CEN445 – Network Protocols and Algorithms

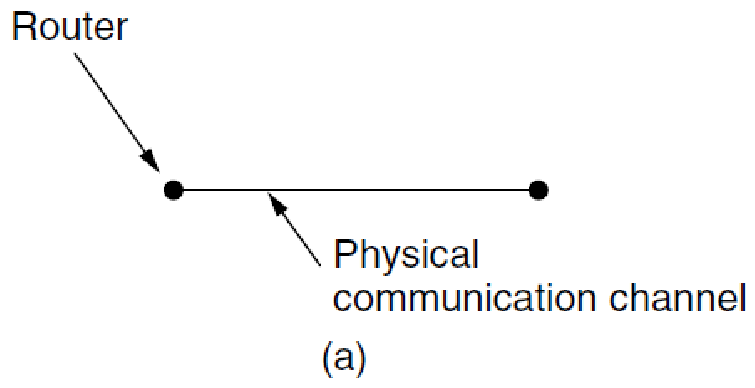
Chapter 6 – Transport Layer

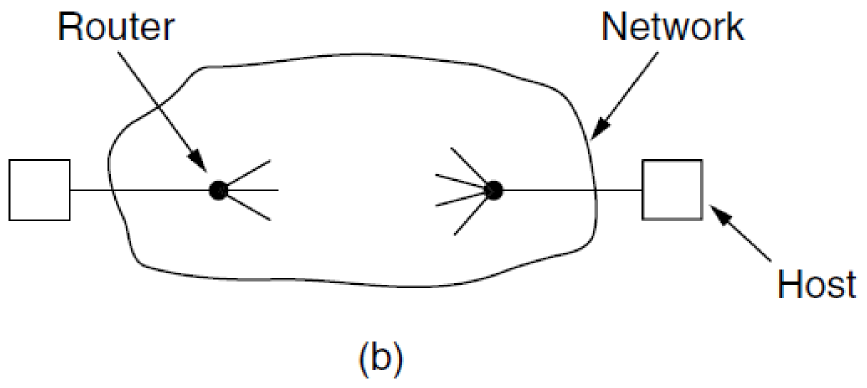
6.2 Elements of Transport Protocols

Dr. Mostafa Hassan Dahshan
Department of Computer Engineering
College of Computer and Information Sciences
King Saud University
mdahshan@ksu.edu.sa
http://faculty.ksu.edu.sa/mdahshan

Elements of Transport Protocols

- Transport protocol similar to data link protocols
- Both do error control and flow control
- However, significant differences exist





