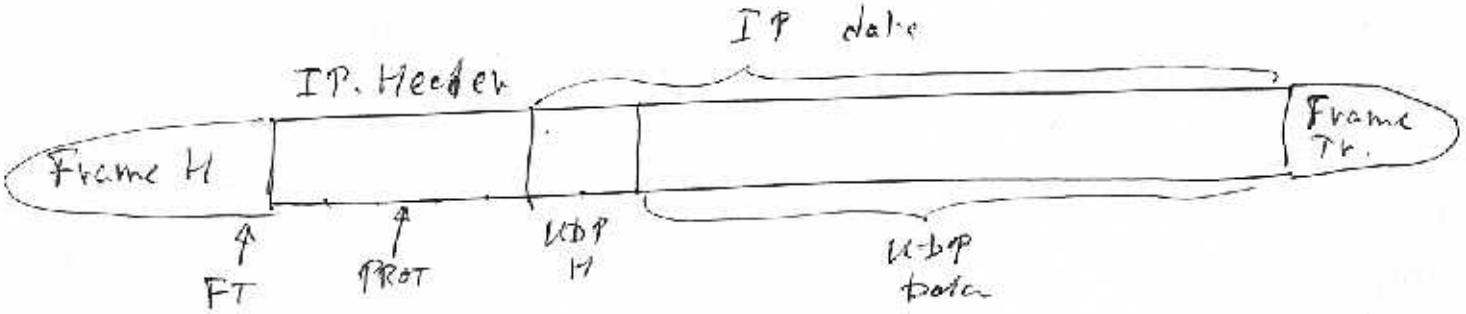
UDP. Comer Ch 12, p 197

User Data gram Protocol.

Ex.

- UDP Packets are carried inside IP packets. (IPv4 or IPv6).



(ethernet carrying IPv4 carrying UDP)

Frame Type = 0800 = 0000 1000 0000 0000

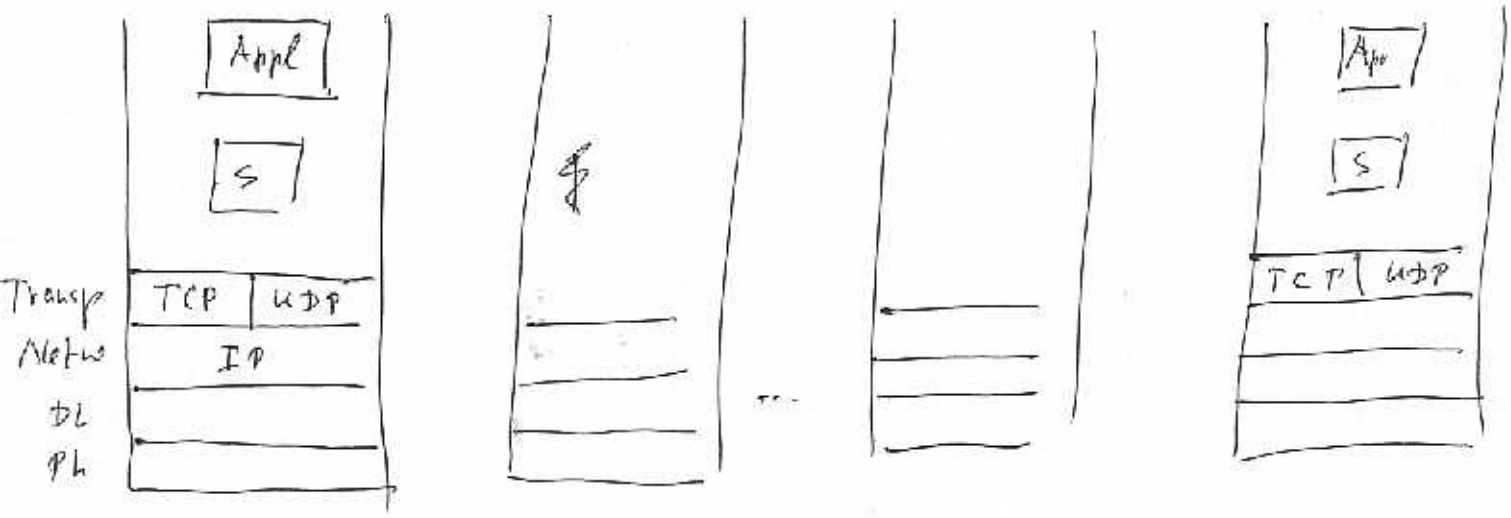
PROT = 17 = 0001 0001 .

Next: Port number.

What do ethernet Frame Type
IPv4 Protocol Identifier
UDP port number

have in common?

each points at next higher protocol.



Paradigm Shift:

The customer is a process.

This is process-to-process communication.

The UDP packet (H & D) does not change from end to end!

Remember: The IP header does change:

(1) TTL --
(2) Fragmentation if necessary
(3) ~~Recomputing~~ Check Options!
(4) Recomputing Checksum.

UDP needs a method to identify the application process.
(In some sense: the socket. That is not part of the ~~stack~~ standard.).

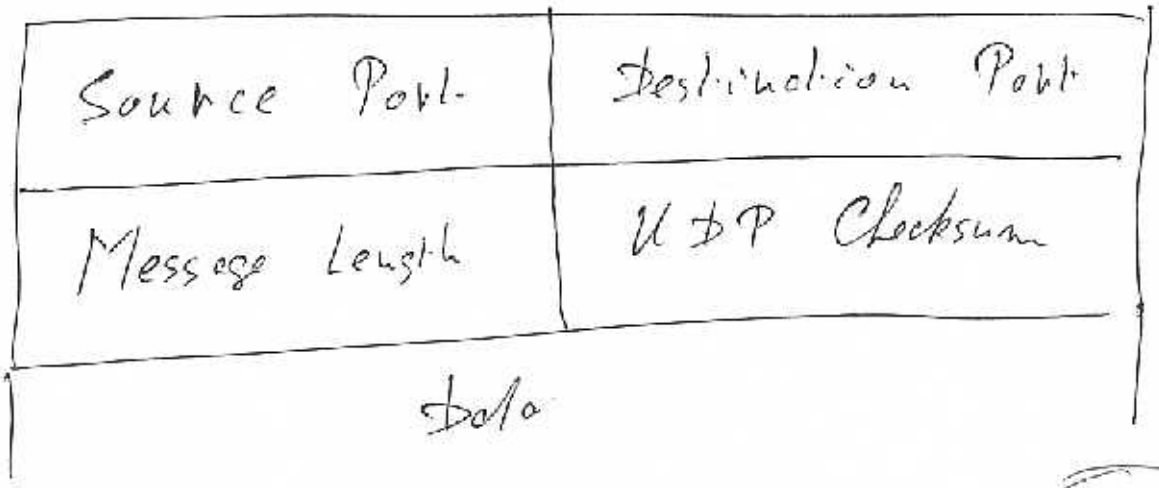
How is the customer identified?

Port Number.

The source must know the port number of the destination.

Format of UDP packet.

Comer p. 199



NO UDP Options

Ports: 16 bits each.
(0 – 65535).

Message Length:
H & D, in bytes.

Checksum: over Pseudo Header (later) and Data.

