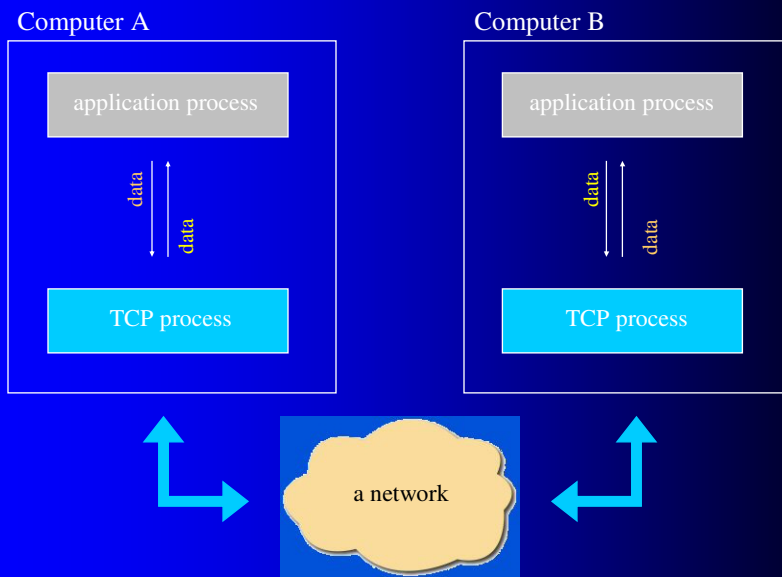


Linux Networking: tcp

David Morgan

TCP context and interfaces



TCP purposes and features

- basic data transfer
- process-to-process multiplexing
- reliability
- flow control
- connections

Transport purposes and features

	<u>TCP</u>	<u>UDP</u>
● process-to-process data transfer	✓	✓
● reliability	✓	*
● flow control	✓	
● connections	✓	

* discard, no recovery

Basic data transfer method

- sending TCP
 - “blocks out” (segments) the data stream
 - gives each block its own packet (“segment”)
- receiving TCP
 - reassembles the blocks into original stream

Multiplexed “process-to-process” transfer

- processes get identifying numbers (“ports”)
- IP address/TCP port pair is a local “socket”
- pair of sockets, one on each of 2 machines, associated with a unique bilateral “connection”
- packets between machines belong to a particular one of the machines’ connections
- overall packet flow contains separate flow for each connection

Reliability

- problems with data

- damaged
- lost
- duplicated
- delivered out-of-order

- solution

- Sending TCP
- number the data
- require acknowledgement
- resend unacknowledged

- Receiving TCP

- acknowledge good data
- discard bad data
- reassemble by the numbers

Flow control

- problem

- sending TCP might overwhelm receiving TCP

- solution

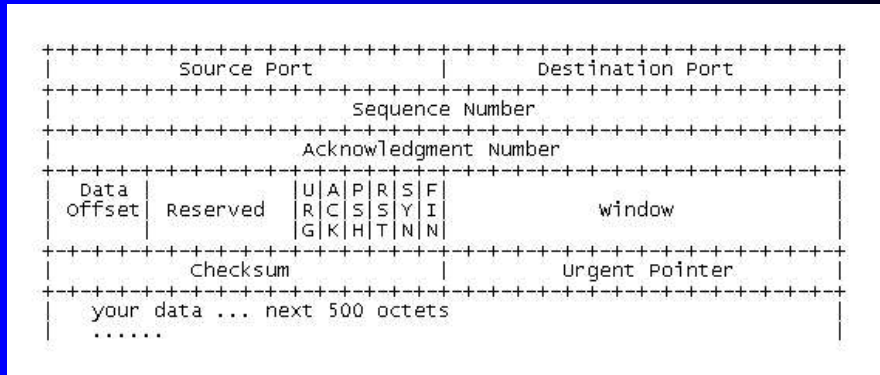
- constrain sender by requiring receiver's permission which data, by number range, may be transmitted

