Performance analysis of two secure tunneling mechanisms: IPSec VPN versus SSL VPN

Jérémie BOULANGER
Ensimag, Grenoble (France)

Bastien BAILLY
Ensimag, Grenoble (France)

June 17, 2009

Abstract

In this study, we are looking at performance of several solutions for IP tunnels to connect different entities of a firm. Latency, bandwidth and CPU load are aspects that we are focusing on. We will show that the choice of a solution rather than another depends mainly on the needed computing ressources and on the ease of implementation.

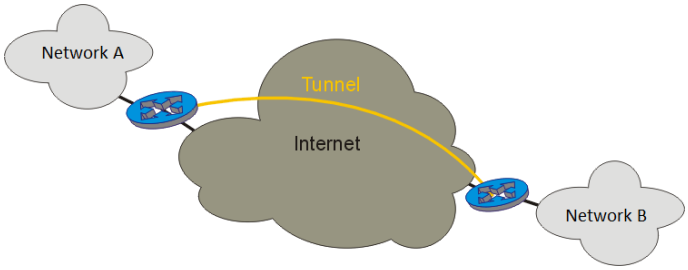

Figure 1 - A tunnel through the Internet

1 Context

Nowadays many companies have worldwide sites as well as mobile workers. In order to ease exchanges between these different entities, it is necessary to connect their networks. However, for economic reasons, each company can't develop its own physical network between each of its sites. It's necessary to use the network of a third party which will route data through the Internet. It is obvious that for a mobile client, the choice is even more justified. Then it's essential to have a secured transmission to protect the confidentiality of trade. To facilitate the interconnection of networks through the Internet, there are several options available such as an IPSec VPN (Virtual Private Network) or an SSL VPN, that both create a tunnel between two machines.

Each solution has its pros and cons, choosing one of these solutions must be made on a case by case according to the needs. The goal of this work is to give you the necessary information to choose the solution which is adapted for you.

2 Theory

In this section we present the theoretical aspects of the several methods that can be used to connect two remote sites. To begin we will recall here the structure of layers used for communication over networks and more specifically the TCP/IP protocols.

2.1 Networking layers and TCP/IP protocols

Communication between two computers is done by adding information to the data helping them to find their way through the network. This information is distributed by each layer that encapsulate the previous layers. Here is a presentation of the different network layers used for TCP/IP transmission.

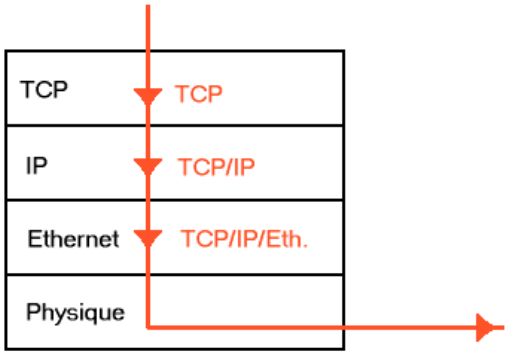

Figure 2 - Example of layers

The encapsulation of data creates a difference between the physical link and data rate because of the size of the headers that are added. For example: Ethernet link at 100Mb/s:

$$Dth = \frac{1518-18-20-20}{1518} * 100 = 96,18Mb/s$$

2.2 IPSec VPN

IPSec is based on a reencapsulation at the IP level. It adds to the original packet a IP header corresponding to the public exit of the tunnel. It's possible to add encryption and authentication (AH) which are not included in this study.

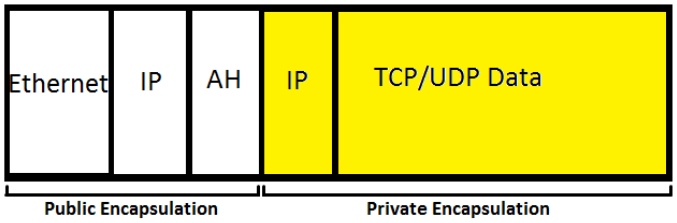

Figure 3 - Overhead due to IPSec

There is an additional loss due to the addition of this header (20 bytes).

