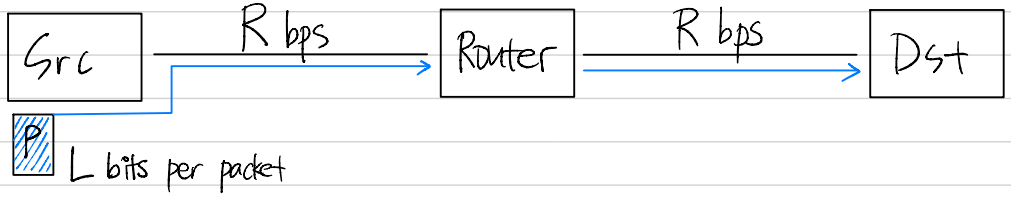
#1 packet switching vs circuit switching

packet switching

host break application-layer messages into packets
via routers & link of path.
packet transmitted at full link capacity

Store-and-forward

entire packet must arrive at router before it can be transmitted on next link

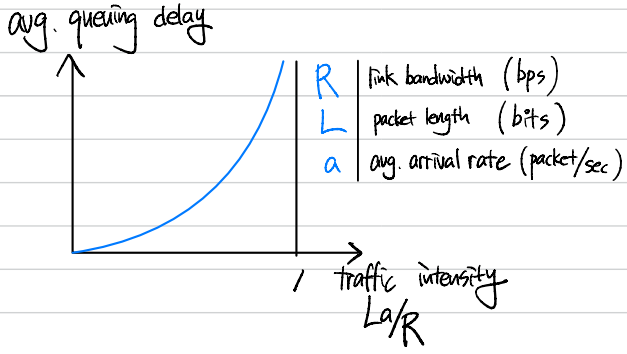


transmission delay = L/R sec
end-end delay = 2 links × L/R = 2L/R (assume no propagation delay)

if R = 7.5 Mbits, L = 1.5 Mbps
transmission delay = 7.5 Mbits / 1.5 Mbps = 5 sec
end-end delay = 2 × 5 = 10 sec

Queuing & Loss

arrival rate to link exceeds transmission rate of link for a period of time
→ queuing: packets will queue. wait to be transmitted on link
→ loss: packets can be dropped (lost) if memory (buffer) fills up.



* packet delay = nodal processing + queuing delay + transmission delay + propagation delay

La/R ≈ 0	little
La/R < 1	acceptible
La/R → 1	large
La/R > 1	infinite

transmission delay = L/R
L packet length (bits)
R link bandwidth (bps)

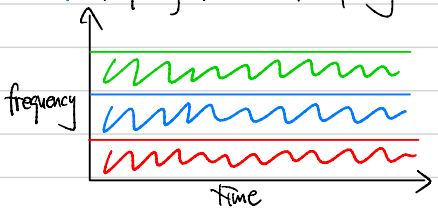
propagation delay = d/s
d length of physical link (m)
s propagation speed (~2×10⁸ m/sec)

#1 packet switching vs Circuit switching

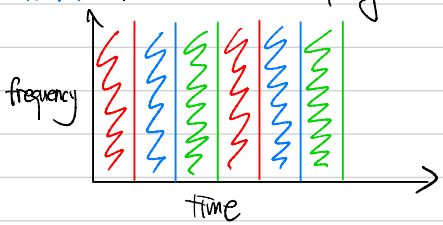
Circuit switching

end-end resources allocated to, reserved for "call" between "source" & "dest"
dedicated resources, No sharing ⇒ circuit-like (guaranteed) performance.
Circuit segment idle if not used by call

FDM : Frequency Division Multiplexing



TDM : Time Division Multiplexing



⇒ if user don't use allocated resource, then it is in idle.

Packet switching vs. Circuit switching

packet switching allows more users than Circuit switching

example) 1 Mbps = 1,000 Kbps
each user : 100 Kbps when active. active probability = 10%

- Circuit switching : 1000 Kbps / 100 Kbps = 10 users
- packet switching : if 35 users exist, probability of active more than 10 of 35 user is less than 0.0004

$$\sum_{i=11}^{35} {}_{35}C_i\,(0.1)^i\,(0.9)^{35-i} = 1 - \sum_{i=0}^{10} {}_{35}C_i\,(0.1)^i\,(0.9)^{35-i}$$

packet switch advantage { great for bursty data
resource sharing : no waste
simpler, no call setup

disadvantage : excessive congestion possible ⇒ delay & loss
sol) protocols needed for RDT, congestion control

Packet switching vs. Circuit switching

	Circuit Switching	Packet Switching
Connection Setting	call setup	No
Resource Allocation	dedicated	shared
Delay	low. consistent	delay / loss congestion
Efficiency	low traffic ⇒ idle	efficient

