


Computer Networks

Link Layer Protocols

Jianping Pan
Fall 2020

* Sorry, you will be muted during lecture for video recording, but you can always ask questions in chat box

Review

- 
- Application layer: HTTP, DNS
 - Transport layer: TCP, UDP
 - Network layer: IP/ICMP; RIP, OSPF, BGP
 - Link layer
 - frame control
 - byte stuffing, bit stuffing
 - error control
 - error detecting code, error correcting code

Error recovery

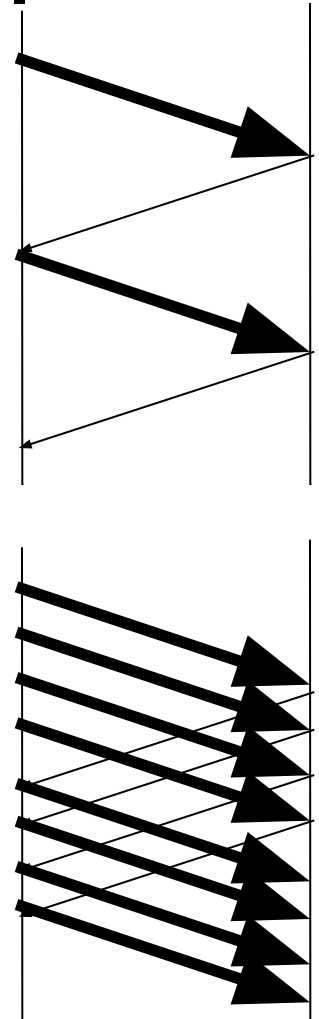
- Positive acknowledgment
 - cumulative acknowledgment
 - acknowledge packet x: acknowledge packets 1..x
 - when timeout, go-back-N
 - selective acknowledgment
 - acknowledge packet x: packet x is received OK
 - when timeout, selective repeat
- Negative acknowledgment
 - report: x is corrupted or *missing*

Today's topics

- Link layer
 - flow control
 - sliding-window-based flow control
 - representative link layer protocols
 - HDLC (High-level Data Link Control)
 - SLIP (Serial Line Internet Protocol)
 - PPP (Point-to-Point Protocol)

