Lab Session 1
Introduction
16 June 2008

The purpose of the lab is to introduce students to basic utilities in Unix that will be useful for the rest of the lab sessions.

1.1 Man Pages

Using man command with various options
$man <topic>

we can use command called apropos to find similar topics
$man –k <topic>
or
$apropos <topic>

Example:
$man –k close
or
$apropos close

This lists all the commands similar to close. i.e file close, connection close etc.

Sometimes a command can have different meanings based on the context. For example time will display the current time (and allows you to set the system time) or can be used to find the the execution time of a given program. Man pages are organized as sections and mentioning the section number will fetch the appropriate page.

$man <sec no> <topic>

Example:

$man 3 perror
perror is used to print a system's error message.

To see some common error messages and their corresponding error numbers look at the file /usr/include/asm_generic/errno.h

Some of the network related errors are numbers 91 to 112.

1.2 Useful System Calls

1.2.1 sleep ()

This command is used to put the process in the sleep state. For ex:- when receiver chokes it can send request to the server asking it to sleep.

1.2.2 signal()

This gives signal handler for error. For ex:-If there is any error (which is identified by the error number returned) you have to manually debug it. Instead of doing this you can write a code which will do the same, store it in a file, and whenever an error i.e a signal occurs, map it to the code and handle it.

Signals can be found in **/usr/include/asm_i386/signal.h**.

1.2.3 alarm()

This is used to generate an alarm signal when required for ex:- To simulate exponential backoff alarms.

1.3 Network Configuration

To check your network configuration use
$ /sbin/ifconfig

Sample output :

```
eth0      Link encap:Ethernet  HWaddr 00:16:76:9B:85:22
          inet addr:10.6.15.60  Bcast:10.6.15.255  Mask:255.255.255.0
          inet6 addr: fe80::216:76ff:fe9b:8522/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:131103 errors:76 dropped:0 overruns:0 frame:0
          TX packets:120220 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:97178409 (92.6 MiB)  TX bytes:86258716 (82.2 MiB)
          Interrupt:201 Base address:0xc800

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:680 errors:0 dropped:0 overruns:0 frame:0
          TX packets:680 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:52216 (50.9 KiB)  TX bytes:52216 (50.9 KiB)
```

The configuration commands that change any settings usually have to be executed at the shell prompt in the super user mode

$su

su stands for substitute user, but most of the time we substitute as a super user.

You can configure an ipaddress using the following command

ifconfig eth0 10.6.15.65 netmask 255.255.255.0

This creates an interface whose name is eth0 {eth – Ethernet, 0 – this is the first interface} and configures it with the address 10.6.15.65.

Add a default gateway to the host using the route command

#route add -net default gateway 10.6.15.254

Check the route added using the route command

#route

Observe the output.

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.6.15.0       *               255.255.255.0   U     0      0        0 eth0
default         deer.cs.iitm.er 0.0.0.0         UG    0      0        0 eth0
```

10.6.15.0 means 10.6.15.1 to 10.6.15.254 (10.6.15.0 and 10.6.15.255 are reserved)

Gateway * means interface does not need any gateway to send the packets, it knows the path by itself.

This happens usually when all the systems are connected as a broadcast medium in the same subnet.

Lab Session 2
Configuring Subnets and LANs
17 June 2008

Ping is a command used to send packets to the specified destination, these packets return back to the sent station.

$ping 10.6.15.82

```
PING 10.6.15.82 (10.6.15.82) 56(84) bytes of data.
64 bytes from 10.6.15.82: icmp_seq=1 ttl=64 time=2.21 ms
64 bytes from 10.6.15.82: icmp_seq=2 ttl=64 time=0.264 ms
64 bytes from 10.6.15.82: icmp_seq=3 ttl=64 time=0.286 ms
64 bytes from 10.6.15.82: icmp_seq=4 ttl=64 time=0.260 ms
64 bytes from 10.6.15.82: icmp_seq=5 ttl=64 time=0.290 ms

--- 10.6.15.82 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.260/0.662/2.210/0.774 ms
```

2.1 Demonstrating Different Subnets in the Same Ethernet LAN:

To demonstrate the concepts of subnet, First divide the students into say 3 groups and assign IP addresses of same subnet. If there are hosts lined up on 3 columns, assign IP addresses as follows:

Column1	Column2	Column3
192.168.1.21	192.168.1.11	192.168.1.1
192.168.1.22	192.168.1.12	192.168.1.2
192.168.1.23	192.168.1.13	192.168.1.3
...	...	...

This is done by each one of them configuring the ipaddress using ifconfig.
Ex:- ifconfig eth0 192.168.1.23 netmask 255.255.255.0

Suppose your IP is 192.168.1.23 and if you do the following:

$ping 192.168.1.1
$ping 192.168.1.11

both will be successful as they are in the same broadcast domain i.e subnet as that of the source.

Now change the IPaddress of each group forming a subnet as follows:-

Column1	Column2	Column3
192.168.3.21	192.168.2.11	192.168.1.1
192.168.3.22	192.168.2.12	192.168.1.2
192.168.3.23	192.168.2.13	192.168.1.3

Now suppose your IP is 192.168.3.23 and you do

$ping 192.168.2.11
$ping 192.168.1.1

both will be unsuccessful because they belong to different subnets. (Read Section 2.3 for exceptions).

It is worth to note that even though the hosts all lie on the same physical LAN segment, the hosts are logically on different subnets and there are no routes to the other subnets.

If we still want to ping the computers in the other subnets, then we should configure our system with one of the IPaddress of that subnet. To do this we use the concept of **interface aliases** i.e we give one or more IP addresses to the same network interface card using the following command.

$ifconfig eth0:1 192.168.1.23

Now let all the 3 groups configure with one more alias address. Let the machines in 192.168.1.0 subnet add alias with an address from 192.168.2.0, the ones from 192.168.2.0 add 192.168.3.0 and ones from 192.168.3.0 add 192.168.1.0.

Example :

Original IP :	192.168.3.21	192.168.2.11	192.168.1.1
	+	+	+
Aliased IP :	192.168.1.1	192.168.3.1	192.168.2.1

Now suppose a machine with IPaddress 192.168.3.23 does the following
$ping 192.168.1.1
$ping 192.168.3.1

Both pings are successful. But,

$ping 192.168.2.1
is unsuccessful

Similarly if we have n subnets then we should have n aliases.

Having n aliases has the following problem:-

1) Every time we have to configure the system with the new IPaddress of that network.
2) Doing this consumes 1 address on each subnet thereby reducing the number of hosts that are on a network.

To solve this problem, one of the system can be configured with IPaddress of all the three subnets and can be made as a default gateway to other subnets. Pick one machine in the LAN and configure it as follows:

```
$ifconfig eth0 192.168.1.254 netmask 255.255.255.0
$ifconfig eth0:1 192.168.2.254 netmask 255.255.255.0
$ifconfig eth0:2 192.168.3.254 netmask 255.255.255.0
$echo 1 > /proc/sys/net/ipv4/ip_forward
```

The first three ifconfig commands puts the host on three different subnets simultaneously. Only this host needs to have aliases. The last command is used to enable IP forwarding by the gateway. IP forwarding will forward packets from one subnet to the other.

The hosts on different subnets can now configure their gateway. For example, 192.168.2.11 would execute the following command:

```
#route add -net default gateway 192.168.2.254
```

2.2 /proc System

/proc/sys/net/ipv4/ip_forward is a kernel variable that can be set or reset by a super user. To disable the gateway from forwarding packets :

$echo 0 > /proc/sys/net/ipv4/forward will change the kernel variable

The variable has 0 set as its default value. /proc has various kernel variables seen as files and directories. /proc variables are similar to registry entries in Windows. For example, to see the information about the processor

$cat /proc/cpuinfo

```
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 15
model           : 4
model name      : Intel(R) Celeron(R) CPU 2.66GHz
stepping        : 9
cpu MHz         : 2659.334
cache size      : 256 KB
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
```

```
cpuid level     : 5
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx
lm constant_tsc up pni monitor ds_cpl tm2 cid cx16 xtpr lahf_lm
bogomips        : 5322.49
```

Similarly the values of some kernel variables which are related to networks can be seen in /proc/sys/net/ipv4

2.3 ARP

ARP stands for Address Resolution Protocol. Given an IP address it returns the interface of the host that has the IP address. To see the contents of ARP cache we use the command

$/usr/sbin/arp

```
Address                  HWtype  HWaddress           Flags Mask            Iface
deer.cs.iitm.ernet.in    ether   00:0C:76:C5:43:6E   C                     eth0
thulasi.cs.iitm.ernet.i  ether   00:16:76:9B:FF:3F   C                     eth0
```

ARP maintains the physical addresses in a local cache which is refreshed every 10 minutes. When you set up the gateways 192.168.1.254 or 2.254 or 3.254 as the default gateway the host machines should strictly not be able to ping other subnets. But, before setting up the configuration if you had pinged other machines, their physical addresses could have been in your ARP cache. Flush the ARP cache after the gateway configuration and ping again. You will not be able to go across subnets anymore (except when the default configuration of gateway is used at the hosts).

The experiment above assumes that the switch that is used to connect the hosts is an intelligent switch which does selective forwarding of frames to hosts. However, if you have switches that broadcast every frame on every port, the results above may vary.

2.4 MTU

We can check the effect of Maximum Transfer Unit, MTU that is used by ethernet.

To change the MTU use the following command

$ ifconfig eth0 192.168.3.23 mtu 1000

By default it is 1500 bytes. To check this effect use secure shell connection.

First each host should run secure shell server. Add a user to access this server and add the username temp and use password temp.

$ /etc/init.d/ssh start
$adduser temp

This will ask you to enter information like name etc, only give the password. Other details are optional. Let another student create a large file using the commands

$cat /etc/passwd /etc/passwd > a
$cat a a > b
$cat b b > c
$cat c c > a

and so on. The effect of MTU is seen only when large amounts of data is transferred.

Now transfer this file using

$scp c temp@<neighbor's ip>:

Example (from 192.168.1.2)

$scp c temp@192.168.1.3:

Keep changing the MTU at both sender and receiver side and observe the output. Note down the time scp takes to transfer the same file using different MTU values. Plot a graph with different MTUs on the x-axis and the time taken in the y-axis. Is there any trend?

Sometimes we observe that for less MTU we get less transmission and for more MTU we get more transmission time. For ex:- when MTU =512 transmission time is 40, when MTU=1000 transmission time is 43. Ask the students for possible explanations.

Lab Session 3
Socket Programming
18 June 2008

A network data transfer can be thought of as a file that is opened in a remote location and being read by an application in another location. Network data transfer is achieved using sockets. Sockets are similar to files and have descriptors.

A connection is a five tuple consisting of

(local id, local process, foreign id, foreign process,protocol)

Half connection is (local id, local process, protocol)

Bind system call performs a half connection.

The socket system calls used for connection oriented and connectionless are different. Refer to Chapter 6. “Berkeley Sockets” of Unix Network Programming by Richard Stevens for connection oriented and connectionless protocols.

3.1 Function Calls :

Server side connectionless:

socket( ) system call creates a socket.

bind( ) system call binds the socket to a port address.

recvfrom( ) is used to read data.

sendto( ) is used to write data.

Client side connectionless:-

socket( ) system call creates a socket.

bind( ) system call binds the socket to a port address.

sendto( ) is used to write data.

recvfrom( ) is used to read data.

Server side connection oriented:-

socket( ) system call creates a socket.

bind( ) system call binds the socket to a port address.

listen( ) is executed before any client sends a request. This means the server is listening to receive the connection.

listen( ) forks a child for each connection that will be established using accept( ).

read( ) is used to read data.

write( ) is used to write data.

Client side connection oriented:-

socket( ) system call creates a socket.

bind( ) system call binds the socket to a port address.

Connect( ) system call is used to request a connection.

read( ) is used to write data.

write( ) is used to read data.

In connectionless model we do not have connect call at the client because there is no connection that will be established. A connectionless server does not listen continuously but provides service when a request arrives from the client.

Server first executes recvfrom( ) to receive messages from clients.

Clients execute sendto( ) and recvfrom( ) which requires IP address because each packet has to be routed independently. To show the concepts of socket programming give the entire program for UDP client and UDP server. In TCP client and TCP server, delete few important lines which has system calls like connect bind etc., put a comment with TODO prefix and ask students to write the code referring to some material.

3.2 Lab Work:

In the lab, you are provided four files udpServer.c, udpClient.c, tcpServer.c and tcpClient.c. The UDP server/client combination is fully functional. Use the UDP server and client code to illustrate some of the function calls.

First you should check the working of UDP client and UDP server code.

Suppose all these files are present in one system 192.168.1.254, copy them to your system using

$scp temp@192.168.1.254: sockets/* .

Open two terminals, one will be used to observe server and another will be used to observe the client.

Compile server first, using command

$gcc udpServer.c –o udpServer

And udpClient.c using

$gcc udpClient.c –o udpClient

in the two separate terminals which are open.

Now run udpServer first so that it is ready to receive the requests.
$ ./udpServer

Then run udpClient with the parameters

$ ./udpClient 192.168.1.7 5500 a b c d

This sends a b c d to the server at 192.168.1.7 to port number 5500 for the number of times specified in the program (see the source code).

We observe that the packets may get jumbled up(you can see, for example, a c d b, instead of a b c d) because it is connectionless and any datagram that is reordered by the network is delivered as is by the transport layer to the application.

Open another terminal and invoke another copy of the client to connect to the same server. Send a different set of messages from this.

$ ./udpClient 192.168.1.7 5500 p q r s

Let both UDP clients send the messages simultaneously. Observe the output at the server several packets may be lost because of unreliable transfer.

We can count the number of packets sent at the sender and the number of packets received by the receiver using a simple counter variable in the code.

The observation shows that if 40,000 packets are sent the number of packets received may be around 9678.

The same experiment can be done using tcpClient and tcpServer. However, in TCP, many packets may be sent together to increase efficiency. Therefore at the receiving end the number of packets received will be less.

The first two program segments below are for a full fledged UDP server and client. The second two are for TCP client and server and has all the essential network function calls deleted out and marked with “TODO” instead. The students can be asked to write appropriate calls with appropriate parameters. You can also emphasize on the return values of various network calls.

udpClient :

```
/* fpont 12/99 */
/* pont.net    */
/* udpClient.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h> /* memset() */
#include <sys/time.h> /* select() */

#define REMOTE_SERVER_PORT 5500
#define MAX_MSG 100


int main(int argc, char *argv[]) {

  int sd, rc, I, j;
  struct sockaddr_in cliAddr, remoteServAddr;
  struct hostent *h;

  /* check command line args */
  if(argc<3) {
    printf(“usage : %s <server> <data1> ... <dataN> \n”, argv[0]);
    exit(1);
  }

  /* get server IP address (no check if input is IP address or DNS name
*/
  h = gethostbyname(argv[1]);
  if(h==NULL) {
    printf(“%s: unknown host ‘%s’ \n”, argv[0], argv[1]);
    exit(1);
  }
```

```
  printf("%s: sending data to '%s' (IP : %s) \n", argv[0], h->h_name,
       inet_ntoa(*(struct in_addr *)h->h_addr_list[0]));

  remoteServAddr.sin_family = h->h_addrtype;
  memcpy((char *) &remoteServAddr.sin_addr.s_addr,
       h->h_addr_list[0], h->h_length);
  remoteServAddr.sin_port = htons(REMOTE_SERVER_PORT) ;

  /* socket creation */
  sd = socket(AF_INET,SOCK_DGRAM,0);
  if(sd<0) {
    printf("%s: cannot open socket \n",argv[0]);
    exit(1);
  }

  /* bind any port */
  cliAddr.sin_family = AF_INET;
  cliAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  cliAddr.sin_port = htons(0);

  rc = bind(sd, (struct sockaddr *) &cliAddr, sizeof(cliAddr));
  if(rc<0) {
    printf("%s: cannot bind port\n", argv[0]);
    exit(1);
  }


  /* send data */
  for(;;){
  for(i=2;i<argc;i++) {
    rc = sendto(sd, argv[i], strlen(argv[i])+1, 0,
            (struct sockaddr *) &remoteServAddr,
            sizeof(remoteServAddr));

    if(rc<0) {
      printf("%s: cannot send data %d \n",argv[0],i-1);
      close(sd);
      exit(1);
    }
  }

  }

  return 1;

}
```

udpServer:

```
/* fpont 12/99 */
/* pont.net    */
/* udpServer.c */

#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close() */
#include <string.h> /* memset() */

#define LOCAL_SERVER_PORT 6500
#define MAX_MSG 100

int main(int argc, char *argv[]) {

  int sd, rc, n, cliLen;
  struct sockaddr_in cliAddr, servAddr;
  char msg[MAX_MSG];

  /* socket creation */
  sd=socket(AF_INET, SOCK_DGRAM, 0);
  if(sd<0) {
    printf("%s: cannot open socket \n",argv[0]);
    exit(1);
  }

  /* bind local server port */
  servAddr.sin_family = AF_INET;
  servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servAddr.sin_port = htons(LOCAL_SERVER_PORT) ;
  rc = bind (sd, (struct sockaddr *) &servAddr,sizeof(servAddr));
  if(rc<0) {
    printf("%s: cannot bind port number %d \n",
         argv[0], LOCAL_SERVER_PORT) ;
    exit(1) ;
  }

  printf("%s: waiting for data on port UDP %u\n",
         argv[0],LOCAL_SERVER_PORT) ;

  /* server infinite loop */
  while(1) {

    /* init buffer */
    memset(msg,0x0,MAX_MSG);


    /* receive message */
    cliLen = sizeof(cliAddr);
    n = recvfrom(sd, msg, MAX_MSG, 0,
             (struct sockaddr *) &cliAddr, &cliLen);

    if(n<0) {
      printf("%s: cannot receive data \n",argv[0]);
      continue;
    }

    /* print received message */
    printf("%s: from %s:UDP%u : %s \n",
         argv[0],inet_ntoa(cliAddr.sin_addr),
```

```
        ntohs(cliAddr.sin_port),msg) ;

  }/* end of server infinite loop */

return 0 ;

}
```

tcpClient code : (Fill in with function calls at appropriate places marked with TODO)

```
/* fpont 12/99 */
/* pont.net    */
/* tcpClient.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */

#define SERVER_PORT 5500
#define MAX_MSG 100

int main (int argc, char *argv[]) {

  int sd, rc, i,j;
  struct sockaddr_in localAddr, servAddr;
  struct hostent *h;

  if(argc < 3) {
    printf("usage: %s <server> <data1> <data2> ... <dataN>\n",argv[0]);
    exit(1);
  }

  h = gethostbyname(argv[1]);
  if(h==NULL) {
    printf("%s: unknown host '%s'\n",argv[0],argv[1]);
    exit(1);
  }

  servAddr.sin_family = h->h_addrtype;
  memcpy((char *) &servAddr.sin_addr.s_addr, h->h_addr_list[0], h-
>h_length);
  servAddr.sin_port = htons(SERVER_PORT);

  //TODO Create a socket

  if(sd<0) {
    perror("cannot open socket ");
    exit(1);
  }

  /* bind any port number */
  localAddr.sin_family = AF_INET;
```

```
  localAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  localAddr.sin_port = htons(0);

  //TODO Bind the socket

  if(rc<0) {
    printf("%s: cannot bind port TCP %u\n",argv[0],SERVER_PORT);
    perror("error ");
    exit(1);
  }

  // TODO connect to server

  if(rc<0) {
    perror("cannot connect ");
    exit(1);
  }

  for(j=10000; j > 0; j--){
  for(i=2;i<argc;i++) {

    rc = send(sd, argv[i], strlen(argv[i]) + 1, 0);

    if(rc<0) {
      perror("cannot send data ");
      close(sd);
      exit(1);

    }

    printf("%s: data%u sent (%s)\n",argv[0],i-1,argv[i]);


  }
  }

return 0;

}
```

tcpServer code: (Fill in with function calls at appropriate places marked with TODO)

```
/* fpont 1/00 */
/* pont.net    */
/* tcpServer.c */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /* close */
```

```
#define SUCCESS 0
#define ERROR   1

#define END_LINE 0x0
#define SERVER_PORT 5500
#define MAX_MSG 100

/* function readline */
int read_line();

int main (int argc, char *argv[]) {

  int sd, newSd, cliLen;

  struct sockaddr_in cliAddr, servAddr;
  char line[MAX_MSG];


  //TODO Create socket

   if(sd<0) {
    perror("cannot open socket ");
    return ERROR;
  }

  /* bind server port */
  servAddr.sin_family = AF_INET;
  servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
  servAddr.sin_port = htons(SERVER_PORT);

  if(bind(sd, (struct sockaddr *) &servAddr, sizeof(servAddr))<0) {
    perror("cannot bind port ");
    return ERROR;
  }

