CS144
An Introduction to Computer Networks

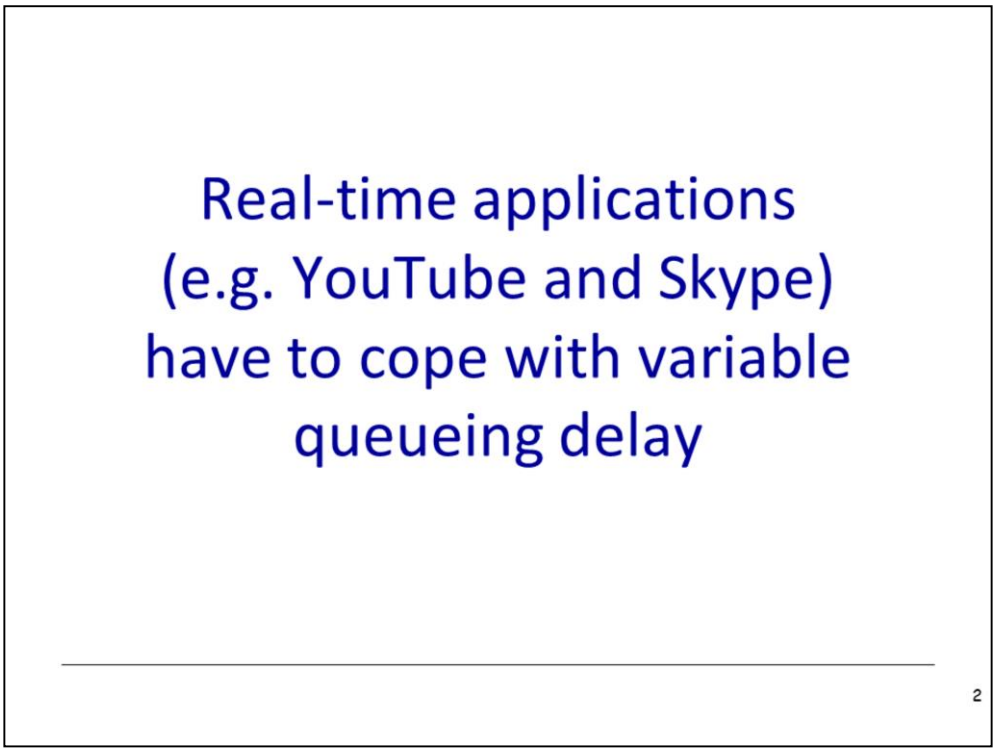
Packet Switching

Playback Buffers

Nick McKeown
Professor of Electrical Engineering
and Computer Science, Stanford University

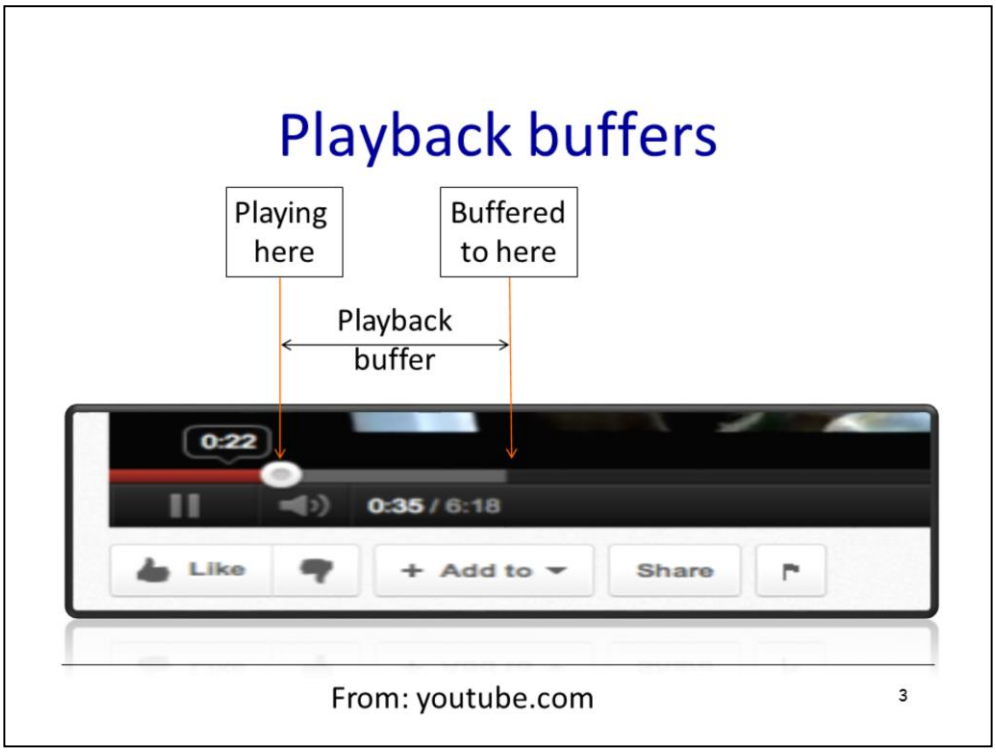
By now you know how to calculate the end to end delay of a packet across a network, and you know that the queueing delay makes the end to end delay variable.

