

cmsc 417 programming assignment #4

(updated)

Goal: Long-distance links tunneled through TCP.

1 Deadline

DUE: Friday, April 28. The late deadline will be Monday, April 31; you have only one late deadline to play with.

2 Context

We're going to tunnel 417-net packets through TCP. We'll use TCP to get long-distance links that would support routing protocols (PA5); there's no sense building a routing protocol on top of the current multicast scheme.

Your implementation should extend previous assignments (support the print neighbor table and sendmsg commands, limit message transmission rate) with a new command "connect" that takes an IP address and port, then treats the connection as a neighbor.

3 (stop, collaborate, and) Listen

One end of the tunnel will listen; allow the kernel to choose a port, then print whatever it has chosen for you in the call to bind. (socket, bind, listen, (select), accept).

Print the port the kernel has chosen for you using:

```
printf("listening on port: %d\n", ntohs(sin.sin_port));
```

The listening socket can accept in a non-blocking mode if the listening socket is passed to select() as if it were for reading, then, when ready, invokes accept (instead of read). You'll never read() off the listening socket.

After you've accepted a TCP socket, send periodic hello messages, as in PA2, across the newly-accepted TCP connection as well.

4 Connect

The connect command should be parsed as follows:

```
struct sockaddr_in sin;
sscanf(command, "connect %d.%d.%d.%d:%d", &a1, &a2, &a3, &a4, &port);
sin.sin_addr.s_addr = htonl((a1<<24)+(a2<<16)+(a3<<8)+a4);
sin.sin_port = htons(port);
```

The connect system call can also be made non-blocking; it's tricky, but very fun to setsockopt O_NONBLOCK, then finish the connection setup when, you guessed it, select returns that the socket is ready for reading. Good stuff, but extra fun for you with no credit at all.

After you've connected, send periodic hello messages, as in PA2, across the TCP connection as well. Upon receiving the first hello message from the TCP connection, print the source address of the hello message as follows:

```
printf("new neighbor: %u\n", ntohl(received_packet.header.source_address));
```

5 Tearing down connections

If you read zero bytes from the TCP connection, tear down the adjacency: remove the neighbor from the list of neighbors, send no more hello packets. If the periodic write returns -1, (especially if errno is EBADF, but reasonable for all other errors too), remove the neighbor, close the socket, and send no more hello packets. If no messages are received in two minutes, close the connection, remove the neighbor, and send no more hello packets.

You may implement a “close” command if you like; doesn't matter to me.

6 The neighbor table

Expect to update your neighbor table with one field: a TCP socket number used to reach that neighbor, if necessary. Although socket zero is valid and socket -1 is not, you may choose either to be the “invalid” or “not a tcp neighbor” indication because socket zero is deterministically stdin and you'll not write to stdin.

To construct the rdfs mask for select, include all that you currently do, then find additional sockets by iterating through the neighbor table. It's not ideal—best is to have a separate list pairing socket descriptors with function pointers (callbacks) to invoke when the socket is ready for reading—but works.

7 sendmsg

Sendmsg must work over the new TCP adjacency.

Please run sendmsg through the same rate-limiting as for other neighbors even though it is less important to do so over TCP.

Ensure that the destination address in sendmsg packets is set correctly. If so, the only needed parameter to the callback is the packet to be sent—the sending callback can find the address of the neighbor by consulting the neighbor table.

8 Hints

- Schedule a hello broadcast event for each new TCP connection.
- Treat TCP adjacencies as neighbors—sendmsg must work!
- Read packets as (1) header, then (2) payload.
- Check that the packet header matches the correct version. That is, use the version (and protocol) fields to make sure that frames received are okay.
- You should be able to start testing by running two instances on the same host and invoking “connect 127.0.0.1:x @2” where x is the port given to the other side by the kernel in bind.
- Finish testing by running an instance on a home machine, or test against a crippled version that won't speak multicast or udp.

- Note `timercmp` and `timersub` macros in `sys/time.h` (or somewhere close): these allow simple comparison or subtraction of two struct `timevals`.
- Disable nagle's algorithm (why?) using:

```
int one=1;
setsockopt(s, IPPROTO_TCP, TCP_NODELAY, &one, sizeof(one));
```

9 Objectives

At the end of this assignment, you should understand:

- tcp sockets.
- tcp server behavior.
- everything needed to build a routing protocol over a peer to peer network.

10 Caveats

TCP is *bytestream-oriented*, which means that `read()` from a TCP socket may return fewer bytes than are part of the packet written from the other side. I'll use 417-net packets that are shorter than 200 bytes so that it is very unlikely that a partial packet will be available for reading. Ideal, instead, is to buffer, at the application level, any partial 417-net packets read from the TCP socket. But for this class, assume that `read`—*called with a header-sized or payload-length-sized buffer, rather than a worst-case large message size as the third parameter*—will return exactly as many bytes as you ask for.

11 Notes

If you discover a neighbor two ways, you may choose either method to reach it. However, if you do not close the TCP connection from the valid neighbor, your implementation should be prepared to handle the incoming packets from that neighbor.

You may still discard or otherwise ignore received packets.