

Play-Out Buffering

CSC 790

WAKE FOREST
UNIVERSITY

Department of Computer Science

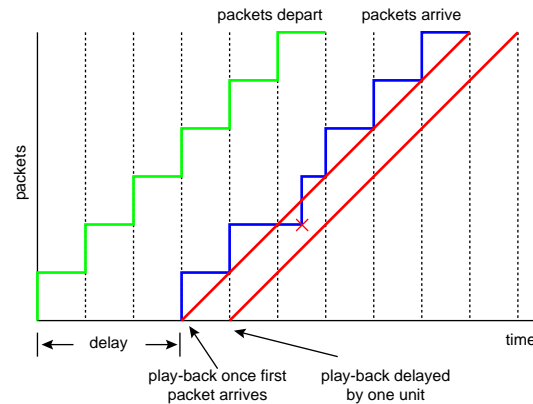
Fall 2009

Internet and Best Effort

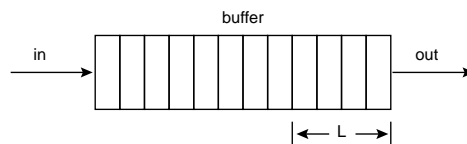
- Internet only provides Best Effort (BE) service
 - Service depends on the current network conditions
- Given BE service, must compensate at the sender and receiver
 - Changes made only at the sender and receiver (not the network)
- In contrast IntServ and DiffServ solutions will modify the network

Jitter Buffer

- Consider a sequence of packet transmitted through the network



- A jitter buffer may be necessary for playing the media
 - Buffer is similar to a leaky bucket



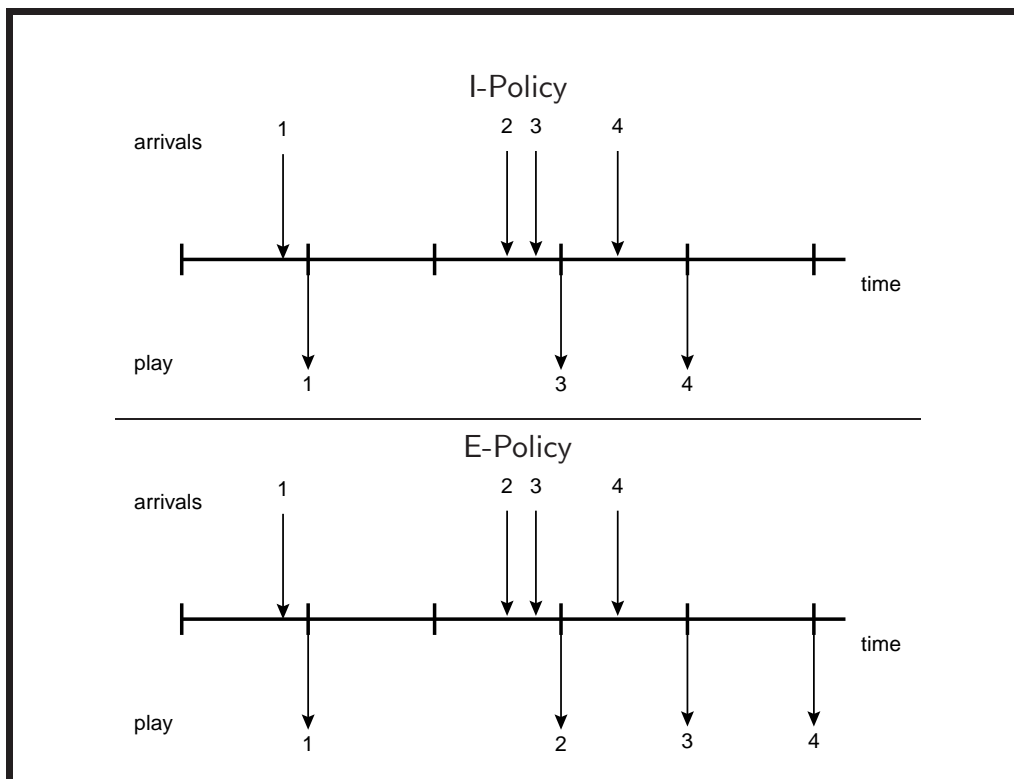
- Let $a(t)$ be the arrival process, $d(t)$ the departure process, and l the jitter buffer length
 - Want to *pre-fetch* l samples before play-out begins
 - l should be large enough to compensate for jitter, but ...

