



Project Report

IPv4 NETWORK MONITORING TOOL (Traffic Analyzer)

Submitted by:-

------------------------------------------------------------------------

Jatin Narula

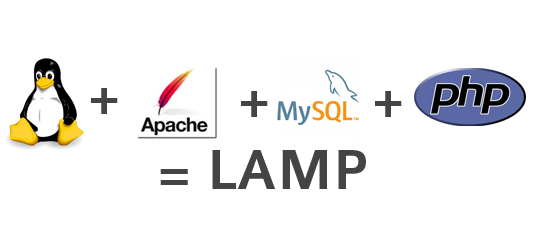
Btech. 2nd Year

Computer Science and Engineering

Motilal Nehru National Institute of Technology

Allahabad

**ACKNOWLEDGEMENT**



-----------------------------------------------------------------

Summer project provides an individual opportunity to explore one’s knowledge, apart from the usual curriculum.

I am grateful to **Mr. Navpreet Singh**, chief Computer Engineer, Computer Centre IIT Kanpur, for providing me the opportunity to work on this project and for his continuous support and guidance.

**CERTIFICATE**

This is to certify that JATIN NARULA of Motilal Nehru National Institute of Technology, Allahabad has worked for his summer internship project on **NETWORK MONITORING TOOL** under my supervision at the end of 2nd year of B. Tech in Computer Science and Engineering.



**Mr. Navpreet Singh**

**Chief Computer Engineer**

**Computer Center**

**Indian Institute of Technology, Kanpur**

Acknowledgement

It has indeed been a great privilege for me to have **Mr. Navpreet Singh, Chief Computer Engineer, Computer Centre, IIT Kanpur,** as my mentor for this project. I take this opportunity to express my utmost gratitude to him. I am indebted to him for his timely and valuable advice.

I also take the opportunity to express my gratitude and indebtedness to **Mr. Saurabh Malhotra** for his inestimable help, regular supervision of my work and unfailing guidance.

Jatin Narula

**Btech (2nd year)**

**Computer Science and Engineering**

**Motilal Nehru National Institute of Technology**

**Allahabad**

CONTENTS

* OBJECTIVE AND PURPOSE
* INTRODUCTION
* LAMP ARCHITECTURE
* L => LINUX
* A => APACHE
* M => MySQL
* P => PHP
* METHODOLOGY
* PACKET SNIFFING
* DATA PROCESSING
* DATA STORAGE
* IDENTIFYING **TOP TALKERS**
* USER INTERFACE
* ABOUT TCPDUMP COMMAND

**OBJECTIVE AND PURPOSE**

The objective of this project is to develop an IPv4 network monitoring tool. Also known as a packet analyzer, protocol analyzer or packet sniffer, it is a computer application that can intercept and log traffic passing over a digital network or part of a network.

Packet sniffers are used by network administrators to manage the traffic, load and monitor any abnormal usage. It also helps in maintaining overall performance of the network.

As data streams flow across the network, the sniffer captures packets using **TCPDUMP** and, if needed, decodes the packet's raw data, showing the values of various fields in the packet, and analyzes its content according to the appropriate logical operator or other specifications. When traffic is captured, either the entire contents of packets can be recorded, or the headers can be recorded without recording the total content of the packet.

**INTRODUCTION**

------------------------------------------------------------------

Packet sniffer is used by network administrators as a tool to log and monitor data flowing in and out the network. The program runs locally on the network and helps administrator to check for congestion in the network and the root cause for it. It can be useful in troubleshooting packet loss and latency.

The program has been designed for:

* TCP (Transmission Control Protocol)
* UDP (User Datagram Protocol)
* ARP (Address Resolution Protocol)

It traces:

* Timestamp
* Source MAC
* Destination MAC
* Source IP
* Destination IP
* source port
* destination port
* Packet length.

**LAMP ARCHITECTURE**

IPv4 Network Monitoring Tool has been designed on the LAMP architecture platform.

The acronym **LAMP** refers to the first letters of the four principal components to build a viable general purpose web server.

* **Linux** ,the operating system
* **Apache**, the HTTP web Server
* **MySQL**, database management system
* **PHP**, the scripting languages

Though the original authors of these programs did not design them all to work specifically with each other, the development philosophy and tool sets are shared and were developed in close conjunction. The software combination has become popular because it is free of cost, open-source, and therefore easily adaptable. When used together, they support web application servers.

**LINUX**



Linux is a Unix-like computer operating system assembled under the model of free and open source software development and distribution. The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991, by Linus Torvalds. Since the C compiler that builds Linux and the main supporting user space system tools and libraries originated in the GNU Project, initiated in 1983 by Richard Stallman, the Free Software Foundation prefers the name GNU/Linux.

Linux was originally developed as a free operating system for Intel x86-based personal computers. It has since been ported to more computer hardware platforms than any other operating system. It is a leading operating system on servers and other big iron systems such as mainframe computers and supercomputers: more than 90% of today's 500 fastest supercomputers run some variant of Linux, including the 10 fastest. Linux also runs on embedded systems (devices where the operating system is typically built into the firmware and highly tailored to the system) such as mobile phones, tablet computers, network routers, building automation controls, televisions and video game consoles; the Android system in wide use on mobile devices is built on the Linux kernel.

**APACHE**



The Apache HTTP Server commonly referred to as Apache, is a web server software program notable for playing a key role in the initial growth of the World Wide Web. In 2009, it became the first web server software to surpass the 100 million website milestone. Apache was the first viable alternative to the Netscape Communications Corporation web server (currently named Oracle iPlanet Web Server). Typically Apache is run on a Unix-like operating system, and was developed for use on Linux.

Apache has been the most popular HTTP server software in use. As of June 2013, Apache was estimated to serve 54.2% of all active websites and 53.3% of the top servers across all domains.

Apache supports a variety of features, many implemented as compiled modules which extend the core functionality. These can range from server-side programming language support to authentication schemes. Some common language interfaces support Perl, Python, Tcl, and PHP. Popular authentication modules include mod\_access, mod\_auth, mod\_digest, and mod\_auth\_digest, the successor to mod\_digest. A sample of other features include Secure Sockets Layer and Transport Layer Security support (mod\_ssl), a proxy module (mod\_proxy), a URL rewriter (mod\_rewrite), custom log files (mod\_log\_config), and filtering support (mod\_include and mod\_ext\_filter).

**MySQL**

MySQL is an open source relational database management system (RDBMS) that runs as a server providing multi-user access to a number of databases. The SQL phrase stands for Structured Query Language.

The MySQL development project has made its source code available under the terms of the GNU General Public License, as well as under a variety of proprietary agreements. MySQL was owned and sponsored by a single for-profit firm, the Swedish company MySQL AB, now owned by Oracle Corporation. MySQL is a popular choice of database for use in web applications, and is a central component of the widely used LAMP open source web application software stack (and other 'AMP' stacks). LAMP is an acronym for "Linux, Apache, MySQL, Perl/PHP/Python." Free-software-open source projects that require a full-featured database management system often use MySQL.

For commercial use, several paid editions are available, and offer additional functionality. Applications which use MySQL databases include: TYPO3, MODx, Joomla, WordPress, phpBB, MyBB, Drupal and other software. MySQL is also used in many high-profile, large-scale websites, including Wikipedia, Google (though not for searches), Facebook, Twitter, Flickr, and YouTube.

**PHP**

PHP is a server-side scripting language designed for web development but also used as a general purpose programming language. PHP is now installed on more than 244 million websites and 2.1 million web servers. Originally created by Rasmus Lerdorf in 1995, the reference implementation of PHP is now produced by The PHP Group. While PHP originally stood for Personal Home Page, it now stands for PHP: Hypertext Preprocessor.

