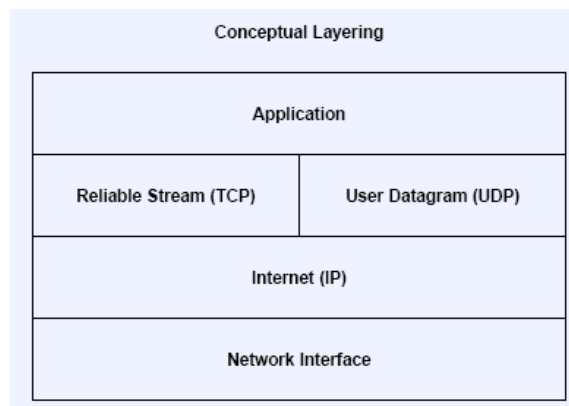


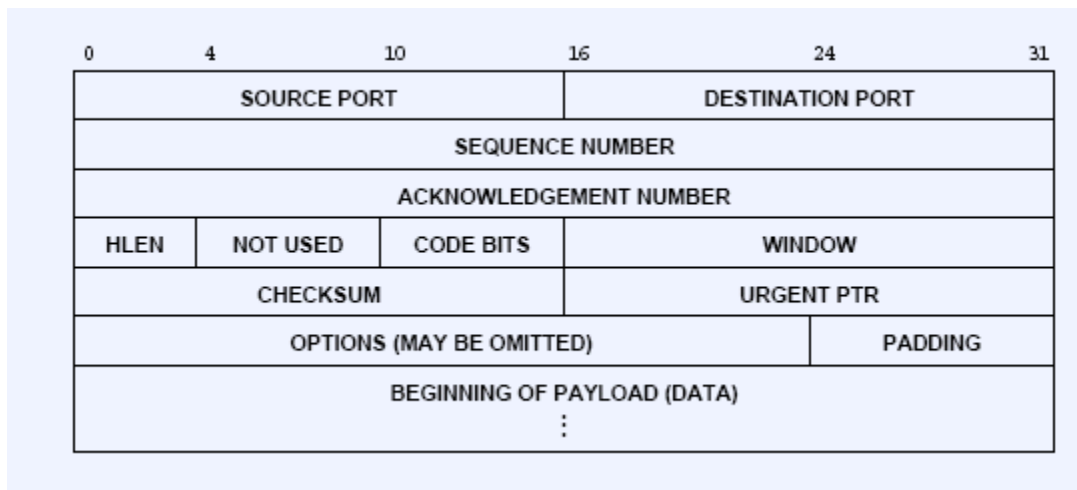
TCP Protocols

(1) TCP Protocol (Transmission Control Protocol)

- ❖ The Need for Stream Delivery
 - Out of order packet delivery
 - Packet delay
 - Packet loss
 - Packet duplicates
- ❖ Properties of TCP
 - Stream Orientation
 - TCP thinks of the data as a stream of bits, divided into 8-bit octets
 - The stream delivery service on the destination machine passes to the receiver exactly the same sequence of octets that the sender passes to it on the source machine.
 - Virtual Circuit Connection
 - Buffered Transfer
 - When transferring data, each application uses whatever size pieces it finds convenient, which can be as small as a single octet.
 - The protocol software is free to divide the stream into packets independent of the pieces the application program transfer.
 - To make transfer more efficient and to minimize network traffic, implementations usually collect enough data from a stream to fill a reasonably large datagram before transmitting it.
 - “Push” mechanism can force a transfer (and delivery) without buffering.
 - Unstructured Stream
 - TCP does not honor structured data streams.
 - Application programs must understand stream content and agree on stream format before they initiate a connection.
 - Full Duplex Connection: transfer in both directions
 - Reliability
 - Positive Acknowledgement with Retransmission.
- ❖ Layering of the three major protocols

