video data in a chunk, irrespective of the rate at which the video has been coded.



# A Generic ABR Algorithm

1. Estimation: The throughput during the  $n^{\text{th}}$  video chunk is estimated using the following equation:

$$R_n = \frac{U_n \tau}{\tilde{T}_n} \quad (7)$$

The buffer size (in seconds) at the end of the  $n^{\text{th}}$  download period is computed by

$$B_n = \max(0, B_{n-1} + \tau - T_n) \quad (8)$$

2. Video rate determination: The ABR algorithm determines the rate  $U_{n+1}$  to be used for the next chunk by looking at several pieces of information, including (1) the sequence of rate estimates  $R_n$  and (2) the buffer occupancy level  $B_n$ . Hence,

$$U_{n+1} = F(B_n, \{R_m : m \leq n\}) \quad (9)$$

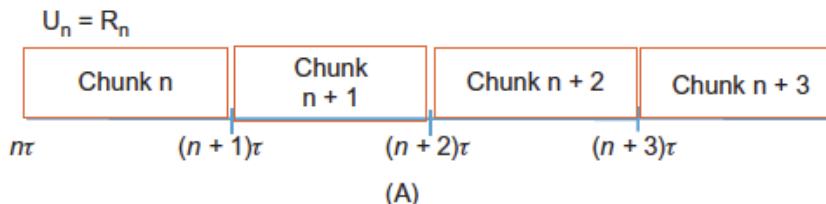
where  $F(\cdot)$  is a rate determination function.

3. Scheduling: The ABR algorithm determines the time when the next chunk is scheduled to be downloaded at by computing an HTTP inter-request time of  $\hat{T}_n$ , such that

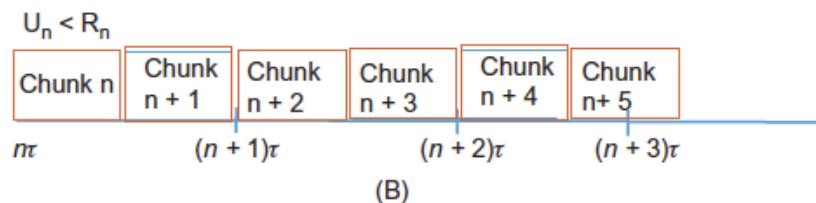
$$\hat{T}_n = G(B_n, \{R_m : m \leq n + 1\}) \quad (10)$$

# Chunk Download Times as a Function of Video Bitrate and TCP Throughput

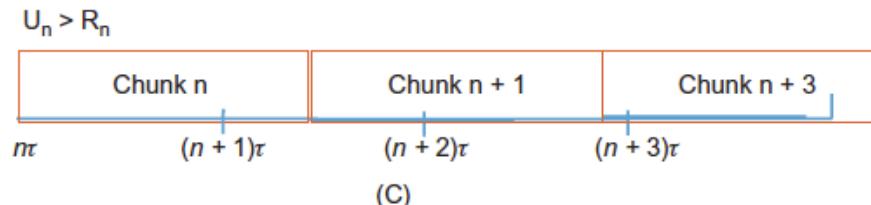
Ideal Situation



Buffer Overflow  
ABR Action:  
Increase  $U_n$



Buffer Underflow  
ABR Action:  
Decrease  $U_n$



Rate at which data is arriving is faster than the rate it is being consumed

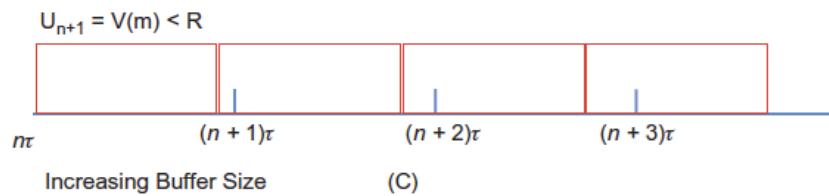
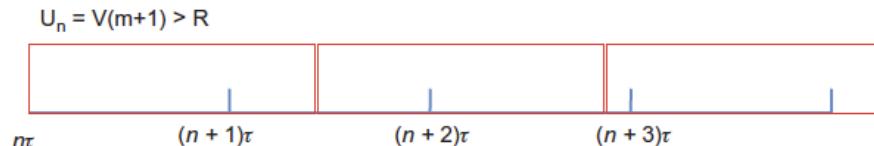
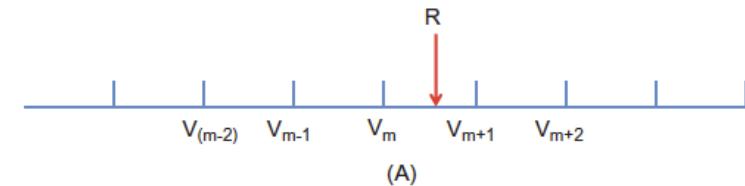
Rate at which data is arriving is slower than the rate it is being consumed

Playout Time

$$\frac{R_n}{U_n} = \frac{\tau}{\tilde{T}_n}$$

Download Duration

# Illustrating Fluctuation in Video Bitrate



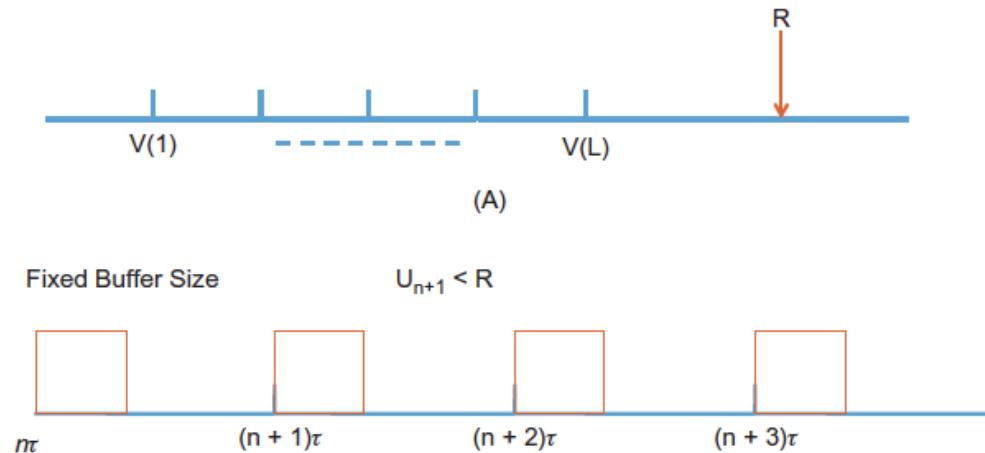
**Buffer Underflow**  
ABR Action: Decrease  $U_n$

**Buffer Overflow**  
ABR Action: Increase  $U_n$

What if  $U_n$  is already  
at its max?