  //TODO : Listen

  while(1) {

    printf("%s: waiting for data on port TCP
%u\n",argv[0],SERVER_PORT);

    cliLen = sizeof(cliAddr);
    //TODO Accept the connection
    if(newSd<0) {
      perror("cannot accept connection ");
      return ERROR;
    }

    /* init line */
    memset(line,0x0,MAX_MSG);

    /* receive segments */
    while(read_line(newSd,line)!=ERROR) {

      printf("%s: received from %s:TCP%d : %s\n", argv[0],
           inet_ntoa(cliAddr.sin_addr),
```

```
           ntohs(cliAddr.sin_port), line);
      /* init line */
      memset(line,0x0,MAX_MSG);

    } /* while(read_line) */

  } /* while (1) */

}


/* WARNING WARNING WARNING WARNING WARNING WARNING WARNING       */
/* this function is experimental.. I don't know yet if it works  */
/* correctly or not. Use Steven's readline() function to have    */
/* something robust.                                             */
/* WARNING WARNING WARNING WARNING WARNING WARNING WARNING       */

/* rcv_line is my function readline(). Data is read from the socket
when */
/* needed, but not byte after bytes. All the received data is read.
*/
/* This means only one call to recv(), instead of one call for
*/
/* each received byte.
*/
/* You can set END_CHAR to whatever means endofline for you. (0x0A is
\n)*/
/* read_lin returns the number of bytes returned in line_to_return
*/
int read_line(int newSd, char *line_to_return) {

  static int rcv_ptr=0;
  static char rcv_msg[MAX_MSG];
  static int n;
  int offset;

  offset=0;

  while(1) {
    if(rcv_ptr==0) {
      /* read data from socket */
      memset(rcv_msg,0x0,MAX_MSG); /* init buffer */
      n = recv(newSd, rcv_msg, MAX_MSG, 0); /* wait for data */
      if (n<0) {
      perror(" cannot receive data ");
      return ERROR;
      } else if (n==0) {
      printf(" connection closed by client\n");
      //TODO Close the connection
      return ERROR;
      }
    }

    /* if new data read on socket */
    /* OR */
    /* if another line is still in buffer */
```

```
    /* copy line into 'line_to_return' */
    while(*(rcv_msg+rcv_ptr)!=END_LINE && rcv_ptr<n) {
      memcpy(line_to_return+offset,rcv_msg+rcv_ptr,1);
      offset++;
      rcv_ptr++;
    }

    /* end of line + end of buffer => return line */
    if(rcv_ptr==n-1) {
      /* set last byte to END_LINE */
      *(line_to_return+offset)=END_LINE;
      rcv_ptr=0;
      return ++offset;
    }

    /* end of line but still some data in buffer => return line */
    if(rcv_ptr <n-1) {
      /* set last byte to END_LINE */
      *(line_to_return+offset)=END_LINE;
      rcv_ptr++;
      return ++offset;
    }

    /* end of buffer but line is not ended => */
    /*  wait for more data to arrive on socket */
    if(rcv_ptr == n) {
      rcv_ptr = 0;
    }

  } /* while */
}
```

Lab Session 4
LAN Trainer Kit
19 June 2008

Refer to LAN Trainer Quick Introduction guide **LAN_TQuick Ref.pdf** for the list of experiments that can be run on the LANT Kit from Benchmark Systems.

Lab Session 5
Netstat and Other Utilities
20 June 2008

For this exercise, take the working code of tcpServer and tcpClient. First run the tcpServer

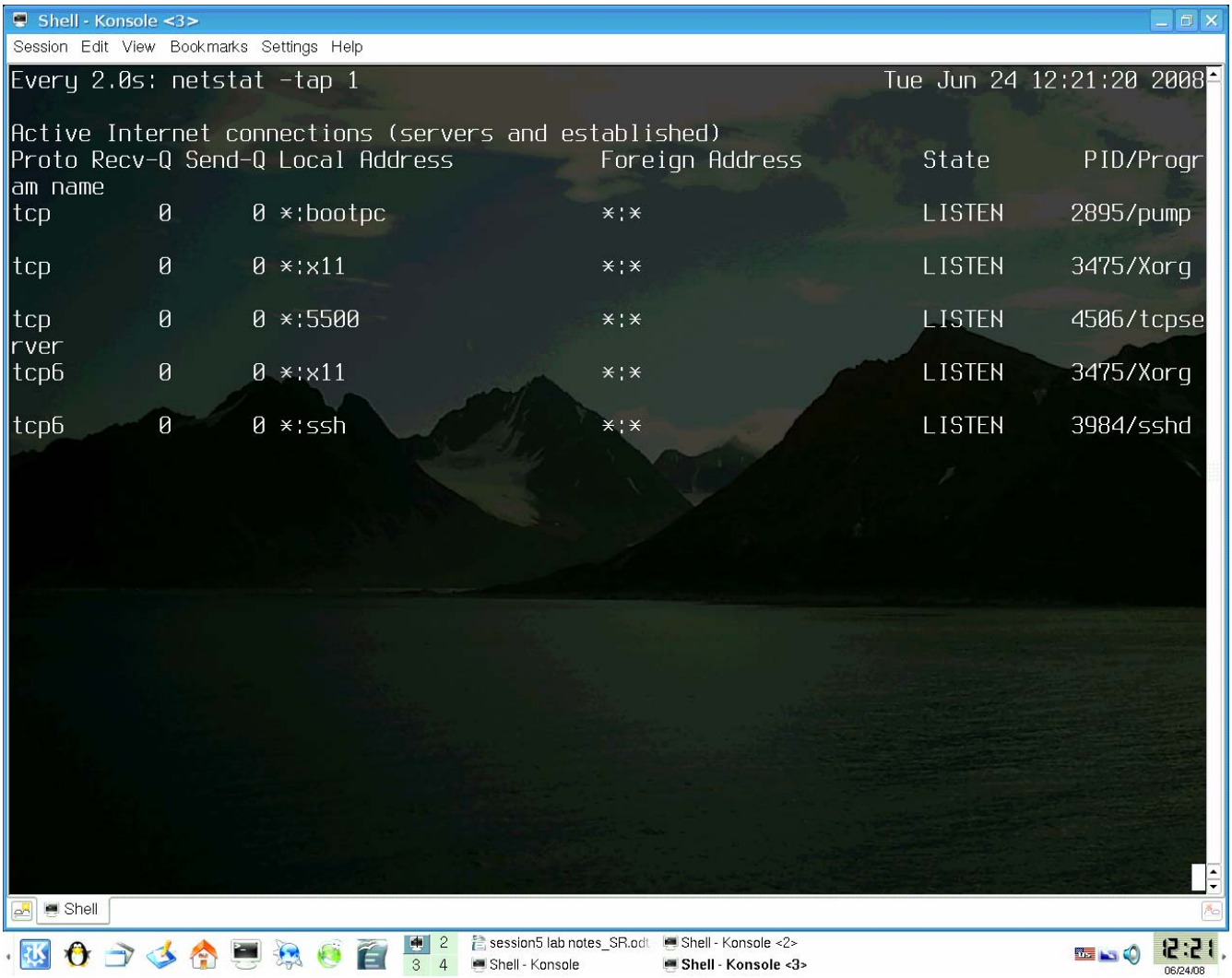
```
root@Knoppix:/ramdisk/home/knoppix# gcc tcpServer.c -o tcpserver
root@Knoppix:/ramdisk/home/knoppix# ./tcpserver
./tcpserver: waiting for data on port TCP 5500
```

Run **netstat** and observe that it shows that the tcp is listening at port 5500.

```
root@Knoppix:/ramdisk/home/knoppix# netstat -tap
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 *:bootpc                *:*                     LISTEN     2895/pump
tcp        0      0 *:x11                   *:*                     LISTEN     3475/Xorg
tcp        0      0 *:5500                  *:*                     LISTEN 4506/tcpserver
tcp6       0      0 *:x11                   *:*                     LISTEN     3475/Xorg
tcp6       0      0 *:ssh                   *:*                     LISTEN     3984/sshd
```

We can add a watch for netstat which keeps displaying the output of netstat every 2 seconds.

```
root@Knoppix:/ramdisk/home/knoppix# watch netstat -tap
```



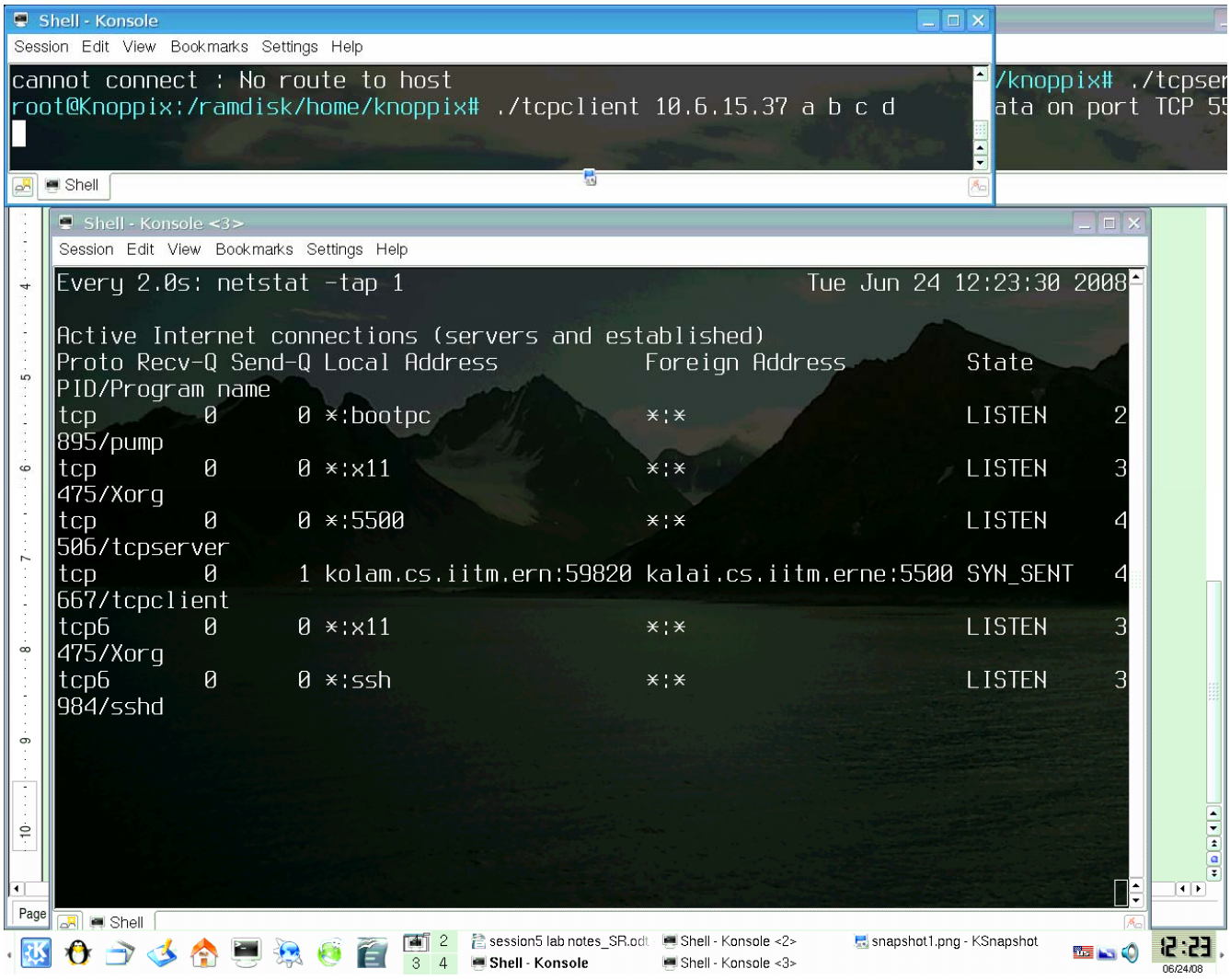
The command lists all TCP connections that are open and the applications that have opened it. In the snapshot provided earlier, line number 3 shows 4506/tcpserver which indicates that the executable **tcpserver** has a process id of 4506. Thus port number 5500 is handled by pid 4506. Recollect from Lab 4 that for a half-connection pid and port number are both important.

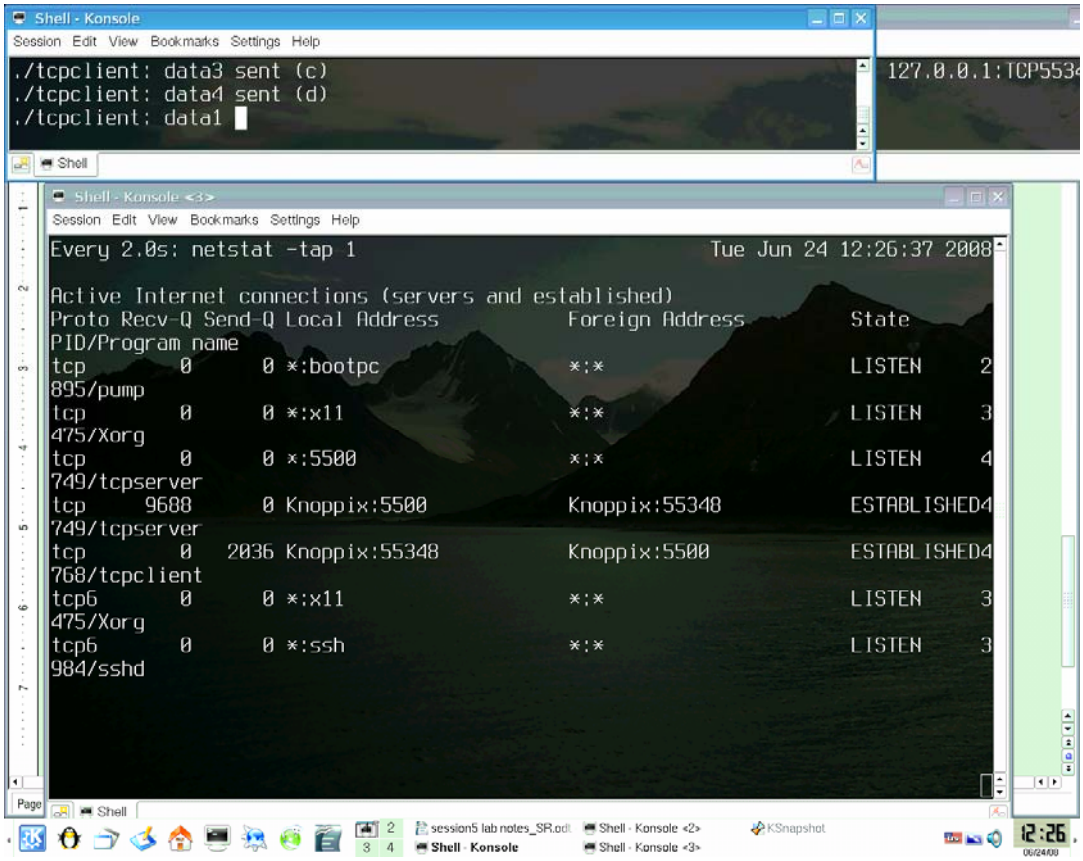
You can quickly verify if the executable tcpserver is indeed having pid 4506 :
root@Knoppix:/ramdisk/home/knoppix# ps -ef | grep tcpserver
root 4506 4488 0 12:13 pts/2 00:00:00 ./tcpserver
root 4543 4530 0 12:16 pts/3 00:00:00 grep --colour=auto tcpserver

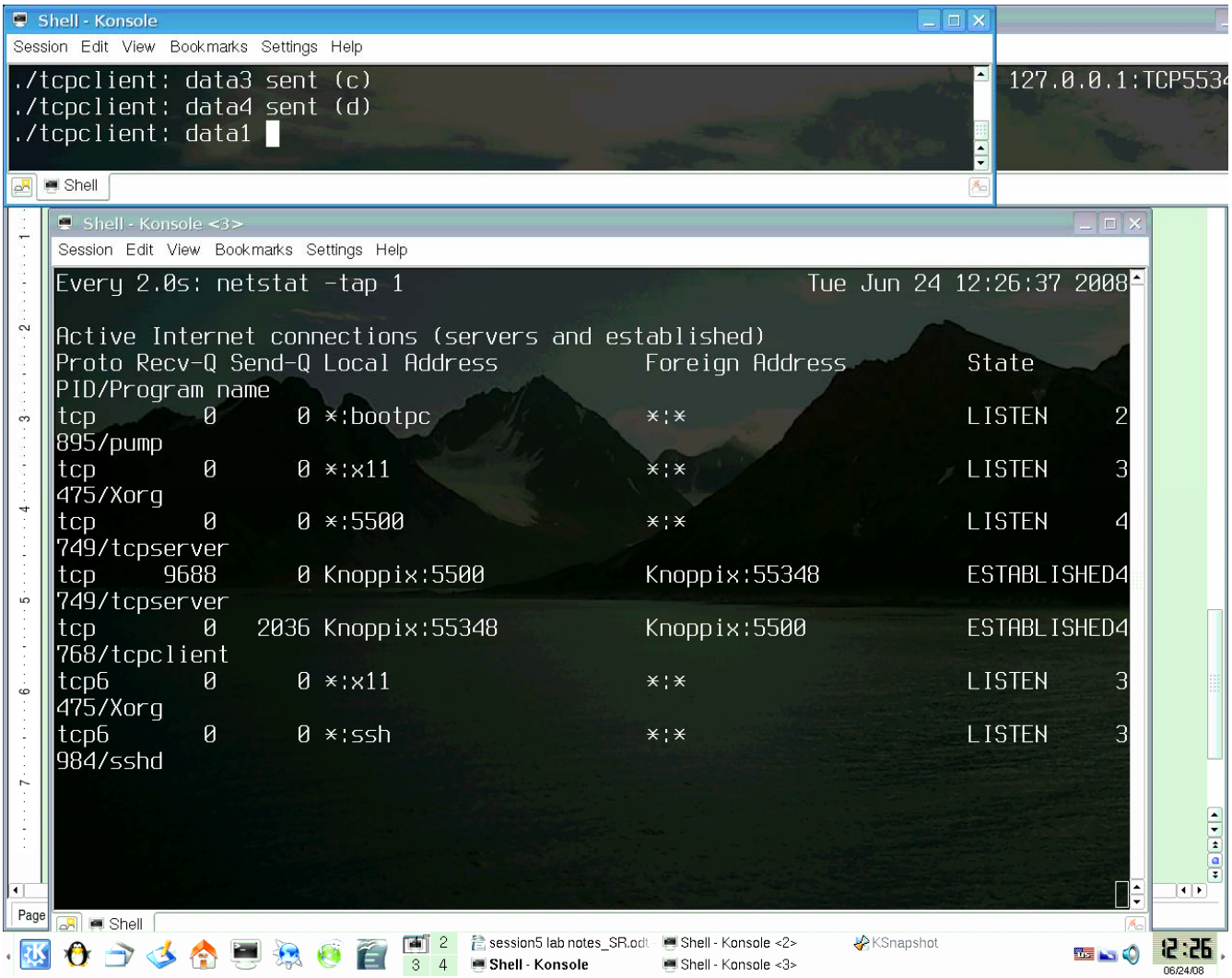
```
Then run the tcpClient
root@Knoppix:/ramdisk/home/knoppix# gcc tcpClient.c -o tcpclient