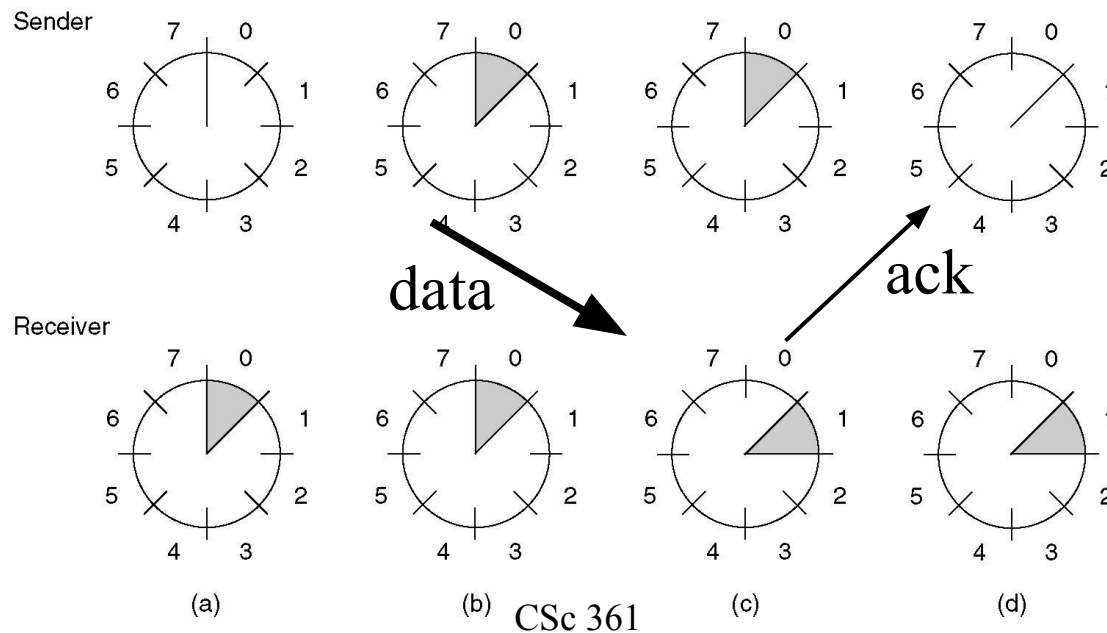
Transmission control

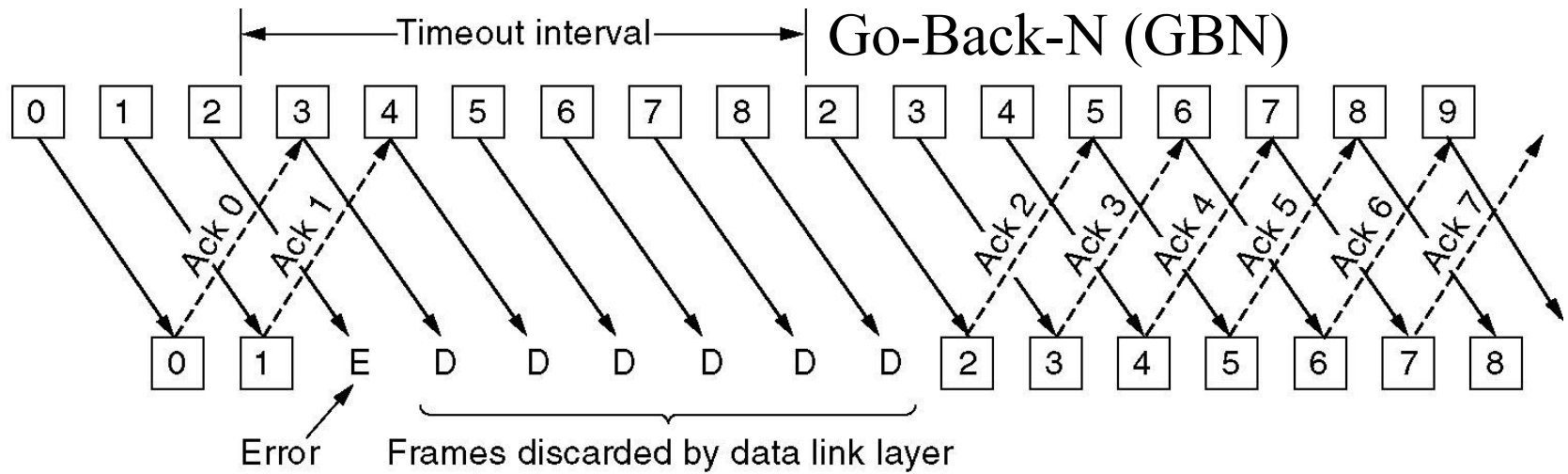
- Stop-and-wait
 - transmit the next packet
 - wait for its acknowledgment
 - or retransmit if timeout
 - low link utilization
- Pipelining
 - higher link utilization
 - issues at receiver
 - buffer limit? out-of-order packets?



