

## CMSC 417 Programming Assignment #5

Due December 10, 2001 (10:00 AM)

### Introduction

You will add a reliable transport protocol to your project in this assignment.

### Transport Protocol

The transport protocol needs to provide reliable in-order delivery of bytes between two threads. Your transport protocol should use a three-way handshake to establish communication, and support graceful shutdown of the socket (with appropriate timeouts to prevent lingering dead connections).

It will need to handle drops, corruption, and duplication of packets.

You are free to design the protocol format for this application. The only requirement is that the Ipv6 next header field must be of type PTCP (106). You will probably want to have a separate thread for each active transport level connection. Also, you will need to use your timer thread to schedule timeouts for re-transmission of dropped packets. Also, you should allow only a finite number of bytes to be buffered on the sender side. This limit should be 8KB by default, but can be changed by the IPv6setSendLimit call (see interface section).

### Interface

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

```
int IPv6listen(int sock, port)
```

This call will associate a socket with a port.

```
int IPv6connect(int sock, char *to, int port)
```

This call will connect a socket to a remote host/port pair. The to field is an Ipv6 address (array of octets form), and the port parameter is a port number.

```
int IPv6accept(int sock)
```

This call will block until another thread tries to connect on this socket. When another process connects, a new socket is created (and returned by Ipv6accept).

```
int IPv6send(int sock, const char *msg, int len)
```

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

```
int IPv6recv (int sock, char *msg, int len)
```

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.

```
int IPv6close(int sock)
```

Close down the passed socket. If the socket is not open, return a negative value as the error code.

```
int IPv6setSendLimit(int sock, int size)
```

Set the maximum send buffer size to size bytes. This is only valid for a TCP socket. It should return 0 on success or -1 if the value is invalid (negative or greater than 256KB) or if the socket is not an active TCP socket.

### Application

In this project you will use the application code we supply to with your network layer. Our code is a simple file transfer program, plus an implementation of ping and traceroute.

## **Garbler**

Like project #4, you should use the garbler rather than calling sendto directly to send an IPv4 udp packet. For this project, the garbler will be configured to drop, corrupt, and duplicate packets.

## **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. Detailed submission information will be posted on the class web page.

Your entire project should build into a library called libIPv6.a. Your Makefile should use the “ar” command to bundle all of your .o files into the file libIPv6.a. There is **no** main program to be submitted with this project, it is simply a library of functions that our test application will call. You should not turn in the ftp application we supply.