PHP code is interpreted by a web server with a PHP processor module which generates the resulting web page: PHP commands can be embedded directly into an HTML source document rather than calling an external file to process data. It has also evolved to include a command-line interface capability and can be used in standalone graphical applications.

The LAMP architecture has become popular in the web industry as a way of deploying web applications. PHP is commonly used as the P in this bundle alongside Linux, Apache and MySQL, although the P may also refer to Python, Perl, or some mix of the three. Similar packages are also available for Windows and OS X, then called WAMP and MAMP, with the first letter standing for the respective operating system. Although both PHP and Apache are provided as part of the Mac OS X base install, users of these packages seek a simpler installation mechanism that can be more easily kept up to date.

METHODOLOGY

----------------------------------------------------------------------

The IPv4 Network Monitoring Tool has been designed using the LAMP platform with an aim to be user-friendly. I have designed a user interface in HTML which uses PHP to run the shell script which captures packets using the ‘TCPDUMP” command. The script then processes the information extracted and modifies it into the desired format. This processed data is stored into a database using MySQL.

The following steps were involved in the designing process:

* Packet Sniffing
* Data Processing
* Data Storage
* User Interface

**PACKET SNIFFING**:

First and most basic step is sniffing packets passing throughout the network. For this I have used shell command “tcpdump”. TCPDUMP is a common packet sniffer that runs under the command line. It allows the user to intercept and display packets being transmitted or received over a network to which the computer is attached. TCPDUMP can provide very detailed information about any network conversation that runs across the wires. The application uses very little overhead, since it's a non-graphical interface. The tcpdump command which has been used is:

tcpdump -ne -tttt -c 1000

**DATA PROCCESSING**:

Second step is to process the captured packets so that they can be sent to database. A shell script is written in the vim text editor. This shell script is the same script that also contains the **tcpdump** command in the beginning. As stated earlier, this script is invoked at runtime when the user signs in to use the tool.

Mainly the following LINUX commands have been used:

* **grep:** it searches for the specified phrase (e.g. ‘ARP’) in the output file provided by the tcpdump command. This output is stored into a new file.
* **cut:** the required fields (e.g. Source IP) are cut and the unnecessary symbols and info is clipped away.
* **sed:** lines which have unwanted output are removed using this command.
* **Paste:** to paste all the cut fields from their respective files into one so that they can be loaded directly into the database.
* **rm:** to permanently remove a file.

**DATA STORAGE:**

After the data is processed and modified into its desirable form it has to be loaded into a database which has different tables for different packet protocols. The following commands are employed:

* CREATE DATABASE<database name>: it creates the database which will store all the processed data.
* USE<database name>: it selects the database.
* CREATE TABLE <table name (table attributes)>: to create the tables for all protocols.
* LOAD DATA LOCAL INFILE <filename> INTO TABLE <table name>: to load the final data of the file into the respective table in the database.

These commands are stored in the same shell script that is called when the user signs in to use the network monitoring tool.

Another shell script is written to delete all the packets information stored in files and MySQL database. This script is invoked when the user logs out after using the services of the tool. Thus no outsider other than the administrator can access the captured packets information without signing in. This script is simply meant for security purpose.

IDENTIFYING TOP TALKERS:

The SOURCE I.P. ADDRESSES of all the packets that are captured are pasted into a single file and processed in a form which gives the number of times each I.P. address has occurred during the capturing process.

The commands used are:

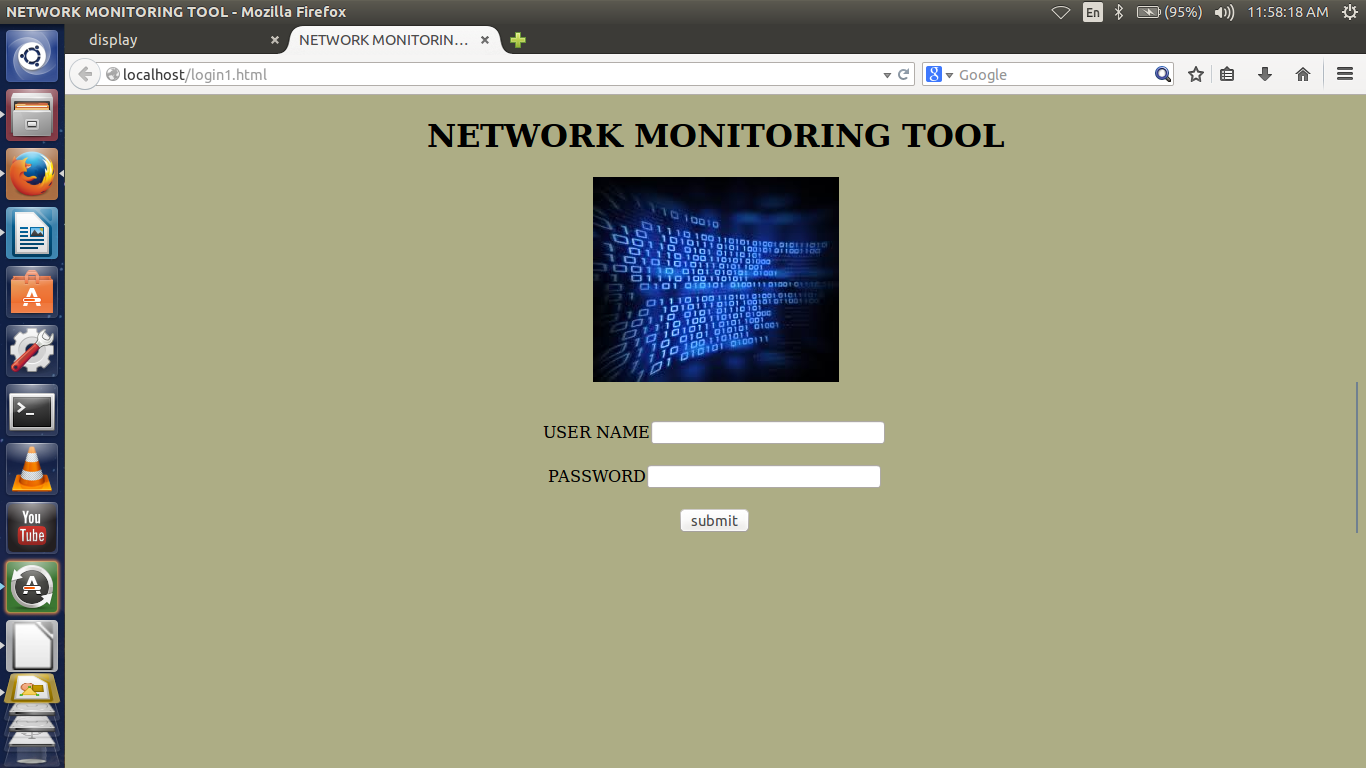
* CAT: for pasting the source I.P. of all protocols into a single file.
* SORT: for sorting the file.
* UNIQ: for removing duplicate entries and counting the number for occurrences.
* HEAD: for printing the top 5 lines of the sorted file.

The final result is loaded into the database and a separate table is created for the top-talkers.

**USER INTERFACE**:

This is next and very essential part of the tool. It is implemented using HTML and PHP.

This is the homepage, which has a link to login page, registration page, help page and about the traffic analyzer. It has a login link for existing users and a Registration link for new users. After clicking login one can go to login page and similarly to the registration page.



