CS 7473 Network Security
Rujit Raval
Project 2

Goal: Add an IP packet reassembly facility to the packet capture tool you built in Project 1.

Concept: RFC 815 was helpful for the project to describe how to reassemble the packets with the concept of Hole. Algorithm is implemented in Hole class and through a synchronized vector. To wrap the holes and vector of holes DatagramBuffer class is used.

Usage:

```
Compile:        javac PacketParser.java
Run:            java PacketParser [-r filename] [...] [-r filename]
```

Implementation Details:

```
Based on RFC:  815
                         IP DATAGRAM REASSEMBLY ALGORITHMS
                                  David D. Clark
                          MIT Laboratory for Computer Science
                       Computer Systems and Communications Group
                                    July, 1982
```

Hole.java:

A partially reassembled datagram consists of certain sequences of octets that have already arrived, and certain areas still to come. We will refer to these missing areas as "holes". Each hole can be characterized by two numbers, hole.first, the number of the first octet in the hole, and hole.last, the number of the last octet in the hole.

```
public class Hole {
        int first;
        int last;
        public Hole(int first, int last) {
                this.first = first;
                this.last = last;
        }
        public String toString(){
                return String.format("Hole:<%s,%s>", this.first, this.last);
        }
}
```

Packet.java:

This class provides two variables to indicate type of the packet and variable to capture validity of the packet. E.g. String Packet_Type and boolean Is_Valid.

EthernetPacket.java:

public EthernetPacket(byte[] packet){}:
Sets the packet type as Ethernet.

public String toString(){}:
Adds the value of Source MAC, Destination MAC, Ethertype and data into the output.

private void Parse(){}:
Get the values of Source MAC, Destination MAC, Ethertype. Method sets the value of EtherType as either IPv4 or ARP.

public String ToHex(){}:
Method converts object to the hex data. This method is called by PacketParser class.

public static String HexString(byte[] b){}:
Method converts hex data into string. This is mainly used to convert type into string.

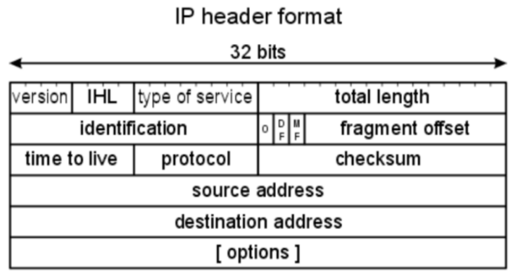
public static int BytesToInt(byte[] Data){}:
Method converts byte data into integer. Used to convert port values into integer in TCPPacket.

public static int UBytesToInt(byte b){}:
Method converts unsigned bytes to integer. Used in ICMPPacket class to convert code and type into integer and ARPPacket class to convert hardware address length and protocol address length into integer.

public static String BytesToBinary(byte[] b){}:
Method converts Bytes to the binary. Which is used to print the (Flags + Fragment offset) value in the binary in IPpacket.java.

public static int bytesToDecimal(byte[] b){}:
Method converts Bytes to the decimal which is used by IPPacket.java to get and print the values of Total length, identification and Time to live.

IPPacket.java:



Class IPPacket extends the class EthernetPacket.

New variables introduced are int IPIdentification, int IPFlagFrag, int IPTTL, int IPPayloadStart, int IPPayloadEnd etc.

public IPPacket(byte[] packet){}:
Sets the packet type as IP.
private void Parse(){}:
Get the values of Header length, protocol, Source IP, Destination IP. It also checks the IP protocol number and sets the protocol values such as TCP if '06', UDP if '11' and ICMP if '01'.

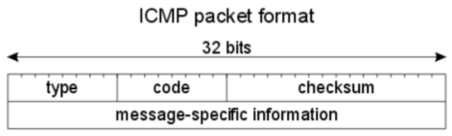
private int GetHeaderLen(char HLenChar){}:
Take a character of Header length as an argument and convert it into integer.

public String toString(){}:
Adds the value of header length, Data Length, Total Length, Identification, Flags, Time to Live, Payload start, Protocol, Source address and destination address into the output.

public boolean LastFrag(){}:
Returns true if MF(More Fragment) value is 0.

public IPPacket AddressFilter(InetAddress Source, InetAddress Destination, InetAddress SourceOR, InetAddress DestinationOR, InetAddress SourceAND, InetAddress DestinationAND) {}:
Method for filtering the addresses for arguments such as -src, -dst, -sord, -sandd which is called through PacketParser class.

ICMPPacket.java:



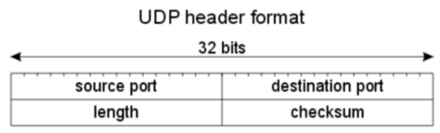
Class checks for 2 fields if the packet type is ICMP, ICMPType and ICMPCode and prints to the output.

public ICMPPacket(byte[] packet){}:
Sets the packet type as ICMP.

private void Parse(){}:
Get the value of type and code.

public String toString(){}:
Adds the value of type and code into the output.

UDPPacket.java:



Class checks for 2 fields if the packet type is UDP, source port and destination port, and prints to the output.

