

# TEL 315 TCP/IP

## Week 6: TCP and UDP

**Gary C. Kessler**  
Champlain College

### TCP/IP Transport Layer

Application Layer	HTTP FTP Telnet Finger DNS POP3/IMAP SMTP Gopher BGP Time/NTP Whois TACACS+ SSH NNTP SSL/TLS (https, etc.) SOCKS	DNS SNMP RIP RADIUS Archie traceroute tftp DHCP Kerberos	Ping tracert	
Transport Layer	TCP		UDP	ICMP OSPF
Internet Layer	IP			ARP
Network Access	Ethernet/802.3 Token Ring (802.5) SNAP/802.2 X.25 FDDI ISDN Frame Relay SMDS ATM Wireless 802.x Fibre Channel xDSL Cable modem DS0/T1/T3 SONET DWDM HDLC PPP SLIP/CSLIP			

# Transport Layer Protocols

---

- TCP/IP's Transport Layer protocols provide end-to-end (host-to-host) communication
  - » Transmission Control Protocol (TCP)
  - » User Datagram Protocol (UDP)
- End-to-end connections identified by host sockets (port number + IP address)

**Source IP address, Source TCP/UDP port,  
Destination IP address, Destination TCP/UDP port**


- » Ports identify services; see /etc/services (Unix) or %systemroot%\system32\drivers\etc\services (WinNT/2K)

# Ports

---

Port No.	Protocol	Application	Port No.	Protocol	Application
7	UDP	echo	80	TCP	http
13	TCP	daytime	110	TCP	pop3
19	UDP	chargen	111	TCP	sunrpc
20	TCP	ftp-data	113	TCP	auth
21	TCP	ftp-control	119	TCP	nntp
22	TCP	ssh	123	UDP	ntp
23	TCP	telnet	137	UDP	netbios-ns
25	TCP	smtp	138	UDP	netbios-dgm
37	UDP	time	139	TCP	netbios-ssn
43	TCP	whois	143	TCP	imap
53	TCP/UDP	dns	161	UDP	snmp
67	UDP	bootps	162	UDP	snmp-trap
68	UDP	bootpc	179	TCP	bgp
69	UDP	ftft	443	TCP	https (http/ssl)
70	TCP	gopher	514	UDP	syslog
79	TCP	finger	520	UDP	rip

## Ports (cont.)

- Applications are identified by port numbers
  - » *Well-known ports* (0-1023) identify the server side of an application, usually implemented so as to run at a high privilege level
    - Specific applications can redefine the port assignment; e.g., 81 and 8080 are common ports for Web servers
  - » *Registered ports* (1024-49151) are publicly defined as a convenience to the Internet community; they can identify the server or client side
  - » *Dynamic and/or private ports* (49152-65535) can be used by any client or server
- Proxy servers and NAT often change the port numbers 
- Hackers probe/scan and firewalls packet filter based upon TCP/UDP port numbers

## Checksums

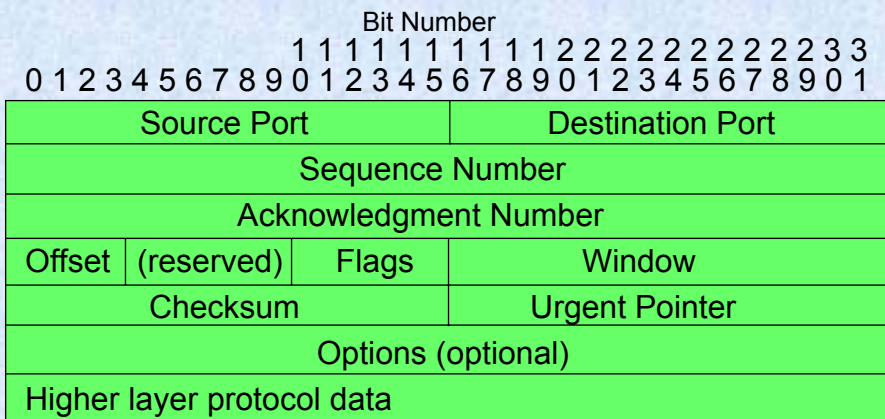
- TCP/UDP use a checksum for error-free host-to-host communication
- Checksum covers a 96-bit pseudoheader, and entire TCP/UDP header and data

```
+-----+-----+-----+-----+
|           Source IP Address           |
+-----+-----+-----+-----+
|           Destination IP Address      |
+-----+-----+-----+-----+
|   | Prot. | TCP Segment/ |
| 0x00 | TCP=6 | UDP Datagram |
|   | UDP=17| Length   |
+-----+-----+-----+-----+
```

# TCP

- Provides Transport Layer functionality
  - » Connection-oriented, reliable virtual circuit service
- Responsible for:
  - » Virtual circuit setup and termination
  - » Acknowledgments
  - » Flow control
- Data unit: Segment (stream of data bytes)
- Operates over IP (Protocol = 6)
- Base document: RFC 793 (STD 7)

# TCP Segment Format



## Other TCP Fields

---

- *Sequence Number*: Sequence number of first data byte from the perspective of the byte stream
- *Acknowledgement Number*: Sequence number of next byte in the stream that is expected
- *Data Offset*: Size of TCP Header in 32-bit words
- *Window*: TCP's flow control mechanism; the number of bytes that the sender of this segment is willing to accept from the receiver
- *Urgent Pointer*: Points to first data byte in segment after urgent data