Environment of the data link layer

Environment of the transport layer

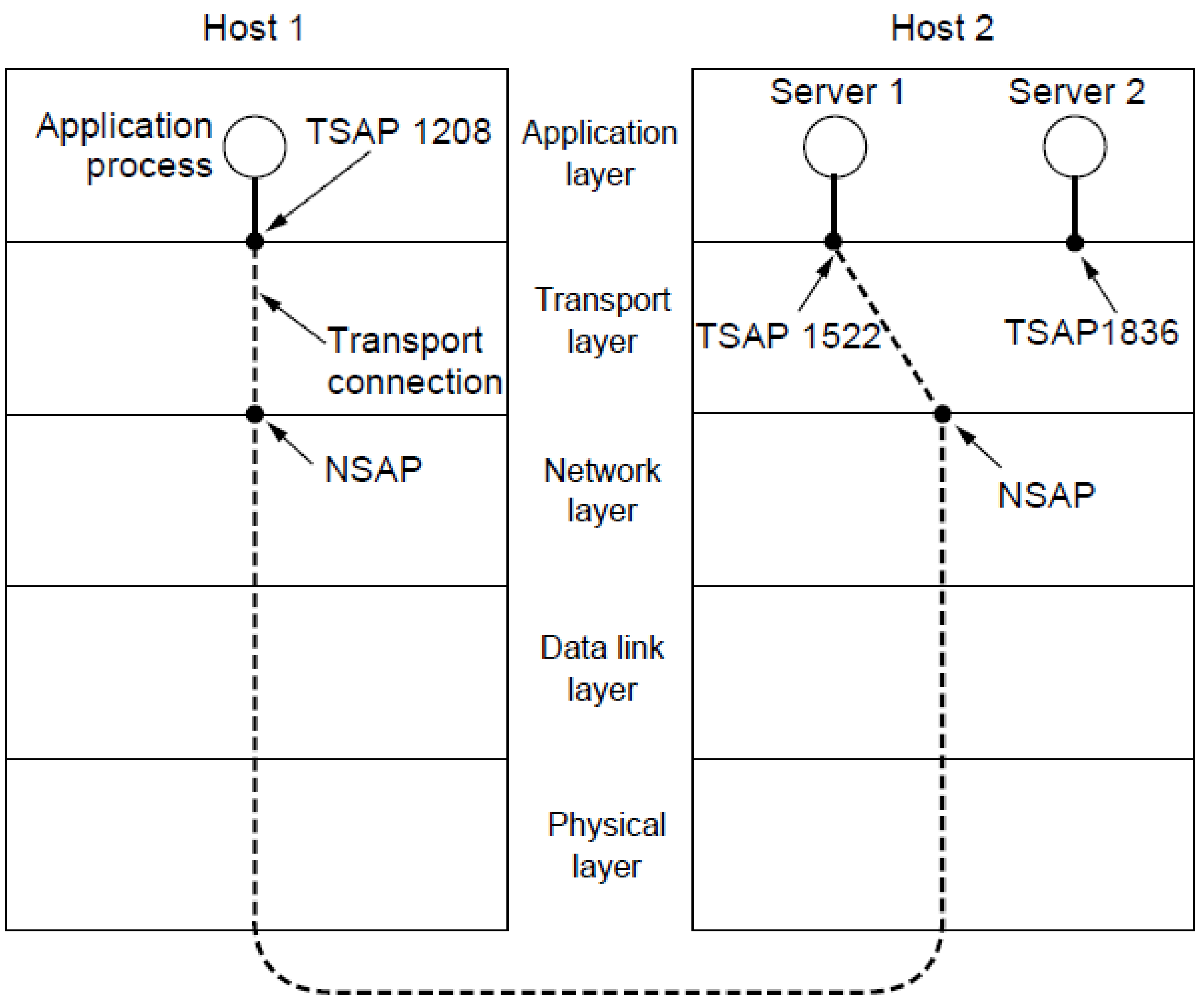
Elements of Transport Protocols

	Data link layer	Transport layer
Communication	directly via physical channel	over the entire network
Addressing	no need to specify address. Just select outgoing line	explicit addressing of destination is required
Connection establishment	over a wire is simple	more complicated
Delay	frame either arrives or lost, not stored and delayed	packets might be stored for seconds and delivered later
Buffering and flow control	simpler	more complicated; large and dynamic number of simultaneous connections

Addressing

- Specify which host process to connect to
- TSAP: Transport Service Access Point
- In TCP, UDP, called **ports**
- Analogy: NSAP. Example: IP address
- Client or server app attaches to TSAP
- Connections run through NSAP
- TSAP to distinguish endpoints sharing NSAP

Addressing


Host 1
Host 2
Application process
TSAP 1208
Server 1
Server 2
Application layer
Transport connection
Transport layer
TSAP 1522
TSAP1836
NSAP
Network layer
NSAP
Data link layer
Physical layer

Addressing

How host1 process knows TSAP # at host2?

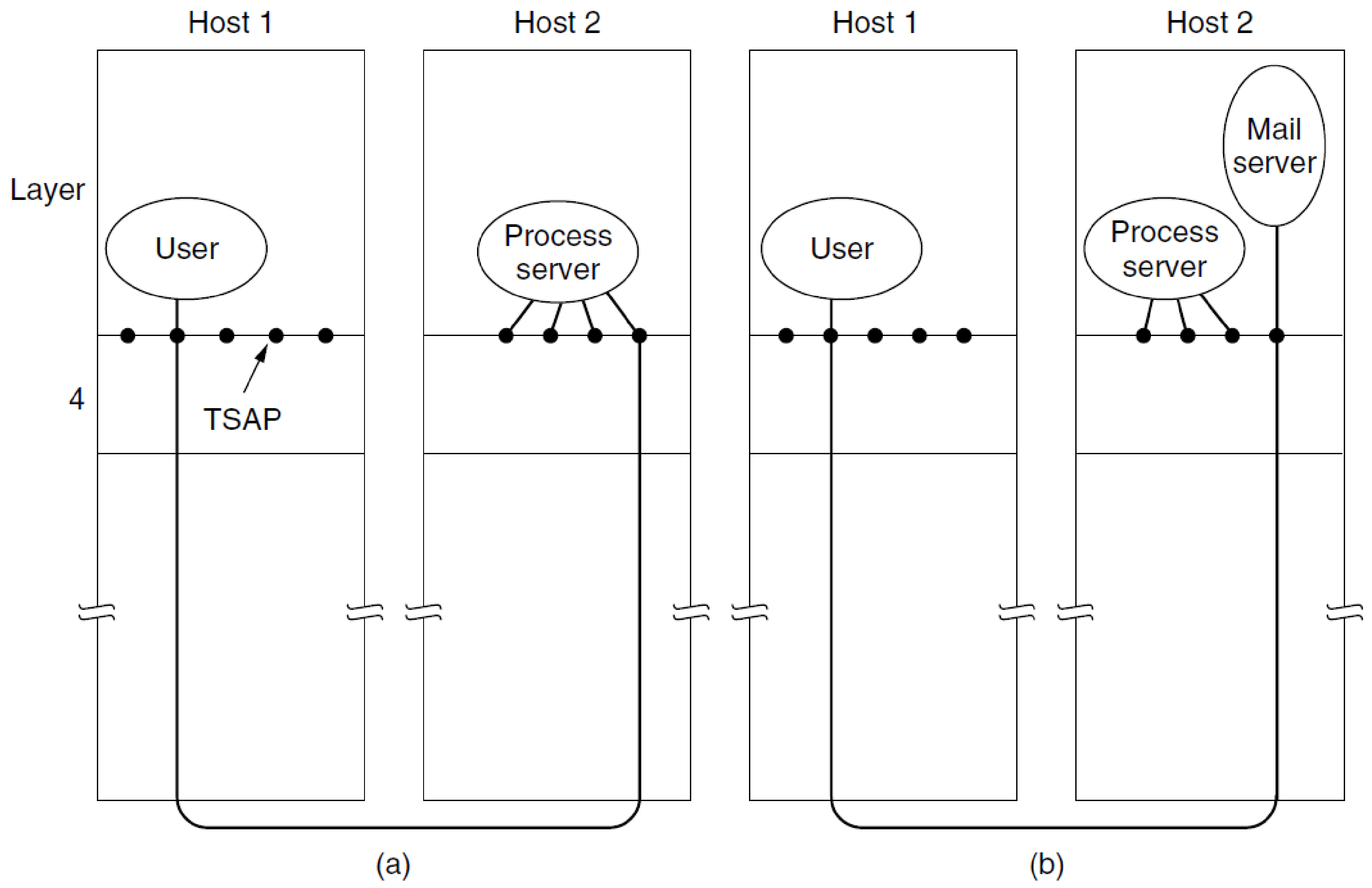
- Stable TSAP # listed in well-known places
  - e.g. /etc/services in UNIX list permanent #s
  - mail server: 25, web server: 80
- Alternatively, special process: **portmapper**
  - listen on well-known TSAP
  - user conn to portmapper, specify service name
  - portmapper sends back TSAP address
  - similar to directory service (دليل الهاتف)

