

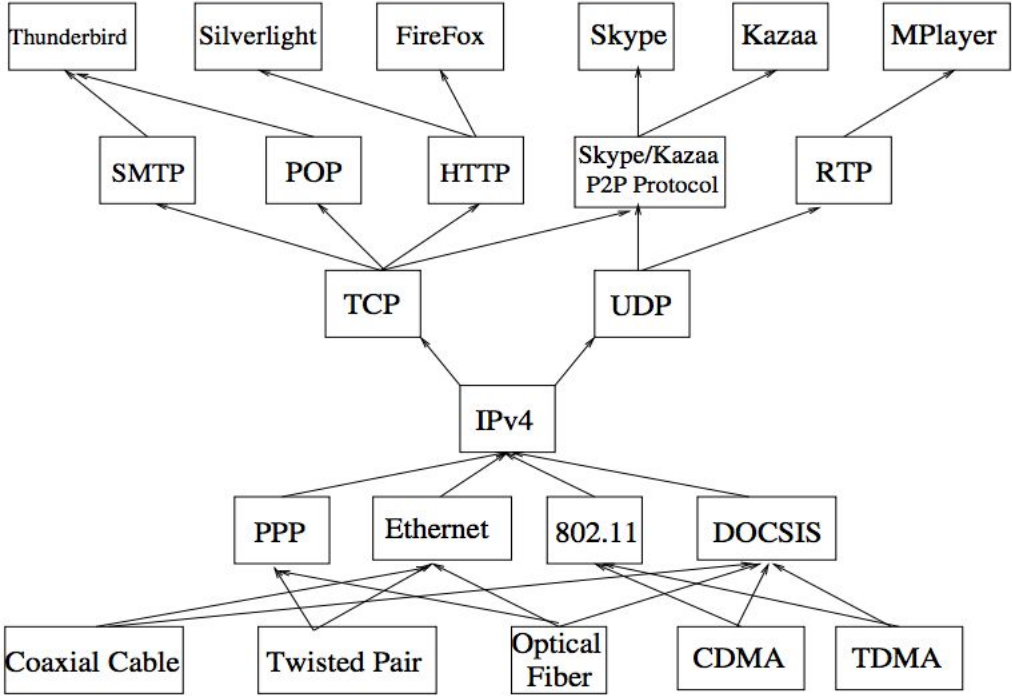
# **CSE 410/565: Computer Security**

Instructor: Dr. Ziming Zhao

# The Postal Analogy

- A- Write a 20 page letter to a foreign country.
- P- Translate the letter so the receiver can read it.
- S- Insure the intended recipient can receive letter.
- **T- Separate and number pages. Like registered mail, tracks delivery and requests another package if one is “lost” or “damaged” in the mail.**
- N- Postal Center sorting letters by zip code to route them closer to destination.
- D- Local Post Office determining which vehicles to deliver letters.

# TCP & UDP



# Transport Layer

- The transport layer is located between the network layer and the application layer. So, it is responsible for providing services to the application layer; it receives services from the network layer.
- Extend network layer (IP)'s service from **host-to-host delivery** to **process-to-process delivery**.

## Processes communicating

- Within same host, two processes communicate using **inter-process communication** (defined by OS).
- Processes in different hosts communicate by exchanging messages

# Processes communicating

- Client process: process that initiates communication
- Server process: process that waits to be contacted
- Applications with P2P architectures have client processes & server processes

# Protocol Field in an IP Datagram

bit # 0                                      7 | 8                                      15 | 16                                      23 | 24                                      31

version	header length	DS	ECN	total length (in bytes)			
Identification			0	D	M	Fragment offset	
time-to-live (TTL)		protocol		header checksum			
source IP address							

Protocol Number	Protocol Name	Abbreviation
1	Internet Control Message Protocol	ICMP
2	Internet Group Management Protocol	IGMP
6	Transmission Control Protocol	TCP
17	User Datagram Protocol	UDP
41	IPv6 encapsulation	ENCAP
89	Open Shortest Path First	OSPF
132	Stream Control Transmission Protocol	SCTP



# Addressing Processes

- To receive messages, a process must have **an identifier**

## TCP Header

		TCP Header																															
Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset	Reserved 0 0 0			N S	C W R	E C E	U R G	A C K	P R H	R S T	S S N	F I N	Window Size																		
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if <i>data offset</i> > 5. Padded at the end with "0" bytes if necessary.)																															
...	...	...																															



# Addressing Processes

- To receive messages, a process must have **an identifier**

## UDP Header

Offsets		UDP Header																															
Octet	Bit	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

# Addressing Processes

- Identifier includes both **IP address** and **port** numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - DNS: 53

# Transmission Control Protocol

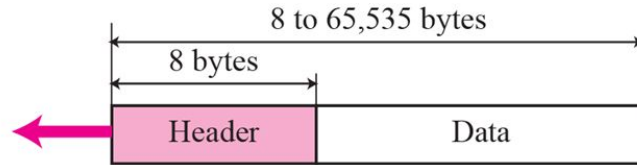
- **Connection-oriented:** setup required between client and server processes
- **Reliable** transport between sending and receiving process
- Flow control: sender won't overwhelm receiver
- Congestion control: throttle sender when network overloaded
- Streaming: Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.
- Does not provide: timing, minimum throughput guarantees

# User Datagram Protocol

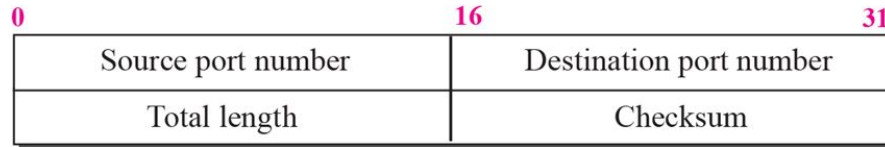
- **Unreliable** data transfer between sending and receiving process
- Datagrams – Packets are sent individually. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.
- Does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee
- Lightweight – There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.

# UDP

- Lightweight communication between processes
  - Avoid overhead and delays of ordered, reliable delivery
  - Send messages to and receive them from a socket



a. UDP user datagram



b. Header format

# Why UDP?

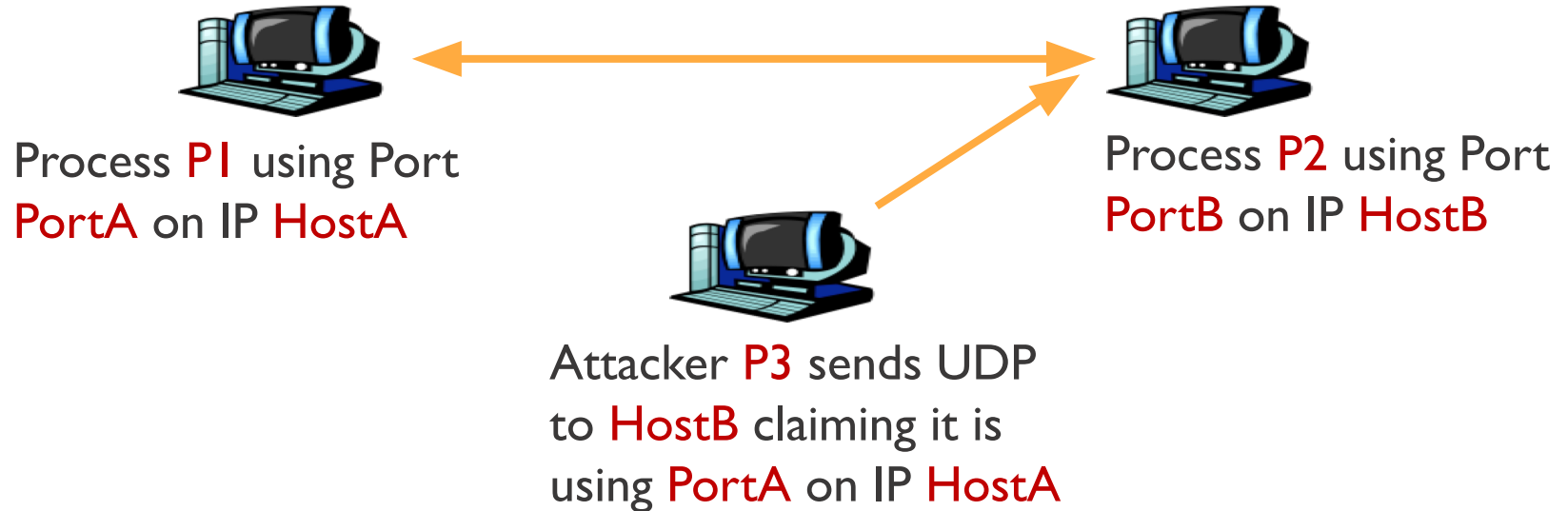
- No delay for connection establishment
  - As soon as an application process sends into transport layer
  - avoids introducing any unnecessary delays
- No connection state
  - No allocation of buffers, parameters, sequence #s, etc.
  - Easier to handle many active clients at once
- Small packet header overhead
  - Only eight-bytes long

# Who uses UDP?

- Multimedia streaming
  - Retransmitting lost/corrupted packets is not worthwhile
  - By the time the packet is retransmitted, it's too late
  - E.g., telephone calls, video conferencing, gaming
- Simple query protocols like Domain Name System
  - Overhead of connection establishment is overkill
  - Easier to have application retransmit if needed

# UDP Spoofing - IP Spoofing

- As easy as IP spoofing, since UDP does not add any other protection





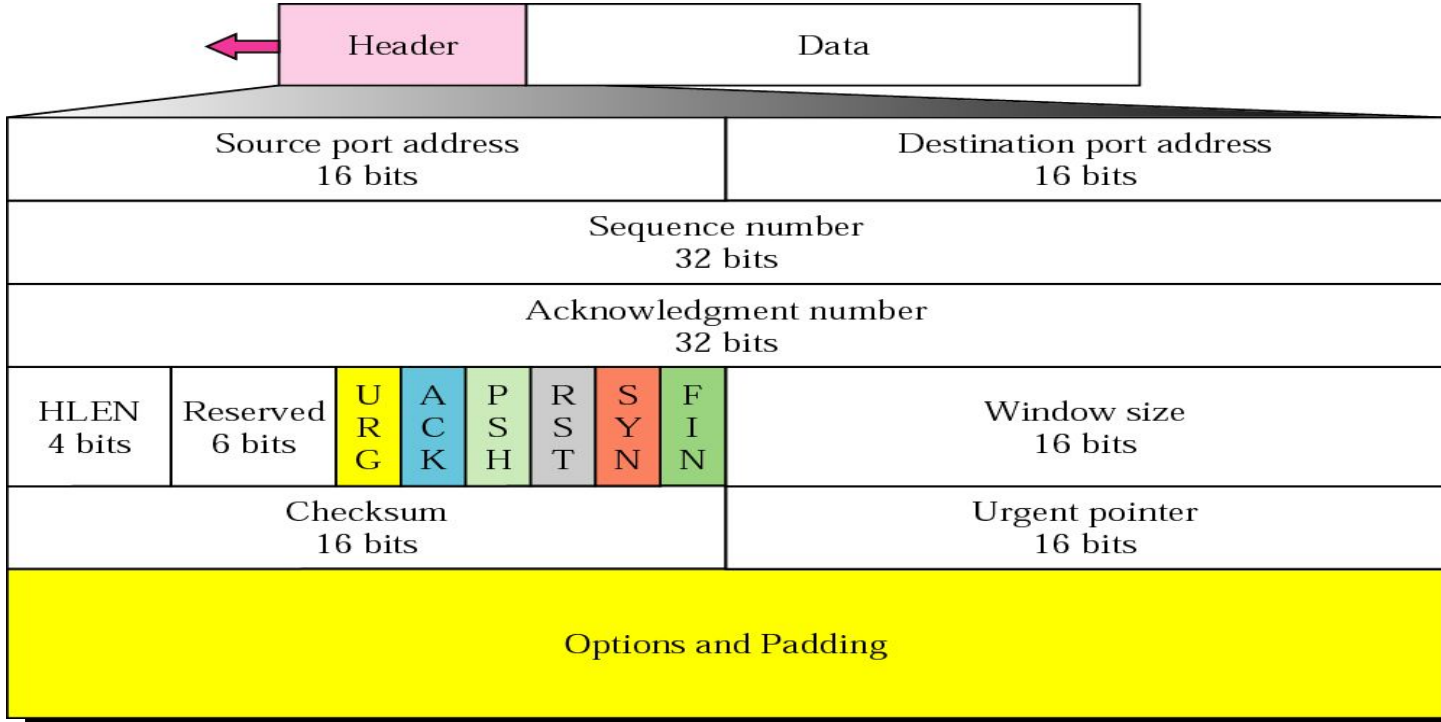
# TCP

- Connection-oriented
  - Explicit set-up and tear-down of TCP session/connection
- Reliable, in-order delivery
  - Checksums to detect corrupted data
  - Acknowledgments & retransmissions for reliable delivery
  - Sequence numbers to detect losses and reorder data