## TCP Control Bits (Flags)

---

```
Bit Number
1 1 1 1 1 1
0 1 2 3 4 5
+---+---+---+---+
|U|A|P|R|S|F|
|R|C|S|S|Y|I|
|G|K|H|T|N|N|
+---+---+---+---+
```

- Urgent Pointer Field Significant (URG)
- Acknowledgement Field Significant (ACK)
- Push (PSH)
- Reset Connection (RST)
- Synchronize Sequence Numbers (SYN)
- Finish (FIN)

# TCP Options

---

- Window scale factor (wscale)
  - » Allows Window value to be up to 32 bits (4 GB) in length; scale factor tells how many bits to shift Window

WINDOW	wscale	Effective Window Size
65535	0	65,535 ( $65535 * 2^0 = 2^{16}$ )
65535	1	131,070 ( $65535 * 2^1 = 2^{17}$ )
65535	2	262,140 ( $65535 * 2^2 = 2^{18}$ )
	:	

- Selective Acknowledgement (SACK)

# Maximum Segment Size

---

- MSS option specifies the largest TCP segment that will be assembled
  - » Since a segment has nominally a 20B header, large MSS values will offer more line efficiency
  - » "Fragmentation is harmful"; path MTU discovery (RFC 1191) finds largest packet that won't require fragmentation
  - » Common MSS is 1460B (Ethernet Info = 1500B); default across WAN is 536B

# Congestion Control

---

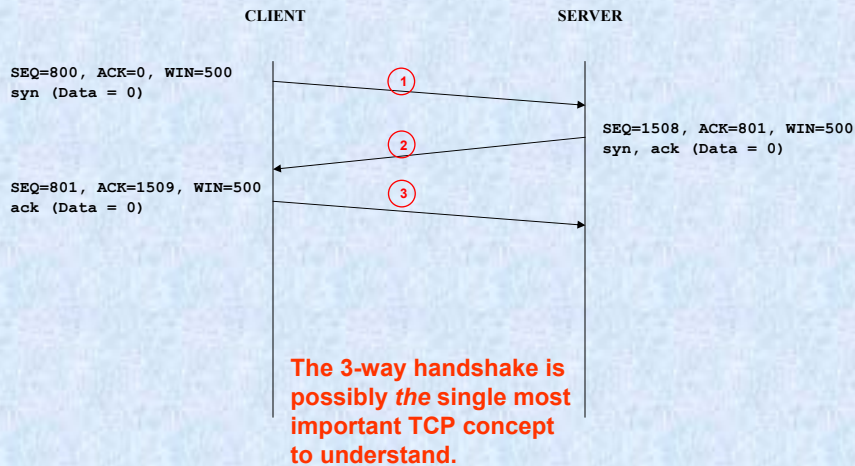
- TCP is a Go-Back-N protocol with ACKs and a transmission timeout mechanism
- Several enhancements to improve performance
  - » Slow Start
  - » Congestion Avoidance
  - » Fast Retransmit
  - » Selective Acknowledgement
  - » Nagle's Algorithm

# TCP Segment Exchange Scenarios

---

- Scenarios
  - » Connection establishment and termination
  - » Data exchange with and without bit errors
  - » Flow control, timeouts, and retransmission
- Ladder diagrams
  - » Indicates status of SYN (S), ACK (A), and/or FIN (F) flags
  - » The value of the Sequence Number (SEQ) and Acknowledgement Number (ACK) fields
  - » The value of the Window (WIN) field
  - » The amount of data sent with the segment

# Connection Establishment



TEL 315

Copyright © 2000-2003, Gary C. Kessler

14

# ISN Selection

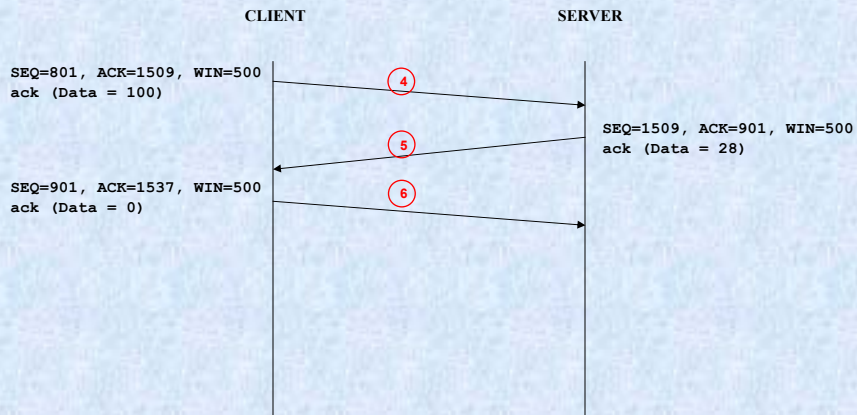
- Many algorithms; OS-dependent
  - » *Static increments*: Increment next ISN value to use by 1 (or other fixed value) every 4  $\mu$ s (or other fixed unit of time). The original method and one still used by Windows and others. The *64K method*, used in 4.4BSD and derivatives, increments by 64,000 every 500 ms and every time a connection is established.
  - » *Random increments*: Used by Solaris, IRIX, FreeBSD, Digital UNIX, Cray, and many others.
  - » *Random ISN*: Used by Linux, OpenVMS, newer AIX.
  - » *Constant*: Uses same ISN for every connection! Some 3Com hubs use 0x803 and Apple LaserWriter printers use 0xC7001

