CSS 503
Final Project: Inter-Segment/Group UDP Broadcast
Professor: Munehiro Fukuda
Due date: see the syllabus

1. Purpose
The final project will implement a UDP multicast relay program that facilitates a user-level UDP multicast environment over multiple network segments and/or multicast groups. UDP multicast is available only within a local network segment or a single multicast group by default, beyond which we need to design a mechanism that delays a UDP multicast message to a different segment or group. One idea is to run such a facilitator at each network segment or group and connect any two of different facilitators with a TCP link that delays a UDP message to the other end. To implement this UDP relay, we will use TCP sockets, UDP multicast sockets, and multithreading.

2. Socket Class
Use ~css503/programming/prog4/Socket.h and Socket.cpp to establish a TCP communication link.

Methods	Descriptions
`Socket(int port)`	A default constructor that creates a new Socket object with a given port.
`~Socket( )`	Closes the socket descriptor maintain in this Socket object.
`int getClientSocket(char[] address)`	Establishes a TCP connection to a given address at port and returns its socket descriptor. After this call, you can use write(sd, buf, size) and read(sd, buf, size) where sd is the return value from getClientSocket( ).
`int getClientSocket( char[] address, int sndbufsize, bool nodelay )`	Establishes a TCP connection to a given address at port, adjusts this socket's send buffer with sndbufsize, disables Naggle's algorithm if nodelay = false, and returns its socket descriptor. In general, you don't have to use this method.
`int getServerSocket( )`	Accepts a TCP connection request from a client and returns its socket descriptor. After this call, you can use write(sd, buf, size) and read(sd, buf, size) where sd is the return value from getServerSocket( ).
`int getServerSocket(int rcvbufsize, bool nodelay)`	Accepts a TCP connection request from a client, adjusts this socket's receive buffer with rcvbufsize, disables Naggle's algorithm if nodelay = false, and returns its socket descriptor. In general, you don't have to use this method.

3. UdpMulticast Class
Use ~css503/programming/prog4/UdpMulticast.h and UdpMulticast.cpp to multicast a UDP message.

Methods	Descriptions
`UdpSocket(char group[], int port)`	A default constructor that opens a UDP datagram socket and has it participate in a given group address, (e.g. "238.255.255.255") at a given port.
`~UdpSocket( )`	A default dstructor that closes the UDP socket.
`int getClientSocket( )`	Uses this socket descriptor as a client multicast socket. It returns a socket descriptor but you do not directly use the descriptor.
`int getServerSocket( )`	Uses this socket descriptor as a server multicast socket. It returns a socket descriptor but you do not directly use the descriptor.

| `bool multicast(char buf[])` | Has a client multicast a give message in buf[] to all servers belonging to the same group@port. It returns true upon a success, otherwise false. |
| `recv(char buf[], int size)` | Has a server receive a multicast message into a given buf[] with the size. It returns true upon a success, otherwise false. |

4. Inter-Segment/Group UDP Broadcast

The key idea is to choose a representative computing node in each different segment (or group), to run a UDP relay program there, and to have it relay local UDP multicast messages through TCP links to remote segments as well as have it multicast to the local segment remote UDP messages that came in through TCP links. See the figure below:
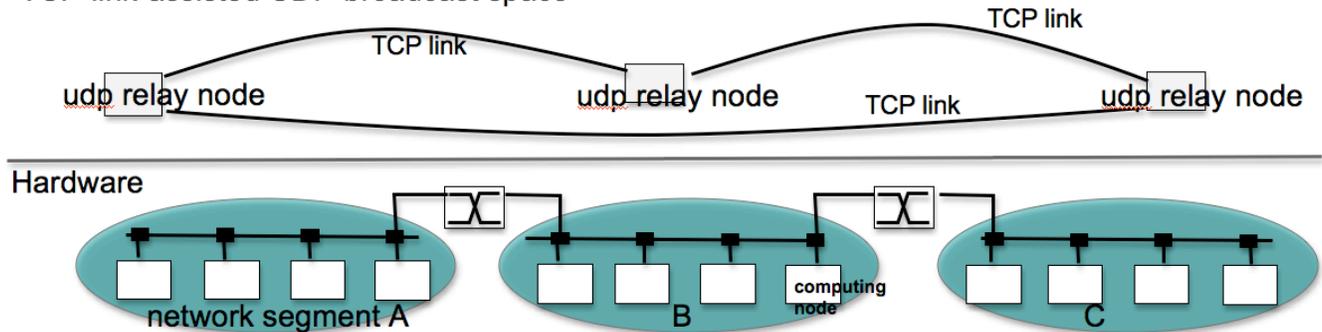
The problem is a cyclic message relay that may even cause a domino effect and thus flood network segments with copies of the same UDP message. To prevent this problem, we will add a UDP relay header to the top of the original UDP message. This header format is described below:

1st byte	2nd byte	3rd byte	4th byte
−32	−31	−30	hop (e.g. 3)
1st UDP Relay's IP address			
2nd UDP Relay's IP address			
3rd UDP Relay's IP address			
The actual UDP message			

The UDP relay header manipulation takes place in the following two cases:

(1) Every time a UDP relay program forwards a local UDP message to a remote segment through TCP, it adds its own 4-byte IP address to the tail of this UDP relay header as incrementing its hop field (4th byte).

