



## CEG 4183: Higher Layer Network Protocols, Fall 2007

### Project Outline

#### General Information

There will be two (2) lab groups per project. All members in both groups will participate in the design phase of the project, but each group will do their own implementation. At the end of the project, the 2 implementations must be compatible; e.g., the client program of one group must work with the server program of the other group, and vice versa.

Each group will demonstrate their project to the TA, including the demonstration of compatibility between the 2 groups. The project can be done at home or at school; however, it must be demonstrated to the TA at school. Hence, ensure that your project can run on the computers in STE 2060 (or on your laptop) in a manner that is demonstrable to the TA. The TA responsible for the project is Stejarel Veres [cveres@site.uottawa.ca](mailto:cveres@site.uottawa.ca).

#### Topics

Some possible project topics are given on page 3. Note that you may choose a topic not listed there as long as you get the professor's permission. After receiving permission from the professor for an unlisted project **OR** selecting a project that is listed here, please register your topic directly with the TA.

#### Design

The members of both groups will work together for the design phase. This design includes:

- Protocol Message Type
- Header (Syntax and Semantics)
- Exchange rules
- Delivery Mechanism (UDP, TCP, ...?)

Consensus must be reached on the design before proceeding to implementation. See lecture 10 (specially slide 10-6) for design guidelines. Note that the design phase is the **ONLY** part of the project where both groups work together. Anything else, such as implementation and report writing, must be done separately by each group!

**Important Note:** in this course's project, you **MUST** directly work with sockets. As such, using CORBA, SOAP, or any other communication framework is not allowed!

## Implementation

Each of the two groups will do their own implementation. At the end of the project, the 2 implementations must be compatible; e.g., the client program of one group must work with the server program of the other group, and vice versa. Any programming language can be used.

## Marking Scheme

Mark will depend on the complexity of the project (less-complex projects will receive fewer marks), final result and the practical outcome, compatibility between the two groups, and the quality of report. Up to 15% bonus will be given to all 4 members if the two groups use different programming languages for their implementation. **Note:** do not spend much effort for the User Interface (UI). This is not a UI-design course, and as such, a simple UI will get you the same mark as a fantastically-done UI. The focus should be on the design of the networking components. Of course, the UI must be reasonable: good enough for a demo to the TA.

## Report Format

Each group will submit one report. Except for the title page, the entire project must be written in 12-point single-spaced Times New Roman text. Headings can be of larger size. In terms of the content, the following format should be followed:

- 1- **Title page:** project title, course code, full name and student # of group members, and date.
- 2- **Abstract:** summarize in one paragraph (no more than half a page) what the goal of the project was, how it was done (Java, C, ...), and what the overall outcome was, all without going into details. Do not write anything else on this page!
- 3- **Table of Contents:** with section headings and their corresponding page number.
- 4- **Introduction:** one to three pages explaining what the purpose of the project is, what are its detailed goals, what are its applications in the real world, and what will be learned as a result of doing this project?
- 6- **Design:** present the design behind the project in detail, including the design choices made and the architectural structure of the software, as well as the design of the Protocol Message Types, Header (Syntax and Semantics), Exchange rules, and Delivery Mechanism.
- 7- **Implementation:** describe your implementation approach, programming language, platform, technical constraints, deployment issues, etc. Do not include program code in this section; rather, present the code in the appendix and refer to it from this section if needed. You may include code snippets for more detailed explanation.
- 8- **Observations:** include screen shots, summarize subjective experience with the application, and discuss the outcome, including any problems.
- 10- **Conclusion:** one to three pages summarizing if the goal was achieved, what we learned from the project, difference between theory and reality, sources of error, and any concluding remarks.
- 11- **Appendix:** include program code here. If you're using tools that generate a lot of code (like Visual Studio or Eclipse), only include the code that you have written. Keep the length reasonable.

## Deliverables and Deadlines

Topic Registration with the TA: October 18<sup>th</sup> 2007 5:30PM, by emailing the TA.

Project Help Session: November 23<sup>rd</sup> 2007 1PM, in class.

Project Demo to the TA: week of November 26<sup>th</sup> (or earlier if it's ready) during normal lab hours.

Project Report due in the drop box: December 10<sup>th</sup> 5:30PM in the box (**also** email soft copy to TA)

## TOPICS

### **IP Multicasting Groupware**

Choose a many-to-many application of your choice, such as a shared whiteboard or a simple multiplayer (3 or more players) game, and implement IP Multicasting communications for it. In other words, you will designate a class D IP address for the application's multicast group, and all messages will be exchanged via this address. This will work on a given LAN segment, such as the one in STE 2060, but will not work beyond that for reasons explained in the class. On the LAN segment, anyone who listens to this IP address will receive messages from other group members, and anyone who sends a message to this IP address will have it delivered to all members: this is automatically taken care of by IP Multicasting.

### **Secure Audio Channel**

Design and implement a program that allows two people to talk over a secure channel on the Internet, meaning that if an intruder captures the audio packets s/he will not be able to understand the conversation. One way to do this would be to encrypt the audio at one end and decrypt it at the other end. In this case, you can assume the users already know each-other's encryption keys (no need to implement key-exchange protocol). Another way would be to use SSL, but this might not yield the required real-time performance. Do not worry about lost packets. Some delay is allowed, even beyond natural conversation thresholds, but minimizing it is highly encouraged.

### **Peer-to-Peer Chat**

Design and implement a program that allows multiple people to have a chat session without a central server. The system should work in a Peer-to-Peer manner; i.e., the users themselves form an overlay tree and a given message is exchanged from one person to another until everyone has received the message. The computer of the person who initiates the chat session (let's call it computer F1) will be the first point of contact for newcomers. When a new person joins, F1 decides the position of the newcomer in the overlay tree. When a person leaves the chat session, F1 reconstructs the tree making sure that the children of the person who just left are reconnected to the tree. See lecture 16 for more details.

### **Interplanetary Communications: Remotely Controlling the Mars Probe**

TCP's reliability is based on acknowledgements; i.e., the sender re-transmits a packet that has not been acknowledged. While this works fine on earth, it is not suitable for interplanetary communications because of large end-to-end delay. From earth, a radio signal takes about 10 minutes to reach mars. This implies a retransmission timeout of at least 20 minutes (!) for TCP, far too long to do reasonable communications. To solve this problem, we can use Forward Error Correction (FEC), whereby the receiver itself corrects errors or even recovers lost packets completely, with a good probability. See [http://en.wikipedia.org/wiki/Forward\\_error\\_correction](http://en.wikipedia.org/wiki/Forward_error_correction) for starters and choose one of the algorithms presented there and implement it over UDP, demonstrating that many corrupted packets are indeed recovered at the receiver itself and without retransmission.

### **TCP for the Future Internet**

Unlike today's Internet where users are limited to a few megabits per second, future networks will be much faster, sometimes in the order of gigabits per second. Today's TCP will not be suitable for such networks. For example, on a 10 Gbps link with a 100 ms round-trip time, it takes TCP (assuming 1500-byte segments) 83,333 segments to reach optimum speed, due to TCP's slow start phenomenon (congestion control). To alleviate this problem, protocols such as Fast TCP, HighSpeed TCP, and Scalable TCP have been proposed. For this project, pick one of these protocols and implement their speed-up algorithm over UDP.