Let a be the arrival process, d the departure process, and l the buffer length. Assuming $d = 2$ packets/sec and $l = 40$ packets, what is the maximum jitter?

So what should you base the buffer length on? What is the major problem?

Play-Out Policy

- When packets arrive late at destination play-out is invoked
 - Determines if/when data is played
- Two simple examples to consider
 - **I-Policy**, data played at fixed intervals, any data arriving after play time is discarded (gaps, no delays)
 - **E-policy**, late data played at next opportunity (no gaps, delay)



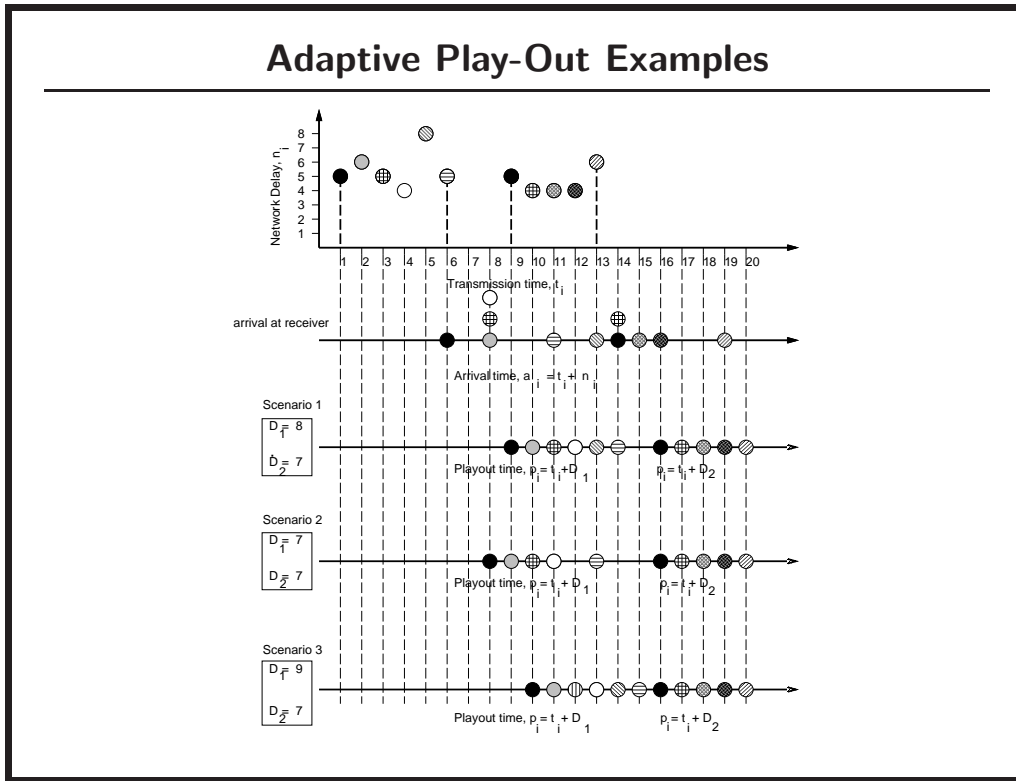
Policy Summary

- I-policy never adjusts display latency, rather one chooses initial display latency at the beginning and maintain it throughout
- In contrast, e-policy dynamically adjusts upward in response to jitter, adjusting latency to be higher than any delay yet observed
- *What is the correct amount of latency?*
 - Depends on the application
- Really prefer policy that can adjust dynamically... (*up and down*)

Adaptive Play-Out Delay

- Previous examples assumed a constant play-out rate
 - *Perhaps it is possible to adjust d (play-out rate)*
- Consider VoIP, conversations consist of talk "spurts"
 - Conversation followed by silence
 - No real need to transmit silence

How can we determine silence periods?
- Therefore it may be possible to artificially elongate and compress silence periods to better manage buffer



