1. (a) Consider the password cracker and what you’ve learned about RPC. Is using RPCs for the worker clients a viable way to write the password cracker? If yes, please write the signatures for the stub functions on both the server and client side, along with a short explanation of what each function does. If not, explain in detail why this is not a good idea.

**Solution:** It probably is: The server makes calls to the worker client asking it to execute a function. Better yet, those calls are idempotent already, so there’s no worry about the failure semantics.

**On the server side:**
```c
/**
 * A range of passwords to check is passed
 * to this function along with the hash
 * and the client to send it to
 * string range : range to check
 * string hash : the hash
 * uint16_t client_id : client to use
 * @return string password if the password
 * was found, null if not found
 */
string crack(string range, string hash, uint16_t client_id);
```

**On the client side:**
```c
/**
 * Receives a range and a hash, iterates through the range
 * checking each potential password against the hash
 * string range : the range
 * string hash : the hash
 * string &password : results of cracking
 */
void crack(string range, string hash, string &password);
```

(b) If you were to write a series of RPCs for a heterogenous system (ie, not the same exact system specs), what variables might you consider and how would you solve the problems that arise? (Hint: an example might be the size of an int, int *, etc). Discuss 3 variables of interest, and describe either a workaround for the problem, or explain why it’s okay for this problem to remain unsolved.
2. (a) A client and a server are communicating using UDP. The server sends to the client 3 packets that each contain 2 characters. The contents of the packets are "AB" "CD" and "EF", which are sent by the server in that order. The client tries to receive messages from the server, and when a message is received, it prints the contents to STDOUT without any additional formatting, and blocks while waiting for the next message. If the server only sends these 3 packets, list all possible outputs by the client to STDOUT, assuming the contents of the packets are never corrupted and no packets are duplicated.

**Solution:** There are 16 different possible outputs: ABCDEF, ABFCD, CDABEF, CDEAF, EABCD, EBCDF, ACDF, AEBF, CDAB, CDEF, EFAB, EFCD, AB, CD, EF, No Output.

5 points. Common errors: -1 if didn’t realize that it’s possible for all packets to be lost. -4 if didn’t realize that packets can be lost.

(b) For each of the listed properties, which transport layer protocol (TCP or UDP) would you use if your application needed that property? Give one succinct reason why. Be specific.

- High throughput
- Sending very large messages
- Short-duration interactions with little data
- Sharing bandwidth fairly between competing flows
- One-to-many communication (called “multicast”)

**Note:** We did not cover all of these properties in detail in class. But you should be able to reason about the answers from what you know about TCP and UDP at a fairly high level.

**Solution:** 10 points total, 2 points each.

- High throughput: As long as reliability is not an issue, UDP can give a higher throughput than TCP. This is because no extra packets are sent to initialize the connection or to acknowledge received packets. It is also because TCP’s congestion-control algorithms can decrease the throughput.

- Sending very large messages: TCP, since it handles the dividing of the message into packets, and can reliably receive all of the message’s packets.

- Short-duration interactions with little data: UDP, since TCP comes with overhead such as initializing the connection and extra header data.

- Sharing bandwidth fairly between competing flows: TCP, since it incorporates congestion-control algorithms that kick in when the network becomes congested, slowing down the data rate if there is not enough bandwidth.

- One-to-many communication (called “multicast”): UDP, since it supports multicasting. TCP does not support multicasting, since it is connection-based.
3. When sending a packet across a network, it may encounter delay due to one of several reasons. If the network looks like this:

```
Sender -------------- Router -------------- Receiver
   Link A           Link B
```

We need to know a few attributes of the network to understand the delay. First comes the length and the width of the links: Their **propagation delay** (the amount of time it takes the first bit of your transmission to get from one side to the other), and their **capacity** (the number of bits per second we can send on the link).

The propagation delay is usually some fraction of the speed of light. For a copper wire, it’s about 0.6c. Let’s assume some reasonable values for a cross-country link with a router in the middle, then:

<table>
<thead>
<tr>
<th>Link</th>
<th>One-way propagation delay</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link A</td>
<td>10ms</td>
<td>1Gbit/second</td>
</tr>
<tr>
<td>Link B</td>
<td>20ms</td>
<td>1Gbit/second</td>
</tr>
</tbody>
</table>

(a) If we run the `ping` program on the sender, it will send by default a small, 64 byte packet to the receiver. The receiver will echo the packet back to the sender, and the sender will print out the time it took.