Flow control

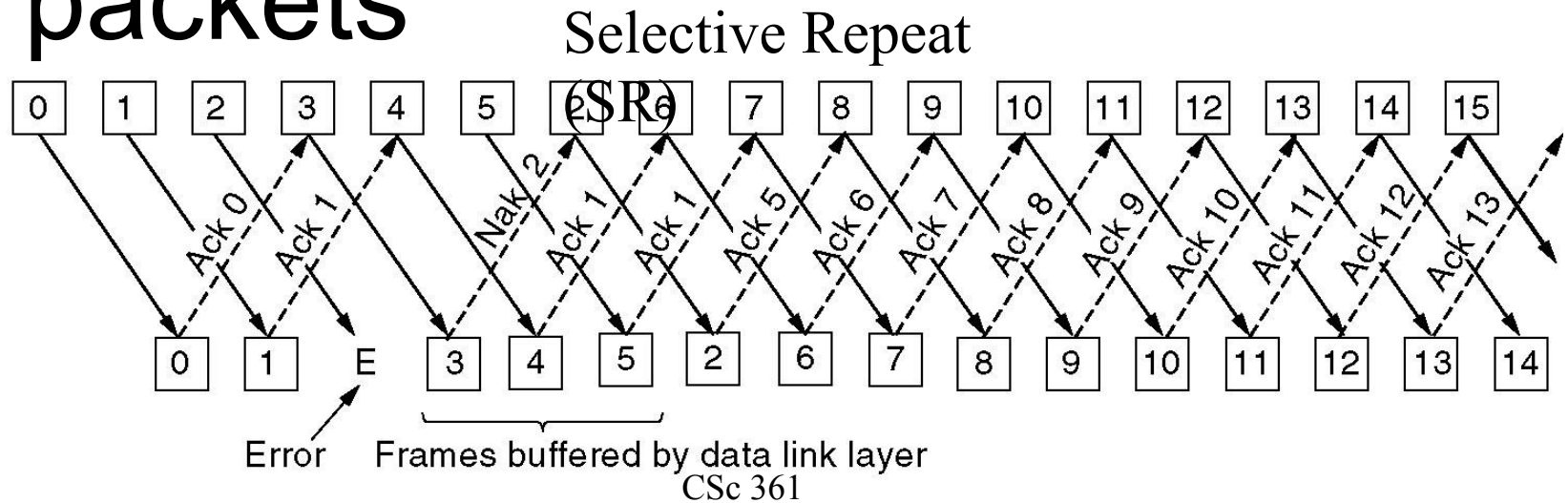
- Sliding window
 - window size < buffer size
 - e.g., window size = 1, i.e., stop-and-wait





Out-of-order packets

Time →
(a)

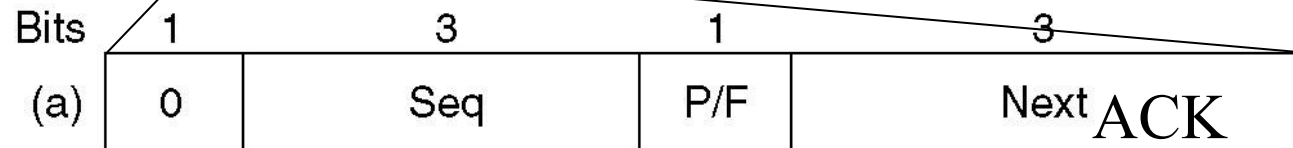


High-level Data Link Control

- HDLC (bit-stuffing) derived from SDLC



- Information



Pull/Final

- Supervisory



- Unnumbered



LSB CSc 361

8

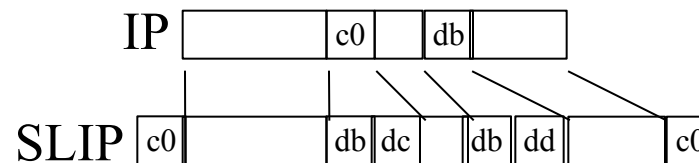
Q: flow, error control?

HDLC frames

- Information
 - sequence number
 - acknowledgment number
- Supervisory
 - acknowledgment number
 - Ready, Not Ready; Reject, Selective Reject
- Unnumbered
 - control frames
 - connectionless, unreliable service