<html>

<head>

<style>

.ram{border-right:2px solid #708090; }

</style>

<title> NETWORK MONITORING TOOL</title>

</head>

<body style="background-color:#ADAD85">

<div align="center">

<h1 style="color:#000000">NETWORK MONITORING TOOL</h1>

<img src="[images3.jpeg](view-source:http://localhost/images3.jpeg)">

<div class="ram">

<form action="[firstphp.php](view-source:http://localhost/firstphp.php)" method="POST">

<br>

<br>

USER NAME<input type="text" name="username" >

<br>

<br>

PASSWORD<input type="password" name="password"><br><br>

<input type="submit" value="submit">

</form>

</div>

</div>

</body>

</html>

While Logging in the inputted email and password is matched with one present in the database using the PHP code which is given as:-

<?php

$con=mysql\_connect('localhost','root');

if(!$con)

{

die("connection failed:".mysql\_error());

}

echo "connected successfully";

mysql\_select\_db('firstphp');

echo $u=$\_POST["username"];

echo $p=$\_POST["password"];

$query="select name,password from rahul";

$result=mysql\_query($query,$con);

while($row=mysql\_fetch\_array($result))

{

if($row['name']==$u && $row['password']==$p)

{

echo "logged in...<br>";

shell\_exec ('/var/www/html/nmt.sh');

header( 'Location: design.html');

$variable = system(" ");

exit();

}

}

echo "try again...";

echo "<br>";

mysql\_close($con);

?>

If the credentials entered match with the one registered earlier, present in the database then the shell script runs in the background and fetches the network packets.

The Shell Script written is shown below:-

#!/bin/bash

/usr/sbin/tcpdump -ne -c 1000 > info

cat info | grep IP > IP

cat info | grep UDP > UDP

cat info | grep ARP > arp

#/\*ARP cut commands \*/

cut -d " " -f1 arp > arptime

cut -d " " -f2 arp > arpsmac

cut -d " " -f4 arp > arpdmac

cut -d " " -f9 arp > arplen

cut -d " " -f12 arp > arpsip

cut -d " " -f14 arp > arpdip

cut -d "," -f1 arpdmac > arpdmac1

cut -d ":" -f1 arplen > arplen1

cut -d "," -f1 arpdip > arpdip1

paste arptime arpsmac arpdmac1 arplen1 arpsip arpdip1 > arp

#/\* IP cut commands \*/

cut -d " " -f1 IP > iptime

cut -d " " -f2 IP > ipsmac

cut -d " " -f4 IP > ipdmac

cut -d " " -f9 IP > iplen

cut -d " " -f10 IP > ipsip

cut -d " " -f12 IP > ipdip

cut -d "," -f1 ipdmac > ipdmac1

cut -d ":" -f1 iplen > iplen1

cut -d "," -f1 ipdip > ipdip1

cut -d "." -f5 ipsip > ipsport

cut -d "." -f5 ipdip > ipdport

cut -d ":" -f1 ipdport > ipdport1

cut -d "." -f1,2,3,4 ipsip > ipsip1

cut -d "." -f1,2,3,4 ipdip1 > ipdip2

paste iptime ipsmac ipdmac1 iplen1 ipsip1 ipdip2 ipsport ipdport1 > ip

#/\* UDP cut commands \*/

cut -d " " -f1 UDP > udptime

cut -d " " -f2 UDP > udpsmac

cut -d " " -f4 UDP > udpdmac

cut -d " " -f9 UDP > udplen

cut -d " " -f10 UDP > udpsip

cut -d " " -f12 UDP > udpdip

cut -d "," -f1 udpdmac > udpdmac1

cut -d ":" -f1 udplen > udplen1

cut -d "," -f1 udpdip > udpdip1

cut -d "." -f5 udpsip > udpsport

cut -d "." -f5 udpdip > udpdport

cut -d ":" -f1 udpdport > udpdport1

cut -d "." -f1,2,3,4 udpsip > udpsip1

paste udptime udpsmac udpdmac1 udplen1 udpsip1 udpdip1 udpsport udpdport > udp

cat ip arp udp > all

cat arpsip ipdip2 udpsip1 > top

sort -nr top > top1

uniq -c top1 > top2

sort -nr top2 > top3

head -n 05 top3 > top4

#/\* SQL DATABASE \*/

mysql -u root -e"

drop database if exists dump;

create database dump;

use dump;

drop table if exists arptable;

drop table if exists iptable;

drop table if exists udptable;

drop table if exists top;

drop table if exists alltable;

create table arptable(atimestamp varchar(20),asource\_mac varchar(25),ades\_mac varchar(25),alength varchar(20),asource\_ip varchar(25),ades\_ip varchar(25));

create table iptable(timestamp varchar(20),source\_mac varchar(25),des\_mac varchar(25),length varchar(20),source\_ip varchar(25),des\_ip varchar(25),source\_port varchar(20), des\_port varchar(20));

create table udptable(utimestamp varchar(20),usource\_mac varchar(25),udes\_mac varchar(25),ulength varchar(20),usource\_ip varchar(25),udes\_ip varchar(25),usource\_port varchar(20),udes\_port varchar(20));

create table top(total varchar(20), source\_ip varchar(30));

create table alltable(alltimestamp varchar(20),allsource\_mac varchar(25),alldes\_mac varchar(25),alllength varchar(20),allsource\_ip varchar(25),alldes\_ip varchar(25),allsource\_port varchar(20),alldes\_port varchar(20));

LOAD DATA LOCAL INFILE 'ip' INTO TABLE iptable;

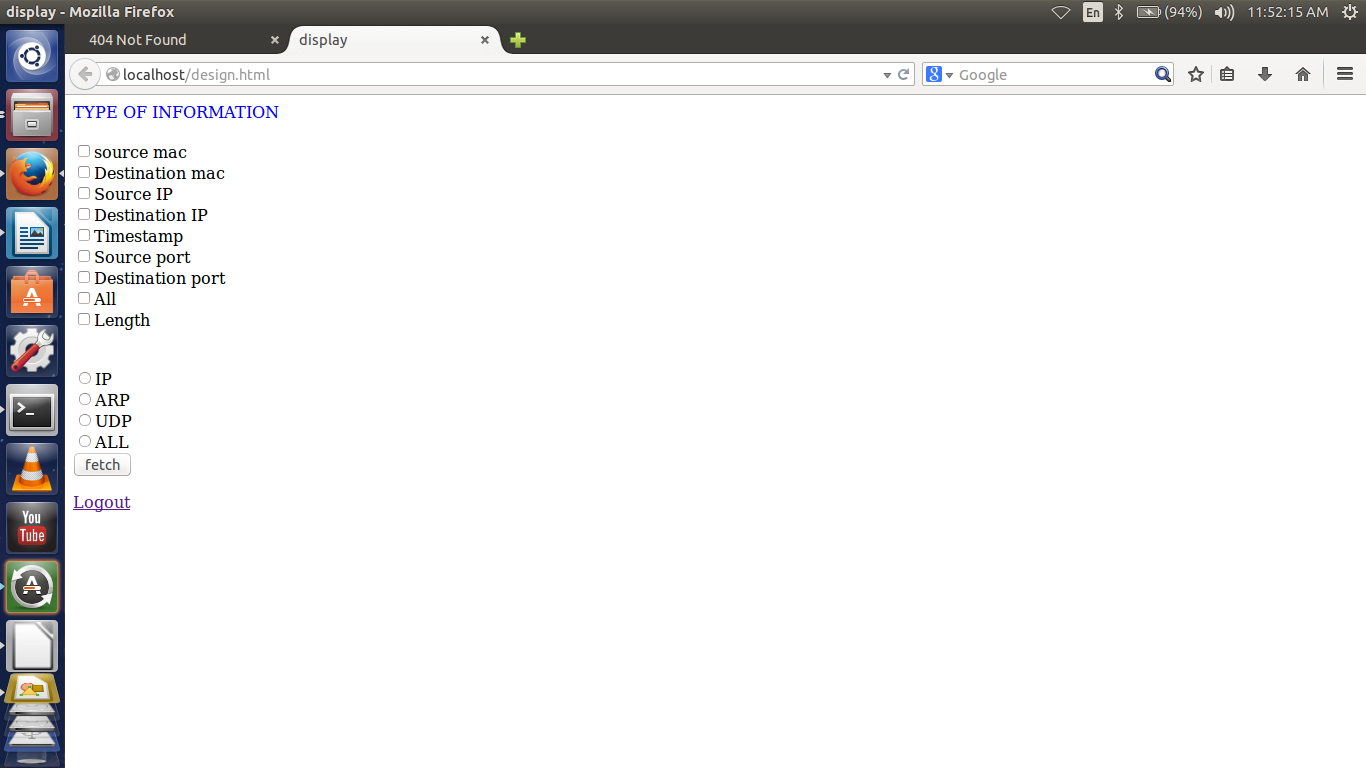
LOAD DATA LOCAL INFILE 'udp' INTO TABLE udptable;

