

# Transport Reliability: Connection Management and TCP

10/14/2009

1

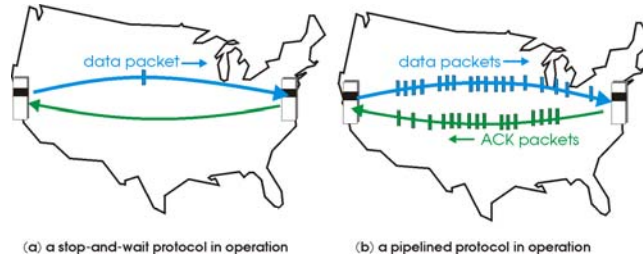
## Admin.: PS2

proj-sol:		proj:	
129 400 3045 FishThread.java		129 400 3045 FishThread.java	
388 1457 12873 Node.java		341 1301 11313 Node.java	
51 167 1145 PingRequest.java		51 167 1145 PingRequest.java	
83 250 2106 SimpleTCPSockSpace.java			
181 605 5248 TCPManager.java		50 128 909 TCPManager.java	
889 3088 26381 TCPSock.java		132 460 3146 TCPSock.java	
60 149 1316 TCPSockID.java			
123 382 3866 TransferClient.java		123 382 3866 TransferClient.java	
147 500 5059 TransferServer.java		147 500 5059 TransferServer.java	
2051 6998 61039 total		973 3338 28483 total	

2

## Recap: Sliding Window Protocols

- Basic idea: using **pipelining**: to make better use of link bandwidth, sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

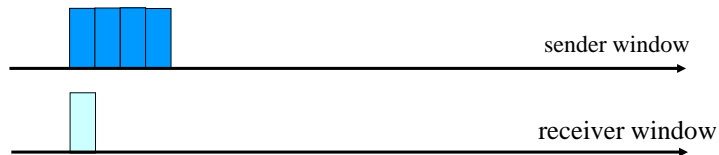


- Two generic forms of pipelined protocols: **go-Back-N**, **selective repeat**

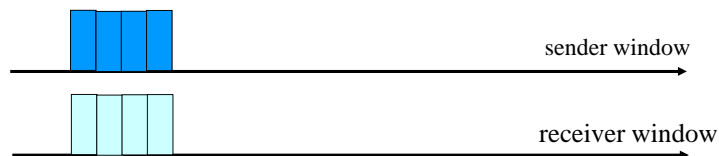
3

## Window Location

- Go-back-n (GBN)



- Selective repeat (SR)



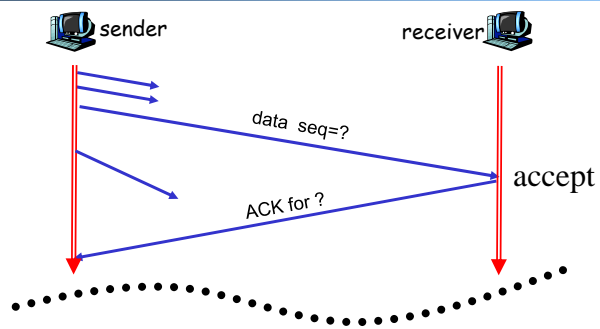
4

## Sliding Window Protocols: Go-back-n and Selective Repeat

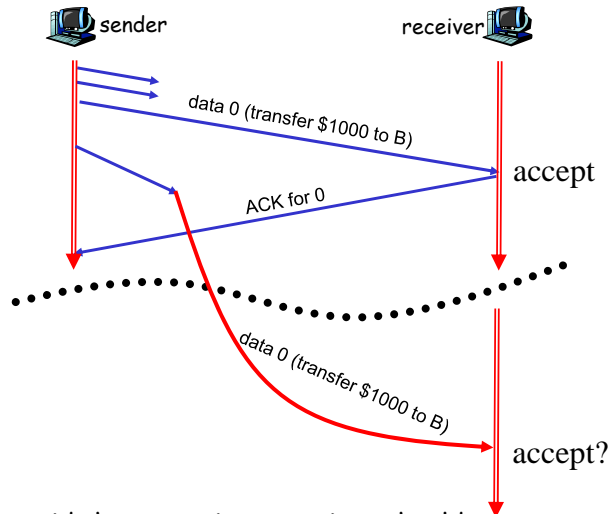
	Go-back-n	Selective Repeat
Buffer size at receiver	1	W
Relationship between M (the number of seq#) and W (window size)	$M > W$	$M \geq 2W$
data bandwidth: sender to receiver (avg. number of times a pkt is transmitted)	Less efficient $\frac{1-p+pw}{1-p}$	More efficient $\frac{1}{1-p}$