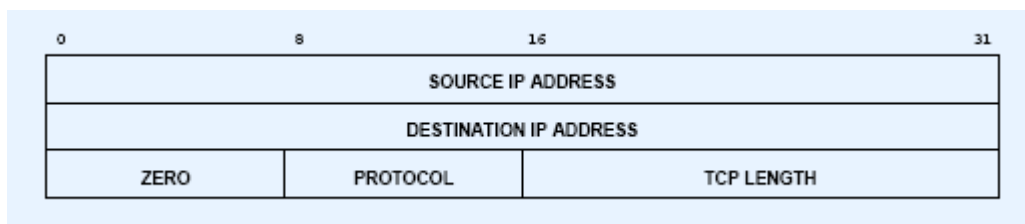


- ❖ TCP ports, connections, and endpoints
 - Endpoint of communication is application program
 - TCP uses port number to identify application
 - TCP connection between two endpoints is identified by four items
 - Sender's IP address
 - Sender's protocol port number
 - Receiver's IP address
 - Receiver's protocol port number
 - A given TCP port number can be shared by multiple connections on the same machine, e.g., the following two connections share the same destination point:
 - (18.26.0.36, 1069) and (128.230.208.110 22)
 - (128.10.2.3, 1184) and (128.230.208.110 22)
- ❖ Reserved TCP Port Numbers
 - Port numbers range from 0 to 65536.
 - Port numbers 0 to 1024 are reserved for privileged services and designated as well-known ports (in other words, only root has the permission to use these reserved port numbers).
 - Example
 - 22: SSH, 23: telnet, 80: http
- ❖ TCP Segment Format



- ❖ HLEN: length of the segment header measured in 32-bit multiples (it is needed because the OPTIONS field varies in length).
- ❖ CHECKSUM: `checksum` (Pseudo header + TCP header + TCP data)

Figure: TCP Pseudo header



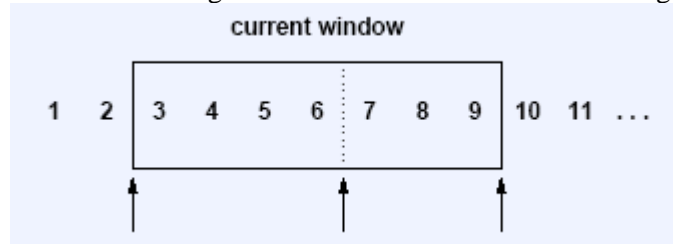
- ❖ **CODE BITS:** specify the purpose and contents of the segment

Figure: Bits of the CODE field in the TCP header.

| Bit (left to right) | Meaning if bit set to 1 |
|---------------------|---|
| URG | Urgent pointer field is valid |
| ACK | Acknowledgement field is valid |
| PSH | This segment requests a push |
| RST | Reset the connection |
| SYN | Synchronize sequence numbers |
| FIN | Sender has reached end of its byte stream |

- ❖ **Sliding Window Mechanism:**

- Used for flow control
- Operate at the octet level, not at the segment or packet level.
- An example of the TCP sliding window is illustrated in the following figure:



- In the above example (sender's window)
 - Octets through 2 have been sent and acknowledge
 - Octets 3 through 6 have been sent but not acknowledged
 - Octets 7 through 9 have not been sent but will be sent without delay
 - Octets 10 and higher cannot be sent until the window moves
- Receiver's window
 - Receiver also maintains a window. However, this window size is defined by the receiver. This size might not be the same as the sender's size.
 - The receiver's window indicates how many out-of-band octets the receiver is willing to accept.
 - If a packet is out of the receiver's window, the packet will be dropped.
- TCP allows the window size to vary over time.
 - Each acknowledgement, which specifies how many octets have been received, contains a *window advertisement* (the WINDOW field) that specifies how many additional octets of data the receiver is prepared to accept. We think of the window advertisement as specifying the receiver's current buffer size.
 - In response to an increased window advertise, the sender increases the size of its sliding window and proceeds to send octets that have not been acknowledged.
 - In response to a decreased window advertisement, the sender decreases the size of its window and stops sending octets beyond the boundary.
 - Window size can be zero (receiver cannot accept additional data at present).

- ❖ Out of Band Data
 - Although TCP is a stream-oriented protocol, it is sometimes important for the program at one end of a connection to send data *out of band*, without waiting for the program at the other end of the connection to consume octets already in the stream.
 - Example: Control-C *interrupts* or *aborts* signals.
 - URG code bit is used to specify such type of TCP data.
 - URGENT POINTER: specify the position in the segment where urgent data ends.