(unlike IP).
Use of UDP checksum is optional.
Zero if unused.

Checksum:

if used it is never zero!

Port numbers: identify processes.
(in most or all OSs: there is a Socket to help.).

Port numbers:

0 – 1023 "Well known" port numbers
IANA

1024 – 49,151 Registered (?) (IANA?)

49,152 – 65535 Dynamic
Ephemeral

A process may "listen at a port".

Sometimes: "take any comer".

Sometimes: only if from specific
IP address / port number.

HTTP listens at port 80.
(mostly TCP, not UDP: later).

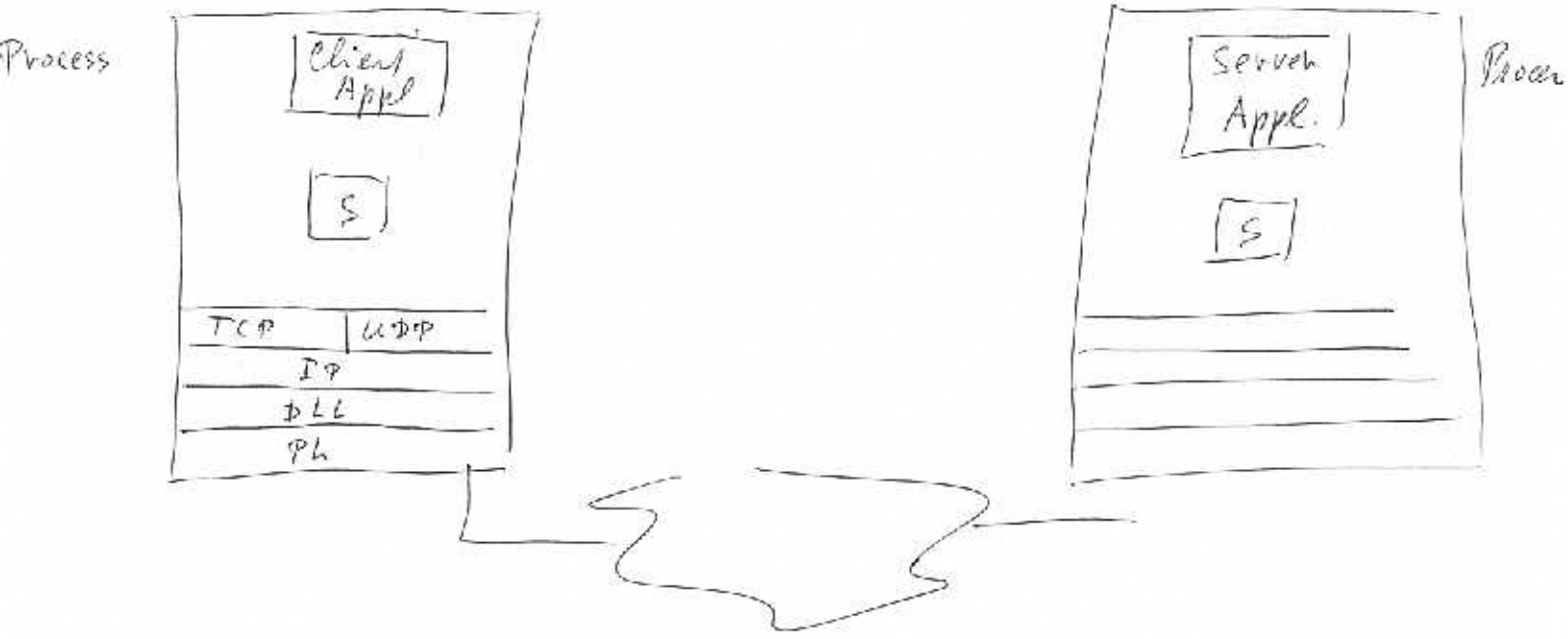
"Time of Day" listens at
UDP port 13.
(date).

Comer p. 205 has list of
port numbers:
well-known.

What do ethernet Frame Type
IPv4 Protocol
UDP portnumber

have in common?

Client-Server Model.



A. Client, Server are <u>processes</u>.

B. Client, Server must <u>already exist</u>.

C. Client procln must know:
   (1) S's host IP address.
   (2) Server's IP address.
   (3) Server's <u>Port number</u>.
   (Also for: UDP port number).

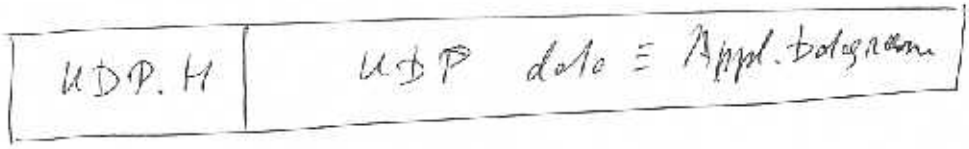
Client needs service from server.
(say date).

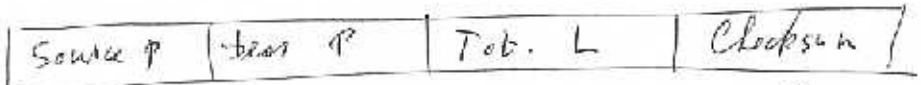
(1). Client asks its OS for a
UDP port number.
("next available").

(2) Client makes "datagram".

(3) Client sends to UDP:
Appl. Data gram. (becomes UDP data).
+ Client IP address
+ Client port number
+ Server IP address
+ Server port number.

4. UDP makes "UDP datagram".



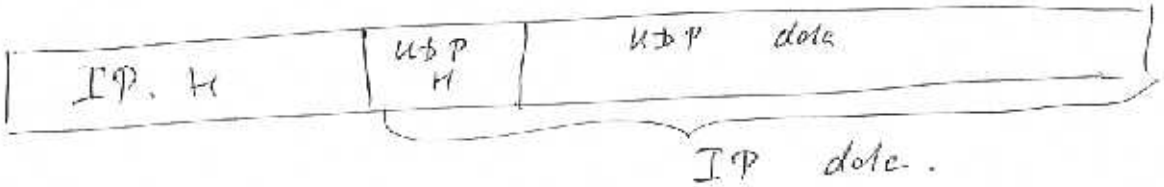
Header: 

P later.

(5) UDP gives IP:

UDP datagram
+ Source IP address (yes!) (why?)
+ Dest IP address

6. IP makes IP datagram.



Protocol number in IP header is 17.

7 At this point fragmentation may (already) occur!

Who determines size of UDP datagram?

Appl!

If the first network has small MTU: fragmentation occurs!

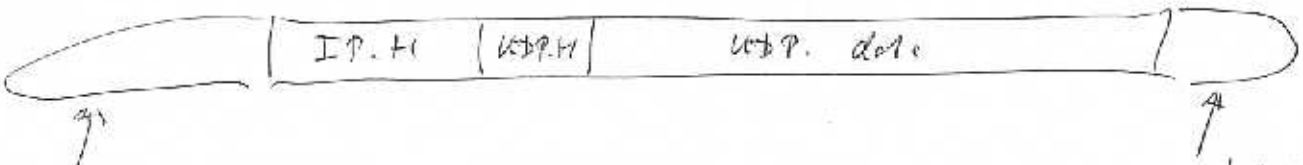
8. IP datagram (or fragment) is/are sent.

