# Advanced Computer Networks

# Project 2: File Transfer Application

Assigned: April 26, 2013

Due: May 31, 2013

## 1 Overview

In this assignment, you will implement a file transfer application. The application will run on top of UDP, and you will need to implement a reliable and congestion control protocol (similar to TCP) for the application. The application client will be able to download a set of files from a server, which is directly connected to the client's host. Your task is to finish the transferring task as soon as possible.

## 2 Project Outline

During the course of this project, you will do the following:

- Implement a file transfer application and minimize the transferring time.
- Implement a congestion control mechanism to ensure fair and efficient network utilization

## 3 Project specification

### 3.1 Scenario

This project is loosely based on a traditional file transfer application, the client knows which server has the file, and sends a request to that server for the given file. In this project, since you can only use UDP as the underlying transfer protocol, you should implement the reliability and congestion control for your application, i.e., you have to design a retransmission mechanism to make sure the file downloaded is exactly the same as the copy in the server. Besides when congestion arise, your application should be aware of this and take necessary measure to mitigate the situation.

The scenario of which your application works is a simple "one-link-network", Figure 1, i.e., two machine connected to each other with only one link. Considering this is the only scenario that is required, you can optimize your application in whatever ways you want. The only constraint is USING UDP as your transfer layer.

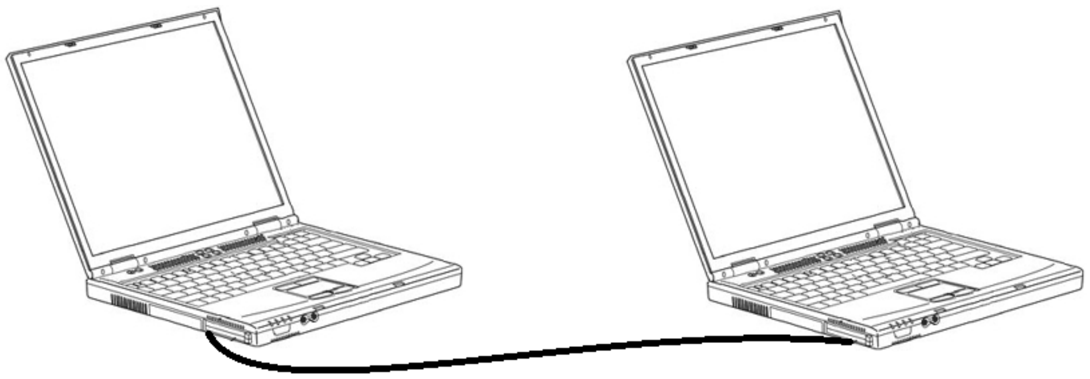

Figure 1 : the "one-link-network" scenario for evaluation

## 3.2 Programming Guidelines

Your peer must be written in the C programming language, no C++ or STL is allowed. You need to use UDP for all the communication for control and data transfer. Your code must compile and run correctly on linux machines. Refer to slides from past recitations on designing modular code, editing makefiles, using subversion, and debugging. As with project1, your implementation should be single-threaded. For network programming, you are not allowed to use any custom socket classes, only the standard libsocket and csapp libraries. These guidelines are similar to project1, except that you may freely use any code from your project1 (even if you switched partners). However, all code you do not freshly write for this assignment should be clearly documented in the README.

## 3.3 Project Tasks

There several tasks to evaluate your application's performance:

### 3.3.1 Task 1 – 100% Reliability & Sliding Windows

The first task is to implement a 100% reliable protocol for file transfer (ie: DATA packets) between two peers with a simple flow-control protocol. Non-Data traffic does not need to be transmitted reliably or with flow-control. To achieve reliability and ordered delivery, you need to use sliding windows on both sides (sender, receiver) of the connection. In this case, the sender is the node that already has the file, while the receiver is the node that sends the request to begin downloading the file. To start the the project, use a fixed-size window of 8 packets. The sender and receiver should ignore packets that fall out of the window.