- Because of the quantization in video rates, it is impossible to get to  $U_n = R$ , and this can result in the video rate constantly fluctuating between  $V(m)$  and  $V(m+1)$  to keep the buffer from overflowing or underflowing.
- To avoid this, most ABR algorithms stop increasing the video rate  $U_n$  after it gets to  $V(m)$  and compensate for the difference between  $V(m)$  and  $R$  by increasing the gap between successive downloads.

# What if the TCP Throughput $R$ is Larger than the Highest Video Bitrate

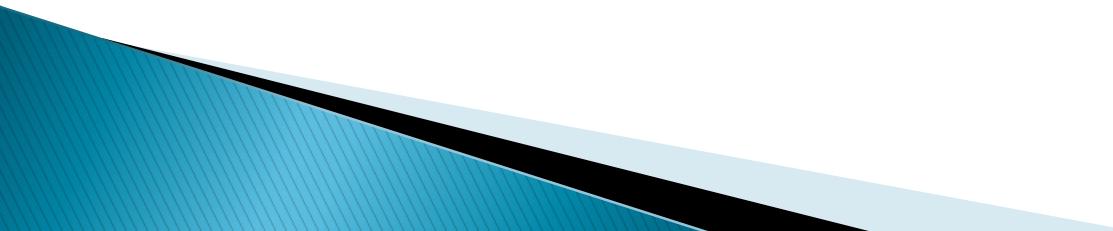


- In this case the receive buffer size keeps increasing.
- If the ABR algorithm observes that the buffer size is increasing even at the maximum video bit rate, then it can stabilize the buffer by increasing the gap between successive downloads

There are two ways in which it can choose the gap size:

- The gap size is fixed at  $\tau$ : In this case, during each period, the amount of video data consumed is equal to the data being added, resulting in a stable buffer.
- After the end of the chunk download, the ABR algorithm waits for the buffer to drain until it reaches a target level before starting the next download.

# Rate Based Algorithms

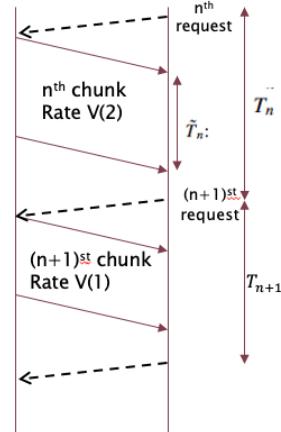


# NTB Algorithm

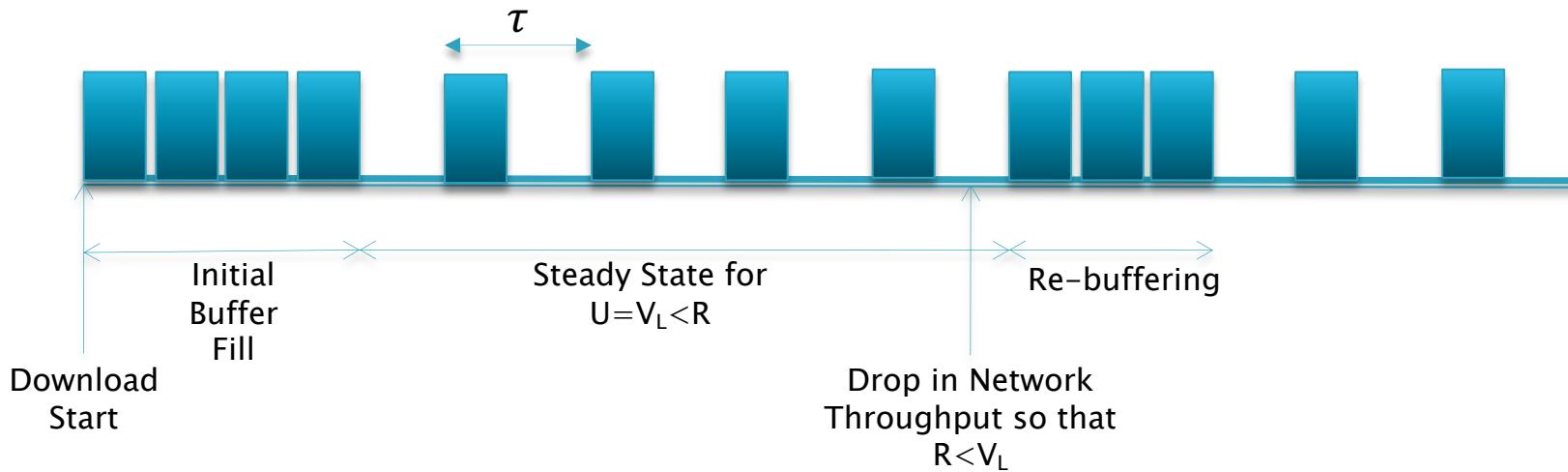
- ▶ The video bit rate for the next download is set equal to the latest value of the TCP throughput estimate, so that

$$U_{n+1} = Q(R_n) \quad \text{where} \quad R_n = \frac{U_n \tau}{\tilde{T}_n}$$

- ▶ The quantization function  $Q$  chooses the video bit rate that is smaller than  $R_n$  and closest to it.
- ▶ The rate estimate cancels out the instantaneous fluctuations in TCP throughput by averaging over an interval, which is typically 2 to 10 seconds long.
- ▶ The NTB algorithm is very effective in avoiding buffer underruns because it reacts instantaneously to fluctuations in TCP throughput. Studies have shown that receive buffer size shows the least amount of variation under NTB compared with the other algorithms.
- ▶ However, on the flip side, it has the largest rate of changes in the video bit rate because it does nothing to dampen any of these fluctuations. As a result, it is not commonly used in practice except as a way to benchmark other ABR algorithms.



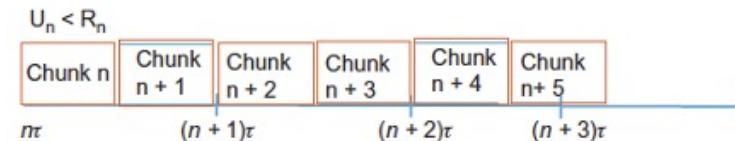
# Dynamics of the NTB Algorithm



This figure shows an example a typical sample path of the NTB algorithm.