Fragmentation:

How many IP ~~header~~ headers?

How many UDP headers?

In case of LAN = ethernet



(LAN. H & LAN. Tr: LAN WRAPPER).

LAN Wrapper: completely new every Hop

IP. H : moderate changes every Hop

UDP. H : No changes!

Unchanged e2e.

9. IP datagram
or fragment
arrive at dest.

IP: re-assemble (if necessary).

Split off UDP datagram.

(10) IP hands to UDP:

UDP datagram.

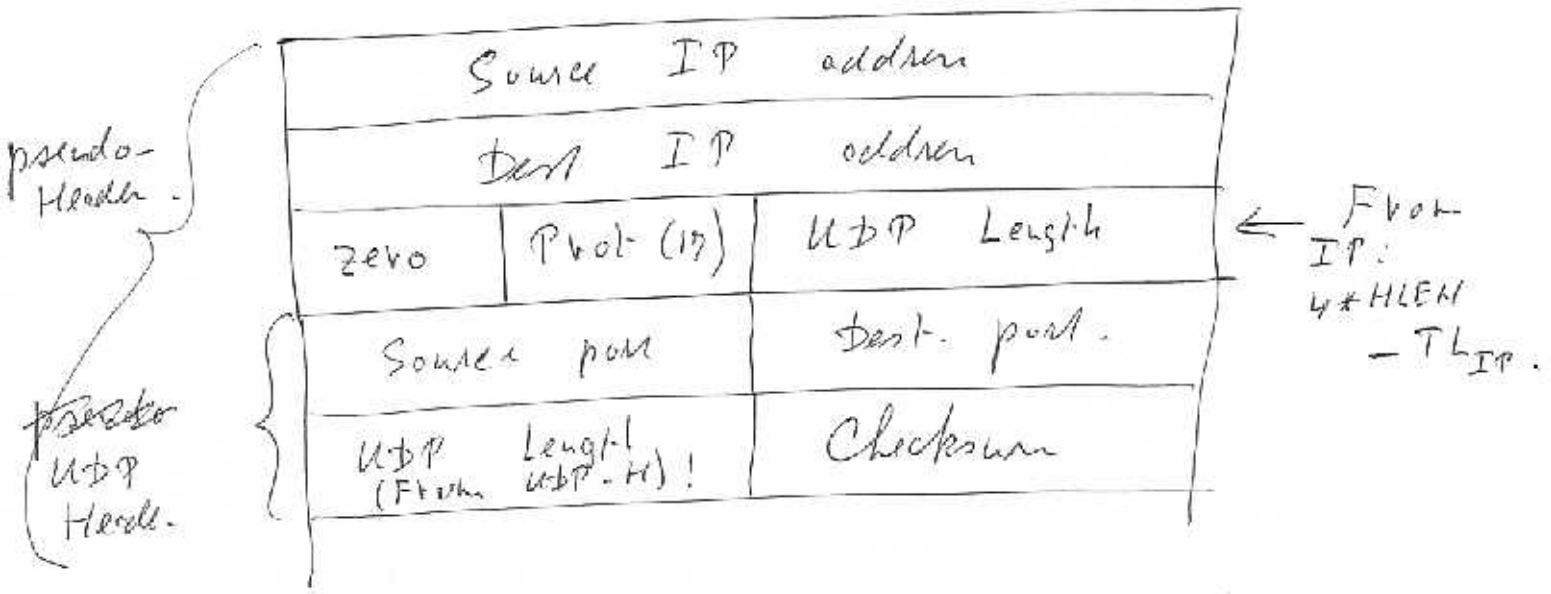
+ Client IP address
+ Server IP address (yes!)
+ UDP Total length (from IP Header!
HLEN*4 − TL_{IP}.

(11) UDP ~~does~~ computes

pseudo-header:

11 contd.

Pseudo Header.



(UDP Length: 2*! from different source)

The source did the same thing!

(check: 1 led all the data to make the pseudo-header).

The source (optionally) computed checksum

The dest (if checksum != zero) to checks checksum.

If UDP data changed
UDP header changed,
Port changed (TCP ⇒ UDP or so)
Length changed,
IP address changed.:

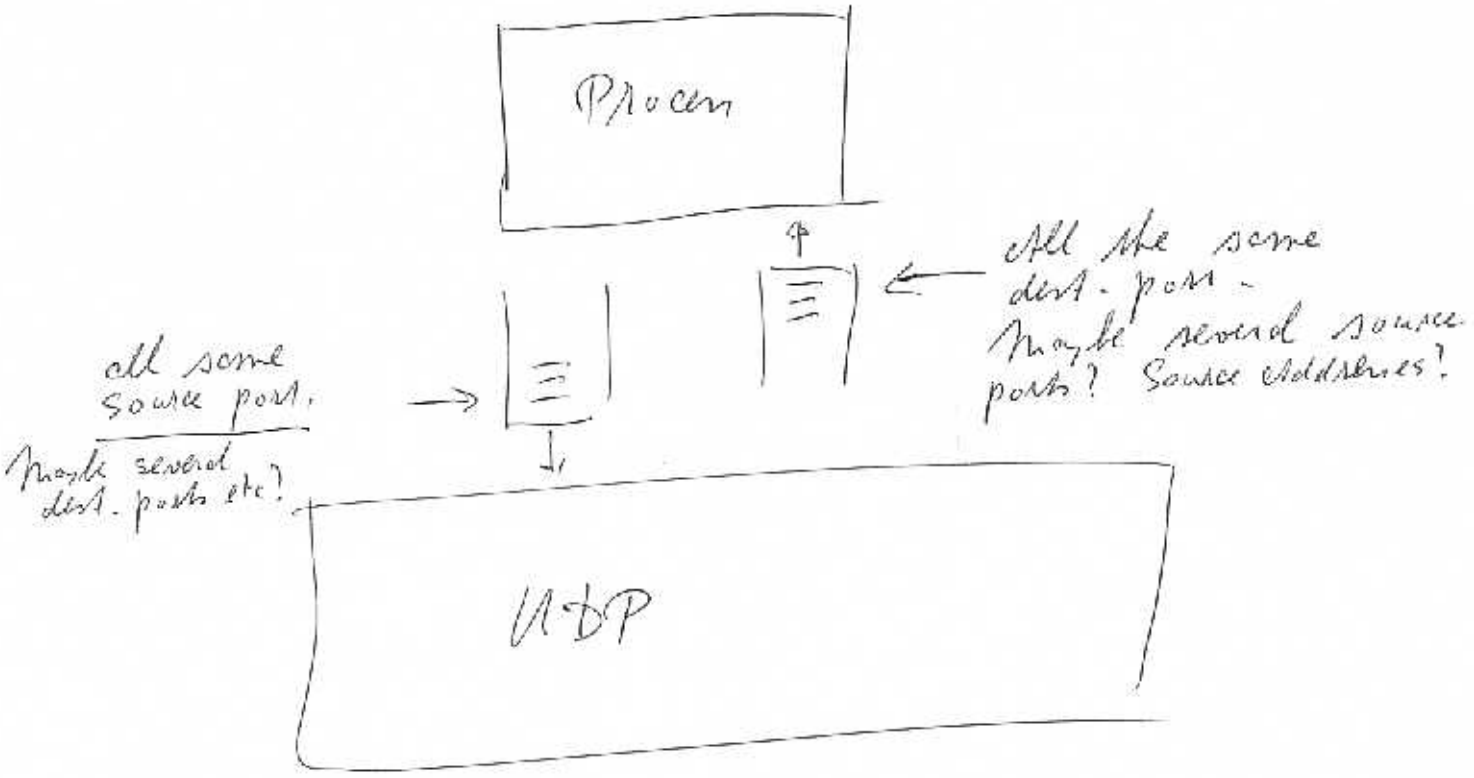
Checksum fails.
~~Packet~~ Datagram drops.