Addressing

- Many server processes used only rarely
- Waste if each process listen to TSAP all day
- Instead, use initial connection protocol
- Spec process server listens all known TSAP
- Act as proxy for lightly used servers
- e.g. inted, xinetd on UNIX

Addressing

User does CONNECT request   Process server spawns request server



Connection Establishment

- Sounds easy; surprisingly tricky!
- Just send REQUEST, wait for ACCEPTED?
- Can lose, delay, corrupt, duplicate packets
- Duplicate may transfer bank money again!
- Protocols must work correct all cases
- Implemented efficiently in common cases
- Main problem is delayed duplicates
- Cannot prevent; must deal with (reject)

Connection Establishment

Solutions for delayed duplicates

- Not reuse transport address (TSAP)
  - problem — difficult to connect to process
- Give each connection unique ID
  - seq # chosen by initiating party
  - update table listing obsolete connections
  - check new connections against table
  - problems — requires maintain certain amount of history
  - if machine crashes, no longer identify old con

Connection Establishment

- To simplify problem, restrict packet lifetime
  - restricted network design: prevent looping
  - mostly used hop counter in each packet: -1 at each hop
  - timestamp in each packet: clock must be synced
- Must also guarantee ACKs are dead
- Assume a value T of max packet lifetime
- *T* sec after packet sent, sure traces are gone
- In the Internet, T is usually 120 seconds

Connection Establishment

New method with packet lifetime bounded

- Label segments with seq # not reused in T
- T and packet rate determine size of seq #s
- 1 packet w given seq # may be outstanding
- Duplicates may still occur, but discarded dst
- Not possible to have delayed duplicate old packet with same seq # accepted at dest

Connection Establishment

How to deal with losing memory after crash?

- Each host has time-of-day clock
  - clocks at different host need not be synced
  - binary counter increments uniform intervals
  - no. of bits must be ≥ of seq #
  - clock must be running even if host goes down
- Initial seq # (k-bits) ← low k-bits of clock
- Seq space must be so large
  - by time # wrap, old pkts w same # are long gone

Connection Establishment

Key problem is to ensure reliability even though packets may be lost, corrupted, <u>delayed</u>, and <u>duplicated</u>

- Don’t treat an old or duplicate packet as new
- (Use ARQ and checksums for loss/corruption)

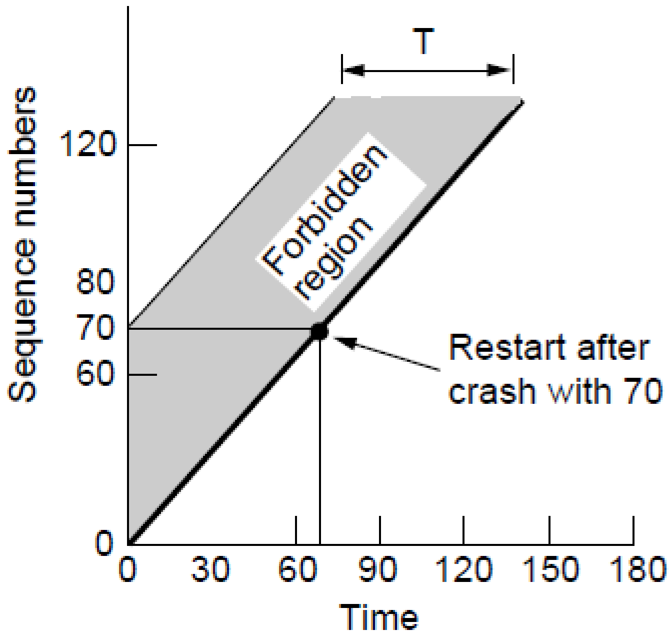
Approach:

- Don’t reuse sequence numbers within twice the MSL (Maximum Segment Lifetime) of 2T=240 secs
- Three-way handshake for establishing connection