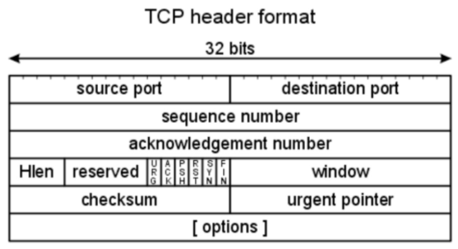
public UDPPacket(byte[] packet){}:
Sets the packet type as UDP.

private void Parse(){}:
Gets the value of source and destination ports.

public String toString(){}:
Adds the value of source and destination ports into the output.

public UDPPacket PortFilter(Integer SourcePortStart, Integer SourcePortEnd, Integer DestinationPortStart, Integer DestinationPortEnd)
Method for filtering the ports. Called through PacketParser class.

TCPPacket.java:



Class checks for 2 fields if the packet type is TCP, source port and destination port, and prints to the output.

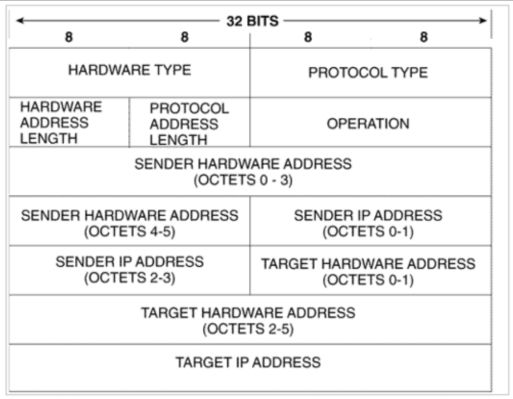
public TCPPacket(byte[] packet){}:
Sets the packet type as TCP.

private void Parse(){}:
Gets the value of source and destination ports.

public String toString(){}:
Adds the value of source and destination ports into the output.

public TCPPacket PortFilter(Integer SourcePortStart, Integer SourcePortEnd, Integer DestinationPortStart, Integer DestinationPortEnd)
Method for filtering the ports. Called through PacketParser class.

ARPPacket.java:


32 BITS
8
8
8
8
HARDWARE TYPE
PROTOCOL TYPE
HARDWARE ADDRESS LENGTH
PROTOCOL ADDRESS LENGTH
OPERATION
SENDER HARDWARE ADDRESS (OCTETS 0 - 3)
SENDER HARDWARE ADDRESS (OCTETS 4-5)
SENDER IP ADDRESS (OCTETS 0-1)
SENDER IP ADDRESS (OCTETS 2-3)
TARGET HARDWARE ADDRESS (OCTETS 0-1)
TARGET HARDWARE ADDRESS (OCTETS 2-5)
TARGET IP ADDRESS

Class checks for fields such as Hardware address length, Protocol address length, Sender hardware address, sender IP address, Target hardware address and Target IP address if the packet type is ARP and prints it to the output.

public ARPPacket(byte[] packet){}:
Sets the packet type as ARP.

private void Parse(){}:
Gets the value of Hardware address length, Protocol address length, Sender hardware address, sender IP address, Target hardware address and Target IP address.

public String toString(){}:
Adds the value of ARP fields into the output.

PacketParser.java:
This is the one with main class.

public PacketParser(){}:
Constructor for opening the adapter. For the better use, it asks for the adapter number to open and use to send the packets.

public static void main(String[] args){}:
Reads the user arguments and calls various methods given below.

private boolean HasAddressArgument(){}:
Returns true if any of the address argument is given either by '-src saddress' , '-dst daddress', '-sord saddress daddress' or '-sandd saddress daddress'.

private boolean HasPortArgument(){}:
Returns true if any of the port argument is given either by '-sport port1 port2' or '-dport port1 port2'.

private void SetSrcPort(String portStart, String portEnd){}:
Method to handle '-sport port1 port2' argument.

private void SetDestPort(String portStart, String portEnd){}:
Method to handle '-dport port1 port2' argument.

private void SetORAddress(String srcAdd, String dstAdd){}:
Reads the '-sord saddress daddress' arguments and set Source and destination addresses for OR.

private void SetANDAddress(String srcAdd, String dstAdd){}:
Reads the '-sandd saddress daddress' arguments and set Source and destination addresses for OR.

private void SetSrcAddress(String srcAdd){}:
Reads the '-src saddress' argument and set Source addresses for filtering the output.

private void SetDestAddress(String dstAdd){}:
Reads the '-dst daddress' argument and set Source addresses for filtering the output.

private void SetPacketType(String Type){}:
Method to set the packet type used in other classes.

private void SetHeaderOnly(){}:
Method to set HeaderOnly variable true if -h argument is given by the user.

private void SetPacketCount(String count){}:
Method to read the -c argument for filtering the number of packets.

private void SetInputFile(String in){}:
Method to read the '-r filename' argument for getting the filename to read from.

private void SetOutputFile(String out){}:
Method to read the '-o filename' argument for save output into the filename.

private void output(Object obj){}:
Method to print the output on console or save into the file. Instead of one it also takes object of SIDHandler as an argument.

private void init(){}:
Method to read the packets from live traffic if no '-r filename' argument given or read from the file if -r argument is set.

public static int HextoBytes(String Hex){}:
Method that converts hex pairs into bytes.

private class DatagramBufferWatcher implements Runnable{}:

Class that implements the thread which handles the package. This class reads the fragments from the buffer and calls the method output(). It also calls checkTimeOut() method and checks for the timeout of 5 seconds.

synchronized private DatagramBuffer NextBuffer(IPPacket fragment){}:
Method uses iterator and it is used to find the next value in the DatagramBuffer.

synchronized private SIDHandler AssemblePacket(IPPacket fragment){}:
Method assembles the fragments until the last fragment is found.

