# CS378
# Computer Networks Lab
# Lab 5

**Ayan Tanwar**
**Roll Number: 22B0931**

# Introduction

In this lab, we implemented a simple network probing tool using UDP Datagram Sockets. The sender sends pairs of packets, while the receiver logs the time of arrival for each packet to estimate the bottleneck link speed. The code is implemented in C and utilizes functions for socket creation, sending and receiving data, and measuring time.

# Proof of $\frac{P}{t_2-t_1} = C$

To prove that $\frac{P}{t_2-t_1} = C$, where:

- $P$ is the size of each packet (in bits).
- $t_1$ is the time of arrival of the last bit of the first packet at the destination $D$.
- $t_2$ is the time of arrival of the last bit of the second packet at $D$.
- $C$ is the bottleneck link speed.

We proceed by induction over the first $k$ links of the path, where $k$ ranges from 1 to $n+1$. The case $k = n+1$ covers the entire path from $S$ (source) to $D$ (destination).

## Base Case: $k = 1$

For the sub-path with only the first link (speed $C_1$), the time difference between the arrival of the last bits of the first and second packets at $R_1$ (the first router) is:

$$t_{2,1} - t_{1,1} = \frac{P}{C_1}$$

Since there is only one link, the bottleneck link speed $C_1^* = C_1$. Therefore:

$$\frac{P}{t_{2,1} - t_{1,1}} = C_1^* = C_1$$

This proves the base case.

### Inductive Hypothesis

Assume the claim holds for the first $k$ links, i.e., $\frac{P}{t_{2,k}-t_{1,k}} = C_k^*$, where $C_k^*$ is the bottleneck link speed for the sub-path of $k$ links.

### Inductive Step

Consider the sub-path with the first $k+1$ links:

- **Case 1:** $C_{k+1} < C_k^*$
  If the $(k+1)$-th link is the bottleneck ($C_{k+1} < C_k^*$), then the new bottleneck speed is $C_{k+1}^* = C_{k+1}$. The time difference between the arrival of the last bits of the packets at the next router is:

$$t_{2,(k+1)} - t_{1,(k+1)} = \frac{P}{C_{k+1}} = \frac{P}{C_{k+1}^*}$$

- **Case 2:** $C_{k+1} \geq C_k^*$
  If the $(k+1)$-th link is not the bottleneck ($C_{k+1} \geq C_k^*$), then $C_{k+1}^* = C_k^*$. The time difference between the arrival of the last bits of the packets remains:

$$t_{2,(k+1)} - t_{1,(k+1)} = t_{2,k} - t_{1,k} = \frac{P}{C_k^*} = \frac{P}{C_{k+1}^*}$$

### Conclusion

By proving the inductive step for both cases, the claim holds for all sub-paths with $k$ links, where $k$ ranges from 1 to $n+1$. For $k = n+1$, the entire path satisfies:

$$\frac{P}{t_2 - t_1} = C$$

# Code Analysis

## (a) Creating Datagram Sockets

The following lines in both `sender.c` and `receiver.c` create the Datagram Sockets:

```
int sockfd;
if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
    perror("Socket creation failed");
    exit(1);
}
```

- **Function:** `socket(int domain, int type, int protocol)` **Arguments:**

  - `domain`: Specifies the communication domain; here, `AF_INET` indicates IPv4.
  - `type`: Specifies the communication semantics; `SOCK_DGRAM` indicates a datagram (UDP) socket.
  - `protocol`: Set to 0 to use the default protocol for the domain and type.

## (b) Sending/Writing Data to the Socket (Sender)

The following lines in `sender.c` send data through the socket:

```
if (sendto(sockfd, packet, packet_size, 0, (struct sockaddr
    *)&dest_addr, sizeof(dest_addr)) < 0) {
    perror("Send failed");
    exit(1);
}
```

- **Function:** `sendto(int sockfd, const void *buf, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen)`

  **Arguments:**

  - `sockfd`: The socket descriptor is an integer that uniquely identifies the socket. It's created when a socket is initialized using the socket() function.
  - `buf`: A pointer to the data that is to be sent through the socket. It points to a block of memory (typically an array or buffer) where the data to be transmitted is stored.
  - `len`: The length of the data.
  - `flags`: Flags for transmission (typically set to 0).
  - `dest_addr`: The destination address.

- `addrlen`: This specifies the size (in bytes) of the address structure pointed to by `dest_addr`. It is typically set using the `sizeof()` operator (e.g., `sizeof(struct sockaddr_in)`). This ensures that the system knows the correct amount of memory to read for the address information.

## (c) Reading Data from the Socket (Receiver)

The following lines in `receiver.c` read data from the socket:

```
packet_size = recvfrom(sockfd, buffer, MAX_PACKET_SIZE, 0, (
    struct sockaddr *)&cliaddr, &len);
```

- **Function:** `recvfrom(int sockfd, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)`

**Arguments:**

- `sockfd`: The socket descriptor is an integer that uniquely identifies the socket. It's created when a socket is initialized using the socket() function.
- `buf`: A buffer to store the received data.
- `len`: The maximum length of the buffer.
- `flags`: Flags for reception (typically set to 0).
- `src_addr`: A pointer to the structure that will store the sender's address.
- `addrlen`: This specifies the size (in bytes) of the address structure pointed to by `dest_addr`. It is typically set using the `sizeof()` operator (e.g., `sizeof(struct sockaddr_in)`). This ensures that the system knows the correct amount of memory to read for the address information.

