Project 2: IP Packet reassembly support
CS 7473 Network Security
Noah M. Jorgenson

Project was at first glance much easier than the first project. This could partially be due to the fact that I was pleased with my initial architecture decided in project 1 which made most of project 2 relatively straight forward. To start I took the advice given in the assignment instructions and read the entirety of RFC 815 which gave me a great idea on how to reassemble the packets. I took their algorithm at a high level and implemented it via a Hole class I created (very simple < 20 lines) and a synchronized vector. Synchronization was important for this project but more on that later. To wrap the holes and vector of holes, I implemented a DatagramBuffer class which encompassed most of the functionality of project 2. The DatagramBuffer consists of the initial fragment that caused the creation of the DatagramBuffer, the vector of fragments passed through the buffer, the vector of holes.When a packet arrives it undergoes a match with each existing DatagramBuffer, if there isn't an exact match on the identifying fields the fragment is used to generate a new DatagramBuffer and added subsequently as the first fragment, otherwise the fragment is added to the DatagramBuffer since it matches and supposedly belongs to this fragment. The method to add a fragment to a DatagramBuffer is the meat of my implementation, it includes logic for resizing packets and ensuring the general algorithm outlined in RFC 815 is followed. To return the tuple of information including the SID, packet and fragments for a reassembled packet and the conditions which lead to the SID, I implemented a class called IDSTuple. The class fields match up to the tuple values. The IDSTuple has a toString method that I use to display the packet in a short one line description including the packet type, size, SID and number of fragments. This is just a proof of the IDSTuple's existence since they are not yet utilized by the system.

Testing my program involved lots of back and forth, trying to determine if I'd correctly built packets from the fragments. Usually I am able to code a fairly good first pass with limited debugging but this was different, I encountered many bugs and had to go back and forth between debugging and developing and opted to do both concurrently to increase my productivity. Ultimately my program appears to parse all the provided files correctly and reassemble the packets as I can see the correct port values are parsed. Also, timeouts for packets are handled every 2 seconds, with a 120 second lifetime on the DatagramBuffers which starts immediately upon their allocation into memory.

During testing I encountered memory issues which made me appreciate just how efficient an implementation on a small device must be, really puts things into perspective for me now that I'm really working so closely with the components which make up what I until recently took for granted (the internet and it's relative stability/consistency).

To run the PacketParser you can simply run PacketParser.java without arguments and it will parse live traffic. It is essentially the same frontend as the first project but with reassembly occurring for IP packets and IDSTuples returned with each reassembled datagram and prepended to the output of the reassembled datagrams. In the case of datagrams which are not fragmented they still receive and IDSTuple and are returned as expected.

Sample Output

```
[tcp](234 b) SID: 1 [1 fragments]
Source MAC: 00-04-75-8D-49-C7
Destination MAC: 00-AA-00-30-91-0D
EtherType: IPv4
Header Length: 20
Total Length: 220
Data Length: 200
Identification: 19155
FlagFrag Binary: 0100000000000000
Flags: 010
Last Fragment?: true
Fragment Offset: 0
First octet: 0
Last octet: 200
TTL: 64
Payload Start: 34
Protocol: TCP
Source Address: /192.168.36.30
Destination Address: /192.168.36.32
Source Port: 80
Destination Port: 1074

[tcp](387 b) SID: 2 [45 fragments]
Source MAC: 00-AA-00-30-91-09
Destination MAC: 00-04-75-8D-49-C7
EtherType: IPv4
Header Length: 20
Total Length: 28
Data Length: 8
Identification: 1725
FlagFrag Binary: 0010000000000000
Flags: 001
Last Fragment?: false
Fragment Offset: 0
First octet: 0
Last octet: 8
TTL: 128
Payload Start: 34
Protocol: TCP
Source Address: /192.168.36.32
Destination Address: /192.168.36.30
Source Port: 1075
Destination Port: 80
```