- The video rate is typically set to the minimum value  $V(1)$  at start-up, but it quickly increases to the maximum bit rate  $V(L)$  because of application of equation 11.
- Downloads are done in a back-to-back manner until the receive buffer is full. When this happens, the download interval is increased to  $\tau$  seconds to keep the average TCP throughput  $R$  equal to  $V(L)$ .
- If the TCP throughput falls below  $V(L)$ , then the receive buffer occupancy starts to reduce, and the algorithm switches to the back-to- back downloads again. This takes the system to the scenario illustrated in Figure 6.9, where in the absence of any smoothing, the algorithm where in the absence of any smoothing, the algorithm will constantly switch between the bit rates that are above and below  $R$ .

# AIMD Based (ATB) Algorithm



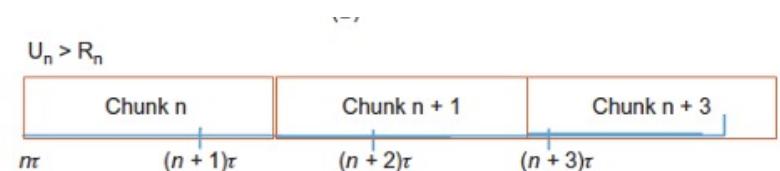
- Using the same notation as before, the ATB algorithm keeps track of the ratio  $\mu$  given by

$$\mu = \frac{\tau}{\widetilde{T}_n} = \frac{R_n}{U_n}$$

- if  $\mu < 1$ , then the video bit rate is larger than the TCP throughput, and the converse is true when  $\mu > 1$ . Hence,  $\mu$  serves as a measure of network congestion.
- The bit rate increment/decrement rules are as follows:

- If  $\mu > 1 + \varepsilon$  (and the buffer size exceeds a threshold) where  $\varepsilon = \max\left(\frac{V(i+1) - V(i)}{V(i)}, \forall i \in [1, \dots, L]\right)$ , then the chunk download happens faster than the rate at which the decoder can play that chunk (see Figure 8B). The video bit rate is switched up from  $V(m)$  to  $V(m + 1)$ , which corresponds to an additive increase policy. Hence, in contrast to NTB, the ATB algorithm increases the bit rate using an additive increase policy, which helps to reduce jumps in video quality. Also, the factor  $\varepsilon$  helps to reduce bit rate oscillations for the case when the TCP throughput lies between two video bit rates (illustrated in Figure 6.9A).
- If  $\mu < \gamma_d$ , where  $\gamma_d < 1$  is the switch-down threshold, then the download of a chunk takes longer than the time required for the decoder to play that chunk, which leads to a reduction in the receive buffer size. Hence, an aggressive switch down is performed, with the reduced video bit rate chosen to be the first rate  $V(i)$  such that  $V(i) < \mu V(c)$ , where  $V(c)$  is the current bit rate. This corresponds to a multiplicative decrease policy. The presence of the factor  $\gamma_d$  may lead to the situation in which the buffer drains down slowly. To prevent an underflow, the algorithm switches to the minimum bit rate  $V(1)$  if the buffer falls below a threshold.

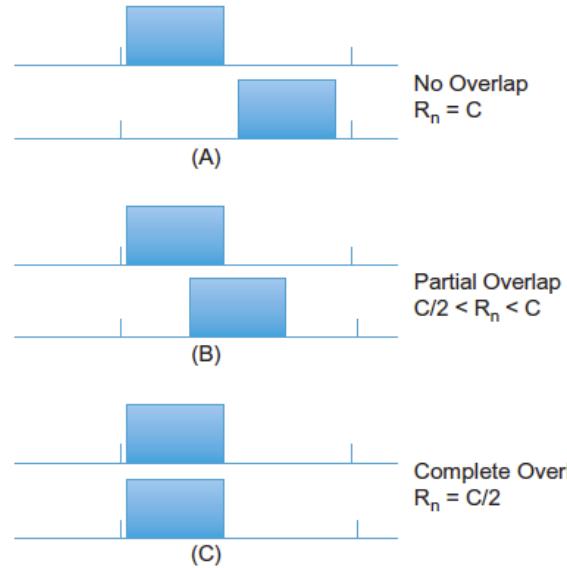
$$\begin{aligned} \frac{R}{V_n} &> 1 + \varepsilon \\ \Rightarrow R &> V_{n+1} \end{aligned}$$



# Issue with Rate Based Algorithms

Getting a good estimate of the TCP rate is not straightforward

- ▶ The chunk download start-time instances of the connections may get stuck in suboptimal locations. For example, if there are three active connections, then the chunk download times of two of the connections may overlap, while the third connection gets the entire link capacity. This leads to unfairness in the bit rate allocation.
- ▶ Connections with higher bit rate tend to see a higher estimate of their chunk's TCP throughput. This is because the chunks of connections with higher bit rates occupy the bottleneck link longer, and as a result, they have a greater chance of experiencing time intervals during which they have access to the entire link capacity.



Even with just 2 connections sharing a bottleneck link, the average TCP throughput over estimates the amount of bandwidth actually available

Correct Estimate

# TCP Bandwidth Overestimates

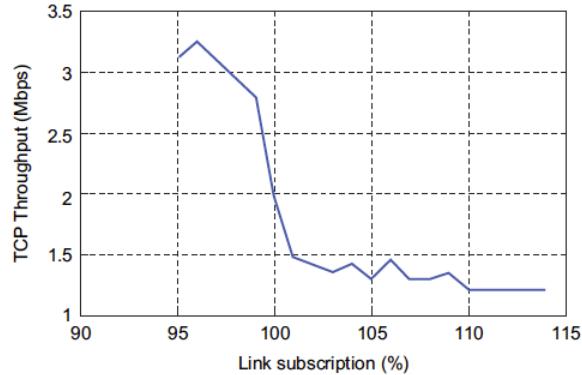


FIGURE 6.17

TCP throughput as a function of link subscription for 100 Adaptive Bit Rate (ABR) sources on a 100-mbps link.

If the link utilization is less than 100%, then the measured throughput is much higher than the fair bandwidth allocation (about three times the fair-share bandwidth in this example). This causes ABR to increase the video bit rate allocations. As a result of this, the link subscription soon rises above 100%, which causes the TCP throughput to fall precipitously, and the cycle repeats.