Keeping in mind both components of the delay we talked about above, what is the minimum time that it will take the ping packet to return to the sender, ignoring processing time at the router and at the sender/receiver?

**Solution:** 60ms for propagation delay. 4*(64 bytes)*(8 bits/byte)/(1,073,741,824 bits/s) = 0.00191ms for delay from capacity across both links. Taking both into account, the minimum time is 60.00191ms (6 pts, -3 for ignoring capacity)

(b) From an Andrew machine, please ping the IP address 206.197.119.139. (This is dga’s web server, so be nice to it.) Tell `ping` to send 10 packets:

```
ping -c 10 206.197.119.139
```

When it completes, ping will print out the min/avg/max and deviation of the round-trip times it observed. What values did you observe?

**Solution:** Anything close to these values works: min: 48.327ms avg: 48.900ms max: 49.736ms dev: 0.404ms (6 pts)

(c) Your friend looks at those values and tells you that it’s impossible for Dave’s web server to be located in China. You think about it for a while and agree. Tell us why by calculating the minimum possible round-trip time from Pittsburgh to a server in China. State your assumptions.

**Solution:** The over-land distance from CMU to the nearest point in China is about 9300km (abt 5800 mi). The speed of light is 299,792 km/s (186,282 mi/s). Assuming the connection does not go through the earth, and it obeys the laws of physics, the minimum round trip time to China would be 2*(9300km)/(299,792km/s) = 62ms. This is greater than the round-trip time to Dave’s web server. (6 pts)

(d) Assume that you wanted to send a command to dga’s web server. To do so, you establish a TCP connection to port 80 and send an HTTP request with a particular format. Assuming you don’t have any pre-cached connections, what is the minimum latency between when you run your program locally to when the command is actually executed on dga’s web server? Explain what needed to happen first.
Solution: Before you can send any commands to the server, you have to first establish the TCP connection with the server, which requires one round trip. After this, one one-way trip to the server is required to send the command to the server. This means the minimum latency would be 1.5* (minimum round-trip time to the web server) = 1.5* (48.327ms) = 72.491ms. (6 pts, -2 if assumed the ACK in the handshake needs to be received by the server before you can send the command)

4. In networks, we often characterize devices by what “layer” they operate at. You’ll often hear people refer to, e.g., a “layer 2 switch” when talking about something like an Ethernet switch. These layers usually refer to a model called the OSI model, which looks like this:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>The data the application is sending</td>
</tr>
<tr>
<td>Presentation</td>
<td>Architecture-independent data representation (think what happens in RPC marshalling), encryption, etc.</td>
</tr>
<tr>
<td>Session</td>
<td>Not often used</td>
</tr>
<tr>
<td>Transport</td>
<td>End-to-end connections and reliability, flow control, etc. (TCP goes here)</td>
</tr>
<tr>
<td>Network</td>
<td>End-to-end addressing, forwarding, etc. (IP goes here)</td>
</tr>
<tr>
<td>Data Link</td>
<td>Physical addressing, packetization, etc. (Ethernet goes here)</td>
</tr>
<tr>
<td>Physical</td>
<td>The way the data is sent as an electrical, optical, wireless, etc. signal on a physical link/wire/etc.</td>
</tr>
</tbody>
</table>

Thus, an Ethernet switch operates at the “data link” layer, or layer 2. Ethernet switches examine Ethernet addresses to figure out the destination. An IP router operates at layer 3, the network layer, and examines the IP address (which is kept the same end-to-end from source to destination) to figure out where to send the packet next.

Recall from class that we usually implement layering by packet encapsulation: We put the UDP header and data “inside” an IP packet. We put the IP packet “inside” an ethernet packet to transmit it on the local Ethernet.

You send a packet from one of the Andrew Linux machines to Dave’s web server (IP address 206.197.119.139). Let’s figure out what this packet looks like.