Ramjee's Algorithm

- An adaptive play-out algorithm, let
 - t_i time packet i is generated by sender
 - r_i time packet i is received by destination
 - p_i time packet i is played at destination
- The end-to-end delay of the i^{th} packet is $r_i - t_i$
 - This will change over time, need an estimate
 - Let d_i be average delay upon receiving packet i

$$d_i = \alpha \cdot (r_i - t_i) + (1 - \alpha) \cdot d_{i-1}$$
 - Let v_i be estimated average deviation of the delay

$$v_i = \alpha \cdot |(r_i - t_i) - d_i| + (1 - \alpha) \cdot v_{i-1}$$
- Estimates are calculated for every packet received

- If packet i is the first in *talk spurt*, its play-out time is

$$p_i = t_i + d_i + \beta \cdot v_i$$

where β is a positive constant, therefore as $v, d \rightarrow \infty$ delay increases (buffering more packets)

- $\beta \cdot v_i$ attempts to delay the play-out so to avoid losses due to late arrivals

- Play-out of subsequent packets in spurt is based on the first, let

$$q_i = p_i - t_i$$

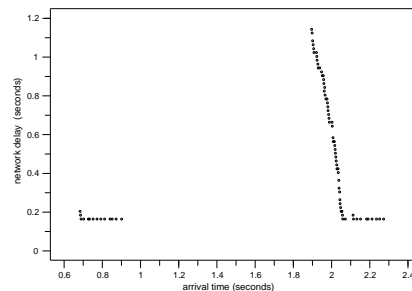
then

$$p_j = t_j + q_i = t_j + (p_i - t_i)$$

- *How do you know if a packet starts a talk spurt?*
 - Compare the timestamp of the i^{th} with the $(i - 1)^{th}$, if $(t_i - t_{i-1}) > x$ ($x = 20$ msec) then talk spurt started

Talk Spikes

- When audio is transmitted via the Internet, *talk spikes* may occur



- A spike is a sudden large increase in end delay, followed by a series of packets arriving almost simultaneously
 - As a result, Ramjee's algorithm may be too slow to react

Too slow to increase delay or decrease delay?

Modification to Ramjee's

- A simple modification is to include spike detection
 - Spike characterized by sudden large increase in delay, detection for beginning is simple check delay between packets

$$\text{if}(|n_i - n_{i-1}| > \text{spikeThreshold})$$

$$d_i = d_{i-1} + n_i - n_{i-1}$$

where $n_i = a_i - t_i$ is the network delay

- spikeThreshold in the original paper is

$$2 \cdot |v_i| + 800$$

where $v_i = 0.125 \cdot |n_i - d_i| + 0.875 \cdot v_{i-1}$

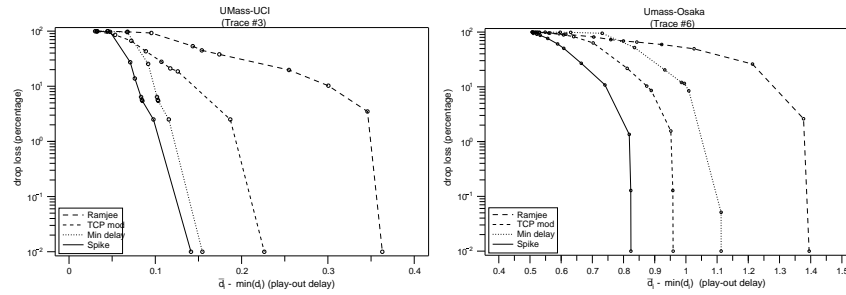
- Determining the end of a spike is more difficult
 - A “prominent characteristic” is the packets immediately after the beginning arrive back-to-back
 - Network delay decreases, thus can use the slope to detect end

- For example, use the most recent delay observations

$$\text{var} = \frac{\text{var}}{2} + \left| \frac{n_i - n_{i-1}}{8} - \frac{n_i - n_{i-2}}{8} \right|$$

- Value will be non-zero and can be used to detect spike end
- See: “Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks,” Ramachandran Ramjee, Jim Kurose, Don Towsley, and Henning Schulzrinne

Example Performance



- Spike detection performs best when delay variation is high

Can we do something similar for video?