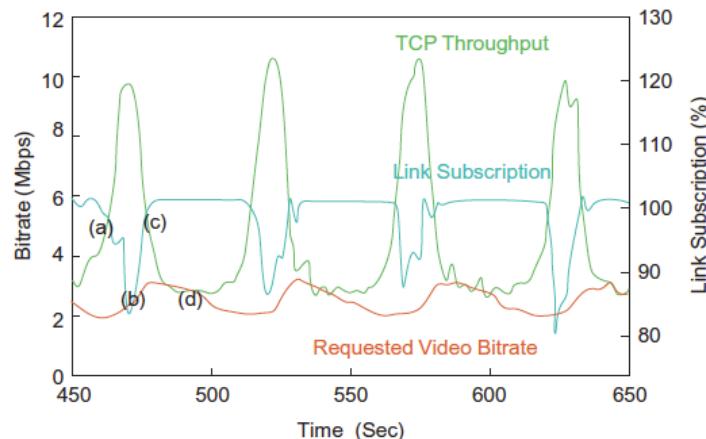


FIGURE 6.18

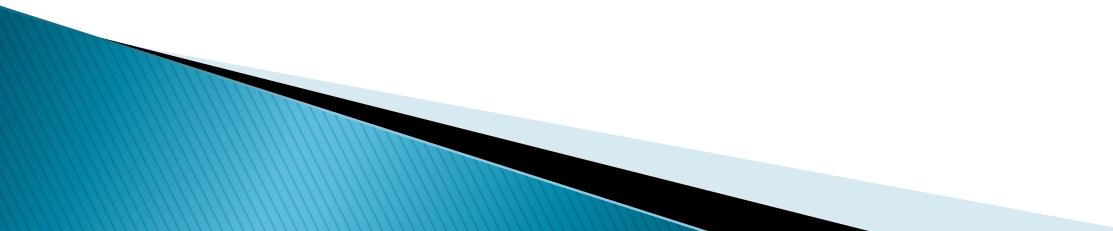
Video bit rate oscillation as a result of the bandwidth cliff effect.

# FESTIVE Algorithm: A Way to Reduce the Effect of BW Over-estimates

- ▶ To reduce the effect of BW unfairness between multiple connections, randomize the start of the chunk download times; that is, instead of downloading a chunk strictly at intervals of  $\tau$  seconds (or equivalently when the buffer size reaches a target value), move it randomly either backward or forward by a time equal to the length of download.
- ▶ Connections with higher bit rate tend to see a higher estimate of their chunk's TCP throughput. This is because the chunks of connections with higher bit rates occupy the bottleneck link longer, and as a result, they have a greater chance of experiencing time intervals during which they have access to the entire link capacity.  
To solve this problem, the rate of increase of the bit rate for a connection should not be linear but should decrease as the bit rate increases (another instance of the averaging principle!). This policy can be implemented as follows: If the bit rate is at level  $k$ , then increase it to level  $k+1$  only after  $k$  chunks have been received. This will lead to a faster convergence between bit rates of two connections that are initially separated from each other.
- ▶ Simple averaging of the TCP throughput estimates is biased by outliers if one chunk sees a very high or very low throughput. To avoid this, use a harmonic mean estimate of the last 20 throughput samples, which is less sensitive to outliers.

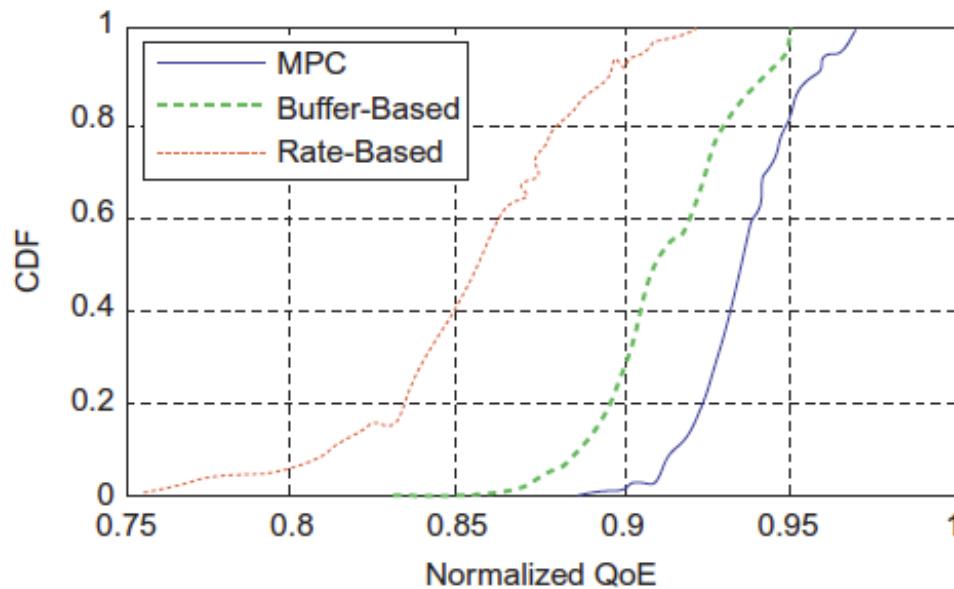
Also see PANDA algorithm to avoid video bitrate oscillation

# Buffer Size Based ABR Algorithms



# Buffer Size based Algorithms

- ▶ Buffer size based ABR algorithms use receive buffer size as their primary input, although most of them also use the TCP throughput estimate to do fine tuning.
- ▶ It has been recently shown that buffer size based algorithms outperform those based on rate estimates using Quality of Experience as a performance measure



Mapping of bitrate to perceived video quality

1. Average video quality:  $\frac{1}{K} \sum_{k=1}^K q(U_k)$
2. Average quality variations:  $\frac{1}{K} \sum_{k=1}^{K-1} |q(U_{k+1}) - q(U_k)|$
3. Total rebuffer time:  $\sum_{k=1}^K \left( \frac{\tau U_k}{R_k} - B_k \right)^+$   
Or number of rebufferings:  $\sum_{k=1}^K \mathbf{1}\left( \frac{\tau U_k}{R_k} > B_k \right)$

The QoE of video segments 1 through K are defined by a weighted sum of these components

$$QoE_1^K = \sum_{k=1}^K q(U_k) - \lambda \sum_{k=1}^{K-1} |q(U_{k+1}) - q(U_k)| - \mu \sum_{k=1}^K \left( \frac{\tau U_k}{R_k} - B_k \right)^+ \quad (19)$$

# Threshold Based Buffer (TBB) Algorithm

- Let  $B(t)$  be the receive buffer level at time  $t$ . Define the following threshold levels for the receive buffer, measured in seconds:  $0 \leq B_{\min} < B_{low} < B_{high}$ . The interval  $b_{tar} = [B_{low}, B_{high}]$  is called the target interval, and  $B_{opt} = (B_{low} + B_{high})/2$  is the center of the target interval. The TBB algorithm tries to keep the buffer level close to  $B_{opt}$ .
- Let  $V(c)$  be the current video bit rate. If the buffer level  $B(t)$  falls below  $B_{low}$  and  $V(c) > R_n$ , then switch to the next lower bit rate  $V(c-1)$ . The algorithm continues to switch to lower bit rates as long as the above conditions are true. Note that if  $B(t) < B_{low}$  and  $V(c) \leq R_n$ , it implies that the buffer level has started to increase, and hence there is no need to switch to a lower rate.
- With the current video bit rate equal to  $V(c)$ , if the buffer level increases above  $B_{high}$  and if  $V(c+1) < R_n^{avg}$ , then switch to the next higher video bit rate  $V(c+1)$ . Note that  $R_n^{avg}$  is the average TCP throughput, where the average is taken over the last few chunks (this number is configurable), which helps to dampen temporary network fluctuations.