p: the loss rate of a packet; M: number of seq# (e.g., 3 bit M = 8); W: window size <sup>5</sup>

## Question: What is Initial Seq#?



## Question: What is Initial Seq#?



- To avoid the scenario, a receiver should *not* accept a seq# unless it is sure that the seq# is not a duplicate or reordered

7

## Reordering and Duplication

- There are many solutions each with its own pitfalls
- Internet solution: for each connection (sender-receiver pair), ensuring that two identically numbered packets are never outstanding at the same time
  - network bounds the life time of each packet
  - a sender will not reuse a seq# before it is sure that all packets with the seq# are purged from the network
    - seq. number space should be large enough to not limit transmission rate

8

## Outline

- Review
- Reliable data transfer
  - perfect channel
  - channel with bit errors
  - channel with bit errors and losses
  - sliding window: reliability with throughput
  - connection management

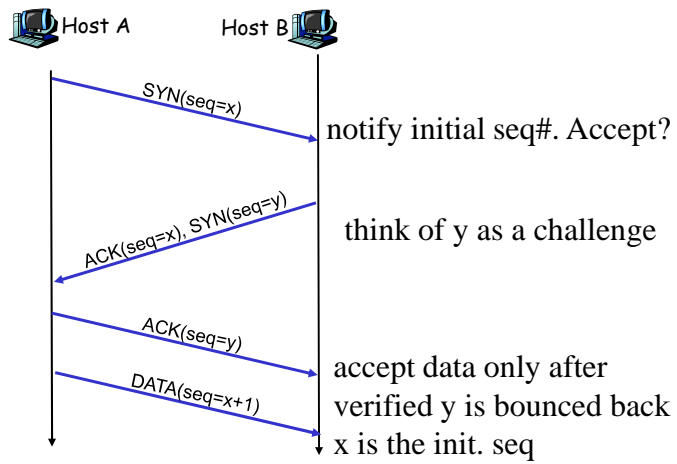
9

## Connection Management: Objectives

- Agree on initial sequence numbers
  - a sender will not reuse a seq# before it is sure that all packets with the seq# are purged from the network
    - the network guarantees that a packet too old will be purged from the network: network bounds the life time of each packet
  - needs connection setup so that the sender tells the receiver initial seq#
- Agree on other initial parameters

10

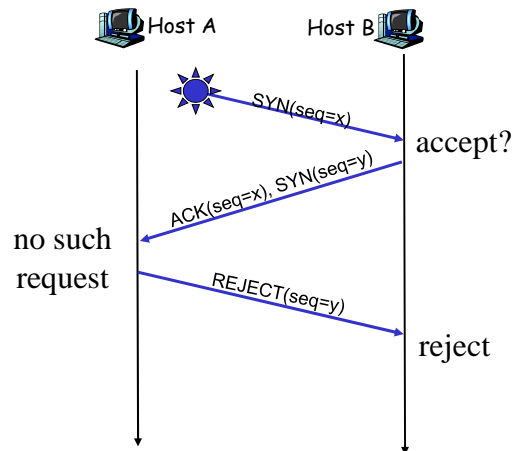
## Three Way Handshake (TWH) [Tomlinson 1975]



