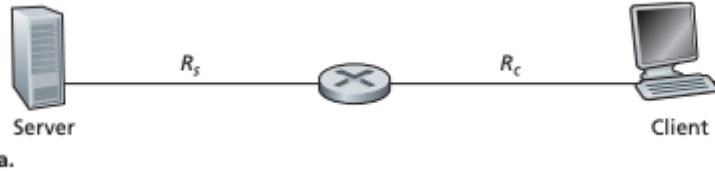


- P23. Consider Figure 1.19(a). Assume that we know the bottleneck link along the path from the server to the client is the first link with rate R_s bits/sec. Suppose we send a pair of packets back to back from the server to the client, and there is no other traffic on this path. Assume each packet of size L bits, and both links have the same propagation delay d_{prop} .
- What is the packet inter-arrival time at the destination? That is, how much time elapses from when the last bit of the first packet arrives until the last bit of the second packet arrives?
 - Now assume that the second link is the bottleneck link (i.e., $R_c < R_s$). Is it possible that the second packet queues at the input queue of the second link? Explain. Now suppose that the server sends the second packet T seconds after sending the first packet. How large must T be to ensure no queuing before the second link? Explain.



a.

a) $R_s < R_c$

$\frac{T + \frac{L}{R_s}}{R_c}$

Pa. Length = L

b) $R_c < R_s$

↳ Yes, the Second Packet can queue @ Time it reaches router. Bc P2 reaches the router Too Soon while Packet 1 still being transmitted on Spws Second link.
Packet 2 finishes arriving @ router b4r the router finishes sending Packet 1

How to insure no queue happens?

↪ packet 2 arrive @ router once
second link is free

$$T + \frac{L}{R_S} + N_{prop} \geq (\frac{L}{R_S} + N_{prop}) + \frac{L}{R_C}$$

for packet 2 for packet 1
Time to free up
router from
packet 1

+ $T \geq \frac{L}{R_C}$

Transmission delay
on second link

Pipelining

- P31. In modern packet-switched networks, including the Internet, the source host segments long, application-layer messages (for example, an image or a music file) into smaller packets and sends the packets into the network. The receiver then reassembles the packets back into the original message. We refer to this process as *message segmentation*. Figure 1.27 illustrates the end-to-end transport of a message with and without message segmentation. Consider a message that is 10^6 bits long that is to be sent from source to destination in Figure 1.27. Suppose each link in the figure is 5 Mbps. Ignore propagation, queuing, and processing delays.
- Consider sending the message from source to destination *without* message segmentation. How long does it take to move the message from the source host to the first packet switch? Keeping in mind that each switch uses store-and-forward packet switching, what is the total time to move the message from source host to destination host?
 - Now suppose that the message is segmented into 100 packets, with each packet being 10,000 bits long. How long does it take to move the first packet from source host to the first switch? When the first packet is being sent from the first switch to the second switch, the second packet is being sent from the source host to the first switch. At what time will the second packet be fully received at the first switch? *Pipelining*
 - How long does it take to move the file from source host to destination host when message segmentation is used? Compare this result with your answer in part (a) and comment.

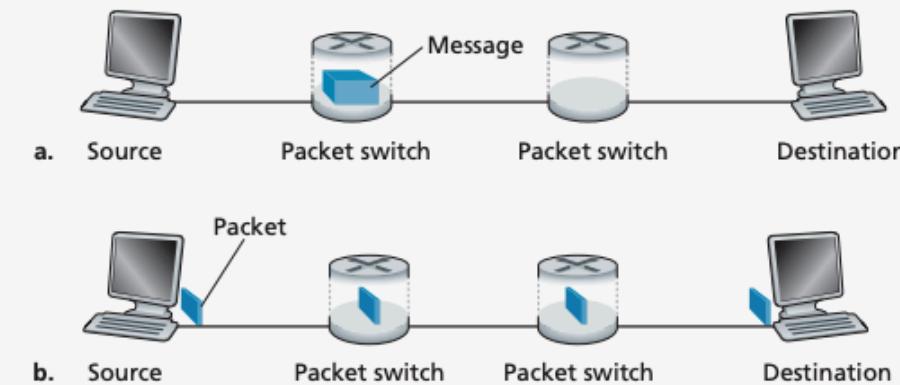


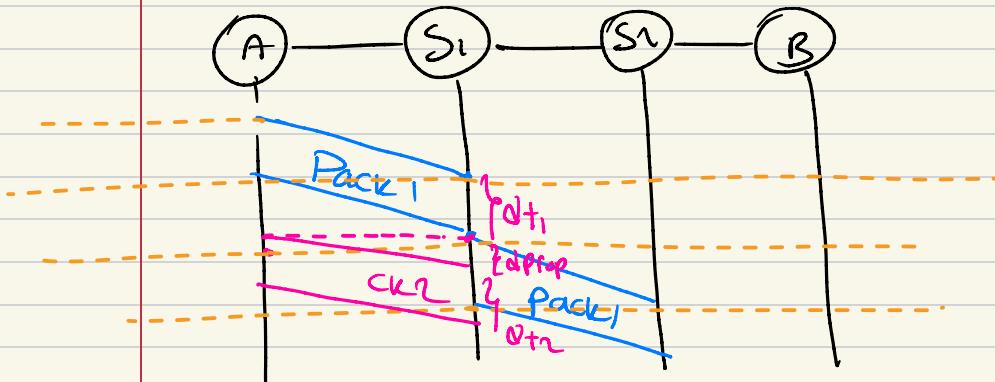
Figure 1.27 End-to-end message transport: (a) without message segmentation; (b) with message segmentation