(12) If all Ok:

UDP hands to Appl. (Server) the
~~data~~ Appl. data given.
plus Client IP address
Client Port number
Server IP address
Server Port number.

If the server wants to send back:
it has all that is needed.

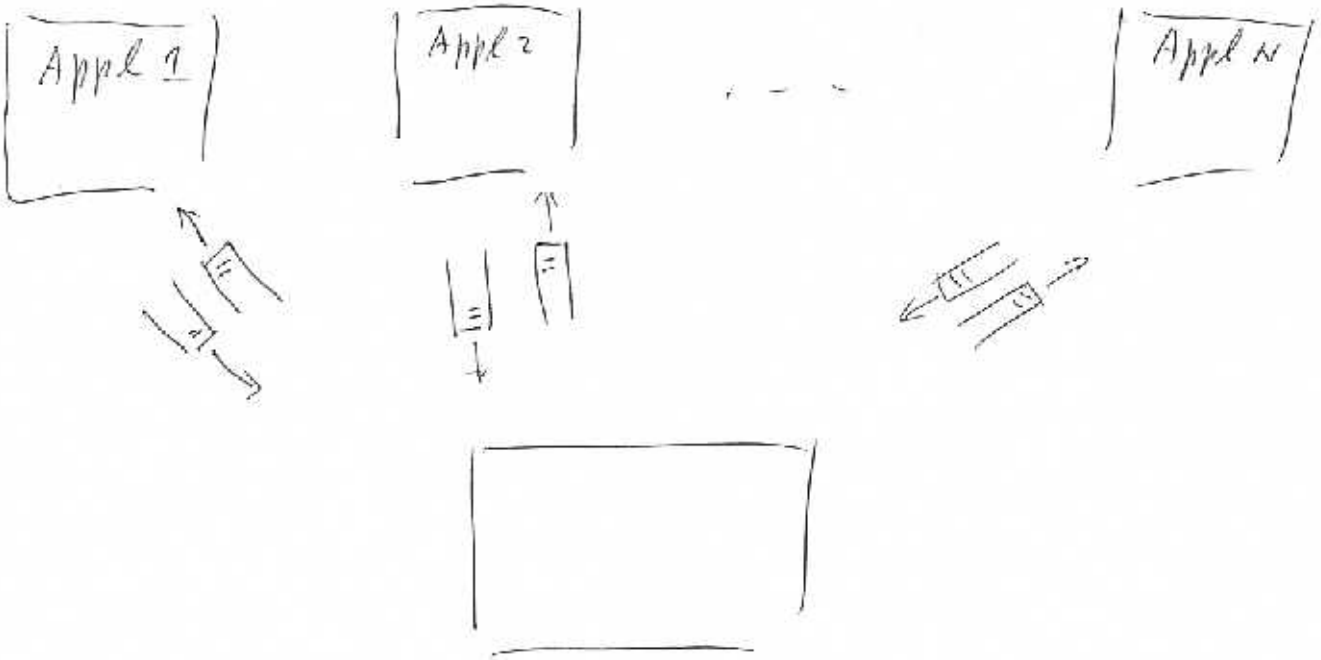
Between UDP and "customer" (process) there is a queue



In Unix, Windows (also Java?) there is an entity called socket that "helps" in between TCP - UDP and the application processes.

Not part of standard.

UDP as multiplexor



UDP : Does not keep state
(per "flow").
(complete Alzheimer).

UDP : connection less.
(Socket exists under control of process).

No Set-up
No Tear-down
No Connection
No Error Correction
No Re-transmission
(in case of loss, damage, ...)
No Flow Control
(What is flow control?)
No Overload Control.
(What is overload control?).

On the other hand!

No Overhead.

UDP used for:

(1) RPC. Remote Procedure Call.

(2) Any time "Customer" (Process) has its own
Flow Control
Error Control
Re-transmission
⋮

When is it worth the trouble?

Real Time Traffic.

VoIP Voice over IP.

Start on TCP.

Comer Ch. 13, p 209.

UDP

"No State" (per Flow).

Connectionless

No Set-up

No Tear-down

No Connection

No Error Connection

No Re Transmission (in case of loss, damage).

No Flow Control (what in flow control?)

No Overload Control (what in overload control?)

Datagram Oriented

TCP.

Lots of "State".

Connection oriented

Set-up (SYN)

Tear-down (FIN)

Connection!

