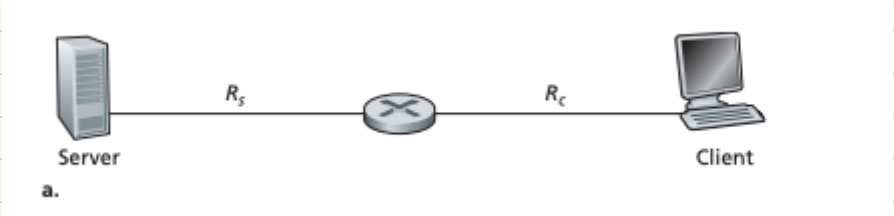
P23. Consider Figure 1.19(a). Assume that we know the bottleneck link along the path from the server to the client is the first link with rate $R_s$ bits/sec. Suppos we send a pair of packets back to back from the server to the client, and there is no other traffic on this path. Assume each packet of size $L$ bits, and both links have the same propagation delay $d_{prop}$.

a. 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?

b. 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) $R_s < R_c$ $\quad T \geq L/R_s$

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 slow second link. Packet 2 finishes arriving @ router before the router finishes sending packet 1

How to insure no queue happens?

↳ packet 2 arrive @ router once second link is free

$$\rightarrow T + L/R_S + d_{prop} \geq \left(L/R_S + d_{prop}\right) + L/R_C$$

$T + L/R_S + d_{prop}$: for packet 2

$L/R_S + d_{prop}$: for packet 1

Time to free up router from packet 1

$$T \geq L/R_C$$

$L/R_C$: 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.

a. 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?

b. 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

b-a)

b-b)

c. 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.

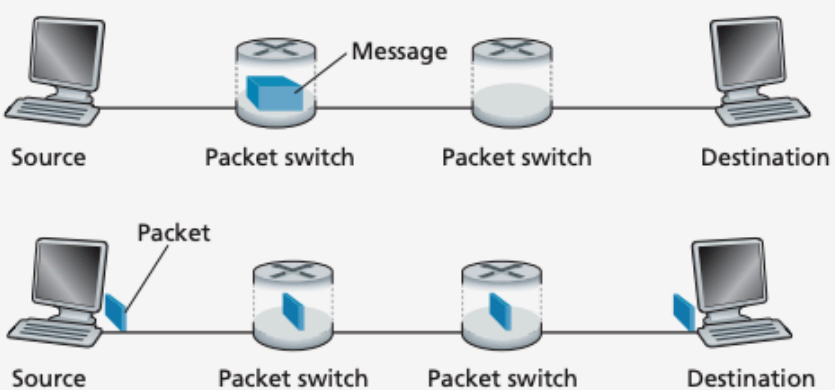


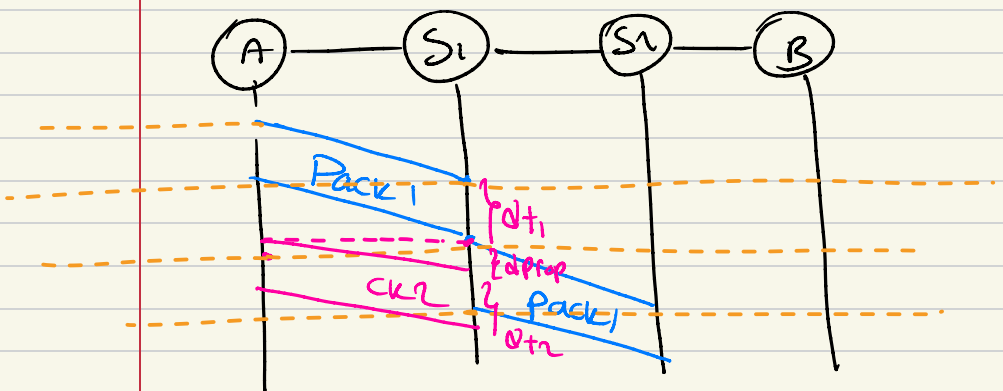

**Figure 1.27** ♦ End-to-end message transport: (a) without message segmentation; (b) with message segmentation

Length of msg = $10^6$ bits

$R = 5 \times 10^6$ bps

a) $3 * (10^6 / 5 \times 10^6) = 0.6$ Second = 600 ms

b) Packet size = $10^4$ bits




b_a) $L/R = 10^4 / 5 \times 10^6 = 2$ ms

b_b) $d_{trans1} + d_{prop2} + d_{trans2} = 4$ ms

C) K = # of links  N = # of packets  $d_t = L/R$ (trans delay)

K = 3  N = 100  $d_t$ = 2 ms

Segmentation + pipeline => (Time to fill pipe) + (Time for remaining to exit)

Fetch = Req + rcv 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 => 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

(99 x 2ms)

= 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 $RTT_1, \ldots, RTT_n$. 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?