## (d) Measuring the Time of Arrival of Packets (Receiver)

The following lines in `receiver.c` measure the time of arrival:

```
gettimeofday(&t1, NULL);
gettimeofday(&t2, NULL);
```

- **Function:** `gettimeofday(struct timeval *tv, struct timezone *tz)`
**Arguments:**

- `tv`: A pointer to the `timeval` structure that stores the time.
- `tz`: A pointer to a `timezone` structure, usually set to `NULL`.

This function retrieves the current time of day with microsecond precision.

### (e) Ensuring Minimal Gap Between Packets (Sender)

In `sender.c`, the code ensures that packets are sent out one after the other and then the wait delay is added :

```
        // Send the first packet
        if (sendto(sockfd, packet, packet_size, 0, (struct
            sockaddr *)&dest_addr, sizeof(dest_addr)) < 0) {
            perror("Send failed");
            exit(1);
        }
        // Immediately prepare and send the second packet
        snprintf(packet, packet_size, "Pkt %d", i * 2 + 2);
        if (sendto(sockfd, packet, packet_size, 0, (struct
            sockaddr *)&dest_addr, sizeof(dest_addr)) < 0) {
            perror("Send failed");
            exit(1);
        }
        // Wait for the specified spacing before sending the
            next pair
        usleep(spacing_ms * 1000);
```

There is no delay or other operation between sending the two packets, which ensures that the first bit of the second packet is transmitted immediately after the last bit of the first packet.

## Experiment-1: Same Machine with Loopback Restriction

In this experiment, both the sender and receiver programs were run on the same machine with the loopback interface restricted to a maximum throughput of 10 Mbps using the command

`sudo tc qdisc add dev lo root tbf rate 10mbit burst 9kbit latency 50ms`.
The following histogram shows the estimates of the bottleneck link speed for 8k bit packets sent with 100ms delay and 100 such pairs:

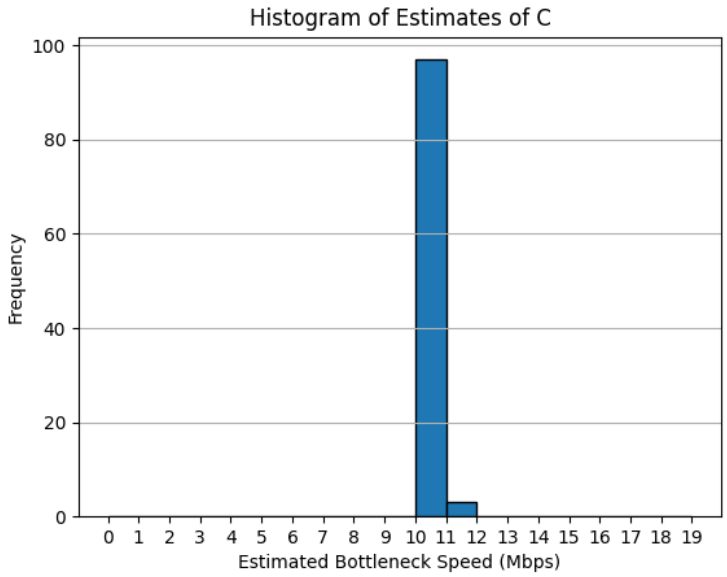


Figure 1: Histogram of estimates of "C" for Experiment-1

It can be clearly observed with the peak at 10Mbps that the bottleneck speed was around that number as the peak indicates that most packets transfer speed was limited by this number which is the bottleneck speed.

# Experiment-2: Different Machines with No Restrictions

The sender and receiver programs were run on different machines at IITB campus without any restrictions on throughput. The traceroute output for each of the three paths is shown below:

```
traceroute to 10.64.74.56 (10.64.74.56), 30 hops max, 60 byte packets
 1  DESKTOP-84B39O5.mshome.net (172.30.208.1)  0.527 ms  0.493 ms  0.416 ms
 2  XiaoQiang (192.168.31.1)  22.204 ms  22.080 ms  22.147 ms
 3  10.6.10.250 (10.6.10.250)  27.402 ms  27.390 ms  27.376 ms
 4  10.250.6.1 (10.250.6.1)  27.352 ms  27.308 ms  27.294 ms
 5  172.16.2.1 (172.16.2.1)  27.200 ms  27.177 ms  27.159 ms
 6  172.16.12.1 (172.16.12.1)  27.039 ms  14.095 ms  14.024 ms
 7  10.64.74.56 (10.64.74.56)  252.460 ms  237.593 ms  237.577 ms
```