Retransmission

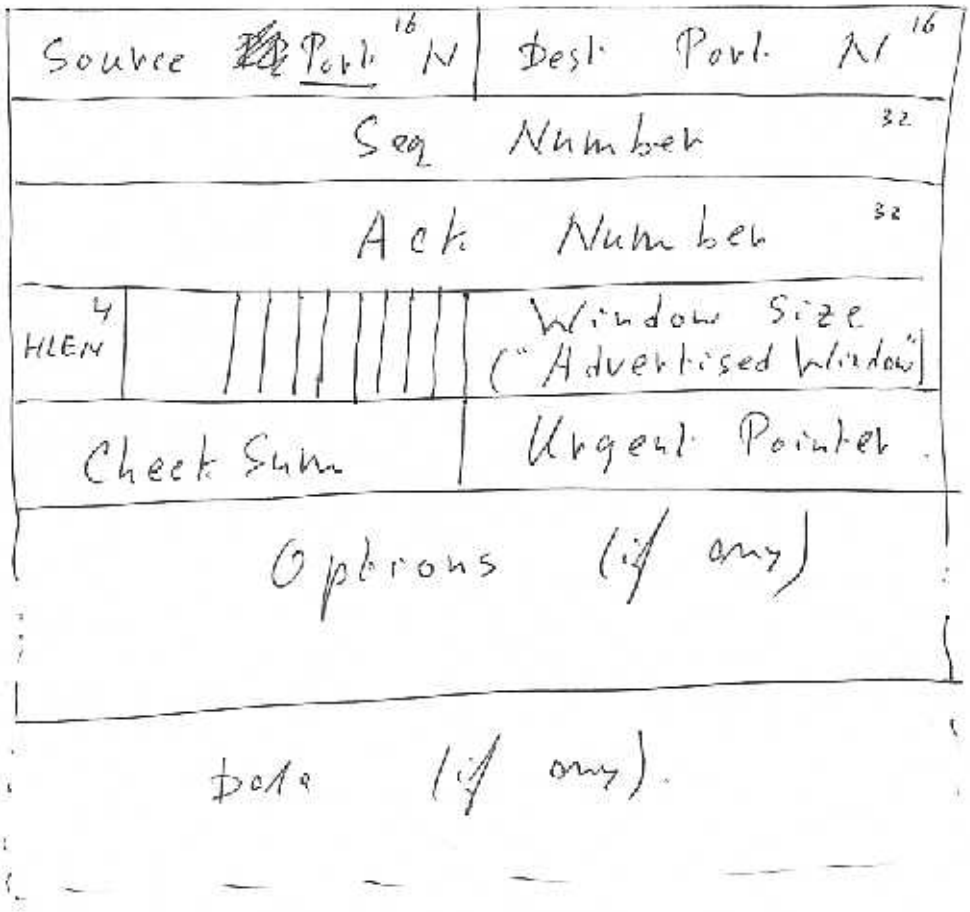
Flow Control

Overload Control

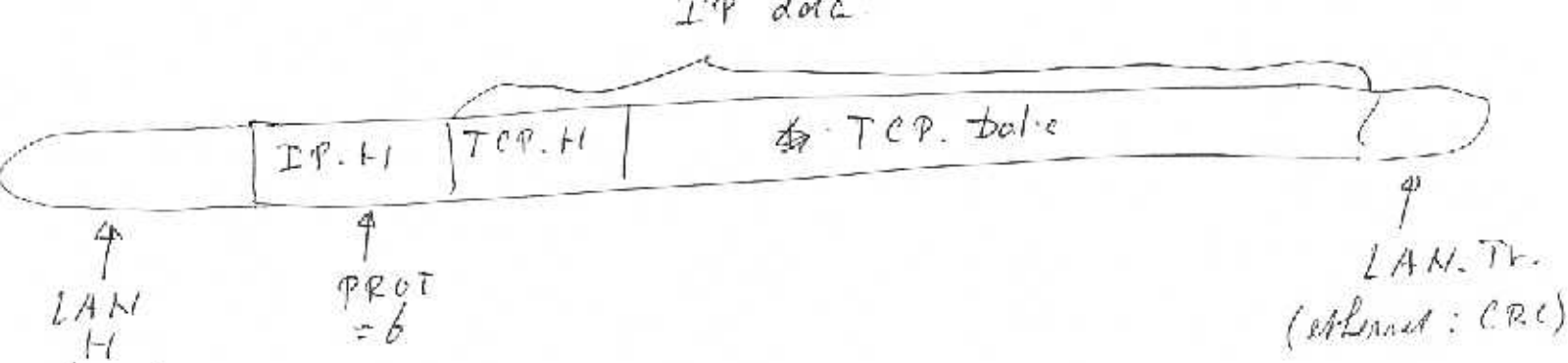
Byte-& Oriented

Byte-Stream Oriented

TCP Header.



Control Flags
Used to be 6.
now 8



TCP is byte oriented.

Bi-directional.

There is connection set-up,
tear-down.

The Bytes are numbered:

init + 1, init + 2, init + 3, ...

32 bit sequence number.

Repeats after $2^{32} = 4,294,967,296$.

Sequence number: the number of the
first TCP data byte in the packet.
(or: if there are no data bytes:
the higher previous number).

Ack number: "I have received from you,
contiguously, everything up to
Ack-1, but not Ack (next expected)

Then may be received more, ~~late~~
higher numbers: gap.

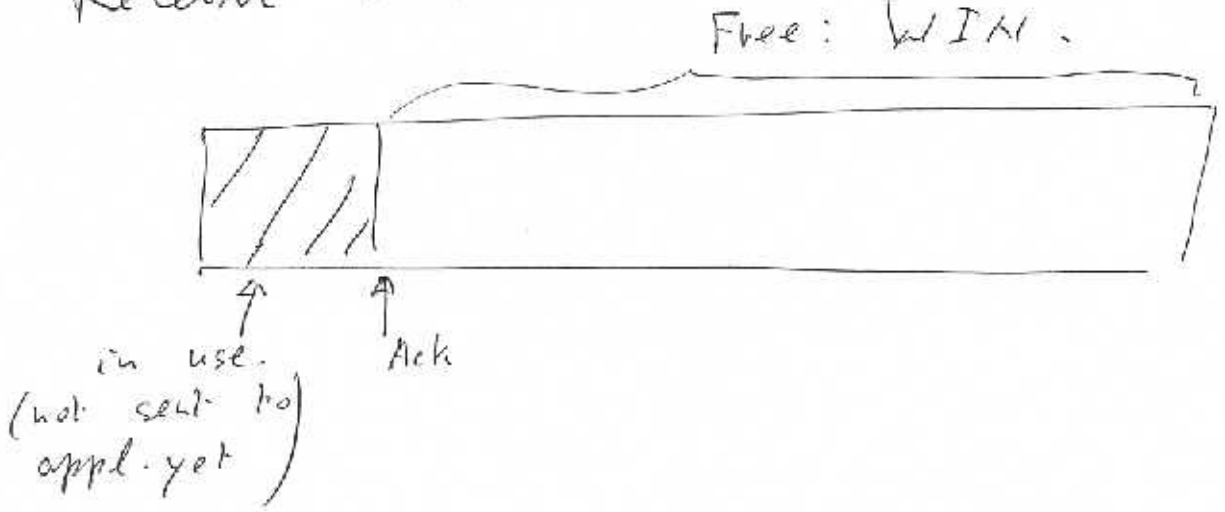
Window WIN

("Advertised Window").

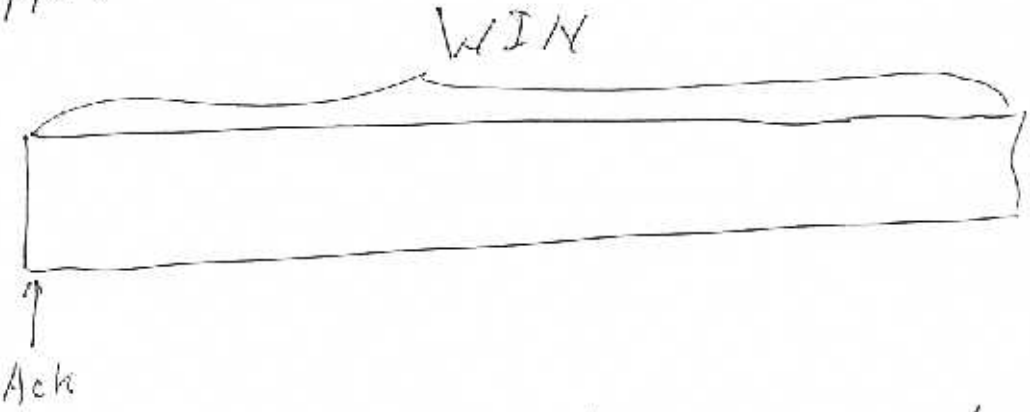
"You can send me bytes up to ACK + WIN.

(Flow Control!)

