CSE 333 Section 8 - Client-Side Networking

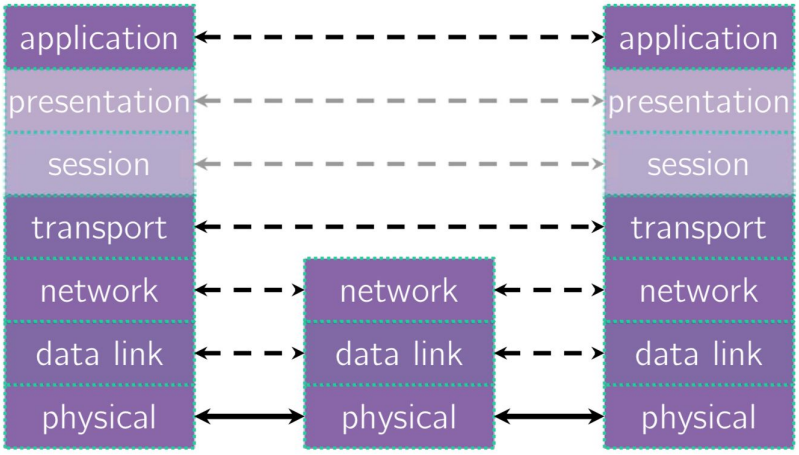
Welcome back to section! We're glad that you're here :)

Networking Quick Review

Exercise 1

a) What are the following protocols used for? (Bonus: In what *layer* of the networking stack is it found?)

- DNS
- IP
- TCP
- UDP
- HTTP



b) Why would you want to use TCP over UDP?

c) Why would you want to use UDP over TCP?

Exercise 2

For each of the following, identify the layer in the network stack that performs the described operation. Be sure you identify the layer, not the protocol.

a) Use WiFi or Bluetooth to transmit data to other hosts.

b) Use media access control (MAC) to figure out when and where to send the packet.

c) Forward a packet from the local wired or wireless network to a different local network if its destination address is not on the same local network.

d) If two packets that make up a message arrive out of order, rearrange them into the correct order before they are transmitted to the process reading the data.

e) Transmit an Ethernet packet on the local network from one host machine's NIC interface address to another's.

f) Resolve what IP address `www.youtube.com` is pointing to.

Step-by-step Client-Side Networking

Step 1. Figure out what IP address and port to talk to. (`getaddrinfo()`)

```
// returns 0 on success, negative number on failure
int getaddrinfo(const char *hostname,    // hostname to lookup
           const char *servname,         // service name
           const struct addrinfo *hints, // desired output
      (optional)
           struct addrinfo **res);       // results structure

struct addrinfo {
     int ai_flags;              // additional flags
     int ai_family;             // AF_INET, AF_INET6, AF_UNSPEC
     int ai_socktype;                // SOCK_STREAM, SOCK_DGRAM, 0
     int ai_protocol;                // IPPROTO_TCP, IPPROTO_UDP, 0
     size_t ai_addrlen;         // length of socket addr in bytes
     struct sockaddr* ai_addr;  // pointer to socket addr
     char* ai_canonname;        // canonical name
     struct addrinfo* ai_next;  // can have linked list of records
}
```

Step 2. Create a socket. (`socket()`)

```
// returns file descriptor on success, -1 on failure (errno set)
int socket(int domain,          // AF_INET, AF_INET6, etc.
           int type,            // SOCK_STREAM, SOCK_DGRAM, etc.
           int protocol);       // usually 0
```

Step 3. Connect to the server. (`connect()`)

```
// returns 0 on success, -1 on failure (errno set)
int connect(int sockfd,                // fd from step 2
            struct sockaddr *serv_addr, // socket addr from step 1
            socklen_t addrlen);        // size of serv_addr
```

Step 4. Transfer data through the socket. (`read()` and `write()`)

```
// returns amount read, 0 for EOF, -1 on failure (errno set)
ssize_t read(int fd, void *buf, size_t count);

// returns amount written, -1 on failure (errno set)
ssize_t write(int fd, void *buf, size_t count);
```

These are the same POSIX calls used for files, so remember to deal with partial reads/writes!