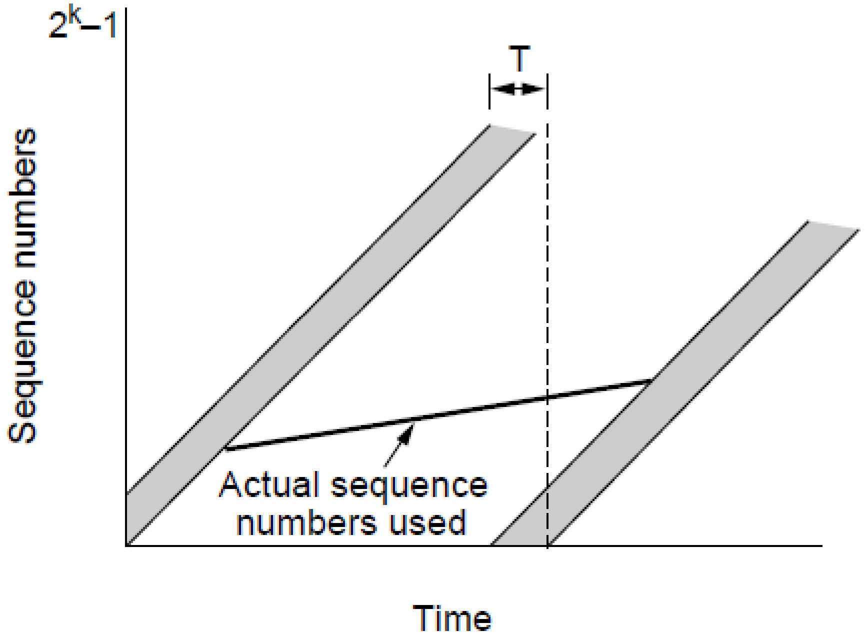
Connection Establishment

Use a sequence number space large enough that it will not wrap, even when sending at full rate

- Clock (high bits) advances & keeps state over crash



Need seq. number not to wrap within T seconds



Need seq. number not to climb too slowly for too long

Example

Suppose that the clock-driven scheme for generating initial sequence numbers is used with a 15-bit wide clock counter. The clock ticks once every 100 msec, and the maximum packet lifetime is 60 sec.

How often need resynchronization take place

(a) in the worst case?

(b) when the data consumes 240 sequence numbers/min?

Solution

Total number of ticks before the clock cycles around is $2^{15} = 32768$ ticks

The time elapsed for the clock to cycle around $= 32768 \times 0.100 = 3276.8$ sec

Max packet lifetime $T = 60$ seconds

(a) The worst case happens when no packets are generated (zero generation rate)

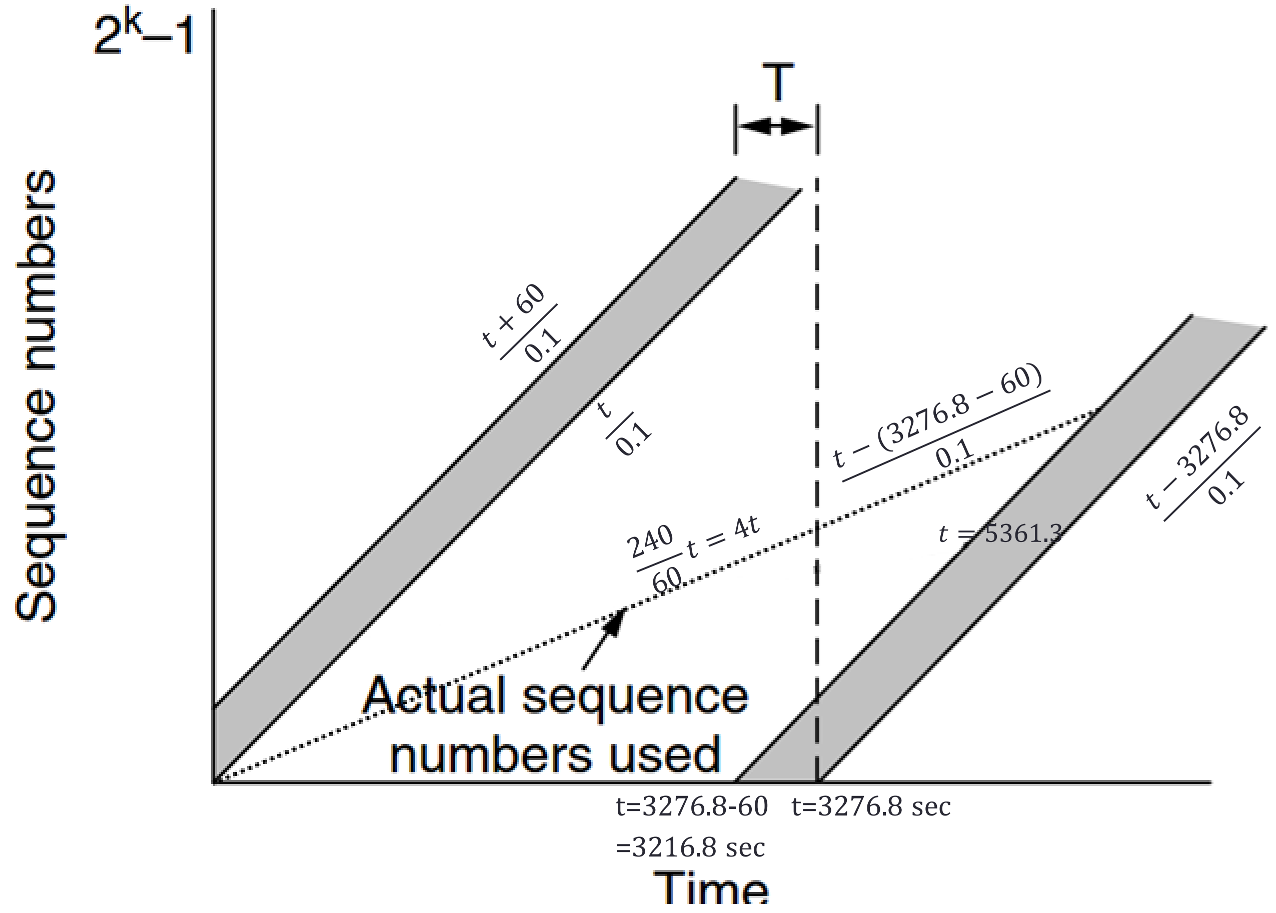
In this case, the sender would enter the forbidden zone at $3276.8 - 60 = 3216.8$ sec

Solution

(b) b. At 240 sequence numbers/min, the actual sequence number is $4t$, where t is the time in sec.