SYN: indicates connection setup

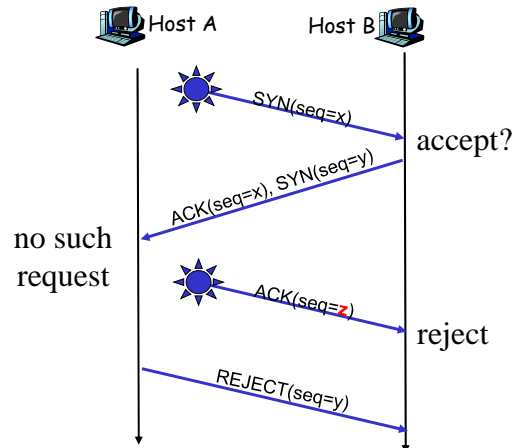
11

## Scenarios with Duplicate Request



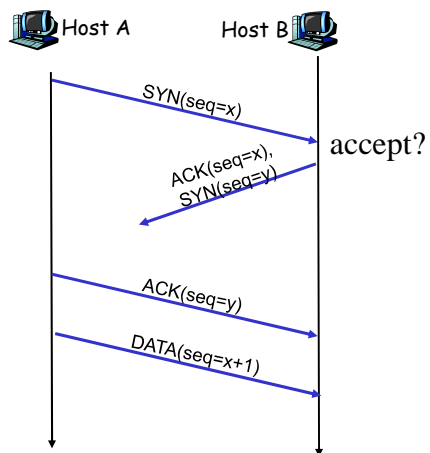
12

## Scenarios with Duplicate Request



13

## Scenarios with TCP Spoofing

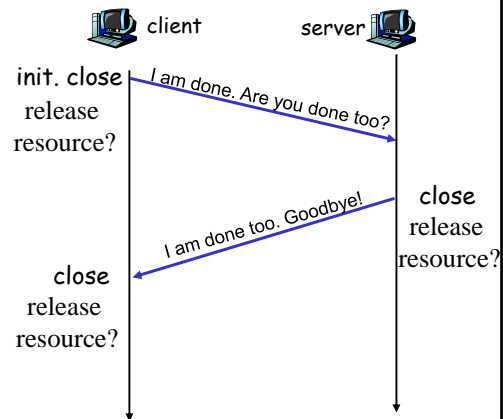


<http://www.juniper.net/security/auto/vulnerabilities/vuln670.html>

14

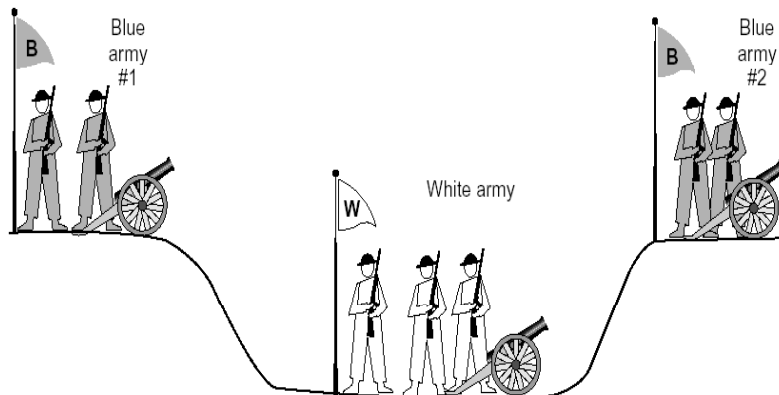
## Connection Close

- Why connection close?
  - so that each side can release resource and remove state about the connection (do not want dangling socket)



15

## General Case: The Two-Army Problem

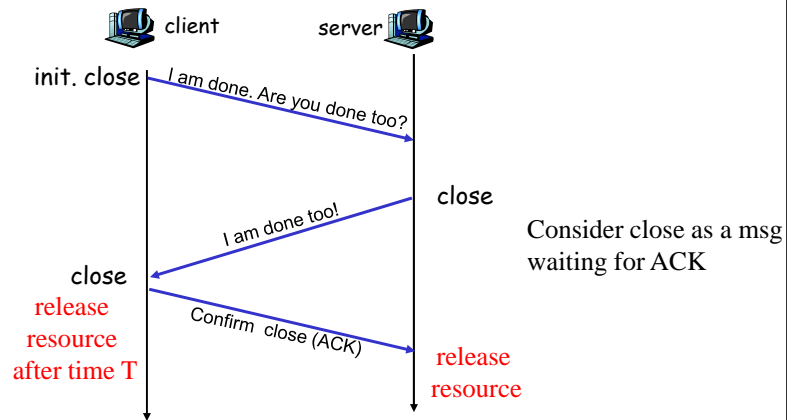


The gray (blue) armies need to agree on whether or not they will attack the white army. They achieve agreement by sending messengers to the other side. If they both agree, attack; otherwise, no. Note that a messenger can be captured!

