TCP Protocol and Its Attacks

Lecture 05

Instructor: Dr. Cong Pu, Ph.D.

cong.pu@okstate.edu

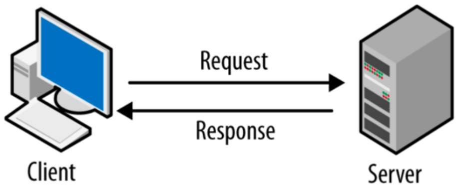
Acknowledgment: Adapted partially from course materials from Dr. Wenliang Du at Syracuse University, Dr. Fengwei Zhang at Southern University of Science and Technology, and Dr. Steven M. Bellovin at Columbia University.

Introduction

- ***Transmission Control Protocol*** (***TCP***)
  - core protocol of Internet Protocol suite
  - sits on top of Internet Protocol (IP) layer
  - provides *reliable* and *ordered* communication channel between apps. (e.g., browser, email, etc) running on network computers
- *Transport layer*: TCP and UDP (User Datagram Protocol)
  - UDP ***does not*** provide *reliable* or *ordered* communication
    - lightweight, low overhead, and ***good*** for applications that do not require *reliability* or *order*
  - TCP provides host-to-host communication services for applications
    - requires both ends to maintain a *connection* (logical)
    - ***no security mechanism*** was built into it when developed
      - eavesdropping, injection, hijacking, disconnection attacks

How TCP Protocol Works

- TCP is quite complicated, with a lot of details
  - cover enough details to understand its *security aspects*
- A pair of simple TCP programs, *client* and *server*, is used to *illustrate how TCP works*



- TCP client program:
  - sending a simple hello message to the server
- TCP server program:
  - use an existing utility to serve as the server
  - command:

    $ nc –nv -l 9090

    - start a TCP server
    - wait on port 9090
    - display whatever is sent from the client

How TCP Protocol Works: Client Program

- TCP client program:

Create a socket
Specify the type of communication
- TCP uses SOCK_STREAM
- UDP uses SOCK_DGRAM

Server info: IP addr. and Port #

Initiate the TCP connection
- connection-oriented protocol
- 3-way handshake
- uniquely identified: src. IP, src. port #, des. IP, des. port #

Do Not Forget to Close the Connection!

Send data

bonus functions for retrieving data:
read(); recv(); recvfrom(); recvmsg()

```
// Step 1: Create a socket
int sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Step 2: Set the destination information
struct sockaddr_in dest;
memset(&dest, 0, sizeof(struct sockaddr_in));
dest.sin_family = AF_INET;
dest.sin_addr.s_addr = inet_addr("10.0.2.17");
dest.sin_port = htons(9090);

// Step 3: Connect to the server
connect(sockfd, (struct sockaddr *)&dest,
        sizeof(struct sockaddr_in));

// Step 4: Send data to the server
char *buffer1 = "Hello Server!\n";
char *buffer2 = "Hello Again!\n";
write(sockfd, buffer1, strlen(buffer1));

write(sockfd, buffer2, strlen(buffer2));
```

write(); send(); sendto(); sendmsg()

How TCP Protocol Works: Server Program

- TCP server program:

Create a socket (same as the client prog.)

```
// Step 1: Create a socket
sockfd = socket(AF_INET, SOCK_STREAM, 0);

// Step 2: Bind to a port number
memset(&my_addr, 0, sizeof(struct sockaddr_in));
my_addr.sin_family = AF_INET;
my_addr.sin_port = htons(9090);
bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct
  sockaddr_in));
```

Bind to a port #

- an app. that connects with others needs to register a port # on the host computer
- when packets arrive, the OS knows which app. is the intended receiver
- *bind()*
- the server uses port 9090

- Popular servers use specific port #s which are well known
  - e.g., web server: 80 and 443
    SSH server: 22
- Client also needs to register a port #
  - no need to use specific port #
  - do not call bind() to register
    - leave it to OS who will assign

How TCP Protocol Works: Server Program

- TCP server program:

Listen for connections

```
// Step 3: Listen for connections
listen(sockfd, 5);
```

The limit of the queue (how many pending connections can be stored in the queue)

- if full, further connection requests will be dropped

- Once the socket is set up, the server call *listen()* to wait for connections
  - telling OS the app. is ready for receiving connection requests
- Once a connection request is received, the OS will go through TCP 3-way handshake protocol with the client to establish a connection
  - an established connection is placed in the queue, waiting for app. to take over the connection

How TCP Protocol Works: Server Program

- TCP server program:

Accept a connection request

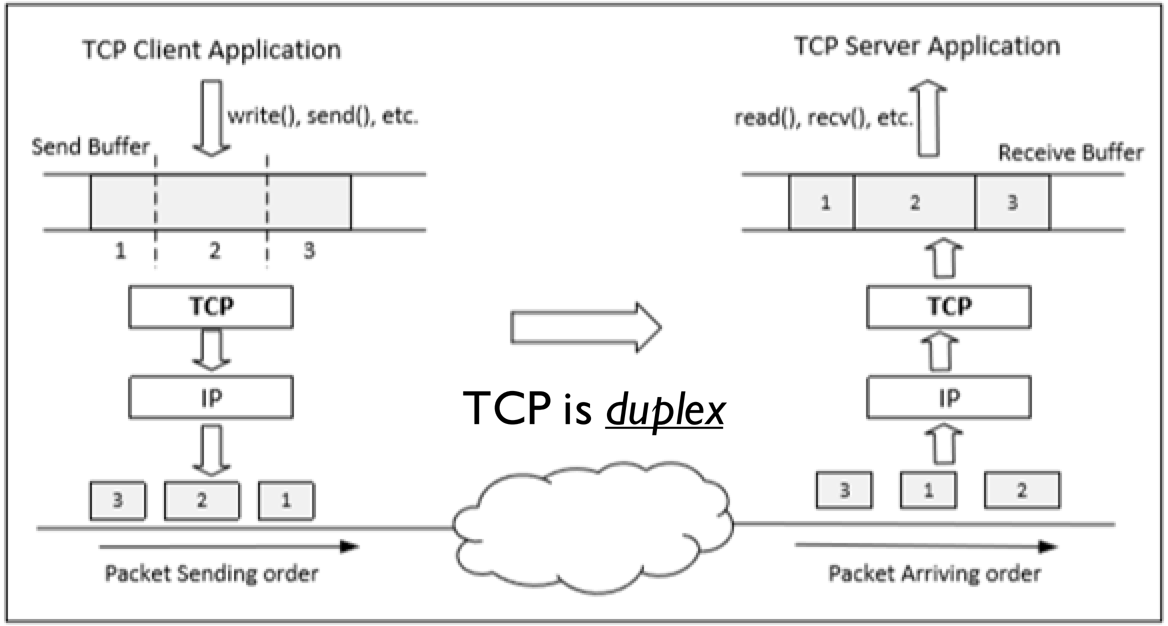
```
// Step 4: Accept a connection request
int client_len = sizeof(client_addr);
newsockfd = accept(sockfd, (struct sockaddr *)&client_addr,
  &client_len);
```

- the app. needs to specifically "accept" the connection before accessing it
  - *accept()*
    - extract the *first* connection request from the *queue*
    - ***create a new socket***
    - ***return a new file descriptor to that socket***
- the socket created before (at the beginning of the program) is only used for listening
  - not associated with any connection
  - a new socket is created when a connection is accepted
  - the app. access the connection via the new socket
  - send and receive data

Data Transmission: Under the Surface

- How TCP data are transmitted

- Once a connection is established, OS allocates *two buffers* for each end,
  - *one for sending data* (send buffer)
  - *one for receiving buffer* (receive buffer)
- When an application needs to send data out, it places data into the TCP *send buffer*.
  - the TCP code in OS decides when to send data
  - TCP usually waits until the data are enough for one packet