The left edge of the forbidden region is $\frac{1}{0.100}(t - 3216.8)$. Equating these two formulas, we find that they intersect at $t = 5361.3$ sec.

Solution



2^k-1
T
Sequence numbers
$\frac{t+60}{0.1}$
$\frac{t}{0.1}$
$\frac{t-(3276.8-60)}{0.1}$
$\frac{t-3276.8}{0.1}$
$\frac{240}{60}t = 4t$
$t = 5361.3$
Actual sequence numbers used
t=3276.8-60 =3216.8 sec
t=3276.8 sec
Time

Connection Establishment

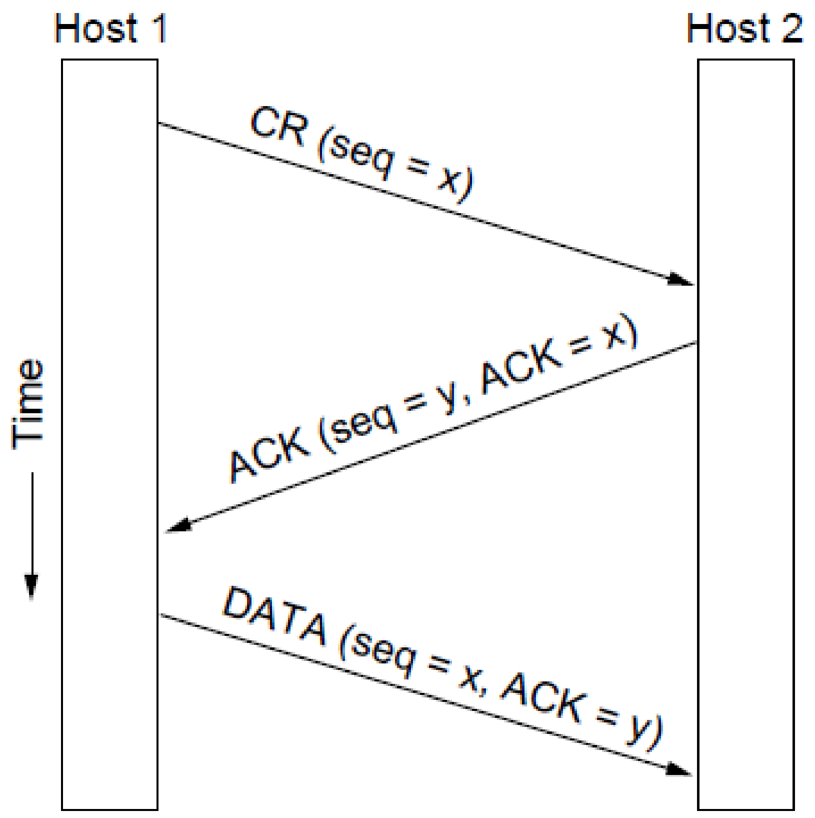
- Clock method work within connection
- Host don’t remember # across connections
- Can’t know if CONN REQUEST with initial seq # is a duplicate of a recent connection
- To solve this, use three-way handshake
- Check with other peer that con req is current
- Used in TCP, with 32-bit seq #
- Clock not used in TCP; attacker can predict

Connection Establishment

Normal Procedure

- H1 choses initial s# *x*
- H2 replies
  - ACKs *x*
  - announce own s# y
- H1replies
  - ACKs *y*
  - with 1st data segment

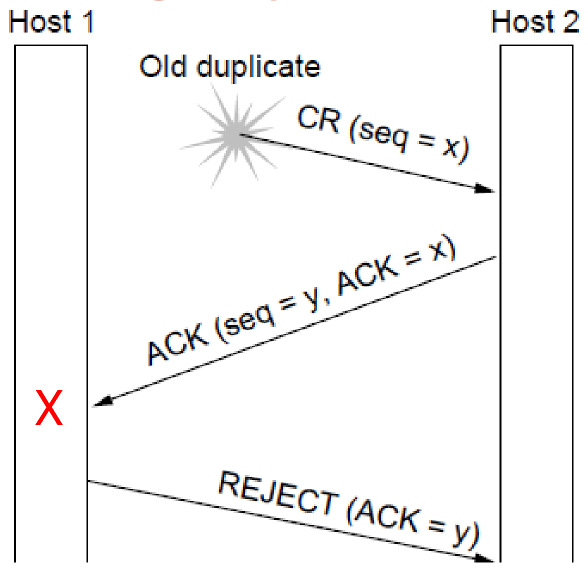
CR = Connection Request

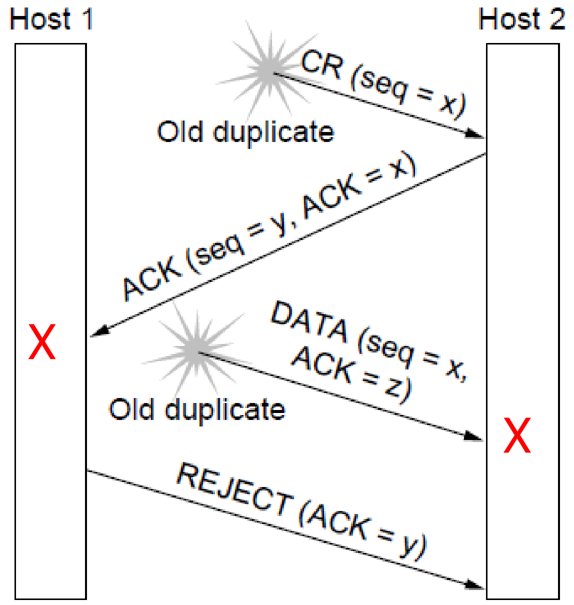


Connection Establishment

Abnormal situations

- Delayed duplicate CR
  - H2 sends ACK to H1
  - H1 rejects
  - H2 knows it was tricked
- Worst case
  DD CR, old ACK floating
  - H2 gets delayed CR, replies
  - H1 rejects
  - H2 gets old DATA, discards
    (z received instead of y)