- ❖ ACKNOWLEDGEMENT NUMBER: specify the sequence number of the next octet that the receiver expects to receive
 - At any time, the receiver will have reconstructed zero or more octets contiguously from the beginning of the stream, but may have additional pieces of the stream from datagrams that arrived out of order. The receiver always acknowledges the *longest contiguous prefix of the stream that has been received correctly* (not including those that are out of order).

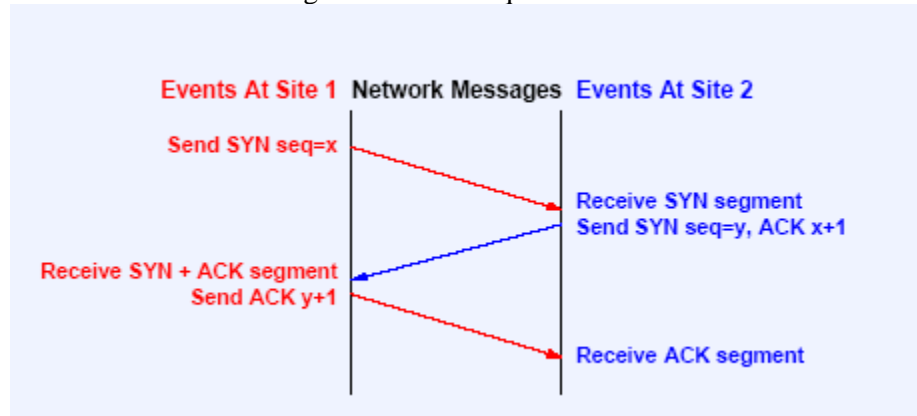
- ❖ Timeout and Retransmission
 - Every time TCP sends a segment, it starts a timer and waits for acknowledgement.
 - If the timer expire, TCP assumes that the segment was lost or corrupted and retransmits it.
 - *Adaptive retransmission algorithm*: TCP monitors the performance of each connection and deduces reasonable values for timeouts.

- ❖ Congestion Control
 - TCP uses another window, called *congestion window*. The actual window is the following
$$\text{Allowed_window} = \min(\text{receiver_advertisement}, \text{congestion_window})$$
 - When congestion occurs, router begins to enqueue datagrams. When routers' queues reach their capacity, routers start to drop datagrams. This causes TCP retransmissions.
 - Retransmissions aggravate congestion instead of alleviating it.
 - To avoid congestion, the TCP standard now recommends using two techniques:
 - *Multiplicative Decrease Congestion Avoidance*: Upon loss of a segment, reduce the congestion window by half (down to a minimum of at least one segment). For those segments that remain in the allowed window, backoff the retransmission timer exponentially.
 - *Slow-Start (Additive) Recovery*: whenever starting traffic on a new connection or increasing traffic after a period of congestion, start the congestion window at the size of single segment and increase the congestion window by one segment each time an acknowledgement arrives.

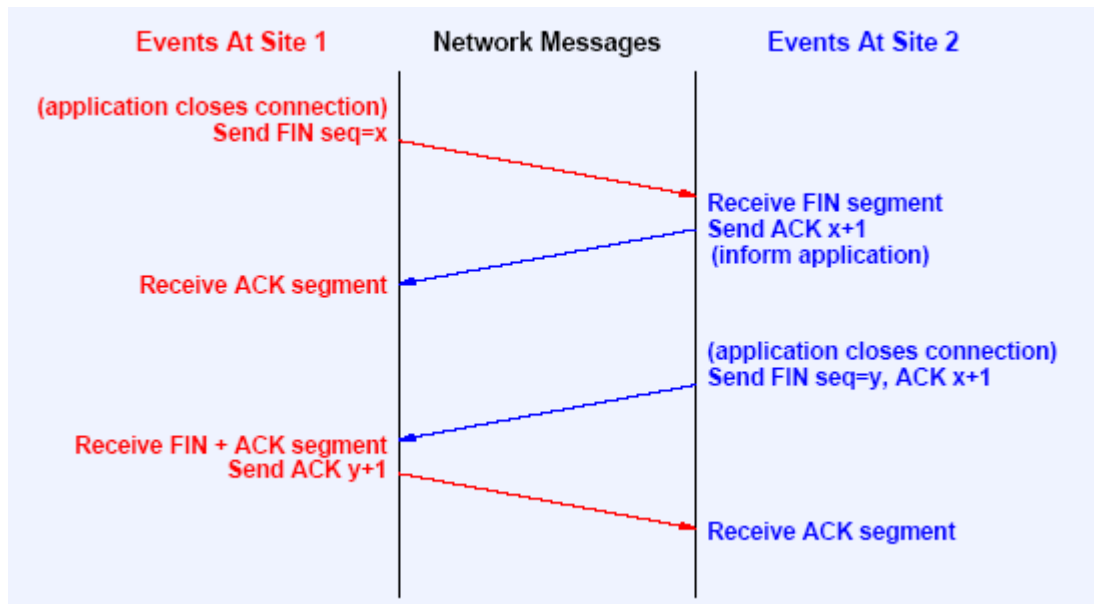
- ❖ Forcing Data Delivery
 - Consider using a TCP connection to pass characters from an interactive terminal to a remote machine. The user expects instant response to each keystroke. If the sending TCP buffers the data, response may be delayed, perhaps for hundreds of keystrokes.
 - TCP provides a *push* operation that an application program can use to force delivery of octets currently in the stream without waiting for the buffer to fill.
 - In addition, a segment with the PSH bit set is sent to the receiver, so the data will be delivered to the application program on the receiving end without waiting for the buffer to be filled.