16

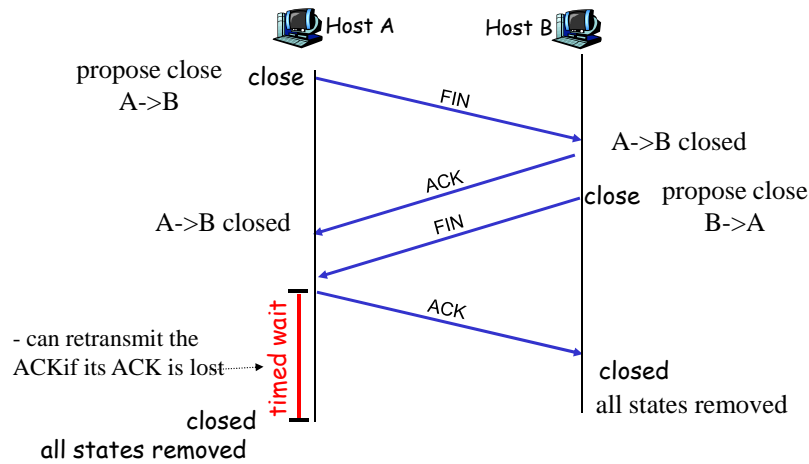
## Connection Close

No message exchange can reach agreement. Need other decision format.



17

## Four Way Teardown



18

## A Summary of Questions

- ❑ How to improve the performance of rdt3.0?
  - ✓ sliding window protocols
- ❑ What if there are duplication and reordering?
  - ✓ network guarantee: max packet life time
  - ✓ transport guarantee: not reuse a seq# before life time
  - ✓ seq# management and connection management
- ❑ How to determine the "right" parameters?

19

## A Summary of Questions

- ❑ How to improve the performance of rdt3.0?
  - ✓ sliding window protocols
- ❑ What if there are duplication and reordering?
  - ✓ network guarantee: max packet life time
  - ✓ transport guarantee: not reuse a seq# before life time
  - ✓ seq# management and connection management
- ❑ How to determine the "right" parameters?