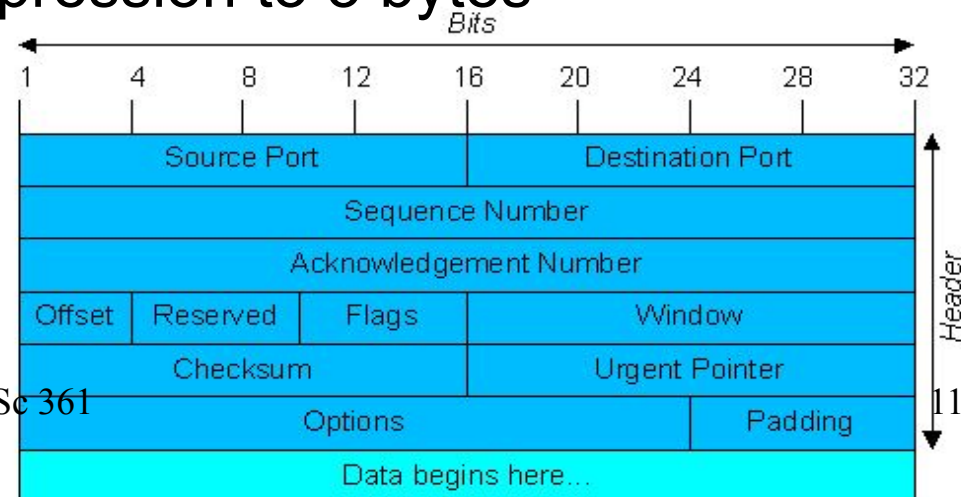
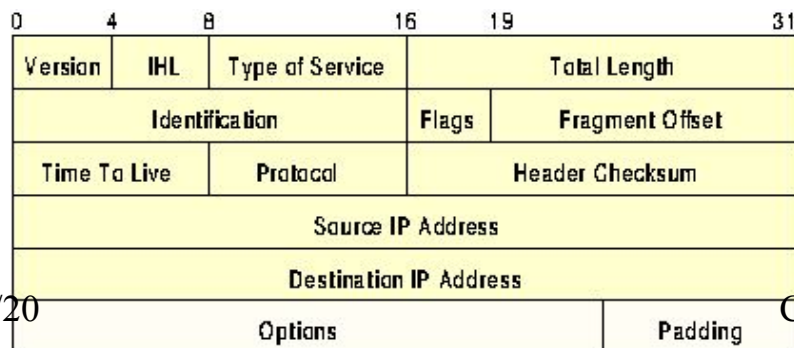
Serial-Line Internet Protocol

- SLIP/PLIP
 - very simple encapsulation for IP only
 - byte stuffing
 - flag: END (0xc0), escape: ESC (0xdb)
 - no flow/error control