TEL 315

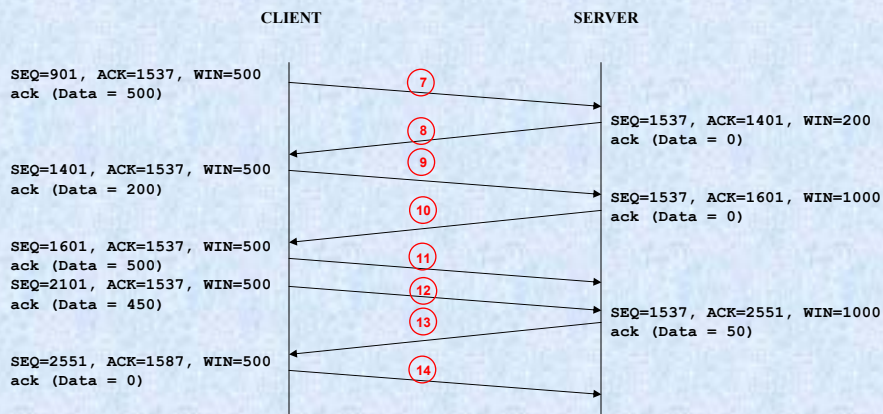
Copyright © 2000-2003, Gary C. Kessler

15

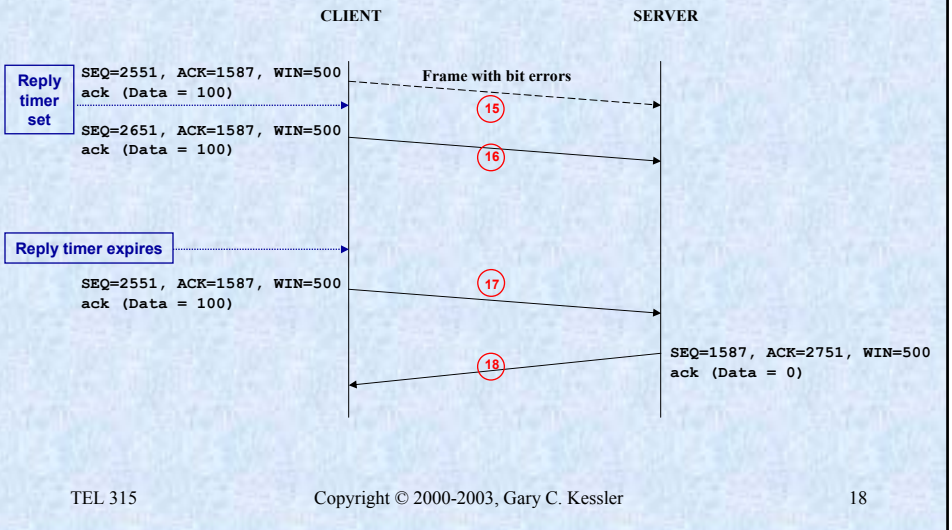
# Data Exchange and Acknowledgement



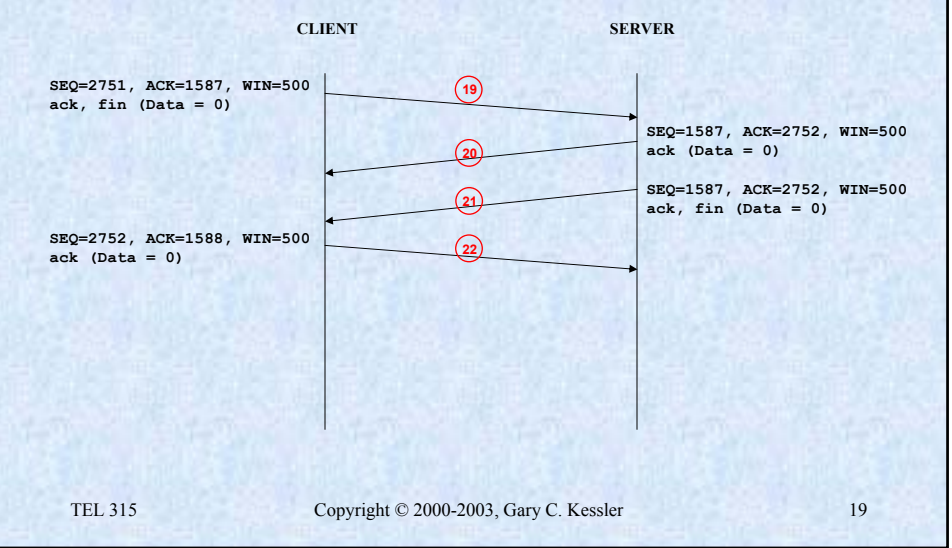
# Data Exchange and Flow Control



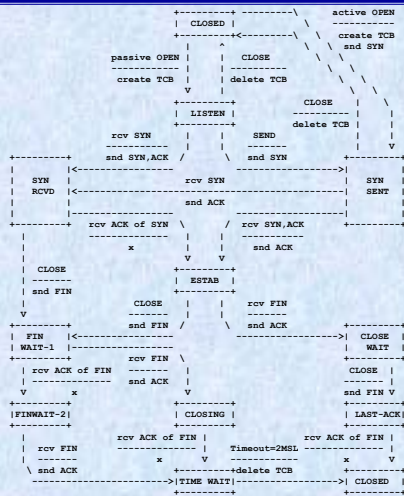
# Errors, Timeouts, and Retransmission



# Terminating the Connection



# TCP State Diagram



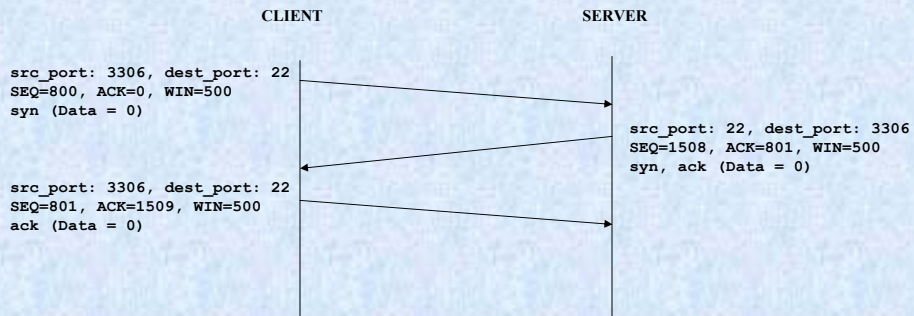
TEL 315

Copyright © 2000-2003, Gary C. Kessler

Fig. 18.12, pg. 241

20

# Normal TCP Connection



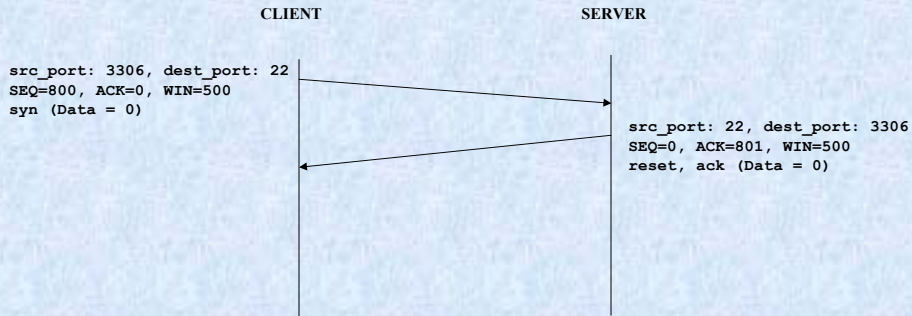
**CLIENT attempts to connect to SERVER port 22 (SSH). SERVER has an SSH daemon so it is listening on port 22.**

TEL 315

Copyright © 2000-2003, Gary C. Kessler

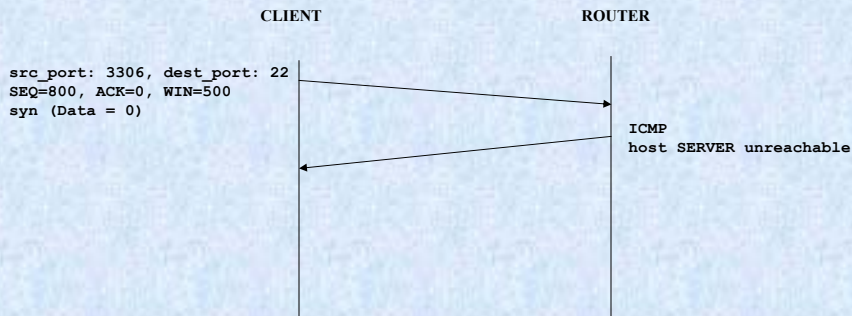
21

# Host Not Listening on TCP Port



**CLIENT attempts to connect to SERVER port 22 (SSH). SERVER does not have an SSH daemon, so it is not listening on port 22.**

# Host Does Not Exist



**CLIENT attempts to connect to SERVER port 22 (SSH). SERVER is shut off right now, so its IP address does not exist on any network.**

## TCP and Long Delays (1)

---

- Bandwidth Delay Product (Fig. 24.5)
  - »  $capacity (b) = bandwidth (bps) * round-trip\ time (s)$
- TCP standard extensions (RFC 1323):
  - » Window scaling
    - Effective 32-bit Window value, or 4GB window
    - Minimum windows size = bandwidth (B/sec) \* RTT (sec)
  - » Selective ACK
  - » Large (64 bit) sequence numbers
    - 56 kbps link: 32-bit sequence numbers wrap in 7 days
    - 10 Gbps SONET link: 32-bit sequence numbers wrap in a few seconds; 64-bit sequence numbers wrap in ~468 years
    - 10 Tbps DWDM link: 64-bit sequence numbers wrap in ~171 days

## TCP and Long Delays (2)

---

- Use timestamp option to track round-trip time per segment rather than per window
- Don't start small with slow start
  - » Initial Window size is 3-4 segments rather than 1-2
- Explicit Congestion Notification (RFC 3168)
  - » Provides mechanism so that one system can advertise that it is cutting the size of its congestion window in half
  - » Receiver learns more quickly about congestion and reduces lost transmissions (which only adds to congestion!)