Figure 2: Traceroute Output for Path 1

```
traceroute to 10.129.3.5 (10.129.3.5), 30 hops max, 60 byte packets
 1  XiaoQiang (192.168.31.1)  17.407 ms  17.317 ms  17.297 ms
 2  10.6.10.250 (10.6.10.250)  18.243 ms  18.221 ms  18.316 ms
 3  10.250.6.1 (10.250.6.1)  23.305 ms  23.282 ms  23.261 ms
 4  172.16.2.1 (172.16.2.1)  23.239 ms  23.216 ms  23.192 ms
 5  10.250.129.2 (10.250.129.2)  18.451 ms  23.143 ms  23.121 ms
 6  10.129.3.5 (10.129.3.5)  18.382 ms  1.672 ms  1.527 ms
```

Figure 3: Traceroute Output for Path 2

```
traceroute to 10.130.154.33 (10.130.154.33), 30 hops max, 60 byte packets
 1  XiaoQiang (192.168.31.1)  1915.254 ms  1915.187 ms  1915.160 ms
 2  10.6.10.250 (10.6.10.250)  1915.139 ms  1915.118 ms  1915.097 ms
 3  10.250.6.1 (10.250.6.1)  1915.186 ms  1915.163 ms  1915.141 ms
 4  172.16.2.1 (172.16.2.1)  1915.009 ms  1914.977 ms  1915.061 ms
 5  10.250.130.2 (10.250.130.2)  1915.038 ms  1915.013 ms  1914.989 ms
 6  10.130.154.33 (10.130.154.33)  1914.964 ms  1.469 ms  1.397 ms
```

Figure 4: Traceroute Output for Path 3

The following histograms show the estimates of the bottleneck link speed for each path for 1k bit packets sent with 100ms delay and 1000 such pairs:

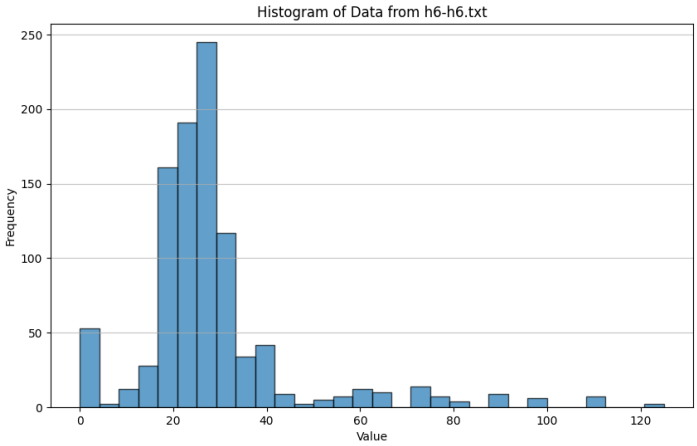


Figure 5: Histogram of estimates of "C" for Path 1

Bottleneck link speed is about 30Mbps

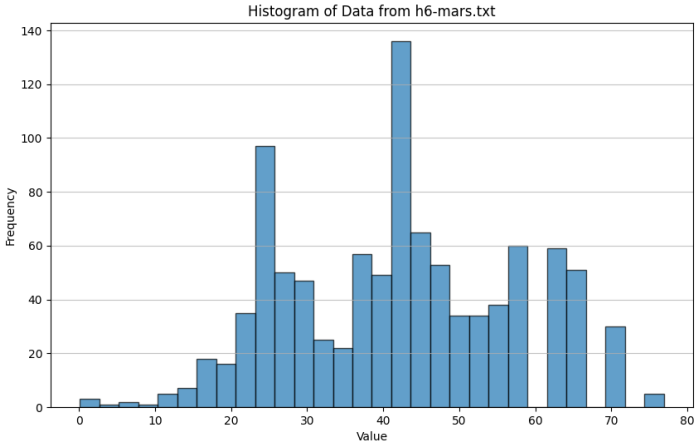


Figure 6: Histogram of estimates of "C" for Path 2

Bottleneck link speed is about 43Mbps

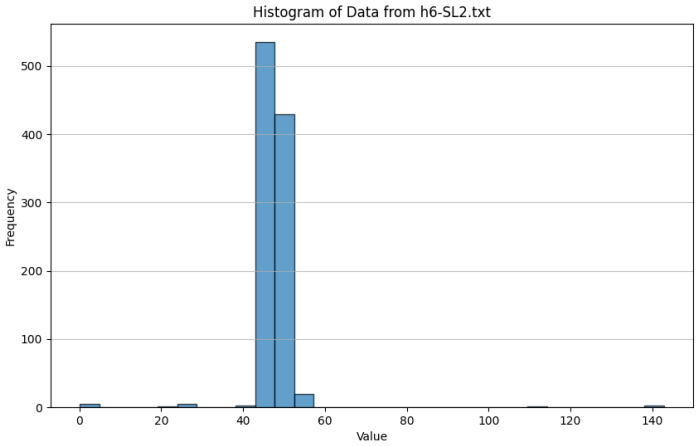


Figure 7: Histogram of estimates of "C" for Path 3

Bottleneck link speed is about 47Mbps