If the video bit rate is already at the maximum value  $V(L)$  or if

$$V(c+1) \geq \alpha R_n^{avg}, \quad (14)$$

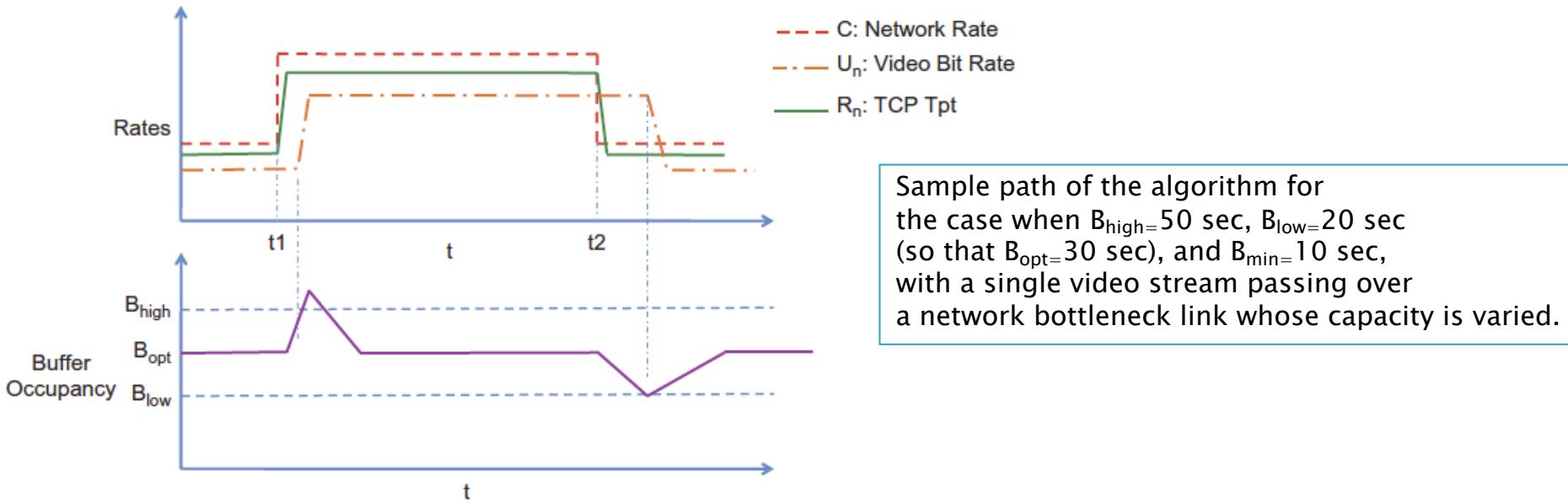
then the algorithm does not start the download of the next chunk until the buffer level  $B(t)$  falls below  $\max(B(t) - \tau, B_{opt})$ . Assuming that initially  $B(t) - \tau > B_{opt}$ , this policy leads to a steady linear decrease in buffer size by  $\tau$  every  $\tau$  seconds until it reaches the target operating point  $B_{opt}$ . If equation 14 is satisfied, then it means that the bit rates  $V(c)$  and  $V(c+1)$  straddle the TCP throughput (as in Figure 6.9A) so that increasing the bit rate to  $V(c+1)$  will cause the buffer to start decreasing.

# Threshold Based Buffer (TBB) Algorithm

- If  $B(t) \in b_{tar}$ , the algorithm does not change the video bit rate to avoid reacting to short-term variations in the TCP throughput. If the bit rate is at maximum value  $V(L)$  or if equation 14 is satisfied, then the algorithm does not start the download of the next chunk until the buffer level  $B(t)$  falls below  $\max(B(t) - \tau, B_{opt})$ . This keeps the buffer level at  $B_{opt}$  in equilibrium.
- If the buffer level falls below  $B_{min}$ , then switch to the lowest video bit rate to  $V(1)$ .

A nice feature of the TBB algorithm is that it enables the designer to explicitly control the trade-off between variation in buffer occupancy and fluctuations in video bit rate in response to varying TCP throughput. This is done by controlling the thresholds  $B_{high}$  and  $B_{low}$ , such that a large value of the difference ( $B_{high} - B_{low}$ ) will reduce video bit rate changes.

# Illustration of the Bitrate and Buffer Size Dynamics for the TBB Algorithm



- When the link capacity goes up at  $t_1 = 200$  sec, the buffer size starts to increase, and when it crosses 50 sec, the algorithm starts to increase the bit rate in multiple steps. When the bit rate reaches the maximum value, the algorithm reduces the buffer size linearly until it reaches  $B_{\text{opt}}$ .
- When the link capacity is reduced at  $t_2 = 400$  sec, it causes the buffer occupancy to decrease, and when it drops below 20 sec, the video bit rate is progressively decreased until it falls below the TCP throughput value. At this point, the buffer starts to fill up again, and when it reaches  $B_{\text{opt}}$ , the algorithm spaces out the chunks to maintain the buffer at a constant value.

# Control Theory Based ABR Algorithms



# Control Theory based Algorithms

- ▶ Formal Control Theory can be used to design ABR Algorithms. These require that the performance measure to be optimized be explicitly defined, in addition to a model for the system under consideration.
- ▶ We will look at two types of algorithms:
  - Model based systems: These require an explicit model for parts of the system that evolve randomly, in this case it is the TCP throughput for the network. Model Predictive Control (MPC) is an example of this type of system
  - Model Free Control: These are based on Reinforcement Learning (RL). Model Free versions of RL can be used for Optimal Control of systems for which sample paths for the random evolution of the system state are available.

