TDTS06: Lab 4 – Fragmentation and MTU discovery

Juha Takkinen, Ph.D.

IDA, Dept. or Computer and Information Science

1.0 Overview

Your goal in lab 4 is to add fragmentation and discovery of the Maximum Transmission Unit (MTU) to the protocol.

You must also, finally, answer a number of follow-up questions concerning the lab assignment.

2.0 What to do first

Copy the files from the lab3 directory to the lab4 directory.

Then, update the protocol stack so that it includes your protocol that you will develop in lab 4: edit the graph.comp file and replace each occurrence of lab3 with lab4.

Make sure that your new copy of the protocol in the lab4 directory still works by recompiling x-kernel and running the server and the client as in lab 3.

3.0 Lab requirements

3.1 Fragmentation (and reassembly)

The messages that the test program pushes to your RDT protocol must now be fragmented by the client before sending them to the server. That is, each message from the test program must be fragmented into smaller packets before transmission. Each fragment will have a header, just as the whole message had in previous labs.

The server must collect and reassemble the fragments into the original message *before delivering them to the application layer* on the server side.

For message fragmentation, use msgBreak to break the message into fragments. Then use msgPush and rdtHdrStore to append the header (see below) to the fragment.

When you reassemble the fragments into the original message at the server side, msgPop is used to retrieve the fragment data and the header of the fragment as separate parameters. This is already done by x-kernel in rdtDemux, where the message is first received from the network. Use msgJoin to reassemble the fragments into a message in rdtPop, which is the receiving function after rdtDemux.

3.2 MTU discovery

The size of the fragments must be provided as a command-line argument to the test program when the client is invoked. The size must then be transferred to the rdt protocol from the test program and subsequently detected by the rdt code.

This has been somewhat prepared for you already; search for packet_size in rdt.c and you will find the place in the code where the parameters are handled. The usage will be: xkernel -c128.1.2.3 filename.txt -pkt 8, that is, the switch “-pkt” followed by the requested fragment size; in this example the fragment/packet size is set to 8. Note that the file name comes before these switches!

The information from the command line is sent to lower layer protocols by calling xControlSessn; this call has been provided to you in the code. This also means that your RDT protocol in rdt.c already has alread been made aware of the packet value from the test program (see rdtControlSessn in rdt.c).

4.0 Testing the protocol

Configure the protocol stack in graph.comp so that it includes all three virtual protocols VDISTORT, VDELAY and VDROP. However, test the no-packet loss case first, i.e., disconnect the virtual protocols when you are testing your code. Then, add one virtual protocol at a time to your test runs, to ensure that the correct mechanisms are in place, until you have all virtual protocols installed.

Start by implementing the fragmentation in rdt.c. Use a hard-coded value for the fragment size. Then, implement the MTU discovery function to the command line and replace the hard-coded value with a call to the correct function and variable.

5.0 Follow-up questions

When you have finished coding and testing the protocol, you must answer the following questions:

1. List some common values of the MTU on the Internet. Try to explain the sizes that you find.
2. Explain how the MTU is related to the MSS used in TCP. Does the MSS include the TCP header? Exemplify.
3. Explain what Path MTU Discovery is and why it is needed.

Explain all of your answers and list correct sources, such as RFCs, if any.

6.0 Demonstrating the solution

Before handing in a lab report you must first demonstrate the resulting protocol to your lab assistant. Remember to remove unnecessary trace printouts and only keep the most important ones. You are recommended to slow down the execution of your protocol, using delays, for an even clearer demonstration.

Before you demonstrate your solution to the lab assistant, give him/her a copy of the code on paper.

In the demonstration you must show that your protocol works according to the requirements, that is, the protocol can handle a partly unreliable channel (vdistort installed), which is shown by clear trace

printouts of the packets that are corrupted and which ones are received correctly. Be prepared to answer questions on specifics in the code and your solution.

When your lab assistant has seen your demo and approved of the functionality, you can hand in a lab report containing the source code of the changed files on paper (see “Coding guidelines” on the course web site) and also the answers to all of the follow-up questions.

When you have been passed in lab 3 you can run the finishing script to hand in all of the code that you have developed in the lab series. See the course web site for more information about this finishing script.