# Packet Decode

## Decoding a frame.....

```
00 90 27 8E A0 B2 00 AA 00 BB DE E8 08 00 45 00
00 35 2C 31 40 00 20 06 22 EC 83 6B 02 08 83 6B
02 C8 06 B0 00 6E 23 7D 3B 26 09 A5 90 F7 50 18
21 F6 67 E7 00 00 50 41 53 53 20 73 65 63 72 65
74 0D 0A A0 1B C2 3D
```

We started by knowing that this is an Ethernet-type frame. The EtherType field value 0x0800 indicates IPv4. The IP Header's Protocol field value 0x06 indicates TCP...

# Packet Decode (TCP)

## TCP segment decode

Hex	Binary	
06	0000 0110	} Source port (1712)
B0	1011 0000	
00	0000 0000	
6E	0110 1110	} Destination port (110/POP3)
23	0010 0011	
7D	0111 1101	} Sequence number (595409702)
3B	0011 1011	
26	0010 0110	
09	0000 1001	
A5	1010 0101	} Acknowledgement number (161845495)
90	1001 0000	
F7	1111 0111	
50	0101 ....	Data offset (5 32-bit words/20 bytes)
....	0000	Reserved

# Packet Decode (TCP cont.)

---

## TCP segment decode (cont.)

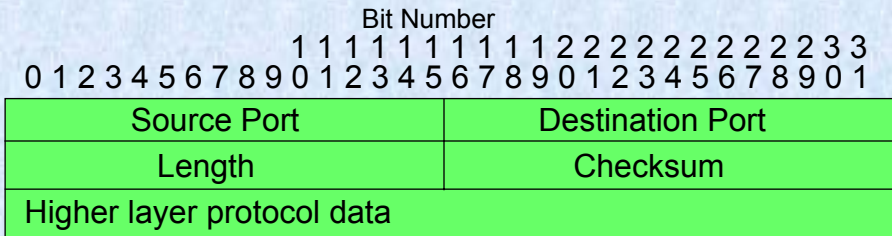
Hex	Binary	
18	00.. ....	Reserved
	..0. ....	Urgent pointer significant (no)
	...1 ....	Acknowledgement field significant (yes)
	.... 1...	Push data (yes)
	.... .0..	Reset (no)
	.... ..0.	Synchronize sequence numbers (no)
	.... ...0	Finish flag (no)
21	0010 0001	} Window size (8694 bytes)
F6	1111 0110	
C7	1100 0111	
E7	1110 0111	} Checksum
00	0000 0000	} Urgent pointer (0)
00	0000 0000	

# UDP

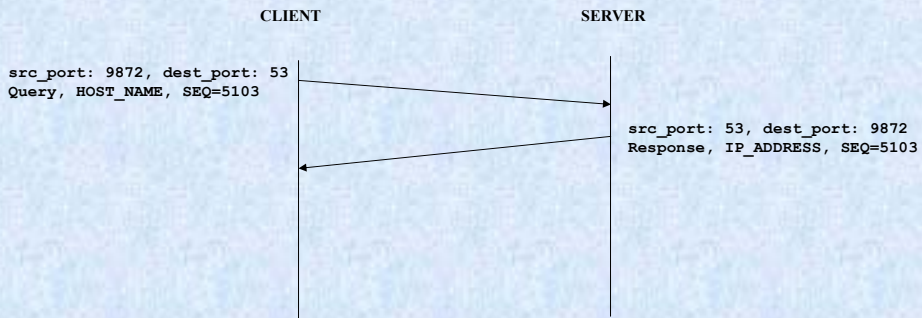
---

- Provides limited Transport Layer functionality
  - » End-to-end connectionless, unreliable datagram service
  - » Main function is to identify application
- Data unit: Datagram
- Operates over IP (Protocol = 17 = 0x11)
- Base document: RFC 768 (STD 6)

# UDP Datagram Format

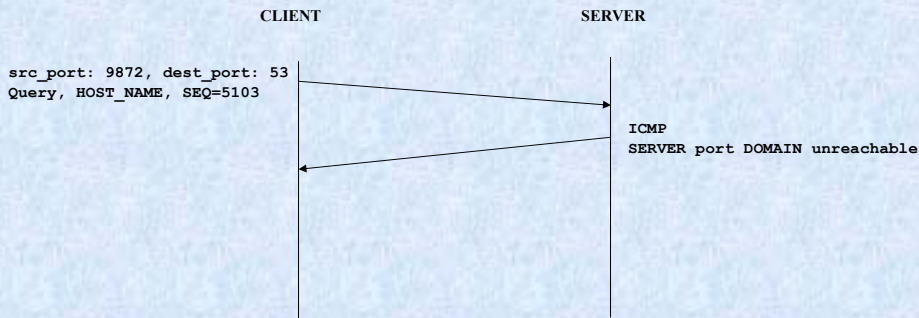


# Normal UDP Communication



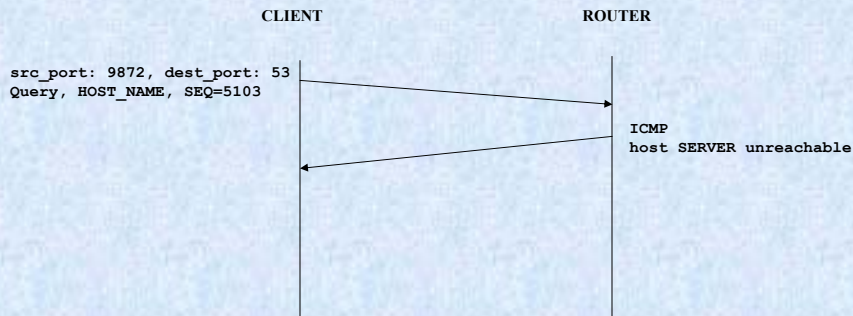
**CLIENT attempts to connect to SERVER port 53 (DNS). SERVER runs BIND so it responds to the name query.**