# MPC Performance Measure: QoE (Quality of Experience)

Define the following Quality of Experience (QoE) elements for the system:

1. Average video quality:  $\frac{1}{K} \sum_{k=1}^K q(U_k)$
2. Average quality variations:  $\frac{1}{K} \sum_{k=1}^{K-1} |q(U_{k+1}) - q(U_k)|$
3. Total rebuffer time:  $\sum_{k=1}^K \left( \frac{\tau U_k}{R_k} - B_k \right)^+$

Or number of rebufferings:  $\sum_{k=1}^K \mathbf{1}\left(\frac{\tau U_k}{R_k} > B_k\right)$

The QoE of video segments 1 through K are defined by a weighted sum of these components

$$QoE_1^K = \sum_{k=1}^K q(U_k) - \lambda \sum_{k=1}^{K-1} |q(U_{k+1}) - q(U_k)| - \mu \sum_{k=1}^K \left( \frac{\tau U_k}{R_k} - B_k \right)^+ \quad (19)$$

# MPC Problem Statement: Maximize QoE

The bit rate adaptation problem, called  $QoE\_MAX_1^K$ , is formulated as follows:

$$\max_{U_1, \dots, U_K} QoE_1^K \quad (20)$$

such that

$$t_{k+1} = t_k + \frac{\tau U_k}{R_k}$$
$$B_{k+1} = \left( B_k - \frac{\tau U_k}{R_k} \right)^+ + \tau$$

Source of Randomness

$$R_k = \frac{1}{t_{k+1} - t_k} \int_{t_k}^{t_{k+1}} R(t) dt$$

$$U_k \in V, \quad B_k \in [0, B_{\max}] \quad \forall k = 1, \dots, K$$

The input to the problem is the bandwidth trace  $R_t, t \in [t_1, t_{K+1}]$ , which is a random sample path realization from the stochastic process that controls TCP throughput variations during the download.

The outputs of  $QoE\_MAX_1^K$  are the bit rate decisions  $U_1, \dots, U_K$ , the download times  $t_1, \dots, t_K$  and the buffer occupancies  $B_1, \dots, B_K$ .

Problem with this algo: At time  $t_k$  when the ABR algorithm is invoked to choose  $U_k$ , only the past throughputs  $\{R_i; i < k\}$  are known; the future throughputs  $\{R_i; i >= k\}$  are not known.

# MPC Optimization: Use of Bandwidth Predictors

- ▶ Bandwidth predictors can be used to obtain predictions defined as  $\{R_i \geq k\}$ .
- ▶ Several bandwidth predictors using filtering techniques such as AR (autoregressive), ARMA (autoregressive moving average), autoregressive integrated moving average (ARIMA), fractional ARIMA (FARIMA), and so on are known, which are all combinations of moving average and autoregressive filtering.
- ▶ Based on this, the ABR algorithm selects bit rate of the next chunk  $k$  as:

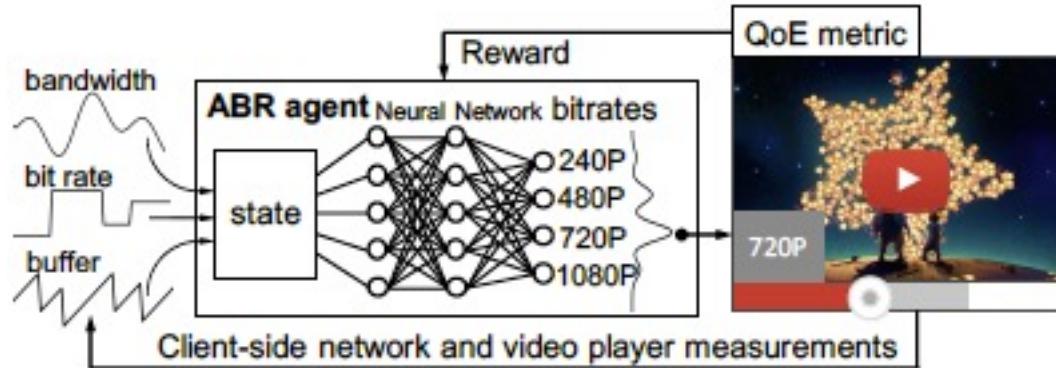
$$U_k = f(B_k, \{\hat{R}_i, i \geq k\}). \quad (21)$$

Model predictive control [20] is a subset of this class of algorithms that choose bit rate  $U_k$  by looking  $h$  steps ahead; that is, they solve the QoE maximization problem  $QoE\_MAX_k^{k+h}$  with bandwidth predictions  $\{\hat{R}_i, k \leq i \leq k + h\}$  to obtain the bit rate  $U_k$ . The bit rate  $U_k$  is then applied to the system, and that along with the actual observed TCP throughput  $R_k$  is used to iterate the optimization process to the next step  $k + 1$ .

# Reinforcement Learning Based ABR Algorithm: Pensieve

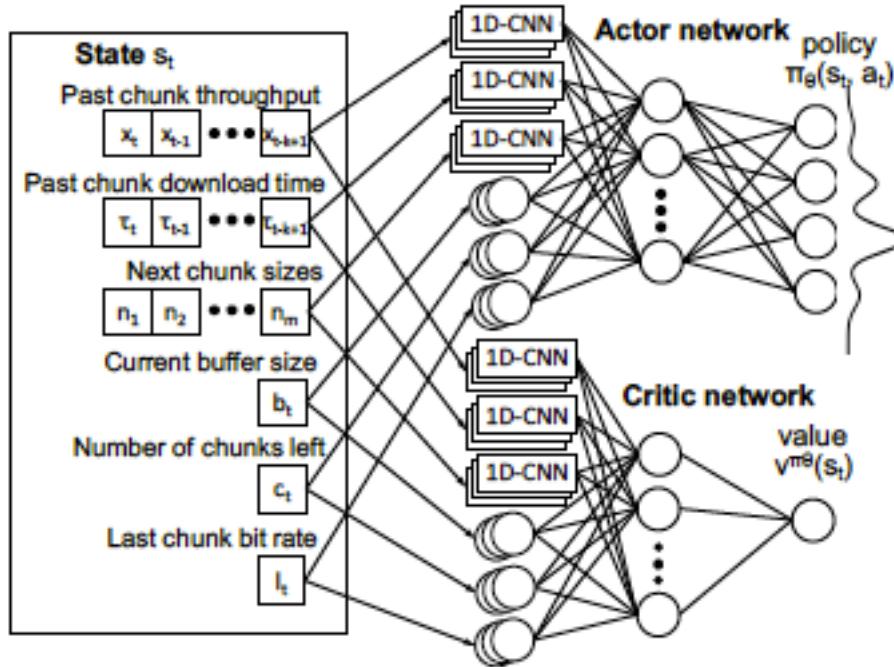
- ▶ Issue with MPC: It relies heavily on accurate throughput estimates that are not always available. When these predictions are incorrect, MPC's performance can degrade significantly.
- ▶ Pensieve is a system that learns ABR algorithms automatically, without using any pre-programmed control rules or explicit assumptions about the operating environment.
- ▶ It learns a control policy for bitrate adaptation purely through experience. During training, Pensieve starts knowing nothing about the task at hand. It then gradually learns to make better ABR decisions through reinforcement, in the form of reward signals that reflect video QoE for past decisions.