#2 Wireshark

example

No.	Time	Source (IP)	Destination (IP)	Protocol	Length	Info (port)
1	0.000000	192.168.1.102	128.119.245.12	TCP	62	1161 → 80 [SYN] Seq=0 Win=16384 Len=0 MSS=1460 SACK_PERM
2	0.023172	128.119.245.12	192.168.1.102	TCP	62	80 → 1161 [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1460 SACK_PERM
3	0.023265	192.168.1.102	128.119.245.12	TCP	54	1161 → 80 [ACK] Seq=1 Ack=1 Win=17520 Len=0
4	0.026477	192.168.1.102	128.119.245.12	TCP	619	1161 → 80 [PSH, ACK] Seq=1 Ack=1 Win=17520 Len=565 [TCP PDU reassembled in 199]
5	0.041737	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [PSH, ACK] Seq=566 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
6	0.053937	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=566 Win=6780 Len=0
7	0.054026	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=2026 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
8	0.054690	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=3486 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
9	0.077294	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=2026 Win=8760 Len=0
10	0.077405	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=4946 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
11	0.078157	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=6406 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]
12	0.124085	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=3486 Win=11680 Len=0
13	0.124185	192.168.1.102	128.119.245.12	TCP	1201	1161 → 80 [PSH, ACK] Seq=7866 Ack=1 Win=17520 Len=1147 [TCP PDU reassembled in 199]
14	0.169118	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=4946 Win=14600 Len=0
15	0.217299	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=6406 Win=17520 Len=0
16	0.267802	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=7866 Win=20440 Len=0
17	0.304807	128.119.245.12	192.168.1.102	TCP	60	80 → 1161 [ACK] Seq=1 Ack=9013 Win=23360 Len=0
18	0.305040	192.168.1.102	128.119.245.12	TCP	1514	1161 → 80 [ACK] Seq=9013 Ack=1 Win=17520 Len=1460 [TCP PDU reassembled in 199]

3 way handshake

Which one is ACK packet from server

ACK flag [ACK], ACK #, len=0

Which packet is related to ACK?

consider seq # & data length

ACK # = seq # + data length

Cumulative ACK

repeated ACK #, gradual increment of ACK #

Delayed ACK

received packets > # send ack packet (view of receiver)

Some delay from segment arrival to ACK transfer

Round Trip Time

select base packet.

find ACK packet of base packet

RTT = time(base packet) − time(ACK packet)

Throughput (sender)

data length transmitted / transmission time

len ⇒ byte. time ⇒ sec.

$\Sigma(\text{len}) \;/\; \Delta\text{time} = \text{byte/sec}$

do not consider ACK.

just focus on "send data"

Estimated RTT

$\text{EstimatedRTT}_n = (1-\alpha) \times \text{EstimatedRTT}_{n-1} + \alpha \times \text{SampleRTT}_n$

* $\alpha = 0.125$

$\text{EstimatedRTT}_1 = \text{SampleRTT}_1$

TimeoutInterval

$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$ (DevRTT: safety margin)

Window size from receiver

free space of buffer.

Slow start

back-to-back packets in one RTT.

#7 TCP client & server Communication

example

Sender → receiver		receiver → sender
① seq = 1000	len = 500	ⓐ ACK = 1500
② seq = 1500	len = 500	ⓑ ACK = 1500
③ seq = 2000	len = 500	ⓒ ACK = 1500
④ seq = 2500	len = 500	

② is lost.
make ACK >2000, retransmit ② segment.
if ③, ④ already delivered well, then ACK would be 3000.

Timeout vs. Fast retransmit

	Fast Transmission	Timeout
trigger	duplicate 3 ACK	No ACK response. timer timeout
detection speed	fast	slow.
action	estimated loss segment	last sent segment
feature	fast recovery	last recovery method

#4 RDT events

TCP Reliable Data Transfer

IP ⇒ unreliable solution : pipelined segments, cumulative ACKs, single retransmission timer
retransmission triggered by timer timeout, duplicate ACKs.

TCP sender events

initialize) nextSeqNum = initialSeqNum, sendBase = initialSeqNum

① data received from App ⇒ create segments with seq # (seq#: byte stream number of first data byte in segment), pass segment to IP
start timer if it is not running (timer : oldest unacked segment. timeout interval)

```
create segment with seq# = nextSeqNum
pass segment to IP
nextSeqNum += data length

if (timer not running)
| start timer.
```

② timeout ⇒ retransmit segment that caused timeout, then restart timer.

```
retransmit not-yet-acked segment with smallest seq #
start timer.
```

③ ACK received. ⇒ if ACK acknowledges previously unACKed segments, update what is known to be ACKed.
start timer if there are still unACKed segments.

#4 RDT events

TCP Reliable Data Transfer

IP ⇒ unreliable solution : pipelined segments, cumulative ACKs, single retransmission timer
retransmission triggered by timer timeout, duplicate ACKs.