Receive ~~Window~~ BUFFER:



/// to appl:

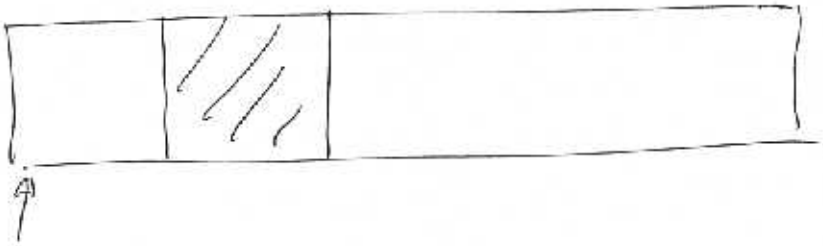


It is legal to send in quick succession two packets, no data, same ACK, but increasing WIN.

Reliable.

Ack

receive packet "out of sequence". (previous one lost?)



Another ACK (this one compulsory)

Same Ack number
Some bit IN
no data

} "duplicate acknowledgement."

mean: got something past Next Expected,
not what wanted.

One of the ways to achieve reliability

Duplicate Acknowledgements:

Same Acknum
Same WIN
No data.

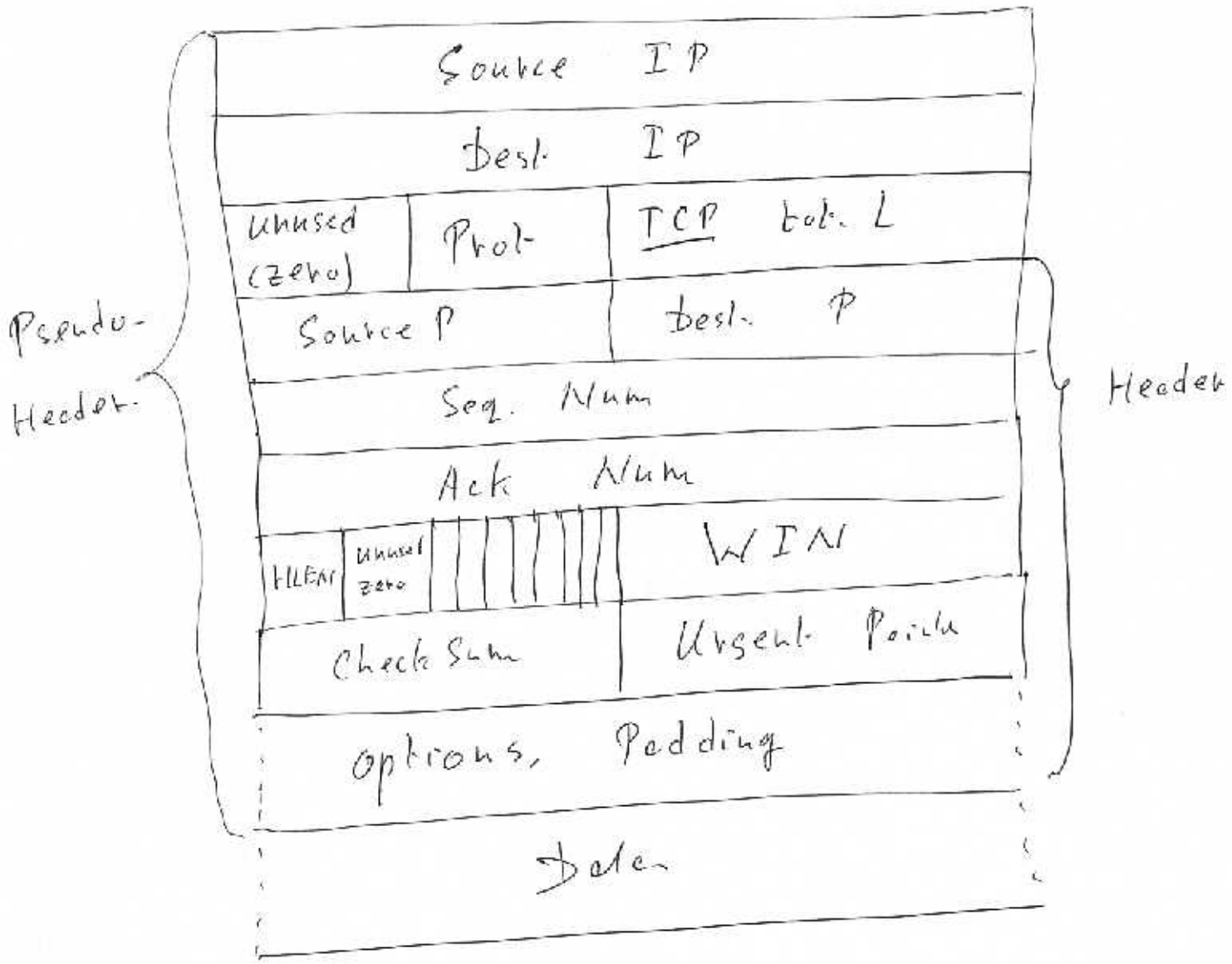
count dup acks.

~~After~~
TCP Reno:
After third duplicate acknowledgement,
(source) assume packet lost,
Re-transmit,
and do more (overload control)

When acknum increases:
reset counter to zero.

RCS

TCP checksum : compulsory.
over pseudo-header and TCP data.

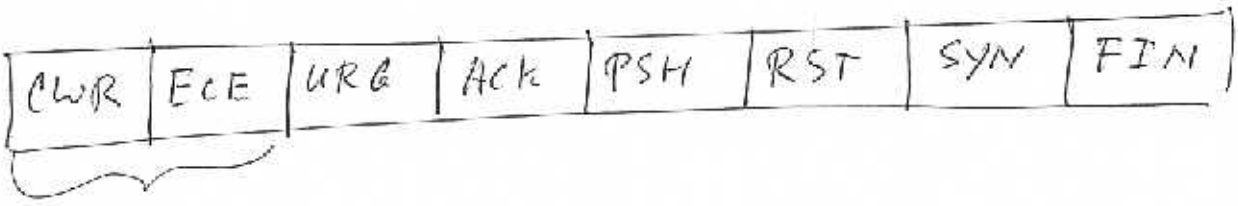


Flags

There used to be 6 Control Flags.

In about 2001 two were added, for ECN. Explicit Congestion Notification.

RFCs 3168 etc.
(also RFC 3540)



CWR: Congestion Window Reduced
ECE: ECN Echo

URG: the urgent pointer is valid.
URGENT pointer: to Relative to Seq.
"Interrupt application"

Ack: Ack is valid
("there has been previous data from you")
Almost always = 1, unless ...

PSH: Push ("do not wait for full MSS").

RST: Reset. (?)

