① 14-10-14

(p.1)

TRANSPORT LAYER

"....not just another layer...
—.... heart of the whole protocol hierarchy...."

→ to provide reliable, cost-effective data transport

6.1 The Transport Service ["end-to-end"]

users → processes in the application layer
i.e. user processes

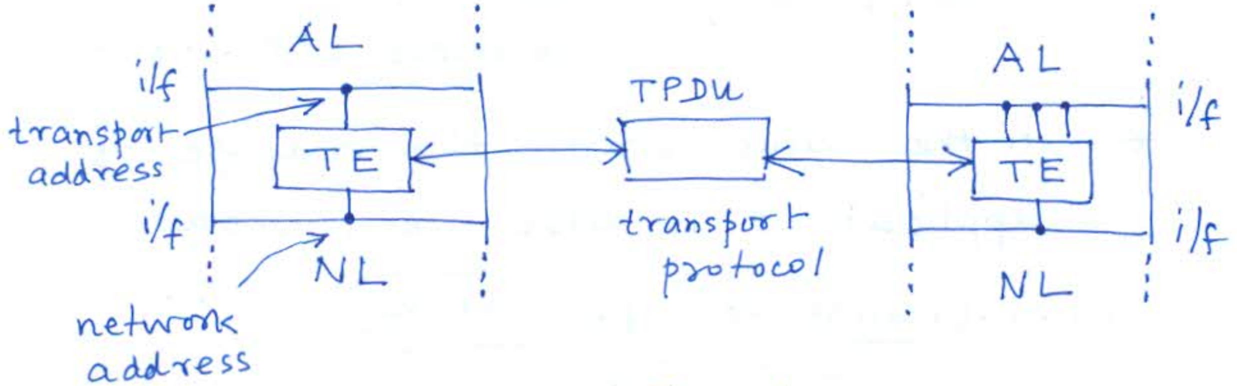


Fig 6-1

service – connection oriented
connectionless

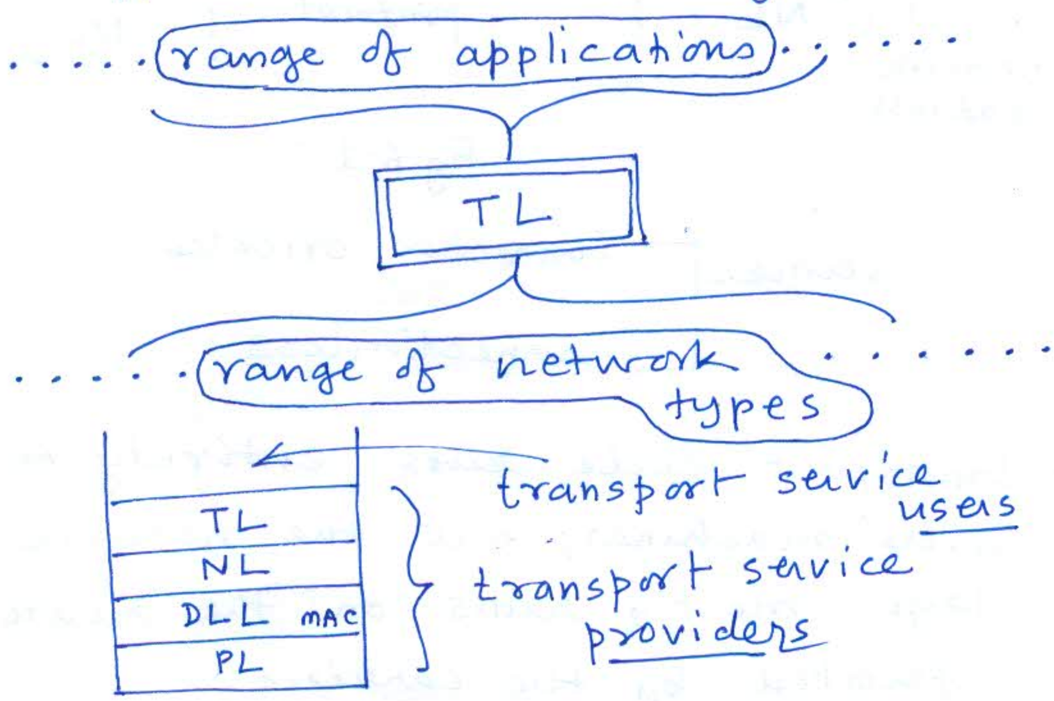
* transport code runs entirely on users' machines, but the network layer mostly runs on the routers operated by the carriers

(p.2) ... users have no real control over the network layer ....