- ❖ Establishing a TCP Connection: three-way handshake
 - The 3-way handshake accomplishes two important functions

- It guarantees that both sides are ready to transfer data.
- It allows both sides to agree on initial sequence numbers.



- ❖ Initial Sequence Numbers
 - Each machine must choose an initial sequence number at random.
 - Non-random sequence numbers have security consequence (discussed later).
- ❖ Closing a TCP Connection
 - When an application program tells TCP that it has no more data to send, TCP will close the connection *in one direction* by sending a segment with the FIN bit set.
 - Once a connection has been closed in a given direction, TCP refuses to accept more data for that direction. Meanwhile, data can continue to flow in the opposite direction until the sender closes it.
 - When both directions have been closed, TCP at each endpoint deletes its record of the connection.



- ❖ TCP Connection Reset
 - For normal shutting down a connection, use FIN.

- Sometimes abnormal conditions arise that force an application program or the network software to break a connection. TCP provides a *reset* facility for such abnormal disconnections.
 - Segment with the RST bit in the CODE field set.
 - The other side responds to a reset segment immediately by aborting the connection.
 - A reset is an instantaneous abort that means that transfer in both directions ceases immediately, and resources such as buffers are released.
-

(2) TCP Attacks

❖ SYN Flooding

- An attacker sends many SYN packets to create multiple connections without ever sending an ACK to complete the connection.
- The victim has to keep the half-opened connection in its memory for certain amount of time (e.g. 75 seconds).
- If there are so many of these malicious packets, the victim quickly runs out of memory.
- Denial of Service (DoS) attack
- Those SYN packets usually use spoofed IP addresses.

❖ TCP Session Hijacking (Mitnick Attack)

- Discussion: Machine A and B. If a user `rlogin` from B to A, A will not ask for a password (e.g. `.rhosts`). You are an attacker. Can you login to A from your own machine?
 - Hint 1: sequence number
 - Hint 2: B's role
- Guessing the sequence numbers Session Hijacking
- SYN flooding B.
- Defense methods
 - Encryption is the only complete defense
 - Checksum carry a keyed hash.

❖ TCP RST Attacks

- Attackers inject an RST segment into an existing TCP connection, causing it to be closed.
- The TCP Reset attack is made possible due to the requirements that a TCP endpoint must accept out of order packets that are within the range of a window size, and the fact that Reset flags should be processed immediately.
- What are the difficulties of spoofing a RST packet to break a remote connection?
 - Sequence number of the connection
 - Source port of the connection (destination port is usually well known for some applications, e.g. SSH uses 22)