# Host Not Listening on UDP Port



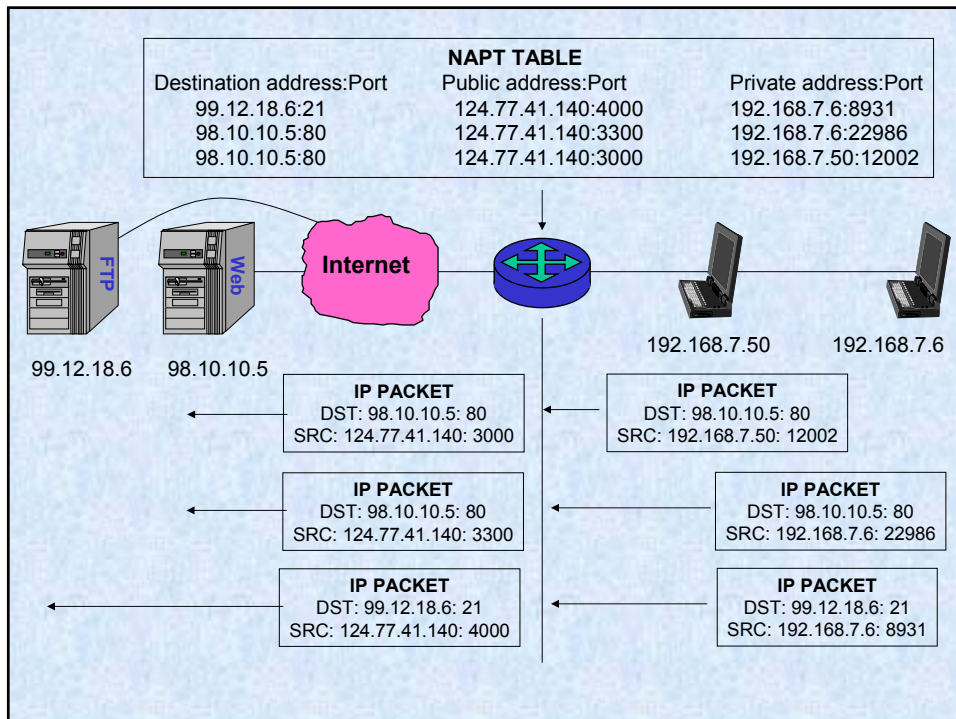
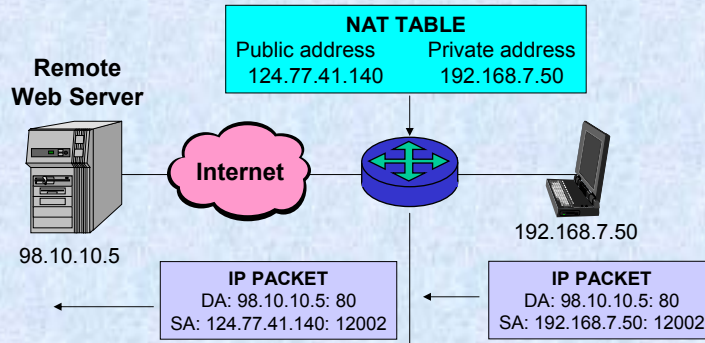
**CLIENT attempts to connect to SERVER port 53 (DNS). SERVER does not run BIND, so it is not listening on port 53.**

# Host Does Not Exist

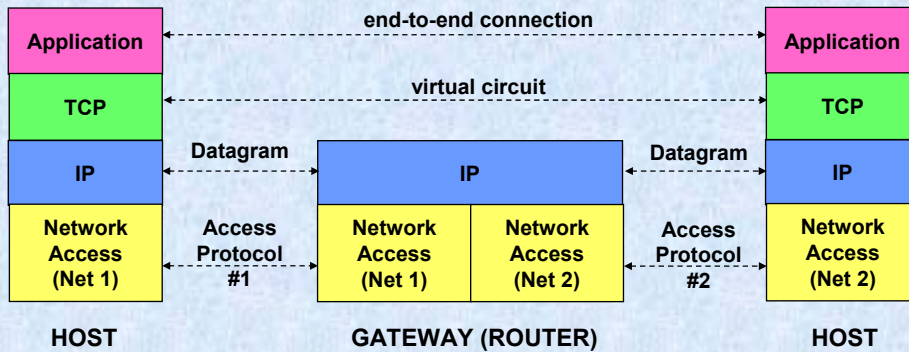


**CLIENT attempts to connect to SERVER port 53 (DNS). SERVER is shut off right now, so its IP address does not exist on any network.**

# Network Address Translation



# Protocol Layer Relationship



# TCP/UDP Protocol Vulnerabilities

- TCP
  - » TCP splicing (aka session hijacking), ISN guessing, small packet attack, SYN attack
- UDP
  - » Connectionless; easy to attack

# ISN/Address Spoofing Attack

---

- **Attacker** attacks **victim** by spoofing a host address trusted by **victim** to access root directory
  - » SYN flood against **trusted host** keeps it off the air.
  - » ISN guessing allows **attacker** to send correct packets to **victim** even though **attacker** not getting responses.
  - » After TCP connection established, **attacker** logs on and issues a `cat "+ +" >/.rhosts` command so that **victim** will trust any user to be root.
  - » **Attacker** logs on as root, fixes log files, and "owns" **victim**.