TCP sender events

① data received from App
② Timeout
③ ACK received. ⇒ if ACK acknowledges previously unACKed segments, update what is known to be ACKed.
Start timer if there are still unACKed segments.

```
y = ACK#,
sendbase = first byte number of oldest unACKed segment

if (y > sendbase) { # received new ACK
    sendbase = y.
    if (not-yet-acked segment exist)
        start timer.
    else
        stop timer.
}
else if(y == sendbase) {    # received duplicate ACK
    if (3-duplicate ACK) {
        fast retransmission.
        start timer.
    }
}
else
    ignore.
```

#4 RDT events

TCP Reliable Data Transfer

IP ⇒ unreliable solution : pipelined segments, cumulative ACKs, single retransmission timer
retransmission triggered by timer timeout, duplicate ACKs.

TCP sender events

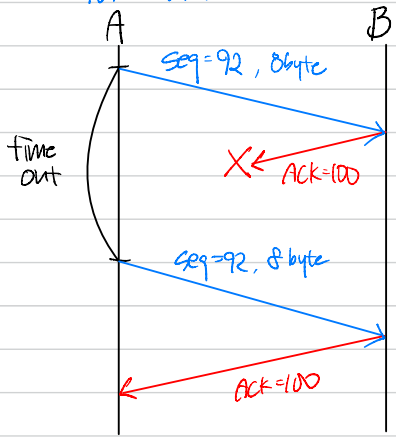
① data received from App ⇒ create segments with seq # (seq#: byte stream number of first data byte in segment), pass segment to IP
start timer if it is not running (timer : oldest unacked segment. timeout interval)

② timeout ⇒ retransmit segment that caused timeout, then restart timer.

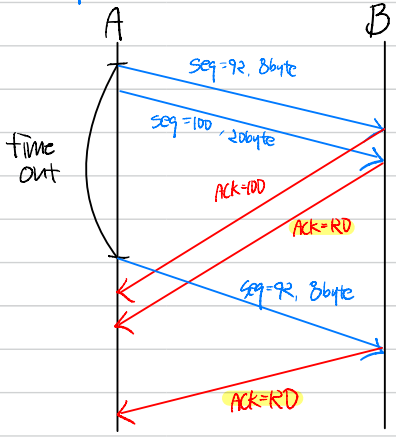
③ ACK received. ⇒ if ACK acknowledges previously unACKed segments, update what is known to be ACKed.
start timer if there are still unACKed segments.

TCP Retransmission Scenarios

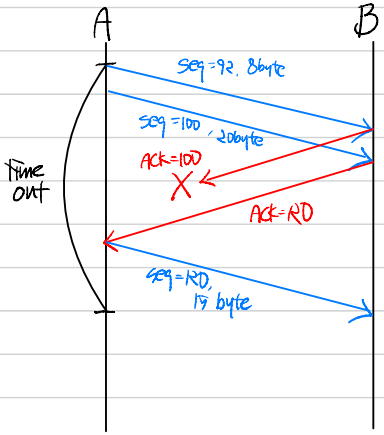
lost ACK



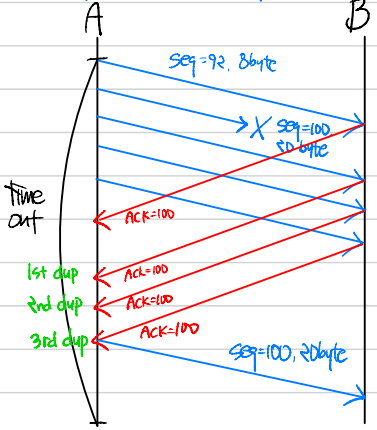
premature timeout



Cumulative ACK



fast retransmit



fast retransmit (y == sendbase)

timeout is relatively long period ⇒ long delay before resending lost packet.
sender sends packet back-to-back ⇒ duplicate Ack if segment lost
receive 3 duplicate ACKs ⇒ resend unACKed smallest seq # segment
don't wait for timeout

#4 RDT events

TCP ACK Generation (receiver)

arrival in-order segment with expected seq # } ⇒ delayed ACK. wait for next segment (reducing ACK overhead)
up to expected seq # already acked. } if no, just send

arrival in-order segment with expected seq # } ⇒ send single cumulative ACK
one other segment pending to ACK }

arrival out-of-order segment higher than expected seq # } ⇒ immediately send duplicate ACK
Gap (loss / delay) detected } (sender quickly detect loss & fast transmission)

arrival of segment that partially or completely fills gap } ⇒ immediately send cumulative ACK

Assignment

bandwidth-delay product
$= R \times d_{propagation}$
⇒ max amount of data on the network circuit at any given time

width of bit $= S/R$, S = propagation speed, R : transmission rate, band-width

Advantage of using message segmentation.

1. efficient use of network resources
2. Error isolation & recovery
3. Improved congestion control
4. Pipelining & Throughput

0.001 sec = 1ms
1Mb = 1000 Kb
1Kb = 1000 b