Many of the applications we use don't particularly care about the variability in end to end delay. For example, when we're downloading a web page or sending an email, we want it to complete quickly, but we don't particularly mind if individual packets take 10 or 12ms to reach the other end.

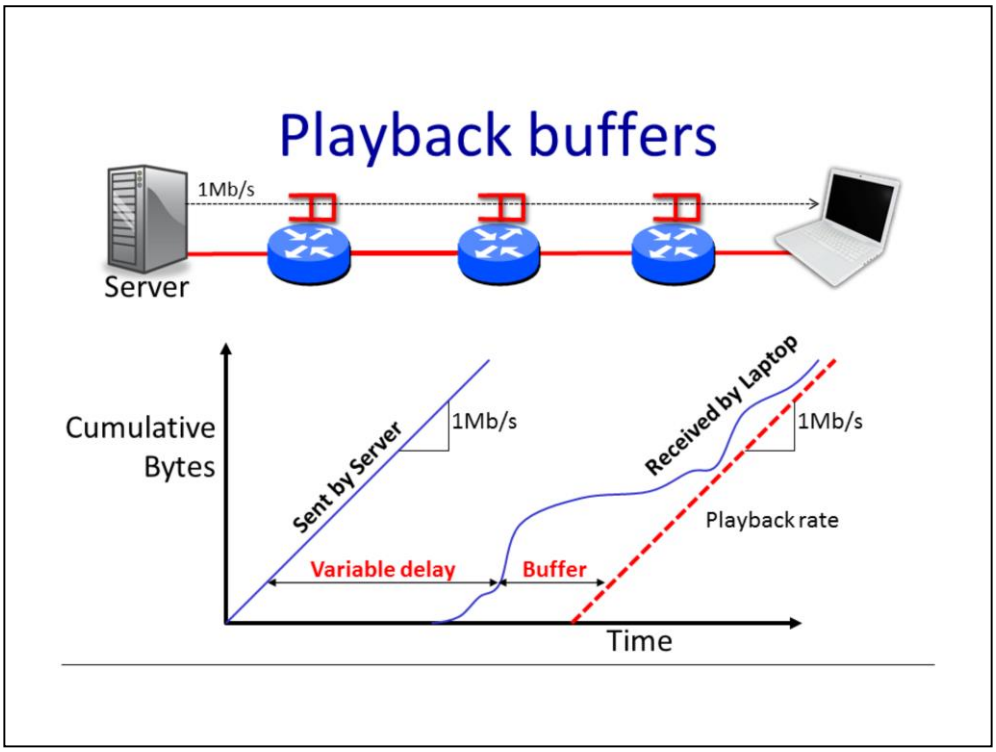


For many of the applications we use, the variable queueing delay isn't a problem - we don't particularly care about the precise arrival time of packets when we are browsing the web or sending email. We *do* care that our web pages load quickly, or our emails are sent promptly – but we don't particular mind if our packets arrive after 100ms or 120ms.