LOAD DATA LOCAL INFILE 'arp' INTO TABLE arptable;

LOAD DATA LOCAL INFILE 'top4' INTO TABLE top;

LOAD DATA LOCAL INFILE 'all' INTO TABLE alltable;";

After the shell script runs, the user is redirected to the Actual Tool page where he has to choose according to his desire as to what kind of service he wants, whether he wants to analyze the ipv4 traffic in the network .



<html>

<head>

<title>display</title>

</head>

<body>

<font color="blue">

TYPE OF INFORMATION<br><br>

</font>

<form action="display.php" method="POST">

<input type="checkbox" name="ch" value="source\_m">source mac<br>

<input type="checkbox" name="ch" value="des\_mac">Destination mac<br>

<input type="checkbox" name="ch" value="source\_ip">Source IP<br>

<input type="checkbox" name="ch" value="des\_ip">Destination IP<br>

<input type="checkbox" name="ch" value="timestamp1">Timestamp <br>

<input type="checkbox" name="ch" value="source\_port">Source port<br>

<input type="checkbox" name="ch" value="des\_port">Destination port<br>

<input type="checkbox" name="ch" value="all">All<br>

<input type="checkbox" name="ch" value="length">Length<br>

<br><br>

<input type="radio" name="packet" value="IP">IP<br>

<input type="radio" name="packet" value="ARP">ARP<br>

<input type="radio" name="packet" value="UDP">UDP<br>

<input type="radio" name="packet" value="ALL">ALL<br>

<input type="submit" value="fetch">

<br>

</form>

<div align="left">

<a href= "login.html">Logout</a>

</div>

</body>

</html>

This page allows the user to choose the protocol type whose packets he wants the tool to display (ARP, UDP, TCP). Next, the user has to choose which fields will be displayed (one or more options can be selected out of TIME STAMP, SOURCE IP, DESTINATION IP, SOURCE MAC, DESTINATION MAC, LENGTH, SOURCE PORT, DESTIANTION PORT).

On clicking on the fetch button the results are fetched from the database and displayed using PHP.

The PHP code to display the chosen fields is given as:-

<html>

<head>

<style>

table, th, tr {

border: 1px solid black;

background-color:lightblue

}

body {background-color:brown}

</style>

</head>

</html>

<?php

$con=mysql\_connect("localhost","root","");

mysql\_select\_db("dump",$con);

$pac=$\_POST['packet'];

$info=$\_POST['ch'];

if($pac=="IP")

{

if($info=="source\_m")

{

echo "<table>";

echo "<tr><th>Source\_mac</th></tr>";

$result=mysql\_query("select source\_mac from iptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['source\_mac'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_mac")

{

echo "<table>";

echo "<tr><th>des\_mac</th></tr>";

$result=mysql\_query("select des\_mac from iptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['des\_mac'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="source\_ip")

{

echo "<table>";

echo "<tr><th>source\_ip</th></tr>";

$result=mysql\_query("select source\_ip from iptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['source\_ip'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_ip")

{

echo "<table>";

echo "<tr><th>des\_ip</th></tr>";

$result=mysql\_query("select des\_ip from iptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['des\_ip'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="source\_port")

{

echo "<table>";

echo "<tr><th>Source\_port</th></tr>";

$result=mysql\_query("select source\_port from iptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['source\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_port")

{

echo "<table>";

echo "<tr><th>des\_port</th></tr>";

$result=mysql\_query("select des\_port from iptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['des\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="all")

{

echo "<table>";

echo "<tr><th>Source\_mac</th><th>des\_mac</th><th>source\_ip</th><th>des\_ip</th><th>timestamp</th><th>length</th><th>source\_port</th><th>des\_port</th></tr>";

$result=mysql\_query("select \* from iptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['source\_mac'];

echo "</th>";

echo "<th>";

echo $r['des\_mac'];

echo "</th>";

echo "<th>";

echo $r['source\_ip'];

echo "</th>";

echo "<th>";

echo $r['des\_ip'];

echo "</th>";

echo "<th>";

echo $r['timestamp'];

echo "</th>";

echo "<th>";

echo $r['length'];

echo "</th>";

echo "<th>";

echo $r['source\_port'];

echo "</th>";

echo "<th>";

echo $r['des\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="length")

{

echo "<table>";

echo "<tr><th>length</th></tr>";

$result=mysql\_query("select length from iptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['length'];

echo "</th></tr>";

}

echo "</table>";

}

}

if($pac=="ARP")

{

if($info=="source\_m")

{

echo "<table>";

echo "<tr><th>source\_mac</th></tr>";

$result=mysql\_query("select asource\_mac from arptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['asource\_mac'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_mac")

{

echo "<table>";

echo "<tr><th>des\_mac</th></tr>";

$result=mysql\_query("select ades\_mac from arptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['ades\_mac'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="source\_ip")

{

echo "<table>";

echo "<tr><th>source\_ip</th></tr>";

$result=mysql\_query("select asource\_ip from arptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['asource\_ip'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_ip")

{

echo "<table>";

echo "<tr><th>des\_ip</th></tr>";

$result=mysql\_query("select ades\_ip from arptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['ades\_ip'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="all")

{

echo "<table>";

echo "<tr><th>Source\_mac</th><th>des\_mac</th><th>source\_ip</th><th>des\_ip</th><th>timestamp</th><th>length</th></tr>";

$result=mysql\_query("select \* from arptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['asource\_mac'];

echo "</th>";

echo "<th>";

echo $r['ades\_mac'];

echo "</th>";

echo "<th>";

echo $r['asource\_ip'];

echo "</th>";

echo "<th>";

echo $r['ades\_ip'];

echo "</th>";

echo "<th>";

echo $r['atimestamp'];

echo "</th>";

echo "<th>";

echo $r['alength'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="length")

{

echo "<table>";

echo "<tr><th>length</th></tr>";

$result=mysql\_query("select alength from arptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['alength'];

echo "</th></tr>";

}

echo "</table>";

}

}

if($pac=="UDP")

{

echo "<table>";

if($info=="source\_m")

{

echo "<tr><th>source\_mac</th></tr>";

$result=mysql\_query("select usource\_mac from udptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['usource\_mac'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_mac")

{

echo "<table>";

echo "<tr><th>des\_mac</th></tr>";

$result=mysql\_query("select udes\_mac from udptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['udes\_mac'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="source\_ip")

{

echo "<table>";

echo "<tr><th>source\_ip</th></tr>";

$result=mysql\_query("select usource\_ip from udptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['usource\_ip'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_ip")

{

echo "<table>";

echo "<tr><th>des\_ip</th></tr>";

$result=mysql\_query("select udes\_ip from udptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['udes\_ip'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="source\_port")

{

echo "<table>";

echo "<tr><th>source\_port</th></tr>";

$result=mysql\_query("select usource\_port from udptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['usource\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_port")