./tcpclient 10.6.15.37 a b c d
```

and observe various states

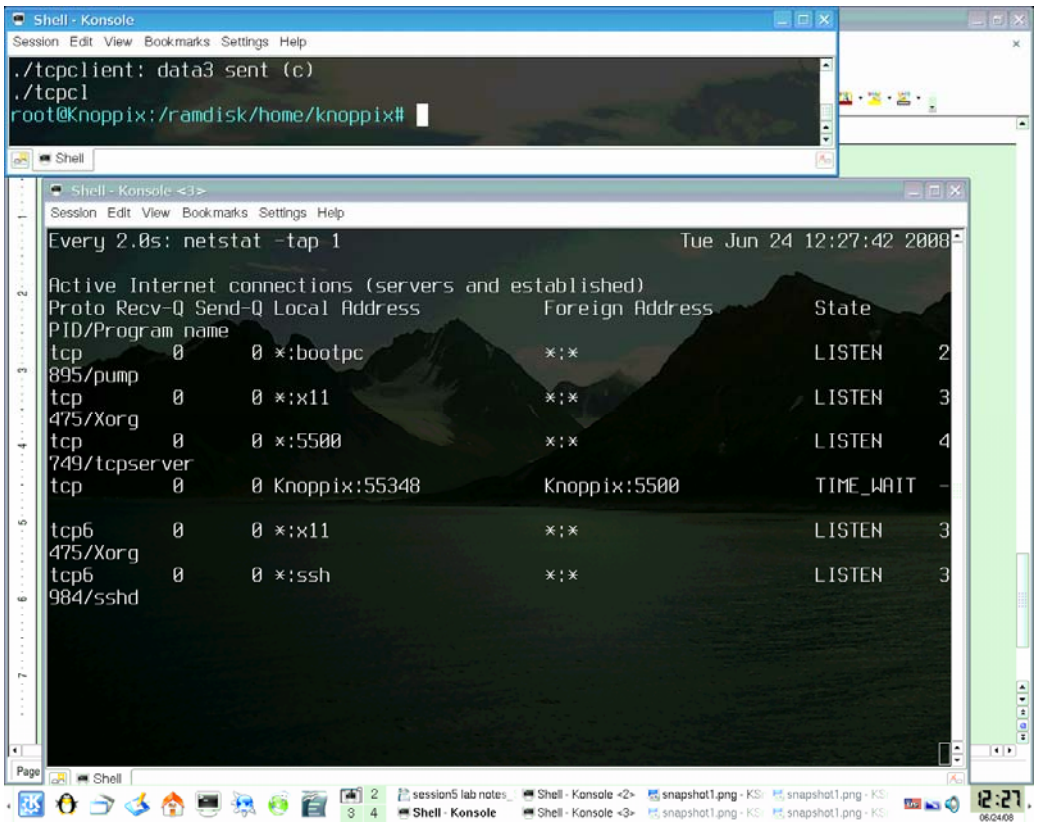
Observe that when client sends a request, the state changes to SYN SENT state and quickly changes to ESTABLISHED.

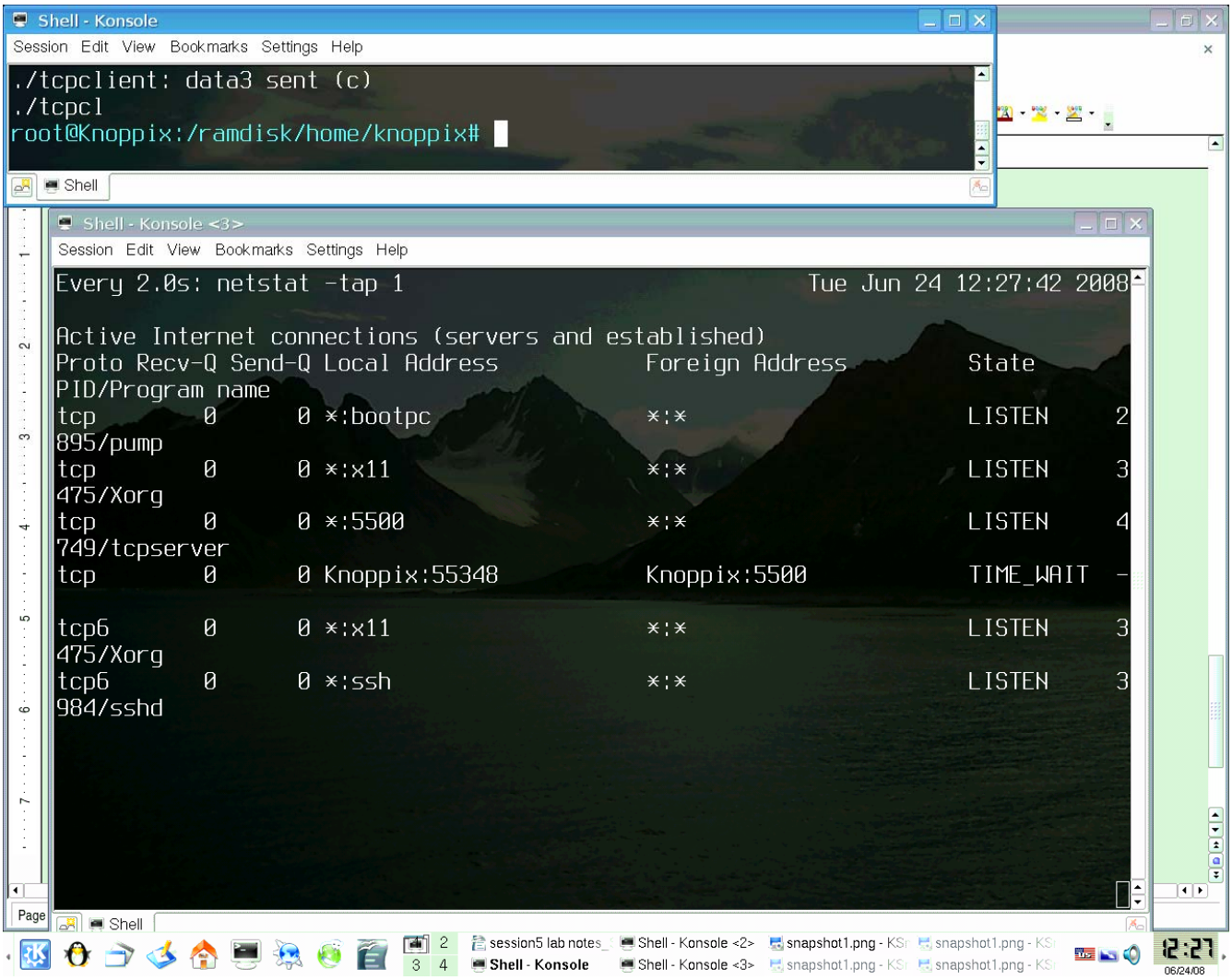



Shell - Konsole
Session Edit View Bookmarks Settings Help
./tcpclient: data3 sent (c)
./tcpclient: data4 sent (d)
./tcpclient: data1
127.0.0.1:TCP553
Shell
Shell - Konsole <3>
Session Edit View Bookmarks Settings Help
Every 2.0s: netstat -tap 1 Tue Jun 24 12:26:37 2008
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
PID/Program name
tcp 0 0 *:bootpc *:* LISTEN 2
895/pump
tcp 0 0 *:x11 *:* LISTEN 3
475/Xorg
tcp 0 0 x:5500 x:x LISTEN 4
749/tcpserver
tcp 9688 0 Knoppix:5500 Knoppix:55348 ESTABLISHED4
749/tcpserver
tcp 0 2036 Knoppix:55348 Knoppix:5500 ESTABLISHED4
768/tcpclient
tcp6 0 0 *:x11 *:* LISTEN 3
475/Xorg
tcp6 0 0 *:ssh *:* LISTEN 3
984/sshd
Page
Shell
session5 lab notes_SR.odt
Shell - Konsole <2>
Shell - Konsole <3>
KSnapshot
12:26

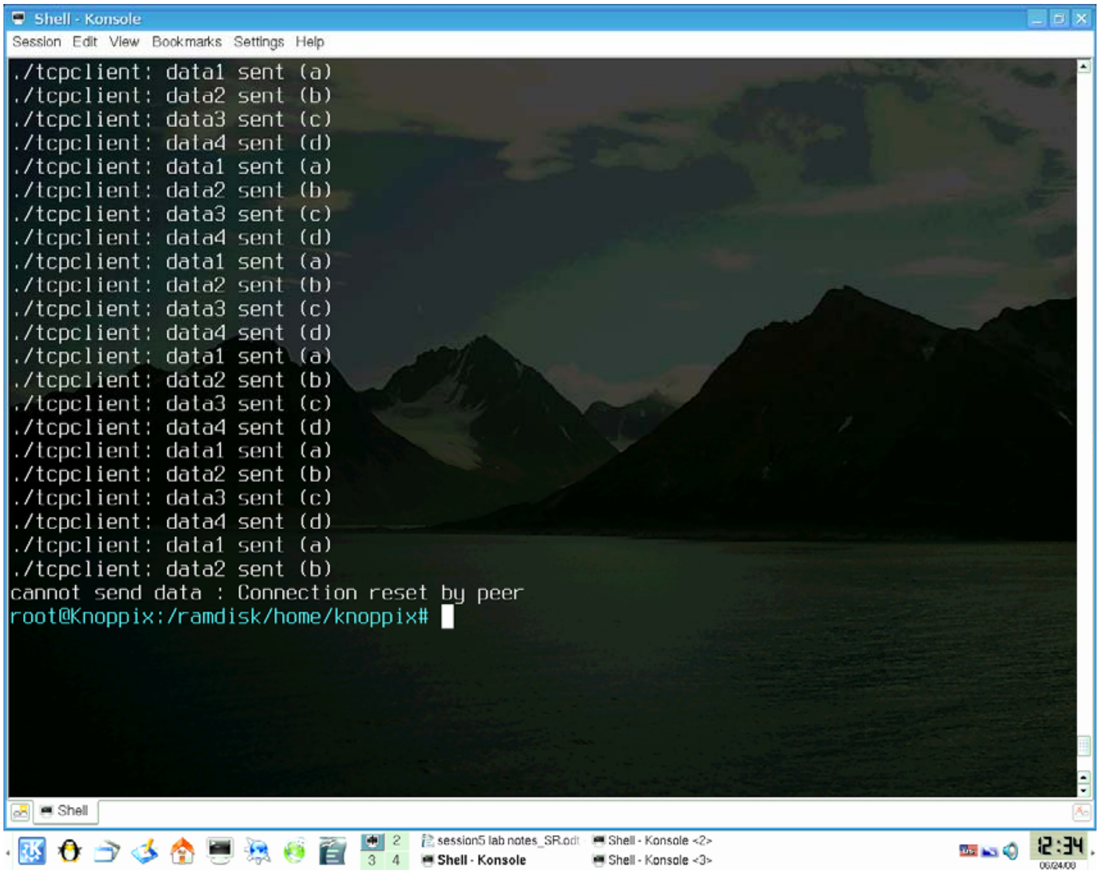


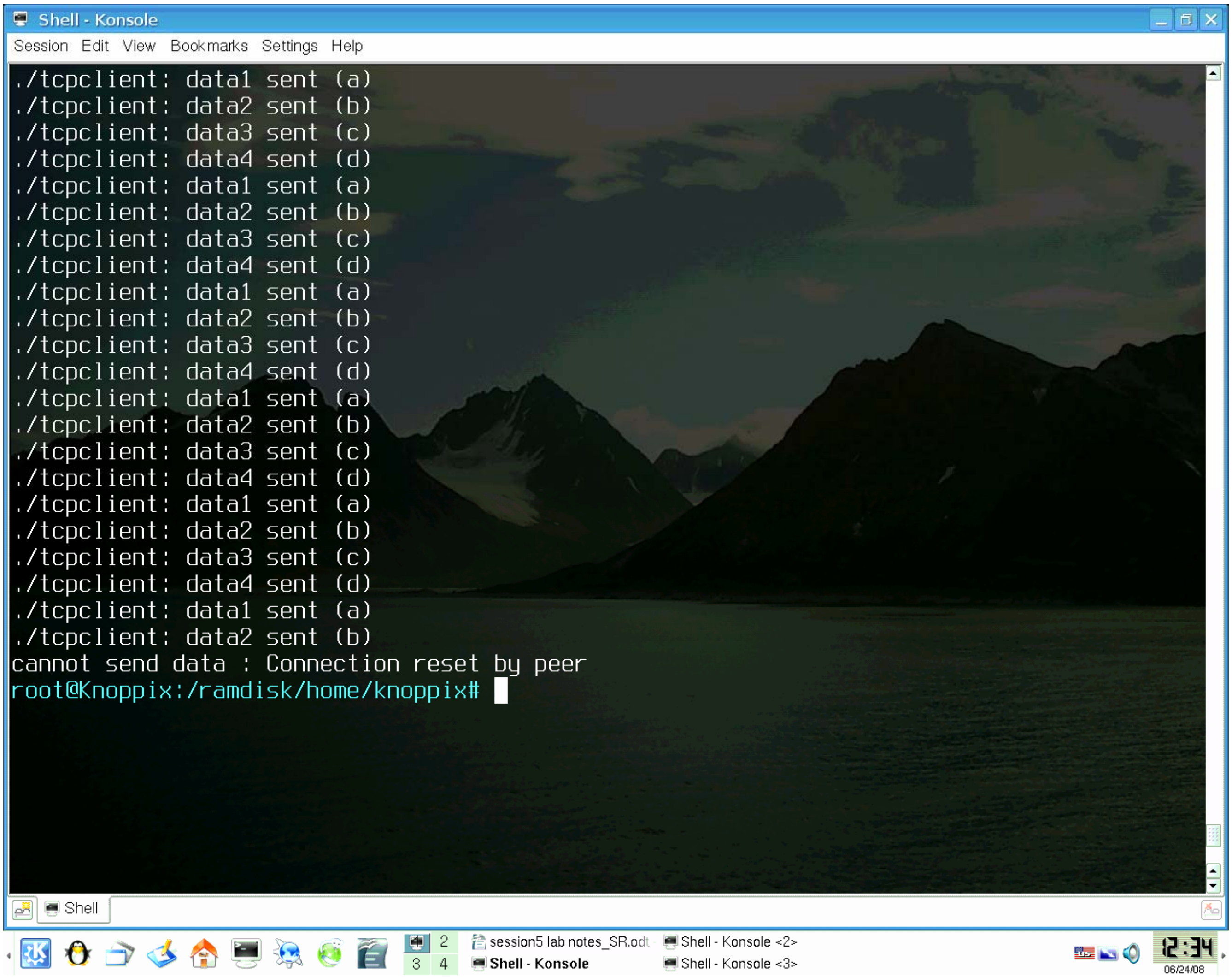
After the data transfer is over, the client goes to FIN WAIT1, FIN_WAIT2 and finally to the TIME_WAIT state.


Shell - Konsole
Session Edit View Bookmarks Settings Help
./tcpclient: data3 sent (c)
./tcpcl
root@Knoppix:/ramdisk/home/knoppix#
Shell
Shell - Konsole <3>
Session Edit View Bookmarks Settings Help
Every 2.0s: netstat -tap 1 Tue Jun 24 12:27:42 2008
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
PID/Program name
tcp 0 0 *:bootpc *:* LISTEN 2
895/pump
tcp 0 0 *:x11 *:* LISTEN 3
475/Xorg
tcp 0 0 *:5500 *:* LISTEN 4
749/tcpserver
tcp 0 0 Knoppix:55348 Knoppix:5500 TIME_WAIT -
tcp6 0 0 *:x11 *:* LISTEN 3
475/Xorg
tcp6 0 0 *:ssh *:* LISTEN 3
984/sshd
Page
Shell
12:27
06/24/08



Now suppose we abruptly terminate the server connection and immediately connect it back then we observe this output at the client side.


Shell - Konsole
Session Edit View Bookmarks Settings Help
./tcpclient: data1 sent (a)
./tcpclient: data2 sent (b)
./tcpclient: data3 sent (c)
./tcpclient: data4 sent (d)
./tcpclient: data1 sent (a)
./tcpclient: data2 sent (b)
./tcpclient: data3 sent (c)
./tcpclient: data4 sent (d)
./tcpclient: data1 sent (a)
./tcpclient: data2 sent (b)
./tcpclient: data3 sent (c)
./tcpclient: data4 sent (d)
./tcpclient: data1 sent (a)
./tcpclient: data2 sent (b)
./tcpclient: data3 sent (c)
./tcpclient: data4 sent (d)
./tcpclient: data1 sent (a)
./tcpclient: data2 sent (b)
./tcpclient: data3 sent (c)
./tcpclient: data4 sent (d)
./tcpclient: data1 sent (a)
./tcpclient: data2 sent (b)
cannot send data : Connection reset by peer
root@Knoppix:/ramdisk/home/knoppix#
Shell
Shell - Konsole
Shell - Konsole <2>
Shell - Konsole <3>
12:34



Various experiments can be designed around the basic framework above. Some examples are

1. Terminate a server abruptly and see what happens at both sides
2. Terminate client abruptly at both sides
3. Open multiple clients and see how TCP handles that
4. When the client or server is heavily loaded, see the TX and RX queues building up.
5. Observe that the queues don't grow infinitely even if you continuously send data.

The students could be asked for possible explanations to each of these behaviors. It is always good to run these experiments many times to see the commonly expected behavior.

Lab Session 6
Wireshark and Packet Sniffing
23 June 2008

6.1 Some Common Unix Tools for Networks:

We can configure the name server using the following command

```
# vi /etc/resolv.conf
nameserver 10.6.0.1
nameserver 10.6.0.2
```

open the resolv.conf and set the nameserver as shown above. Be careful the line should not start with any space, if it does, it may give some errors.

We can resolve names using nslookup command

```
# nslookup www.google.com

knoppix@Knoppix:~$ nslookup google.com
Server:         10.6.0.2
Address:        10.6.0.2#53

Non-authoritative answer:
Name:   google.com
Address: 64.233.187.99
Name:   google.com
Address: 72.14.207.99
Name:   google.com
Address: 64.233.167.99
```

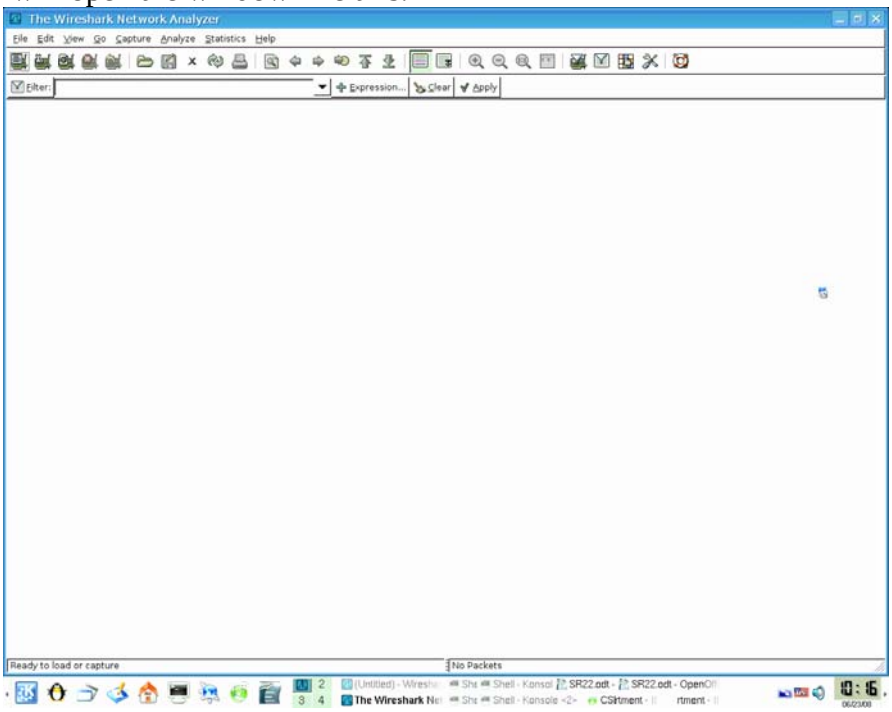
If you want to know the host name from the ipaddress we can use this command

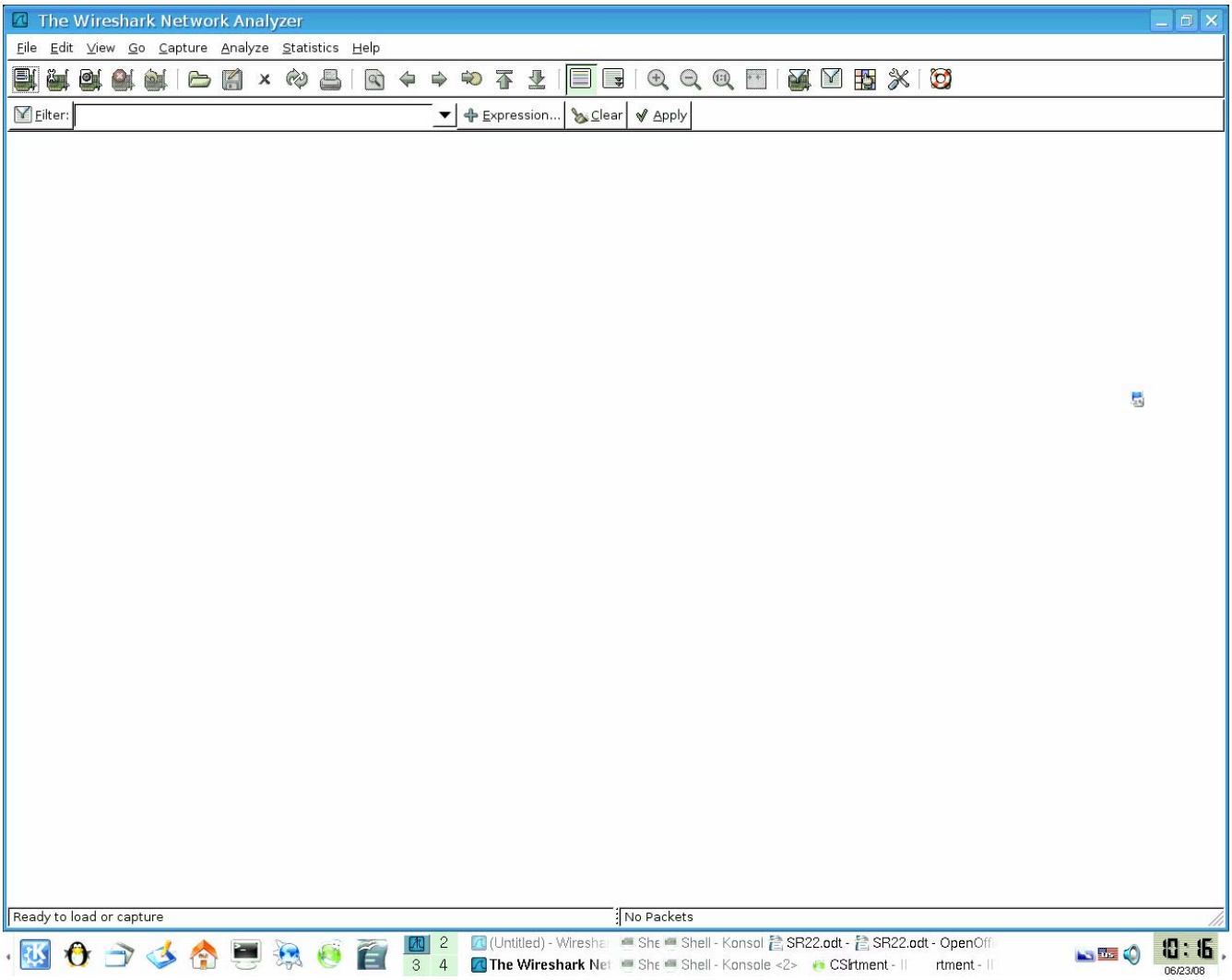
```
knoppix@Knoppix:~$ nslookup 64.233.187.99
Server:         10.6.0.2
Address:        10.6.0.2#53

Non-authoritative answer:
99.187.233.64.in-addr.arpa      name = jc-in-
f99.google.com.

