**CSE 333 Section 8 - Client-Side Networking**

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

***Computer Networking Review***

Exercise 1

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

| DNSIPTCPUDPHTTP | Reliable transport protocol on top of IP.Translating between IP addresses and host names.Sending websites and data over the Internet.Unreliable transport protocol on top of IP.Routing packets across the Internet. |
| --- | --- |

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

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

| struct sockaddr { // size: really small sa\_family\_t sa\_family; // Address family (AF\_\* constants) . . .}struct sockaddr\_in { // size: small sa\_family\_t sin\_family; // Address family: AF\_INET in\_port\_t sin\_port; // Port in network byte order struct in\_addr sin\_addr; // IPv4 address . . .}struct sockaddr\_in6 { // size: quite large sa\_family\_t sin6\_family; // Address family: AF\_INET6 in\_port\_t sin6\_port; // Port number struct in6\_addr sin6\_addr; // IPv6 address . . .}struct sockaddr\_storage { // size: really large sa\_family\_t ss\_family; // Address family (AF\_\* constants) . . .} |
| --- |

***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 failureint getaddrinfo(const char \*hostname, // hostname to lookupconst char \*servname, // service nameconst struct addrinfo \*hints, // desired output (optional)struct addrinfo \*\*res); // results structurestruct 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 2
Fitting the Pieces Together. The following diagram depicts the basic skeleton of a C/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.