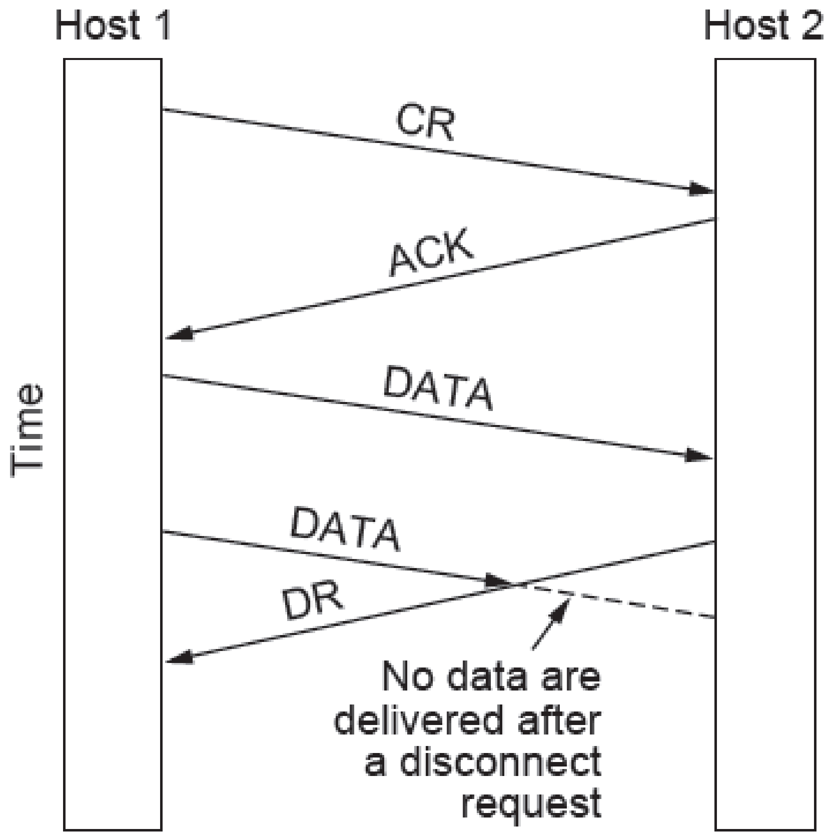




Connection Release

- Easier than establish
- However, some pitfalls

- Asymmetric release
  - the way telephone works
  - 1 party hangs up, con broken
  - abrupt; may cause data loss
  - better protocol needed



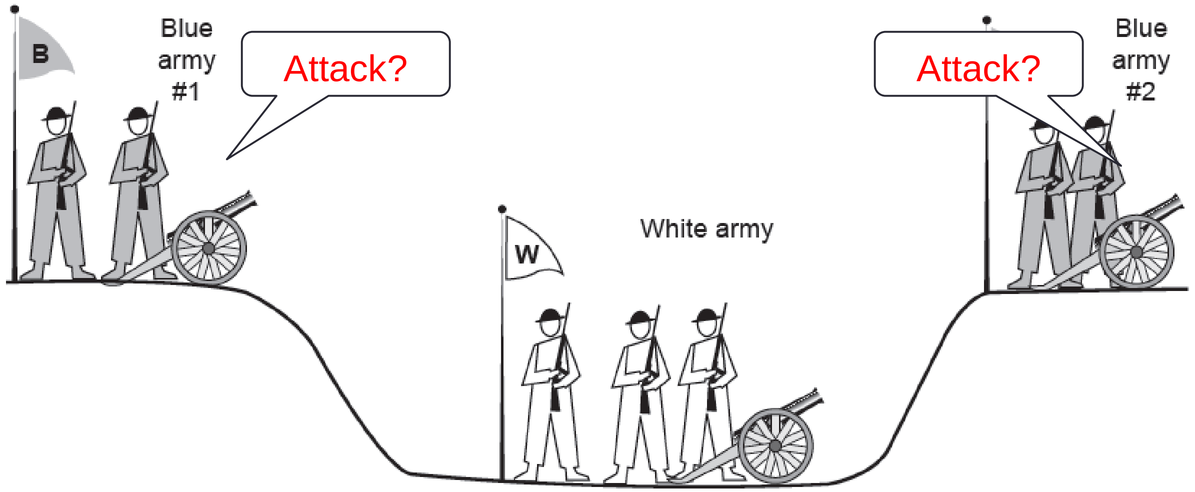
Connection Release

Symmetric release

- Each direction is released independently
- Can receive data after sending DISCONNECT
- H1: I am done, are you done too?
- H2: I am done too, goodbye
- Two-army problem: unreliable channel

Connection Release

- Two-army problem
  - each blue army < white army, but together are larger
  - need to sync attack
  - however, only com channel is the valley (unreliable)
  - 3-way handshake? B1 can't know ACK arrived
  - making 4-way handshake doesn't help either