# TCP

- Stream-of-bytes
  - Application sends and receives a stream of bytes, not messages
- Flow control
  - Prevent overflow of the receiver's buffer space
- Congestion control
  - Adapt to network congestion for the greater good

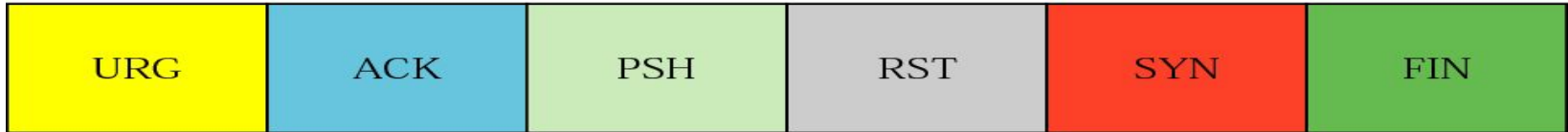
# TCP Header



# Six Fields

URG: Urgent pointer is valid  
ACK: Acknowledgment is valid  
PSH: Request for push

RST: Reset the connection  
SYN: Synchronize sequence numbers  
FIN: Terminate the connection



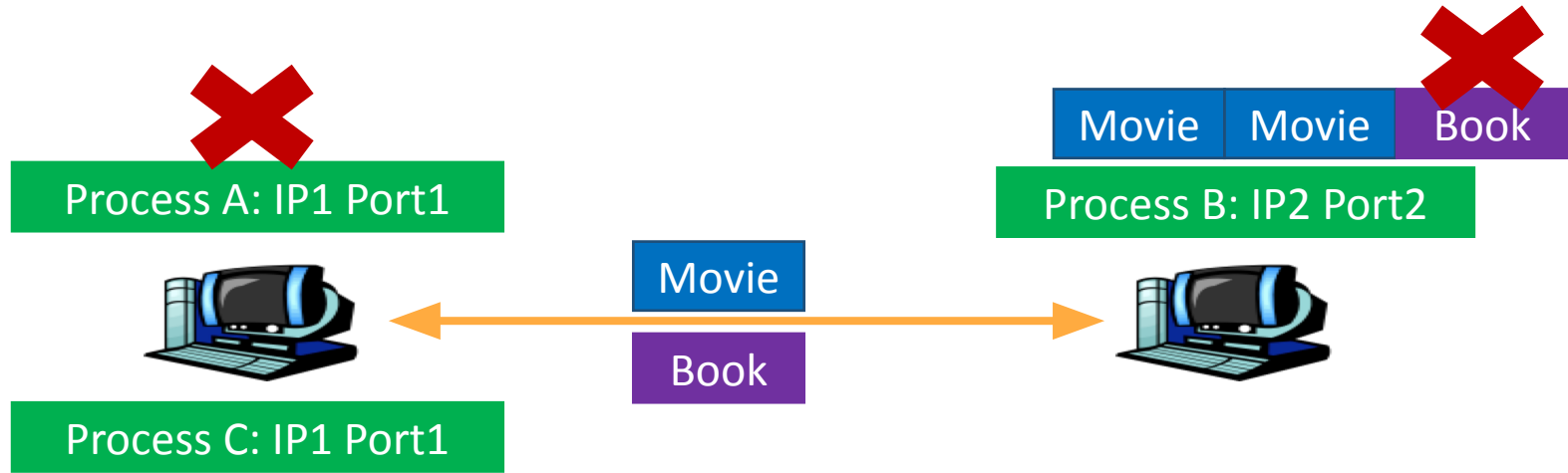
# Data loss and reordering

- Sequence Number
- Acknowledge Number

# Sequence and Acknowledge Number

- **Sequence Number**: The bytes of data being transferred in each connection are numbered by TCP.
- The numbering starts with an arbitrarily generated number.
- **Acknowledge Number**: The value of the acknowledgment field in a segment defines **the number of the next byte a party expects to receive**.

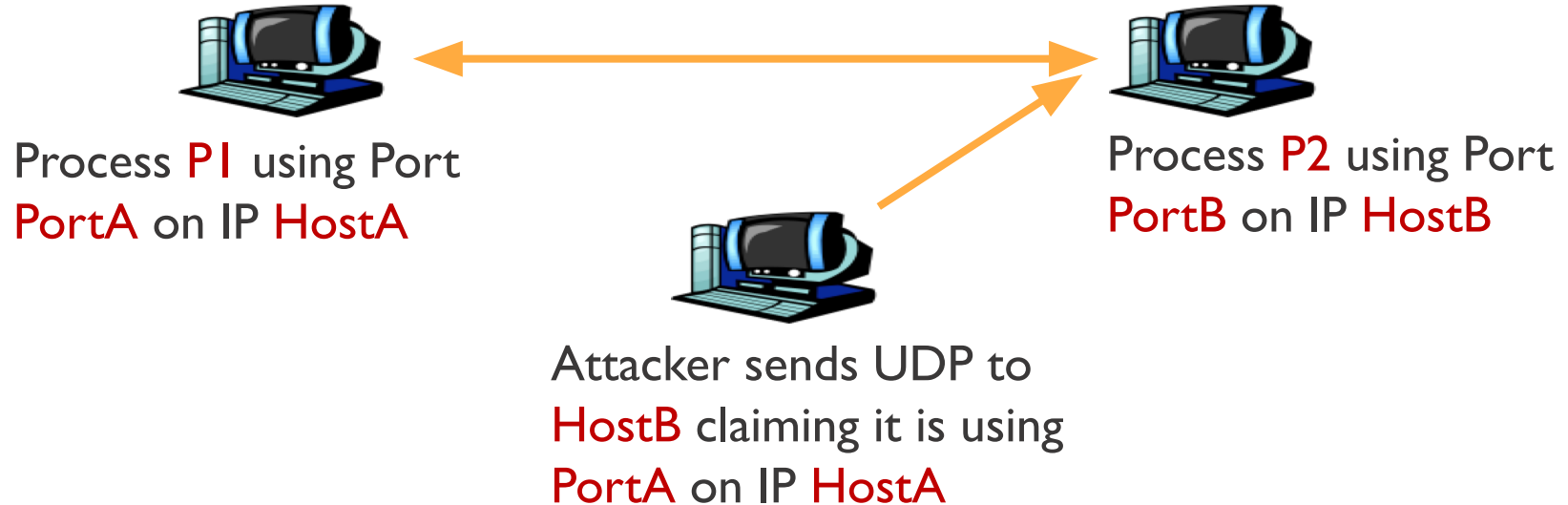
# Why Random Sequence Number?



- IP addresses and port #s uniquely identify a connection
- Eventually, though, these port #s do get used again
- There is a chance an old packet is still in flight and might be associated with the new connection

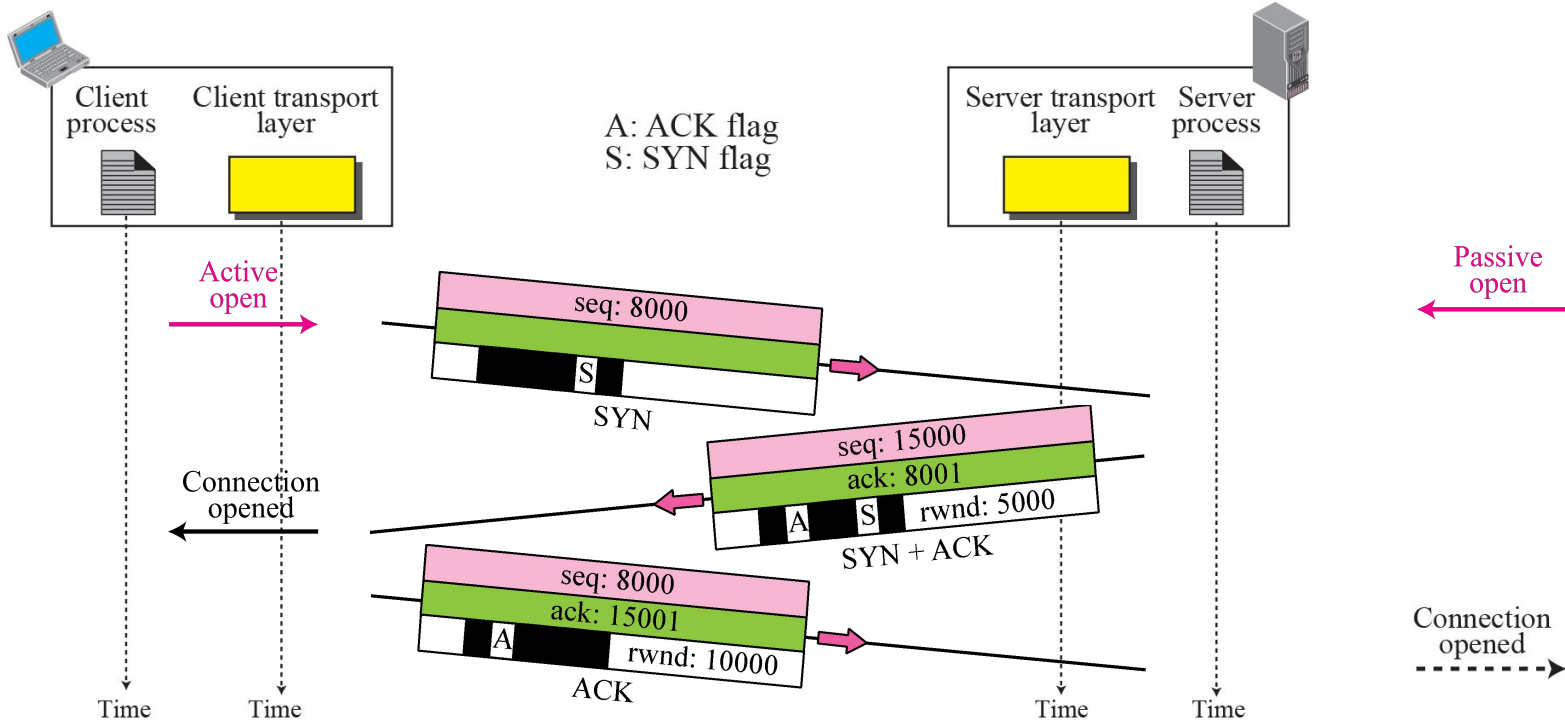
# Security Implications of Sequence Num

- Need to 1) spoof IP and Port; 2) use a valid sequence number to inject malicious data to an established connection