But some applications *have* to care about the queueing delay; particularly real time applications such as streaming video and voice. Let's take a look at an example. Over the next few minutes I'm going to explain why queueing delay makes life hard for these applications. It's serves as a good illustration of queueing delay, and how we mitigate the problem in practice. Basically, because the applications don't know precisely when the packets will show up, they can't be sure they will have a voice or video sample in time to deliver it to the user. And so they build up a reserve of packets in something called the PLAYBACK BUFFER…



You've all seen a playback buffer before. This is a screenshot from a YouTube client. The red line on the left shows how much of the video we've watched so far. The dot shows where we've got to so far, and the grey area shows how much of the video the client has received but hasn't played back yet. The client deliberately tries to get ahead just in case some of the packets are delayed and don't arrive in time, or in case there is a temporary outage. When designing a playback buffer we have to think about how far ahead we want to buffer, and how much we want to accumulate in the buffer before we start playing back the video to the user. Let's take a closer look…



Imagine we are watching a YouTube video on the laptop on the right. In our example, the video is being streamed at 1Mb/s from the server on the left, and passes through several routers along the path.

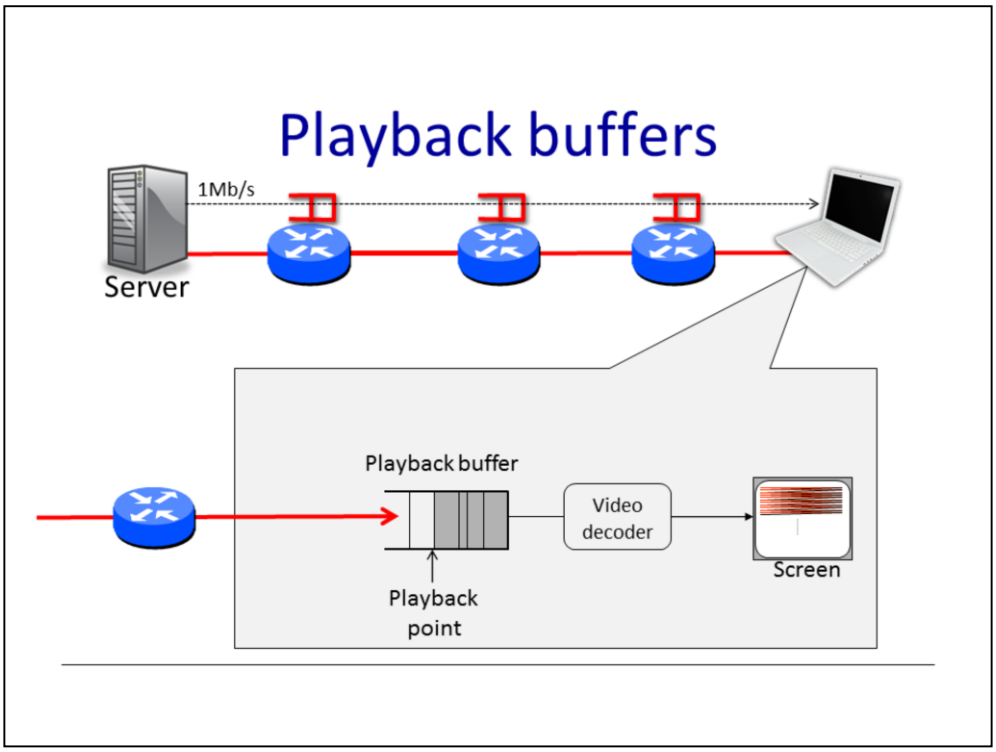
The graph shows the cumulative number of byte sent by the server. Because the server is sending at a constant 1Mb/s, the gradient is 1Mb/s. i.e. after 1s it has sent 1Mbits of data, and after 10seconds it has sent 10Mbits.

