

## 1 IP Fragmentation and Reassembly

We discussed this project in detail during lecture. Your program will perform fragmentation and reassembly of your simulated packets. Your program should fragment and reassemble these packets according to the IP algorithm.

## 2 Cooperation

I have divided out portions of this project for group work. The group work should not require much meeting time. As a group, the class must decide on

1. the format of the headers
2. the data structure for the header and data
3. the format of the message between router and client (same as between client and router)
4. the format of the message between routers

You should be able to do all or the majority of this group work via e-mail. You may NOT work as a group on any of the coding aspects of this project.

As a group, you must also decide how the router and its associated client communicate. The role of the client and router are described below. The only other part of the project that involves group work will be testing. For testing, I will combine clients and routers from different work to form a network.

You will need to customize your header since not all fields of the IP header are necessary for this assignment. When this project has been completed, the following outcomes should be realized.

- Understand thoroughly IP fragmentation and reassembly
- Understand the use of a header in peer to peer communication
- Understand some of the router's responsibility
- Be able to create socket-based programs
- Be able to create UDP-based programs

## 3 Preparation

All on-line work for this course will be done in a directory named CEN6520 in your UNIX account. Each programming project will be done in a separate subdirectory of the CEN6520 directory. Create the project2 directory and change to that directory for the remainder of this project.

```
mkdir ~/CEN6520/project2
cd ~/CEN6520/project2
```

## 4 The Problem

The objective of this project is to write C code to simulate the fragmentation and reassembly performed by IP. You are required to implement two types of processes: a `client` process and a `router` process. You will execute your router process for each router indicated in the network layout given below, and you will execute your client process for each client indicated in the network layout given below. Each client(router) process must be the same executable.

Many simplifying assumptions are made so the emphasis of the project is fragmentation and reassembly. Each client is connected to a unique router. A router can have at most one client connected to it. The layout of routers and clients are given below. The client's responsibility is to **send** and **receive** segments. To accomplish a send, the **client hands** a segment to its router. A receive is accomplished by the router **delivering** a segment to its client. You must decide how to handle the exchange of segments between client and router. This should be a *group* decision. If client A sends a segment to client B, then B must receive exactly one identical segment. Once a segment is handed to a router, it becomes a packet. You can assume the packet is not fragmented at the initial router, i.e., the router that is connected to the sending client. The router that is connected to the destination client must reassemble the packet and the client must receive the segment originally sent to it. Once a reassembled packet is handed to a client, it becomes a segment.

Your program must use UDP datagrams for communication between router processes. Although UDP does not handle lost or damaged datagrams, it will be safe to assume no lost or damaged datagrams if you test your program in the Unix lab, i.e., you do not have to ack your datagrams. You can hard code IPs and ports so your routers exist at well known TSAPs (IP, port pair). I suggest using `ifdefs/defines` or command line arguments to ease modifications. Each router should have only one UDP socket. More than one will cause your program to be unnecessarily complex. If you are developing your program at home, remember it will be tested in the Unix lab. You will need to make any necessary modifications (e.g., IP numbers) prior to the Unix testing.

## 5 Routing

Each `router` process must execute on a different machine Client processes must execute on different machines from each other,i.e., a a router and its connected client can execute on the same machine but two clients can not execute on the same machine.

A router is responsible for both fragmentation and reassembly of packets. Remember a packet is not reassembled until it reaches its final destination router. Each router will maintain a routing table to facilitate routing and determine when fragmentation/assembly is necessary. Assume non-adaptive routing! Remember this means the next hop to each destination is hard coded in the routing table. More than one datagram can arrive at a router *at the same time*. Your program must handle this situation. This will specifically be tested. Hint: a UDP socket can be specified as blocking.

The network layout is given as an undirected graph with vertices and edges. There are 10 vertices, labeled A, B, C, D, E, F, `Client1`, `Client2`, `client3`, and `Client4`. Only the edges listed below are present in the graph.

```
(client1, A) (client2, B) (client3, E) (client4, F)
(A,C)      (A,D)      (B,C)      (C,D)
(C,F)      (D,E)      (F,E)
```

The routing tables are given as ordered triples in the format (*Destination, Router, MTU*). Below are the tables for each router.

A::	B::	C::
B, C, 150	A, C, 150	A, A, 150
C, C, 150	C, C, 150	B, B, 150
D, D, 100	D, C, 150	D, D, 60
E, C, 150	E, C, 150	E, F, 70
F, C, 150	F, C, 150	F, F, 70
D::	E::	F::
A, A, 100	A, D, 75	A, C, 70
B, C, 60	B, F, 80	B, C, 70
C, C, 60	C, F, 80	C, C, 70
E, E, 75	D, D, 75	D, E, 80
F, E, 75	F, F, 80	E, E, 80

## 6 Testing

The minimum test cases are:

- Client 1 sends a segment to Client 3: segment size = 500 bytes
- Client 3 sends a segment to Client 2: segment size = 400 bytes
- Client 2 sends a segment to Client 4: segment size = 200 bytes
- Client 2 sends a segment to Client 3: segment size = 300 bytes
- Client 3 sends a segment to Client 4: segment size = 100 bytes

For testing, when a client sends a segments, have the client write this segment to a text file. When a client receives a segment, Write the received to another text file. By executing `diff` on the two files you can check to ensure that reassembly is functioning properly. Each segment must be unique.

Please see the sample `sendudp.c` and `recvudp.c` code. Note that this code is written in `main()`. You should not take this code as a starting point. Your code should be functionalized appropriately.

## 7 Grading

After projects are submitted, we will test the project as a group and then individually by appointment. Please set up the testing environment (e.g., open all window sessions, remove received segments) prior to our testing time.

## 8 Due Date: Friday, Feb 12th at 11:59pm

Late assignments will not be accepted.