(2) Every time a UDP relay program receives a UDP message from a remote segment through TCP, it will scan the UDP relay header to examine if it includes the IP address of where this relay program is running. If so, the same message has been once delayed by this relay program and thus should be simply discarded.

5. UdpRelay Class

This is the class that you will design to facilitate inter-segment/group UDP multicast. The class is instantiated from the following driver program (located as ~css503/programming/prog4/driver.cpp).

```
#include "UdpRelay.h"
#include <iostream>   // cerr

using namespace std;

int main( int argc, char *argv[] ) {
  // verify the argument.
  if ( argc != 2 ) {
    cerr << "usage: bcast groupIp:groupPort" << endl;
    return -1;
  }

  UdpRelay udprelay( argv[1] );

  return 0;
}
```

The UdpRely constructor receives a string, (i.e., argv[1] in the above driver program) that includes groupIp:groupPort, (e.g., "238.255.255.255:12345"). UdpRely manipulates two ports:
  (1) Group Port: initialized with argv[1] and used for UDP multicast
  (2) TCP Port: hard-coded with the last 5 digits of your student ID and used for TCp connection

The UdpRelay instantiates the following three threads:
  (1) commandThread:
      Waits for a user to type the following commands from the keyboard input, (i.e., cin). The available commands include:

Commands	Descriptions
`add remoteIp:remoteTcpPort`	Adds a TCP connection to a remote network segment or group whose representative node's IP address and TCP port are remoteIP and remoteTcpPort. It then instantiates relayOutThread that keeps reading a UDP multicast message through this TCP connection from remoteIp and multicasting the message to the local group, (i.e., groupIp).
`delete remoteIp`	Deletes the TCP connection to a remote network segment or group whose representative node's IP address is remoteIp. It also terminates the corresponding relayOutThread.
`Show`	Shows all TCP connections to remote network segments or groups.

`help`	Summarizes available commands like:
	
	`UdpRelay.commandThread: accepts...`
	`    add remoteIp:remoteTcpPort`
	`    delete remoteIp`
	`    show`
	`    help`
	`    quit`

`Quit`	Terminates this UdpRelay program. (Optional: you don't have to implement it. In fact, the key answer hasn't implemented it.)

(2) acceptThread:
Creates a Socket object with a given TCP port, (namely the last 5 digits of your student ID); and thereafter keeps accepting a TCP connection request from a remote UdpRely; checks if another TCP connection has been already established to that remote node; if so, deletes the former connection; and starts relayOutThread that keeps reading a UDP multicast message relayed through this TCP connection from the remote node and multicasting it to the local group, (i.e., groupIp).

(3) relayInThread:
Creates a UdpMulticast object with a given groupIp and groupPort, and thereafter keeps catching a local UDP multicast message. Every time relayInThread receives a UDP multicast message, it scans the multicast header to examine if it includes the local UdpRelay's IP address. If so, it simply discards this message. Otherwise relayInThread forward this message through TCP connections to all the remote network segments/groups.

As described above, every time commandThread and acceptThread establish a new TCP connection to a remote UdpRely program, they launch a new **relayOutThread**:

Keeps reading a UDP multicast message relayed through this TCP connection from the remote node; scans the multicast header to examine if it includes the local UdpRelay's IP address; if so, simply discards this message, otherwise multicasts it to the local group, (i.e., groupIp).