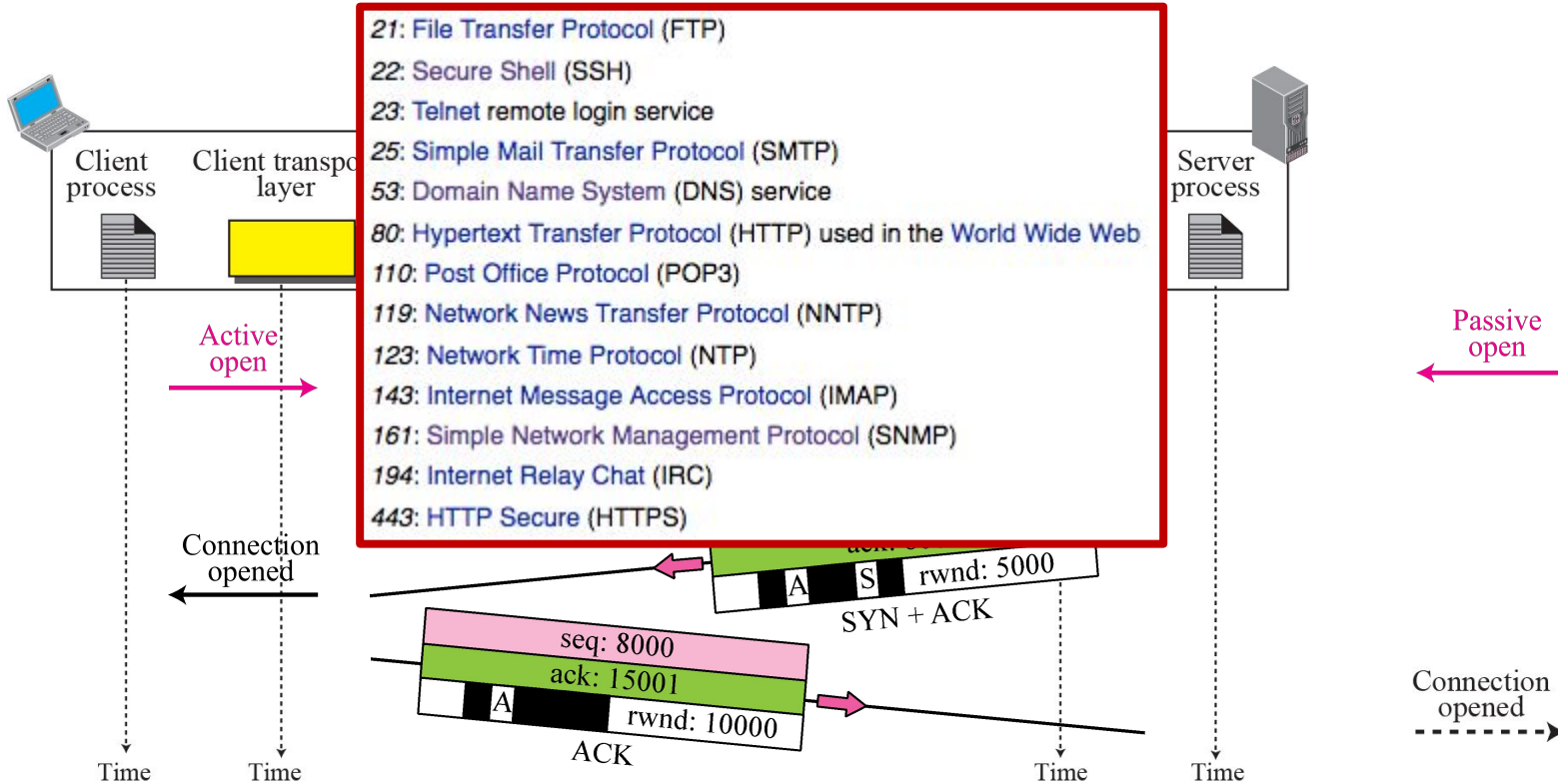




# TCP Three-way Handshake



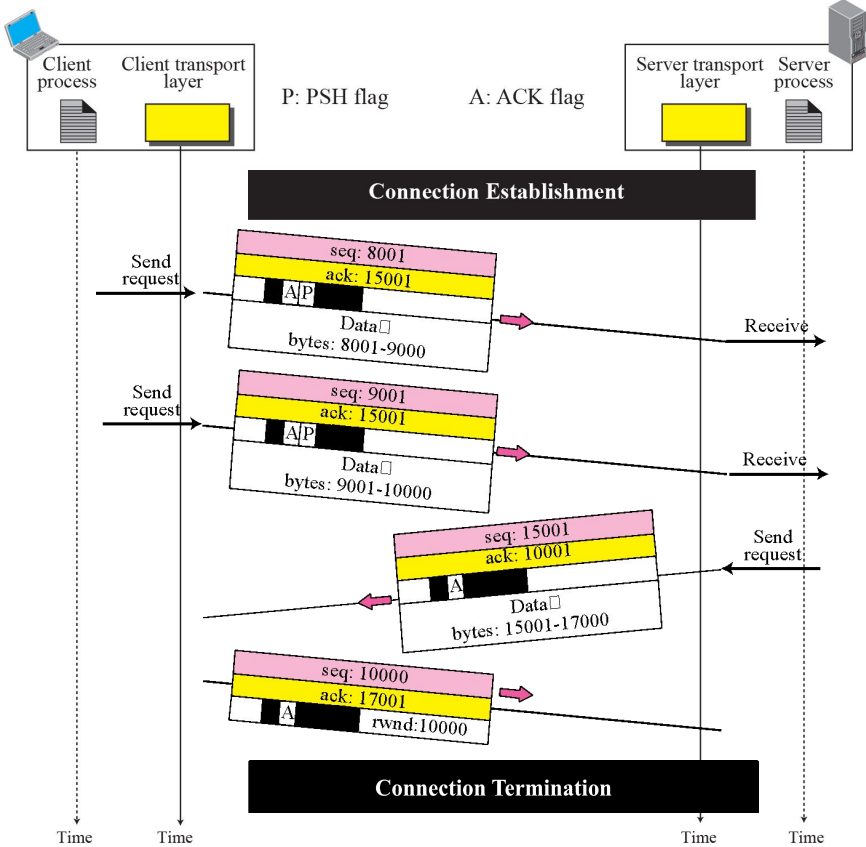
# TCP Three-way Handshake



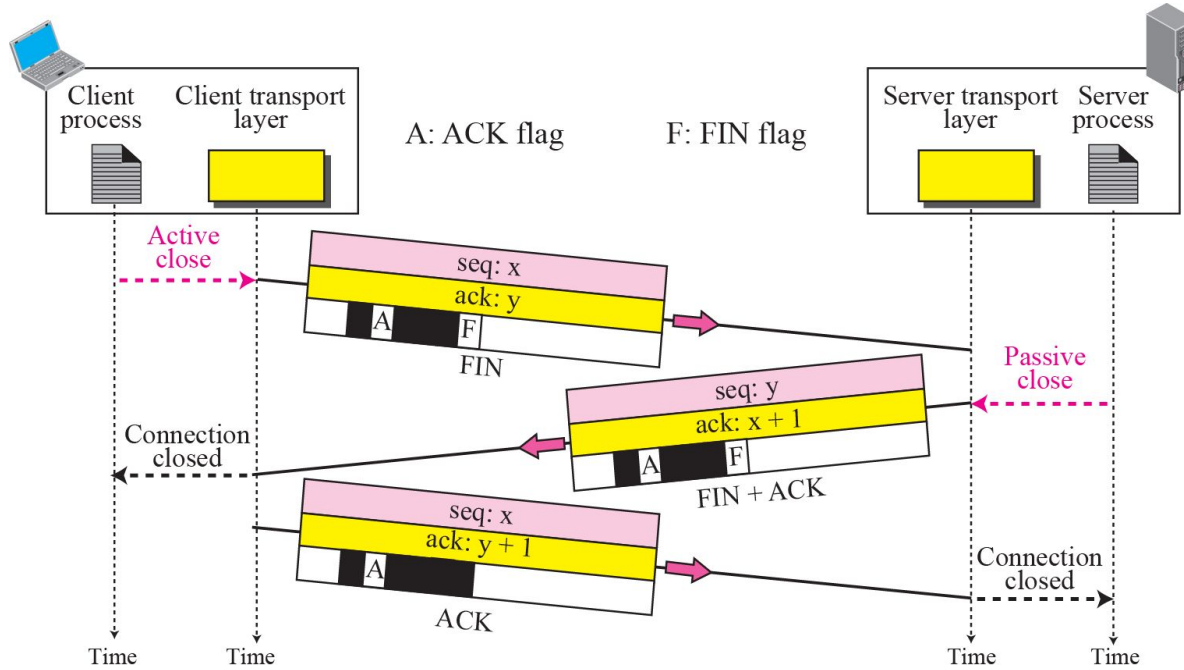
# TCP Three-way Handshake

- A SYN segment cannot carry data, but it consumes one sequence number.
- A SYN + ACK segment cannot carry data, but does consume one sequence number.
- An ACK segment, if carrying no data, consumes no sequence number.

