CS F303
Computer Networks

Group Number : 39

Network Packets Capture and Analysis

Final Report

Anirudh Kumar Bansal	2014A7PS081P
Chirag Agarwal	2014A7PS033P
Lakshit Bhutani	2014A7PS095P

https:github.com/compnetproj/packetcapture

April 19, 2017

Contents

1 Problem Statement

Design and implementation of a simple Packet-Capturing and Analysis Tool capable of capturing application, transport, network and data link level structured protocol data units and provide a simple visual analysis.

The protocols to be analyzed at each level are :

- Application layer Protocols : HTTP and DNS
- Transport layer Protocols : TCP and UDP
- Network layer Protocol : IP
- Link layer Protocol : MAC (Ethernet)

2 Scope of work

2.1 Project Overview

The tool captures packets on selected interfaces to which the machine is connected and further does the following analysis on it.

- Filtering of packets based on various properties
- Display of network traffic graph
- Conversation (exchange of messages) between two hosts
- Protocol hierarchy analysis
- Packet length analysis

2.2 Project Deliverables

It enables the user to choose an interface on which a given number of packets can be captured. Moreover various different scheme of analysis can be performed on the captured packets with multiple options available for each.

3 Design Challanges

1. On ordinary operation of socket packet capture, only packets sourced from or directed to the local machine was captured. Promiscuous mode refers to the special mode of Ethernet hardware, in particular network interface cards (NICs), that allows a NIC to receive all traffic on the network, even if it is not addressed to this NIC. Switching to this mode allowed us to overcome the challenge faced.
2. According to our earlier reports it was stated that the project will have a gtk based graphical user interface. However the complexity of implementation and design were a little cumbersome. Due to time constraints it was decided to follow simple terminal based display.

4 Implementation Problems

1. The project uses gnuplot for drawing traffic and packet length analysis graphs. The size of the graph was dynamic and scales needed to be adjusted so that all labels could fit properly.
2. Display of filtered packets and conversations required proper alignment of headings with the items in vertical space. This was done manually for suitable display and was considered not so fruitful by the team.

5 Limitations

- Root level privileges are required to run the tool.
- The project has been developed and tested on the Linux platform and its performance on any other operating system cannot be guaranteed.
- The number of packets captured are limited by 2000 at a time and have to be predeclared.
- It has been observed that packets do get garbled in high network traffic environment.

6 Architectural Design

The application uses raw sockets for packet capture. Usually the payload is encapsulated in transport layer segment and the application is unaware of the existence of protocol headers that are broadcasted with the payload. When the packets are captured using raw sockets, these headers are included with the payload and hence can be interpreted by the application layer.

7 Structural Design

The program has been modularized into a driver module and various utility functions which are called at appropriate times from the main code. The driver program has a menu based interface for the user to select options for packet analysis. The functionality to analyze a packet is further divided into various smaller functions where each function handles a particular protocol and related attributes.

8 Implementation Details

8.1 Sockets

A network socket is an internal endpoint for sending or receiving data at a single node in a computer network. Sockets can be of three types -

- *Datagram sockets*, also known as connectionless sockets, which use User Datagram Protocol (UDP).
- *Stream sockets*, also known as connection-oriented sockets, which use Transmission Control Protocol (TCP).
- *Raw sockets*, typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are made accessible to the application.

Socket call used in our application is :

```
socket(AF_PACKET , SOCK_RAW , htons(ETH_P_ALL))
```

1. `AF_PACKET` : This is used to send or receive an arbitrary ethernet frame or when we want to deal with the packets at protocol level.
2. `SOCK_RAW` : Used to capture raw packets so that the headers are available to the application.
3. `ETH_P_ALL` : Used to receive all protocols. All incoming packets of that protocol type will be passed to the packet socket before they are passed to the protocols implemented in the kernel.

First the IP header part of the packet is captured (excluding the ethernet header) and stored in a variable of type *struct iphdr*. Then using its protocol field value, its protocol (TCP, UDP or some other protocol) is obtained and the packet is further processed accordingly. The various parameters at different levels captured are :

8.2 Physical Layer - Ethernet

The raw packet captured in the buffer is typecasted to *struct ethhdr*. This structure has all the header fields as its attributes. It is defined in the header file net/ethernet.h.