* transport layer compensates for the shortcomings of the NL, to provide better QoS [ref. previous chapter]

* also makes it possible to change underlying NL
  dial-up line / wired LAN / cable wireless LAN .....

* in the same way, the range of applications is also very broad



3

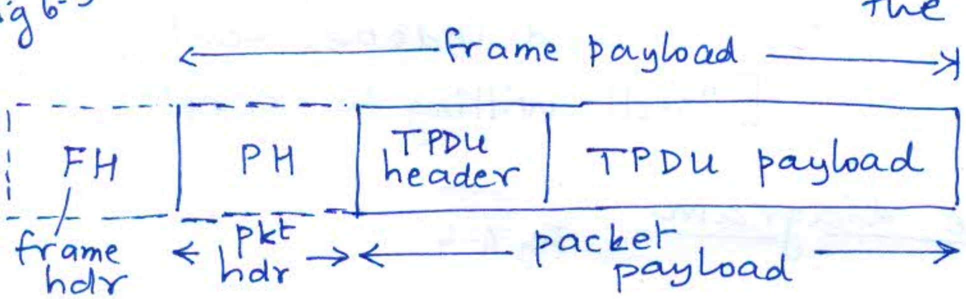
14-10-14

P.3

Primitives for a simple connection-oriented transport service

Primitive	Pkts sent	Meaning
LISTEN	none	block until a process tries to connect
CONNECT	connection request	actively attempt to establish a connection
SEND	data	send data (what else?)
RECEIVE	none	block until a data packet arrives
DISCONNECT	disconnection request	this side is trying to disconnect the connection

Fig 6-3



17-10-14

(P.1) "at TL, even a simple unidirectional data exchange is more complicated than at NL"

- every data packet is acknowledged
- control TPDU packets also are acknowledged (implicitly or explicitly)
- details "not visible" to user processes
- includes things such as timers, retransmissions, ... etc.

to the users → a "magical", reliable bit pipe [byte pipe?]

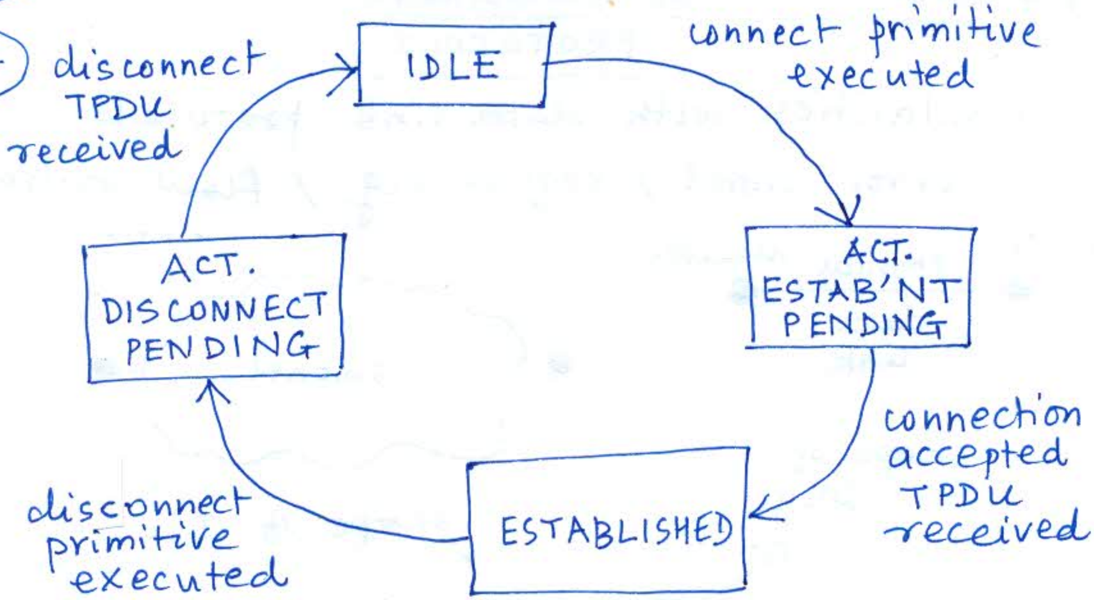
DISCONNECT

- asymmetric — transport user at either end of connection issues the request
- symmetric - each direction is closed separately and independently ["still willing to accept....."]