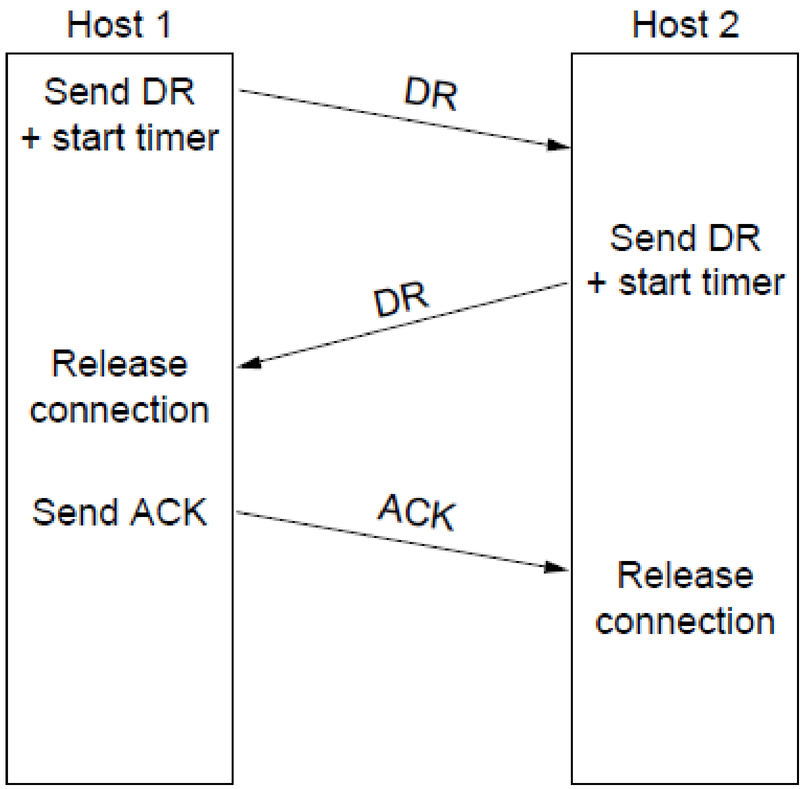


Connection Release

- Let each side independently decide its done
- Easier to solve

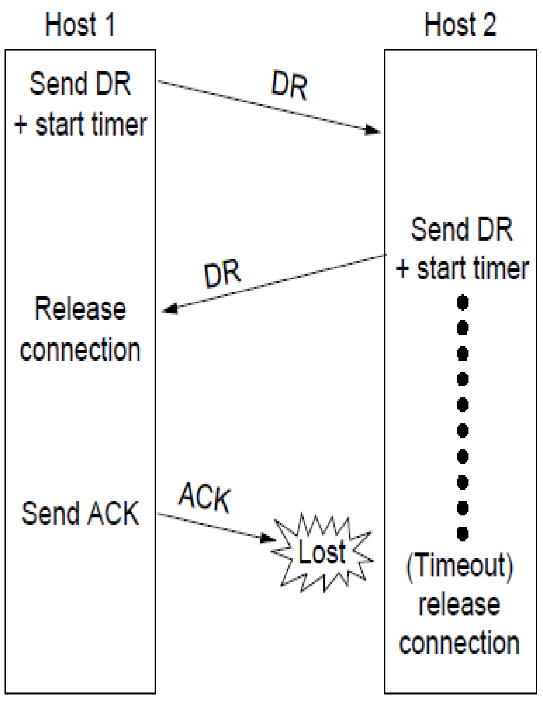
Normal release sequence

- H1 send DR, start timer
- H2 responds with DR
- when H1 recv DR, release
- when H2 recv ACK, release

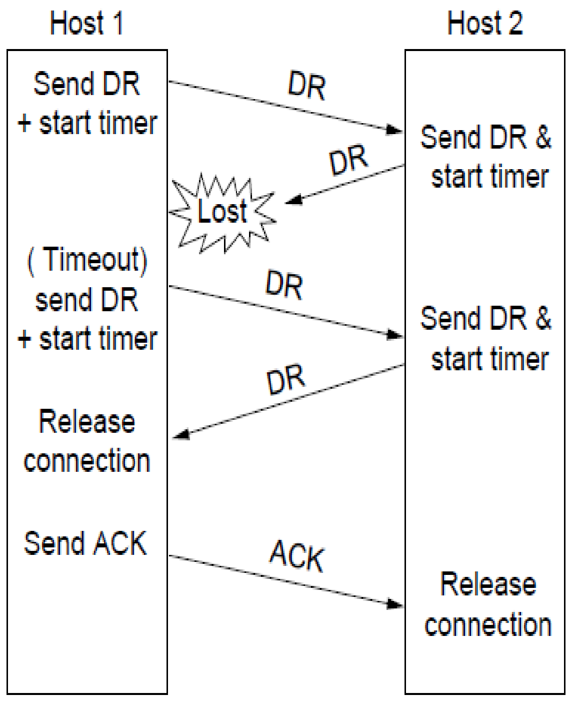


Connection Release

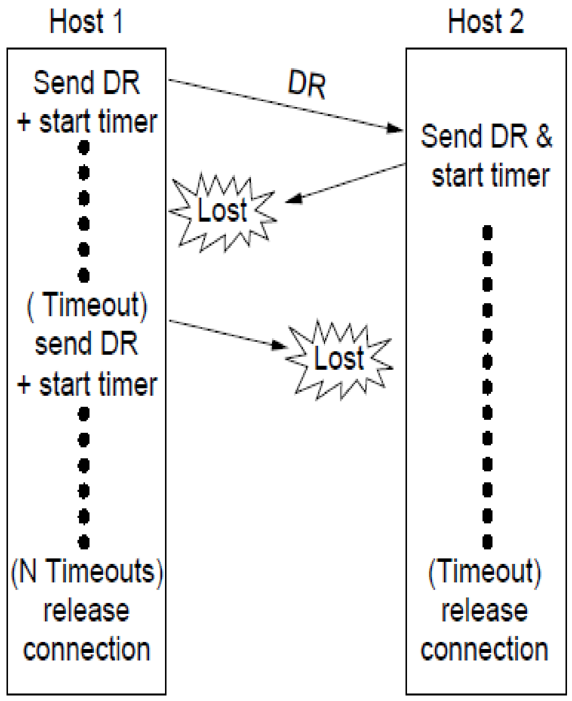
Error cases, handled by timers, retransmissions