20

## Outline

---

- Recap
- Reliable data transfer
  - *TCP reliability*

21

## TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

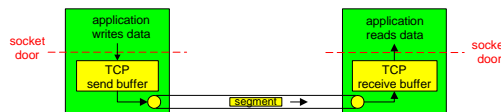
---

- Point-to-point reliability: one sender, one receiver
  
- Flow controlled and congestion controlled

22

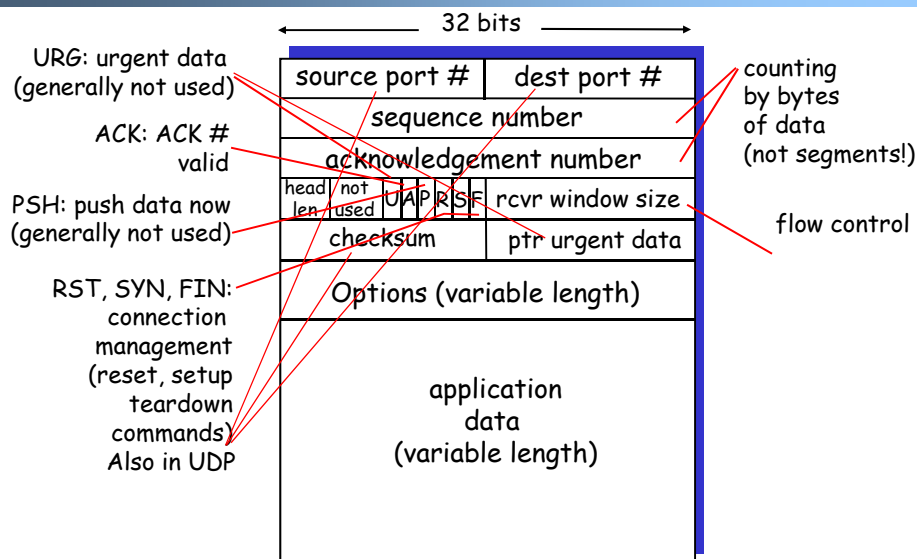
# TCP Reliable Data Transfer

- **Connection-oriented:**
  - connection management
    - setup (exchange of control msgs) init's sender, receiver state before data exchange
    - close
- **Full duplex data:**
  - bi-directional data flow in same connection
- **A sliding window protocol**
  - a combination of go-back-n and selective repeat:
    - send & receive buffers
    - cumulative acks
    - TCP uses a single retransmission timer
    - do not retransmit all packets upon timeout



23

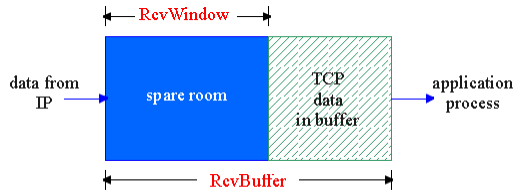
# TCP Segment Structure



24

# Flow Control

- receive side of a connection has a receive buffer:



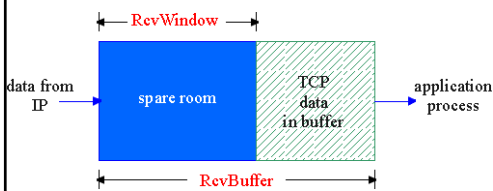
- app process may be slow at reading from buffer

## flow control

sender won't overflow receiver's buffer by transmitting too much, too fast

- speed-matching service: matching the send rate to the receiving app's drain rate

# TCP Flow Control: How it Works



- spare room in buffer = **RcvWindow**

source port #		dest port #	
sequence number			
acknowledgement number			
head len	not used	U	A
P	R	S	F
checksum		rcvr window size	
ptr urgent data			
Options (variable length)			
application data (variable length)			

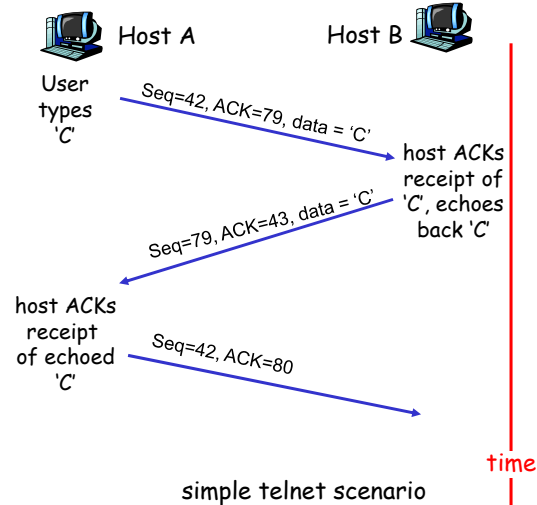
## TCP Seq. #'s and ACKs

### Seq. #'s:

- byte stream "number" of first byte in segment's data

### ACKs:

- seq # of next byte **expected** from other side
- cumulative ACK



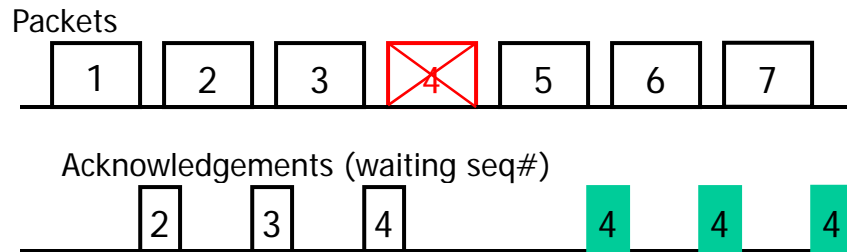
27

## Fast Retransmit

- Problem: Timeout period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - resend segment before timer expires