TCP connections

- reliability/flow control require state info
- each TCP initializes/maintains it for each data stream
- connection ends, state info data structures freed

TCP packet (segment) header

32 bits



“Flag” bits

TCP flags field

U	A	P	R	S	F
R	C	S	S	Y	I
G	K	H	T	N	N

TCP Header

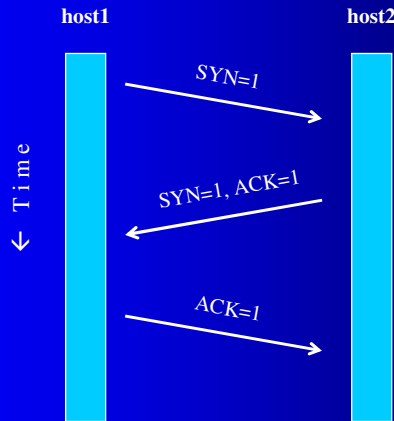
Source Port		Destination Port						
Sequence Number								
Acknowledgment Number								
Data offset	Reserved	U	A	P	R	S	F	Window
		R	C	S	S	Y	I	
		G	K	H	T	N	N	
Checksum			Urgent Pointer					
your data ... next 500 octets								
.....								

- URG = urgent
- ACK= acknowledgement
- PSH = push
- RST = reset
- SYN = synchronize
- FIN = finish

Establishing a “connection”

- client sends packet with SYN bit set
- server returns packet with SYN & ACK set
- client sends packet with ACK set
- called “3-way handshake”
- connection establishment’s signature sequence

3-way handshake



TCP - SYN

The screenshot shows a Wireshark capture of a TCP SYN packet. The packet list pane shows two frames:

No.	Time	Source	Destination	Protocol	Info
13	9.000000	thermador	www.google.com	TCP	32825 > http [SYN] Seq=1592481968 Ack=0 Win=6940 Len=0 MSS=1460 TSV=18763944 TSEB=0
14	9.100000	www.google.com	thermador	TCP	http > 32825 [SYN, ACK] Seq=1694990987 Ack=1592481969 Win=32200 Len=0 MSS=1400 TSV=75

The packet details pane for Frame 13 (74 on wire, 74 captured) shows the following information:

- Ethernet II
- Internet Protocol, Src Addr: thermador (10.100.13.138), Dest Addr: www.google.com (216.239.39.100)
- Transmission Control Protocol, Src Port: 32825 (32825), Dest Port: http (80), Seq: 1592481968, Ack: 0
- Source port: 32825 (32825)
- Destination port: http (80)
- Sequence number: 1592481968
- Header length: 40 bytes
- Flags: 0x0002 (SYN)
- Options: (20 bytes)
- Maximum segment size: 1460 bytes

The flag field is expanded to show:

- 0... .. = Congestion Window Reduced (CWR): Not set
- ..0. = ECN-Echo: Not set
- ...0. = Urgent: Not set
- ...0 = Acknowledgment: Not set
-0... = Push: Not set
-0 = Reset: Not set
-0.1. = Syn: Set
-0 = Fin: Not set

A red circle highlights the "Syn: Set" flag, with a red arrow pointing to the text: **SYN flag set indicates new connection request**. Another blue circle highlights the destination IP address "www.google.com (216.239.39.100)" in the packet list pane, with the word "Server" written below it.

TCP - SYN/ACK

The screenshot shows a packet capture in Wireshark. The selected packet is a TCP SYN/ACK from www.google.com (216.239.39.100) to thernador (10.100.13.138). The flags field is expanded to show: SYN: Set, ACK: Set, and others as Not set. Red circles highlight the SYN and ACK flags, with the text "SYN and ACK Flags set" overlaid in red.