Final ACK lost:
Host 2 times out



Lost DR:
H1 starts over



Host 1
Host 2
Send DR + start timer
DR
Send DR & start timer
Lost
(Timeout) send DR + start timer
Lost
(N Timeouts) release connection
(Timeout) release connection

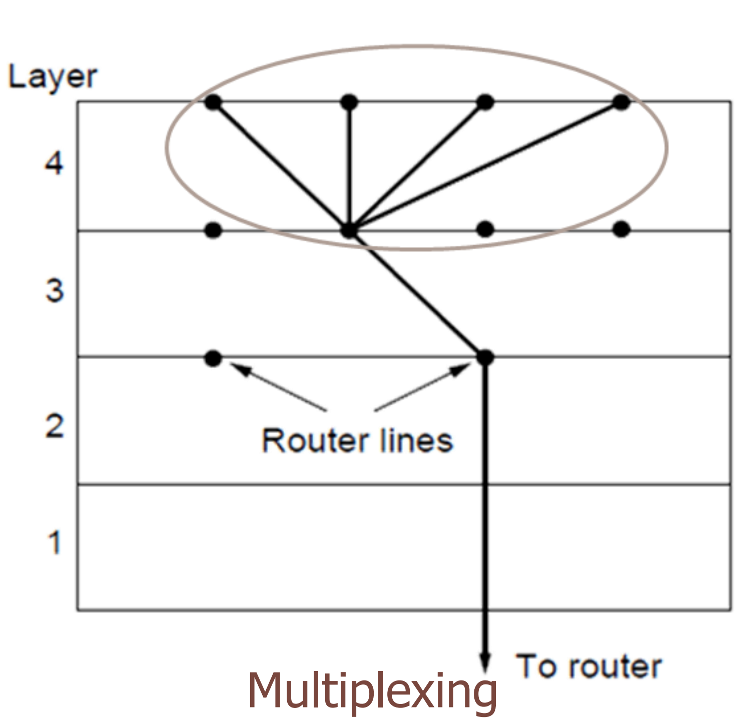
Extreme:
Many lost DRs
both release after N

Connection Release

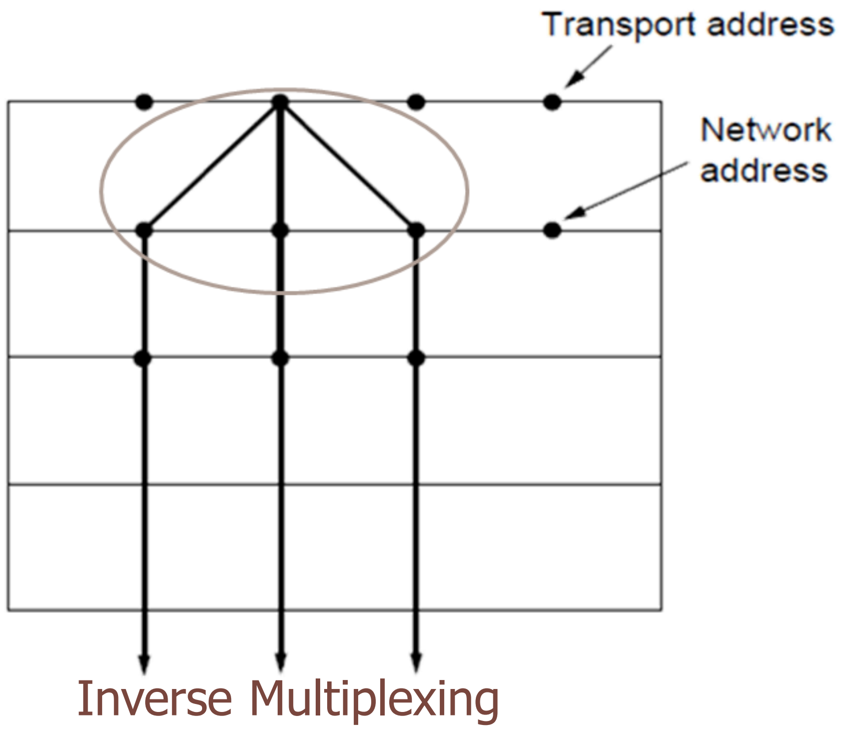
- Protocol usually suffices; can fail in theory
- after N lost attempts; half open connection
- Not allowing give up, can go on forever
- To kill half open connections, automatically disconnect if no received segments in X sec
- Must have timer reset after each segment
- Send dummy segments to keep con alive
- TCP normally does symmetric close, with each side independently close ½ con w FIN

Multiplexing

- Transport, network sharing can either be:
  - Multiplexing: connections share a network address
  - Inverse multiplexing: addresses share a connection



Multiplexing



Inverse Multiplexing