Authoritative answers can be found from:
187.233.64.in-addr.arpa nameserver = ns2.google.com.
187.233.64.in-addr.arpa nameserver = ns3.google.com.
187.233.64.in-addr.arpa nameserver = ns4.google.com.
```

```
187.233.64.in-addr.arpa nameserver = ns1.google.com.
ns1.google.com  internet address = 216.239.32.10
ns2.google.com  internet address = 216.239.34.10
ns3.google.com  internet address = 216.239.36.10
ns4.google.com  internet address = 216.239.38.10
```

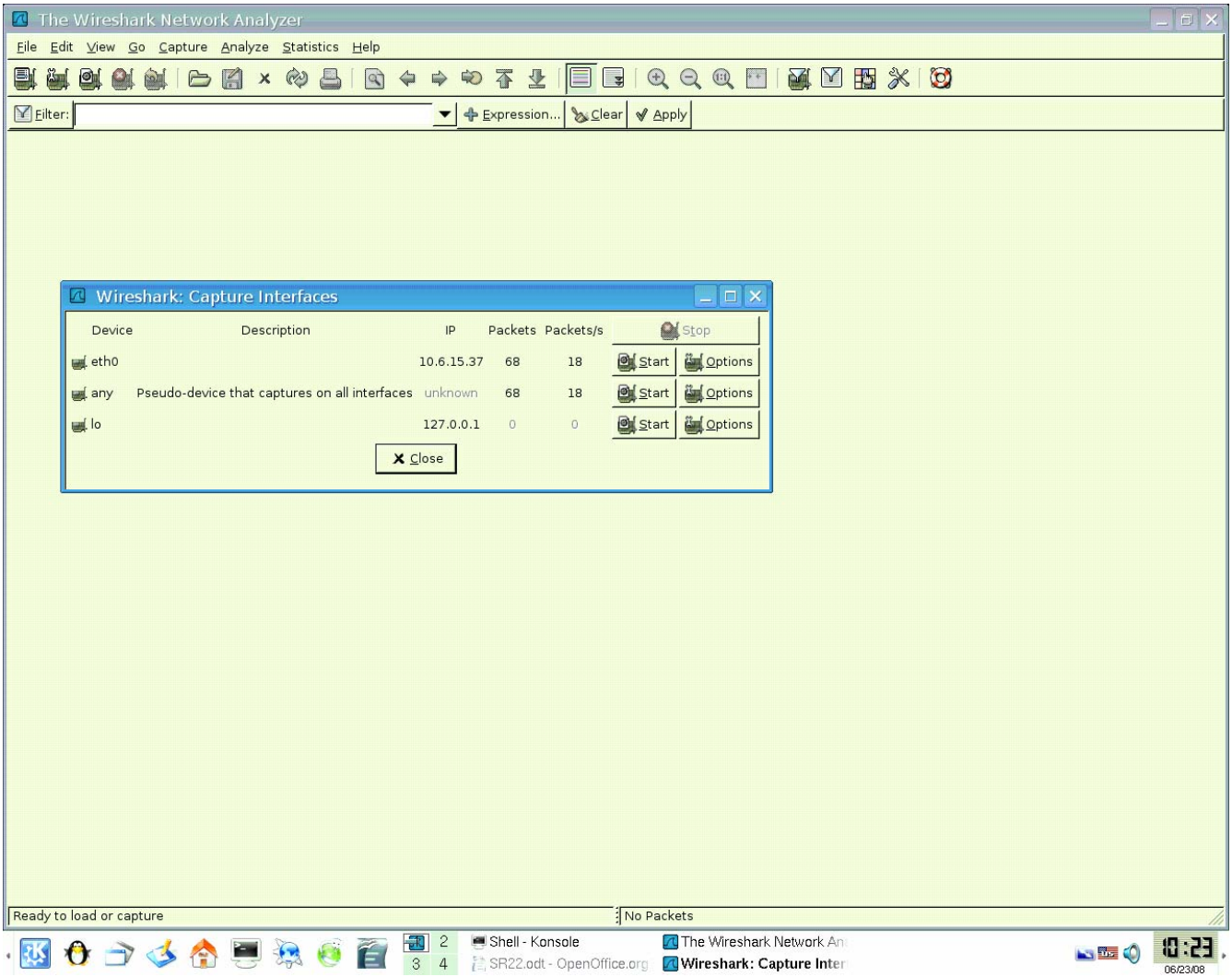
A tool called wireshark is used for packet sniffing.

$wireshark
will open the window like this.

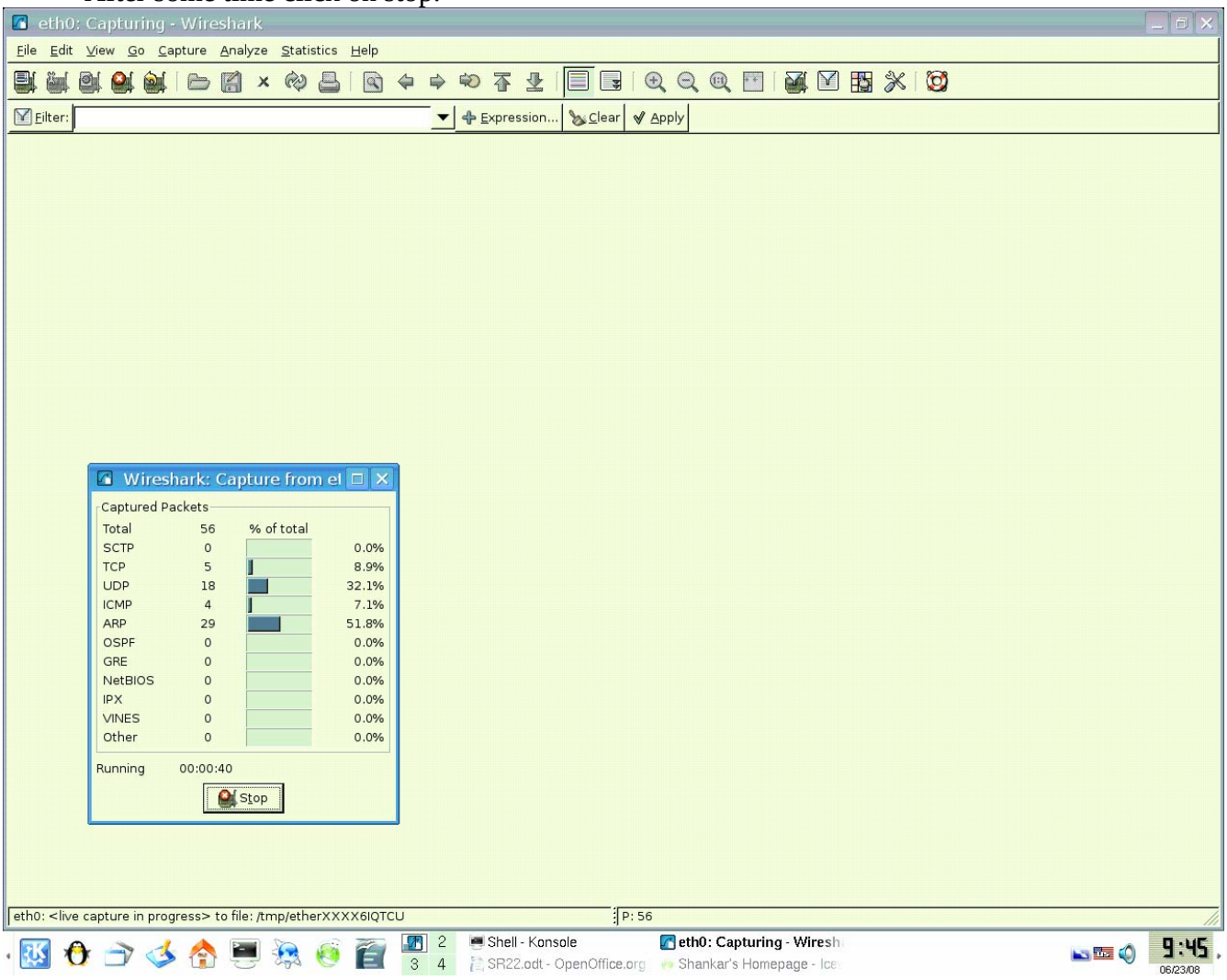


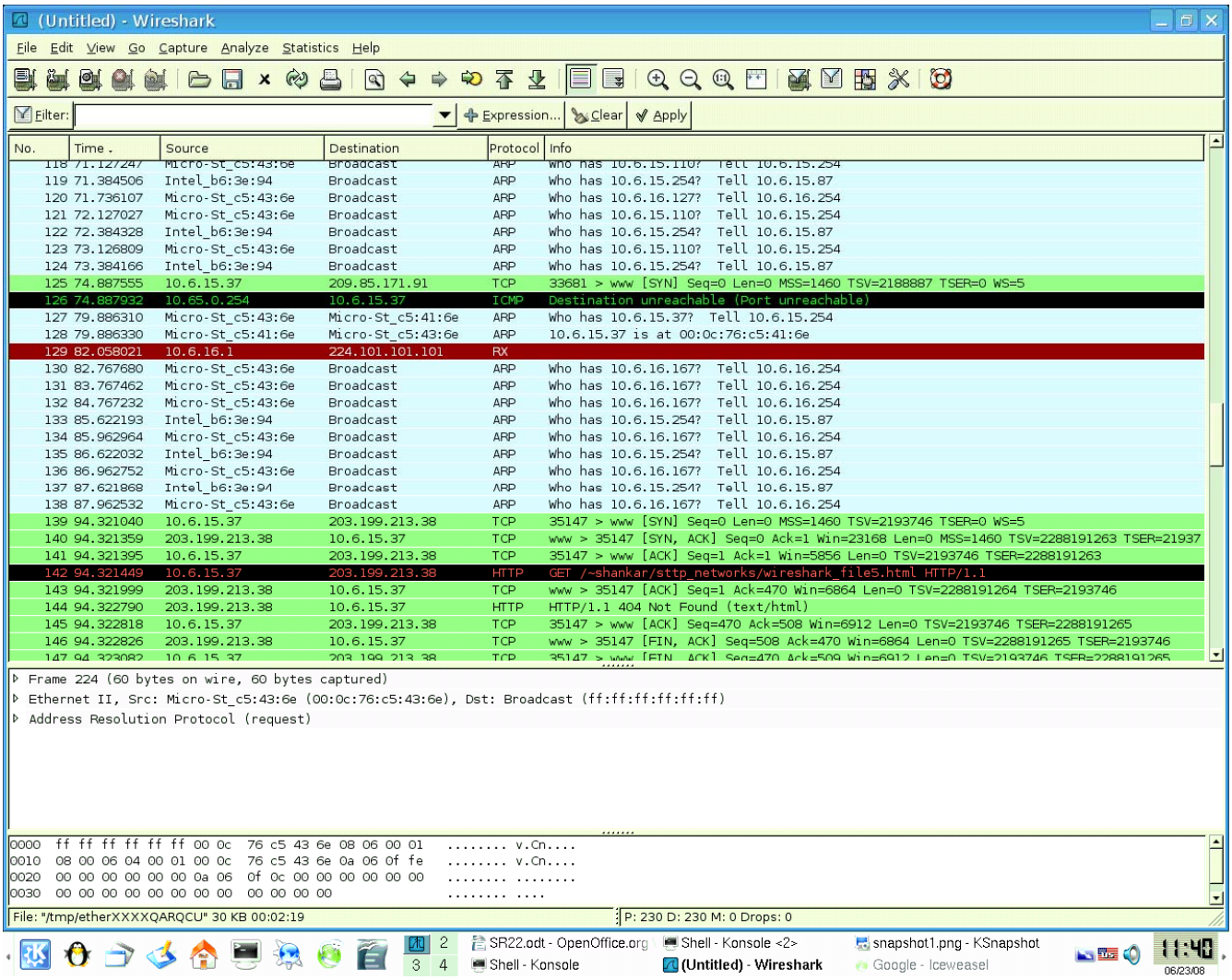

The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply
Ready to load or capture
No Packets

Click on capture and then select interface. We get a output like this.

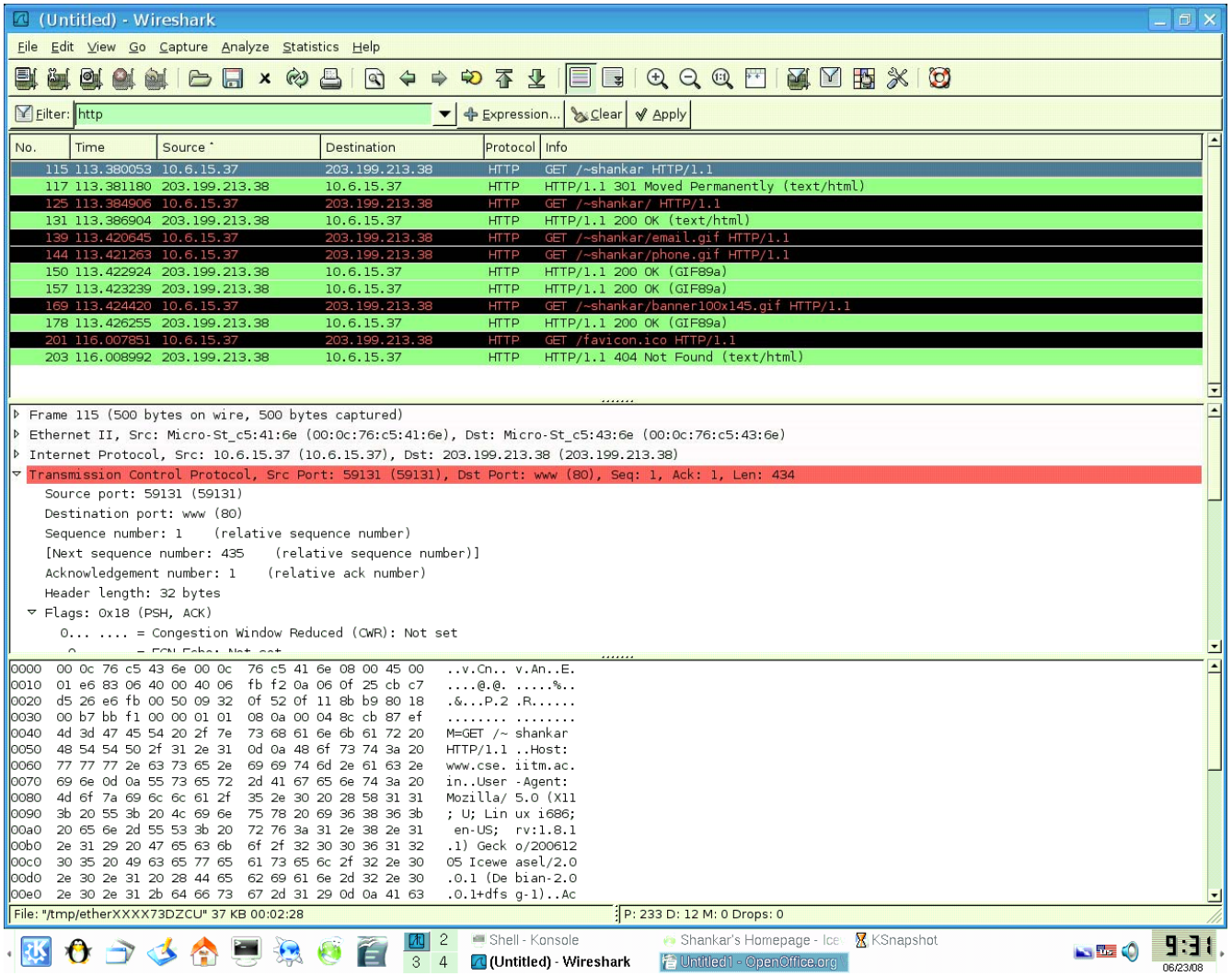


Then click on start.. This will start capturing the packets. Open a website and see the wireshark showing the number of SCTP, TCP, UDP and other frames that were captured. After some time click on stop.





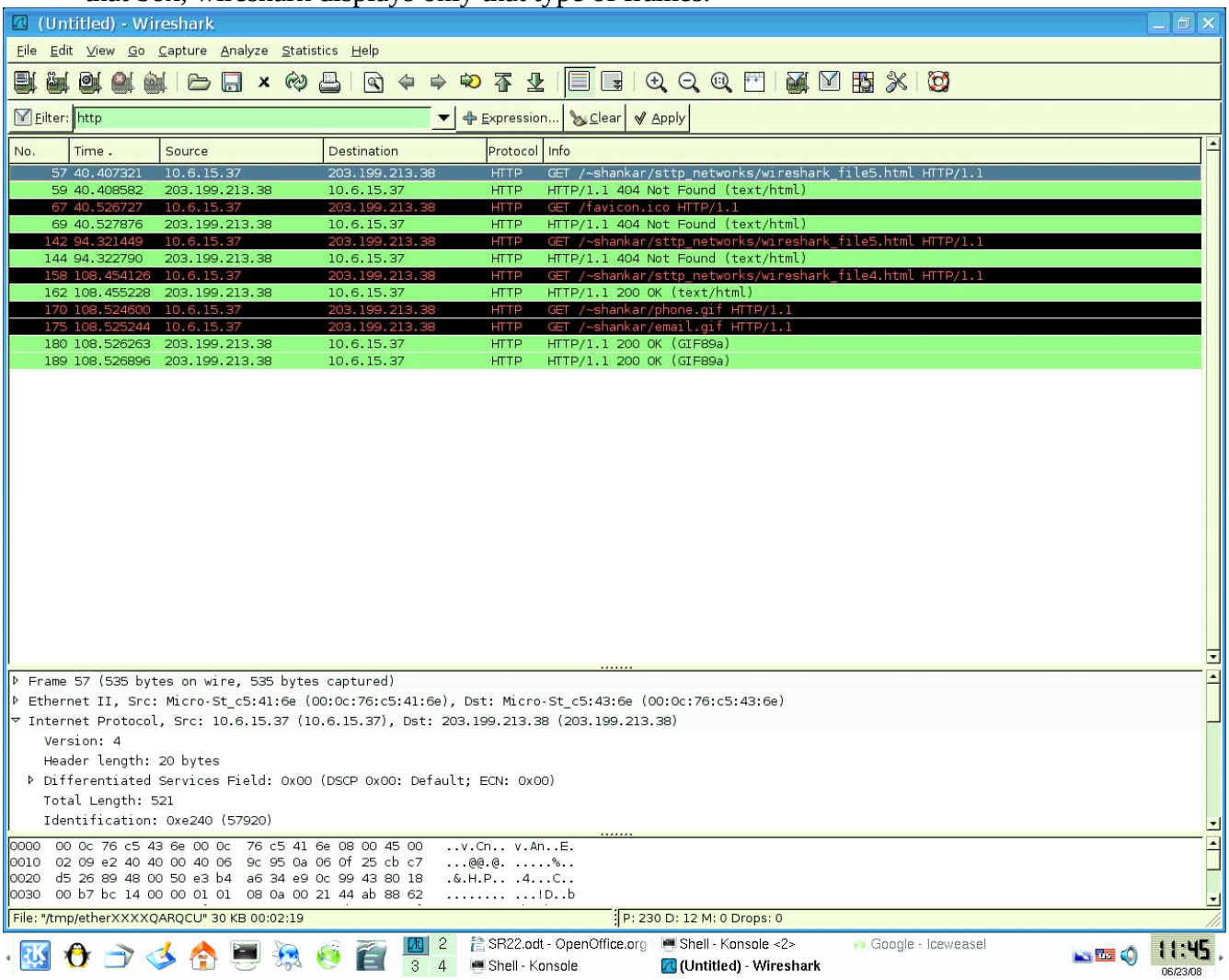
You will see a series of outputs.



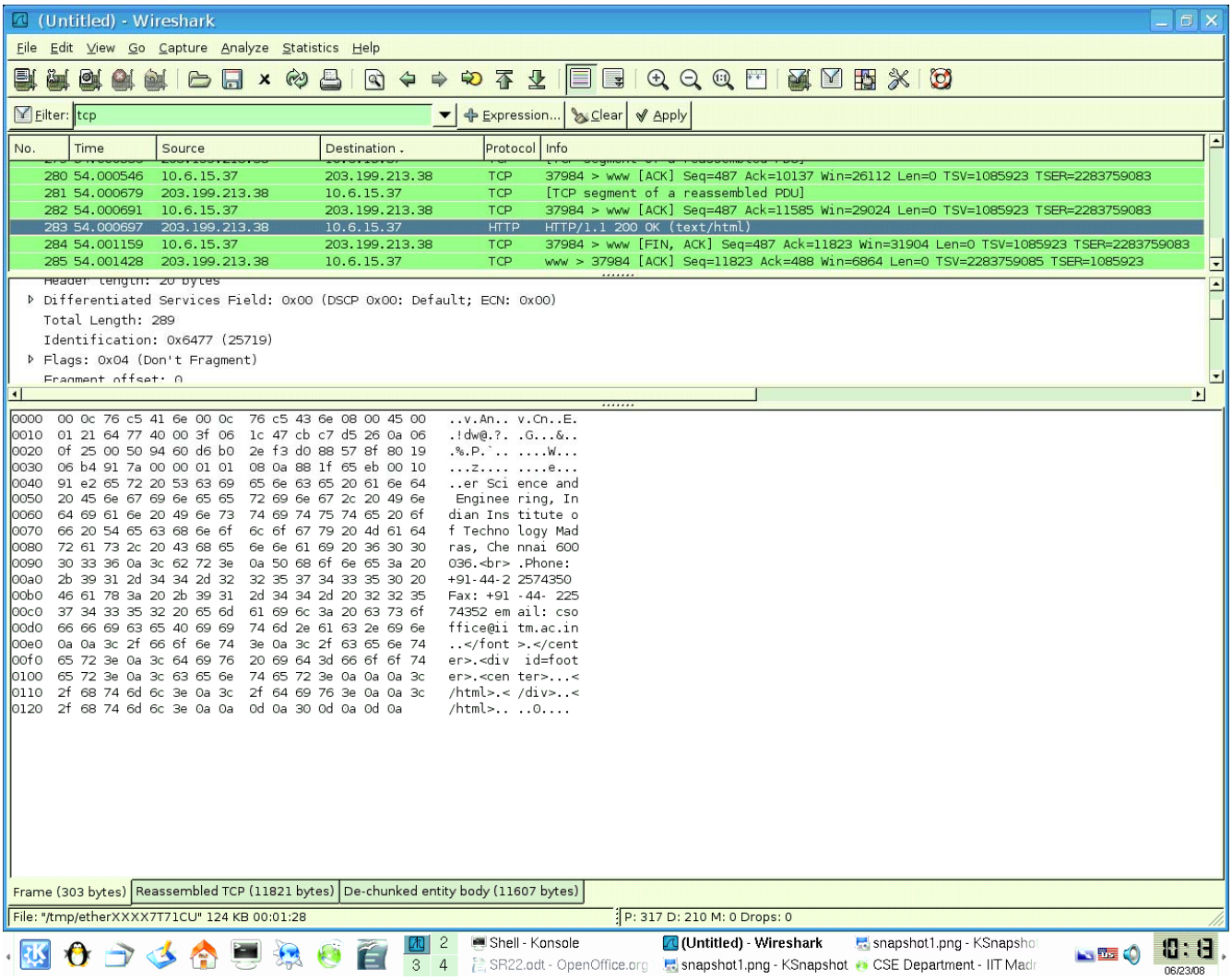
Observe that the output window has 3 sections

1) First section gives information about the sl no, which is given by the wireshark, followed by the timestamp, the source and the destination along with the protocol used to transfer it and some information.
2) Second section gives details about the frame on a layer by layer basis. By clicking on the small triangle to the left of the line, the details pertaining to the particular protocol is expanded and shown. For example, if you click on TCP, it gives details of the protocols like the source port, destination port, sequence number and details of flag etc. Similarly we have details of http, ethernet, frame and internet protocol.
3) Third section gives the hexadecimal dump along with the contents of payload

You can choose the type of packets using filter at the left, type the file you want to find in that box, wireshark displays only that type of frames.



Wireshark also allows following a particular TCP stream by right clicking on the row and selecting the follow tcp stream.

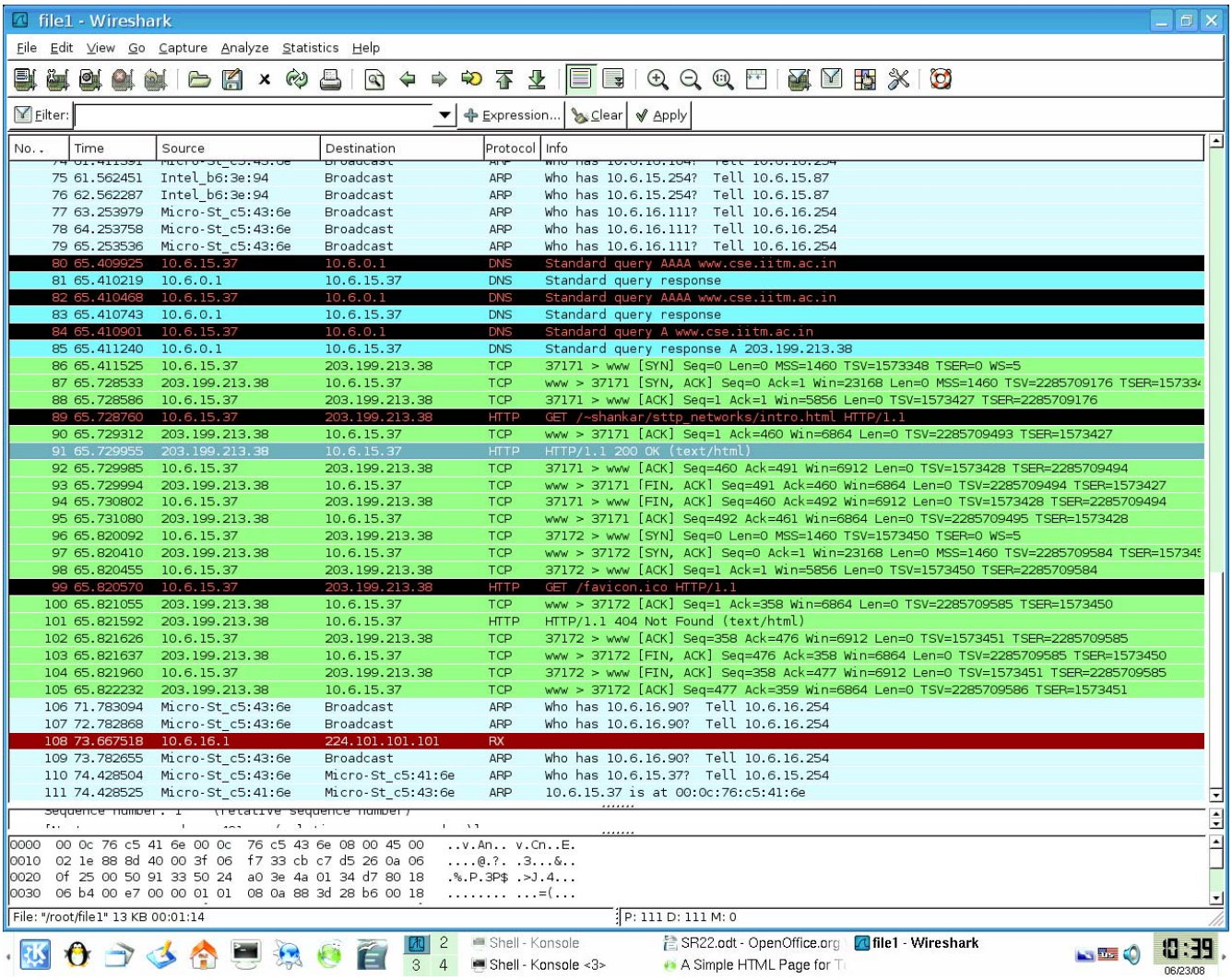


Exercise- 1

Now close the previous wire shark clear all the cache of browser selecting tools, clear private data, start wireshark again and then open the following website which will transfer only one packet.

Http://www.cse.iitm.ac.in/~shankar/sttp_networks/intro.html

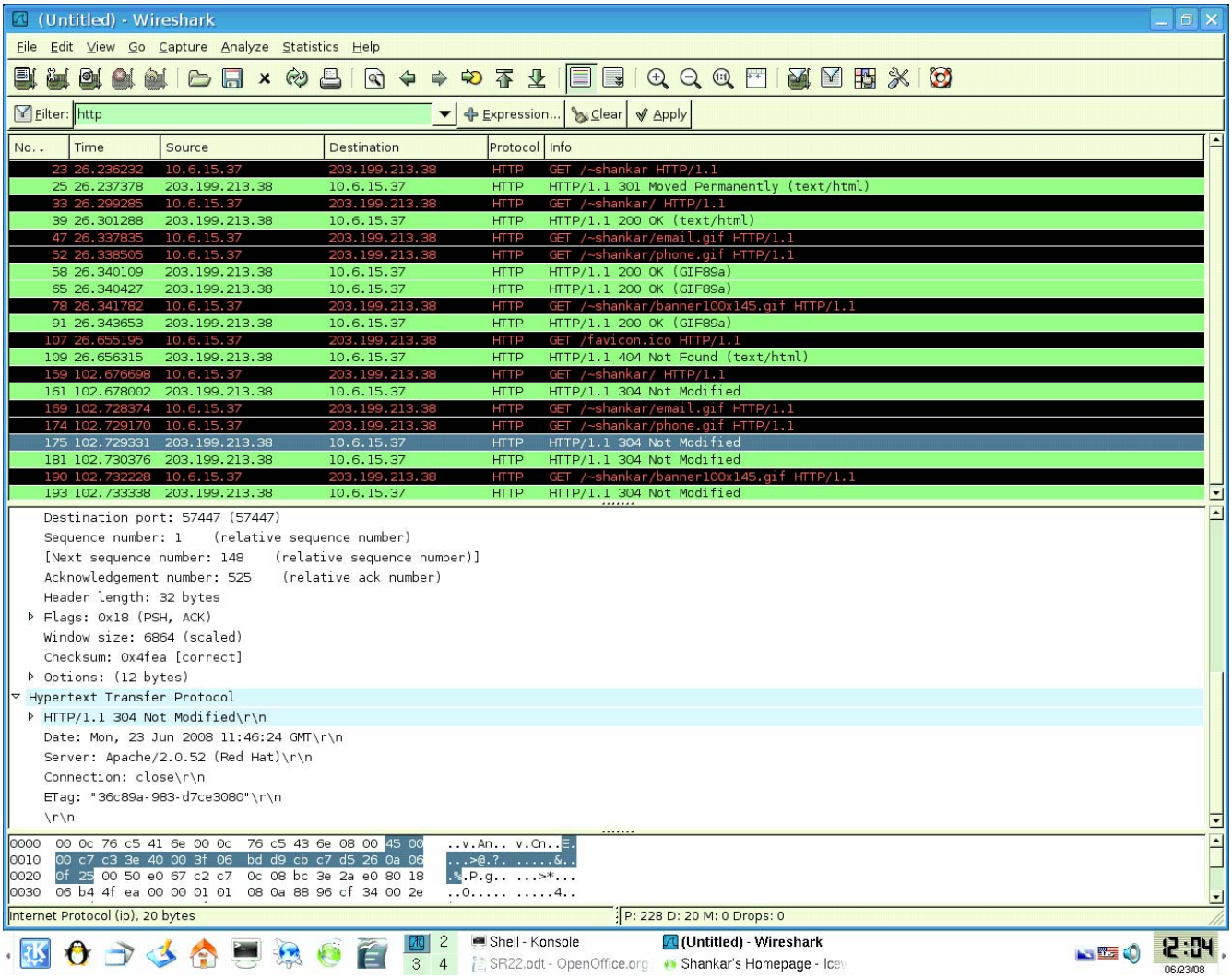
Now observe the output of tcp, we can see that the connection is first established using syn, syn ack is sent and data is transferred before terminating.



Exercise- 2

Now close the previous wireshark clear all the cache of browser selecting tools, clear private data, start wireshark again and then open the following website which will transfer only one packet, and then after some time refresh the same page again (select view and click on reload).

Http://www.cse.iitm.ac.in/~shankar/sttp_networks/intro.html



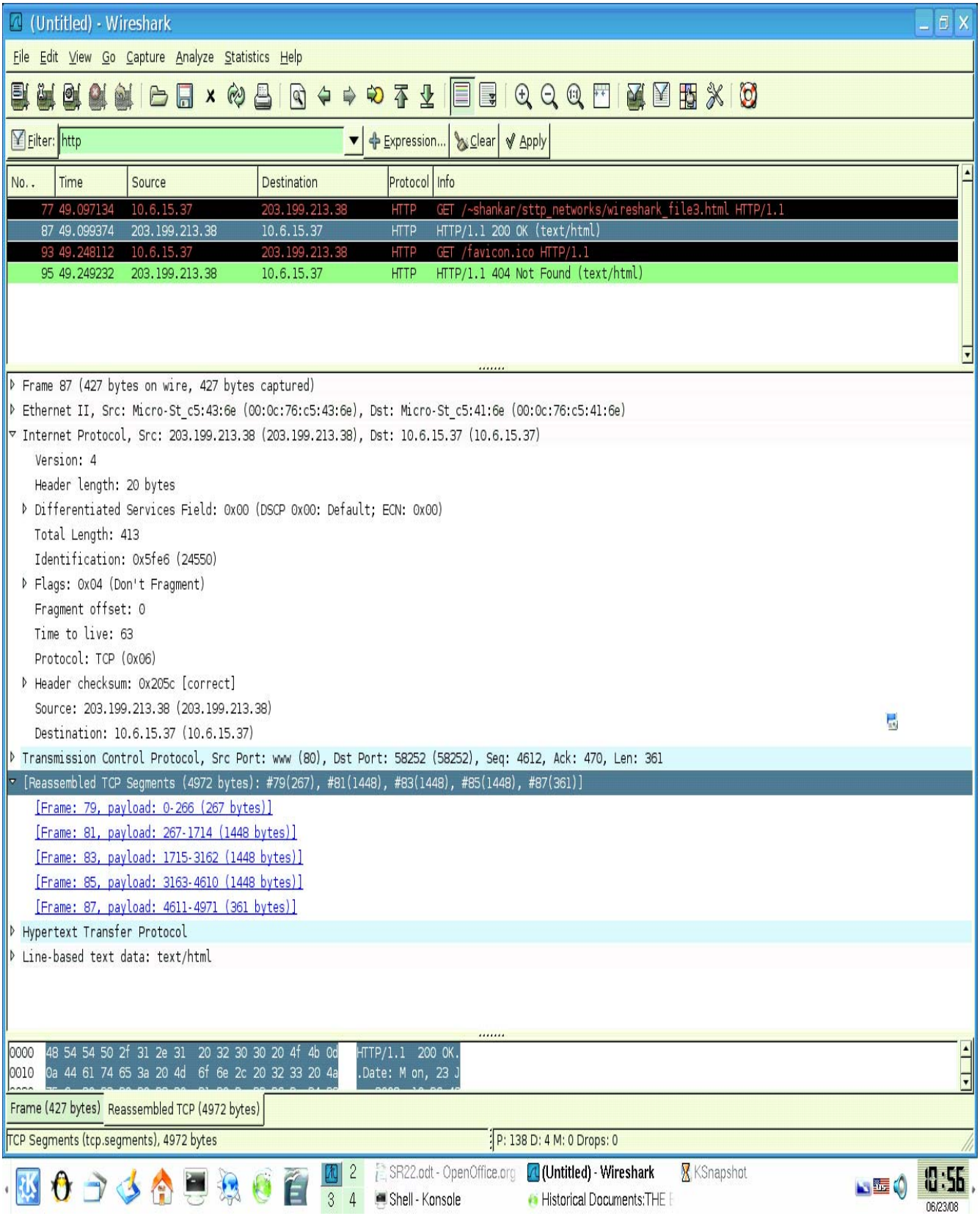