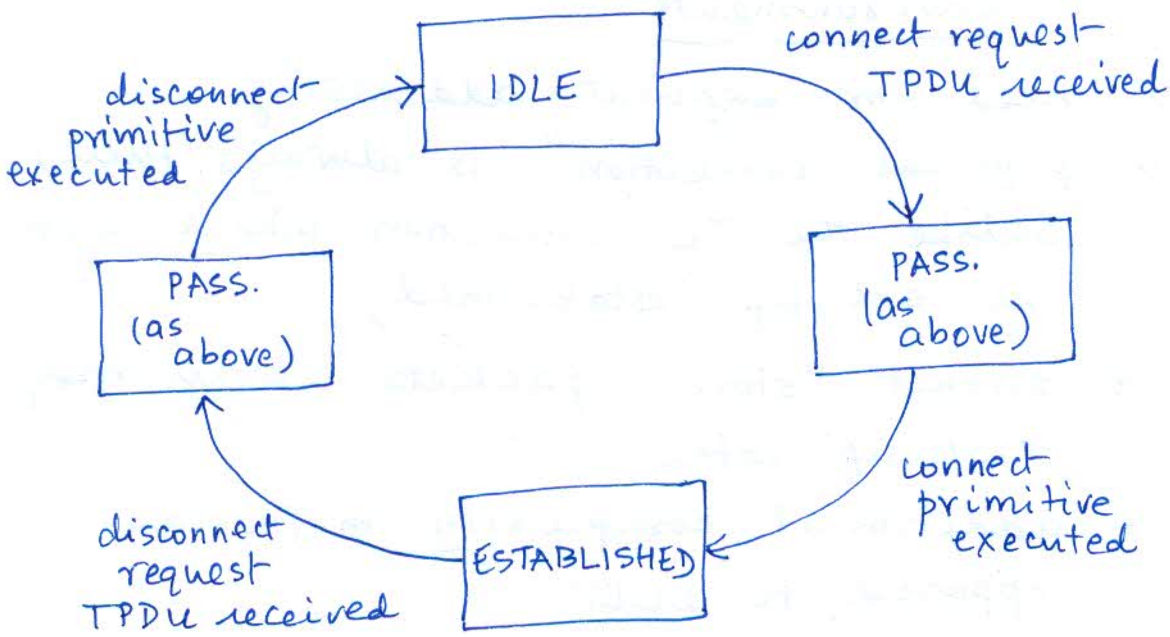
* State diagram Fig 6-4

17-10-14

P.2



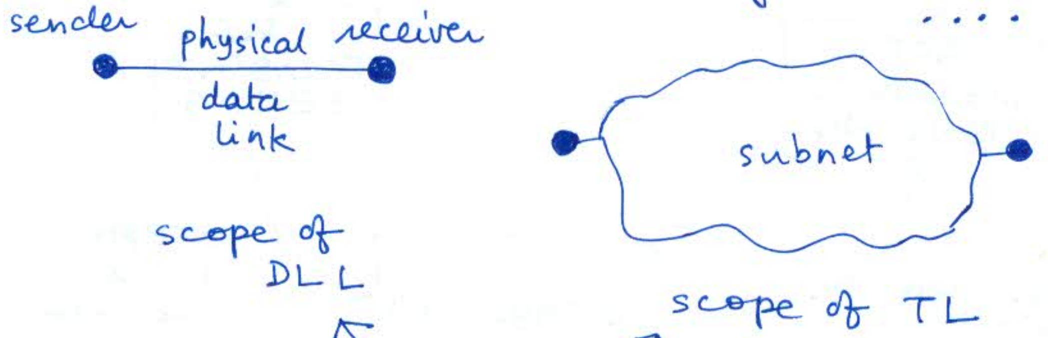
"CLIENT"



"SERVER"

6.2 ELEMENTS OF TRANSPORT PROTOCOLS

→ similarities with data link protocols
error control / sequencing / flow control ....



significant differences arise in the two protocols because of the difference in their respective "environments"

