The University of Queensland
School of Information Technology and Electrical Engineering

COMS4200/7200
Semester 2, 2013

Assignment 1 (Weighting 25%, 100 Marks)

This Assignment is due Friday, 6/9/2013, 11pm

A Simple Reliable Transport Protocol (SRTP) and File Transfer Application

Goal

The aim of this assignment is to implement the key functionality of a simple, reliable transport protocol (SRTP) on top of UDP, as part of a file transfer application, similar to TFTP (RFC 1350). The main task is to provide reliability of the protocol by implementing an Automatic Repeat Request (ARQ) mechanism.

Since we are not implementing our protocol in the kernel, such as is typically the case for transport protocols such as TCP, the protocol will not be generic, but will be integrated with a simple file transfer application.

The best source of information for this assignment is the book "UNIX Network Programming, The Sockets Networking API', W. Richard Stevens et al. In the following, we will refer to this book simply as the 'textbook'.

For this assignment, you need to make quite a few important design decisions, and there will be a few challenges and problems along the way. You are allowed and encouraged to discuss high level concepts via the newsgroup, but please do not share any code of more than a couple of lines. Please don't hesitate to ask if you have any questions.

Protocol Functionality

In this assignment, you will implement a simple message-based, reliable transport protocol on top of UDP. Since UDP is connectionless and unreliable, you need to implement the following functionality:

- Reliability
  - The most important feature of the protocol is reliability. Messages (packets) need to be delivered reliably, even in the case of an unreliable network with packet loss and data corruption. Reliability is to be implemented via an Automatic Repeat Request (ARQ) mechanism. More details on reliability are provided below.

- In-order data delivery
  - SRTP needs to guarantee that messages are received by the application at the receiver in the same order as they were sent. This means the protocol needs to be robust against packet reordering in the network. It is also required each message is delivered only once, i.e. no duplicated messages are delivered to the application at the receiver.

- Connection oriented
  - Implementing reliability requires keeping state at both the server and the client, which entails that a connection needs to be established.
  - Unlike TCP connections, SRTP connections can be unidirectional or *simplex* for data. Data can only flow from the client to the server. The server only sends control packets to the clients, e.g. acknowledgements.

You do not need to implement congestion control or flow control, which is a functionality typically found in transport protocols such as TCP. However, you need to discuss in your implementation report, how these two features can be implemented and integrated with your ARQ mechanism and transport protocol.

SRTP Protocol

For your protocol, you need to design and implement the following:

Protocol API
Even though there will be some unavoidable integration between the protocol and the application, you need to define a clear API that separates the protocol functionality from the application. This needs to be clearly described in the implementation report.

Packet Format and Header
You need to define a packet header for your SRTP protocol, and clearly describe its syntax and semantics in the implementation report. For example, you will need a sequence number field in the header
Important: Your code should run correctly on both little endian and big endian machines.

Connection Establishment and Teardown
You need to design and implement a simple connection establishment and teardown mechanism for your protocol. For this, you need to design and implement a state machine at both the client and the server. This can be inspired by TCP.

ARQ Protocol

For this assignment you need to implement reliability using the go-back-n ARQ protocol. (For example, see Chapter 3 of the book "Computer Networks", by Andrew Tanenbaum, or "Computer Networking", Section 3.4, by Jim Kurose for a detailed description).

Alternatively, you have the option of implementing the considerably simpler *stop-and-wait* protocol. However, in this case the maximally achievable number of marks for the ARQ mechanism is capped at 30, compared to a maximum of 50 marks for *Go-back-N. (*More details on the assignment marking is provided below.)

An important aspect of the ARQ protocol is the choice of the **retransmission timeout**. The overall goal is for the protocol to have good performance (throughput) under a wide range of network path conditions, i.e. with different bandwidth and end-to-end delay. It is important that you discuss your design choices and potential trade-offs that you have to make in your implementation report. Another important feature you need to provide is error detection, i.e. you need a mechanism that allows you to detect corruption of packets. For simplicity, you can consider using the UDP checksum for this, but you need to be aware of its limitations.

Concurrency

The server needs to support concurrency, i.e. it needs to be able to handle multiple client requests/connections at the same time. A simple way to implement a concurrent TCP server is via the *fork()* system call. (see the textbook Section 4.8 Concurrent Servers).

The problem is that we are not using TCP and therefore we cannot call *listen()* and *accept()* on a socket. The *accept()* call returns a new socket descriptor for a socket on the server with the same port number, and the kernel makes sure that any new data for this socket from the client is passed to the right process. In UDP, this does not work. Section 22.7 (Concurrent UDP servers) of the textbook talks about this problem and provides a potential solution.

If blocking I/O and *fork()* are used to provide concurrency, you need to make sure that processes are properly terminated, i.e. you need to clean up any *zombie processes.*

Handling of Errors

If either side of the connection detects that the other side is misbehaving (i.e. deviation from the expected protocol behaviour), it should reset the connection, e.g. by sending a RST packet as in the case of TCP.