SYN: This is a SYN Packet

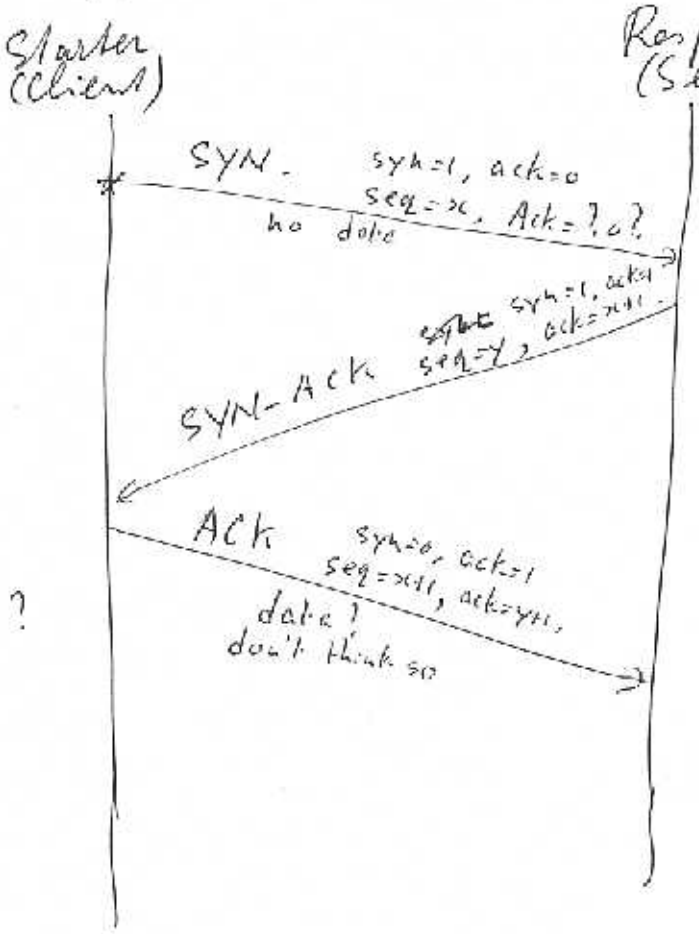
FIN: This is a FIN packet.

HLEN: TCP header length in 32 bit words.

ECN: experimental standard.

Connection Set-up:

Three Way Handshake.



Now data can Flow.

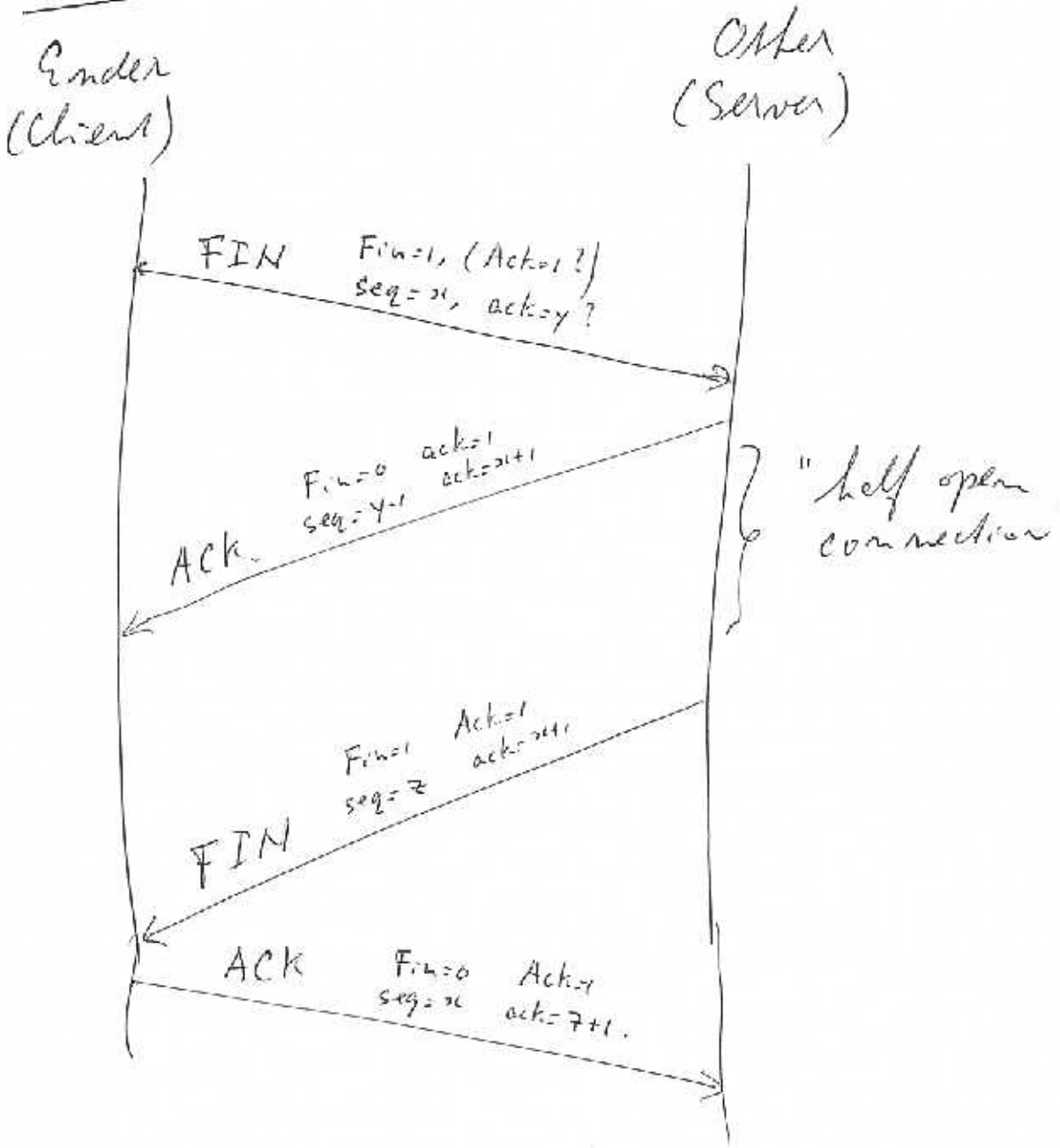
This set-up is <u>Robust</u> under:

Loss of packets.

Two parties starting at the "~~same~~ same" time, etc.

Connection Tear-down.

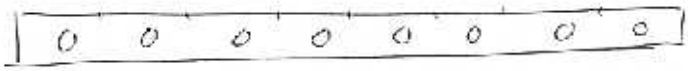
Four way handshake -



However, in case of "no half open connection" I have seen a <u>different</u> pattern!

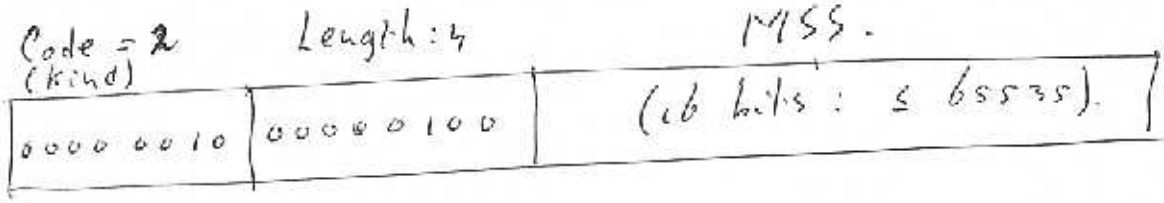
TCP options.
Comer p. 223 etc.

(1) End of Options
E.O



(2) No-Op.