28

## Triple Duplicate Ack



29

## Fast Retransmit:

```
event: ACK received, with ACK field value of y
  if (y > SendBase) {
    ...
    SendBase = y
    if (there are currently not-yet-acknowledged segments)
      start timer
    ...
  }
  else {
    increment count of dup ACKs received for y
    if (count of dup ACKs received for y = 3) {
      resend segment with sequence number y
    }
    ...
  }
```

a duplicate ACK for  
already ACKed segment

fast retransmit

30

# TCP: reliable data transfer

Simplified  
TCP  
sender

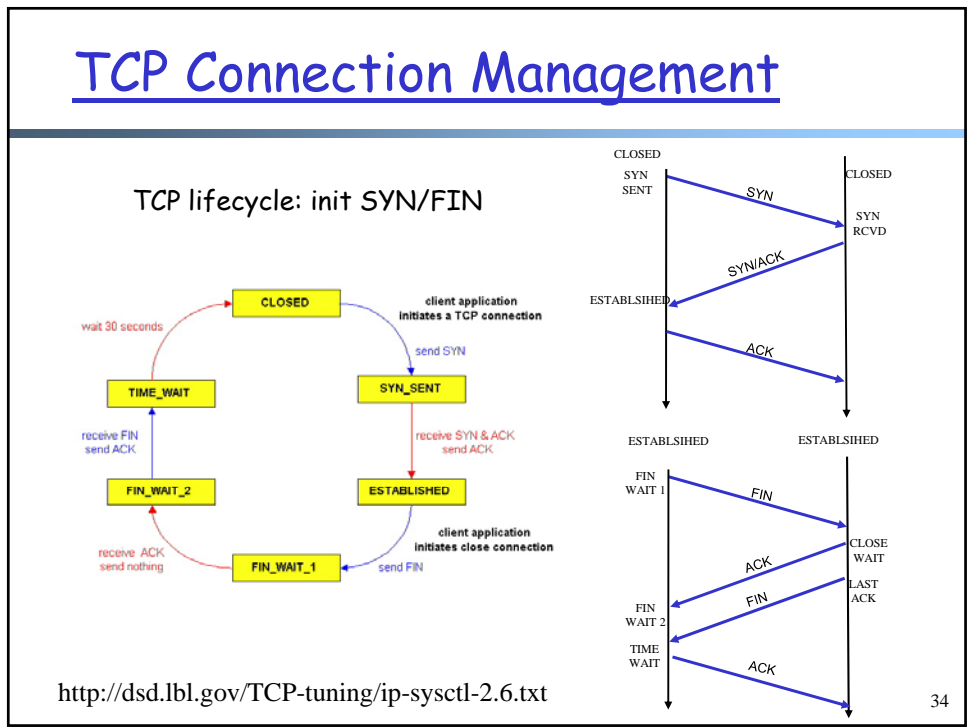
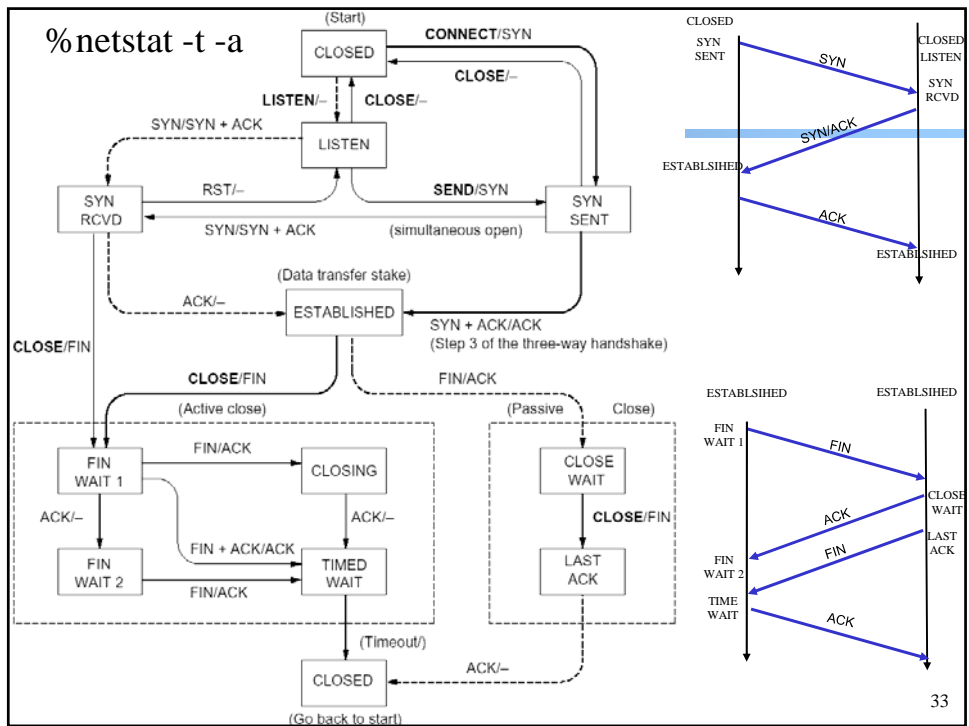
```

00 sendbase = initial_sequence number agreed by TWH
01 nextseqnum = initial_sequence number by TWH
02 loop (forever) {
03   switch(event)
04     event: data received from application above
05       if (window allows send)
06         create TCP segment with sequence number nextseqnum
07         if (no timer) start timer
08         pass segment to IP
09         nextseqnum = nextseqnum + length(data)
10       else put packet in buffer
11     event: timer timeout for sendbase
12       retransmit segment
13       compute new timeout interval
14       restart timer
15     event: ACK received, with ACK field value of y
16       if (y > sendbase) { /* cumulative ACK of all data up to y */
17         cancel the timer for sendbase
18         sendbase = y
19         if (no timer and packet pending) start timer for new sendbase
20         while (there are segments and window allow)
21           sent a segment;
22       }
23     else { /* y==sendbase, duplicate ACK for already ACKed segment */
24       increment number of duplicate ACKs received for y
25       if (number of duplicate ACKS received for y == 3) {
26         /* TCP fast retransmit */
27         resend segment with sequence number y
28         restart timer for segment y
29       }
30   } /* end of loop forever */