However, be careful not to interpret an unreliable network as protocol misbehaviour of the other side. Control packets can be delayed, duplicated or reordered in an unreliable network.

The server must be able to handle misbehaving clients and not crash/exit or close other client connections.

Application: FTP – Client and Server

To simplify the implementation of the protocol, the protocol functionality is integrated with an application, i.e. we assume a simple file transfer application.

You need to implement a simple server and client to test your protocol implementation, and they need to implement the following interface and features:

The client is started as follows: *ftclient [–d] <server> <port> < filename>*

The client establishes a SRTP connection to the server with address or name <server> on port <port>. The ftclient needs to accept both IP addresses (e.g. 192.168.1.1) as well as fully qualified domain names (e.g. moss.itee.uq.edu.au) for the <server> command line parameter.

After successfully connection establishment, the client transmits the local file with name <file>, terminates the connection and then exits.

The server is started as follows: *ftserver [–d] < port>*

The server listens for a SRTP connection on port <port>, and after a connection has been successfully established, receives the file and stores it in the local directory.

The file to be transferred needs to be stored at the server in the local directory from where the server was started, under its original name. If a file with such a name already exists in the server's directory, the transfer should be aborted, and an error should be reported by the client application.

Both the server and client need to support an optional debug option, enabled via the *–d* flag. When enabled, this feature should print out relevant and meaningful information about the state and actions of the program and protocol.

The client and server application need to properly parse command line parameters and implement error handling, e.g. deal with invalid inputs such as negative port numbers, etc.

Testing

You need to come up with a suitable method to test your implementation. To test the reliability mechanism, you need a way to emulate packet delay, packet loss, packet re-ordering and packet duplication. You also might want to consider limiting the bandwidth of your links, in order to test scenarios with different delay bandwidth products.

netem is a Linux tool that provides these features:
http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

You need to discuss your testing approach and methodology in the implementation report.

Implementation Requirements

Submission needs to come with a makefile that compiles on the student server. You will need to show how you build your project from scratch as part of the demo.

It is a good idea to put all the protocol related definitions, in a single header file, *srtp.h.*

The server application should be implemented in the file *ftserver.c*.
The client application should be implemented in the file *ftclient.c.*

Implementation Report

The implementation report needs to address the following points:

- Detailed description of your design of the SRTP protocol, including its API, packet format, connection establishment mechanism, ARQ mechanism, etc. The description should be precise, concise, and provide sufficient detail to allow someone else to implement a client or server, compatible with your implementation. This description or specification could be in the format of an IETF RFC, for example.

- The Implementation Report you should provide a critical discussion of the main design decisions you made, the main challenges you have encountered and the remaining problems and bugs. This includes issues such as:
    - Implementation of concurrency, choice of retransmission time-out, etc.
    - Testing mechanism
    - …
- The report needs to include a brief discussion on how flow-control could be implemented and integrated in the ARQ protocol

Modifications to Specifications

The specifications might not be 100% complete or 100% clear to you. **Please do not hesitate to ask if you have any questions regarding the assignment.** Please also monitor your email, the course newsgroup, or the course website for clarifications and/or corrections to the above information.

Marking

An important aspect of the assessment for this assignment is a live demonstration of your implementation. You will be asked to demonstrate the different features of your implementation, as well as to explain different parts of your source code.

The marks for the assessment of this assignment (Total: **120 marks**) are as follows:

- Functionality (**90 marks)**
    - Connection Establishment and termination (10 marks)
    - ARQ protocol, Go-back-N (50 marks)
        - Max 30 marks for stop-and-wait
    - Server and client application functionality (10)
    - Concurrency (10 marks)
    - Error handling (10 marks)

- Organisation of Code , Documentation, Coding Style, makefile and README file **(10 Marks)**
    - Your code must adhere to a consistent coding style. At the beginning of the project, each team needs to define a common coding style. You need to define how you are going to use comments, how to give meaningful names to variables and functions, how you format your code etc.
    - It is absolutely crucial that you provide a lot of comments in your code. When marking your assignment, coding style, clarity and an appropriate amount of helpful comments will have a significant weight. In case your code does not work as it should, it is a lot easier to give partial marks if the code is clear and it is easy to see what you are trying to do.
    - You need to provide a makefile that allows building your application.
    - You need to create a README file that describes how to build and use your application. Special information such as limitations of the functionality, deviations from the specifications and known bugs need to be mentioned here.

- Implementation Report **(20 Marks)**
    - The implementation report needs to contain the relevant information, as described above.

Academic Merit, Plagiarism, Collusion and Other Misconduct

You should read and understand the statement on academic merit, plagiarism, collusion and other misconduct contained within the course profile and the document referenced in that course profile. Submitted source code will be subject to electronic plagiarism detection.

Submission Instructions

The submission deadline is **Friday 6/9/2013, 11pm.**

Penalties for late submission apply as specified in the course profile.

The assignment needs to be submitted via Blackboard. More details on this will be provided closer to the submission deadline.

The following items need to be submitted:

- A 'tarball' with all your source code, including a working makefile and a README file
- Implementation report (pdf format)