

433-353 Networks and Communications

Project 2 : Packet fragmentation and reassembly

Department of Computer Science and Software Engineering
University of Melbourne

1 Introduction

A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed. The word datagram commonly refers to the Network Layer packet used by the Internet Protocol (IP). In this project we examine the IP version 4 (IPv4) packet header and fragmentation/reassembly. The format of the IPv4 packet header is shown in Fig. 1.

1 byte		1 byte		1 byte		1 byte	
version	HL	type of service		total length			
identification				D	M	fragment offset	
				F	F		
time to live		protocol		header checksum			
source address							
destination address							
options							

Figure 1: IPv4 header format.

Consult your lecture notes and text book for further details.

The Transport Layer (layer 4) of the OSI model generates data to be sent across the network and gives this data to the Network Layer (layer 3). Usually this data represents Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) information, but in this project the data will simply be text from/to standard I/O so that you may verify your results. You will need to implement Network Layer functions that both fragment large data to be sent and reassemble fragments that are received.

2 Program files

The program files are available at `/home/subjects/353/local/Project2` and includes a `Makefile`. Make a copy of these files and build the system. You may need to change the `Makefile` to omit/include the libraries. As is, it should compile on the Department servers without trouble. However it does not work as is, because the `layer3.c` send/receive functions do absolutely nothing.

NOTE: There are slight changes to fix some problems with the previous project 1 files, so DO NOT use the previous project files. Also, you will see that the Layer 2 CRC functions are there. You may check your previous project answer this way. You will not be required to modify Layer 2 or Layer 1.

The program operates similarly to Project 1. You first execute a `vhub` and then connect a `vhost`. The `vhub` is programmed to terminate after 10 minutes of operation (avoiding large numbers of forgotten `vhubs`). You may well change this by editing the `vhub` source code.

3 Fragmentation/reassembly

The Layer 2 message transfer unit (MTU) is 150 bytes. However Layer 4 is sending up to 1000 byte messages. Therefore, Layer 3 must fragment the messages into packets that will fit within the Layer 2 frames.

Fragmentation is discussed in the lecture notes (<http://www.cs.mu.oz.au/353/notes/node156.html>). The fields that are directly relevant to fragmentation are *total length*, *identification*, *don't fragment flag*, *more fragment flag* and *fragment offset*. The *header length* is also important (since the total length is the length of the data plus the length of the header), however the headers in this project will all have the same length. Other fields may be set to any value as they are ignored. Having said this, in practice the fragmentation and reassembly process must also check the source address, but we will not worry about this.

When data is received from Layer 4, in the form of a Layer 3 interface data unit (IDU), then the data must be put into one or more packets such that each packet is no more than 150 bytes in total length. Since the header length of a packet is 20 bytes, this leaves 130 bytes for data. No padding is required for data that is less than 130 bytes.

If the data exceeds 130 bytes then it must be put into several packets. These packets are then fragments. Each of the fragments is sent (as individual packets) by giving them to Layer 2. The receiver (peer Layer 3) must reassemble the fragments in order to extract all the data before giving the data up to the Layer 4. Reassembly is the opposite of fragmentation.

3.1 Points to note

- Fragment offsets are multiples of 8. This means that the number of bytes in each fragment, apart from the last fragment, must be a multiple of 8 (otherwise there will be “gaps” in the data). The largest multiple that is less than 130 bytes is 124, so each internal fragment can contain at most 124 bytes of data.
- The identification field is used to identify fragments that are part of the same message. You must use this field and may put any identifier that you wish (of course making sure to put the same identifier for fragments of the same message).
- Because the fragments that you send will be received in the order that you sent them, your reassembly becomes easier. However you **MUST** write your reassembly function so that it can:
 - reassemble fragments that are received *out* of order,
 - identify fragments using the identification field and so be able to reassemble fragments belonging to different messages and
 - be able to handle messages of the maximum possible lengths.
- Because of the requirements above, you will need to dynamically allocate memory as fragments arrive, since you cannot allocate enough memory for 2^{16} messages each of length 2^{16} . Buffer management is the difficult challenge for reassembly. Don't allocate memory that is never used!!
- You may write a test host that sends fragments out of order and as different messages in order to test that your reassembly algorithm works in the general cases. We will supply a stripped binary executable test host in the week prior to submission that does this for you (we can't give you the source code because it requires part of the solution).

4 Administration

This project is worth 15% of your total mark for this subject. Roughly 5% is allocated for the project report, 3% for correct fragmentation and 7% for correct reassembly. The due date for this project is set to Tuesday, October 28th, 5pm.

4.1 Groups

You should remain in the same groups that you formed for Project 1. You may change groups but this will be handled on a case by case basis, or we may post further instructions on how to go about nominating a group.

4.2 How to submit

Only one person in the group should submit. The other members should not submit.

You must submit a project report (report.txt) file and your modified layer3.c file. You should not modify or submit any other files:

```
prompt> submit 353 2 layer3.c report.txt
```

Your project report should be no longer than 2 pages. It should contain a complete description of all interesting variables that you needed to add to your program, all functions that you added and the general method that you used to fragment/reassemble the messages (you may use pseudo-code explanation). Your report is also an opportunity to express any problems that you encountered and any assumptions that you needed to make. You need not worry about the efficiency of your algorithms. You need only ensure that your algorithms are correct. A layer3.c that doesn't compile or that has no changes will most likely receive no marks.

Any missing/extra details will be posted to the 353 news group and WWW pages. Be sure to check both these sources for updates concerning this project.