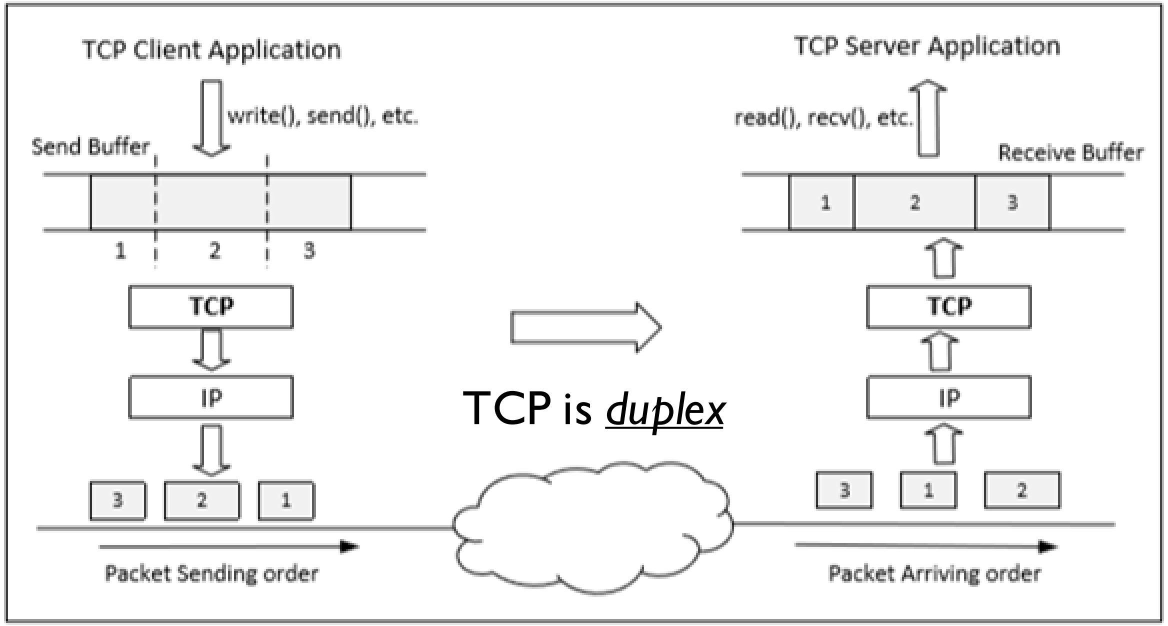
- Each ***byte*** in *send buffer* has a ***seq. #*** associated with it
  - stored in the TCP header (***seq. # field***)
- Client use the seq. # to place data in right position inside the receive buffer

Data Transmission: Under the Surface

- How TCP data are transmitted

- Once data are placed in the receive buffer, they are merged into a single data stream
  - regardless of whether they come from the same packet or different ones

- When the receive buffer gets enough data, TCP will make data available to the app.

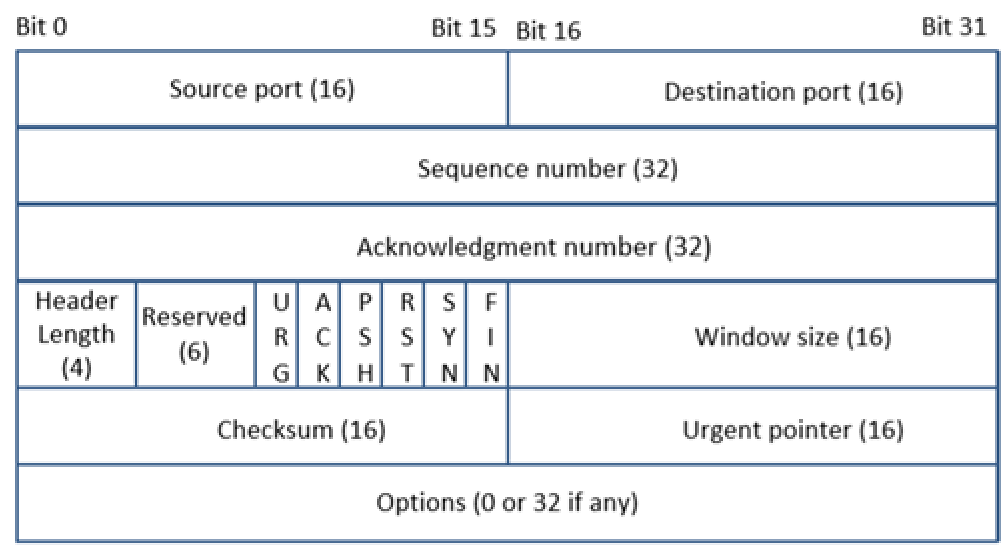


- The receiver must inform the sender that data have been received
  - send out ***acknowledgement packets (ack packets)***
  - for *performance reason*, does not acknowledge each packet that is has received
    - it tell the sender the ***next seq. #*** that it ***expects to receive*** from the sender

TCP Header (cont.)

- The TCP part of an IP packet is called TCP *segment*, which starts with a ***TCP header***, followed by a ***payload***.

