CSE 333: Systems Programming

Section 9

IP and TCP Packets

IP Packets

- So far we've seen:
  - Data transfer with TCP-based reads and writes (no need to know about underlying packets)
  - Data transfer with UDP-based reads and writes (explicitly send and receive packets)
- IP (Internet Protocol) packets are what facilitate these two types of transfers

IP Packets

- IP packet header format (from Wikipedia):

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	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags				Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															

- Packets sent over the network all have this IP header, which indicates the version (4 for IPv4), the header length (in 4-byte words), the protocol (TCP, UDP, etc.), the source, the destination, and so forth
- What happens if the checksum isn't correct?

IP Packets

- IP packet header format (from Wikipedia):

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	Version				IHL				DSCP						ECN		Total Length															
4	32	Identification																Flags			Fragment Offset											
8	64	Time To Live								Protocol								Header Checksum															
12	96	Source IP Address																															
16	128	Destination IP Address																															
20	160	Options (if IHL > 5)																															

- The IP packet header is followed by data for whichever protocol is being used
  - For TCP, the data is a TCP header, followed by the contents of the message itself
  - For UDP, the data is just the contents of the message

TCP Packets

- TCP packet header format (from Wikipedia):

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				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if Data Offset > 5, padded at the end with "0" bytes if necessary) ...																															

- When a client connects to a server using TCP, the client and server participate in a three-way handshake using packets of this form

TCP Packets

- Initial message is a SYN:

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 M (some arbitrary number)																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum SYN bit set to high																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if Data Offset > 5, padded at the end with "0" bytes if necessary) ...																															

- The client picks an arbitrary sequence number, sets the flag for SYN, and dispatches the packet to the server

TCP Packets

* The receiver responds with a SYN/ACK:

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 N (some arbitrary number)																															
8	64	Acknowledgment number (if ACK set) M + 1																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																SYN and ACK set to high								Urgent pointer (if URG set)							
20 ...	160 ...	Options (if Data Offset > 5, padded at the end with "0" bytes if necessary) ...																															

* The server picks an arbitrary sequence number, sets the acknowledgment number to M + 1, sets the flags for both SYN and ACK, and dispatches the packet to the client

TCP Packets

* The client responds with an ACK:

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 M + 1																														
8	64	Acknowledgment number (if ACK set) N + 1																														
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size													
16	128	Checksum																Urgent pointer (if URG set)														
20 ...	160 ...	Options (if Data Offset > 5, padded at the end with "0" bytes if necessary) ...																														

* The client confirms receipt of the server's SYN, at which point the client and server have established a connection

TCP Packets

- The sequence number of future packets is based on the number of bytes sent by the client or server
- When one of them sends data, it does so with the SYN flag set
  - The receiver confirms receipt with an ACK and an acknowledgment number equal to the sender's sequence number
- The client and server terminate the connection by exchanging (and acknowledging) FIN packets
  - The RST flag can also be used to reset the connection

TCP Packets

- It seems like things can go wrong in this process, though…
  - What should the server do if it receives a SYN but the client never responds to its SYN/ACK?
  - What should the client or server do if the other side never acknowledges a FIN?

Port Scanning

- To see which ports are open on a particular machine, a simple TCP port scanner can try to connect to each one in sequence through a SYN, SYN/ACK, SYN handshake (i.e. via the connect() function)
- Some machines detect such port scans, though, and filter incoming connections from the host

Port Scanning

- There is a trick we can play, however:
  - Rather than opening a full TCP connection and then closing it, we can simply send a SYN, wait for a response, and then terminate the connection (using an RST to reset it)
  - If the target responds with a SYN/ACK, then the port is open, and if it responds with an RST, then the port is closed

Section exercise

- Finish implementing a raw socket port scanner
- You only need to write the code for processing received packets; see raw_scanner.cc
- Run the program with the -t flag for the simple TCP port scanner or -r for the raw socket port scanner, e.g. ./port_scan -t 127.0.0.1
- You'll need to run as root to use the raw socket port scanner, unfortunately (sudo ./port_scan -r …)
- Make sure you find the same ports open on your machine with the TCP scan versus the raw socket scan
- Submit raw_scanner.cc to the Dropbox when done