The following figure illustrates the structure of UdpRelay:
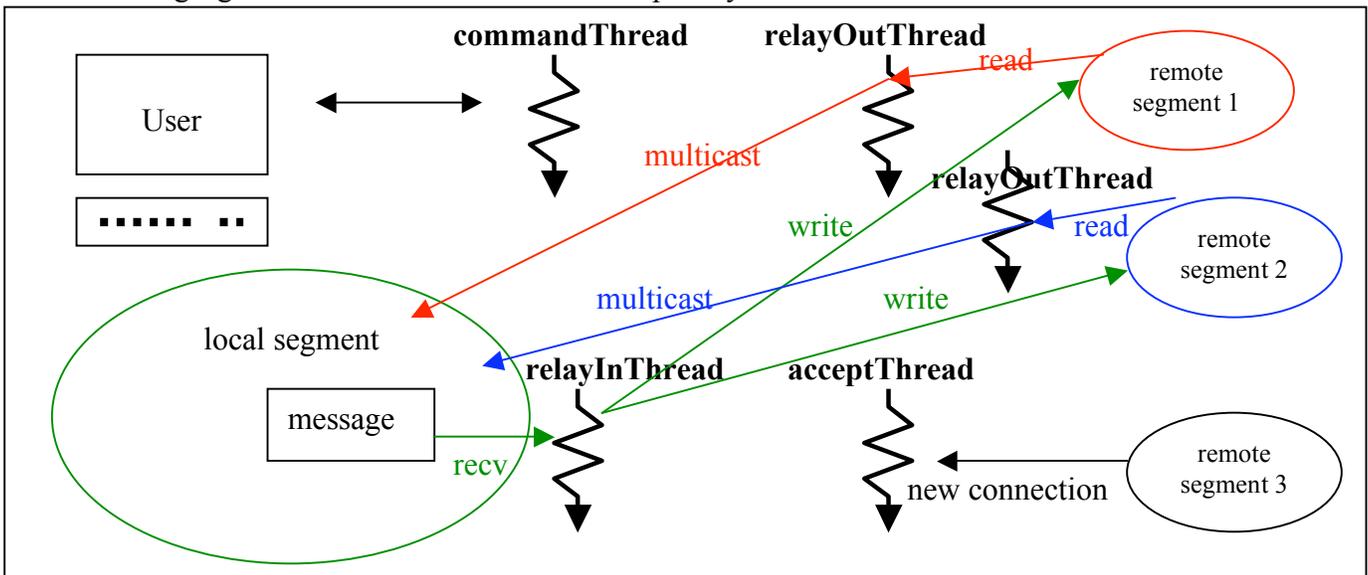
6. Statement of Work

Follow the procedures shown below:
- (1) Implement UdpRelay.h and UdpRelay.cpp.
- (2) Compile it with driver.cpp:

```
g++ UdpRelay.cpp UdpMulticast.cpp Socket.cpp driver.cpp -o UdpRelay
```

- (3) Test your UdpRelay using 6 computing nodes:

Network Segments	Computing Nodes (uw1-320-01 ~ 23)	Programs and commands
237.255.255.255	Node 1	UdpRelay 237.255.255.255:yourUdpPort
	Node 2	java BroadcastServer 237.255.255.255:yourUdpPort
238.255.255.255	Node 3	UdpRelay 238.255.255.255:yourPort
	Node 4	java BroadcastServer 238.255.255.255:yourUdpPort
239.255.255.255	Node 5	UdpRelay 239.255.255.255:yourPort add Node1:yourPort add Node3:yourPort
	Node 6	java BroadcastServer 239.255.255.255:yourUdpPort
	Node 7	java BroadcastClient 239.255.255.255:yourUdpPort Hello!