```

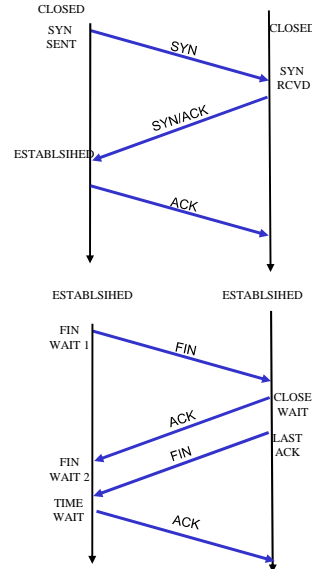
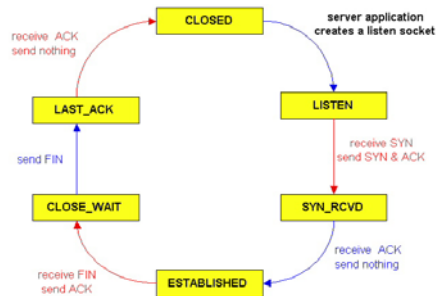
## TCP Receiver ACK Generation [RFC 1122, RFC 2581]

Event at Receiver	TCP Receiver Action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate send ACK, provided that segment starts at lower end of gap



# TCP Connection Management

TCP lifecycle: wait for SYN/FIN



35

## A Summary of Questions

- ❑ How to improve the performance of rdt3.0?
  - ✓ sliding window protocols
- ❑ What if there are duplication and reordering?
  - ✓ network guarantee: max packet life time
  - ✓ transport guarantee: not reuse a seq# before life time
  - ✓ seq# management and connection management
- ❑ How to determine the "right" parameters?

36