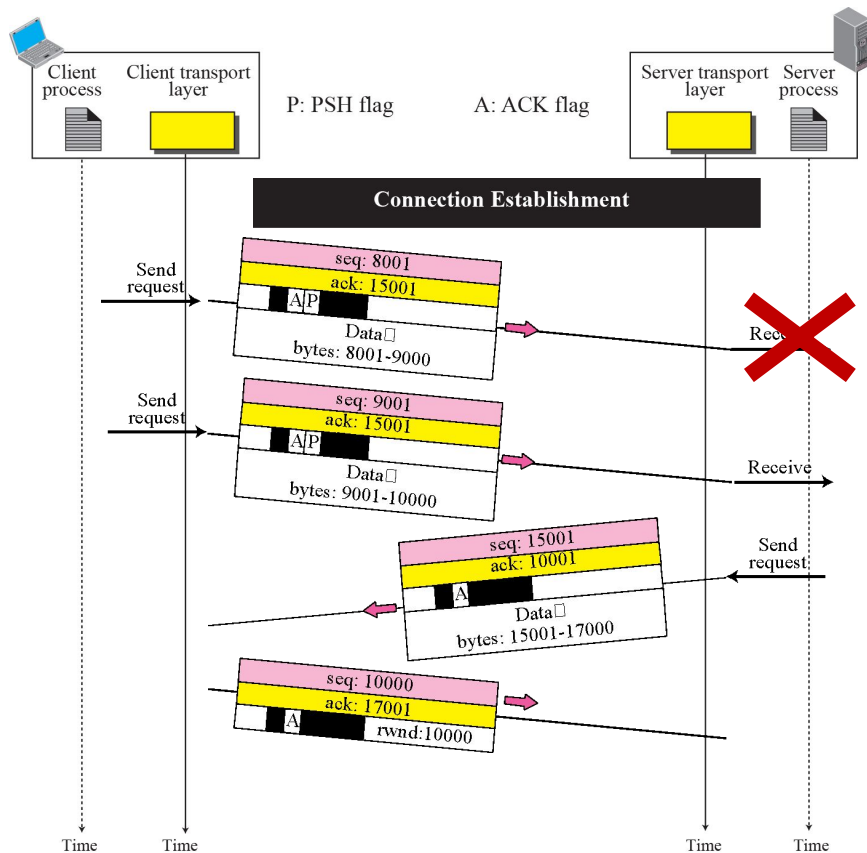
# Data Transfer after Handshake



# Connection Termination Handshake



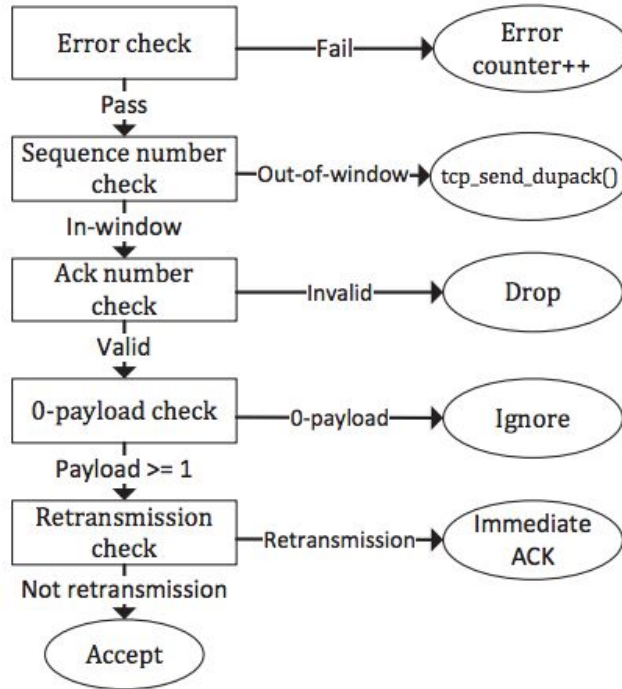
# Data Transfer after Handshake



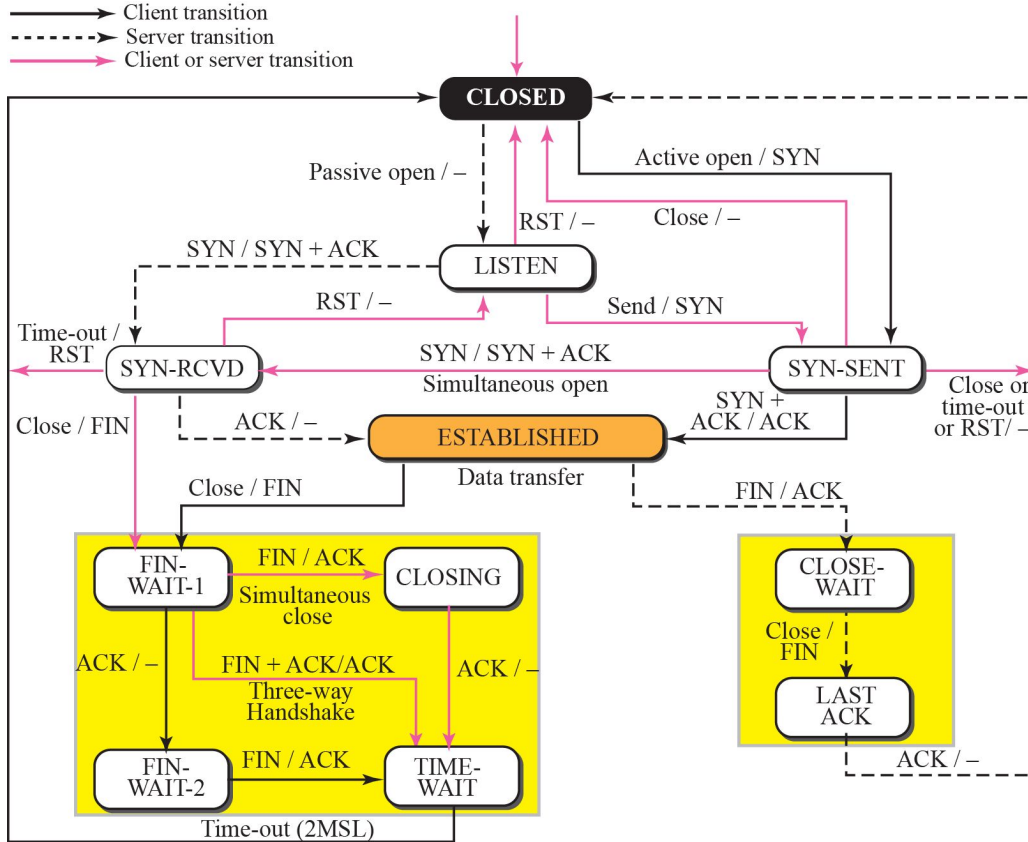
Expected Sequence  
Number 8001

Expected Sequence  
Number from 8001  
to 18001  
Window Size 10000

# Incoming TCP Packet Validation Logic



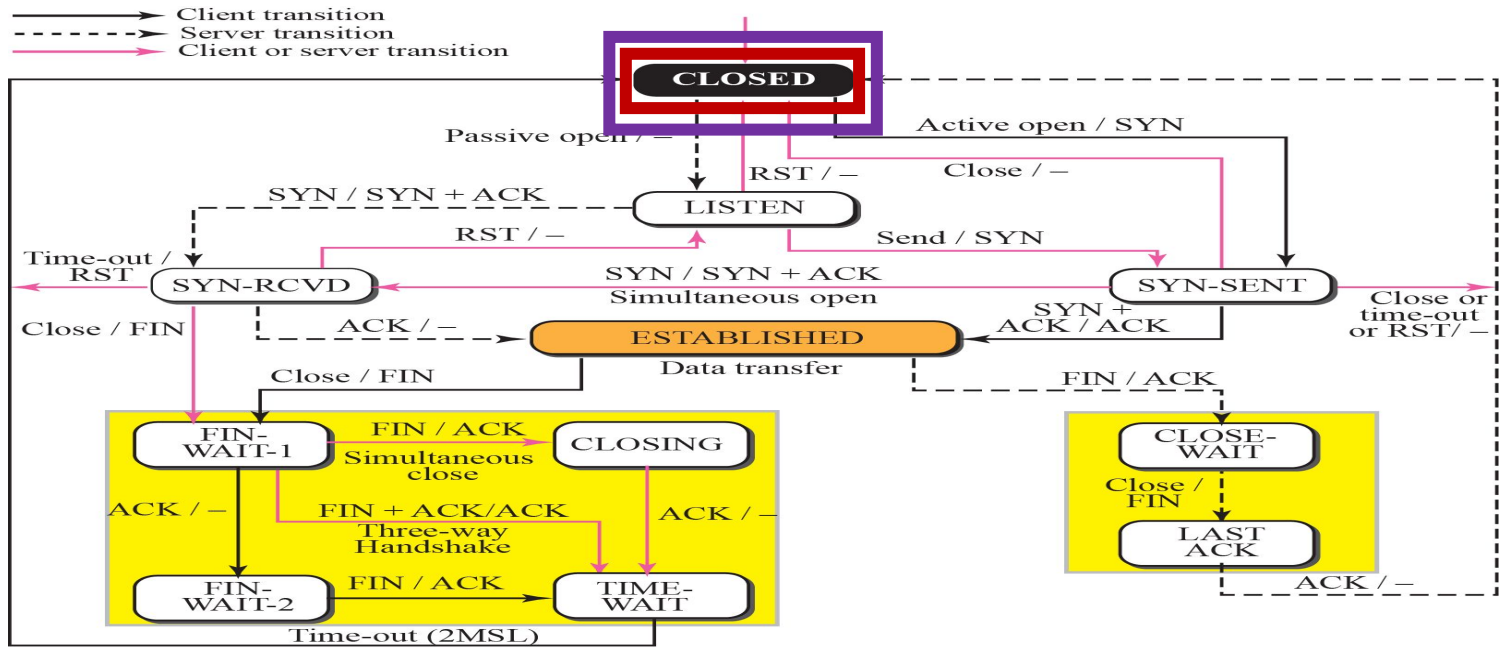
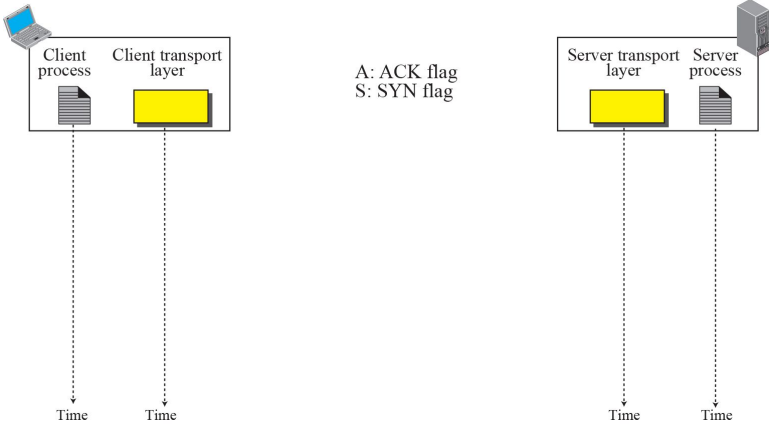
# State Machine





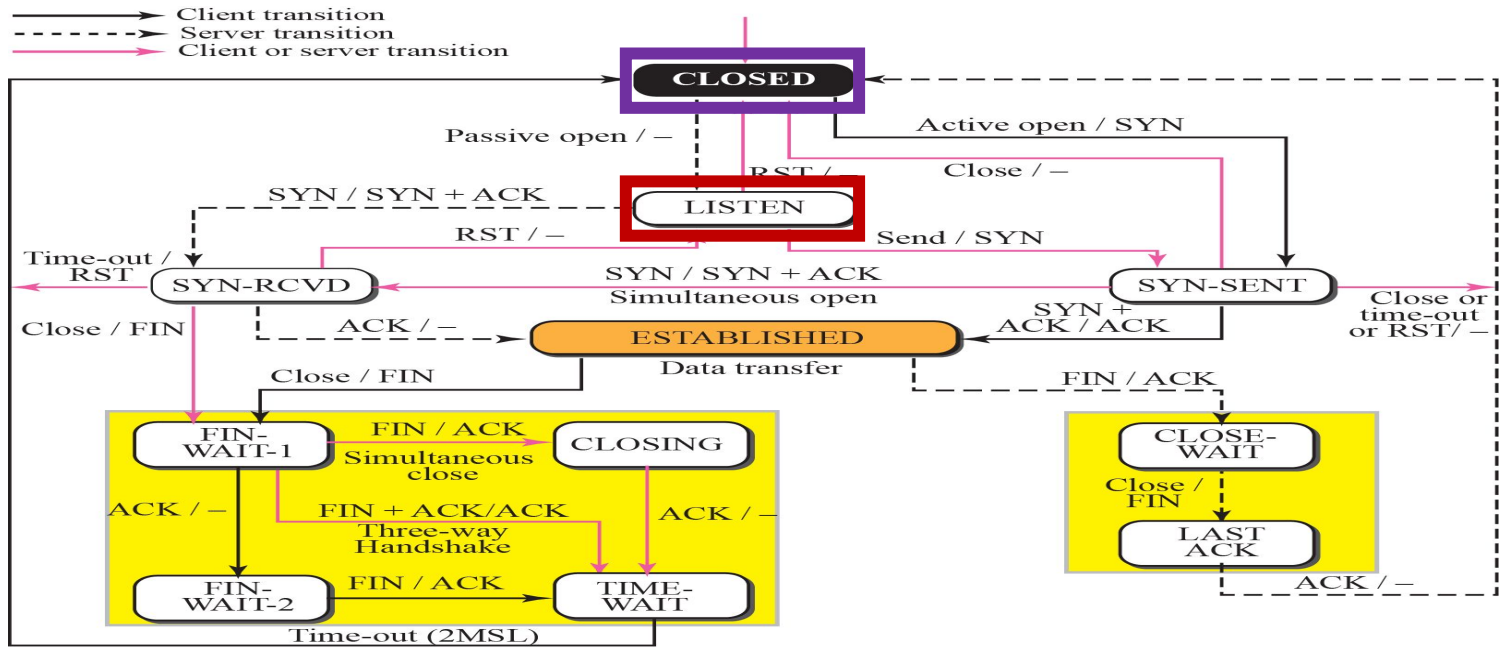
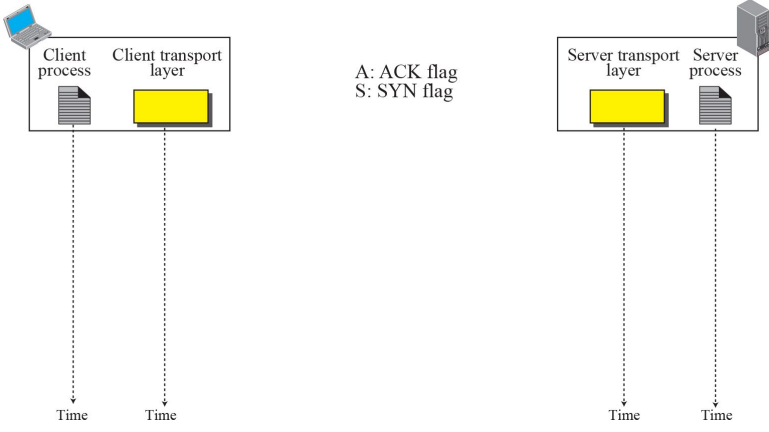
Server State

Client State



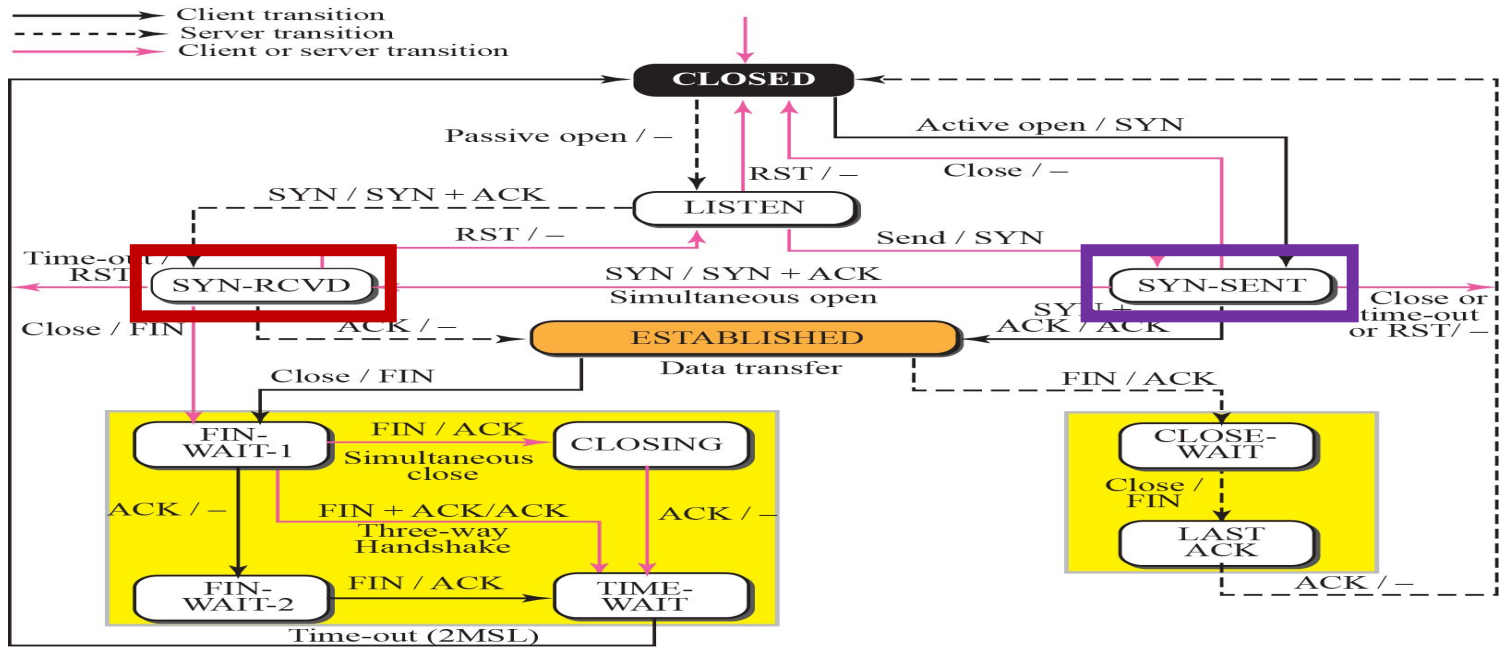
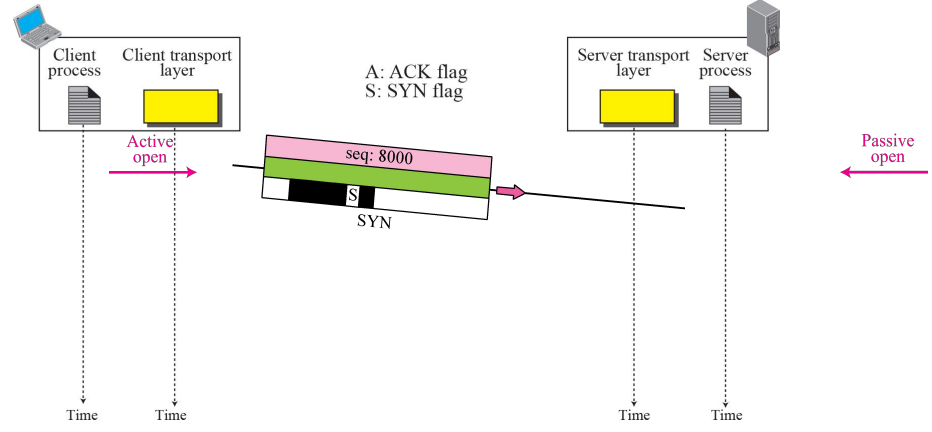
Server State