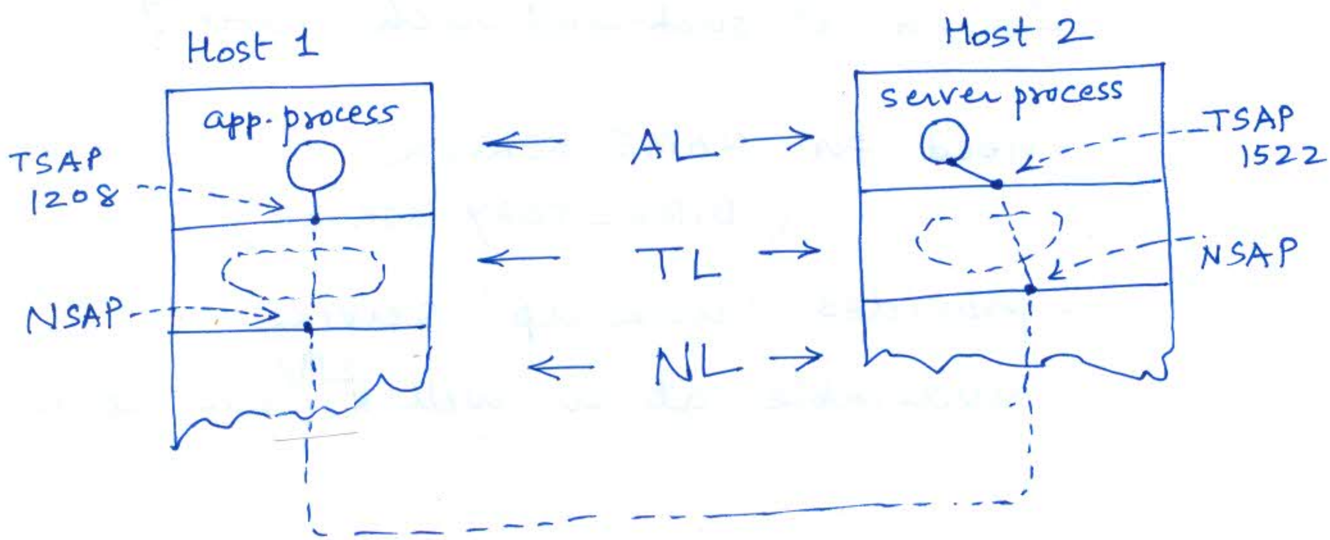
* need for explicit addressing
* physical "connection" is always there, unlike the TL connection which must be set up (established)
* subnet "stores" packets which may "pop up" later
* additional complexity in TL as opposed to DLL

[use of "tables"]

17-10-14 (p.4)

Addressing

Fig 6-8



multiple transport endpoints may share an NSAP

Example

1. A ToD server on H-2 attaches itself to TSAP 1522 to await an incoming "call", e.g. by executing LISTEN
2. App. process on H-1 issues CONNECT with TSAP 1208 as source and TSAP 1522 destn
3. } request for ToD sent] as many times as needed
4. } server responds
5. Transport connection is "released" ["table entries cleaned up"]

p.5

How does the user process know that such-and-such server is to be found at such-and-such TSAP?

need for NAME SERVER
(DIRECTORY SERVER)

- provides "look up" service
- available at a well-known address

9

31-10-14

p.1

Elements of Transport Protocols (continued)

6.2.2 Connection Establishment

* should require an exchange of TPDUs between two "transport entities"

→ CONNECTION REQUEST

← CONNECTION ACCEPTED

* simple ... but

→ the network can lose, store and duplicate packets

[recall: delays in congested network can cause timeouts + retransmissions. Transport layer has little or no direct control over these processes.]

somewhat imaginary + extreme example: an entire bank transaction executed twice!

* problem - delayed duplicates

possible solutions -

- throw-away transport addresses
- connection identifiers along with some recent history in TEs [lost if m/c crashes .... ?]

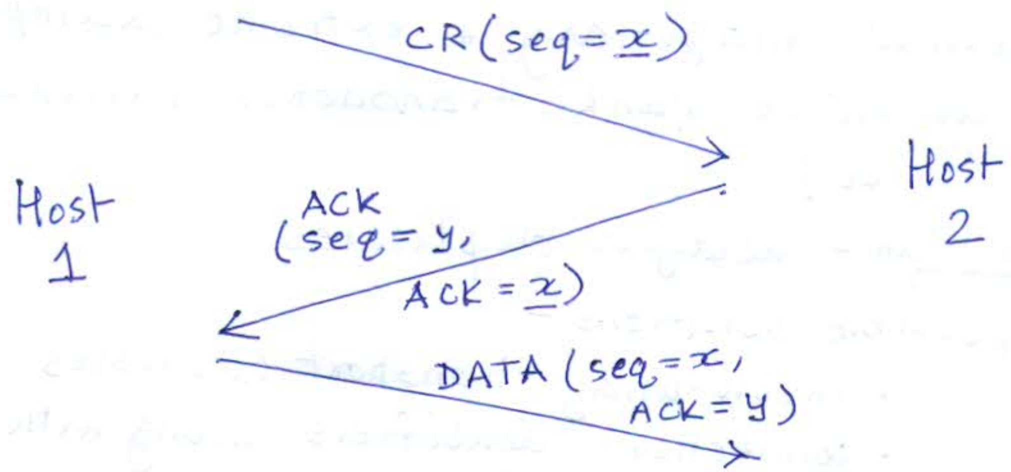
p.2

- limiting packet lifetimes
  - → subnet design
    hop count
    time stamps
    [need synchronization]
  - → note reliance on n/w layer design

