CS378 Lab-5 Report

Saksham Rathi (22B1003)
Department of Computer Science, IIT Bombay

Nandan Manjunath (22B0920)
Department of Computer Science, IIT Bombay

Part 1: (Theory)

Let us prove this by induction.

Case1 : number of links(k) = 1

Packet 1 is ahead and reaches $t_1 = P/C_1$ where C_1 is the link's bandwidth. Packet 2 waits until Packet 1 is completely sent and then is sent. It takes time $P/C_1 + P/C_1$, which is $t_2 = 2 * P/C_1$.

In this case, $P/(t_2 - t_1)$ is C_1, which is the correct bottleneck bandwidth. Therefore, the base case is true.

Case2 : True for number of links(k) then true for links(k+1)

Until it covers k links, let the time they take be t_1 and t_2, and bottleneck bandwidth be C which is $P/(t_2 - t_1)$ (induction assumption). This bandwidth C is the minimum among all links k. Let $k + 1^{th}$ link bandwidth be D.

Case2.1: If D is greater than or equal to C

Now new t_1' will become $t_1 + P/D$ and as $D >= C$, t_1' will be less than t_2 therefore, by the time complete Packet 2 crosses link k, complete Packet 1 will cross $k + 1^{th}$ link also. So, Packet 2 doesn't wait, and the new t_2' will be $t_2 + P/D$. In this case, $P/(t_2' - t_1')$ is C, which matches the bottleneck bandwidth.

Case2.2: Id D is less than C

Then, the bottleneck bandwidth will become D. The new t_1' is $t_1 + P/D$. and as $D < C$, t_1' will be more than t_2 therefore Packet2 has to wait until Packet 1 is completely sent across $k + 1^{th}$ link. This waiting time is $t_1' - t_2$, and then it is sent. The new t_2' is t_2+waiting time $+P/D$ which is $t_2 + t_1' - t_2 + P/D$. In this case, $P/(t_2' - t_1')$ is D, which matches the bottleneck bandwidth.

By induction, we proved that bottleneck bandwidth can be calculated as $P/(t_2 - t_1)$

Part 2: (Implementation)

(a) Here is the code for creating datagram sockets at the sender side:

```c
1 int sockfd;
2 struct sockaddr_in dest_addr;
3 sockfd = socket(AF_INET, SOCK_DGRAM, 0); // UDP socket
4 if (sockfd == -1) {
5     // Socket creation failed
6     perror("Socket creation failed");
7     return -1;
8 }
9 // Set up the destination address structure
10 dest_addr.sin_family = AF_INET;
11 dest_addr.sin_port = htons(8080); // Set an appropriate port
12 dest_addr.sin_addr.s_addr = inet_addr(dest_ip);
```

Listing 1: Sender Datagram Socket Creation

The first line declares an integer variable sockfd which will store the file descriptor for the socket once it is created. The second line declares a variable *dest_addr* of type struct *sockaddr_in*. This structure is used to specify the destination address for the socket communication, including details like the IP address and port number. The third line creates a new socket by calling the socket() function. This function takes three arguments.

- The first argument specifies the address family, which is AF_INET in this case. This indicates that we are using the IPv4 protocol.
- The second argument specifies the type of socket, which is SOCK_DGRAM. This indicates that we are creating a datagram socket, which is used for connectionless communication.
- The third argument is the protocol, which is 0 in this case. This allows the system to choose the appropriate protocol based on the type of socket and the address family.

If the socket creation fails, the function prints an error message using perror() and returns -1. The tenth line sets the address family of the destination to AF_INET, which means it is using the IPv4 protocol. The second last line sets the destination port number for the socket. The htons() function is used to convert the port number (8080 in this case) from host byte order (which varies by machine architecture) to network byte order (which is big-endian). The last line sets the IP address of the destination in the *sin_addr.s_addr* field. The *inet_addr*() function converts a human-readable IPv4 address (passed as *dest_ip*) into a format suitable for storing in the *sin_addr.s_addr* field.

Here is the code for creating datagram sockets at the receiver side:

```c
int sockfd;
struct sockaddr_in server_addr, client_addr;
char buffer[BUFFER_SIZE];
socklen_t addr_len = sizeof(client_addr);
sockfd = socket(AF_INET, SOCK_DGRAM, 0); // UDP socket
if (sockfd == -1) {
    // If the socket creation fails, print an error message and
      return -1
    perror("Socket creation failed");
    fclose(fp);
    return -1;
```

```
11      }
12      // Set up the server address structure
13      server_addr.sin_family = AF_INET;
14      server_addr.sin_port = htons(8080); // Listening on port number
            8080
15      server_addr.sin_addr.s_addr = INADDR_ANY;
16      if (bind(sockfd, (struct sockaddr*)&server_addr, sizeof(
           server_addr)) == -1) {
17          // If the bind fails, print an error message, close the
               socket, close the file, and return -1
18          perror("Bind failed");
19          close(sockfd);
20          fclose(fp);
21          return -1;
22      }
```

Listing 2: Sender Datagram Socket Creation

The third line is the buffer, where we will receive inputs fron the sender. The fourth line is the length of the client address structure. The fifth line is similar to the sender side's code. The tenth line sets the IP address of the server to INADDR_ANY, which means the server will listen on all available network interfaces. The eleventh line binds the socket to the server address. The bind() function associates the socket with the server address so that it can receive packets sent to that address. If the bind fails, the function prints an error message using perror(), closes the socket using close(), closes the file using fclose(), and returns -1. The bind() function takes three arguments.

- The first argument is the socket file descriptor.
- The second argument is a pointer to the server address structure cast to a struct sockaddr pointer.
- The third argument is the size of the server address structure.

(b) Here is the code for sending data to the socket:

```
1   sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr*)&
       dest_addr, sizeof(dest_addr));
```

Listing 3: Sending Data to the Socket