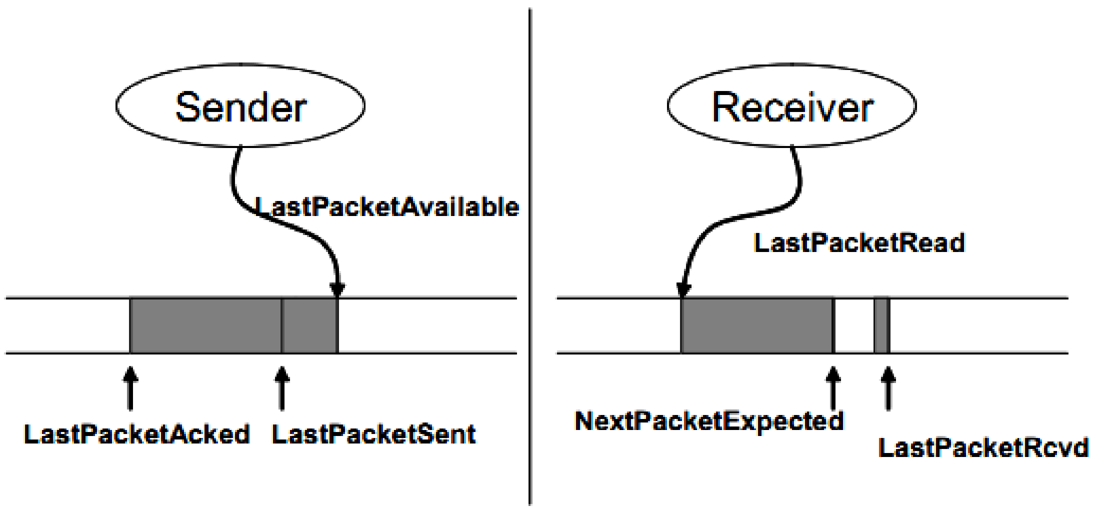

Figure 2: Sliding Window

Figure 2 shows the sliding windows for both side. The sender slides the window forward when it gets an ACK for a higher packet number. The receiver slides the window forward when the application reads more packets bytes from the buffer to increment the LastPacketRead There is a sequence number associated with each packet and the following constraints are valid for sender and receiver (hint your peers will likely want to keep state very similar to that shown here):

Sending side

- LastPacketAcked ⩽ LastPacketSent
- LastPacketSent ⩽ LastPacketAvailable
- LastPacketAvailable− LastPacketAcked ⩽ WindowSize
- packet between LastPacketAcked and LastPacketAvailablemust be “buffered” – you can either implement

this by buffering the packets or by being able to regenerate them from the datafile.

Receiving side

- LastPacketRead < NextPacketExpected
- NextPacketExpected ⩽ LastPacketRcvd + 1
- LastPacketRcvd− NextPacketExpected ⩽ WindowSize
- bytes between NextPacketExpected and LastPacketRcvd must be “buffered.” As above, you could implement this by actually buffering the data, or by writing the data in its correct order to the data file.

When the sender sends a data packet it starts a timer for it. It then waits for a fixed amount of time to get the acknowledgment for the packet. Whenever the receiver gets a packet it sends an acknowledgment for NextPacketExpected. That is, upon receiving a packet with sequence number = 8, the reply would be “ACK 8”, but only if all packets with sequence numbers less than 8 have already been received. These are called cumulative acknowledgements. The sender has two ways to know if the packets it sent did not reach the receiver: either a time-out occurred, or the sender received “duplicate ACKs.”

- If the sender sent a packet and did not receive an acknowledgment for it before the timer for the packet expired, it resends the packet.
- If the sender sent a packet and received duplicate acknowledgments, it knows that the next expected packet (at least) was lost. To avoid confusion from re-ordering, a sender counts a packet lost only after 3 duplicate ACKs in a row.

When you send a request to that sever host, set a timer to retransmit the request after some period of time (less than 5 seconds). You should have reasonable mechanisms to recognized when successive timeouts of DATA or request traffic indicates that a hosts has likely crashed.

### 3.3.2 Task 2 – Optimizations:

Extra Credit / Competition Section

For this section, we would measure how well you can optimize the speed of file transferring. We will have different sets of files for transferring mission, your application should be intelligent enough to handle all given cases. In order to achieve high throughput, you may design some heuristic method to reduce the cost of acknowledging and retransmitting. Also, the window size also plays a crucial in affecting the transferring speed.

Below is the typical types of file sets to be transferred:

- A single huge file, e.g., a movie file, which is usually several GB in size;
- A mass of tiny files, e.g., log files, which are usually one or two KB in size;
- A combination of large files and tiny files.

# 4 Grading

This information is subject to change, but will give you a high-level view of how points will be allocated when grading this assignment. Notice that many of the points are for basic file transmission functionality and simple congestion control. Make sure these work well before moving to more advanced functionality or worrying about corner-cases.

- **Basic file transferring [50 points]:** The client should be able to set up connection with the server with explicit message, and transfer the specific set of files correctly. This section requires reliability to handle packet loss.
- **Robustness [20 points]:** Your implementation should be robust to crashing peers, and should attempt to restore automatically from the break point.
- **Efficiency [up to 40 points]:** You should implement heuristics and protocol techniques that will help your peers transfer files faster. We will measure the

average download speed for multiple cases.

## 5 Hand-in

As in project 1, code submission for the final deadline will be done through your subversion repositories. You will receive an email with your Team#, Person#, and associated password soon after the assignment is posted. You can check out your subversion repository with the following command where you must change your Team# to “Team1” for instance, and your P# to the correct number such as “P1”:

The grader will check directories in your repository for grading, which can be created with an “svn copy”:

- Checkpoint X – YOUR REPOSITORY/tags/checkpointX
- Final Handin – YOUR REPOSITORY/tags/final
- Contest Handin (optional) – YOUR REPOSITORY/tags/contest
- Extra Credit (optional) – YOUR REPOSITORY/tags/extracredit