No.	Time	Source	Destination	Protocol	Info
13	0.000000	thernador	www.google.com	TCP	32825 > http [SYN] Seq=1592481968 Ack=0 Win=5840 Len=0 MSS=1460 TSV=18752944 TSEQ=0
14	0.100000	www.google.com	thernador	TCP	32825 > http [ACK] Seq=1592481969 Ack=163499588 Win=5840 Len=0 TSV=18752954 TSEQ=755
15	0.100000	thernador	www.google.com	TCP	32825 > http [ACK] Seq=1592481969 Ack=163499588 Win=5840 Len=0 TSV=18752954 TSEQ=755

TCP ACK

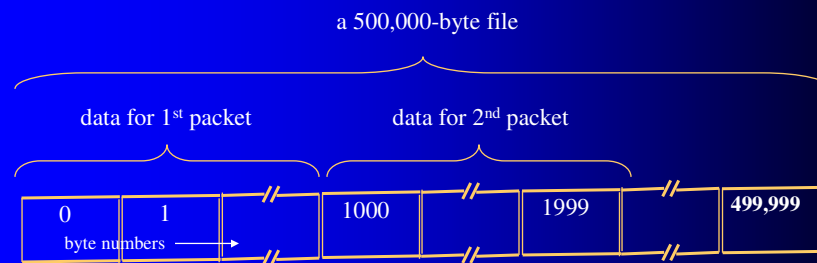
The screenshot shows a packet capture in Wireshark. The selected packet is a TCP ACK from thernador (10.100.13.138) to www.google.com (216.239.39.100). The flags field is expanded to show: ACK: Set, and others as Not set. A red circle highlights the ACK flag, with the text "ACK Flag" overlaid in red.

No.	Time	Source	Destination	Protocol	Info
13	0.000000	thernador	www.google.com	TCP	32825 > http [SYN] Seq=1592481968 Ack=0 Win=5840 Len=0 MSS=1460 TSV=18752944 TSEQ=0
14	0.100000	www.google.com	thernador	TCP	http > 32825 [SYN, ACK] Seq=163499587 Ack=1592481969 Win=3200 Len=0 MSS=1400 TSV=75
15	0.100000	thernador	www.google.com	TCP	32825 > http [ACK] Seq=1592481969 Ack=163499588 Win=5840 Len=0 TSV=18752954 TSEQ=755

TCP is “stream oriented”

- data transmitted during connection viewed as one continuous stream
- bytes are consecutively numbered
- stream segmented into packets for transmittal

File deconstruction into sequenced packets



Packet's sequence number is the byte-stream number of the 1st data byte in the packet.

sequence number assignments:

1st packet – 0
2nd packet – 1000
3rd packet – 2000
etc

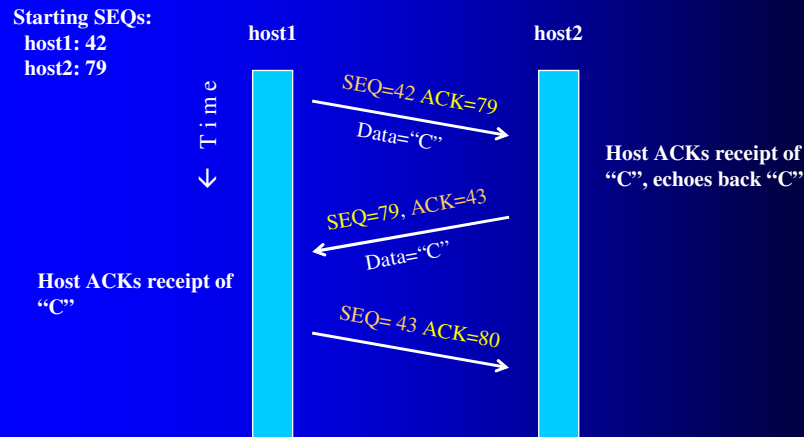
Sequence numbers

- relative to byte stream, not packet series
- initial sequence number randomly chosen
 - during connection setup handshake
 - actual byte count does not start from zero
- two number sequences
 - TCP carries 2 flows (full-duplex)
 - a separate sequence for each flow/direction