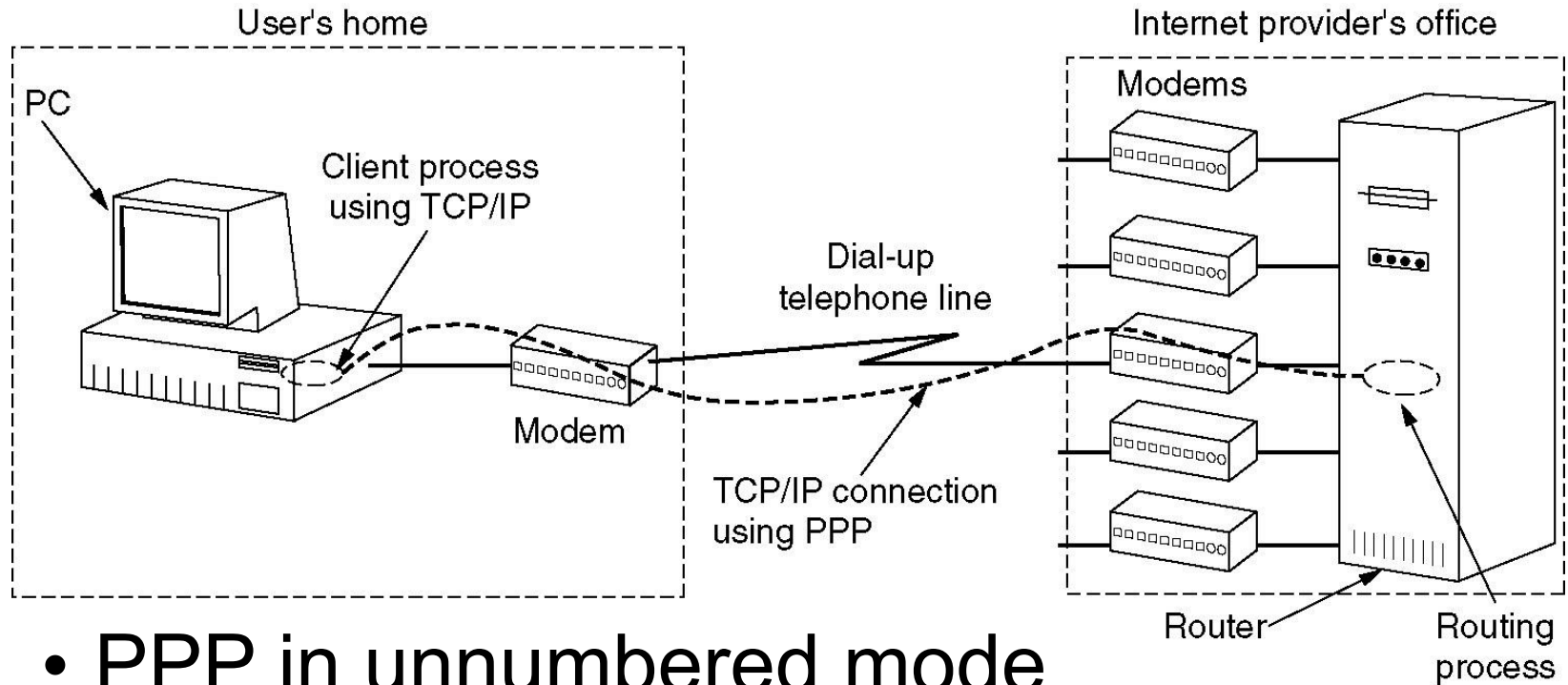
Compressed SLIP

- Compressed SLIP (CSLIP)
 - TCP/IP header: at least 20+20 bytes
 - some static, some predictable
 - Telnet data payload: 1 byte
 - delta encoding
 - Von Jacobson compression to 3 bytes