SIDHandler.java:
Introduces the concept of SID for project 3. 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 SIDHandler.

The frame is an ARP packet. Return an sid of zero, the ARP packet and a one packet list consisting only of the ARP packet.
final static int ArpSID = 0;

The IP fragments assemble correctly with no overlapping or oversize. Return an sid of one, the reassembled packet and a list of the fragments.
final static int CorrectSID = 1;

The IP fragments assemble correctly with overlapping. Return an sid of two, the reassembled packet and the list of fragments.
final static int OverlapSID = 2;

The IP fragments correspond to an IP segment larger than 64K. Return an sid of three, the first segment and a list of all fragments.
final static int OversizeSID = 3;

In the event that a fragment times out, return an sid of four, the first segment and a list of all partially processed segments with the same id.
final static int TimeoutSID = 4;

public SIDHandler(int sid, EthernetPacket packet, Vector<EthernetPacket> fragments){}:
Sets the value of sid, packet, fragments and packet type.

public String toString(){}:
Prints the value of PacketSize, SID and Number of fragments to the output.

DatagramBuffer.java:
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 is compared

with 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.

public DatagramBuffer(IPPacket Frag) {}:
Constructor sets various values for current packet.

public SIDHandler GenerateTuple(){}:
Sets the value of SID as 1 if it is correct, 2 if overlap is found and 3 if the packet is oversized.

public SIDHandler OversizeTuple(){}:
public SIDHandler TimeOutTuple(){}:
These methods call the SIDHandler with appropriate arguments.

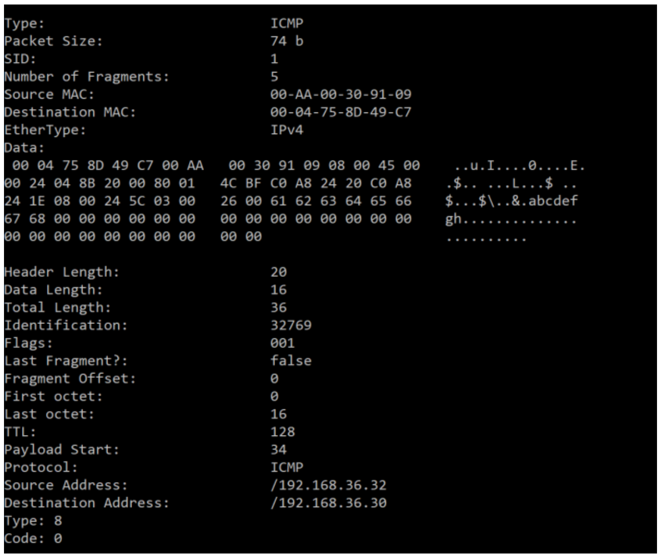
public void IsTimeOut(){}:
Method checks for the timeout.

public boolean Match(IPPacket Frag){}:
When a packet arrives, it is compared with existing DatagramBuffer. If it is matched, it returns true, false otherwise.

synchronized public SIDHandler AddFragment(IPPacket Frag) {}:
This method is used in PacketParser to add the fragments together until the last hole is found. Follows the RFC 815.

public static int bytesToDecimal(byte[] b){}:
Method converts bytes data into decimal.

Sample Output:



```
Type:                           ICMP
Packet Size:                    74 b
SID:                            1
Number of Fragments:            5
Source MAC:                     00-AA-00-30-91-09
Destination MAC:                00-04-75-8D-49-C7
EtherType:                      IPv4
Data:
 00 04 75 8D 49 C7 00 AA   00 30 91 09 08 00 45 00    ..u.I....0....E.
00 24 04 8B 20 00 80 01   4C BF C0 A8 24 20 C0 A8    .$.. ...L...$ ..
24 1E 08 00 24 5C 03 00   26 00 61 62 63 64 65 66    $...$\..&.abcdef
67 68 00 00 00 00 00 00   00 00 00 00 00 00 00 00    gh..............
00 00 00 00 00 00 00 00   00 00                      ..........

Header Length:                  20
Data Length:                    16
Total Length:                   36
Identification:                 32769
Flags:                          001
Last Fragment?:                 false
Fragment Offset:                0
First octet:                    0
Last octet:                     16
TTL:                            128
Payload Start:                  34
Protocol:                       ICMP
Source Address:                 /192.168.36.32
Destination Address:            /192.168.36.30
Type: 8
Code: 0
```