# Penseive



- Pensieve represents its control policy as a neural network that maps “raw” observations (e.g., throughput samples, playback buffer occupancy, video chunk sizes) to the bitrate decision for the next chunk.
- The resulting QoE is then observed and passed back to the ABR agent as a reward. The agent uses the reward information to train and improve its neural network model.
- The neural network provides an expressive and scalable way to incorporate a rich variety of observations into the control policy.
- Pensieve trains this neural network using A3C, a state-of-the-art actor-critic RL algorithm.
- To train its models, Pensieve uses simulations over a large corpus of network traces.

# Penseive: Actor–Critic Network



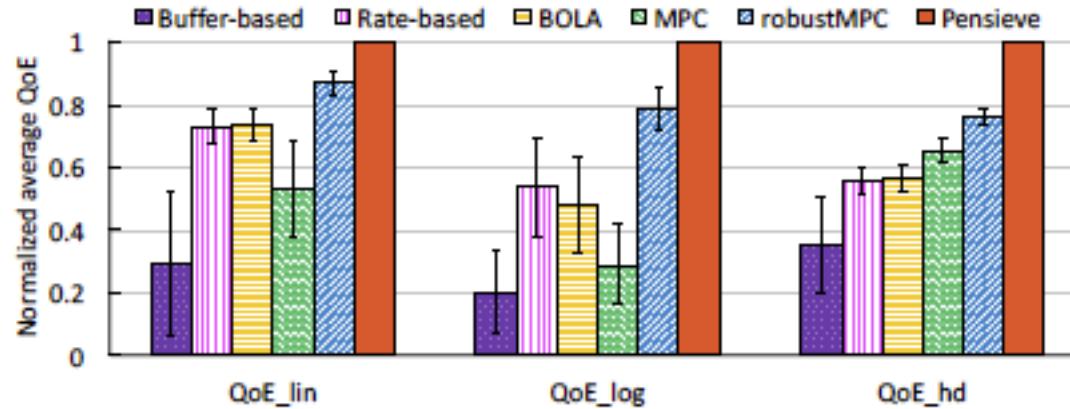
**Policy:** Upon receiving  $s_t$ , Pensieve's RL agent needs to take an action  $a_t$  that corresponds to the bitrate for the next video chunk. The agent selects actions based on a *policy*, defined as a probability distribution over actions  $\pi : \pi(s_t, a_t) \rightarrow [0, 1]$ .  $\pi(s_t, a_t)$  is the probability that action  $a_t$  is taken in state  $s_t$ . In practice, there are

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) A(s_t, a_t), \quad \text{Updating the Actor Network parameters}$$

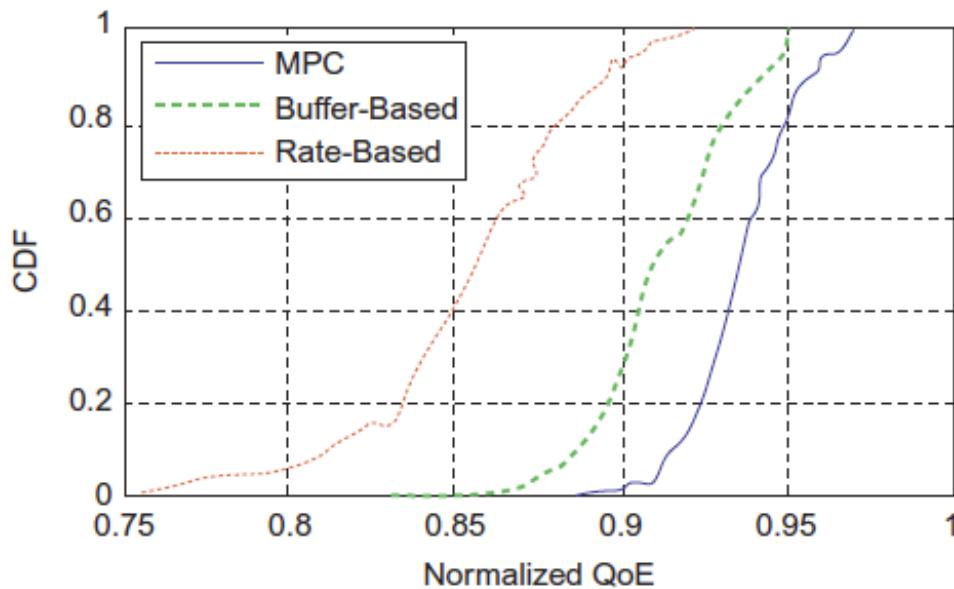
$$\theta_v \leftarrow \theta_v - \alpha' \sum_t \nabla_{\theta_v} (r_t + \gamma V^{\pi_\theta}(s_{t+1}; \theta_v) - V^{\pi_\theta}(s_t; \theta_v))^2, \quad \text{Updating critic network parameters}$$

where  $V^{\pi_\theta}(\cdot; \theta_v)$  is the estimate of  $v^{\pi_\theta}(\cdot)$ , output by the critic network, and  $\alpha'$  is the learning rate for the critic. For an experience  $(s_t, a_t, r_t, s_{t+1})$  (i.e., take action  $a_t$  in state  $s_t$ , receive reward  $r_t$ , and transition to  $s_{t+1}$ ), the advantage  $A(s_t, a_t)$  can now be estimated as  $r_t + \gamma V^{\pi_\theta}(s_{t+1}; \theta_v) - V^{\pi_\theta}(s_t; \theta_v)$ . See [24] for more details.