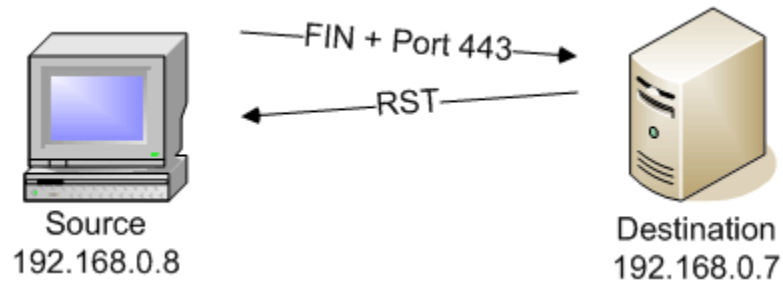
❖ TCP Port Scanning

- TCP SYN scan
- TCP connect() scan

- FIN, Xmas Tree or Null scan: closed ports are required to reply with an RST, while open ports must ignore the packets in question (RFC).

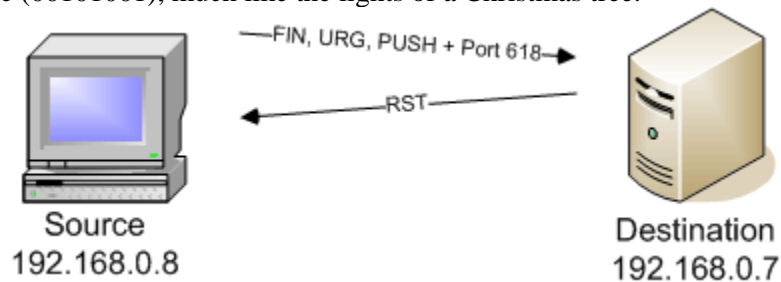
- **FIN Scan**

The FIN scan's 'stealth' frames are unusual because they are sent to a device without first going through the normal TCP handshaking. If a TCP session isn't active, the session certainly can't be formally closed!



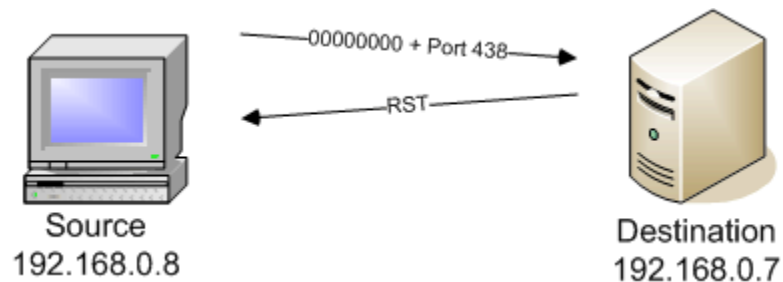
- **The Xmas Tree Scan**

The Xmas tree scan sends a TCP frame to a remote device with the URG, PUSH, and FIN flags set. This is called a Xmas tree scan because of the alternating bits turned on and off in the flags byte (00101001), much like the lights of a Christmas tree.



- **NULL Scan**

The null scan turns off all flags, creating a lack of TCP flags that should never occur in the real world.



- ❖ **Fingerprinting hosts**

- What makes it possible?

- Different OS choose unique values for certain fields, such as TTL, TOS, TCP window size, TCP options.
- Different OS may choose different way to response (not exactly follow RFC).