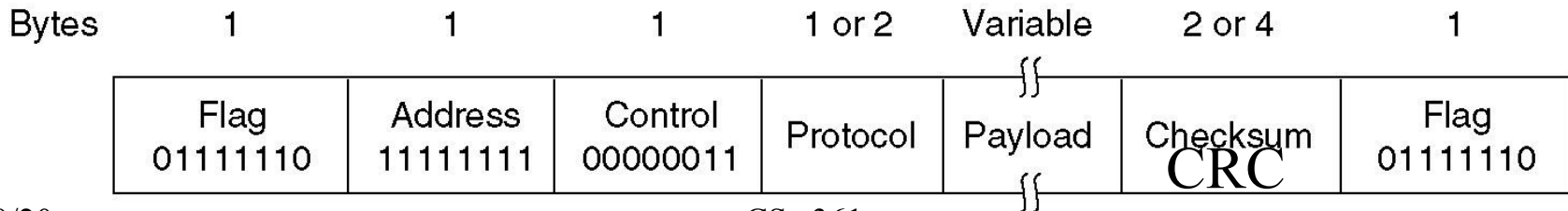


Point-to-Point Protocol

- PPP, PPPoE

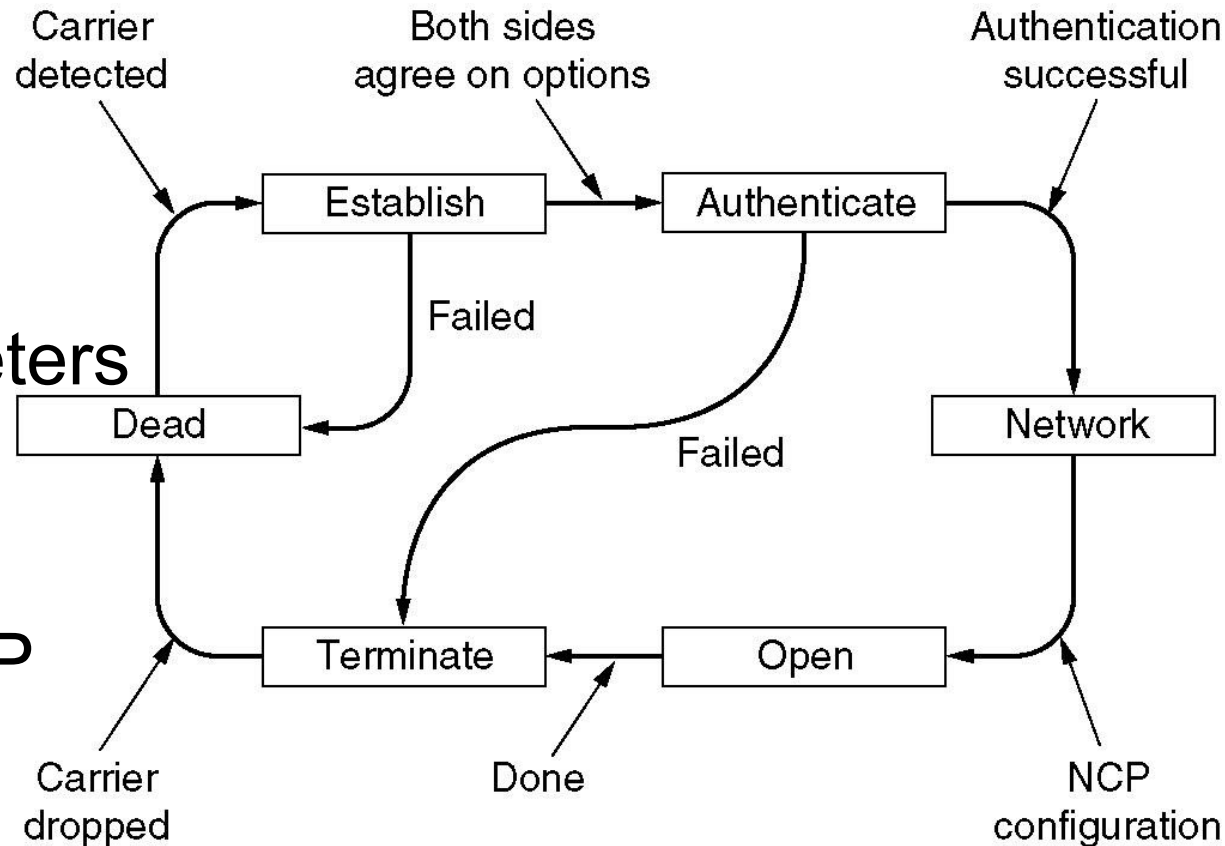


- PPP in unnumbered mode



PPP state diagram

- LCP
 - link parameters
- NCP
 - network parameters
- Authentication
 - PAP, CHAP, EAP



LCP frame types

Name	Direction	Description
Configure-request	$I \rightarrow R$	List of proposed options and values
Configure-ack	$I \leftarrow R$	All options are accepted
Configure-nak	$I \leftarrow R$	Some options are not accepted
Configure-reject	$I \leftarrow R$	Some options are not negotiable
Terminate-request	$I \rightarrow R$	Request to shut the line down
Terminate-ack	$I \leftarrow R$	OK, line shut down
Code-reject	$I \leftarrow R$	Unknown request received
Protocol-reject	$I \leftarrow R$	Unknown protocol requested
Echo-request	$I \rightarrow R$	Please send this frame back
Echo-reply	$I \leftarrow R$	Here is the frame back
Discard-request	$I \rightarrow R$	Just discard this frame (for testing)

LCP negotiation

- Maximum-Receiver-Unit (MRU)
- Authentication-Protocol
- Quality-Control
- Magic-Number (loopback detection)
- Protocol-Field-Compression (PFC)
- Address-and-Control-Field-Compression

IPCP

- Protocol
 - 0x8021: IPCP, i.e., NCP for IP
 - 0x0021: IP
- IPCP frame types
 - Configure-*, Terminate-*, Code-Reject
- Negotiation
 - compression: VJ compression (0x002d)
 - IP address: indicate or request one

PPP is more than just dialup

- PPPoA: PPP over ATM [RFC 2364]
 - some ADSL ISPs using ATM transport
- PPPoE: PPP over Ethernet [RFC 2516]
 - often used by DSL and cable modem ISPs
- Packet over SONET/SDH
 - mostly point-to-point backbone links
- L2TP: *Layer-2 Tunneling* Protocol (PPP/IP)
- PPTP: Point-to-Point TP (PPP+GRE)

This lecture

- Link layer
 - flow control
 - sliding window
 - HDLC, SLIP, PPP
 - frame, error and flow control
- Explore further
 - PPP: RFC 1661, 1332
 - <http://www.cs.uvic.ca/~pan/csc450f05/ppp.pdf>

Next lectures

- Medium access control

Q1 [1]

Please write down the last digit of your V00# number (say y), write down $b=y\%4$ (% means the modulo operation), and briefly describe and compare the b -th concepts pair only in your own words, not the bullet points copy-and-pasted from the slides, or those elsewhere.