You can see HTTP's NOT_MODIFIED part in action. You will notice that the browser realized that it has a old copy and asks the server to serve the page only if the server has a copy newer than a particular date. The server now will just reply back with Not Modified message that the browser takes and does not re-fetch the webpage.

Exercise- 3

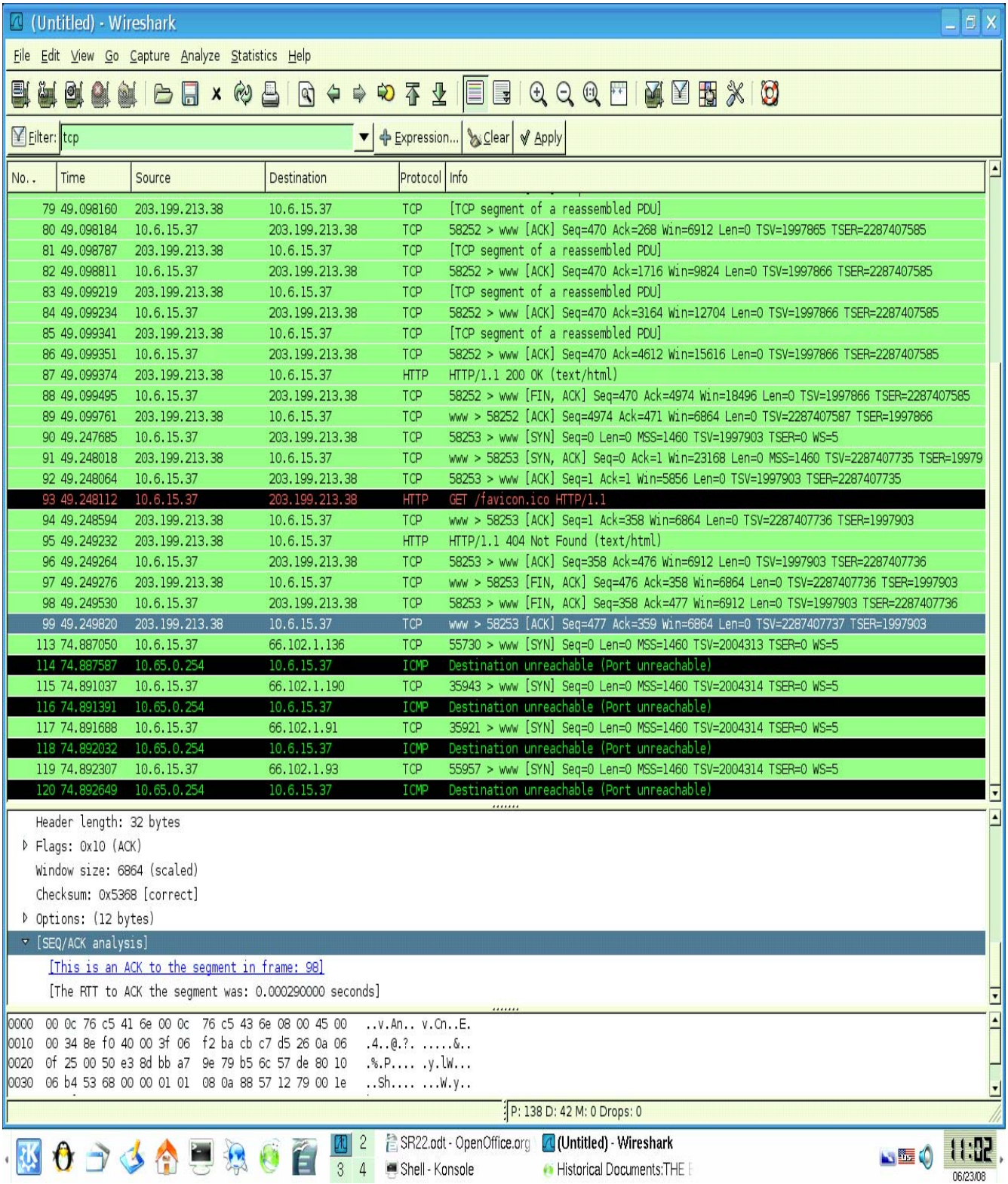
Now close the previous wireshark clear all the cache of browser, start wireshark again and then open the following website which has huge amount of data compared to the previous one and observe the RTT field.

Http://www.cse.iitm.ac.in/~shankar/sttp_networks/sttp_networks/wireshark_file3.html

Snapshot below for exercise 3 which shows reassembled packets.

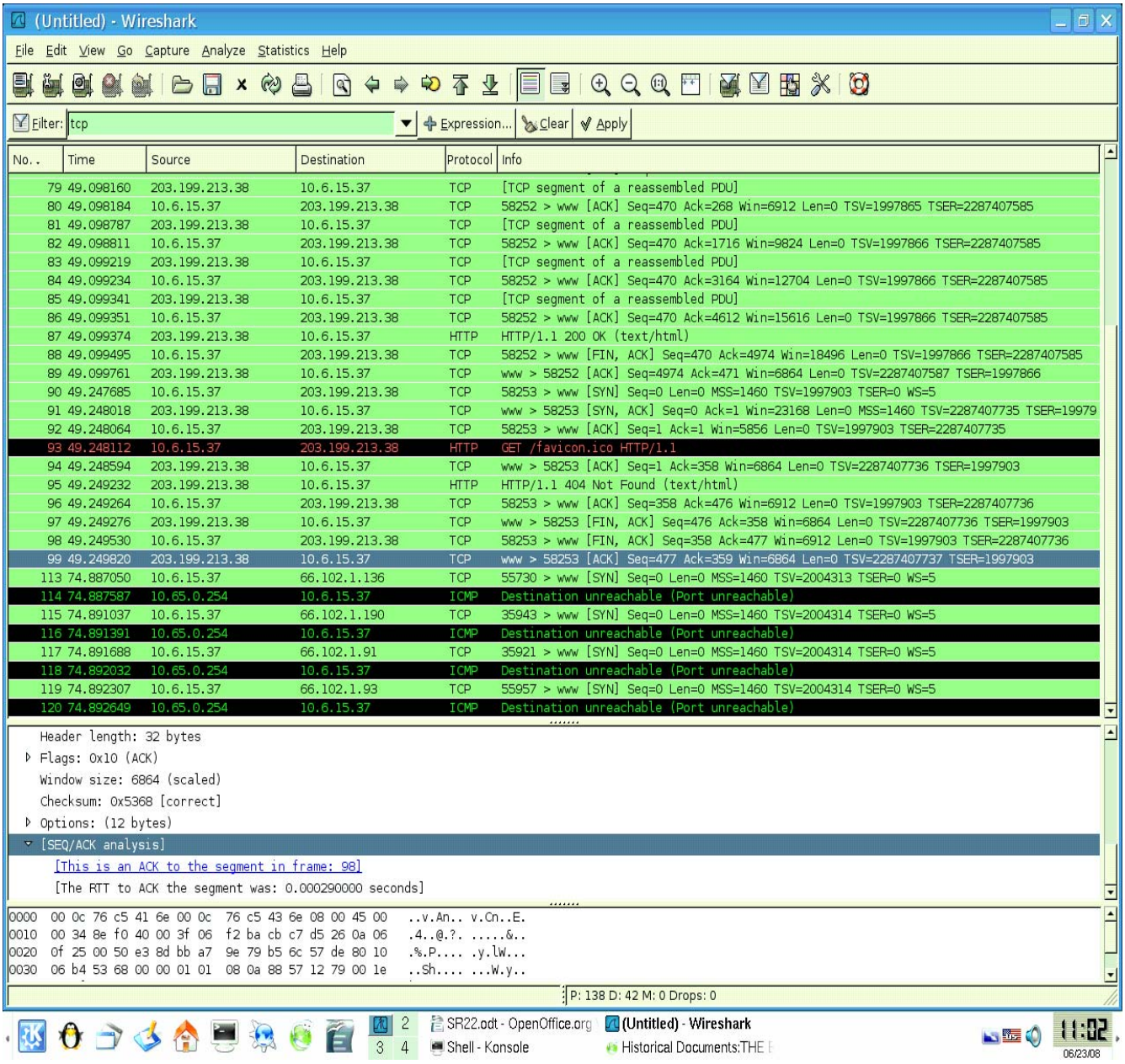

(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter: http
Expression... Clear Apply
No. . Time Source Destination Protocol Info
77 49.097134 10.6.15.37 203.199.213.38 HTTP GET /~shankar/sttp_networks/wireshark_file3.html HTTP/1.1
87 49.099374 203.199.213.38 10.6.15.37 HTTP HTTP/1.1 200 OK (text/html)
93 49.248112 10.6.15.37 203.199.213.38 HTTP GET /favicon.ico HTTP/1.1
95 49.249232 203.199.213.38 10.6.15.37 HTTP HTTP/1.1 404 Not Found (text/html)
Frame 87 (427 bytes on wire, 427 bytes captured)
Ethernet II, Src: Micro-St_c5:43:6e (00:0c:76:c5:43:6e), Dst: Micro-St_c5:41:6e (00:0c:76:c5:41:6e)
Internet Protocol, Src: 203.199.213.38 (203.199.213.38), Dst: 10.6.15.37 (10.6.15.37)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
Total Length: 413
Identification: 0x5fe6 (24550)
Flags: 0x04 (Don't Fragment)
Fragment offset: 0
Time to live: 63
Protocol: TCP (0x06)
Header checksum: 0x205c [correct]
Source: 203.199.213.38 (203.199.213.38)
Destination: 10.6.15.37 (10.6.15.37)
Transmission Control Protocol, Src Port: www (80), Dst Port: 58252 (58252), Seq: 4612, Ack: 470, Len: 361
[Reassembled TCP Segments (4972 bytes): #79(267), #81(1448), #83(1448), #85(1448), #87(361)]
[Frame: 79, payload: 0-266 (267 bytes)]
[Frame: 81, payload: 267-1714 (1448 bytes)]
[Frame: 83, payload: 1715-3162 (1448 bytes)]
[Frame: 85, payload: 3163-4610 (1448 bytes)]
[Frame: 87, payload: 4611-4971 (361 bytes)]
Hypertext Transfer Protocol
Line-based text data: text/html
0000 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d HTTP/1.1 200 OK.
0010 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 33 20 4a .Date: M on, 23 J
Frame (427 bytes) Reassembled TCP (4972 bytes)
TCP Segments (tcp.segments), 4972 bytes
P: 138 D: 4 M: 0 Drops: 0
SR22.odt - OpenOffice.org
(Untitled) - Wireshark
KSnapshot
Shell - Konsole
Historical Documents:THE
10:56
06/23/08

To find RTT observe the SEQ/ACK analysis in Window 2 of Wireshark under TCP.



Exercise- 4

Now close the previous wireshark clear all the cache of browser selecting tools, clear private data, start wireshark again and then open the following website which has text data, and two gif files.



Now observe that there will be 3 separate tcp connections opened to transfer text data, gif file1 and gif file2.

Lab Session 7
Firewalls Using IPTables
25 June 2008

In this lab, you will learn how to configure a firewall using IPTables and the basics of nmap.

Start with configuring the IP address and default gateway using