Because fo the variable queueing delay in the network, the cumulative arrivals at the laptop might look like this. The biggest component of the delay is the propagation and packetization delay; the variable part of the delay is the queueing delay in the packet buffers. The actual shape of the arrival graph could look very different from this, I just made up this shape. However, we do know a couple of things about it. First, the overall end to end delay can't be less than the packetization and propagation delay – it has a lower bound. It also has an upper bound: The queueing delay can't be larger than the sum of all the queueing delays in the routers along the path. Because the routers have finite buffers, and because they serve the packets in FIFO order, there is a maximum delay. Unfortunately, this delay could be huge – the buffers often delay packets by half a second or more. We also know that the cumulative arrivals are non-decreasing – the value can only increase. And finally, the

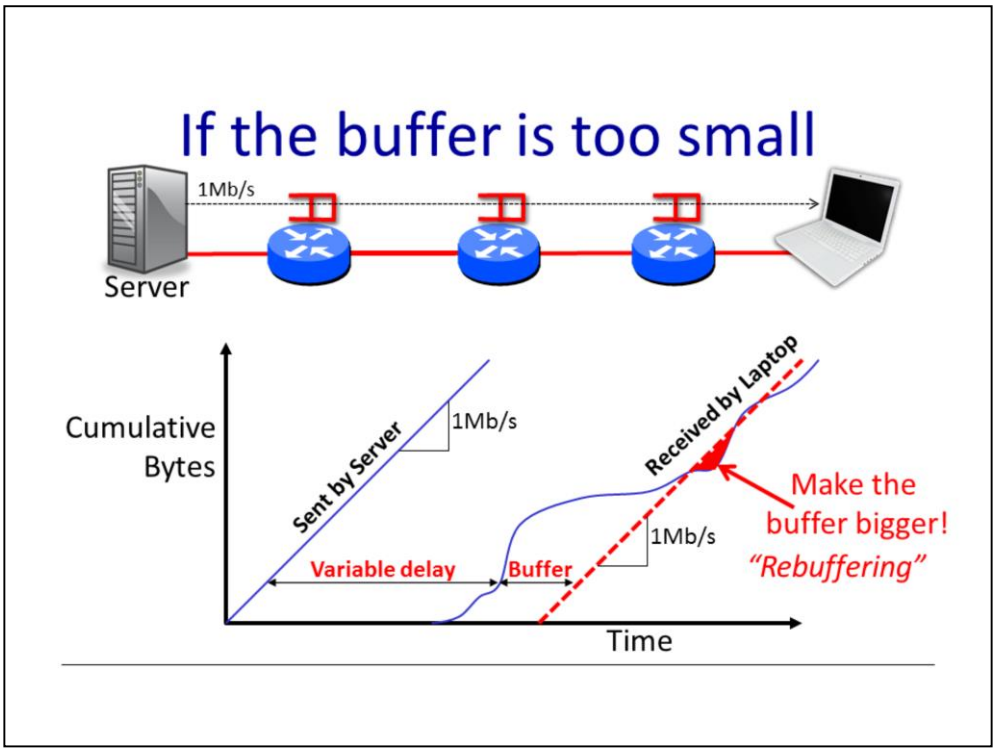
instantaneous arrival rate can't exceeed the speed of the last link.

OK, with all those caveats, let's look at what the client needs to do.

This red line shows the rate at which the client needs to playback the video to the user – it's exactly the same as the rate sent by the server, 1Mb/s. The client needs to BUFFER up enough bytes of data, so that it never goes empty. We say that if the playback buffer goes empty it under-runs. It means the user has no more video and the screen has to freeze. Something we've all seen before. In my example here, the buffer never goes empty and the video is fine.



If we look inside the client we can see the playback buffer and the place it has got to in the video. This is the big dot on the YouTube video client. After the data leaves the playback buffer it goes into the video decoder and placed on the screen.