{

echo "<table>";

echo "<tr><th>des\_port</th></tr>";

$result=mysql\_query("select udes\_port from udptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['udes\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="all")

{

echo "<table>";

echo "<tr><th>Source\_mac</th><th>des\_mac</th><th>source\_ip</th><th>des\_ip</th><th>timestamp</th><th>length</th><th>source\_port</th><th>des\_port</th></tr>";

$result=mysql\_query("select \* from udptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['usource\_mac'];

echo "</th>";

echo "<th>";

echo $r['udes\_mac'];

echo "</th>";

echo "<th>";

echo $r['usource\_ip'];

echo "</th>";

echo "<th>";

echo $r['udes\_ip'];

echo "</th>";

echo "<th>";

echo $r['utimestamp'];

echo "</th>";

echo "<th>";

echo $r['ulength'];

echo "</th>";

echo "<th>";

echo $r['usource\_port'];

echo "</th>";

echo "<th>";

echo $r['udes\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="length")

{

echo "<table>";

echo "<tr><th>length</th></tr>";

$result=mysql\_query("select ulength from udptable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['ulength'];

echo "</th></tr>";

}

echo "</table>";

}

}

if($pac=="ALL")

{

if($info=="source\_mac")

{

echo "<table>";

echo "<tr><th>source\_mac</th></tr>";

$result=mysql\_query("select allsource\_mac from alltable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['allsource\_mac'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_mac")

{

echo "<table>";

echo "<tr><th>des\_mac</th></tr>";

$result=mysql\_query("select alldes\_mac from alltable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['alldes\_mac'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="source\_ip")

{

echo "<table>";

echo "<tr><th>source\_ip</th></tr>";

$result=mysql\_query("select allsource\_ip from alltable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['allsource\_ip'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_ip")

{

echo "<table>";

echo "<tr><th>des\_ip</th></tr>";

$result=mysql\_query("select alldes\_ip from alltable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['alldes\_ip'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="source\_port")

{

echo "<table>";

echo "<tr><th>source\_port</th></tr>";

$result=mysql\_query("select allsource\_port from alltable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['allsource\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="des\_port")

{

echo "<table>";

echo "<tr><th>des\_port</th></tr>";

$result=mysql\_query("select alldes\_port from alltable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['alldes\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="all")

{

echo "<table>";

echo "<tr><th>Source\_mac</th><th>des\_mac</th><th>source\_ip</th><th>des\_ip</th><th>timestamp</th><th>length</th><th>source\_port</th><th>des\_port</th></tr>";

$result=mysql\_query("select \* from alltable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['allsource\_mac'];

echo "</th>";

echo "<th>";

echo $r['alldes\_mac'];

echo "</th>";

echo "<th>";

echo $r['allsource\_ip'];

echo "</th>";

echo "<th>";

echo $r['alldes\_ip'];

echo "</th>";

echo "<th>";

echo $r['alltimestamp'];

echo "</th>";

echo "<th>";

echo $r['alllength'];

echo "</th>";

echo "<th>";

echo $r['allsource\_port'];

echo "</th>";

echo "<th>";

echo $r['alldes\_port'];

echo "</th></tr>";

}

echo "</table>";

}

if($info=="length")

{

echo "<table>";

echo "<tr><th> length</th></tr>";

$result=mysql\_query("select alllength from alltable");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['alllength'];

echo "</th></tr>";

}

echo "</table>";

}

}

if($pac=="Source")

{

if($info)

{

echo "<table>";

echo "<tr><th> Source\_ip </th><th> Total</th></tr>";

$result=mysql\_query("select \* from top");

while($r=mysql\_fetch\_array($result))

{

echo "<tr><th>";

echo $r['source'];

echo "</th>";

echo "<th>";

echo $r['total'];

echo "</th></tr>";

}

echo "</table>";

