Section 6: Basic Networking and Socket Programming

CS162

July 15, 2019

Contents

1 Vocabulary **2**

2 Warmup **3**

3 Problem **4**

3.1 Designing the Internet 4

3.2 Socket Programming: Implementing an Echo Server 6

1 Vocabulary

- **TCP** - Transmission Control Protocol (TCP) is a common L4 (transport layer) protocol that guarantees reliable in-order delivery. In-order delivery is accomplished through the use of sequence numbers attached to every data packet, and reliable delivery is accomplished through the use of ACKs (acknowledgements).
- **Flow Control** - Flow control is the process of managing the rate of data transmission such that a fast sender doesn't overwhelm a slow receiver. In TCP, flow control is accomplished through the use of a sliding window, where the receiver tells the sender how much space it has left in its receive buffer so that the sender doesn't send too much.
- **RPC** - Remote procedure calls (RPCs) are simply cross-machine procedure calls. These are usually implemented through the use of stubs on the client that abstract away the details of the call. From the client, calling an RPC is no different from calling any other procedure. The stub handles the details behind marshalling the arguments to send over the network, and interpreting the response of the server.
- **socket** - Sockets are an abstraction of a bidirectional network I/O queue. It embodies one side of a communication channel, meaning that two must be required for a communication channel to form. The two ends of the communication channel may be local to the same machine, or they may span across different machines through the Internet. Most functions that operate on file descriptors like read() or write() work on sockets. but certain operations like lseek() do not.

2 Warmup

a) (True/False) IPv4 can support up to 2^{64} different hosts.

32 bits per IP

b) (True/False) Port numbers are in the IP packet.

Transport Layer

c) (True/False) UDP has a built in abstraction for sending packets in an in order fashion.

only TCP. UDP unordered

d) (True/False) TCP is built in order to provide a reliable and ordered byte stream abstraction to networking.

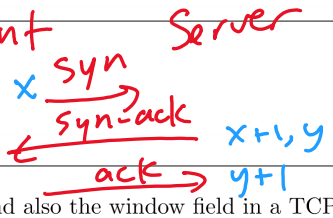
e) (True/False) TCP attempts to solve the congestion control problem by adjusting the sending window when packets are dropped.

f) In TCP, how do we achieve logically ordered packets despite the out of order delivery of the physical reality? What field of the TCP packet is used for this?

Seq no

g) Describe how a client opens a TCP connection with the server. Elaborate on how the sequence number is initially chosen.

3way handshake



h) Describe the semantics of the acknowledgement field and also the window field in a TCP ack.

Window: how much more it is ready to receive

acknowledgement: how much it has received

3 Problem

3.1 Designing the Internet

In class we learned about two fundamental concepts on internet architecture: layering, fate sharing, and the end to end principle.

a) List the 5 layers specified in the TCP/IP model. Layering adds modularity to the internet and allows innovation to happen at all layers largely in parallel. What is the function of each layer?

1. physical – raw bit transport
2. Datalink – packet abstraction, local delivery of packets
3. Network – global delivery of packets
   - narrow waist: only speak IP
4. Transport – E2E communication, stream → packet, packet → stream, TCP, UDP
5. Application – support for apps.

b) When the internet was very young, there was an intense debate between packet switching and circuit switching. Define the two concepts, and discuss the pros and cons of each.

packet switching – data stream → packets & are transported individually

circuit switching – reserve bandwidth, user can send B bits every second

easy to transport multiple things

better quality

c) The fate sharing principle dictates where data should be stored in the internet. In the words of David Clark: The fate-sharing model suggests that it is acceptable to lose the state information associated with an entity if, at the same time, the entity itself is lost. The idea begin fate sharing is that in a distributed system, state should be colocated with the entities that rely on that state. That way the only way to suffer a critical loss of state is if the entity that cares about it also fails, in which case it doesn't matter. What does the fate sharing principle say about the argument of packet switching verses circuit switching?

packet switching is good – flow state in hosts, so if hosts go down, so does this

versus circuit has state in switches & routers

d) The end to end principle is one of the most famed design principles in all of engineering. It argues that functionality should **only** be placed in the network if certain conditions are met. Otherwise, they should be implemented in the end hosts. These conditions are:

- Only If Sufficient: Don't implement a function at the lower levels of the system unless it can be completely implemented at this level.
- Only If Necessary: Don't implement anything in the network that can be implemented correctly by the hosts.
- Only If Useful: If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement.