Time to Querying IP = $RTT_1 + RTT_2 + \cdots + RTT_n$

(not successful) = $\sum_{i=1}^{n} RTT$

+

Time to Querying IP_succesful = $RTT_0$ (Handshake + ACK)

+

Time to Get obj = $RTT_0 + d_{trans}$ (Req + Res)

answer ⟶ $\sum_{i=1}^{n} RTT_i + 2RTT_0$

Key = HTML References X objects = objects fetched after HTML

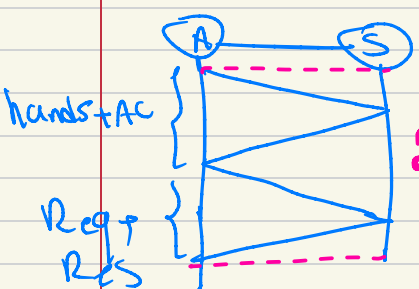
P8. Referring to Problem P7, suppose the HTML file references eight very small objects on the same server. Neglecting transmission times, how much time elapses with

a. Non-persistent HTTP with no parallel TCP connections?

b. Non-persistent HTTP with the browser configured for 6 parallel connections?

c. Persistent HTTP?

a) 1 html file + 8 obj = 9 obj




DNS look up = $\sum_{i=1}^{n} RTT_n$

(non-persistent)

2RTT$_0$ for one packet

$$\sum_{i=1}^{n} RTT_i + 18RTT_0$$

b) Fetching base HTML & learning 8 obj
↳ non-persistent = 2RTT$_0$
· 6 Parallel non-persisting = 2RTT$_0$

Config for 6 parallel connections → next 2 packets also sent in parallel
· 2 Parallel non-consistent = 2RTT$_0$

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

c) $\sum_{i=1}^{n} RTT_i + 10RTT_0$ persistent HTTP w/out pipelining

C) Persistent HTTP with Pipelining

- DNS look up = $\sum_{i=1}^{n} RTT_i$
- Establish TCP = 1 $RTT_0$
- Req/Rec base HTML = 1 $RTT_0$
- Pipeline to get 8 back 2 back Req + responses back = 1 $RTT_0$

$$\sum_{i=1}^{n} RTT_i + 3RTT_0$$

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.

10 packet Send @ same time → bandwidth / 10

$R_L = 150$ b/s

$L_d = 10^5$ bit

$L_c = 200$ bit

10 Ref obj

1/N bandwidth for each packet

↳ downloaded data fit in a single Pkt

↳ Case 1: Single Packet — no parallel "np"

time to send a packet on np → $d_{trans} = \frac{10^5}{150}$

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

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

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 cont of parallel connecti_ goes up

parallel connecti_ doesn't incrse totall input

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

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

↳ non-persistant parallel System

Time = (10 parallel Syn-Ack) + (10 parallel packet) + (1 non-Par)
(1 Sync-Ack)
(1/10 bandwidth)

Time (One packet) to Sent $\frac{400}{15} + \frac{400}{150} + \frac{100000}{15} + \frac{100000}{150}$

Time to Send packets non-per parallel $\frac{4400}{150} + \frac{1100000}{150} = 7362.67\ s$

Time to Send Packet in Persist (no parallel) $(1 * Syn\text{-}Ack) + (10 * Par\ packet) + (1 * Par\ packet)$

not much diff

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

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.

a. 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.

b. 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).

c. 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.

d. Comment on the advantages of using a small packet size.

a) Payload Size = 8L bit

Packtization delay = $\frac{8L}{128000}$ seconds

$\frac{L}{16}$ ms

b) Case1 => $1500/16 = 93.75$ ms ! unpleasant echo

Case2 => $50/16 = 3.125$ ms good for voice

$d_{trans} = L/R = (\text{Total Payload + header})/R$

Packet Size $= 8(L+5)$ bits

Case1 $\Rightarrow$ $8(1505)/622\times10^{6} =$

$$1.94\times10^{-2}\ ms$$

Case2 $= 8(55)/622\times10^{6} =$

$$7.07\times10^{-4}\ ms$$

d) it reduces Packtization delay + Reduce voice latency + Reduce echo + good for real time applications