<u>sequence numbers</u> can be used with TPDUs such that "by the time the numbers <u>wrap-around</u>, old TPDUs with the same number disappear"

<u>three-way handshake</u>

Fig 6-11 (a)



(b), (c) — how delayed duplicates are rejected

(11)
31-10-14
(p.3)

6.2.3 Connection Release

→ asymmetric release can result in loss of data [Fig 6-12]

→ symmetric release - should have agreement from both entities

two-army problem

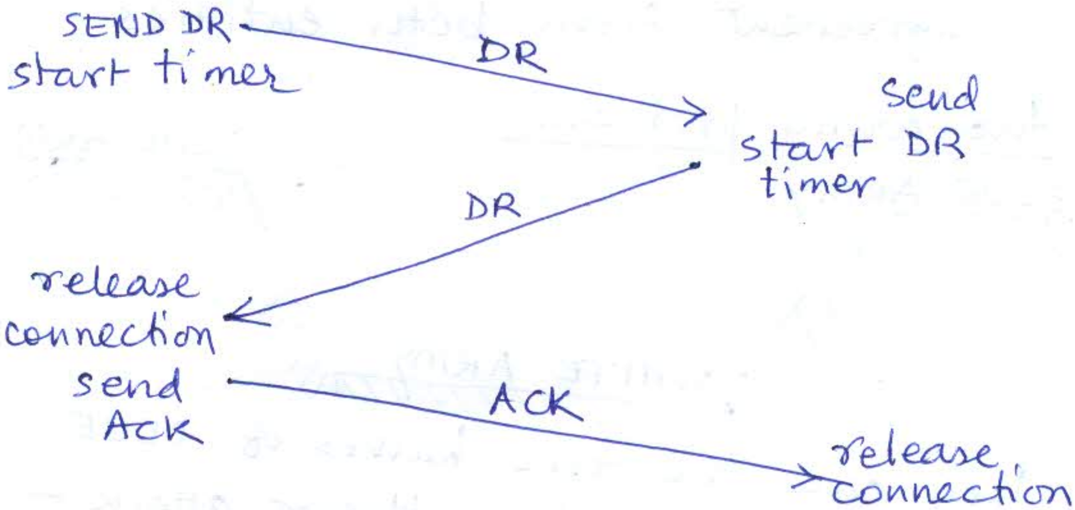
how can the two halves of BLUE ARMY synchronize their attack - when they have only an unreliable communication channel (foot messenger must pass through valley)?

It can be proven that no protocol exists that is guaranteed to work!

key - Is the last message of the protocol essential?
- yes ..... what if it is lost?
- no ..... remove it

p.4 three-way handshake with a timer provides an adequate solution –

Fig 6-14 (a)



timer expiry (time out) takes care of lost TPDUs *

see parts (b), (c), (d) of Fig 6-14

except in some extreme cases

another possibility –
release connection upon a long period of inactivity

(13) (p.2)

high bandwidth traffic

e.g. file transfer

→ "full" window of buffers, so that data is transferred at maximum possible speed

→ the nature of application plays a key role; recall this was not an issue at data link layer

→ control TPDUs may allow some form of "negotiation" between sender and receiver

→ number of buffers can be made dynamic; in effect we can have a variable size window

→ with current technology, memory size (i.e. availability of buffers) would not usually be a limiting factor

another factor -

"carrying capacity" of the subnet **

adjacent routers exchange pkts @ x pkts/sec

* BUT NOTE...!

k disjoint paths between H_1, H_2

$\therefore$ nett rate $\leq kx$ pkts/sec

(14)

4-11-14

(P. 1)

6.2.4 FLOW CONTROL & BUFFERING

→ closely related aspects of managing a transport connection

→ a host may have numerous connections
Recall - 'client' and 'server' are both hosts!

We assume datagram subnet (as in IP)

→ sender should buffer ~~frames~~ TPDUs because
because they may have to be retransmitted
[as in data link layer]

→ receiver may or may not dedicate specific buffers to specific connections
[a buffer pool may be used]

details of buffer mgmt

→ fixed size buffers or
variable size buffers
linked or circularly linked buffers

... recall 'dynamically allocated memory' e.g. malloc, free etc.

"low bandwidth traffic"
→ acquire buffer(s) dynamically at both ends as needed
e.g. remote login [range of meaning!]