The format of TCP header

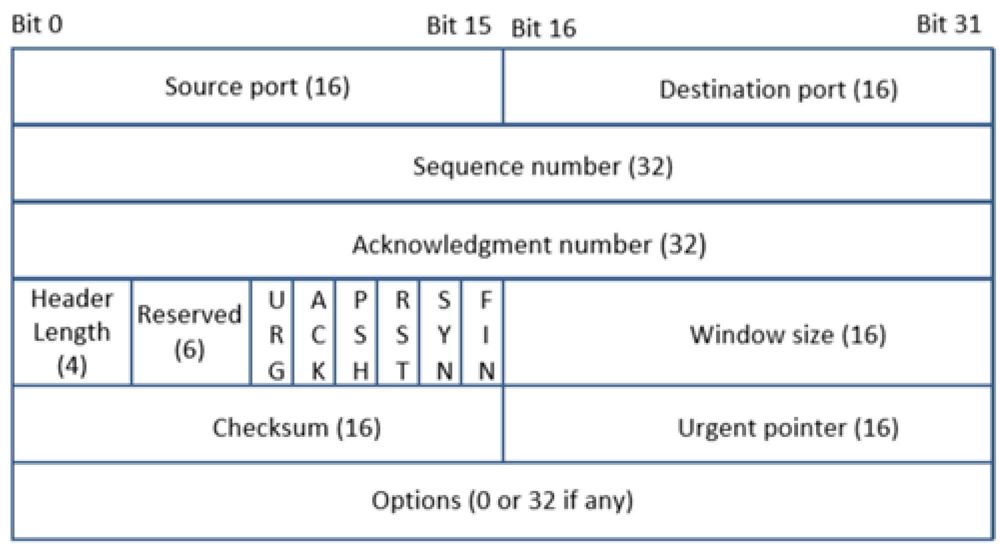


- Source and Destination port (16 bits each): Specify port numbers of the sender and the receiver.
- Sequence number (32 bits): Specifies the sequence number of the ***first byte*** in the TCP segment. If SYN bit is set, it is the initial sequence number.
- Acknowledgement number (32 bits): Contains the value of the ***next sequence number expected*** by the sender of this segment. Valid only if ACK bit is set.

TCP Header (cont.)

- The TCP part of an IP packet is called TCP *segment*, which starts with a ***<u>TCP header</u>***, followed by a ***<u>payload</u>***.

The format of TCP header



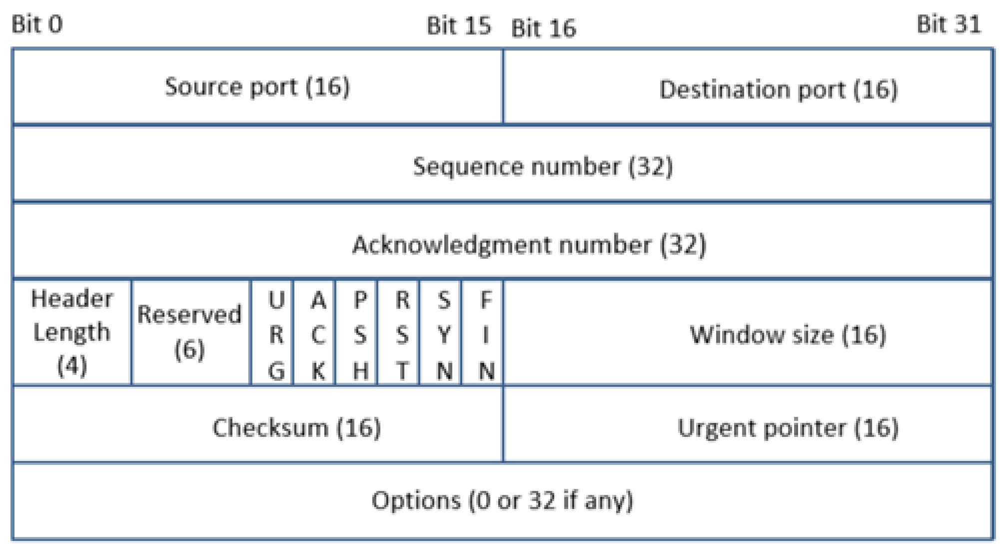
- Header length (4 bits): Length of TCP header is measured by the number of 32-bit words in the header, so we multiply by 4 to get number of bytes in the header.
- Reserved (6 bits): This field is ***<u>not used</u>***.
- Code bits (6 bits): There are six code bits, including SYN,FIN,ACK,RST,PSH and URG.
- Window (16 bits): Window advertisement to specify *<u>the number of bytes</u>* that the sender of this TCP segment is *<u>willing to accept</u>*. The purpose of this field is for flow control.

TCP Header (cont.)

- The TCP part of an IP packet is called TCP *segment*, which starts with a ***<u>TCP header</u>***, followed by a ***<u>payload</u>***.