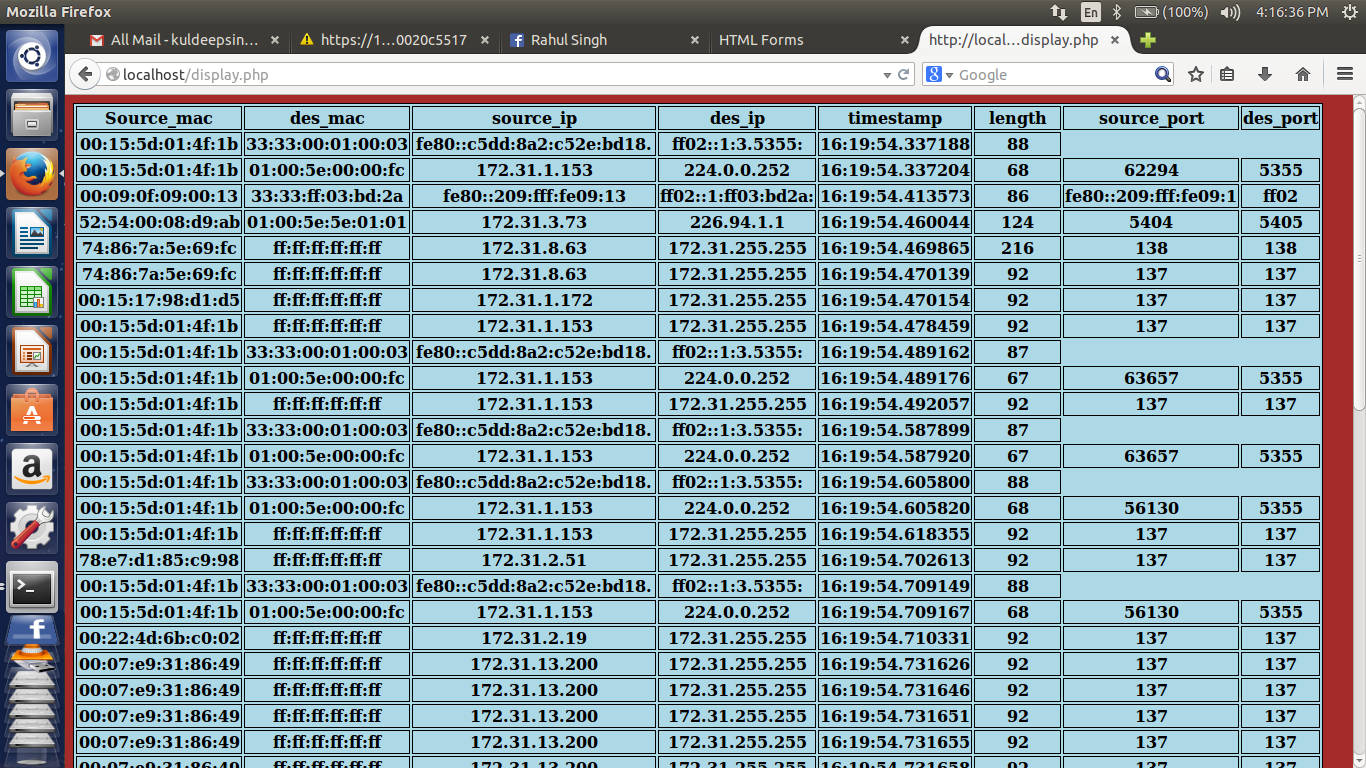
}

}

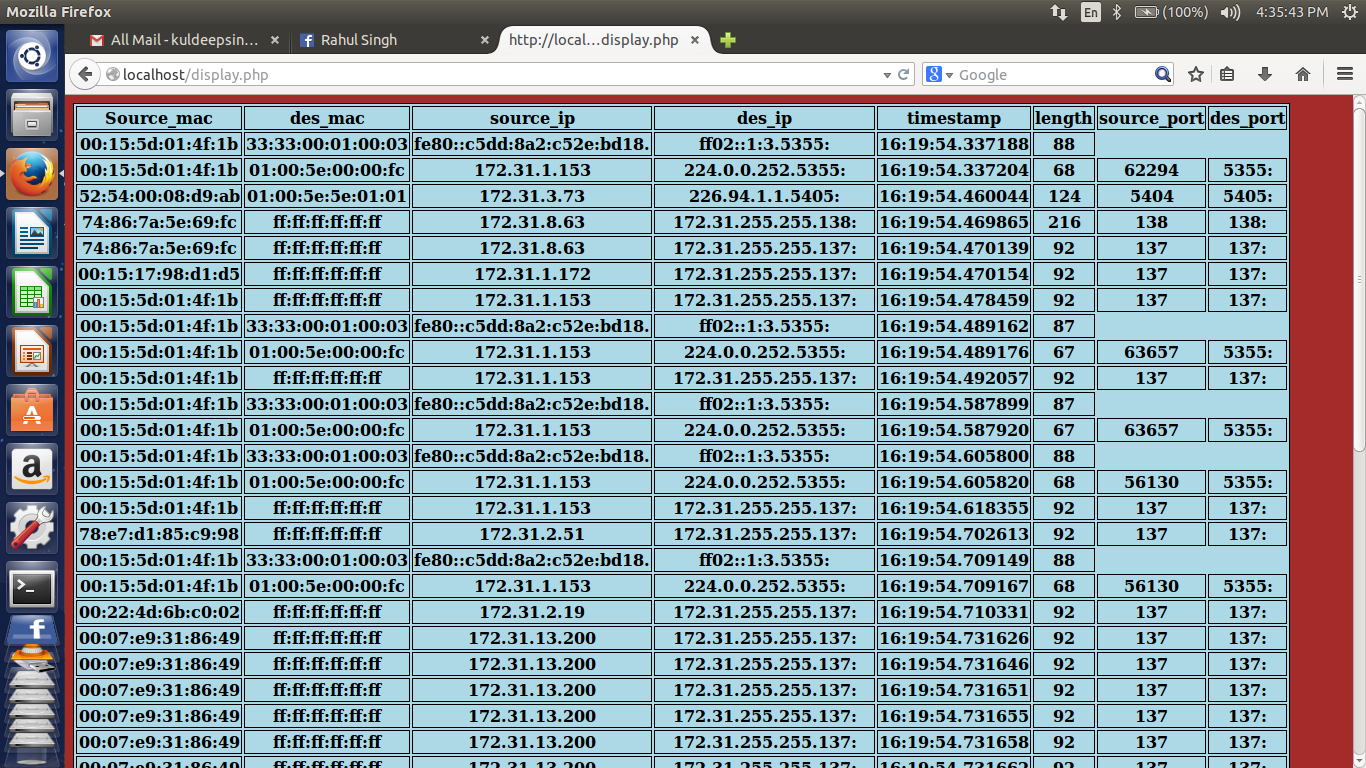
mysql\_close(con);

?>

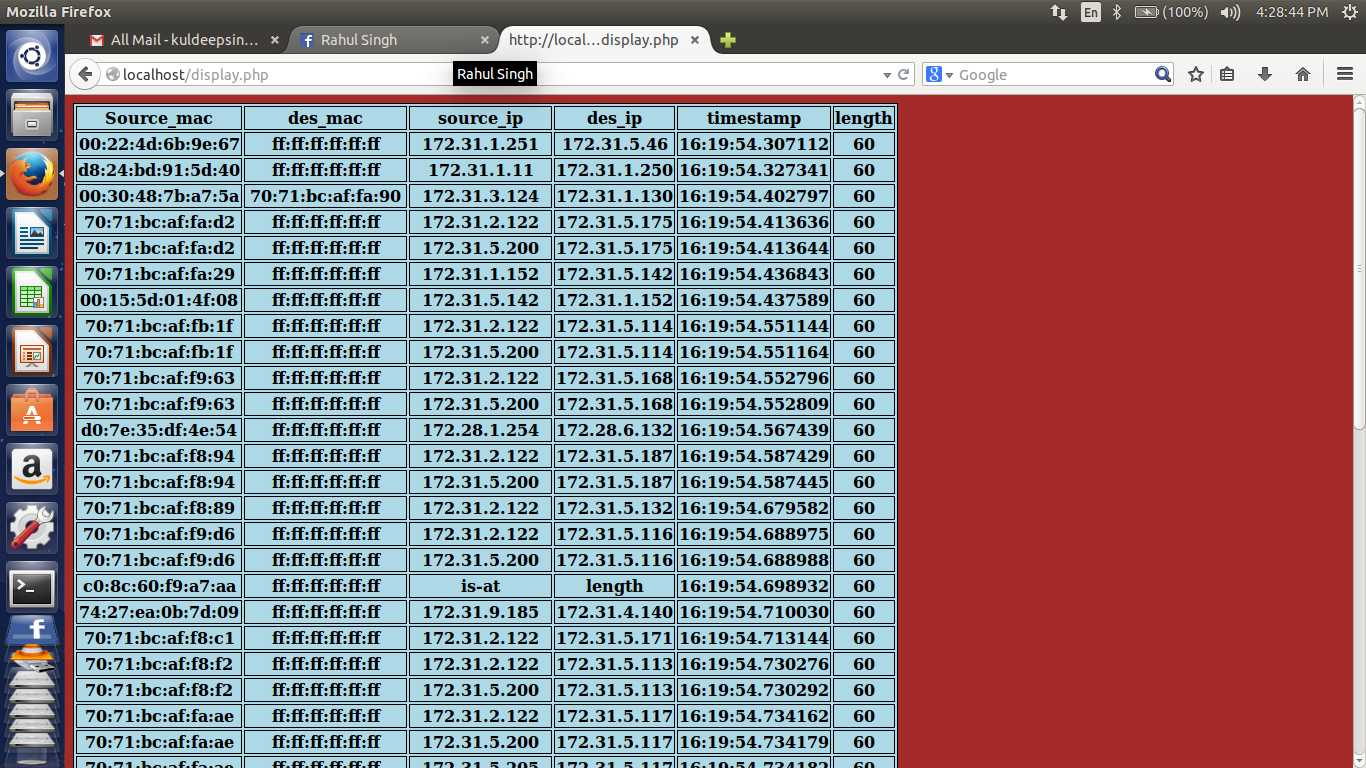
The Result for all fields of TCP is shown as:-



The result for all fields of UDP is :



The result for all fields of ARP is :



## 

## ABOUT THE TCPDUMP COMMAND

## -----------------------------------------------------------------

## TCPDUMP - DUMP TRAFFIC ON A NETWORK

**SYNTAX:**

**tcpdump** [ **-AbdDefhHIJKlLnNOpqRStuUvxX** ] [ **-B** *buffer\_size* ] [ **-c** *count* ]   
[ **-C** *file\_size* ] [ **-G** *rotate\_seconds* ] [ **-F** *file* ]   
[**-i** *interface* ] [ **-j** *tstamp\_type* ] [ **-m** *module* ] [ **-M** *secret* ]   
[ **-r** *file* ] [ **-V** *file* ] [ **-s** *snaplen* ] [ **-T** *type* ] [ **-w** *file* ]   
[ **-W** *filecount* ]   
[ **-E** *[spi@ipaddr](mailto:spi@ipaddr) algo:secret,...* ]   
[ **-y** *datalinktype* ] [ **-z** *postrotate-command* ] [ **-Z** *user* ]   
[ *expression* ] 

**DESCRIPTION**

*Tcpdump* prints out a description of the contents of packets on a network interface that match the boolean *expression*. It can also be run with the **-w** flag, which causes it to save the packet data to a file for later analysis, and/or with the **-r** flag, which causes it to read from a saved packet file rather than to read packets from a network interface. It can also be run with the **-V** flag, which causes it to read a list of saved packet files. In all cases, only packets that match *expression* will be processed by *tcpdump*.