P.3

4-11-14

a more realistic approach -

network handles c TPDUs/sec

cycle time = r sec

- transmission
- propagation
- queueing
- processing

$\Big\}$ data + ack

→ sender's window = cr

estimated

c and r can be monitored by the sender using probe TPDUs

6.2.5 Multiplexing

→ multiple TSAPs share an NSAP

→ one TSAP utilizes multiple NSAPs

+ ⇒ [the kind of "multiplexing" which happens over data links]

6.2.6 Crash recovery

router crash → Transport layer handles

host crash → trickier - especially on the server side

write and Ack

or

Ack and write ?

→ no perfect solution at Transport Layer

→ application can do more, as needed

16

7-11-14

p.1

6.5 TCP
Transmission Control Protocol

"... specifically designed to provide a reliable end-to-end bytestream over an unreliable internetwork."

→ tries to dynamically adapt
→ and be robust in the face of many kinds of failures

TCP entity accepts user data streams ...
... sends each piece as a separate IP datagram"

"TCP" may refer to
- transport entity
- the protocol

socket number ⇒ IP address of host + 16 bit local port number

connection ⇒ identified by the socket identifiers at the two ends

[i.e. no separate "connection number" or "vc number"]

"daemon" processes handle the work associated with ports

implementation detail

(17)

(p.2) - byte stream
- full duplex
- point-to-point — no multicasting or broadcasting

Fig 6-28 → self study

buffering → not under direct user/appn control
[some indirect control through the use of a couple of flags]

TCP segment -
20 byte header
+ optional part
+ data

each byte has a 32-bit sequence no.
[i.e. also a 32 bit acknowledgement no.]

flags - URG urgent
ACK acknowledgement
PSH push data
RST reset connection
SYN used to establish connection
FIN " " release connection
[+ 3 related to ECN]