- Tool: `nmap -O -v host` : identify OS version and tell you how difficult it is to predict initial sequence #.
 - The FIN probe: Here we send a FIN packet (or any packet without an ACK or SYN flag) to an open port and wait for a response. The correct RFC 793 behavior is to NOT respond, but many broken implementations such as MS Windows, BSDI, CISCO, HP/UX, MVS, and IRIX send a RESET back. Most current tools utilize this technique.
 - The BOGUS flag probe -- Queso is the first scanner to use this clever test. The idea is to set an undefined TCP "flag" (bit 7 or 8, counting from the left) in the TCP header of a SYN packet. Linux boxes prior to 2.0.35 keep the flag set in their response. However, some operating systems seem to reset the connection when they get a SYN+BOGUS packet. This behavior could be useful in identifying them. Update: Bit 8 (and 9) are now used as the "ECN field" for TCP congestion control.
 - TCP ISN Sampling -- The idea here is to find patterns in the initial sequence numbers chosen by TCP implementations when responding to a connection request. These can be categorized in to many groups such as the traditional 64K (many old UNIX boxes), Random increments (newer versions of Solaris, IRIX, FreeBSD, Digital UNIX, Cray, and many others), True "random" (Linux 2.0.*, OpenVMS, newer AIX, etc). Windows boxes (and a few others) use a "time dependent" model where the ISN is incremented by a small fixed amount each time period.
 - ICMP Message Quoting -- The RFCs specify that ICMP error messages quote some small amount of the IP packet that causes various errors. For a port unreachable message, almost all implementations send only the required IP header + 8 bytes back. However, Solaris sends back a bit more and Linux sends back even more than that. The beauty with this is it allows `nmap` to recognize Linux and Solaris hosts even if they don't have any ports listening.
 - IPID sampling
 - SYN Flood Resistance
 - Overlapping Fragmentation Handling
 - Don't Fragment bit
 - TCP Initial Window
 - ICMP Error Message Quenching
- ❖ The Security of the Initial Sequence Number (ISN)
- If an attacker can find out current sequence number that is being used by an existing TCP connection, it can inject a valid TCP segment into the existing TCP connection.
 - If the attacker is within the same LAN, it can sniff the sequence number.
 - If the attacker is not within the same LAN, it has to guess the sequence number.
 - To guess ISN:
 - All possible values for ISN: 2^{32} .
 - We only need to make sure that the guessed ISN is within the receiver's current window; otherwise, the TCP packet with this guessed ISN will be discarded by the receiver.

- If 16K window size is used, on average, it only takes $2^{32} / 2^{14} = 2^{18} = 262,144$ tries to hit the window.
 - With a T1 line at 4,370 packets a second, the attacker would be able to exhaust all possible windows within only 60 seconds.
- Initial window size for various operating systems (from Watson [2]). The packets required for a successful guess are based on the equation: **($2^{32} / \text{Initial Window Size}$)**

| Operating System | Initial Window Size | Packets Required |
|---------------------------------------|---------------------|------------------|
| Windows 2000 5.00.2195 SP4 | 64512 | 66,576 |
| Windows XP Home Edition SP1 | 64240 | 66,858 |
| HP-UX 11 | 32768 | 131,071 |
| Nokia IPSO 3.6-FCS6 | 16384 | 262,143 |
| Cisco 12.2(8) | 16384 | 262,143 |
| Cisco 12.1(5) | 16384 | 262,143 |
| Cisco 12.0(7) | 16384 | 262,143 |
| Cisco 12.0(8) | 16384 | 262,143 |
| Windows 2000 5.00.2195 SP1 | 16384 | 262,143 |
| Windows 2000 5.00.2195 SP3 | 16384 | 262,143 |
| Linux 2.4.18 | 5840 | 735,439 |
| Efficient Networks 5861 (DSL) v5.3.20 | 4096 | 1,048,575 |

❖ Adjusting Default TCP Window Size

- Windows 2000: Tuning of Window size can be accomplished by adjusting registry settings. The registry keys of interest can be found in the registry at this location:
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters](#)
- Solaris: Adjusting the default TCP window size in Solaris can be accomplished using the `ndd` command.


```
# ndd -set /dev/tcp tcp_xmit_hiwat [0-65535]
# ndd -set /dev/tcp tcp_recv_hiwat [0-65535]
```
- Linux (2.4.x kernels): The following two variables can be added to the `/etc/sysctl.conf` file. The names “rmem” and “wmem” correspond to receive and transmit respectively. After settings these values, execute the “`sysctl -p`” command to have them take effect.


```
net.core.rmem_default = [0-65535]
```

```
net.core.wmem_default = [0-65535]
```

❖ Guessing the source port

- When a TCP connection is made, the combination of the source port and IP address and the destination port and IP address results in a unique fingerprint that can be used to differentiate between all active TCP connections.
- Most of the TCP attacks assume that the attacker already knows the destination port and IP address as well the source port and IP address. The destination port and IP address are easy, as they are generally published. The source IP address is also generally easy, as this is simply the client that is being spoofed. The only piece that can frequently be difficult to find is the source port.
- For example, if an operating system randomly assigns source ports from a pool that ranges from 1025 through 49,152 (such as OpenBSD), this increases the difficulty of performing a reset attack 48,127 times as the attacker would have to try their sequence attack with every possible port number. In our example with 16k windows, we determined that with known endpoints it would require 262,144 packets to guarantee a successful reset attack. However, if using random ports as we've described, it would now require 262,144 times 48,127, or 12,616,204,288 packets. An attack of that size would all but certainly be detected and dealt with before a brute force reset would occur.
- Unfortunately, most operating systems allocate source ports sequentially, including Windows and Linux. A notable exception is OpenBSD, which began randomizing source port allocation in 1996.
- The following chart represents observations of source port selection from various Operating Systems (from Watson [2]):