Adjusting Display Latency

- Audio can be modeled as series of sound and silence
 - Adjust latency between the talk spurts
- Perhaps possible to adjust display latency between talk spurts
 - Observe delay over last m audio fragments (*save as set*)
 - Discard k largest delays from set
 - Display latency is greatest remaining delay from set

Does this work for all types of video?

Monitoring the Queue

- Measuring the end-to-end delay is difficult, instead observe the length of the display queue over time
 - If queue length constant, constant delay
 - If queue reduces, delay increasing
 - If queue increases, delay is decreasing

For each possible display queue length we define a threshold value. The threshold value for queue length n specifies a duration (measured in frame times) after which, if the display queue has continuously contained more than n frames, we will conclude that display latency can be reduced without increasing the frequency of gaps. This policy has the effect of reducing display latency quickly after long delays due to large bursts of network traffic, but approaching minimal latency slowly.

- Implementation

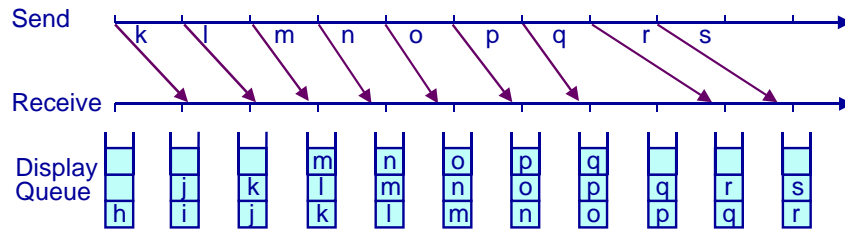
To implement queue monitoring, we associate an array of counters and threshold values with the display queue. Each time we wish to display a new frame, we first perform a thresholding operation. If the display queue has length m , then counters 2 through $m - 1$ are incremented and all other counters are reset. If any counter exceeds its associated threshold value, all the counters are reset and the oldest frame is discarded from the display queue. The oldest remaining frame is then displayed.

- But wait, there's more

An important principle in this implementation is that the thresholding operation will never discard frames unless the display queue contains more than two frames. The last frame in the display queue should never be discarded because there must be a frame available for display after the thresholding operation completes.

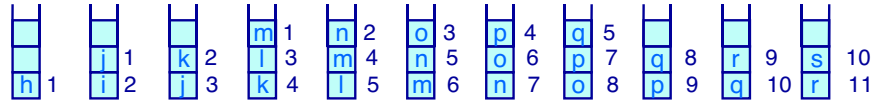
- See "An Empirical Study of Delay Jitter Management Policies," Donald L. Stone and Kevin Jeffay

Queue Monitoring Example

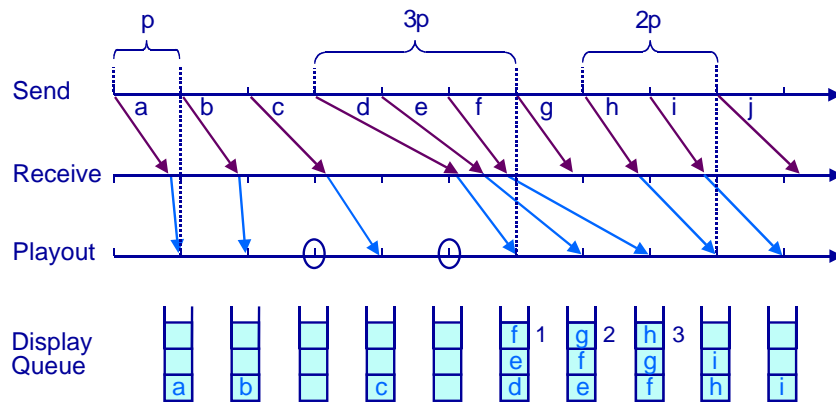


- Numbering per frame would be the following

Display Queue



- Example, thresholds = 3, 10



- When count exceeds threshold, oldest frame discarded
 - Frame g discarded at time 9