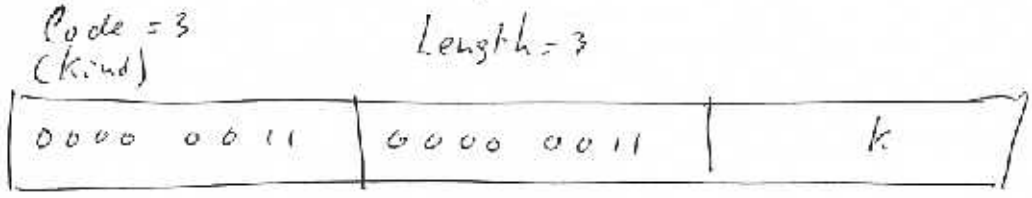
(3) MSS. Maximum Segment Size.



The next options are not in Comer.
You have to know them "temporarily",
Because they occur in the tcp dump data
you will look at.

MSS: "I am willing to accept from you
TCP packets with up to MSS data bytes."
Only in SYN, SYN-ACK packets.
Not necessarily symmetrical!
No negotiation!

4 Window Scale factor. Only in SYN-



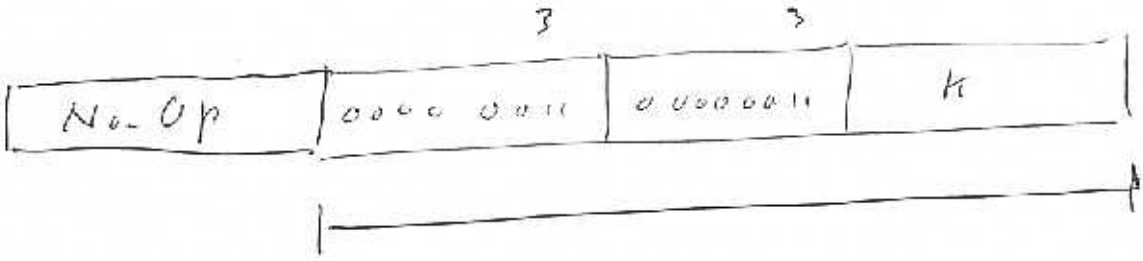
~~k~~ k: 8 bits : could be 255
~~but~~ only $0 \le k \le 16$ is legal.

"IF I send you WIN,
I mean a window of
$WIN * 2^k$.

Why k=16 mentioned?

Default: k=0. ("WIN = WIN").

Could be



5. Time Stamp.

(TCP Time Stamp).

kind
Code = 8 L=10

0000 1000	0000 1010
Time Stamp Value	
Time Stamp Echo Reply.	

Time in msec. since last
Universal Time midnight.

Could be

No-Op	No-Op	

Something Strange in tcpdump.

with NTP

(6) SACK.

Selective Acknowledgement.

RFC 2018

RFC 2883.

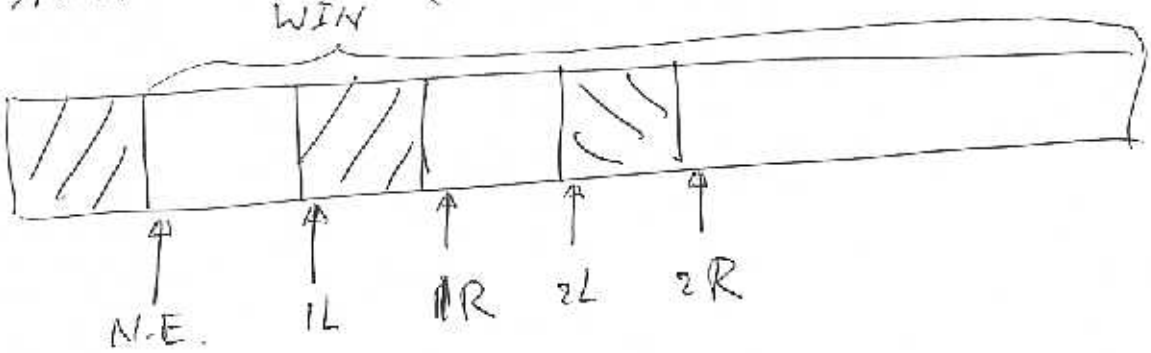
Two components:

(1) In SYN: "I am SACK capable".

(2) Later: used to describe what data is missing.

First (2).

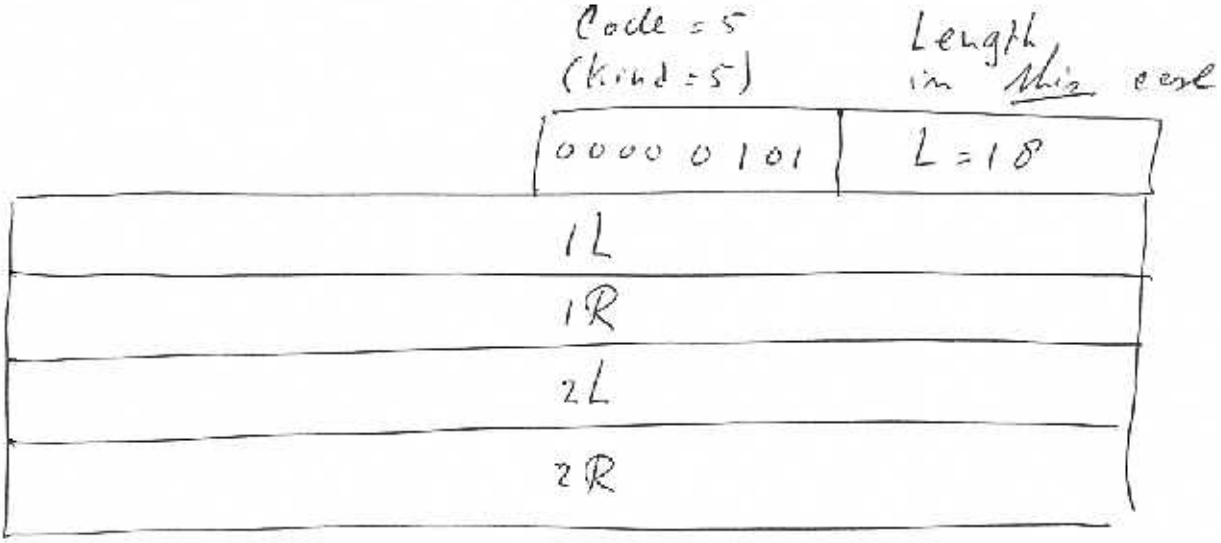
Suppose the receiver is in the situation (receive buffer)



Old way: Send dup ack: "N.E".

No details of what did, did not, arrive.

Now there is the TCP-SACK option



N.E. (next expected) is in the acknowledgement field.

1L, ..., 2R : absolute numbers (sequence numbers).

How many gaps can this describe?

Say n gaps: 2n entries.
(HLEN ≤ 15 = (1111))

$2*n+1+5 \leq 15 \quad 2*n \leq 9 \quad n \leq 4.5$

$\boxed{n \leq 4}$. Up to 4 gaps.
If no other options.

Problem: Source Behavior.

"When to Re-Resend?

---

(1) "I understand SACK" at SYN.

(SYN only).

Code = 4 (kind) L = 2

0000 0100	0000 0010

// both are SACK-capable,

it will be used.

SACK: often seen in tcp dump output in lab.