0. Regular vs simultaneous open in TCP connection establishment

regular open initiated by the client, with three-way handshake of syn, syn|ack, ack to establish the connection
simultaneous open when both ends initiate the connection establishment by syn, syn, syn|ack and syn|ack

1. Active vs passive close in TCP connection release

active close initiated by the end that first sends out fin, and thus has to wait to ack the other end's fin to release
passive close is the end that first receives fin to ack, and can release the connection once its own fin is acked

2. Sender's vs receiver's window in TCP flow control

sender's window is the minimum of the receiver's window, sender's buffer size, and the congestion window
receiver's window is the receiver's available buffer size to accommodate data beyond the acknowledgment #

3. Flow control vs congestion control in TCP

flow control is to avoid a faster sender overwhelming a slow receiver's buffer size by window or rate control
congestion control is to avoid a sender or a group of senders overwhelming the router buffer inside network

Q2 [1]

Please write down the second last digit of your V00# number (say x), write down $a=x\%4$ (% means the modulo operation), and briefly describe and compare the a -th concepts pair only in your own words, not the bullet points copy-and-pasted from the slides, or those elsewhere.

0. Slow start vs congestion avoidance

slow start happens when the congestion window (cwnd) < slow-start threshold (ssthresh), doubling cwnd per round-trip time (rtt); congestion avoidance happens when cwnd > ssthresh, increasing cwnd one mss per rtt

1. Timeout retransmit vs fast retransmit

timeout retransmit happens when sender timeout happens, and fast retransmit happens when the sender receives enough duplicate acknowledgments, both ssthresh=cwnd/2 and cwnd=1 mss, followed by slow start

2. Fast retransmit vs fast recovery

fast retransmit happens when the sender receives enough duplicate acknowledgments, ssthresh=cwnd/2 and cwnd=1 mss, followed by slow start, but for fast recovery, cwnd=ssthresh=cwnd/2, followed by cong avoidance

3. Cumulative acknowledgment vs duplicate acknowledgment

cumulative acknowledgment acknowledges all consecutively received data, despite the triggering data packets duplicate acknowledgments are triggered when receiving the data packets after one or more missing packets

See example this Wednesday

Q3 [2]

Example: If your course instructor's V00# is V00123456, 56 is his last 2-digit number, 34 is his second last 2-digit number, 12 is his third last 2-digit number (or second first 2-digit number), and 00 is his fourth last 2-digit number (more often called the first 2-digit number), which seems always 00 for all UVic members now. Now he is sending his V00# in 56 34 and 12 as if in hexadecimal, and needs to calculate the TCP/IP checksum for it.

Please write down the last, second last and third last 2-digit numbers of your V00#, treat them as hexadecimal numbers, and calculate the TCP/IP checksum if you are sending out these three numbers in such a sequence.

pad 00 for an integer multiply of 16 bits, add with carry bit, wrap around the leading carry bit, and complement

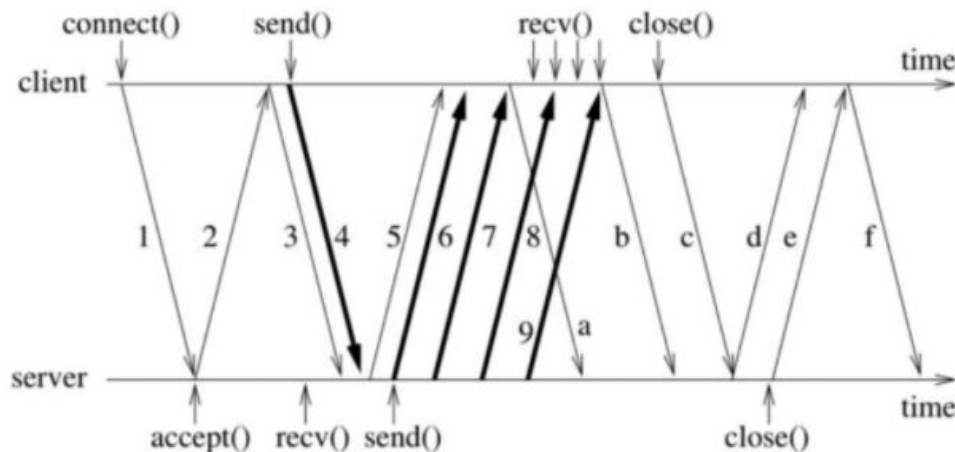
When does the TCP/IP checksum fail to detect bit errors in these three numbers, and why?