*Tcpdump* will, if not run with the **-c** flag, continue capturing packets until it is interrupted by a SIGINT signal (generated, for example, by typing your interrupt character, typically control-C) or a SIGTERM signal (typically generated with the [**kill**](http://localhost/cgi-bin/man/man2html?1+kill)(1) command); if run with the **-c** flag, it will capture packets until it is interrupted by a SIGINT or SIGTERM signal or the specified number of packets have been processed.

When *tcpdump* finishes capturing packets, it will report counts of:

packets ``captured'' (this is the number of packets that *tcpdump* has received and processed);

packets ``received by filter'' (the meaning of this depends on the OS on which you're running *tcpdump*, and possibly on the way the OS was configured - if a filter was specified on the command line, on some OSes it counts packets regardless of whether they were matched by the filter expression and, even if they were matched by the filter expression, regardless of whether *tcpdump* has read and processed them yet, on other OSes it counts only packets that were matched by the filter expression regardless of whether *tcpdump* has read and processed them yet, and on other OSes it counts only packets that were matched by the filter expression and were processed by *tcpdump*);

packets ``dropped by kernel'' (this is the number of packets that were dropped, due to a lack of buffer space, by the packet capture mechanism in the OS on which *tcpdump* is running, if the OS reports that information to applications; if not, it will be reported as 0).

On platforms that support the SIGINFO signal, such as most BSDs (including Mac OS X) and Digital/Tru64 UNIX, it will report those counts when it receives a SIGINFO signal (generated, for example, by typing your ``status'' character, typically control-T, although on some platforms, such as Mac OS X, the ``status'' character is not set by default, so you must set it with **[stty](http://localhost/cgi-bin/man/man2html?1+stty)**(1) in order to use it) and will continue capturing packets.

Reading packets from a network interface may require that you have special privileges. Reading a saved packet file doesn't require special privileges. 

**OPTIONS**

**-A**

Print each packet (minus its link level header) in ASCII. Handy for capturing web pages.

**-b**

Print the AS number in BGP packets in ASDOT notation rather than ASPLAIN notation.

**-B**

Set the operating system capture buffer size to *buffer\_size*, in units of KiB (1024 bytes).

**-c**

Exit after receiving *count* packets.

**-C**

Before writing a raw packet to a savefile, check whether the file is currently larger than *file\_size* and, if so, close the current savefile and open a new one. Savefiles after the first savefile will have the name specified with the **-w** flag, with a number after it, starting at 1 and continuing upward. The units of *file\_size* are millions of bytes (1,000,000 bytes, not 1,048,576 bytes).

**-d**

Dump the compiled packet-matching code in a human readable form to standard output and stop.

**-dd**

Dump packet-matching code as a **C** program fragment.

**-ddd**

Dump packet-matching code as decimal numbers (preceded with a count).

**-D**

Print the list of the network interfaces available on the system and on which *tcpdump* can capture packets. For each network interface, a number and an interface name, possibly followed by a text description of the interface, is printed. The interface name or the number can be supplied to the **-i** flag to specify an interface on which to capture.

This can be useful on systems that don't have a command to list them (e.g., Windows systems, or UNIX systems lacking **ifconfig -a**); the number can be useful on Windows 2000 and later systems, where the interface name is a somewhat complex string.

The **-D** flag will not be supported if *tcpdump* was built with an older version of *libpcap* that lacks the **pcap\_findalldevs()** function.

**-e**

Print the link-level header on each dump line. This can be used, for example, to print MAC layer addresses for protocols such as Ethernet and IEEE 802.11.

**-E**

Use *[spi@ipaddr](mailto:spi@ipaddr) algo:secret* for decrypting IPsec ESP packets that are addressed to *addr* and contain Security Parameter Index value *spi*. This combination may be repeated with comma or newline separation.

Note that setting the secret for IPv4 ESP packets is supported at this time.

Algorithms may be **des-cbc**, **3des-cbc**, **blowfish-cbc**, **rc3-cbc**, **cast128-cbc**, or **none**. The default is **des-cbc**. The ability to decrypt packets is only present if *tcpdump* was compiled with cryptography enabled.

*secret* is the ASCII text for ESP secret key. If preceded by 0x, then a hex value will be read.

The option assumes RFC2406 ESP, not RFC1827 ESP. The option is only for debugging purposes, and the use of this option with a true `secret' key is discouraged. By presenting IPsec secret key onto command line you make it visible to others.

In addition to the above syntax, the syntax *file name* may be used to have tcpdump read the provided file in. The file is opened upon receiving the first ESP packet, so any special permissions that tcpdump may have been given should already have been given up.

**-f**

Print `foreign' IPv4 addresses numerically rather than symbolically (this option is intended to get around serious brain damage in Sun's NIS server --- usually it hangs forever translating non-local internet numbers).

The test for `foreign' IPv4 addresses is done using the IPv4 address and netmask of the interface on which capture is being done. If that address or netmask are not available, available, either because the interface on which capture is being done has no address or netmask or because the capture is being done on the Linux "any" interface, which can capture on more than one interface, this option will not work correctly.

**-F**

Use *file* as input for the filter expression. An additional expression given on the command line is ignored.

**-G**

If specified, rotates the dump file specified with the **-w** option every *rotate\_seconds* seconds. Savefiles will have the name specified by **-w** which should include a time format. If no time format is specified, each new file will overwrite the previous.

If used in conjunction with the **-C** option, filenames will take the form of `*file*<count>'.

**-h**

Print the tcpdump and libpcap version strings, print a usage message, and exit.

**-H**

Attempt to detect 802.11s draft mesh headers.

**-i**

Listen on *interface*. If unspecified, *tcpdump* searches the system interface list for the lowest numbered, configured up interface (excluding loopback). Ties are broken by choosing the earliest match.

On Linux systems with 2.2 or later kernels, an *interface* argument of ``any'' can be used to capture packets from all interfaces. Note that captures on the ``any'' device will not be done in promiscuous mode.

If the **-D** flag is supported, an interface number as printed by that flag can be used as the *interface* argument.

**-I**

Put the interface in "monitor mode"; this is supported only on IEEE 802.11 Wi-Fi interfaces, and supported only on some operating systems.

Note that in monitor mode the adapter might disassociate from the network with which it's associated, so that you will not be able to use any wireless networks with that adapter. This could prevent accessing files on a network server, or resolving host names or network addresses, if you are capturing in monitor mode and are not connected to another network with another adapter.

This flag will affect the output of the **-L** flag. If **-I** isn't specified, only those link-layer types available when not in monitor mode will be shown; if **-I** is specified, only those link-layer types available when in monitor mode will be shown.

**-j**

Set the time stamp type for the capture to *tstamp\_type*.

**-J**

List the supported time stamp types for the interface and exit. If the time stamp type cannot be set for the interface, no time stamp types are listed.

**-K**

Don't attempt to verify IP, TCP, or UDP checksums. This is useful for interfaces that perform some or all of the checksum calculation in hardware; otherwise, all outgoing TCP checksums will be flagged as bad.

**-l**

Make stdout line buffered. Useful if you want to see the data while capturing it. E.g.,

**tcpdump -l | tee dat**

or

**tcpdump -l >dat& tail -f dat**

Note that on Windows,``line buffered'' means ``unbuffered'', so that WinDump will write each character individually if **-l** is specified.

**-U** is similar to **-l** in its behavior, but it will cause output to be ``packet-buffered'', so that the output is written to stdout at the end of each packet rather than at the end of each line; this is buffered on all platforms, including Windows.