(a) Use the `ifconfig` command (/sbin/ifconfig) to find out the IP address and Ethernet MAC address of the Andrew Linux machine you’re using. The Linux ifconfig output calls the IP address the “inet” address and the MAC address the “HWaddr”. (The “-a” flag to ifconfig will show all interfaces, but you’ll have to figure out which one is the main interface. Which one has the most received - RX - and sent - TX - packets? Ignore lo, that’s local.). What are the addresses?

Solution: Any reasonable answers accepted. (5 points)

(b) Remember that your computer isn’t directly connected to Dave’s Ethernet—the packet has to go from CMU to Dave’s ISP. Therefore, a packet from you to Dave will have to go through multiple layer 3 “hops”, each of which forwards the packet to the next hop. Use the `traceroute` utility (/bin/traceroute) to discover the IP-layer hops that your packet will take on the way to Dave’s web server. Reproduce the output below.

Solution: Any reasonable answers accepted. (5 points)
(c) From the above discussion, you know that the destination IP address in the packet has to be that of the web server. So how does the packet get from your andrew machine to the next “hop”?

**Solution:** The sending machine sets the ethernet-layer destination to the MAC address of the first-hop router. The IP destination remains that of the final destination (dga’s web server). Useful but not required part of the answer: The machine looks up this next-hop MAC address in its ARP table. (5 points.)

(d) Use the `route` command (`/sbin/route`) to show the contents of the local routing table on the andrew machine. This table tells the machine where to send packets in order to reach a particular destination. The “default” route is the one used, on these machines, to reach the rest of the Internet. HINT: Use `route -n` to show the IP address of the default gateway instead of a chopped-up hostname, but note that with -n, the default route is called "0.0.0.0". Include below the line of output matching the default route. Does the gateway address match what you saw from traceroute?

**Solution:** Any reasonable solutions accepted that fall into the same subnet as the addresses reported in part (a). (5 points). In many cases, the addresses did not match that from traceroute. As explained in class, this is due to a peculiarity of the way traceroute works. The routers are allowed to send packets back from any valid IP address belonging to the router. In many cases, it was using an address different from the one in the next hop.

(e) Use the `arp` command (`/sbin/arp -na`) to show the contents of the local table that maps IP addresses to the Ethernet address that owns that address. What is the Ethernet address of the next hop for your packet?

**Solution:** Any MAC address accepted. But had to be an Ethernet MAC address. Common mistakes: Using a DNS name or an IP address. (5 points)

(f) You send a single UDP packet from port 8888 on that host to port 9999 on Dave’s web server (IP 206.197.119.139). Draw the packet headers for the Ethernet, IP, and UDP layers in the correct order, showing where the application data goes. You don’t have to show all of the fields for the headers, but do show the source and destination addresses (for Ethernet and IP) and the ports (for UDP). You’ll probably want to google to see the packet formats, and note that Ethernet is tricky.

**Solution:** Common mistakes: Using IP addresses instead of Ethernet MAC addresses in the Ethernet header. Not listing the actual addresses. Not showing the Ethernet trailing checksum at the end of the packet.

5. Consider the following code snippet.

```c
int check_next_ten_passwords(char *str) {
    char *tmp = malloc(strlen(str) + 1);
    strcpy(tmp, str);
    for (int i = 0; i < 10; i++) {
        if (!strcmp(crypt(tmp, "aa"), PASSWORD_STRING)) {
            printf("I FOUND IT! %s\n", tmp);
            return 1;
        }
        tmp[0]++;
    }
    return 0;

```
(a) This code has a bug that does not affect its ability to check the next 10 passwords. What is it?

**Solution:** The memory allocated to tmp is never freed. Every time the function is called, it will leak a small amount of memory.

(b) If you used this code by creating a thread that called the function to test a password and then destroyed the thread, the bug would affect you. What would happen if you instead `fork()`'d a new process to test the password, and then had the process exit? Explain briefly (1-2 sentences) the difference.

**Solution:** Using fork, the memory leak would not be apparent: The resources would be allocated in the child and freed when it exited, despite the lack of an explicit call to `free()`. In the threaded version, the resources would remain allocated within the process, even if the thread exited.