if an even number of bits that are an integer multiply of 16 bits away are flipped, the checksum will fail to detect

Q4 [4]

Example: If your course instructor's V00# is V00123456, 456 is his last 3-digit number, and 123 is his second last 3-digit number. His TCP initiator picks his last 3-digit number as the initial sequence number, and his TCP responder picks his second last 3-digit number as the initial sequence number. To simplify, they are in decimal.

Please write down the last and second last 3-digit numbers of your V00#, treat them as decimal numbers, and your TCP initiator picks your last 3-digit number as the initial sequence number, and your TCP responder picks your second last 3-digit number as the initial sequence number. Please answer questions based on your V00#.



The figure above shows the exchange of TCP packets between a client and a server application, where the client sends a request of 100 bytes (Packet #4) and the server sends a response of 4,000 bytes (Packet #6, #7, #8 and #9 of 1,000 bytes each). The total receiver buffer size at each side is 4,000 bytes. System calls triggering the exchange of TCP packets are also listed for your convenience (recv() buffer size is 1,000 bytes). Please write down the Set Flags (e.g., SYN, ACK, PSH, FIN, etc), Sequence Number, Data Length, Acknowledgment Number, Window Size, and the purpose of each packet from Packet 1 to Packet f. E.g., for

Packet 2, Set Flags: SYN|ACK; SeqNo: your second last 3-digit number; DataLen: 0; AckNo: your last 3-digit number + 1; WinSize: 4000; Purpose: acknowledge Packet 1 and synchronize from the server.

the last 3-digit number is yyy, and the second last 3-digit number is xxx, according to students v00#

- 1. syn; yyy; 0; - (undefined); 4000; syn from the client*
- 2. syn|ack; xxx; 0; yyy+1; 4000; ack pkt 1 and syn from the server*
- 3. ack; yyy+1; 0; xxx+1; 4000; ack pkt 2*
- 4. ack|psh; yyy+1; 100; xxx+1; 4000; the request from the client*
- 5. ack; xxx+1; 0; yyy+101; 4000; ack pkt 4*
- 6. ack; xxx+1; 1000; yyy+101; 4000; the first response packet from the server*
- 7. ack; xxx+1001; 1000; yyy+101; 4000; the second response packet from the server*
- 8. ack; xxx+2001; 1000; yyy+101; 4000; the third response packet from the server*
- 9. ack|psh; xxx+3001; 1000; yyy+101; 4000; the fourth/last response packet from the server*
 - a. ack; yyy+101; 0; xxx+2001; 2000; ack pkt 6 and 7 with reduced window size (before the client's recv())*
 - b. ack; yyy+101; 0; xxx+4001; 4000; ack pkt 8 and 9 with updated window size (after the client's recv()s)*
 - c. ack|fin; yyy+101; 0; xxx+4001; 4000; fin from the client*
 - d. ack; xxx+4001; 0; yyy+102; 4000; ack pkt c*
 - e. ack|fin; xxx+4001; 0; yyy+102; 4000; fin from the server*
 - f. ack; yyy+102; 0; xxx+4002; 4000; ack pkt e*

Further, based on the last digit (y) of your V00# and $b=y\%4$, and if Packet 6+b is lost in the network, what will TCP respond to the situation? Please consider that TCP has integrated flow, error and control congestion.

*if packet 6 is lost, packet 7, 8 and 9 can trigger enough duplicate acknowledgments for fast retransmit/covery
if packet 7, 8 or 9 is lost, there are not enough duplicate acknowledgments, so timeout retransmit will happen*