**-L**

List the known data link types for the interface, in the specified mode, and exit. The list of known data link types may be dependent on the specified mode; for example, on some platforms, a Wi-Fi interface might support one set of data link types when not in monitor mode (for example, it might support only fake Ethernet headers, or might support 802.11 headers but not support 802.11 headers with radio information) and another set of data link types when in monitor mode (for example, it might support 802.11 headers, or 802.11 headers with radio information, only in monitor mode).

**-m**

Load SMI MIB module definitions from file *module*. This option can be used several times to load several MIB modules into *tcpdump*.

**-M**

Use *secret* as a shared secret for validating the digests found in TCP segments with the TCP-MD5 option (RFC 2385), if present.

**-n**

Don't convert addresses (i.e., host addresses, port numbers, etc.) to names.

**-N**

Don't print domain name qualification of host names. E.g., if you give this flag then *tcpdump* will print ``nic'' instead of ``nic.ddn.mil''.

**-O**

Do not run the packet-matching code optimizer. This is useful only if you suspect a bug in the optimizer.

**-p**

*Don't* put the interface into promiscuous mode. Note that the interface might be in promiscuous mode for some other reason; hence, `-p' cannot be used as an abbreviation for `ether host {local-hw-addr} or ether broadcast'.

**-q**

Quick (quiet?) output. Print less protocol information so output lines are shorter.

**-R**

Assume ESP/AH packets to be based on old specification (RFC1825 to RFC1829). If specified, *tcpdump* will not print replay prevention field. Since there is no protocol version field in ESP/AH specification,*tcpdump* cannot deduce the version of ESP/AH protocol.

**-r**

Read packets from *file* (which was created with the **-w** option). Standard input is used if *file* is ``-''.

**-S**

Print absolute, rather than relative, TCP sequence numbers.

**-s**

Snarf *snaplen the* bytes of data from each packet rather than the default of 65535 bytes. Packets truncated because of a limited snapshot are indicated in the output with ``[|*proto*]'', where *proto* is the name of the protocol level at which the truncation has occurred. Note that taking larger snapshots both increases the amount of time it takes to process packets and, effectively, decreases the amount of packet buffering. This may cause packets to be lost. You should limit *snaplen* to the smallest number that will capture the protocol information you're interested in. Setting *snaplen* to 0 sets it to the default of 65535, for backwards compatibility with recent older versions of *tcpdump*.

**-T**

Force packets selected by "*expression*" to be interpreted the specified *type*. Currently known types are **aodv** (Ad-hoc On-demand Distance Vector protocol), **carp** (Common Address Redundancy Protocol), **cnfp**(Cisco NetFlow protocol), **pgm** (Pragmatic General Multicast), **pgm\_zmtp1** (ZMTP/1.0 inside PGM/EPGM), **radius** (RADIUS), **rpc** (Remote Procedure Call), **rtp** (Real-Time Applications protocol), **rtcp** (Real-Time Applications control protocol), **snmp** (Simple Network Management Protocol), **tftp** (Trivial File Transfer Protocol), **vat** (Visual Audio Tool), **wb** (distributed White Board), **zmtp1** (ZeroMQ Message Transport Protocol 1.0) and **vxlan** (Virtual eXtensible Local Area Network).

Note that the **pgm** type above affects UDP interpretation only, the native PGM is always recognised as IP protocol 113 regardless. UDP-encapsulated PGM is often called "EPGM" or "PGM/UDP".

Note that the **pgm\_zmtp1** type above affects interpretation of both native PGM and UDP at once. During the native PGM decoding the application data of an ODATA/RDATA packet would be decoded as a ZeroMQ datagram with ZMTP/1.0 frames. During the UDP decoding in addition to that any UDP packet would be treated as an encapsulated PGM packet.

**-t**

*Don't* print a timestamp on each dump line.

**-tt**

Print an unformatted timestamp on each dump line.

**-ttt**

Print a delta (micro-second resolution) between current and previous line on each dump line.

**-tttt**

Print a timestamp in default format proceeded by date on each dump line.

**-ttttt**

Print a delta (micro-second resolution) between current and first line on each dump line.

**-u**

Print undecoded NFS handles.

**-U**

If the **-w** option is not specified, make the printed packet output ``packet-buffered''; i.e., as the description of the contents of each packet is printed, it will be written to the standard output, rather than, when not writing to a terminal, being written only when the output buffer fills.

If the **-w** option is specified, make the saved raw packet output ``packet-buffered''; i.e., as each packet is saved, it will be written to the output file, rather than being written only when the output buffer fills.

The **-U** flag will not be supported if *tcpdump* was built with an older version of *libpcap* that lacks the **pcap\_dump\_flush()** function.

**-v**

When parsing and printing, produce (slightly more) verbose output. For example, the time to live, identification, total length and options in an IP packet are printed. Also enables additional packet integrity checks such as verifying the IP and ICMP header checksum.

When writing to a file with the **-w** option, report, every 10 seconds, the number of packets captured.

**-vv**

Even more verbose output. For example, additional fields are printed from NFS reply packets, and SMB packets are fully decoded.

**-vvv**

Even more verbose output. For example, telnet **SB** ... **SE** options are printed in full. With **-X** Telnet options are printed in hex as well.

**-V**

Read a list of filenames from *file*. Standard input is used if *file* is ``-''.

**-w**

Write the raw packets to *file* rather than parsing and printing them out. They can later be printed with the -r option. Standard output is used if *file* is ``-''.

This output will be buffered if written to a file or pipe, so a program reading from the file or pipe may not see packets for an arbitrary amount of time after they are received. Use the **-U** flag to cause packets to be written as soon as they are received.

The MIME type *application/vnd.tcpdump.pcap* has been registered with IANA for *pcap* files. The filename extension *.pcap* appears to be the most commonly used along with *.cap* and *.dmp*. *Tcpdump* itself doesn't check the extension when reading capture files and doesn't add an extension when writing them (it uses magic numbers in the file header instead). However, many operating systems and applications will use the extension if it is present and adding one (e.g. .pcap) is recommended.

**-W**

Used in conjunction with the **-C** option, this will limit the number of files created to the specified number, and begin overwriting files from the beginning, thus creating a 'rotating' buffer. In addition, it will name the files with enough leading 0s to support the maximum number of files, allowing them to sort correctly.

Used in conjunction with the **-G** option, this will limit the number of rotated dump files that get created, exiting with status 0 when reaching the limit. If used with **-C** as well, the behavior will result in cyclical files per timeslice.

**-x**

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet (minus its link level header) in hex. The smaller of the entire packet or *snaplen* bytes will be printed. Note that this is the entire link-layer packet, so for link layers that pad (e.g. Ethernet), the padding bytes will also be printed when the higher layer packet is shorter than the required padding.

**-xx**

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet, *including* its link level header, in hex.

**-X**

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet (minus its link level header) in hex and ASCII. This is very handy for analysing new protocols.

**-XX**

When parsing and printing, in addition to printing the headers of each packet, print the data of each packet, *including* its link level header, in hex and ASCII.

**-y**

Set the data link type to use while capturing packets to *datalinktype*.

**-z**

Used in conjunction with the **-C** or **-G** options, this will make *tcpdump* run " *command file* " where *file* is the savefile being closed after each rotation. For example, specifying **-z gzip** or **-z bzip2** will compress each savefile using gzip or bzip2.

Note that tcpdump will run the command in parallel to the capture, using the lowest priority so that this doesn't disturb the capture process.

And in case you would like to use a command that itself takes flags or different arguments, you can always write a shell script that will take the savefile name as the only argument, make the flags & arguments arrangements and execute the command that you want.

**-Z**

If *tcpdump* is running as root, after opening the capture device or input savefile, but before opening any savefiles for output, change the user ID to *user* and the group ID to the primary group of *user*.

This behavior can also be enabled by default at compile time.