# What Ports Are Open?

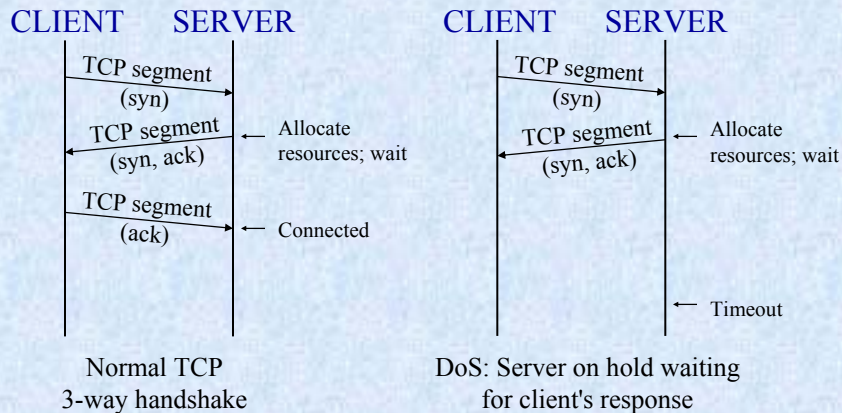
---

- `netstat`

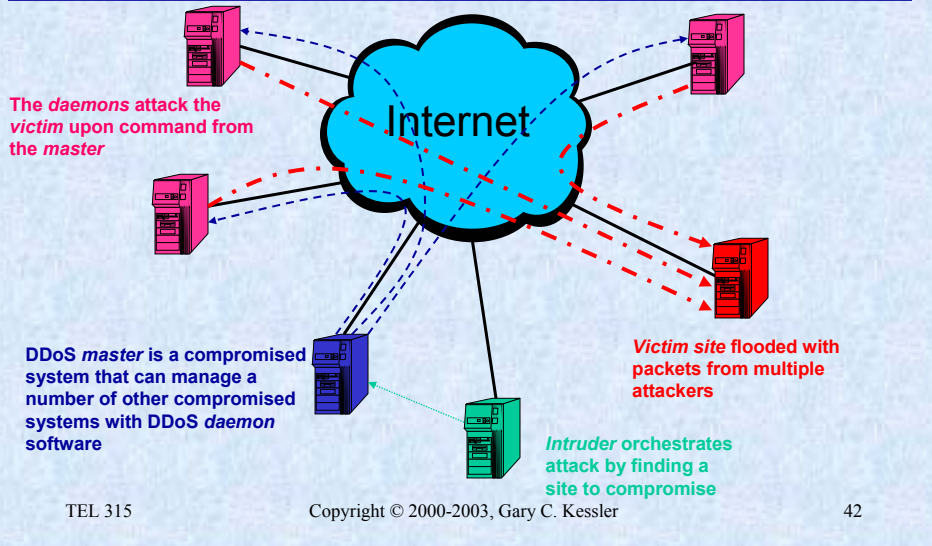
## Disable Unused Services

- Since nearly all protocols have vulnerabilities (whether known today or not!), it is important to remove or disable *all* unused services
- **Example:** *echo* (7/tcp) and *chargen* (19/tcp)
  - » Crafting a packet with the destination port = 7 and source port = 19 will cause endless loop and eventually consume all system resources

