CMSC 417 Programming Assignment #4

Due November 12, 1999 (**5:00 PM**)

Introduction

You will add packet forwarding, ICMP Echo (ping), and traceroute support to your IPv6 system from project #3. This project will also introduce the use of a garbler function that will allow you to introduce link-level errors in your packets to verify that higher levels are able to recover from these errors.

Packet Forwarding

This project will add packet forwarding to your network project. When a packet arrives at a node, you should check the destination address and see if it is intended for this host. If not, your router should consult its routing table to identify the next hop for the packet. If an appropriate next hop is found, your router should decrement the hop count field by one. If the hop count field is one or more, it should forward the packet to the next hop router (using the garb_sendto call). If the hop count is zero, the router sends an ICMP TIME_EXCEEDED message back to the source host and does not forward the packet (The code for this message type is zero). If there is no routing table entry for the specified host, you should send an ICMP DESTINATION_UNREACHABLE message to the source node (The code for this message is 0).

ICMP Support

In this project you will add support for ICMP echo packets. The format of an ICMP packet is:

Field	Bits	Description
Type	8	ICMP Request
Code	8	Generally zero
Checksum	16	ICMP Checksum
Reserved	32	Must be Zero
Address	128	IPv6 Address

ICMPv6 Header Format

The type field can be one of four values (in this project):

Value	Description
1	Destination Unreachable
3	Time Exceeded
128	Echo Request
129	Echo Response

ICMPv6 Types

The checksum should is computed as the one's complement of the one's complement sum of the data in the ICMP packet (including the ICMP header and the IPv6 header). When computing the checksum, the contents of the checksum field should be zero.

Interface

To allow you to write code that uses your new interface, you will construct an implementation of the IPv6 socket layer. The header file for this routine is in IPv6socket.h. The interface routines for the socket layer are:

```
int IPv6init(int argc, char **argv)
```

This function is called once to initialize the networking layer. This function should create any of the long-lived threads in your networking code (i.e., the timer and routing threads).

```
int IPv6socket(int family)
```

This function creates an IPv6 socket. The family parameter indicates which protocol family this socket is associated with. The possible values are IPV6_PROTO_ICMP, IPV6_PROTO_UDP, and IPV6_PROTO_TCP.

```
int IPv6sendto(int sock, const char *msg, int len, unsigned char *to)
```

Send a message to a remote host using the ipV6 address specified in to. The message (msg) is len bytes long.

```
int IPv6recvfrom(int sock, char *msg, int len, unsigned char *from)
```

Receive a message on an IPv6 socket. If the pending message is longer than len bytes, it should be truncated to len bytes and the rest of the message discarded. If a sock is of type IPV6_PROTO_ICMP, all ICMP messages received by the host should be forwarded onto this socket. However, ECHO packets should be responded to internally to your network layer.

Ping and TraceRoute

In this project you will use your new capabilities to write a ping and traceroute utility.

Both ping and traceroute are ICMP utilities that use the ICMP protocol’s hello packets to learn about that status of the network. The ping utility sends an ICMP ECHO packet to a destination and waits for an ICMP ECHO response packet to come back. The traceroute command also uses a HELO packet, but it uses the IPv6 hop limit field to discover the path that a packet takes through the network. The first packet sent by a traceroute command should have a hop limit value of 1. This will cause the first hop router to drop the packet and send back an ICMP TIME EXCEEDED packet (which you can use to get the address of the first hop router). From there, you should increase the number of hops until the packet reaches its destination.

Your overall main program for this project should be a command line tool that reads a line of input in the form <command> <ipv6addr> and runs the ping utility if <command> is “ping” and traceroute if <command> is “traceroute.”

Garbler

The final component of this project is to use the garbler routine to induce errors, and cause drops and duplicates in your packets. Rather than calling sendto directly to send an IPv4 udp packet, you will use the routine garb_sendto to cause the packet to be sent. The garbler reads a configuration file that determines the probability of certain types of failures in the network. See the file ipv6-garb.init for a list of the parameters. You can use the routine garb_print_stats to print out the statistics about the type of errors the garbler has introduced into your projects.

Implementation Requirements

You should submit a tar file that contains the source code for your project, and a Makefile. You should also submit a script file for **each** of the nodes using the configuration files supplied.