# Penseive Performance



# Buffer Size based Algorithms



Mapping of bitrate to perceived video quality

1. Average video quality:  $\frac{1}{K} \sum_{k=1}^K q(U_k)$
2. Average quality variations:  $\frac{1}{K} \sum_{k=1}^{K-1} |q(U_{k+1}) - q(U_k)|$
3. Total rebuffer time:  $\sum_{k=1}^K \left( \frac{\tau U_k}{R_k} - B_k \right)^+$   
Or number of rebufferings:  $\sum_{k=1}^K \mathbf{1}\left( \frac{\tau U_k}{R_k} > B_k \right)$

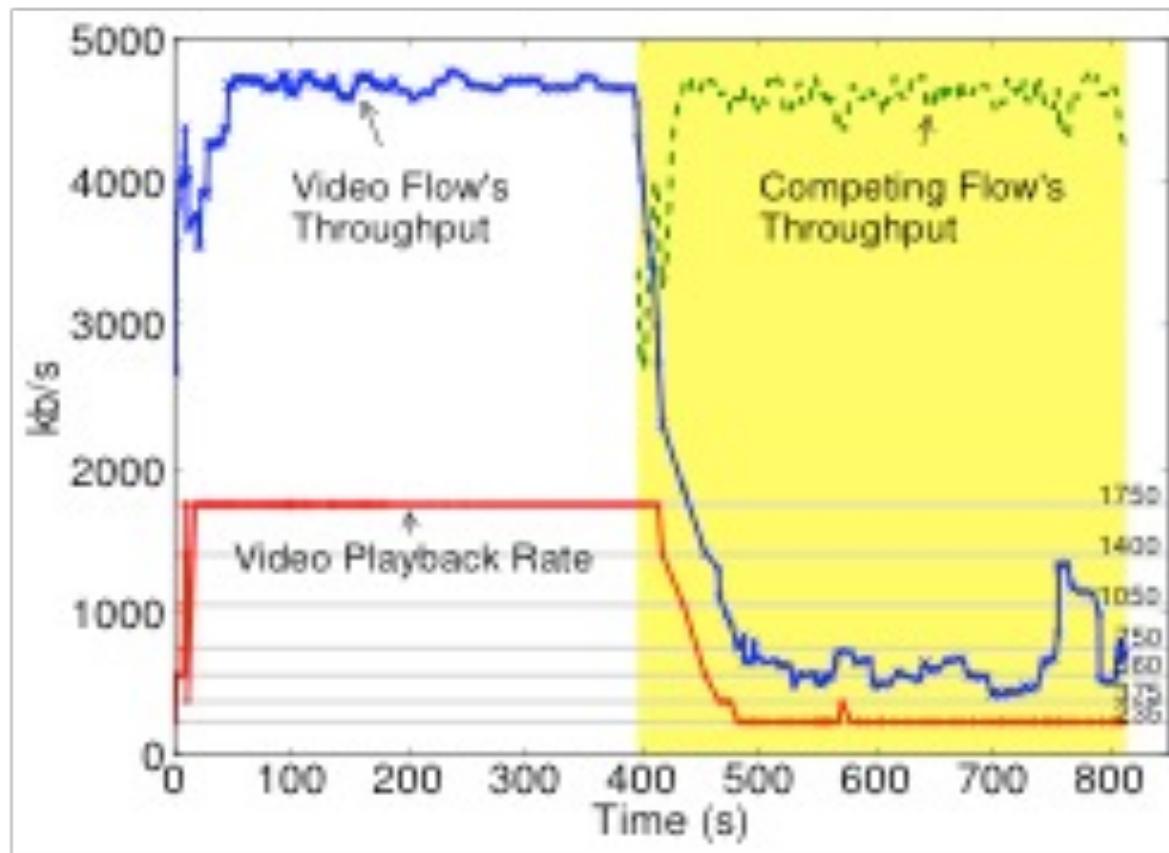
The QoE of video segments 1 through K are defined by a weighted sum of these components

$$QoE_1^K = \sum_{k=1}^K q(U_k) - \lambda \sum_{k=1}^{K-1} |q(U_{k+1}) - q(U_k)| - \mu \sum_{k=1}^K \left( \frac{\tau U_k}{R_k} - B_k \right)^+ \quad (19)$$

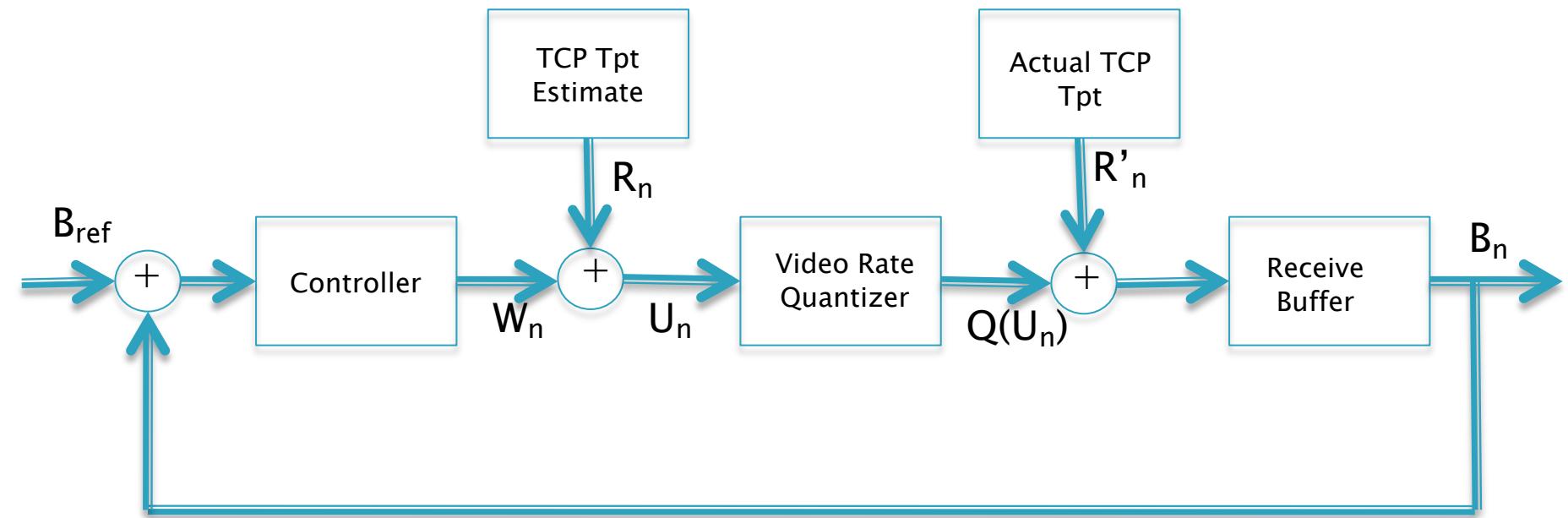
# Interaction between ABR and TCP



# Loss in video throughput in the presence of a competing TCP flow



# Application of Control Theory to the ABR Algorithm



# Further Reading

- ▶ Chapter 6 of Internet Congestion Control