```
knoppix@Knoppix:~$ ifconfig eth0 10.6.15.37 netmask 255.255.255.0
knoppix@Knoppix:~$ route add -net default gateway 10.6.15.254 netmask 255.255.255.0
```

IPTables is a firewall application that is popular with many system admins. Check the contents of the iptables.

```
#iptables -L

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

iptables –L lists all the rules that are currently in place to filter packets.

General syntax:-
iptables [-t tables] command [match] [target/action]

Example :

```
iptables -A INPUT -p tcp --dport 23 -j DROP
```

First create the ssh server using

iptables has three tables, Filtering, NAT and Mangle. By default, Filtering table is used and this lab exercise is all done on the filtering table. In all the commands below, the –t option is never exercised.

For filters, there are three chains available in every firewall : INPUT, OUTPUT and FORWARDING. INPUT and OUTPUT control traffic to/from the firewall and FORWARDING table deals with data transfer to/from the green network from/to the red network.

To start with, let us explore the INPUT chain. We will start ssh service and add a rule that prohibits ssh connections using firewalls.

```
root@Knoppix:/ramdisk/home/knoppix# /etc/init.d/ssh start
Generating SSH1 RSA host key...
Generating public/private rsa1 key pair.
Your identification has been saved in /etc/ssh/ssh_host_key.
Your public key has been saved in /etc/ssh/ssh_host_key.pub.
The key fingerprint is:
f0:3b:65:2d:06:57:28:80:92:86:19:b0:51:c4:cf:ee
Generating SSH RSA host key...
Generating public/private rsa key pair.
Your identification has been saved in /etc/ssh/ssh_host_rsa_key.
Your public key has been saved in /etc/ssh/ssh_host_rsa_key.pub.
The key fingerprint is:
5a:27:04:9b:be:35:a9:04:2d:07:5b:d2:fd:c8:93:ca
Generating SSH2 DSA host key...
Generating public/private dsa key pair.
Your identification has been saved in /etc/ssh/ssh_host_dsa_key.
Your public key has been saved in /etc/ssh/ssh_host_dsa_key.pub.
The key fingerprint is:
99:c9:2f:ca:4f:4f:7b:45:d6:c4:5c:d7:b1:4a:bd:a4
Starting OpenBSD Secure Shell server: sshd.
```

Without firewall, the remote login works as imagined:

```
root@Knoppix:/ramdisk/home/knoppix# ssh root@10.6.15.37
The authenticity of host '10.6.15.37 (10.6.15.37)' can't be established.
RSA key fingerprint is 5a:27:04:9b:be:35:a9:04:2d:07:5b:d2:fd:c8:93:ca.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.6.15.37' (RSA) to the list of known hosts.
Password:
```

Now give the command to drop the ssh connection, the port is 22. The command below says that the input for the ssh with the port number 22 will be quietly dropped.

root@Knoppix:/ramdisk/home/knoppix# iptables -A INPUT -p tcp --dport 22 -j DROP

After the DROP, try connecting to the SSH server:

root@Knoppix:/ramdisk/home/knoppix# ssh root@10.6.15.37

You will observe that the server never responds back at all.

Remove the DROP rule from the INPUT chain using
root@Knoppix:/ramdisk/home/knoppix# iptables -D INPUT -p tcp --dport 22 -j DROP

Now try to add a rule which REJECTs the connections rather than DROP

root@Knoppix:/ramdisk/home/knoppix# iptables -A INPUT -p tcp --dport 22 -j REJECT
root@Knoppix:/ramdisk/home/knoppix# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination

REJECT tcp -- anywhere anywhere tcp dpt:ssh reject-with icmp-port-unreachable

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

Checking back on the server for SSH connections, we get

root@Knoppix:/ramdisk/home/knoppix# ssh root@10.6.15.37
ssh: connect to host 10.6.15.37 port 22: Connection refused

Now we can see the ICMP message as shown above. If a server refuses connection, it could be that the firewall is blocking it or the SSH server is simply not listening in. Ensure that the SSH server is indeed running and the firewall is the one which is rejecting connections, check that ssh port 22 is still open using the netstat command

root@Knoppix:/ramdisk/home/knoppix# netstat -tap
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
PID/Program name
tcp 0 0 *:bootpc *:* LISTEN 2 922/pump
tcp 0 0 *:x11 *:* LISTEN 3 480/Xorg
tcp6 0 0 *:x11 *:* LISTEN 3 480/Xorg
tcp6 0 0 *:ssh *:* LISTEN 4 029/sshd

In iptables, the actions taken happen according to the order in which rules are written:

root@Knoppix:/ramdisk/home/knoppix# iptables -A INPUT -p tcp --dport 22 -j REJECT
root@Knoppix:/ramdisk/home/knoppix# iptables -A INPUT -p tcp --dport 22 -j DROP
root@Knoppix:/ramdisk/home/knoppix# ssh root@10.6.15.37
ssh: connect to host 10.6.15.37 port 22: Connection refused
Now it rejects the packets

We can delete all the rules in the iptable using
root@Knoppix:/ramdisk/home/knoppix# iptables -F
Here -F stands for flush

Now change the order of commands and observe the output.
root@Knoppix:/ramdisk/home/knoppix# iptables -A INPUT -p tcp --dport 22 -j DROP
root@Knoppix:/ramdisk/home/knoppix# iptables -A INPUT -p tcp --dport 22 -j REJECT
root@Knoppix:/ramdisk/home/knoppix# ssh root@10.6.15.37

Now it drops the packets

This is because iptable matches the first rule, if it does not match it tries to match the second and so on otherwise it tries to match with the default which is accept.

7.2 More on Matches

Previously, the configuration was denying SSH connections to every host. We will now configure the firewall to deny connections only to a specific host.

Suppose my system is 10.6.15.37 and my neighbor's ip is 10.6.15.36. Ask the neighbor to add a rule to deny connections coming from your host. The neighbor would have to execute the following:

iptables -A INPUT -s 10.6.15.37 -p tcp --dport 22 -j DROP

and i try to login to ssh again then we get

root@Knoppix:/ramdisk/home/knoppix# ssh root@10.6.15.36
--blank--

Now suppose he wants to block all the systems in the subnet then he can do so using

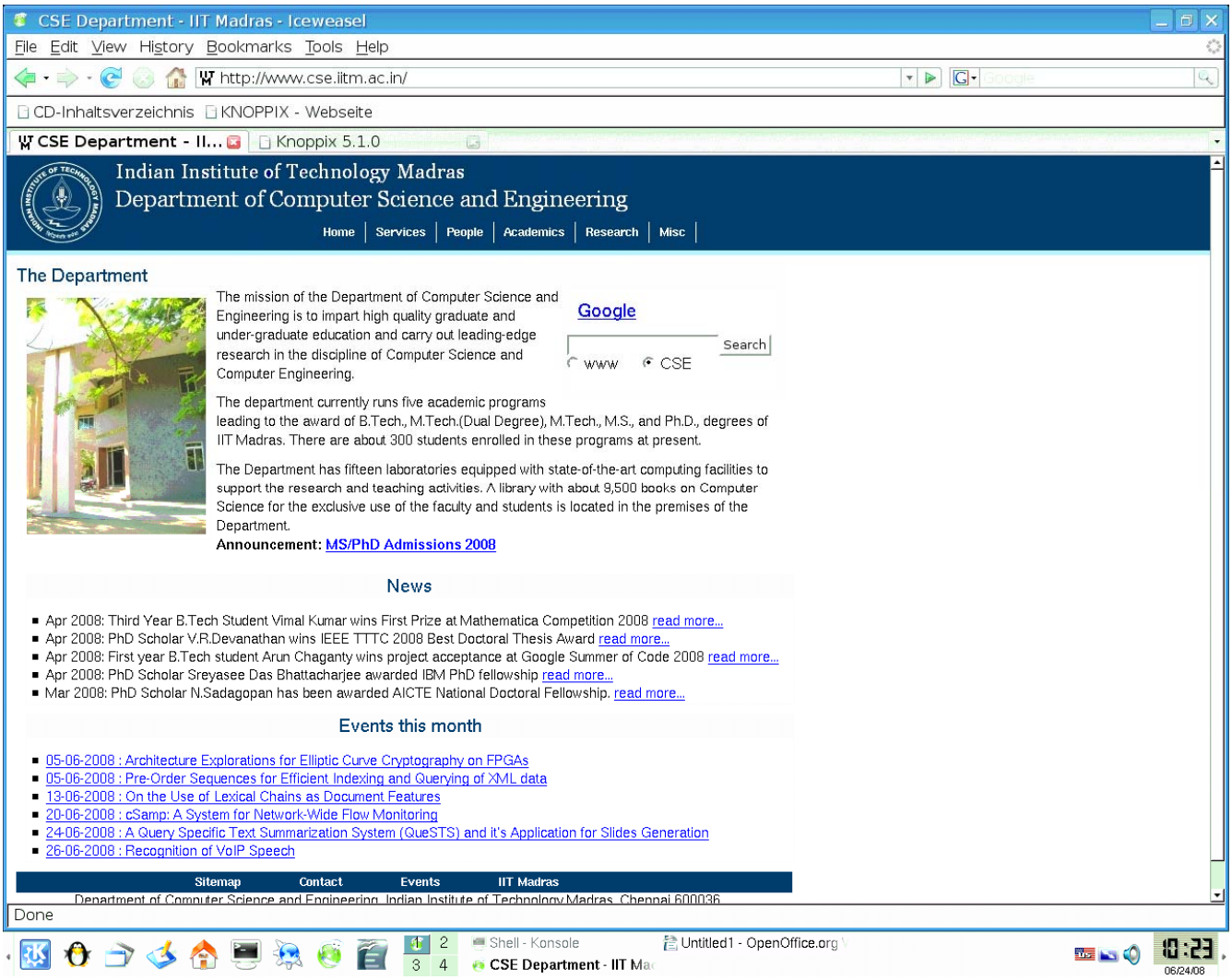
iptables -A INPUT -s 10.6.15.36/24 -p tcp --dport 22 -j DROP

Here give one of the ipaddress of the subnet say 10.6.15.36 and subnet mask as /24 which means taking the 32 bit IP address and mask with 255.255.255.0 thereby dropping all the packets from the subnet 10.6.15.0

Suppose instead of /24 if we had 10.6.15.36/32 then it would drop packets only from the specific machine 10.6.15.36.

7.3 OUTPUT Chain

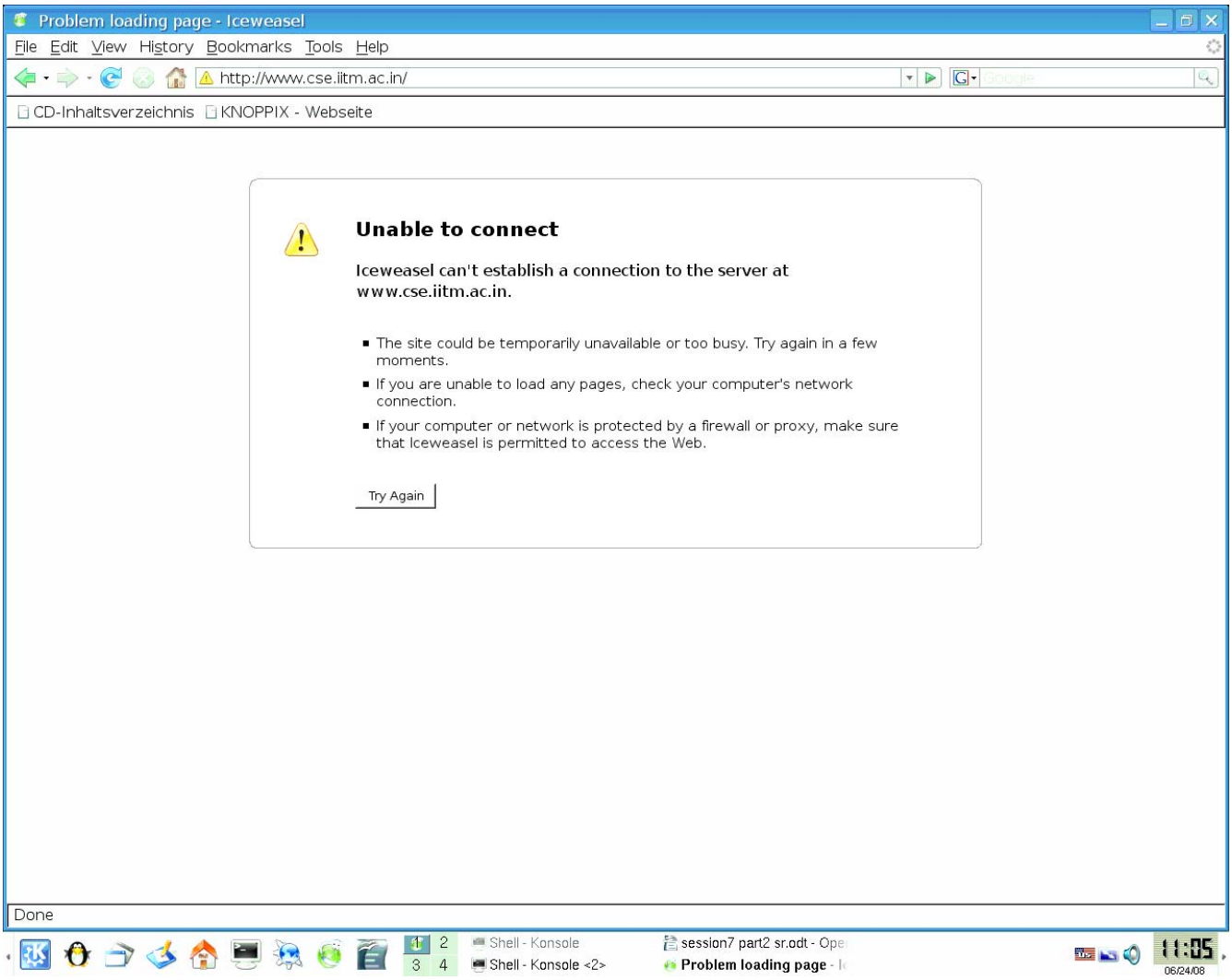
Let us now control the output connections. Open any website of your choice and ensure that the web site indeed loads on your browser. We try the http://www.cse.iitm.ac.in webpage here.