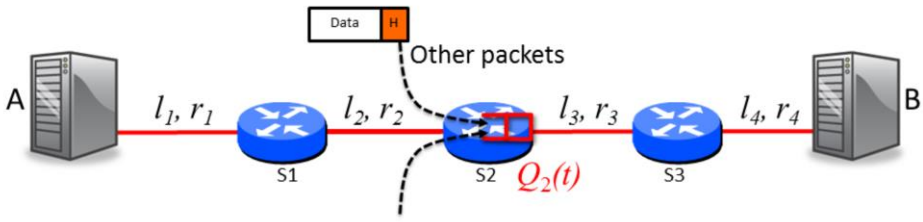


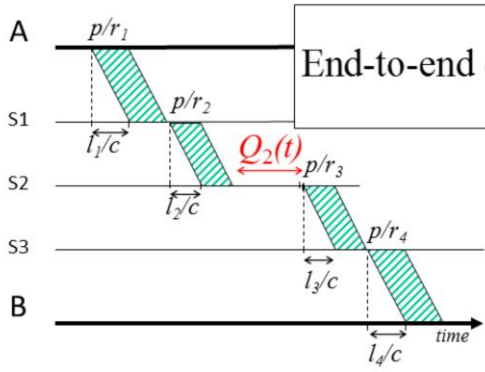
If we make the buffer too small, or start playing back the video too soon, then the playback buffer goes empty and the screen freezes. We have to pause, build up the buffer again, then resume playing out the video at 1Mb/s again. This is often called a Rebuffering Event, and can be pretty annoying. If you're watching this video over a slow link, or from a long way away you migth experience a rebuffering event. You can fix the problem by streaming at a slower rate, or by downloading the video ahead of time.

Playback buffer

- With packet switching, end-to-end delay is variable.
- We use a playback buffer to absorb the variation.

- We could just make the playback buffer very big, but then the video would be delayed at the start.
- Therefore, applications estimate the delay, set the playback buffer, and resize the buffer if the delay changes.

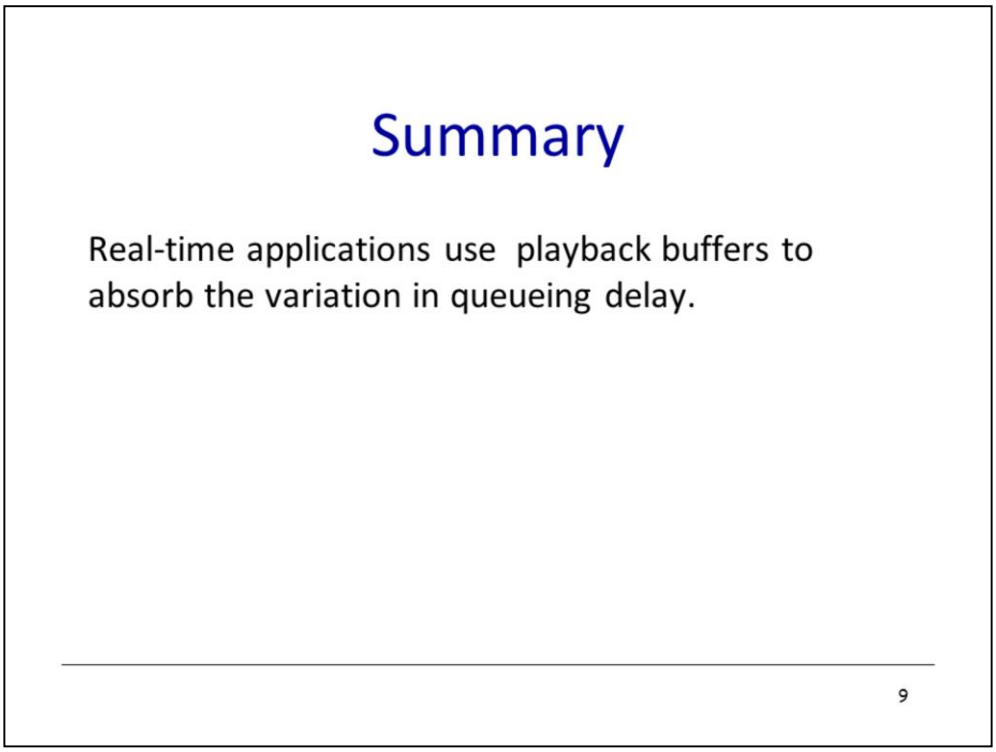




End-to-end delay, $t = \sum_i \left(\frac{p}{r_i} + \frac{l_i}{c} + Q_i(t) \right)$

Queueing adds variable and unpredictable delay

8



<Read from slide>

This is the end of the video on end to end delay. I'll see you again in the next video when I tell you about playback buffers. Packet Switching 3, where I'm going to tell you about a simple deterministic model for understanding variable packet delay.

<end>