Client State




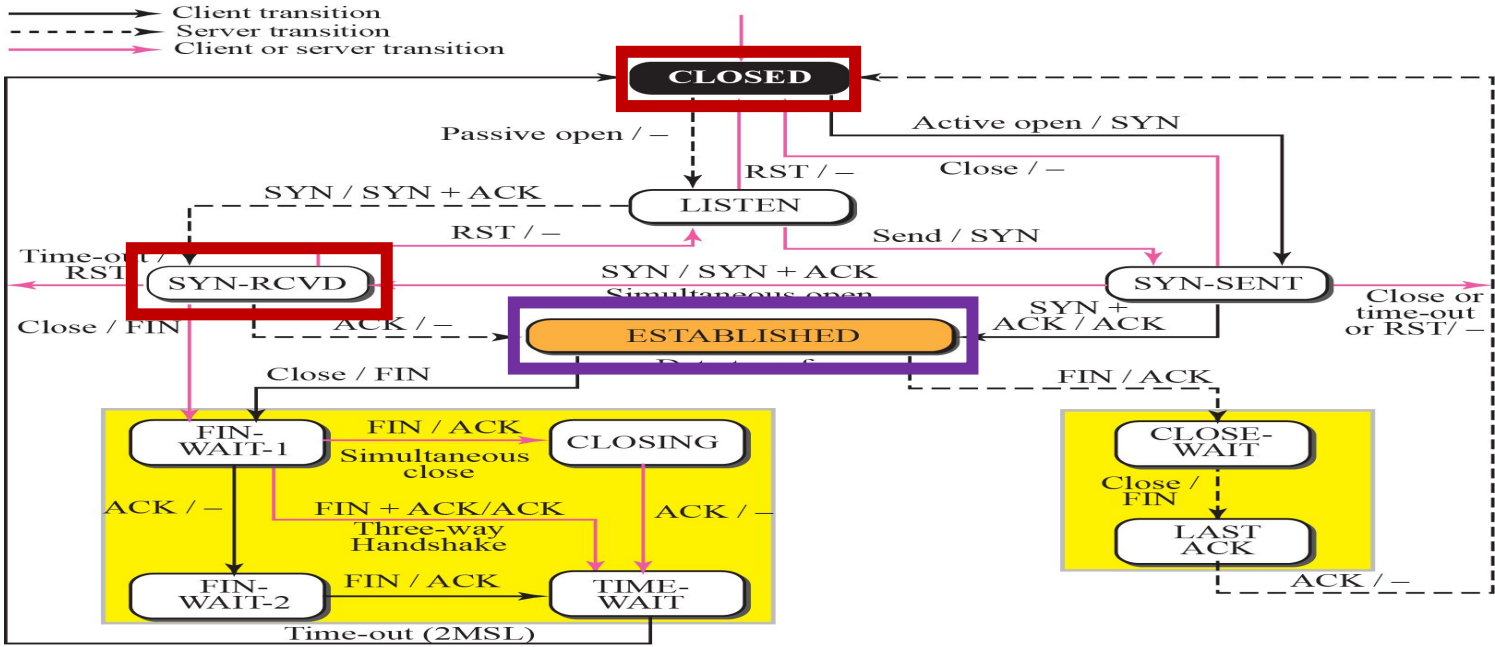
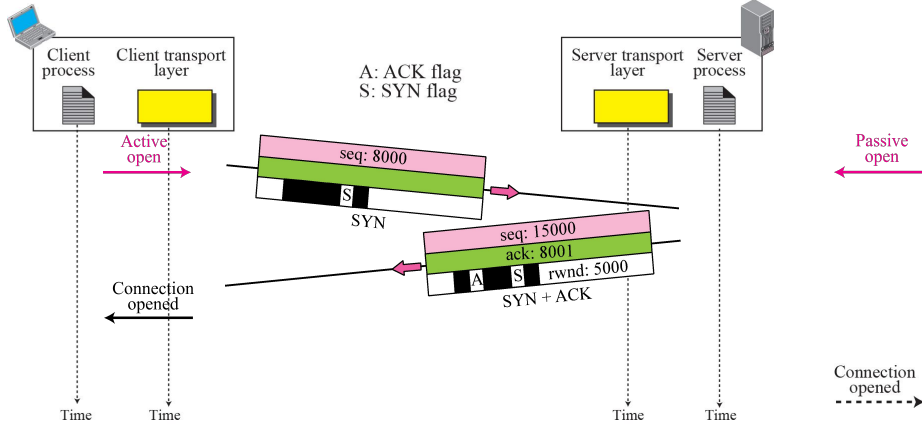
Server State  

Client State  



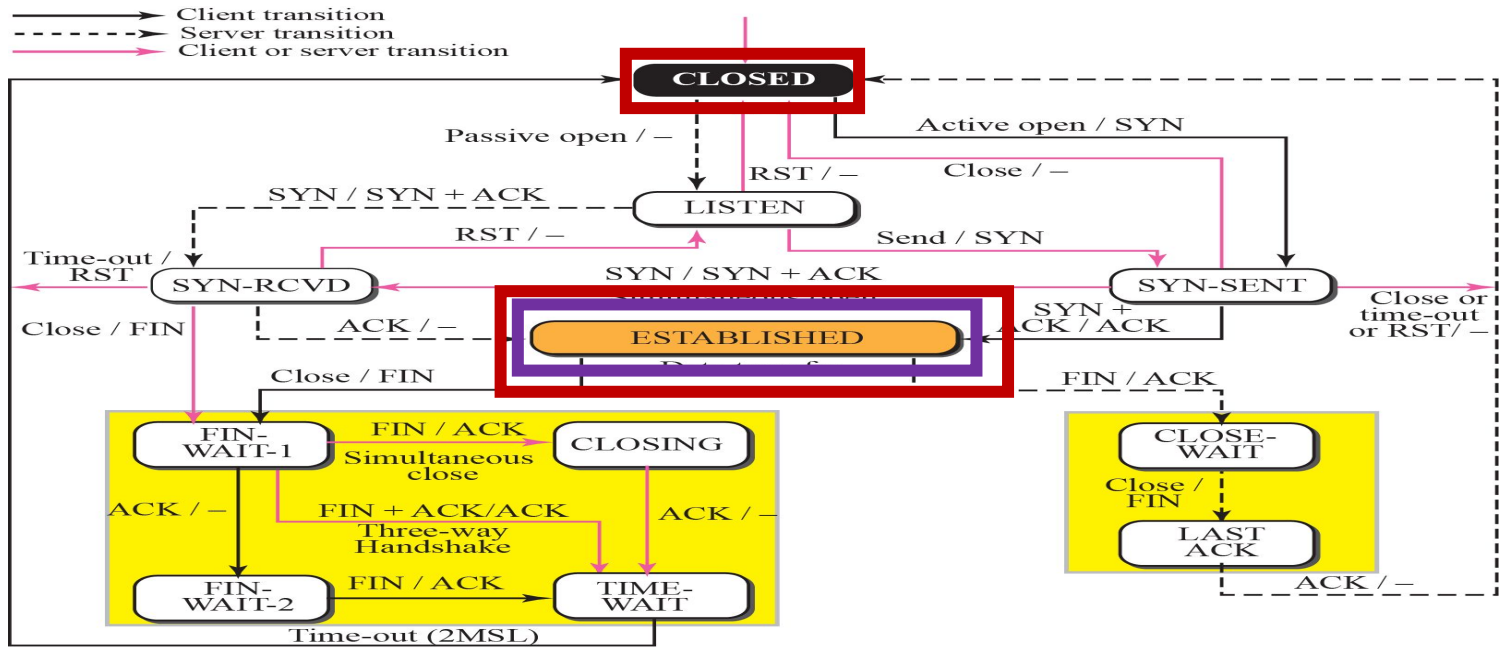
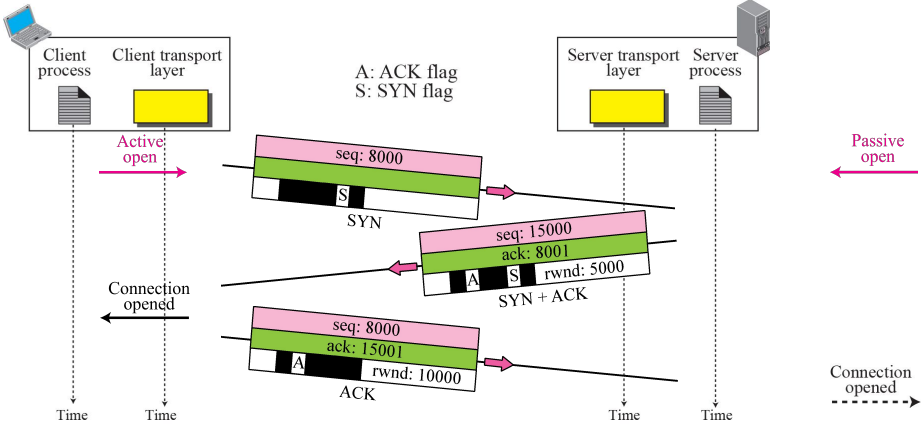
Server State 