18

7-11-14

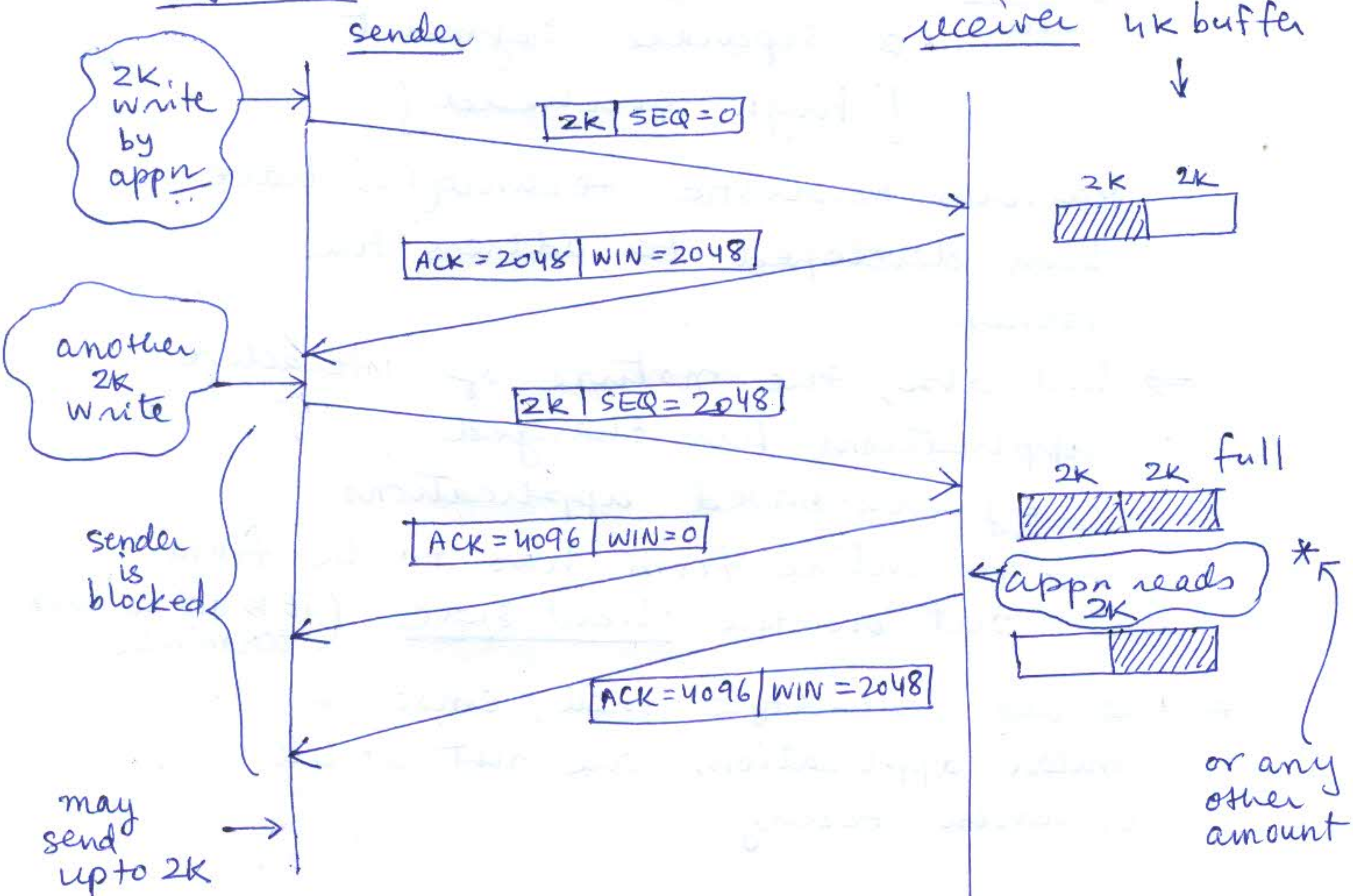
p.3

establishing & releasing a connection → essentially similar to what we have seen earlier

6.5.8 TCP Transmission policy

"window management not directly tied to acknowledgements"

Fig 6-34



(p.4) - sender not required to transmit data as soon as they come in from appⁿ

- receivers not required to send ACKs for every segment

⇒ issues related to interactive applications

should / can each byte be sent in a separate segment [high overhead]

various heuristic techniques have been developed to address the issues.

⇒ but also, the nature of interactive applications has changed.

e.g. web-based applications

→ an entire form has to be filled out on the client side (it is not a 'dumb' terminal)

* the old UNIX-style shell, emacs or similar applications are not at all common today

20

7-11-14

p.5

Self-study "silly window syndrome" and its suggested solution

6.5.9 TCP congestion control

→ why?

→ symptom: lost packet (which is assumed to be dropped pkt)

see Fig 6-36 fast / slow network feeding a low / high capacity receiver

which is the bottleneck?

network capacity or receiver capacity

two "windows" — specified/granted by receiver
— congestion window

→ sender will send according to the lower of these two

new connection → CW = max. segment size, MSS of that connection

on timely acknowledgement

$$CW = CW + MSS$$

suppose $CW = n \times MSS$
and all are acknowledged in time
then $CW = 2n \times MSS$

(21)

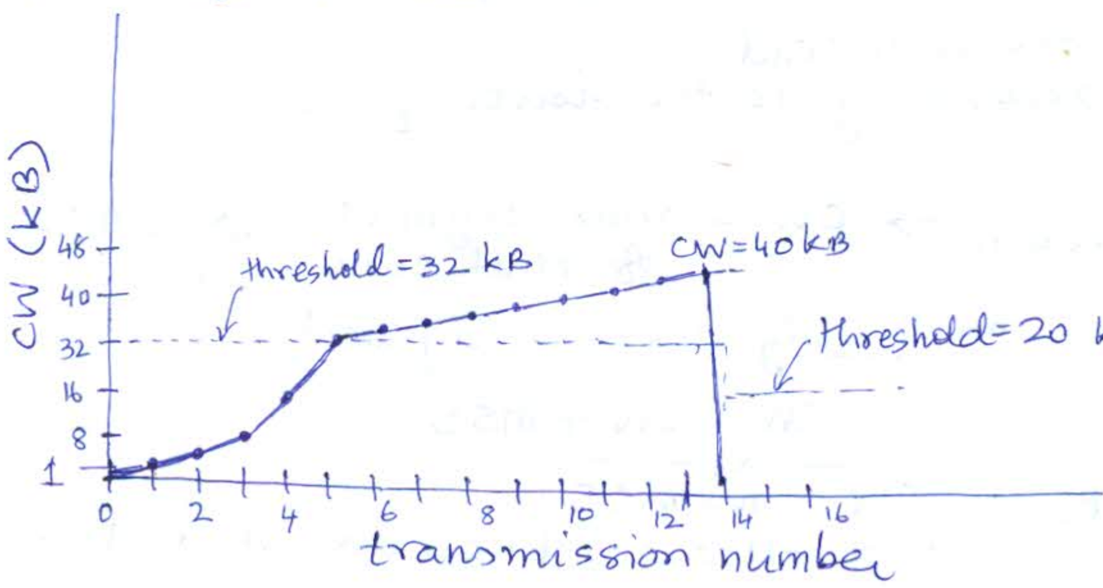
(p.6) thus regular timely acknowledgements cause doubling of CW
this is called "slow start"
[but not to exceed receiver's window size]

third parameter: threshold
initially 64 KB

on a timeout, $\begin{cases} \text{threshold} = \dfrac{\text{current CW}}{2} \\ \text{CW} = \text{MSS} \end{cases}$

beyond the threshold, the CW increases linearly [not doubling] on each "burst" of successful CW transmission