Now reject all the outgoing connections that connect to port 80 on any remote machine using the command below.

```
root@Knoppix:/ramdisk/home/knoppix# iptables -A OUTPUT -p tcp --dport 80 -j REJECT
```

The output is as shown below and the message on the browser shows that the server is reachable but the connection is refused.



Now execute this command to DROP the packets, for this first you should drop the rule to REJECT packets as the iptable starts matching from the first case

```
root@Knoppix:/ramdisk/home/knoppix# iptables -D OUTPUT -p tcp --dport 80 -j REJECT
root@Knoppix:/ramdisk/home/knoppix# iptables -A OUTPUT -p tcp –dport 80 -j DROP
```

Contacting the webserver now leaves the browser retrying the fetch of the webpage without showing any results.

Some of the commands that iptables take are:
Commands
-A : append a rule
-D : delete a rule
-R : replace a rule
-L : List the rules
-F : flush all rules

For Matches
-p : protocol

-s : source host/network
--dport : destination port
-d : destination host/network
--sport : sending port

Jump targets :

-j LOG

iptables can save the commands given to it in a file and read the rules from a file. **iptables-save and iptables –restore** are the two utilities that can come in handy for automation.

7.4 Connection Tracking:

To maintain states of various protocols, a firewall must be able to effectively track connections. ip_conntrack is a module that can be added to the kernel for such an operation :

root@Knoppix:/ramdisk/home/knoppix# modprobe ip_conntrack

Listing the connections shows many UDP and TCP connections below :

```
root@Knoppix:/ramdisk/home/knoppix# cat /proc/net/ip_conntrack
tcp      6 40 SYN_SENT src=10.6.15.38 dst=10.6.15.37 sport=34679 dport=22 packets=2
bytes=120 [UNREPLIED] src=10.6.15.37 dst=10.6.15.38 sport=22 dport=34679 packets=0
bytes=0 mark=0 secmark=0 use=1
tcp      6 53 TIME_WAIT src=10.6.15.38 dst=203.199.213.38 sport=38166 dport=80 packets=15
bytes=1214 src=203.199.213.38 dst=10.6.15.38 sport=80 dport=38166 packets=15 bytes=16785
[ASSURED] mark=0 secmark=0 use=1
tcp      6 53 SYN_SENT src=10.6.15.38 dst=209.85.153.104 sport=52711 dport=80 packets=1
bytes=60 [UNREPLIED] src=209.85.153.104 dst=10.6.15.38 sport=80 dport=52711 packets=0
bytes=0 mark=0 secmark=0 use=1
tcp      6 114 SYN_SENT src=10.6.15.38 dst=209.85.153.104 sport=52715 dport=80 packets=1
bytes=60 [UNREPLIED] src=209.85.153.104 dst=10.6.15.38 sport=80 dport=52715 packets=0
bytes=0 mark=0 secmark=0 use=1
tcp      6 111 SYN_SENT src=10.6.15.38 dst=10.6.15.37 sport=34687 dport=22 packets=2
bytes=120 [UNREPLIED] src=10.6.15.37 dst=10.6.15.38 sport=22 dport=34687 packets=0
bytes=0 mark=0 secmark=0 use=1
udp      17 113 src=10.6.15.38 dst=10.6.0.1 sport=32790 dport=53 packets=5 bytes=312
src=10.6.0.1 dst=10.6.15.38 sport=53 dport=32790 packets=5 bytes=818 [ASSURED] mark=0
secmark=0 use=1
tcp      6 38 SYN_SENT src=10.6.15.38 dst=72.14.221.190 sport=59282 dport=80 packets=1
bytes=60 [UNREPLIED] src=72.14.221.190 dst=10.6.15.38 sport=80 dport=59282 packets=0
bytes=0 mark=0 secmark=0 use=1
tcp      6 38 SYN_SENT src=10.6.15.38 dst=72.14.221.91 sport=49799 dport=80 packets=1
bytes=60 [UNREPLIED] src=72.14.221.91 dst=10.6.15.38 sport=80 dport=49799 packets=0
bytes=0 mark=0 secmark=0 use=1
udp      17 10 src=10.6.16.1 dst=10.6.15.38 sport=111 dport=732 packets=3 bytes=168
[UNREPLIED] src=10.6.15.38 dst=10.6.16.1 sport=732 dport=111 packets=0 bytes=0 mark=0
secmark=0 use=1
tcp      6 114 TIME_WAIT src=10.6.15.38 dst=203.199.213.38 sport=38170 dport=80
packets=16 bytes=1266 src=203.199.213.38 dst=10.6.15.38 sport=80 dport=38170 packets=16
bytes=16837 [ASSURED] mark=0 secmark=0 use=1
tcp      6 38 SYN_SENT src=10.6.15.38 dst=72.14.221.136 sport=52104 dport=80 packets=1
bytes=60 [UNREPLIED] src=72.14.221.136 dst=10.6.15.38 sport=80 dport=52104 packets=0
bytes=0 mark=0 secmark=0 use=1
tcp      6 97 SYN_SENT src=10.6.15.38 dst=10.6.15.37 sport=34686 dport=22 packets=2
bytes=120 [UNREPLIED] src=10.6.15.37 dst=10.6.15.38 sport=22 dport=34686 packets=0
bytes=0 mark=0 secmark=0 use=1
```

```
tcp      6 38 SYN_SENT src=10.6.15.38 dst=72.14.221.93 sport=52441 dport=80 packets=1
bytes=60 [UNREPLIED] src=72.14.221.93 dst=10.6.15.38 sport=80 dport=52441 packets=0
bytes=0 mark=0 secmark=0 use=1
```

You can inspect some important constants used by TCP by looking at variables in the netfilter portion of the kernel.

root@Knoppix:/ramdisk/home/knoppix# ls /proc/sys/net/ipv4/netfilter/ip_conntrack*
/proc/sys/net/ipv4/netfilter/ip_conntrack_buckets
/proc/sys/net/ipv4/netfilter/ip_conntrack_checksum
/proc/sys/net/ipv4/netfilter/ip_conntrack_count
/proc/sys/net/ipv4/netfilter/ip_conntrack_generic_timeout
/proc/sys/net/ipv4/netfilter/ip_conntrack_icmp_timeout
/proc/sys/net/ipv4/netfilter/ip_conntrack_log_invalid
/proc/sys/net/ipv4/netfilter/ip_conntrack_max
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_be_liberal
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_loose
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_max_retrans
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_close
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_close_wait
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_established
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_fin_wait
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_last_ack
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_max_retrans
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_syn_recv
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_syn_sent
/proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_time_wait
/proc/sys/net/ipv4/netfilter/ip_conntrack_udp_timeout
/proc/sys/net/ipv4/netfilter/ip_conntrack_udp_timeout_stream

We can see the output of various variables using the cat command as shown below.

```
root@Knoppix# cat proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_established
```
432000

```
root@Knoppix # cat /proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_max_retrans
```
3

```
root@Knoppix# cat /proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_close
```
10
```
root@Knoppix# cat /proc/sys/net/ipv4/netfilter/ip_conntrack_tcp_timeout_close_wait
```
60

7.5 nmap

Nmap is a useful Unix utility to perform various kinds of scans on hosts in a network. Let us first find out which of the hosts from 10.6.15.32 to 10.6.15.40 is turned on.

```
root@Knoppix:/ramdisk/home/knoppix# nmap -sP 10.6.15.32-40
```

```
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-06-24
10:44 EDT
Host agilam.cs.iitm.ernet.in (10.6.15.33) appears to be up.
MAC Address: 00:0C:76:C5:43:2A (Micro-star International CO.)
Host kani.cs.iitm.ernet.in (10.6.15.36) appears to be up.
MAC Address: 00:0C:76:C5:42:B9 (Micro-star International CO.)
Host kolam.cs.iitm.ernet.in (10.6.15.38) appears to be up.
Host kuyil.cs.iitm.ernet.in (10.6.15.39) appears to be up.
MAC Address: 00:0C:76:C5:41:9B (Micro-star International CO.)
Nmap finished: 9 IP addresses (4 hosts up) scanned in 7.529 seconds
```

Try

```
 root@Knoppix:/ramdisk/home/knoppix# nmap -sT -P0 10.6.15.50

Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-06-24
10:49 EDT
Interesting ports on veyyam.cs.iitm.ernet.in (10.6.15.50):
Not shown: 1677 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
68/tcp   open  dhcpc
6000/tcp open  X11

Nmap finished: 1 IP address (1 host up) scanned in 0.100 seconds
```

Nmap can perform a series of scans of ports ranging from the most legal (according to TCP protocol) to very illegal scans.

SYN scan : Send a SYN packet and see if a particular port is open. If the port is open you get SYN+ACK which is promptly ignored. Lack of response is recorded by Nmap as port being closed.

```
root@Knoppix:/ramdisk/home/knoppix# nmap -sS -P0 10.6.15.50
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2008-06-24
10:50 EDT
Interesting ports on veyyam.cs.iitm.ernet.in (10.6.15.50):
Not shown: 1677 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
68/tcp   open  dhcpc
6000/tcp open  X11
MAC Address: 00:0C:76:C5:40:EB (Micro-star International CO.)
Nmap finished: 1 IP address (1 host up) scanned in 0.495 seconds
```

ACK Scan : Send an ACK for a connection that was never made to begin with. This is illegal according to TCP protocol and such a packet will receive TCP_RESET message. ACK scan is performed using –sA option.

root@Knoppix:/ramdisk/home/knoppix# nmap -sA -P0 10.6.15.36

Starting Nmap 4.11 (http://www.insecure.org/nmap/) at 2008-06-24 10:55 EDT
All 1680 scanned ports on kani.cs.iitm.ernet.in (10.6.15.36) are UNfiltered
MAC Address: 00:0C:76:C5:42:B9 (Micro-star International CO.)
Nmap finished: 1 IP address (1 host up) scanned in 1.577 seconds

OS Scan: This is useful for OS discovery.

root@Knoppix:/ramdisk/home/knoppix# nmap -O 10.6.15.36

Starting Nmap 4.11 (http://www.insecure.org/nmap/) at 2008-06-24 10:58 EDT
Interesting ports on kani.cs.iitm.ernet.in (10.6.15.36):
Not shown: 1678 closed ports
PORT STATE SERVICE
68/tcp open dhcpc
6000/tcp open X11
MAC Address: 00:0C:76:C5:42:B9 (Micro-star International CO.)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.0 - 2.5.20, Linux 2.4.7 - 2.6.11

Nmap finished: 1 IP address (1 host up) scanned in 2.630 seconds

You can also send a series of flags using the –scanflags option.

root@Knoppix:/ramdisk/home/knoppix# nmap --scanflags FINSYN 10.6.15.36
Starting Nmap 4.11 (http://www.insecure.org/nmap/) at 2008-06-24 11:09 EDT
sendto in send_ip_packet: sendto(5, packet, 44, 0, 10.6.15.36, 16) => Operation not permitted
sendto in send_ip_packet: sendto(5, packet, 44, 0, 10.6.15.36, 16) => Operation not permitted
sendto in send_ip_packet: sendto(5, packet, 44, 0, 10.6.15.36, 16) => Operation not permitted
Interesting ports on kani.cs.iitm.ernet.in (10.6.15.36):
Not shown: 1677 closed ports
PORT STATE SERVICE
68/tcp open dhcpc
80/tcp filtered http
6000/tcp open X11
MAC Address: 00:0C:76:C5:42:B9 (Micro-star International CO.)
Nmap finished: 1 IP address (1 host up) scanned in 2.751 seconds

For more details read the man pages of nmap.

Lab Session 8
Apache, SSL and Proxy using Squid
25 June 2008

In debian based systems, to install apache2 run the following command

```
$ apt-get install apache2
```

First check the status of the apache2 server using the following command. (This is for knoppix)

```
root# cat /etc/default/apache2
# 0 = start on boot; 1 = don't start on boot
```

Change `NO_START to 0.`

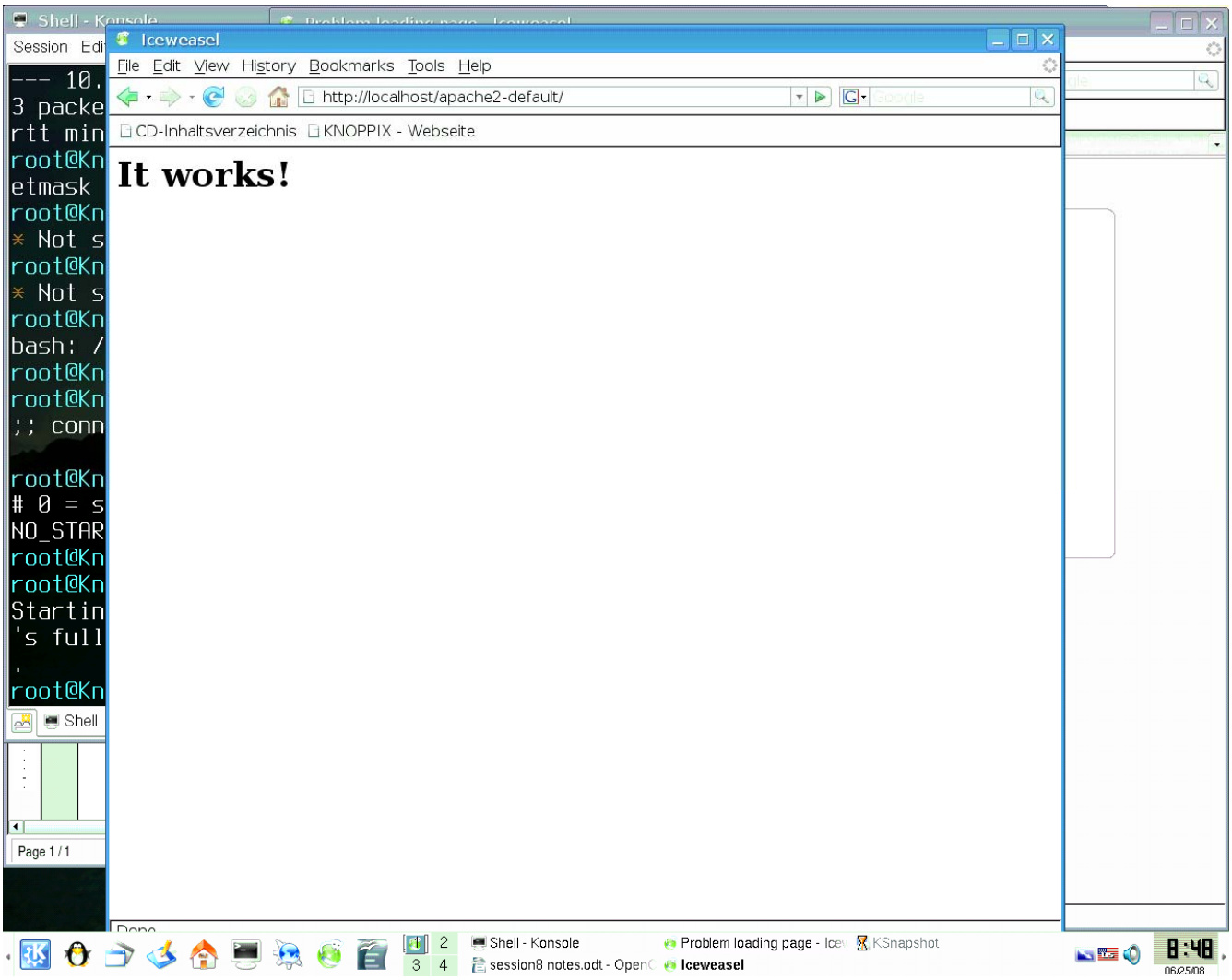
Save this file and now run the apache2 using this command.

```
root# /etc/init.d/apache2 start
```

Starting web server (apache2)...apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.0.1 for ServerName .

Now open the browser and type the url as
http://localhost

We get a single page output which says “It Works”.



Now go to this folder

```
root# cd /etc/apache2
```

All apache related configuration is stored in this folder

```
root:/etc/apache2# ls
apache2.conf  envvars     mods-available  ports.conf       sites-enabled
conf.d        httpd.conf  mods-enabled    sites-available  ssl
```

apache2.conf – This file has core apache configurations like number of connections it needs to keep alive at any point, etc..

ports.conf: List of ports for apache to listen.
root@Knoppix# vi ports.conf
Listen 80

There is a directory called mods_available – these are modules available. Basic apache is a only a static web server. For additional functionality modules need to be loaded. For examaple php4.conf and php4.load are the files for php4 module

```
root@Knoppix:/etc/apache2# cd mods-available
root@Knoppix:/etc/apache2/mods-available# ls
actions.load          cern_meta.load     file_cache.load    proxy_balancer.load
alias.load            cgi.load           filter.load        proxy_connect.load
asis.load             cgid.conf          headers.load       proxy_ftp.load
auth_basic.load       cgid.load          ident.load         proxy_http.load
auth_digest.load      charset_lite.load  imagemap.load      rewrite.load
authn_alias.load      dav.load           include.load       setenvif.load
authn_anon.load       dav_fs.conf        info.load          sick-hack-to-update-modules
authn_dbd.load        dav_fs.load        ldap.load          speling.load
authn_dbm.load        dav_lock.load      log_forensic.load  ssl.conf
authn_default.load    dbd.load           mem_cache.conf     ssl.load
authn_file.load       deflate.conf       mem_cache.load     status.load
authnz_ldap.load      deflate.load       mime.load          suexec.load
authz_dbm.load        dir.conf           mime_magic.conf    unique_id.load
authz_default.load    dir.load           mime_magic.load    userdir.conf
authz_groupfile.load  disk_cache.conf    negotiation.load   userdir.load
authz_host.load       disk_cache.load    php4.conf          usertrack.load
authz_owner.load      dump_io.load       php4.load          version.load
authz_user.load       env.load           proxy.conf         vhost_alias.load
autoindex.load        expires.load       proxy.load
cache.load            ext_filter.load    proxy_ajp.load
```

In debian based systems to install php4 module use this
$ apt-get install libapache2-mod-php4

Running the above command will only put the php4 in the mods-available folder.

To enable these modules we should put it in mods-enabled directory to do this we should do
$a2enmod php4
(a2enmod is apache2 enable module)

```
knoppix@Knoppix:/etc/apache2$ cd mods-enabled
knoppix@Knoppix:/etc/apache2/mods-enabled$ ls
alias.load            authz_host.load  env.load          setenvif.load
auth_basic.load       authz_user.load  mime.load         status.load
authn_file.load       autoindex.load   negotiation.load
authz_default.load    dir.conf         php4.conf
authz_groupfile.load  dir.load         php4.load
```

```
knoppix@Knoppix:~$ ls -l /etc/apache2/mods-enabled
```

```
total 0
lrwxrwxrwx 1 root root 28 Dec 18  2006 alias.load -> ../mods-available/alias.load
lrwxrwxrwx 1 root root 33 Dec 18  2006 auth_basic.load -> ../mods-
available/auth_basic.load
lrwxrwxrwx 1 root root 33 Dec 18  2006 authn_file.load -> ../mods-
available/authn_file.load
lrwxrwxrwx 1 root root 36 Dec 18  2006 authz_default.load -> ../mods-
available/authz_default.load
lrwxrwxrwx 1 root root 38 Dec 18  2006 authz_groupfile.load -> ../mods-available
/authz_groupfile.load
lrwxrwxrwx 1 root root 33 Dec 18  2006 authz_host.load -> ../mods-
available/authz_host.load
lrwxrwxrwx 1 root root 33 Dec 18  2006 authz_user.load -> ../mods-
available/authz_user.load
lrwxrwxrwx 1 root root 32 Dec 18  2006 autoindex.load -> ../mods-
available/autoindex.load
lrwxrwxrwx 1 root root 26 Dec 18  2006 dir.conf -> ../mods-available/dir.conf
lrwxrwxrwx 1 root root 26 Dec 18  2006 dir.load -> ../mods-available/dir.load
lrwxrwxrwx 1 root root 26 Dec 18  2006 env.load -> ../mods-available/env.load
lrwxrwxrwx 1 root root 27 Dec 18  2006 mime.load -> ../mods-available/mime.load
lrwxrwxrwx 1 root root 34 Dec 18  2006 negotiation.load -> ../mods-
available/negotiation.load
lrwxrwxrwx 1 root root 27 Dec 18  2006 php4.conf -> ../mods-available/php4.conf
lrwxrwxrwx 1 root root 27 Dec 18  2006 php4.load -> ../mods-available/php4.load
lrwxrwxrwx 1 root root 31 Dec 18  2006 setenvif.load -> ../mods-available/setenvif.load
lrwxrwxrwx 1 root root 29 Dec 18  2006 status.load -> ../mods-available/status.load
```

Similarly to disable the module do

```
root@Knoppix:/UNIONFS/etc/apache2# a2dismod php4
Module php4 disabled; run /etc/init.d/apache2 force-reload to fully disable.
```

For the changes to take place restart the server.