Client State 

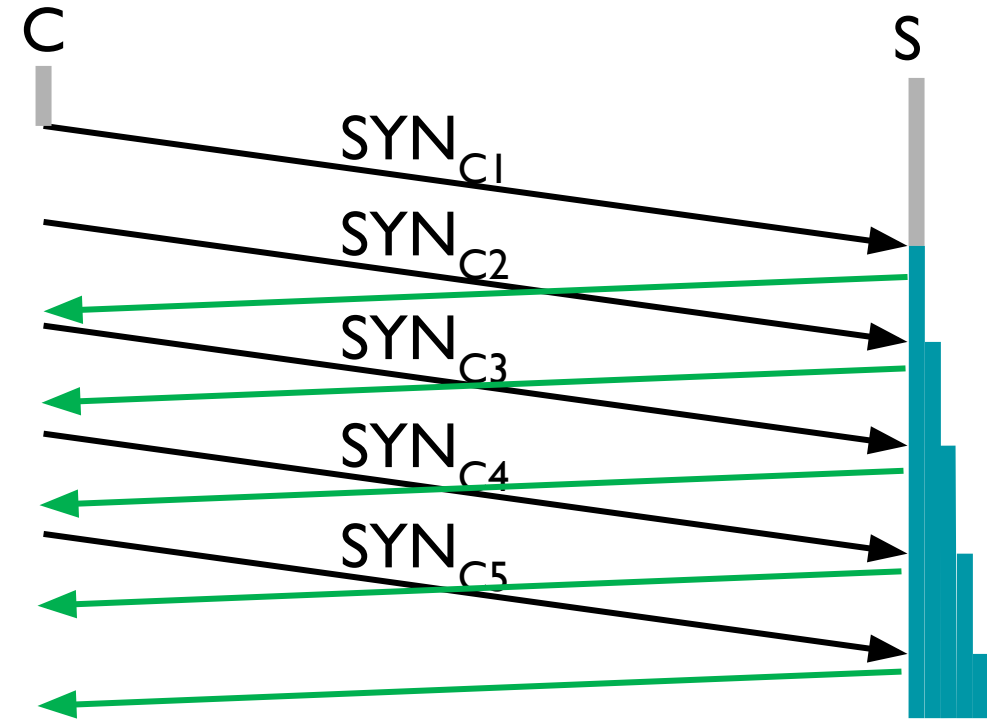


Server State

Client State

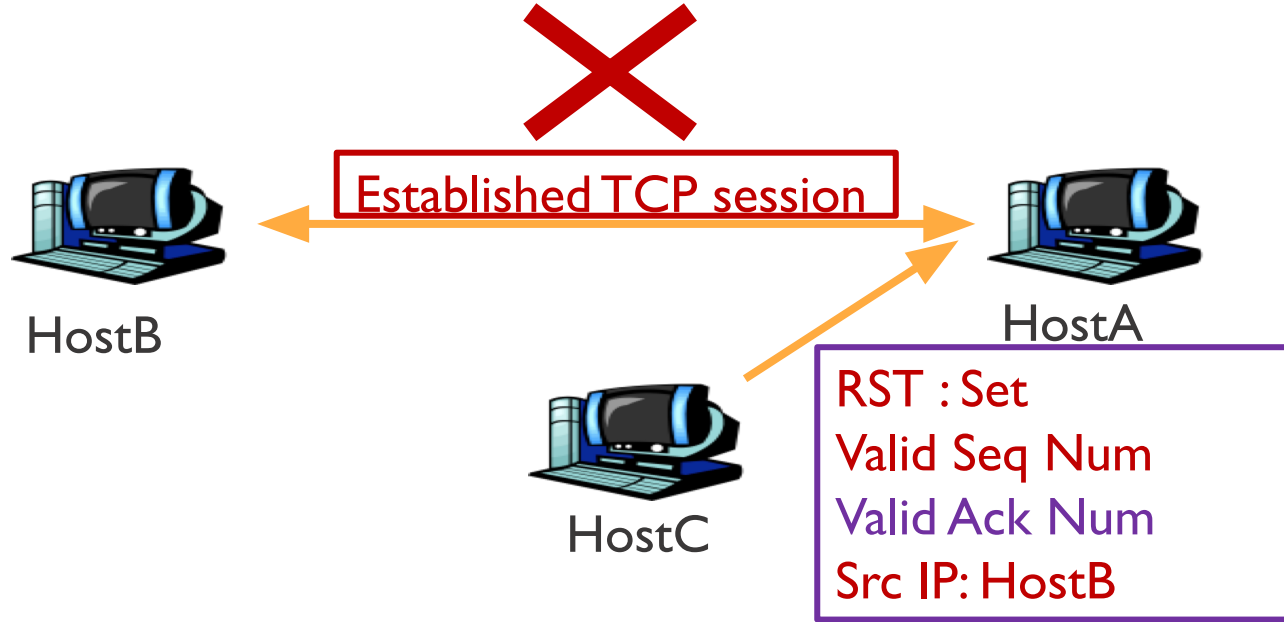


# SYN Flooding



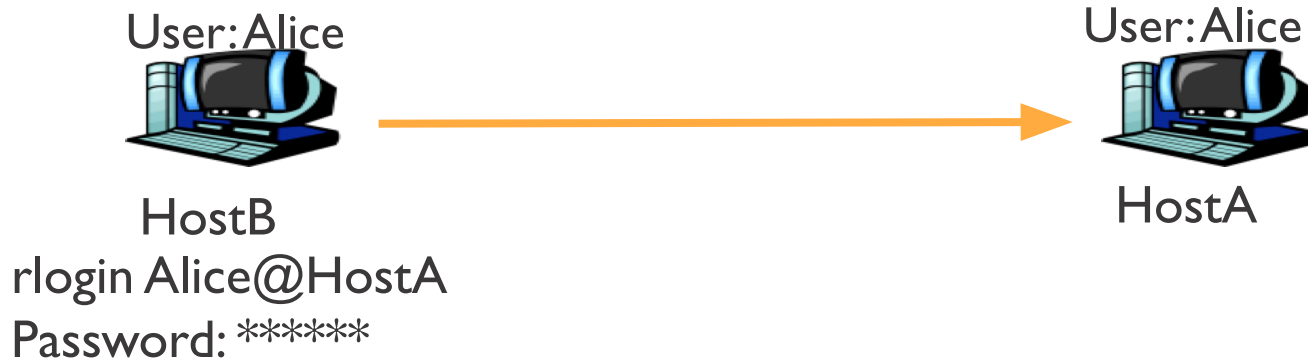
- Attacker sends many connection requests
  - *May* use spoofed source IP addresses
- Victim allocates resources for each request
  - Connection requests exist until timeout
- Resources exhausted  $\Rightarrow$  requests rejected
- Donot need to guess sequence number

# TCP Reset Attack



# rlogin

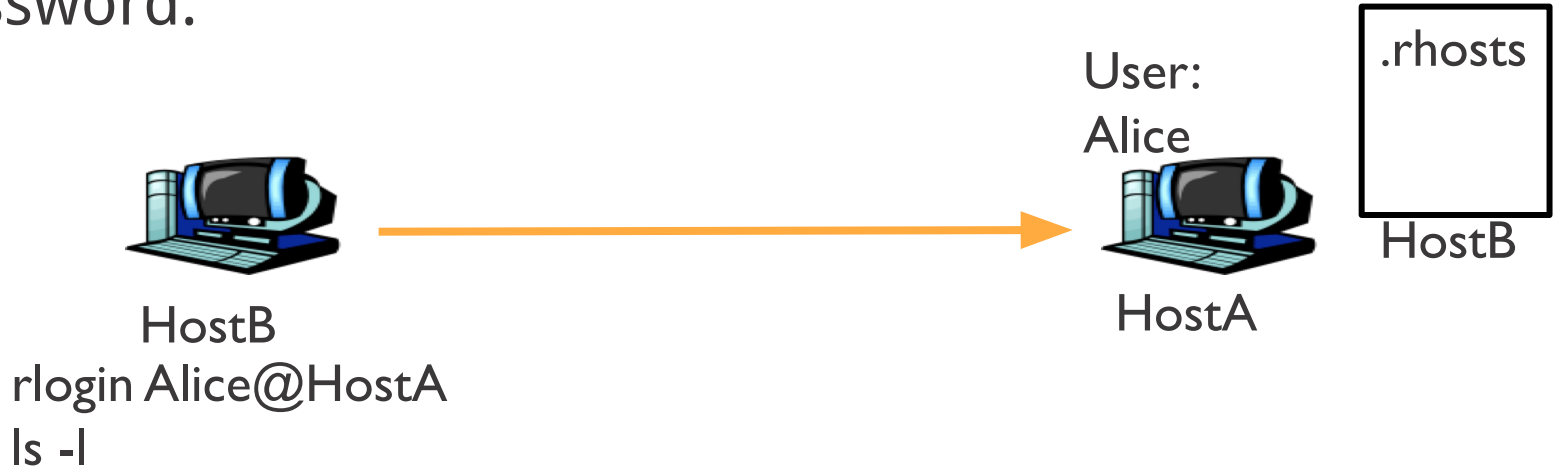
- rlogin is a software utility for Unix-like computer operating systems that allows users to log in on another host via a network, communicating via **TCP** port 513. (like Telnet, ssh)





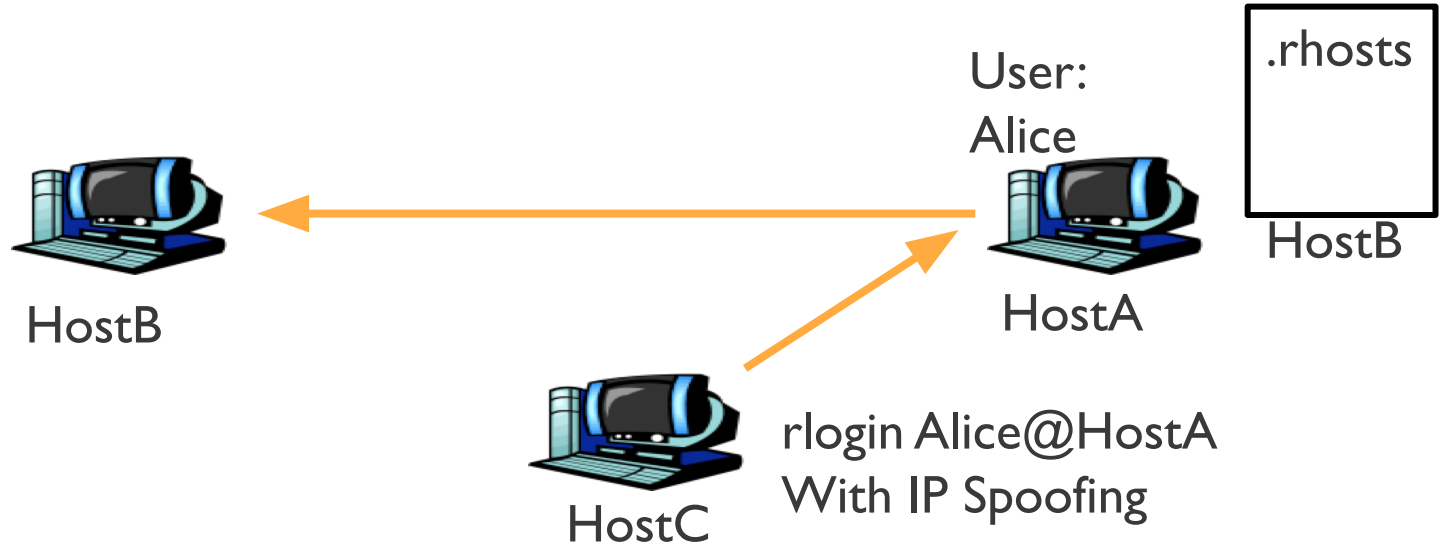
# rlogin

- Alice can specify trusted hosts in `~/.rhosts`.
- If a connection is from a trusted host, permission is granted to log in remotely without having to supply a password.