- *Destination Address* : MAC address of the destination host.
- *Source Address* : MAC address of the source host.
- *Protocol Number* : Protocol ID

8.3 Network Layer - Internet Protocol

The size of the ethernet header is found using the size of the *struct ethhdr* used in Physical layer analysis. The raw packet captured in the buffer + the Ethernet header is typecasted to *struct iphdr*. This structure has all the IP header fields as its attributes. It is defined in the header file *netinet/ip.h*

- *IP Version* : 4/6
- *IP Header Length* : Number of 32-bit words forming the header
- *IP Type of Service* : Indicates quality of service needed from the network, usually 0.
- *IP Total Length* : Combined length of header and data (in bytes)
- *Identification* : 16-bit number which together with the source address uniquely identifies this packet.
- *TTL*: Time to Live for the packet
- *Protocol* : Indicates the type of transport packet being carried (1 = ICMP, 2 = IGMP, 6 = TCP, 17= UDP)
- *Checksum* : A 1's complement code inserted by the sender to check if the packet is received error-free.
- *Source IP* : IP address of the source
- *Destination IP* : IP address of the destination

8.4 Transport Layer - Transmission Control Protocol

The TCP header fields are captured and analyzed by first typecasting the captured packet to *struct ethhdr* and then to *struct iphdr*. The size of ethhdr reflects the size of ethernet header and then using the attribute *ihl* of the struct, IP header length is found. The Ethernet header+ IP header + captured packet combined together are typecasted to *struct tcphdr*. This structure is then further used to extract the various fields of the TCP header. The structures used for TCP are defined in the header file *netinet/tcp.h* .

- *Source Port* : Source port address
- *Destination Port* : Destination port address
- *Sequence number* : The sequence number of the first data octet in this segment (except when SYN is present). If SYN is

present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1

- *Acknowledgment Number* : If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent
- *Header length* : Length of TCP header
- *Urgent flag*: indicates that the Urgent pointer field is significant
- *Acknowledgement flag* : indicates that the Acknowledgment field is significant. All packets after the initial SYN packet sent by the client should have this flag set
- *Push flag* : Push function. Asks to push the buffered data to the receiving application
- *Reset flag* : Reset the connection
- *Synchronise flag* : Synchronize sequence numbers. Only the first packet sent from each end should have this flag set
- *Finish flag* : Last package from sender
- *Window* : The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.
- *Checksum* : A 1's complement code inserted by the sender to check if the packet is received error-free.
- *Urgent Pointer* : This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment.

8.5 Transport Layer - User Datagram Protocol

The procedure for UDP is almost similar to that used for TCP. The structures used for UDP are defined in the header file *netinet/tcp.h*.

- *Source Port* : Source port address
- *Destination Port* : Destination port address
- *UDP Length* :The number of bytes comprising the combined UDP header information and payload data
- *UDP Checksum* : A 1's complement code inserted by the sender to check if the packet is received error-free.

8.6 Application Layer - HyperText Transfer Protocol

HTTP packets are identified using source and destination port number usually 80 and 443 (HTTPS). Data payload is available using the buffer obtained from raw packet capture but in hexadecimal notation which is further converted to plain text for better readability. HTTP generally uses TCP as the underlying transport layer protocol.

8.7 Application Layer - Domain Name System

DNS packets are identified using source and destination port number usually 53. Multicast DNS which uses port 5353 is clubbed together with DNS. UDP is generally used as the underlying transport layer protocol for DNS.

The details of the captured packets are written in a log file while the basic details about the packets like MAC addresses, IP addresses and port numbers (of both the source as well as the destination) can be displayed to the user in case of packet filtering. This content is presented in a user-friendly manner. For around 500 packets captured the log file size is 200 KB.

9 Major learnings

- Packet capture using raw sockets which allows direct sending and receiving of Internet Protocol packets without any protocol-specific transport layer formatting.
- Working with multiple interfaces to which the local machine is connected to and binding the socket to a particular interface for specific packet capture.
- Interpreting various protocol headers at multiple layers and extracting useful information from them.
- Plotting and display of graphs of data extracted from the packets using gnuplot.

10 Conclusions

- The tool made by the team is a packet capturing and analysis utility for educational purposes.
- The tool enables easy visualization of network traffic and packet sizes through graphs.
- The tool can be used to get a rough estimate of the types of protocols used by applications at the transport level.
- The tool can be used to understand the basic network protocol hierarchy.
- The tool is useful in client server applications as conversations between hosts can be captured.

11 Possible future extensions

- The tool in its present form can analyze limited number of protocols at each layer. This can be further extended to a larger protocol stack.
- The graphical user interface of the tool is at a premature terminal level and can be made more attractive and informative.
- The tool offers few options for packet analysis. This can be extended further for detailed network and statistical analysis to extract more information.