Length of msg = 10^6 bits

$$R = 5 \times 10^6 \text{ bps}$$

$$a) 3 \times \left(\frac{10^6}{5 \times 10^6} \right) = 0.6 \text{ Second} = 600 \text{ ms}$$

b) Packet Size = 10^4 bits



$$b-a) \frac{L}{R} = \frac{10^4}{5 \times 10^6} = 2 \text{ ms}$$

$$b-b) \quad d_{trans1} + d_{prop2} + d_{trans2} = \underline{4 \text{ms}}$$

$$C) K = \# \text{ of links} \quad N = \# \text{ of packets} \quad \Delta t = \frac{L}{R} \xrightarrow{\text{trans delay}}$$

$K = 3 \quad N = 100 \quad dt = 2 \text{ ms}$

Segmentation + pipeline \Rightarrow (Time to find pipe) + (Time for
removing to
exit)

Fetch & Recv resource from a Server

=> We calculate time for first packet to reach destination because it shows the time it takes to fill the pipeline. Once first packet hits destination \Rightarrow output of pipeline is constant & every 2ms we receive another packet.
 (Parallel / Pipeline flow)

Time for first packet to reach destination:
 (2ms x 3 links)

+

remaining packets arrive @ every 2ms
 (99x2ms)

= 204 ms

- P7. Suppose within your Web browser you click on a link to obtain a Web page. The IP address for the associated URL is not cached in your local host, so a DNS lookup is necessary to obtain the IP address. Suppose that n DNS servers are visited before your host receives the IP address from DNS; the successive visits incur an RTT of RTT_1, \dots, RTT_n . Further suppose that the Web page associated with the link contains exactly one object, consisting of a small amount of HTML text. Let RTT_0 denote the RTT between the local host and the server containing the object. Assuming zero transmission time of the object, how much time elapses from when the client clicks on the link until the client receives the object?

$$\text{Time to Querying IP} = RTT_1 + RTT_2 + \dots + RTT_n$$

(not successful) $\Rightarrow \sum_{i=1}^n RTT_i$ +

$$\text{Time to Querying IP successful} = RTT_0 + (\text{handshake + ACK})$$

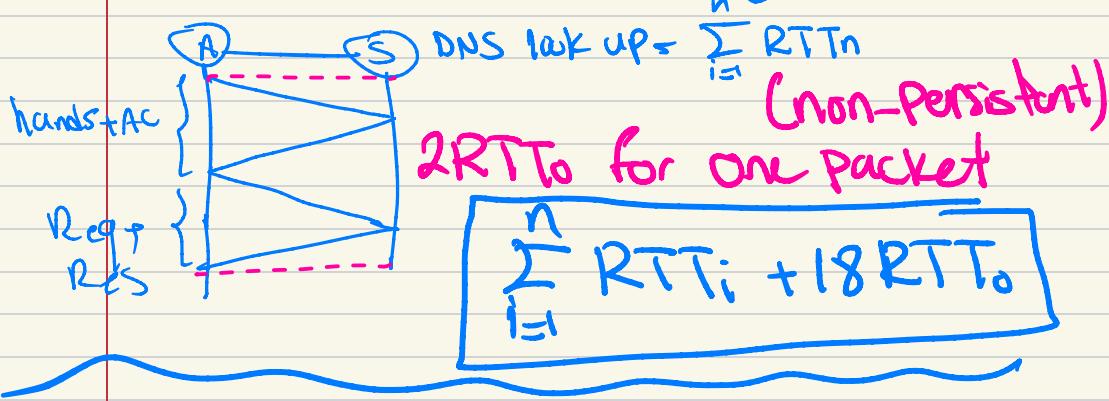
$$\text{Time to Get obj} = RTT_0 + t_{trans}(\text{Req + Res})$$

Answer $\rightarrow \sum_{i=1}^n RTT_i + 2RTT_0$

X Key \Rightarrow HTML References X objects \Rightarrow Objects Fetched after HTML

- P8. Referring to Problem P7, suppose the HTML file references eight very small objects on the same server. Neglecting transmission times, how much time elapses with
- Non-persistent HTTP with no parallel TCP connections?
 - Non-persistent HTTP with the browser configured for 6 parallel connections?
 - Persistent HTTP?