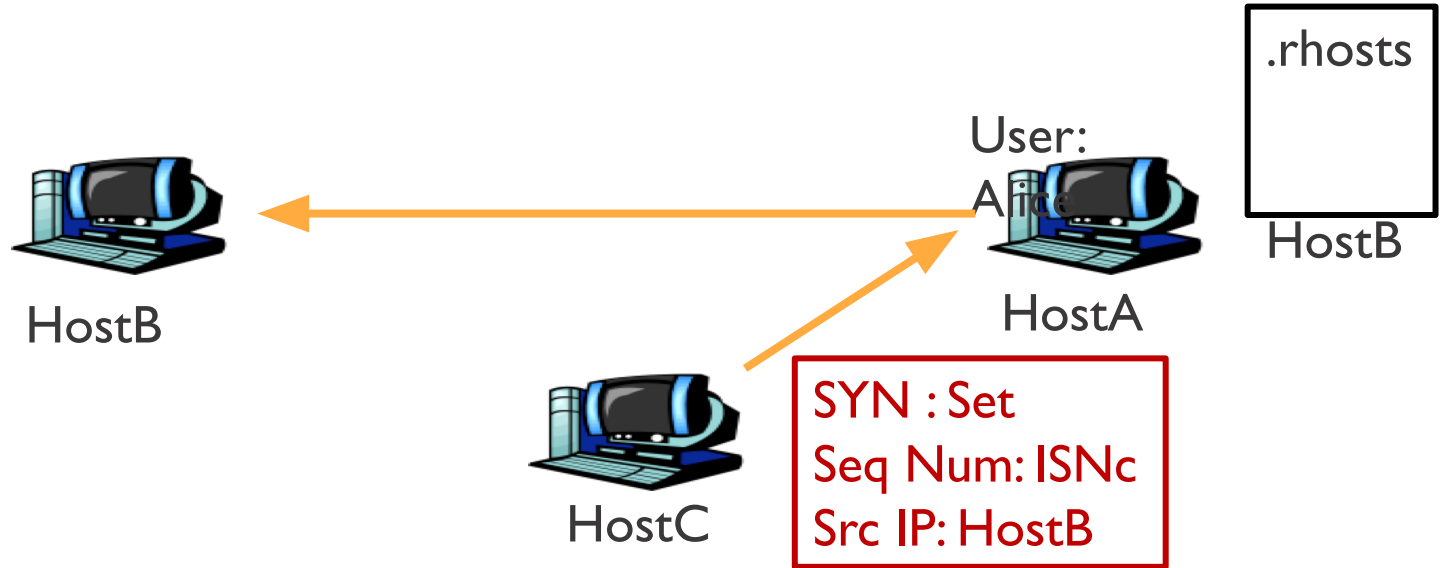
# rlogin

- Attacker from HostC can execute commands on HostA



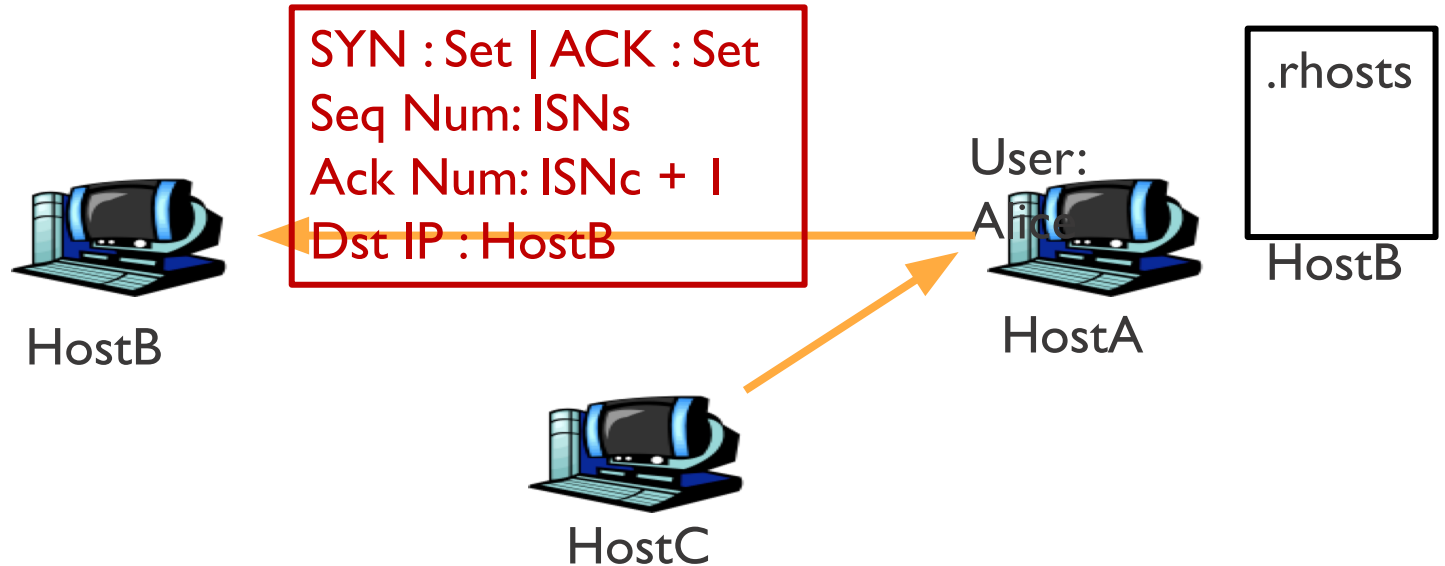
# rlogin

- Attacker from HostC can execute commands on HostA



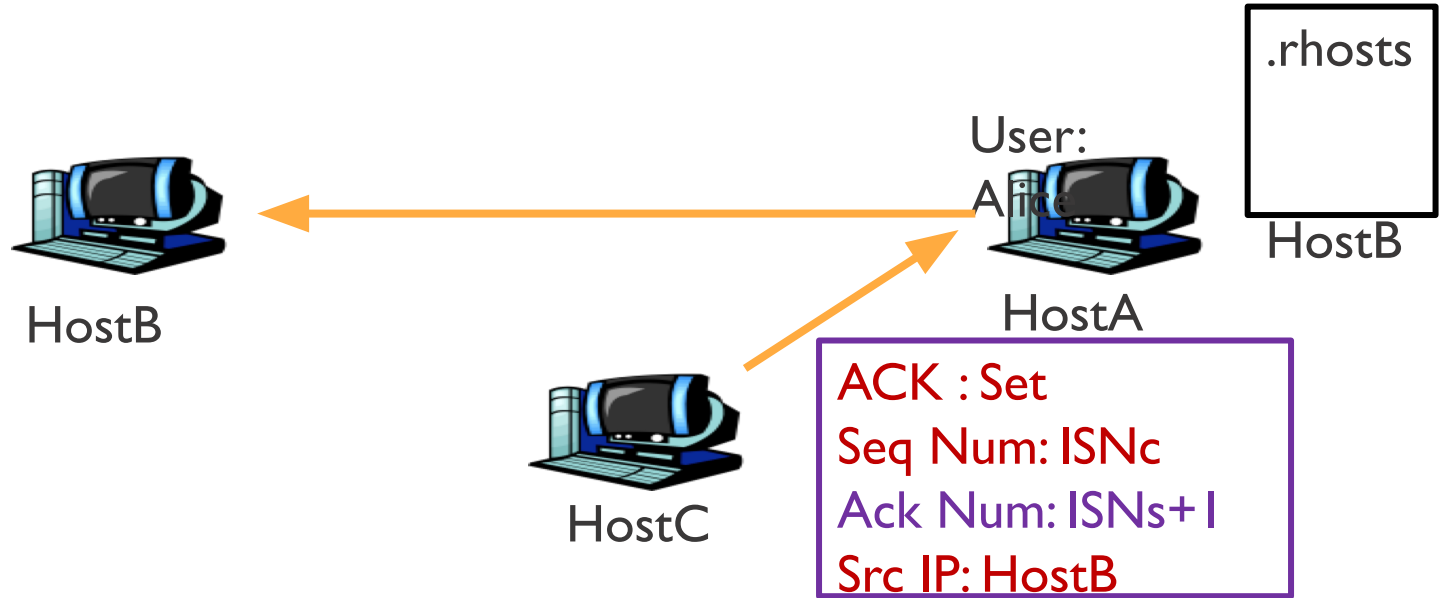
# rlogin

- Attacker from HostC can execute commands on HostA



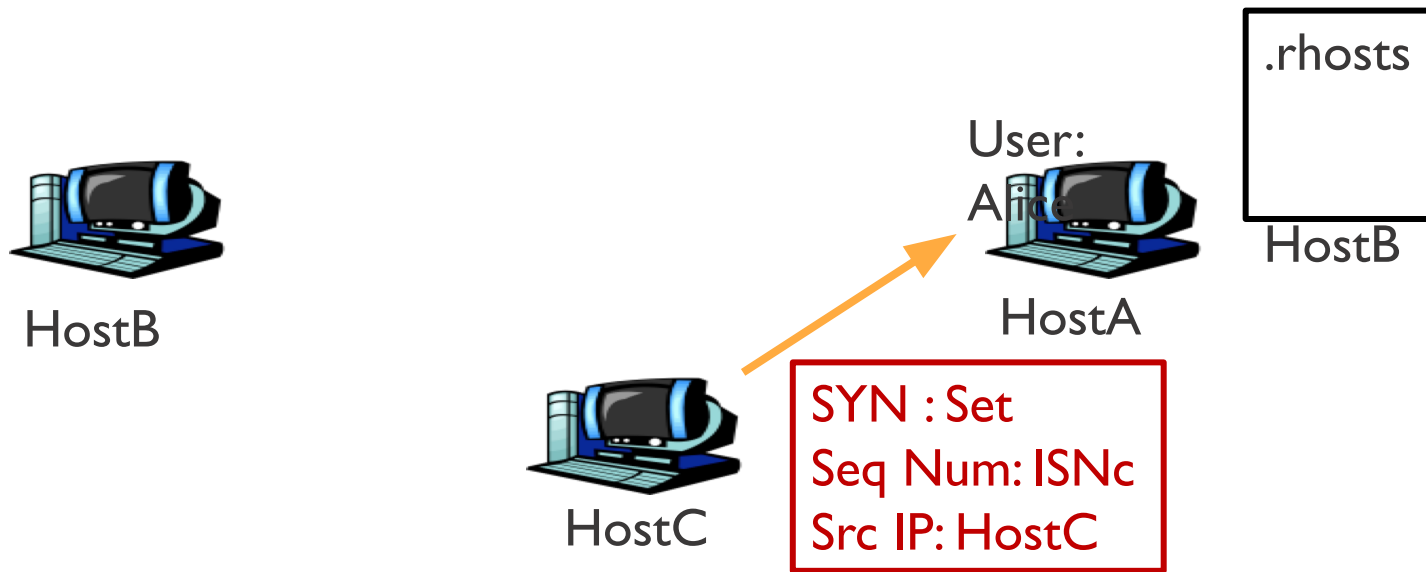
# rlogin

- Attacker from HostC can execute commands on HostA



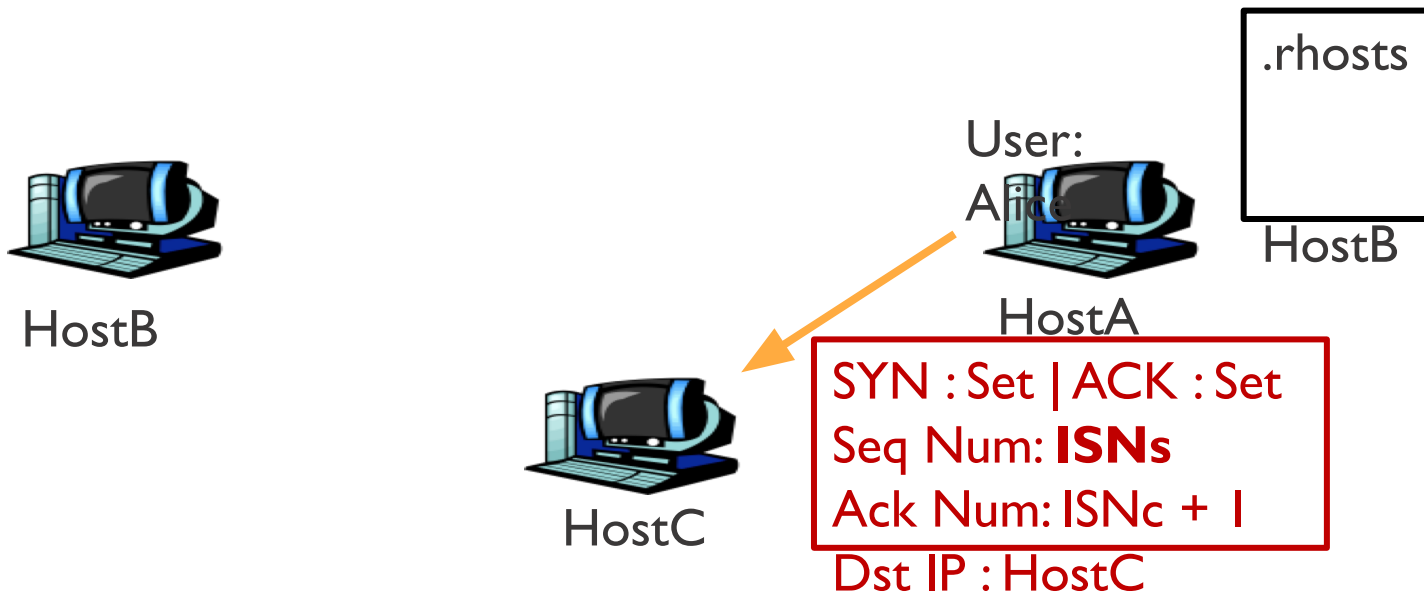
# TCP Seq Num Prediction

- Robert Morris, 1985
- 4.2BSD maintains a **global initial sequence number (all processes share this)**, which is incremented by 64 after a connection is started (**NOT random**);