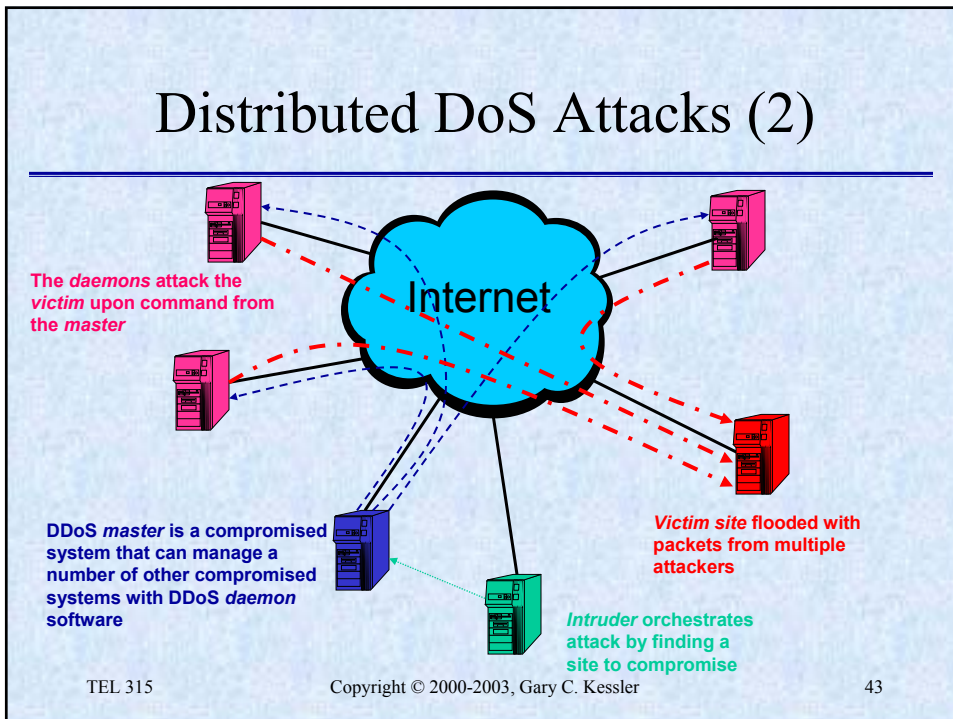
## TCP SYN DoS Attack



# Distributed DoS Attacks (1)



# Distributed DoS Attacks (2)



## Distributed DoS Tools

---

- Many DDos "tools" available
  - » Trinoo (aka Trin00), Tribe Flood Network (TFN), Stacheldraht, TFN2K, Trinity, Shaft, fapi, Trank
- Communication via TCP, UDP, and/or ICMP Echo/Echo Reply messages
- Tools can be used to remotely launch a variety of attacks, including UDP flood, TCP SYN flood, ICMP Echo Request flood, and SMURF

## DDoS Communication

---

<b>DDoS Tool</b>	<b>Intruder-to-master Communication</b>	<b>Master-to-daemon Communication</b>	<b>Daemon-to-master Communication</b>
Trinoo	27665/tcp	27444/udp	31335/udp
Tribe Flood Network (TFN)	ICMP Echo/Echo Reply	ICMP Echo Reply	ICMP Echo/Echo Reply
Stacheldraht	16660/tcp	65000/tcp	ICMP Echo Reply
Trinity	6667/tcp	6667/tcp (also 33270/tcp)	n/a
Shaft	20432/tcp	18753/udp	20433/udp

# DDoS Defenses

---

- Main defense requires cooperation: Networks and ISPs must perform ingress/egress filtering and block directed broadcast messages
- System admins and users must ensure that no unknown services are running
- Maintain up-to-date software patches
- Monitor traffic and logs, and use an IDS

---

# APPENDIX

## TCP Performance Management

## TCP Congestion Control (1)

---

- TCP is a Go-Back-N protocol with ACKs and a transmission timeout mechanism
- Several enhancements have been added
  - » Tahoe
  - » Reno
  - » SACK
  - » NewReno

## TCP Congestion Control (2)

---

- TCP is a Go-Back-N protocol with ACKs and a transmission timeout mechanism
- Several enhancements have been added
  - » Tahoe
  - » Reno
  - » SACK
  - » NewReno

# Slow Start

---

- Slow start (§20.6)
  - » Designed to match transmission speed of this TCP connection to the capacity of the network
  - » Sender has a congestion window (*cwnd*) that specifies the number of bytes that it can transmit without ACK
  - » Sender can send up to  $\min(cwnd, offered\ window)$  bytes
  - » *cwnd* initially 1 segment; incremented every time an ACK arrives (yielding exponential growth)

# Congestion Avoidance

---

- Congestion avoidance (§21.6)
  - » Designed to minimize packet loss due to network congestion
  - » Slow start threshold (*ssthresh*) is segment size where congestion last occurred (initially 65,535); operate in slow start mode while  $cwnd \leq ssthresh$ , otherwise operate in congestion avoidance mode
  - » In congestion avoidance mode, increment *cwnd* by  $1/cwnd$  whenever an ACK is received
  - » Congestion is receiving 3 duplicate ACKs with outstanding data
    - When encountered, *ssthresh* set to half of  $\min(cwnd, offered\ window)$  bytes and *cwnd* set to  $(ssthresh + 3 * \text{segment size})$  bytes
    - Retransmit outstanding data even if retransmit timer has not expired (*fast retransmit*)

## Selective Acknowledgement (1)

---

- TCP SACK Option (RFC 2018)
- Allows receiver to accept out-of-sequence data
  - » SACK option includes range of bytes correctly received
- Reduces the amount of retransmission

## Selective Acknowledgement (2)

---

- TCP SACK Option (RFC 2018)
- Allows receiver to accept out-of-sequence data
  - » SACK option includes range of bytes correctly received
- Reduces the amount of retransmission

# Nagle Algorithm

---

- Nagle algorithm (§19.4, RFC 896)
  - » Some applications PUSH small amounts of data, yielding *tinygrams* (e.g., rlogin asynch character generates a 41B packet)
  - » Nagle aggregates data and only sends when there is no outstanding data
  - » Can be disabled for real-time applications that require the PUSH data (e.g., X-Windows and mouse movements)

# Silly Window Syndrome

---

- SWS (§22.3, RFC 813) is the exchange of tiny amounts of data instead of full segments
  - » Can be caused by receiver advertising a small window rather than waiting to advertise a larger window, or the sender sending small amounts of data instead of waiting for more data to send
  - » Rule for receivers: Increment offered window by a full segment
  - » Rule for senders: Only send full segments, segment with >50% of the largest offered window seen, or everything that is in the buffer if there is no outstanding data

# Appropriate Byte Counting

---

- TCP Congestion Control with Appropriate Byte Counting (RFC 3465)
  - » Proposed experimental protocol, Feb. 2003
  - » Increases a sender congestion window (cwnd) based upon the number of bytes acknowledged rather than by the number of ACKs received (which is the current scheme, per RFC 2581)
  - » With ABC, cwnd better reflects the true traffic shape, optimizes use of bandwidth, and still prevents an overly-aggressive sender