For example: Ethernet link at 100Mb/s

$$Dth = \frac{1518-18-20-20-20}{1518} * 100 = 94,8Mb/s$$

2.3 SSL VPN

We examine here the case of OpenVPN, an open-source software used to create SSL tunnels reencapsulating IP into UDP.

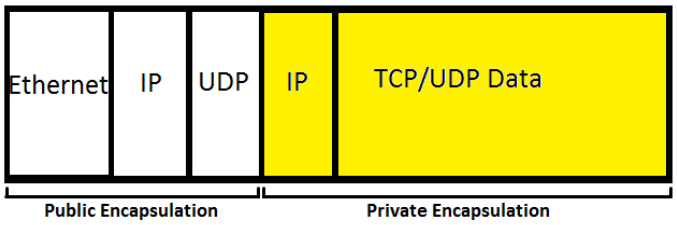

Figure 4 - Overhead due to OpenVPN

Here we lose another 8 bytes due to UDP header.

For example: Ethernet link at 100Mb/s
Without encryption:

$$Dth = \frac{1518-18-20-20-20-8}{1518} * 100 = 94,33Mb/s$$

With encryption:

$$Dth = \frac{1518-18-20-20-20-8-41}{1518} * 100 = 91,63Mb/s$$

We add in this case the authentication header of 41 bytes of OpenVPN

2.4 Experimentation rate

Experimentaly, we obtain a 94,1 Mb/s rate for a direct TCP connection, a 92,9 Mb/s rate with an IPSec tunnel and a 92 Mb/s rate using an SSL VPN. This results show us the influence of the length of the overhead added by each protocol.

3 Experimentation

3.1 Platform tests

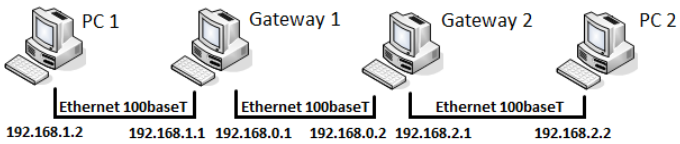

Figure 5 - Platform tests

The tests were performed on four identical machines running under FreeBSD 6.1. The computers are equipped with Pentium D 2.8 GHz dual-core processors and 1GB of DDR2 RAM. The network cards used are Intel 82546EB Gigabit Ethernet adaptater. Those cards are limited to 100Mb/s. For some unknow reason, the medium is then used in half-duplex mode.

3.2 Tools used

To carry out the tests we used the following tools:

3.2.1 OpenVPN

OpenVPN in its 2.1 RC17 version to complete the SSL tunnel available here: http://openvpn.net/index.php/open-source/downloads.html

3.2.2 client / server software

This is a client/server software we wrote in order to measure the transition time from the TCP to the Ethernet layer. The source code is available in appendix. The software establishes a TCP connection between the server and the client. Then it reads the current systemtime and sends a message to the TCP socket. We then measure the arrival time on the network adapter determined using the library "pcap" (which captures the packets on the ethernet card) and the difference gives the protocol stack processing time. We have checked that the measurement precision is sufficient (≈4us).

3.2.3 The package ipmt

The package ipmt contains utilities to evaluate the performance of connection. We use here especially udpmt and tcpmt respectively to create a UDP and TCP connection whose flow is configurable. In our study these tools will make a background traffic to simulate the presence of many users.

3.2.4 CPUkiller

CPUkiller is a tool we wrote whose goal is to generate CPU load. The source code is available in annex. It creates as many threads as there are parameters, that represent the load on each core on the machine. (For instance: ./CPUkiller 70 50 generates a charge of 70% on one core and a charge of 50% over the second core for an average of 60%)

3.2.5 charge_cpu.sh

charge_cpu.sh is a script we wrote in order to measure the average of the CPU load. It relies on vmstat, a standard Unix system tool to measure CPU. The script extracts the idle value of vmstat and makes its average. The source code is given in annex.