Acknowledgement number

- also byte-stream relative
- is sequence number next-expected from partner
- acknowledges receipt of all prior bytes
- therefore called “cumulative” acknowledgement
- acknowledgements are piggybacked
 - client-to-server acks ride with server-to-client data
 - server-to-client acks ride with client-to-server data

Numbering example*: ("C" keystroke in telnet)

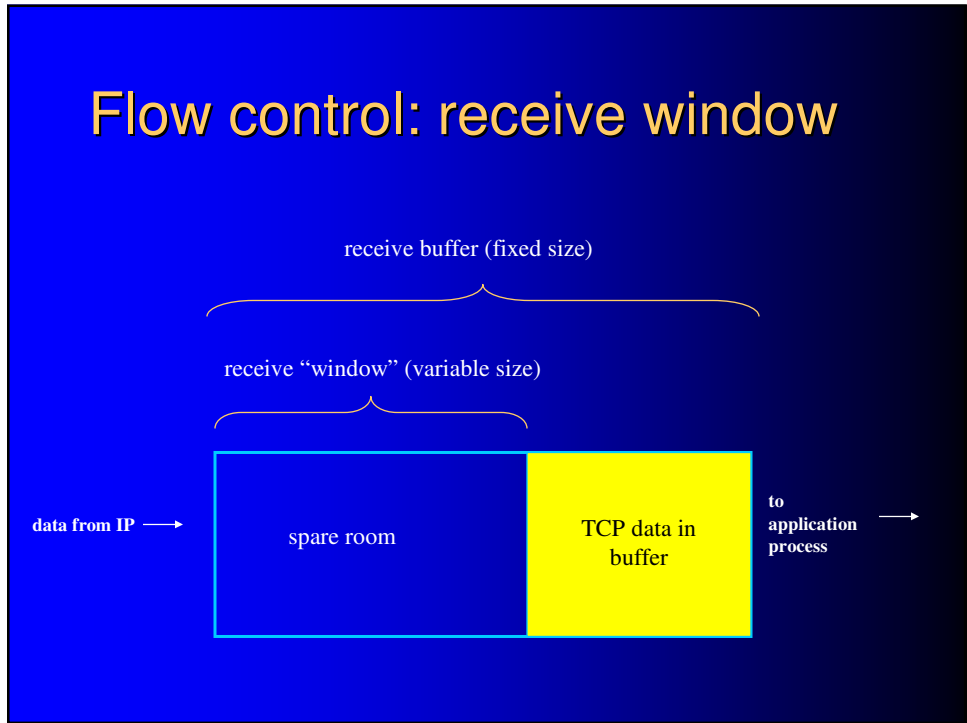


* Kurose & Ross, p. 234

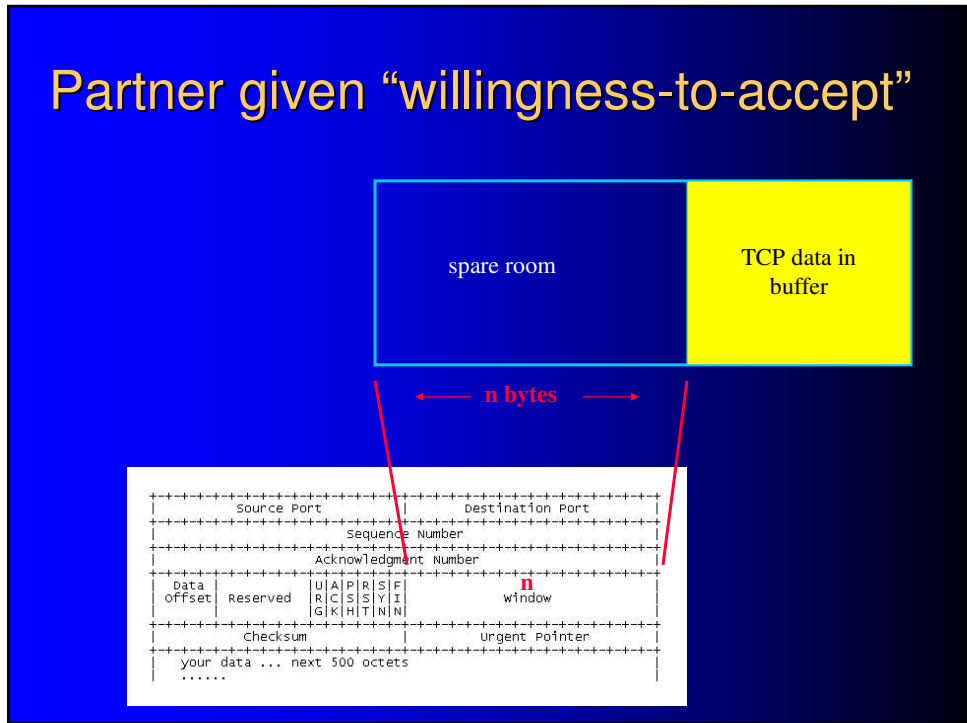
Traffic control

- flow control
 - adapt rate to partner's capacity
 - depends on spare room in partner's receive buffer
- congestion control
 - adapt rate to intervening path's capacity
 - depends on "just-about-anything"

Flow control: receive window



Partner given "willingness-to-accept"



Congestion control

- cap sent-but-unacknowledged data amount
- congestion limit can exceed flow limit