Note:
- (a) BroadcastServer.class and BroadcastClient.class are located at~css503/programming/prog4/java/
- (b) Node 1 through to 7 may be any seven machines of uw1-320-01 ~ uw1-320-23.
- (c) yourUdpPort and your TcpPort may be identical.
- (d) Start node1, node2, node3, node4, node5, and node6 in this order. Thereafter, run node7.

```
...uw1-320-11:~/programming/prog4 — ssh — 80x24
[css503@uw1-320-11 prog4]$ pwd
/net/metis/home2/css503/programming/prog4
[css503@uw1-320-11 prog4]$ ./UdpRelay 237.255.255.255:50764
UdpRelay: booted up at 237.255.255.255:50764
% UdpRelay: registered uw1-320-15
UdpRelay: received 18 bytes from uw1-320-15 msg = Hello!
UdpRelay: broadcast buf[22 bytes] to 237.255.255.255:50764
```

```
...320-12:~/programming/prog4/java — ssh — 80x24
[css503@uw1-320-12 java]$ pwd
/net/metis/home2/css503/programming/prog4/java
[css503@uw1-320-12 java]$ java BroadcastServer 237.255.255.255:50764
uw1-320-17.uwb.edu: Hello!
```

```
...uw1-320-13:~/programming/prog4 — ssh — 80x24
[css503@uw1-320-13 prog4]$ pwd
/net/metis/home2/css503/programming/prog4
[css503@uw1-320-13 prog4]$ ./UdpRelay 238.255.255.255:50764
UdpRelay: booted up at 238.255.255.255:50764
% UdpRelay: registered uw1-320-15
UdpRelay: received 18 bytes from uw1-320-15 msg = Hello!
UdpRelay: broadcast buf[22 bytes] to 238.255.255.255:50764
UdpRelay: received 17 bytes from uw1-320-15 msg = Olá!
UdpRelay: broadcast buf[21 bytes] to 238.255.255.255:50764
```

```
...320-14:~/programming/prog4/java — ssh — 80x24
[css503@uw1-320-14 java]$ pwd
/net/metis/home2/css503/programming/prog4/java
[css503@uw1-320-14 java]$ java BroadcastServer 238.255.255.255:50764
uw1-320-17.uwb.edu: Hello!
uw1-320-17.uwb.edu: Olá!
```

```
...uw1-320-15:~/programming/prog4 — ssh — 80x24
[css503@uw1-320-15 prog4]$ pwd
/net/metis/home2/css503/programming/prog4
[css503@uw1-320-15 prog4]$ ./UdpRelay 239.255.255.255:50764
UdpRelay: booted up at 239.255.255.255:50764
% add uw1-320-11:50764
UdpRelay: registered uw1-320-11
UdpRelay: added uw1-320-11:6
% add uw1-320-13:50764
UdpRelay: registered uw1-320-13
UdpRelay: added uw1-320-13:7
% UdpRelay: relay Hello! to remoteGroup[uw1-320-11:50764]
UdpRelay: relay Hello! to remoteGroup[uw1-320-13:50764]
delete uw1-320-11
UdpRelay: deleted uw1-320-11
% UdpRelay: relay Olá! to remoteGroup[uw1-320-13:50764]
delete uw1-320-13
UdpRelay: deleted uw1-320-13
%
```

```
...320-16:~/programming/prog4/java — ssh — 80x24
[css503@uw1-320-16 java]$ pwd
/net/metis/home2/css503/programming/prog4/java
[css503@uw1-320-16 java]$ java BroadcastServer 239.255.255.255:50764
uw1-320-17.uwb.edu: Hello!
uw1-320-17.uwb.edu: Olá!
uw1-320-17.uwb.edu: Adeus!
```

```
...320-17:~/programming/prog4/java — ssh — 80x24
[css503@uw1-320-17 java]$ pwd
/net/metis/home2/css503/programming/prog4/java
[css503@uw1-320-17 java]$ java BroadcastClient 239.255.255.255:50764 Hello!
[css503@uw1-320-17 java]$ java BroadcastClient 239.255.255.255:50764 Olá!
[css503@uw1-320-17 java]$ java BroadcastClient 239.255.255.255:50764 Adeus!
[css503@uw1-320-17 java]$
```

7. Teamwork

You may work on this final project independently or in teamwork with another student, (i.e., a group of up to two students including you). If you work in a team, you must comply with the following guidelines:

(1) **The work assignment table:** All team members may turn in a joint report. (I am reluctant to repeat reading the same reports.) However, the report must include the work assignment table that details who worked on which portion of code, functionality, and/or tiers.

(2) **Confidential collegial evaluation:** Please evaluate your partner's contribution to the project, envelop it, and turn in this evaluation with your joint report.

8. What to Turn in

This programming assignment is due at the beginning of class on the due date. Please turn in the following materials in a hard copy. No email submission is accepted.

Criteria	Grade
Documentation of your UdpRely implementation including explanations and illustration in one or two pages. No need to repeat this work specification. (No more than two, otherwise – 2pts).	20pts
Source code that adheres good modularization, coding style, and an appropriate amount of commends. • 25pts: well-organized and correct code receives • 23pts: messy yet working code or code with minor errors receives • 20pts: code with major bugs or incomplete code receives	25pts
Execution output that verifies the correctness of all your implementation as well as covers many test cases. • 25pts: Statement of Work Step 3's snapshot plus additional outputs for add, delete, show, help, and quit. • 20pts: Statement of Work Step 3's snapshot plus additional outputs that however do not include all commands. • 15pts: Statement of Work Step 3's snapshot only • 15pts: Some outputs for add, delete, show, help, and quit only (but not including Statement of Work Step 3's snapshot) • 10pts: Some marginal outputs • 5pts: No results	25pts
Discussions in one or two pages. (No more than two, otherwise – 2pts) • Limitation and possible extension of your program (+20pts) • Discussions on Statement of Work Step 3's snapshot (+5pts)	25pts
Lab Session 4 If you have not yet turned in a hard copy of your source code and output or missed this session, please turn in together with program 4.	5pts
Teamwork is evaluated by checking if your project is achieved in collaboration with another student. No work assignment table receives -5pts. A poor collegial evaluation receives -5pts.	(-10pts)
Individual Work receives +5 extra pts.	5pts
Total Note that this final project takes 17% of your final grade.	100pts