a) 1 HTML file + 8 obj = 9 obj



b) Fetching base HTML & learning 8 obj
non-Persistent = $2RTT_0$ vs
6 Parallel non-Persistent = $2RTT_0$

Config for 6 parallel connections \rightarrow next 2 packets also sent in parallel

.2 Parallel non-consistent = $2RTT_0$

$$\text{Answer} = \boxed{\sum_{i=1}^n RTT_i + 6RTT_0}$$

c) $\sum_{i=1}^n RTT_i + 10RTT_0$

~~Problem Saying Object Size matches Packet size → each obj fits into exactly one pkt~~

- P10. Consider a short, 10-meter link, over which a sender can transmit at a rate of 150 bits/sec in both directions. Suppose that packets containing data are 100,000 bits long, and packets containing only control (e.g., ACK or

handshaking) are 200 bits long. Assume that N parallel connections each get $1/N$ of the link bandwidth. Now consider the HTTP protocol, and suppose that each downloaded object is 100 Kbits long, and that the initial downloaded object contains 10 referenced objects from the same sender. Would parallel downloads via parallel instances of non-persistent HTTP make sense in this case? Now consider persistent HTTP. Do you expect significant gains over the non-persistent case? Justify and explain your answer. \heartsuit

↓
lo
packet
Send
@

Same
time
↓
bandwidth
↓
To

$$R_L = 150 \text{ b/s}$$

$$L_d = 10^5 \text{ bit}$$

$$L_c = 200 \text{ bit}$$

10 Ref Obj

$1/N$ bandwidth for each
packet

↳ downloaded data fit in a single Pkt

↳ Case 1: Single Packet → no parallel

Time to send a packet \rightarrow $d_{trans} = \frac{10^5}{150}$ "np"

Time to send a packet on parallel \rightarrow $d_{trans\ parallel} = \frac{10^5}{150/10} = \frac{10^5}{15}$

↳ $d_{trans\ parallel} > d_{trans\ non\ parallel}$

↑ takes more time to have
Parallel Systems
Doesn't make sense

b) no significant gain because the bottleneck is the link rate while sending data packets in a parallel system which would be smaller as the count of parallel connections goes up

Parallel connections doesn't increase total input

Persistent HTTP saves repeated connections control packets which are so small relative to big amount of data transmitting one 150 b/s instead of

- Req is control packet
- Persistent doesn't work w/ Parallel

~~A~~ ↳ non-persistent parallel System ~~A~~

$$\text{Time} = \underbrace{(\text{Syn-Ack})}_{1 \text{ parallel}} + \underbrace{(\text{10 parallel packets})}_{\frac{1}{15} \text{ bandwidth}} + (\text{1 non-Par}) + (\text{1 Sync-Ack})$$

Time → (One pack to send)

$$\frac{400}{15} + \frac{400}{150} + \frac{100000}{15} + \frac{100000}{150}$$

Time to Send packets non-per parallel

$$\frac{400}{150} + \frac{1100000}{150} = 7362.67 \text{ s}$$

Time to Send packet in persist (no parallel)

$$\frac{400}{150} + 11 \left(\frac{100000}{150} \right) = 7,333.33$$

↑
not much diff

- P27. In this problem, we explore the use of small packets for Voice-over-IP applications. One of the drawbacks of a small packet size is that a large fraction of link bandwidth is consumed by overhead bytes. To this end, suppose that the packet consists of P bytes and 5 bytes of header.
- Consider sending a digitally encoded voice source directly. Suppose the source is encoded at a constant rate of 128 kbps. Assume each packet is entirely filled before the source sends the packet into the network. The time required to fill a packet is the **packetization delay**. In terms of L , determine the packetization delay in milliseconds.
 - Packetization delays greater than 20 msec can cause a noticeable and unpleasant echo. Determine the packetization delay for $L = 1,500$ bytes (roughly corresponding to a maximum-sized Ethernet packet) and for $L = 50$ (corresponding to an ATM packet).
 - Calculate the store-and-forward delay at a single switch for a link rate of $R = 622$ Mbps for $L = 1,500$ bytes, and for $L = 50$ bytes.
 - Comment on the advantages of using a small packet size.

a) Payload Size = $8L$ bit

$$\text{Packetization Delay} = \frac{8L}{128000} \text{ seconds}$$

b) Case1 $\Rightarrow 1500/16 = 93.75$ ms !

Case2 $\Rightarrow 50/16 = 3.125$ ms

good for voice
 (unplisnt echo)

$$d_{trans} = \frac{L}{R} = \frac{\text{Total Payload + header}}{R}$$

$$\text{Packet Size} = 8(L+5) \text{ bits}$$

$$\text{Case 1} \Rightarrow 8(1505) / 622 \times 10^6 =$$

$$1.94 \times 10^{-2} \text{ ms}$$

$$\text{Case 2} = 8(55) / 622 \times 10^6 =$$

$$7.07 \times 10^{-4} \text{ ms}$$

- a) it reduces packetization delay
- + Reduce voice latency
- + Reduce echo + good for real time applications