- Checksum (16 bits): The checksum is calculated using part of IP header, TCP header and TCP data.
- Urgent Pointer (16 bits): If the URG code bit is set, the first part of the data contains urgent data (do not consume sequence numbers).
  - The urgent pointer specifies where the urgent data ends and the normal TCP data starts.
  - Urgent data is for priority purposes as they do not wait in line in the receive buffer and will be delivered to the applications immediately.

The format of TCP header

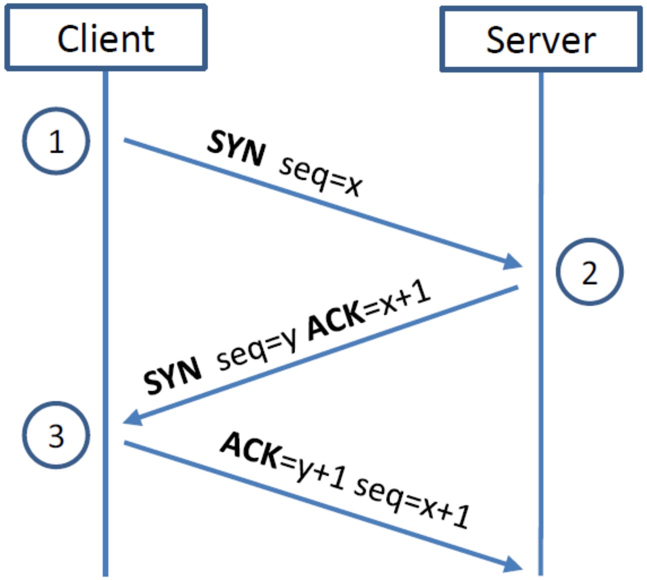


- Options (0-320 bits): TCP segments can carry a variable length of options which provide a way to deal with the limitations of the original header.

TCP 3-Way Handshake Protocol

- In TCP protocol, client and server need to establish a TCP connection *before* taking to each other
  - server needs to make itself ready for such connection by entering LISTEN state (invoking *listen()*)
  - client needs to initiate the connection via ***3-way handshake***

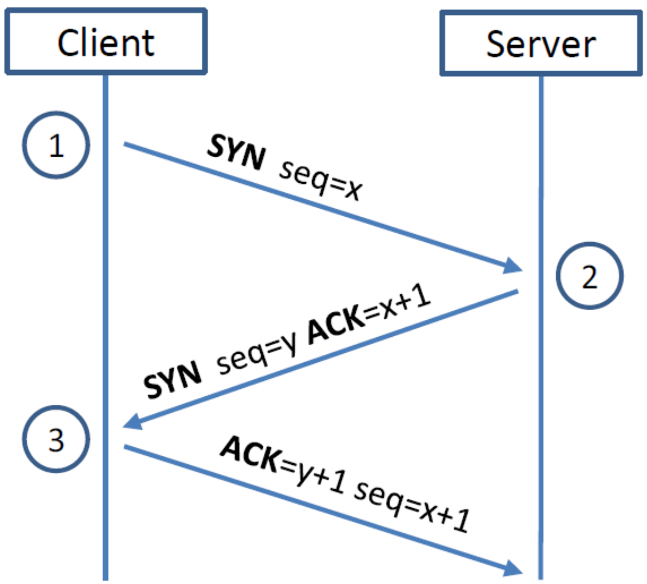
1. ***SYN Packet***: The client sends a special packet called ***SYN packet*** to the server using a *randomly generated number x* as its sequence number.
2. ***SYN-ACK Packet***: On receiving it, the server sends a reply packet (***SYN-ACK packet***) using its own *randomly generated number y* as its sequence number.
3. ***ACK Packet***: Client sends out ***ACK packet*** to conclude the handshake



TCP 3-Way Handshake Protocol

- In TCP protocol, client and server need to establish a TCP connection *before* taking to each other
  - server needs to make itself ready for such connection by entering LISTEN state (invoking *listen()*)
  - client needs to initiate the connection via ***3-way handshake***

- When the server receives the initial SYN packet, it uses TCB (Transmission Control Block) to store the information about the connection.
  - At this step, the connection is not fully established yet.
    - called half-open connection as only client-to-server connection is confirmed.
  - The server stores the TCB in a queue and take it out of the queue after receiving ACK packet from the client.



TCP 3-Way Handshake Protocol

- In TCP protocol, client and server need to establish a TCP connection _before_ taking to each other
  - server needs to make itself ready for such connection by entering LISTEN state (invoking *listen()*)
  - client needs to initiate the connection via ***3-way handshake***

- If ACK doesn't arrive, the server will resend SYN+ACK packet.
- The TCB will eventually be discarded after a certain time period or if ACK packet never comes