Take for example the concept of reliability: making all efforts to ensure that a packet sent is not lost or corrupted and is indeed received by the other end. Using each of the three criteria, argue if reliability should be implemented in the network.

i) Only If Sufficient

No, network can send elem, but not confirm it was received
→ end user needs to implement some reliability

ii) Only If Necessary

No. Can do fully in hosts.

iii) Only If Useful

maybe. need solns: general idea: performance improvements could be worth it.

e) An important concept of router design is the separation of the data plane and the control plane. The control plane is like the brains of routing: it makes decisions of where to forward the packets. In constructs a routing table, which is a mapping of packet metadata (usually just the destination IP but sometimes may include source IP, flow ID, etc) to an outgoing port. The data plane takes the routing table, and is responsible for the actual action of looking up a packet and finding its chosen outbound port.

Traditionally, both the control plane and the data plane reside on the routers themselves and each router works in a distributed fashion to calculate their individual routing tables. Recently, there has been a movement to detach the control plane from the routers and instead have one or a set of centralized controllers that act as the control plane for all routers. What may be the pros and cons of doing this?

\SDN software defined networking

distributed control: unflexible & hard to update,
SDN centralizes control
- concerns about security, scalability.

3.2 Socket Programming: Implementing an Echo Server

Alice wants to implement a simple echo server: whenever the client send input to the server, the server simply sends the same input back the client. To do this, Alice writes client-specific code and server-specific code using sockets. However, Alice hasn't been to lecture lately and needs your help to fill out the following code snippets. Assume that the server IP address is servIP, server port is servPort, that messages are 256 bytes or shorter, and that all calls to read and write succeed.

Fill out the client code

```
int main() {
    char *msg = "I should have went to lecture";

    /* Create a socket. */
    int sockfd = ____socket(AF_INET, SOCK_STREAM, 0);____;
    if (sockfd == -1) {
        perror("Failed to create a new socket.\n");
        exit(1);
    }

    /* Specify an address port for the socket */
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = ____AF_INET____;
    server_address.sin_port = ____htons(servPort)____;
    server_address.sin_addr.s_addr = inet_addr(____servIP____);

    /* Connect to the server */
    int status = ____connect(sockfd, &server_address, sizeof(server_address))____;
    if (status == -1) {
        perror("Failed to establish connection.\n");
        exit(1);
    }

    char server_response[256];

    /* Send message to server */
    ____write(sockfd, msg, msg_len)____;
    /* Receive message from server */
    ____read(sockfd, server_response, 256)____;

    printf("Server responded with: %s\n", server_response);
    close(____sockfd____);
}
```

htons convert between host & network

Fill out the server code

```
int main() {
    /* Create a socket */
    int sockfd = ____socket(AF_INET, SOCK_STREAM, 0)____;
    if (sockfd == -1) {
        perror("Failed to create a new socket.\n");
        exit(1);
    }

    /* Specify an address and port for the socket */
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = ____AF_INET____;
    server_address.sin_port = ____htons(serPort)____;
    server_address.sin_addr.s_addr = ____INADDR_ANY____;

    /* Assign address and port to the socket */
    int status = ____bind(sockfd, &server_address, sizeof(server_address))____;
    if (status == -1) {
        perror("Failed to assign address and port to socket\n");
        exit(1);
    }

    /* Listen for new connections */
    status = ____listen(sockfd, 50)____;
    if (status == -1) {
        perror("Failed to listen for new connections\n");
    }

    char client_msg[256];
    int num_read;

    /* Serve forever
    while (1) {
        /* Accept new client connection */
        int client_socket = ____accept(sockfd, NULL, NULL)____;
        if (client_socket < 0) {
            perror("Failed to accept new client connection"\n);
            continue;
        }

        /* Receive message from client */
        ____num_read = read(client_socket, client_msg, 256)____;
        /* Send message to client */
        ____write(client_socket, client_msg, num_read)____;

        close(____client_socket____);
    }
}
```

don't need to bind socket to specific IP

marks socket as passive acceptor of connections

length of queue