| OPERATING SYSTEM | OBSERVED INITIAL SOURCE PORT | OBSERVED NEXT SOURCE PORT SELECTION METHOD |
|-----------------------------|------------------------------|--|
| Cisco 12.2(8) | 11000 | Increment by 1 |
| Cisco 12.1(5) | 48642 | Increment by 512 |
| Cisco 12.0(7) | 23106 | Increment by 512 |
| Cisco 12.0(8) | 11778 | Increment by 512 |
| Windows 2000 5.00.2195 SP4 | 1038 / 1060 | Increment by 1 |
| Windows 2000 5.00.2195 SP3 | 1060 | Increment by 1 |
| Windows XP Home Edition SP1 | 1050 | Increment by 1 |
| Linux 2.4.18 | 32770 | Increment by 1 |
| Nokia IPSO 3.6-FCS6 | 1038 | Increment by 1 |

❖ ICMP Attacks against TCP

- More details can be found at <http://www.gont.com.ar/drafts/icmp-attacks-against-tcp.html>.
- Thus, for ICMP messages generated by hosts, we can only expect to get the entire IP header of the original packet, plus the first 64 bits of its payload. For TCP, that means that the only fields that will be included are: the source port number, the destination port number, and the 32-bit TCP sequence number. This clearly imposes a constraint on the possible security checks that can be performed, as there is not much information available on which to perform them.

❖ ICMP Blind Connection-reset attacks

- The Host Requirements RFC [4] states that a host SHOULD abort the corresponding connection when receiving an ICMP error message that indicates a hard error.
- A single ICMP packet could bring down all the TCP connections between the corresponding peers.
- Counter-measures
 - The following types of ICMP messages are not reasonable, should be ignored:
 - ICMP type 3 (Destination Unreachable), code 2 (protocol unreachable)
 - ICMP type 3 (Destination Unreachable), code 3 (port unreachable)
 - ICMP type 3 (Destination Unreachable), code 4 (fragmentation needed and DF bit set)
 - Delaying the connection-reset:
Rather than immediately aborting a connection, a TCP could abort a connection only after an ICMP error message indicating a hard error has been received a specified number of times, and the corresponding data have already been retransmitted more than some specified number of times.

❖ Blind throughput-reduction attacks

- The Host requirements RFC states hosts MUST react to ICMP Source Quench messages by slowing transmission on the connection. Thus, an attacker could send ICMP Source Quench (type 4, code 0) messages to a TCP endpoint to make it reduce the rate at which it sends data to the other party. While this would not reset the connection, it would certainly degrade the performance of the data transfer taking place over it.
- However, as discussed in the Requirements for IP Version 4 Routers RFC 1812, research seems to suggest ICMP Source Quench is an ineffective (and unfair) antidote for congestion. Thus, we recommend hosts to completely ignore ICMP Source Quench messages.

(3) References

Figures, texts, and tables used in this lecture notes come from the following sources.

- [1] Comer's TCP/IP slides

- [2] Slipping in the Window: TCP Reset attacks, by Paul A. Watson (http://www.osvdb.org/reference/SlippingInTheWindow_v1.0.doc).
- [3] Secrets of Network Cartography: a comprehensive guide to nmap 3.81 (<http://www.networkuptime.com/nmap/index.shtml>).
- [4] Remote OS detection via TCP/IP Stack FingerPrinting, by Fyodor, 2002 (<http://www.insecure.org/nmap/nmap-fingerprinting-article.html>).
- [5] The TCP/IP Guide (http://www.tcpiptime.com/free/t_toc.htm).
- [6] ICMP attacks against TCP, by Gont, 2004. (<http://www.gont.com.ar/drafts/icmp-attacks-against-tcp.html>)