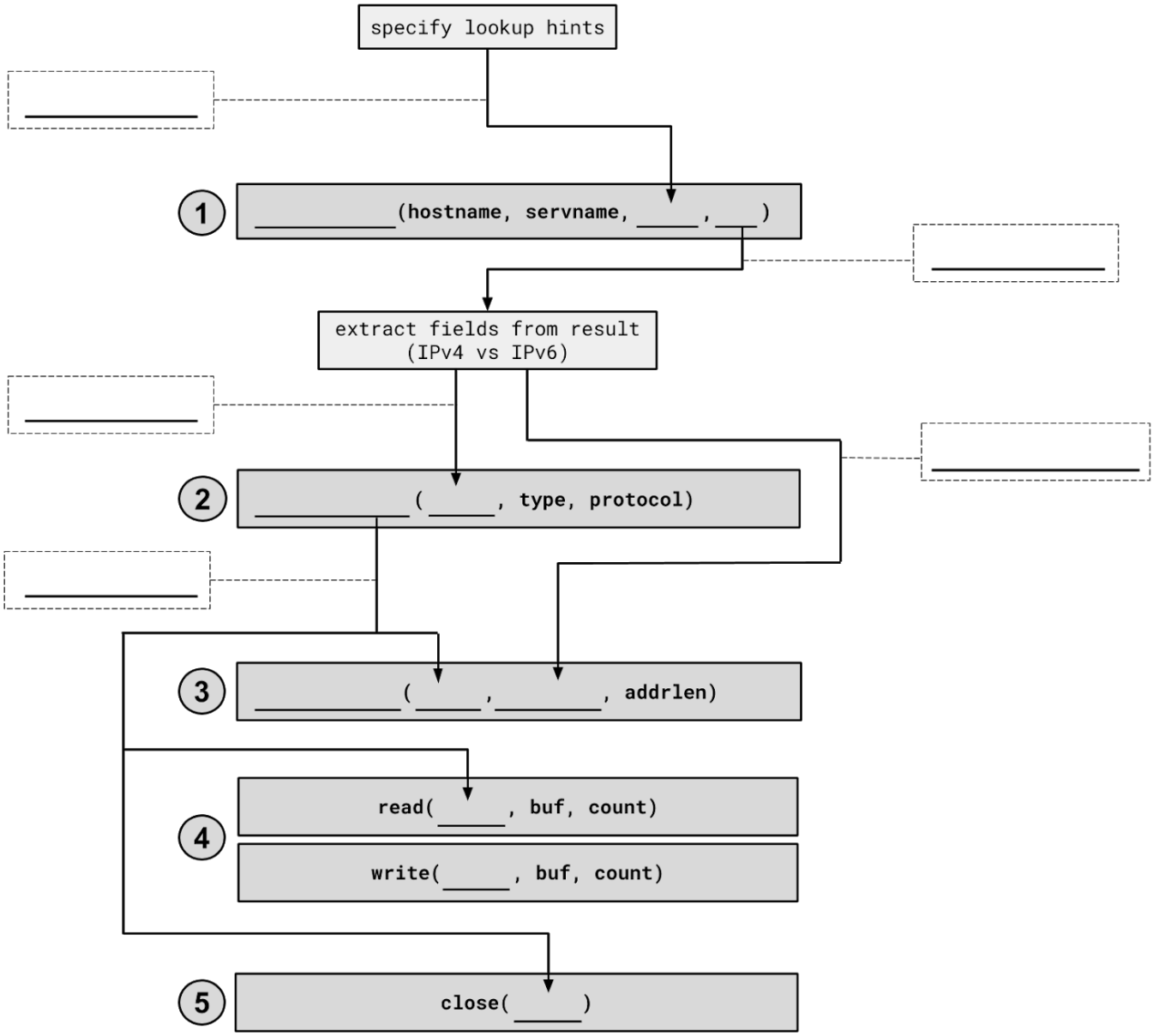
Step 5. Close the socket when done. (`close()`)

```
// returns 0 for success, -1 on failure (errno set)
```

```
int close(int fd);
```

Exercise 3

Fitting the Pieces Together. The following diagram depicts the basic skeleton of a C++ program for client-side networking, with arrows representing the flow of data between them. Fill in the names of the functions being called, and the arguments being passed. Then, for each arrow in the diagram, fill in the type and/or data that it represents.



Exercise 15 Skeleton

Write a C++ program that accepts three command line arguments:

- the hostname of a server
- the port number of that server

- the name of a local file

The program should connect (via TCP) to the server on the supplied hostname and port. Once connected, the program should read the bytes from the local file, and it should write those bytes over the TCP connection. Once all of the bytes have been written, the program should close the TCP connection and exit. Make sure to appropriately handle any errors that can occur as part of each of these steps!

```
int main (int argc, char **argv) {

     // Get the hostname and port from the arguments.


     // Get an appropriate sockaddr structure.


     // Create a socket.


     // Connect to the remote host.


     // Iterate through the file and send to server.


     // Clean up.


     return EXIT_SUCCESS;
}
```