For changes to take place we should do

```
root@Knoppix:/UNIONFS/etc/apache2# /etc/init.d/apache2 force-reload
Forcing reload of web server (apache2)...apache2: Could not reliably determine the
server's fully qualified domain name, using 127.0.0.1 for ServerName
apache2: Could not reliably determine the server's fully qualified domain name, using
127.0.0.1 for ServerName .
```

```
root@Knoppix# ls /etc/apache2/mods-enabled
alias.load       authz_default.load    authz_user.load  dir.load   negotiation.load
auth_basic.load  authz_groupfile.load  autoindex.load   env.load   setenvif.load
authn_file.load  authz_host.load       dir.conf         mime.load  status.load
```

```
root@Knoppix:/UNIONFS/etc/apache2# vi sites-available/default
NameVirtualHost *
<VirtualHost *>
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www/
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
                # This directive allows us to have apache2's default start page
                # in /apache2-default/, but still have / go to the right place
                RedirectMatch ^/$ /apache2-default/
        </Directory>

        ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
        <Directory "/usr/lib/cgi-bin">
                AllowOverride None
                Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
        </Directory>

        ErrorLog /var/log/apache2/error.log

        # Possible values include: debug, info, notice, warn, error, crit,
        # alert, emerg.
        LogLevel warn

        CustomLog /var/log/apache2/access.log combined
        ServerSignature On

    Alias /doc/ "/usr/share/doc/"
    <Directory "/usr/share/doc/">
        Options Indexes MultiViews FollowSymLinks
        AllowOverride None
        Order deny,allow
        Deny from all
```

```
        Allow from 127.0.0.0/255.0.0.0 ::1/128
    </Directory>

</VirtualHost>
```

Virtual host means in one machine we host two different websites..

Suppose we have one machine with ip 202.141.25.100 and we want to create 2 websites say dcf.iitm.com and syslab.iitm.com, each of it will be a virtualhost.

When a request comes for these sites then first request is taken by the apache server which checks the website address and opens the page corresponding to it.

```
  DocumentRoot /var/www
```

```
root@Knoppix:/ramdisk/home/knoppix# cd /var/www/
root@Knoppix:/var/www# ls
apache2-default  index.html  search.html  users

root@Knoppix:/var/www# cd /var/www/apache2-default
root@Knoppix:/var/www/apache2-default# ls
apache_pb.gif  apache_pb22.gif  apache_pb22_ani.gif
apache_pb.png  apache_pb22.png  index.html
```

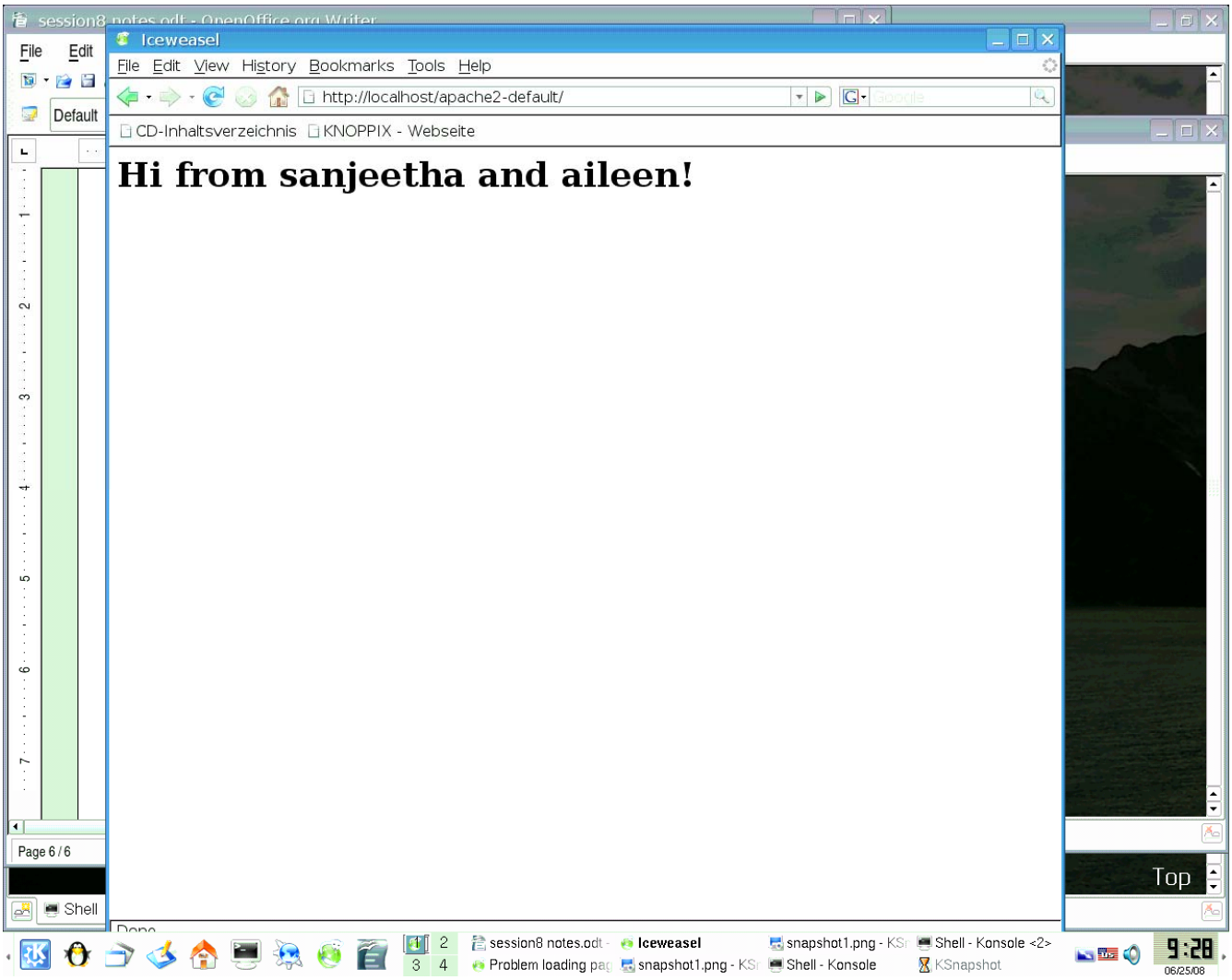
Now check the file index.html

```
root@Knoppix:/var/www/apache2-default# vi index.html
<html><body><h1>it works!</h1></body></html>
```

Now change it to your own words like

```
<html><body><h1>Hi from sanjeetha and aileen/h1></body></html>
```

now refresh the webpage localhost



If we want to have two different roots like /www/dcf amd /www/sys then we should write two codes for 2 virtual hosts in the root@Knoppix:/UNIONFS/etc/apache2# vi sites-available/default

and we should include the commands like
<virtualHost *>
Document Root /www/dcf/
serverName dcf.iitm.com
.
.
<\virtualHost *>

<virtualHost *>
Document Root /www/syslab/
serverName syslab.iitm.com.
.

.
<\virtualHost *>

We can authenticate the users trying to access apache
The command below gives various authorization modules that are available.

root@Knoppix:/UNIONFS/etc/apache2# ls mods-available/auth*
mods-available/auth_basic.load mods-available/authnz_ldap.load
mods-available/auth_digest.load mods-available/authz_dbm.load
mods-available/authn_alias.load mods-available/authz_default.load
mods-available/authn_anon.load mods-available/authz_groupfile.load
mods-available/authn_dbd.load mods-available/authz_host.load
mods-available/authn_dbm.load mods-available/authz_owner.load
mods-available/authn_default.load mods-available/authz_user.load
mods-available/authn_file.load

Directory-wise settings are given in this format:

```
<Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
        # This directive allows us to have apache2's default start page
        # in /apache2-default/, but still have / go to the right place
        RedirectMatch ^/$ /apache2-default/
 </Directory>
```

The partial output of root@Knoppix:/UNIONFS/etc/apache2# vi sites-available/default shown above tells the permissions set to the files in the document root directory /var/www.

To override global directory configuration using .htaccess files use:
AllowOverride All
To disable overriding use:
AllowOverride None

8.2 HTTPS

To setup https i.e., secure http:

Url will be like: https://www.foo.com

In firefox we can see that secure web pages url box becomes yellow
1) Http will send a certificate which is signed by some sites which confirm the authenticity.

Sometimes if the website does not have proper digital signatures, the browser informs the user

2) Openssl provides security between 2 hosts for a connection. i.e data like passwords will be encrypted.

To install it in debian based systems:
root@Knoppix:/UNIONFS/etc/apache2# apt-get install openssl ssl-cert
Reading package lists... Done
Building dependency tree... Done
openssl is already the newest version.
ssl-cert is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.

To get the certificate run this

root@Knoppix:/UNIONFS/etc/apache2# openssl req -new -x509 -days 365 -nodes \ -out /etc/apache2/apache.pem -keyout /etc/apache2/apache.pem
Generating a 1024 bit RSA private key
.++++++
.....++++++
writing new private key to '/etc/apache2/apache.pem'

You are about to be asked to enter information that will be incorporated
into your certificate request. What you are about to enter is what is called a Distinguished Name or a DN. There are quite a few fields but you can leave some blank
For some fields there will be a default value, If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:
Email Address []:

Enable the ssl module:
root@Knoppix:/UNIONFS/etc/apache2# a2enmod ssl
Module ssl installed; run /etc/init.d/apache2 force-reload to enable.

Now restart the server to see the changes
root@Knoppix:/UNIONFS/etc/apache2# /etc/init.d/apache2 restart

Now go to sites-available/default and add the following in the file as shown below
root@Knoppix:/UNIONFS/etc/apache2# vi /etc/apache2/sites-available/default
NameVirtualHost *:443

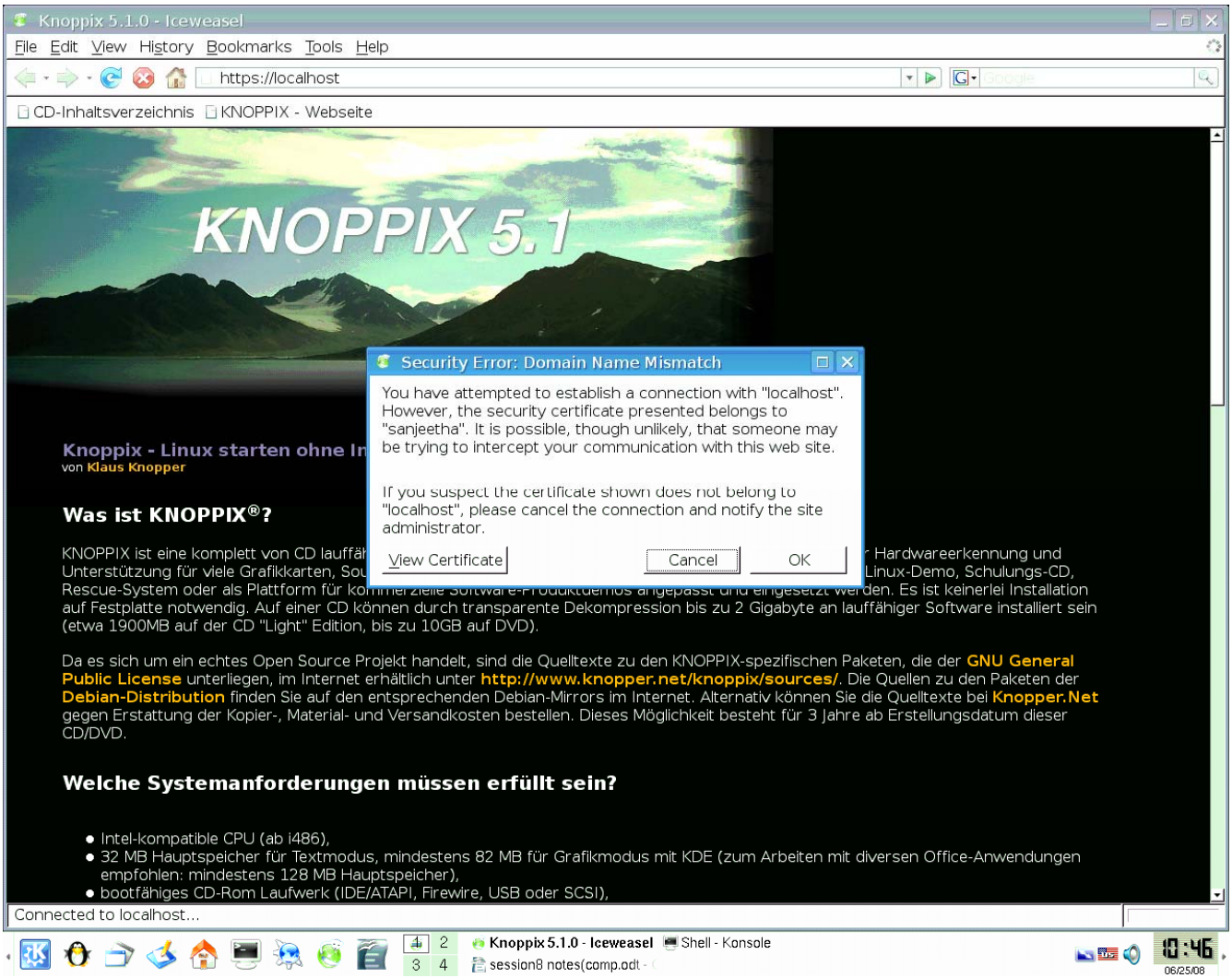
```
<VirtualHost *:443>
        SSLEngine on
        SSLCertificateFile /etc/apache2/apache.pem
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www/
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
                # This directive allows us to have apache2's default start page
                # in /apache2-default/, but still have / go to the right place
                RedirectMatch ^/$ /apache2-default/
 </Directory>
```

Observe that <VirtualHost *> is changed to <VirtualHost *:443> and add 2 lines
SSLEngine on
SSLCertificate /etc/apache2/apache.pem

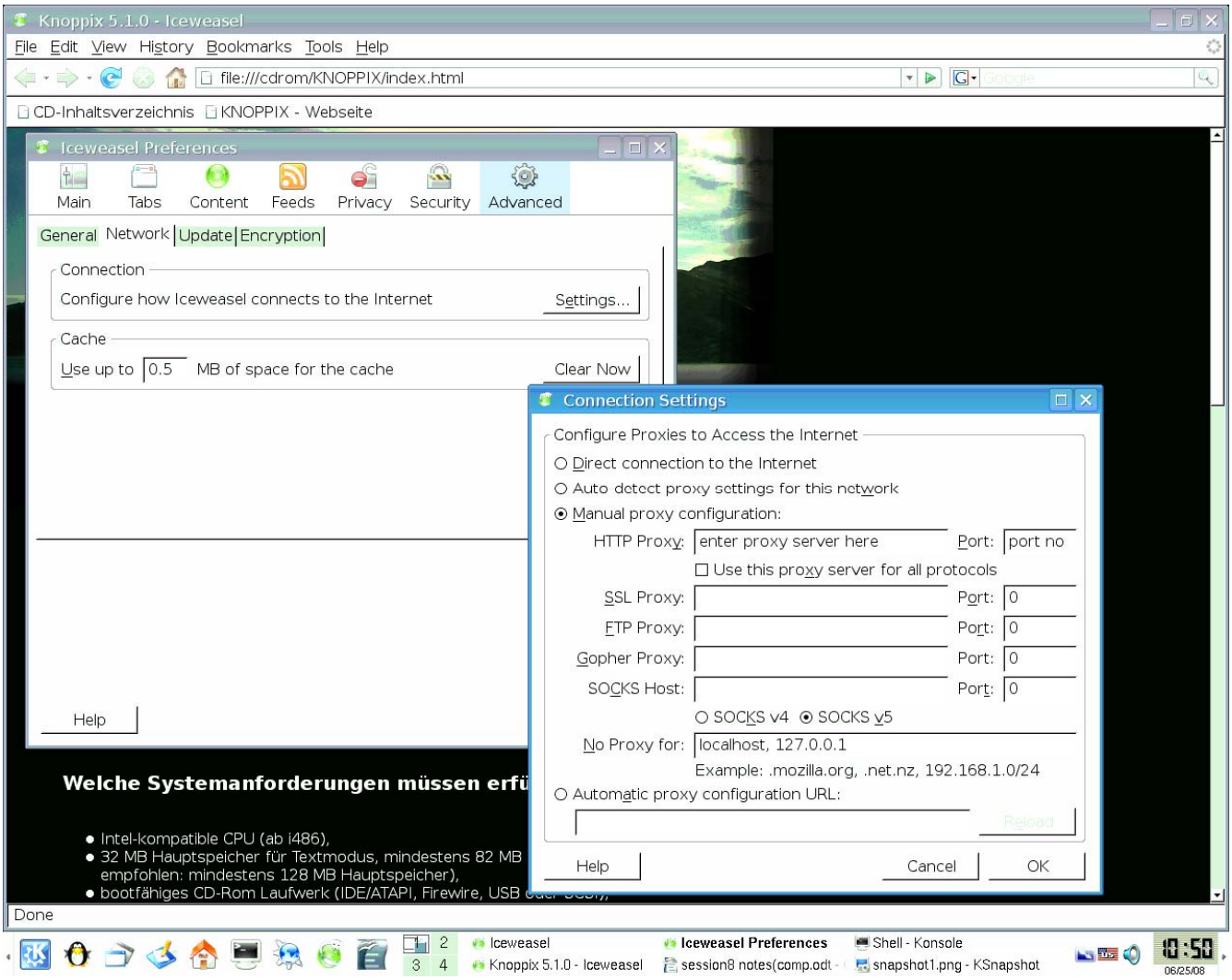
Now go to ports.conf and add the line Listen 443 as shown below

root@Knoppix:/UNIONFS/etc/apache2# vi /etc/apache2/ports.conf
Listen 80
Listen 443

Now we can check that https is running by typing https://localhost



Now we can see that the site is asking for the certificate of the website that has to be opened. A message appears on successful opening of the https connection and observe that the bar is yellow.



8.3 Squid Proxy server

Proxy server is useful for the

Caching of pages saves bandwidth

Imposing restrictions on the usage of the internet

To install squid use
root@Knoppix:/UNIONFS/etc/apache2# apt-get install squid
Reading package lists... Done
Building dependency tree... Done
squid is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgrade

Configuration file for this is /etc/squid/squid.conf

Suppose IP address of the proxy server is 192.168.2.5 to set the port for this server is 8080, change these values in squid.conf

http_port 192.168.2.5:8080

Suppose if anyone wants to use the internet they should give the proxy ip as 192.168.2.5 and 8080 as the port number.

We can put various restrictions using ACL (Access Control List). For example, students are allowed to access Internet between a given time say from 10pm to 12 pm.

acl network1 src 192.168.2.0/24
acl stands for access control list

Partial output of squid.conf

```
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl SSL_ports port 443 563
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443 563     # https, snews
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 631         # cups
acl Safe_ports port 777         # multiling http
acl Safe_ports port 901         # SWAT
acl purge method PURGE
acl CONNECT method CONNECT

http_access allow manager localhost
http_access deny manager
# Only allow purge requests from localhost
http_access allow purge localhost
http_access deny purge
# Deny requests to unknown ports
http_access deny !Safe_ports
# Deny CONNECT to other than SSL ports
http_access deny CONNECT !SSL_ports
#
# INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS
```

#
http_access allow localhost
And finally deny all other access to this proxy
http_access deny all

There are 2 types of proxy servers

1) Direct proxy
   The client users know about the proxy. The users have to explicitly enter the proxy ip address and the port number.
2) Transparent proxy
   The users will not be able to know that they are working on the proxy. To set up transparent proxy use iptables rules to forward all the packets coming to the gateway to send to the proxy.