- vary the cap per perceived network congestion
 - cap more severely when packet loss rate rises
 - relax cap when it drops

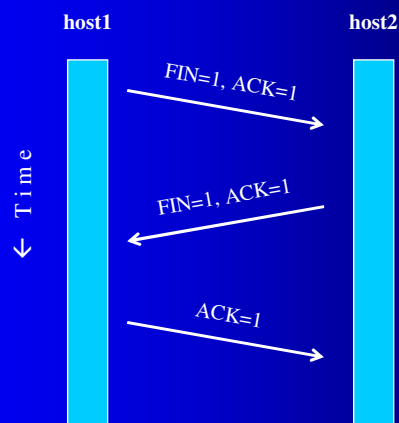
TCP Socket

- connection defined by socket pair
 - combination of IP address and port = socket
- client IP = 10.100.13.138
- client Port = 32825
 - client Socket = 10.100.13.138:32825
- server IP = 216.239.39.100
- server Port = 80 (http Default)
 - Server Socket = 216.239.39.100:80

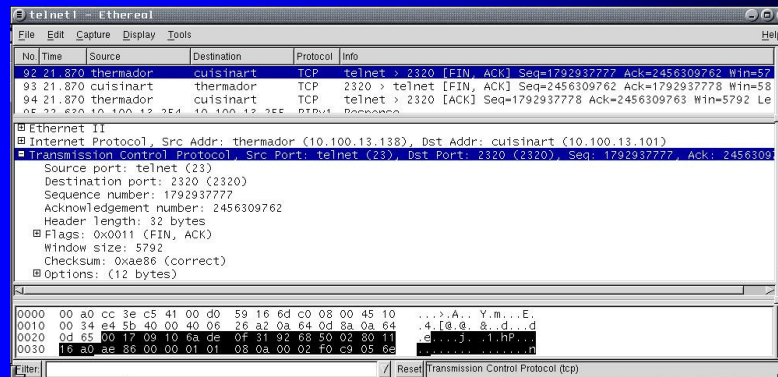
well-known TCP ports

- 21 - ftp
- 22 - ssh
- 23 - telnet
- 25 - smtp (Simple Mail Transport Protocol)
- 80 - http
- 110 - pop3 (Post Office Protocol)
- 123 - Network Time Protocol

TCP connection teardown



FIN/ACK



Biblio

- [Computer Networking](#), Kurose & Ross, Addison-Wesley, 2003; Chapter 3 “Transport Layer”
- “Telnet Protocol Specification,” RFC 854, 1983