(part of) Fig 6-37



14-11-14

(P.1)

6.5.10 TCP Timer Management

Retransmission timer –

see Fig. 6-38

p.d. of ACK arrival times in | DLL
| TCP – changes w/ time

if timeout is too short
→ unnecessary retransmissions

if timeout is too long
→ long retransmission delay after packet loss

Solution –

for each TCP connection, maintain a variable – RTT – which is an estimate of round trip time for that connection

on every timely Ack

$$RTT = \alpha RTT + (1-\alpha) M$$

where M is the measured time of the Ack, and

α = "smoothing factor" typical = 7/8

$$\text{timeout value} = \beta \cdot RTT$$

Q: value of β in a changing environment?

(23)

(p.2) proposal (1988) - make β proportional to the s.d. of the distribution of Ack arrival times

→ use mean deviation as an approximation to s.d.

$$D = \alpha D + (1-\alpha)\,|RTT - M|$$

[efficient evaluation is also a criterion]

"most TCP implementations" use

$$\text{timeout} = RTT + 4D$$

Q: which 'M' value to use if an Ack comes after two transmissions of a segment?

A: [valid on most TCP implementations]

- do not update RTT on retransmitted segments
- timeout doubled on each failure

ALSO - persistence timer - to send a window probe to receiver

keepalive timer - for idle connections

closing state timer = 2x maximum pkt lifetime

14-11-14
(P-3)

6.6 PERFORMANCE ISSUES

- scale
- complexity
- frequently, "poor performance" and no one knows why
- "art" or science ??
- "... system-oriented issues tend to be transport related ..."
- bandwidth - delay product

  self-study Fig. 6-41

6.6.2 Network Performance Measurement

- sample size should be large enough
- samples should be representative
- clock resolution
- controlled experimentation
- errors introduced by caching
- "understand what you are measuring"
- linear extrapolations usually do not work

  ex. simple queue

* some of these points apply to NS-2 simulations as well

6.6.3 System Design for Better Performance

- CPU speed is more important than network speed
- Reduce packet count to reduce software overhead
- Minimize context switches
- Minimize copying
- You can buy more bandwidth but not lower delay
- Avoiding congestion is better than recovering from it
- Avoid timeouts

6.4.1 Introduction to UDP ← self-study

basically IP packet with source and destination port #s

↓

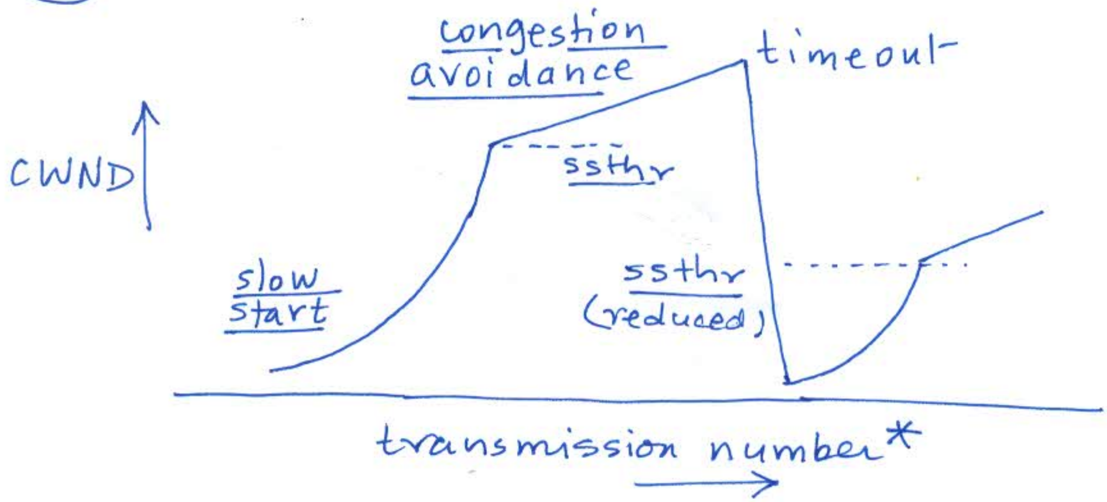
basic concept of RPC

(26)

18-11-14

(P.1)

TCP Flavours (summary)



* to be understood in terms of CWND worth of "burst" of data sent

fast retransmit

three duplicate ACKs trigger retransmit

fast recovery

each dup ACK indicates scope to send segment / avoid going to CWND = 1 mss (slow start)

selective ACKs

using the optional part of header