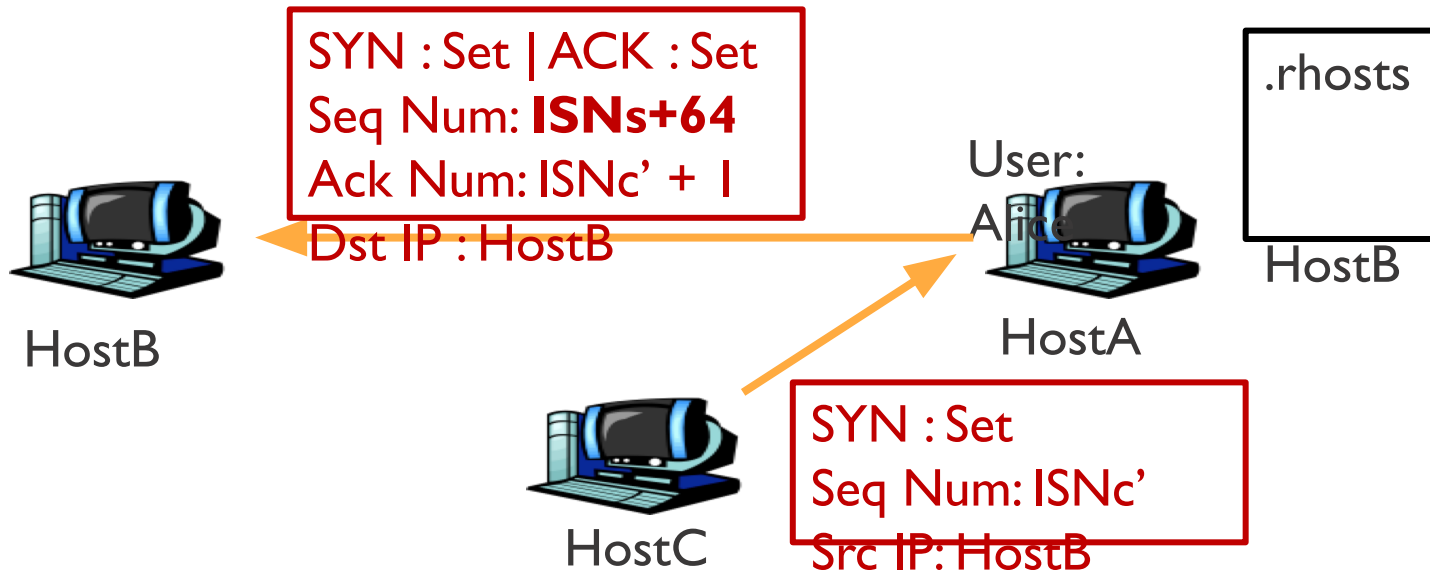
# TCP Seq Num Prediction

- Robert Morris, 1985
- 4.2BSD maintains a **global initial sequence number (all processes share this)**, which is incremented by 64 after a connection is started (**NOT random**);



# TCP Seq Num Prediction

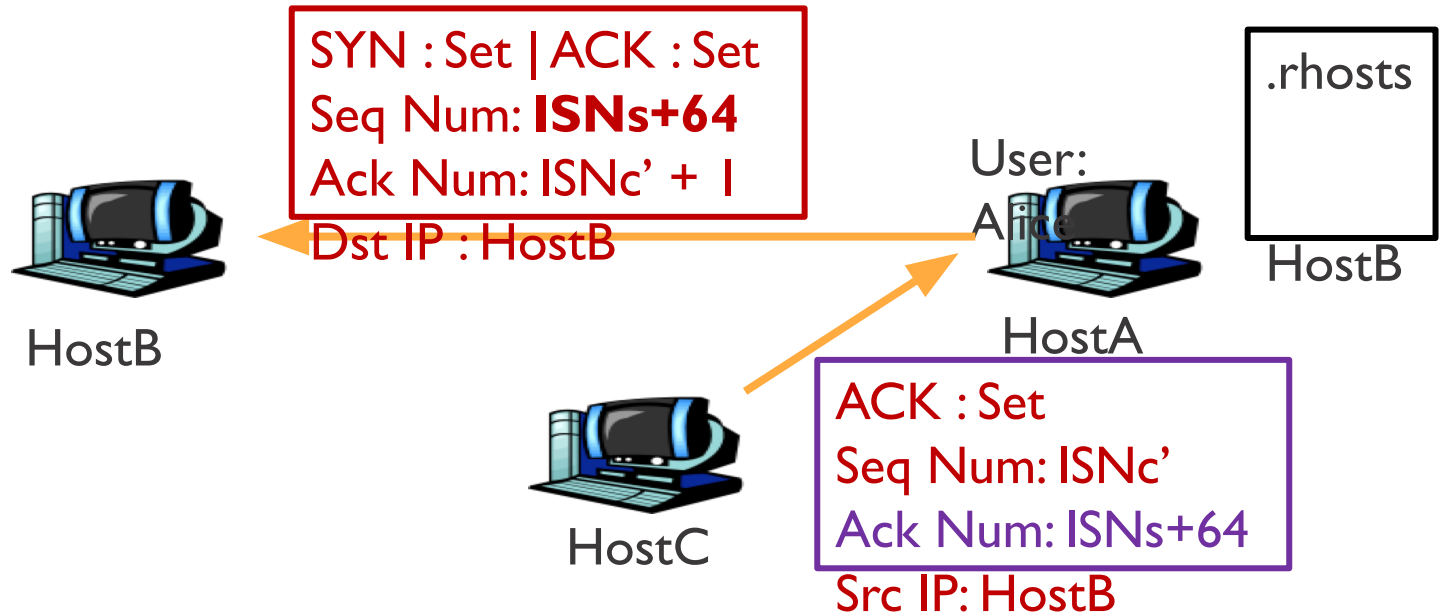
- Robert Morris, 1985
- 4.2BSD maintains a **global initial sequence number (all processes share this)**, which is incremented by 64 after a connection is started (**NOT random**);





# TCP Seq Num Prediction

- Robert Morris, 1985
- 4.2BSD maintains a **global initial sequence number (all processes share this)**, which is incremented by 64 after a connection is started (**NOT random**);



# Defense

- Random sequence number

# TCP Session Hijacking Attack

Hijack or inject data to an alive TCP connection

- TCP Session Hijacking
- TCP Injection
  
- Spoof IP address (relatively easy)
- Get the valid Seq & Ack numbers (relatively difficult)



HostB



HostC



HostA

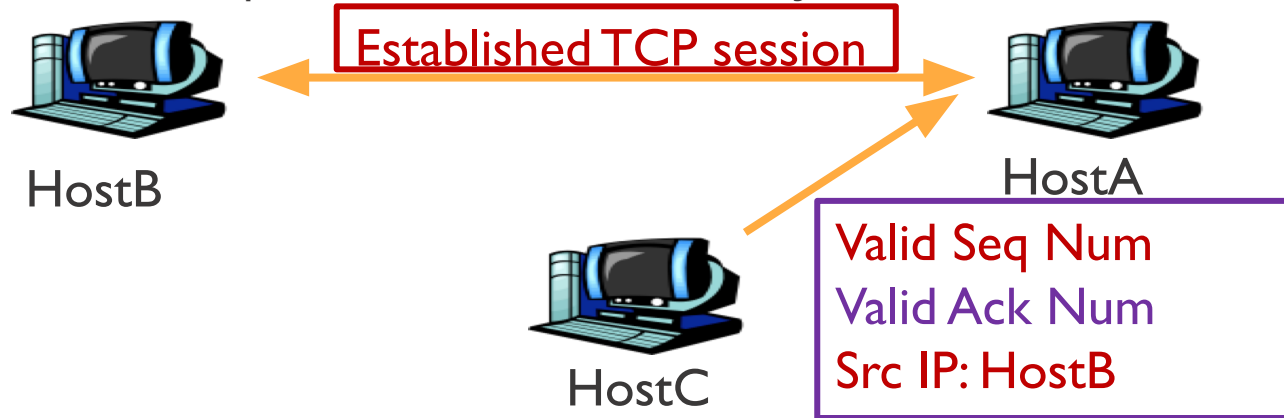


Valid Seq Num  
Valid Ack Num  
Src IP: HostB

# TCP Session Hijacking Attack

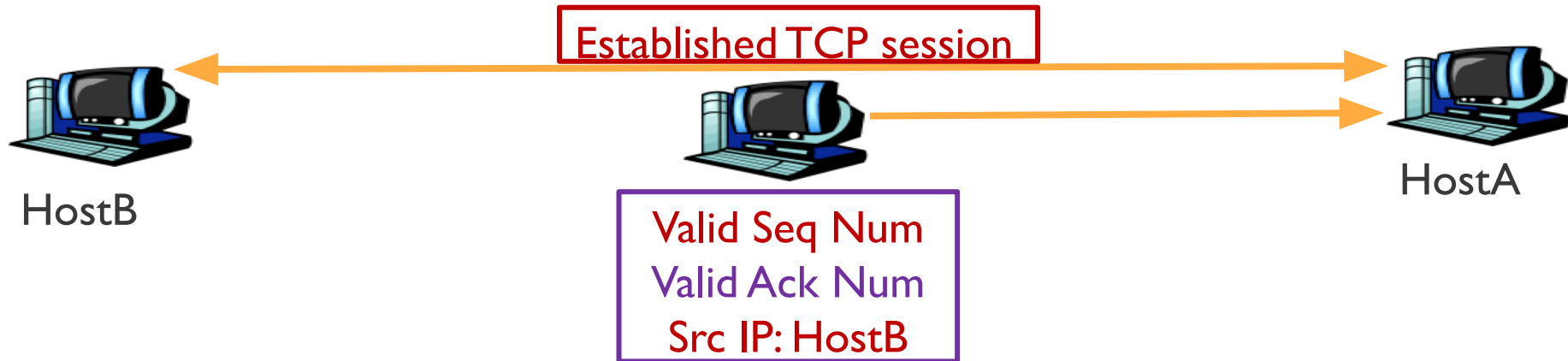
Hijack or inject data to an alive TCP connection

- TCP Session Hijacking
- TCP Injection
  
- Spoof IP address (relatively easy)
- Get the valid Seq & Ack numbers (relatively difficult)



# TCP Session Hijacking Attack

- Attacker sits **on the data path** between the communicating two parties

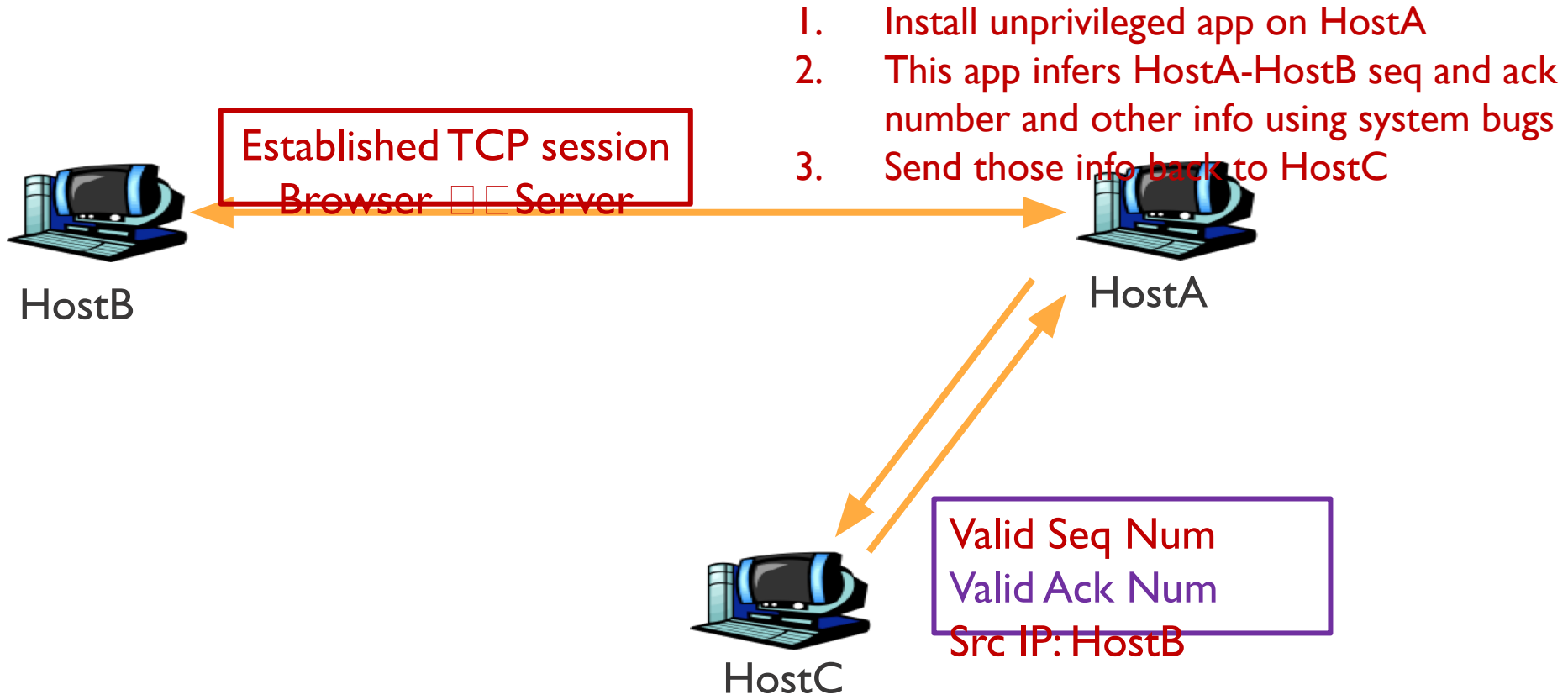


# TCP Session Hijacking Attack

- ISP injects rogue advertisements



# Off-path TCP Session Hijacking Attack



# Off-path TCP Session Hijacking Attack