3.3 Results

Gateway 1 and Gateway 2 can use three different communication methods: a direct connection,

an unencrypted IPSec connection and an unencrypted OpenVPN connection. The configuration depends on the route added in each of the gateways connecting PC1 to PC2. However, no change is necessary on PC1 and PC2.

3.3.1 Testing time of transition

The goal of the first test is to determine if the transition time, generally low compared to the propagation time, would not increase significantly when the flow to route in the tunnel increases. This phenomenon, which would increase the RTT between two customers, can deteriorate the performance of applications such as video-conferencing. In this test we want to assess the transition time between TCP and Ethernet in the gateways while increasing the background traffic between PC1 and PC2. To do this, we use the client/server software which gives us the minimum, average and maximum time from the client side transmission transition (down) and the server side reception transition (up). The background's flow is simulated by udpmt and not tcpmt to avoid the problem of the half-duplex mode. The measurements are performed with a step of 10Mb/s (between 0 and 90Mb/s of a useful rate). Then, we send 200 packets with the client and retrieve the data with the server. The results are plotted in figure 6 and 7.

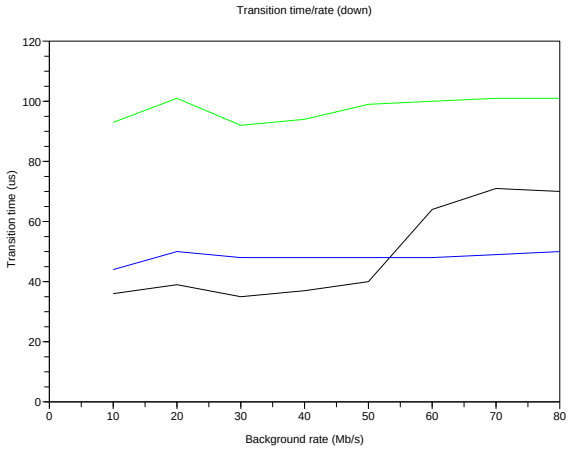

Figure 6 - Transition Time from TCP to Ethernet

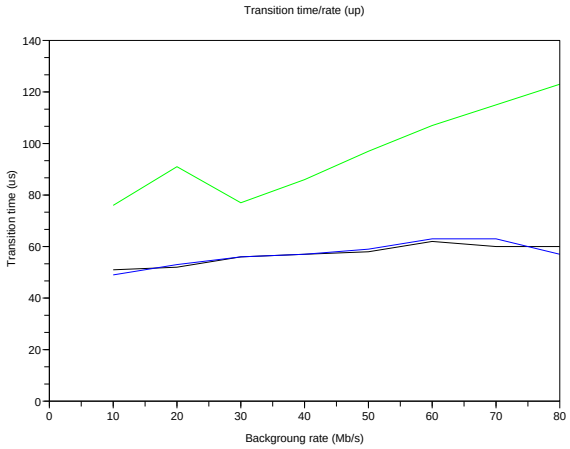

Figure 7 - Transition Time from Ethernet to TCP

The use of IPSec is similar to a direct connection watching the time of protocol transitions. We can also see that OpenVPN, implemented in user space, which requires copies from the kernel space, to the user space, and back to the kernel space, doubles the time of transition between Ethernet and TCP. However values obtained (about 100us) are negligible even for high speeds compared to the propagation delay time on the Internet (Paris-Lyon $\approx$ 7ms). It appears that the transition time protocol should not be determinant in the choice of the solution.

Direct

IPsec

OpenVPN

3.3.2 CPU load test

To ensure a certain QoS, it is necessary that the machine acting as a gateway has sufficient computing resources to achieve the forwarding

of all the packets in the tunnel without latency losses. To do this, we look at the evolution of the CPU load modifying the flow passing through the bridge. For this manipulation we used udpmt, a tool creating a flow between PC1 and PC2. Then we used the script charge_cpu.sh to obtain the average load corresponding to each rate. The step is 5Mb/s (between 5Mb/s and 90Mb/s of a useful rate). The results are presented in the following figures.

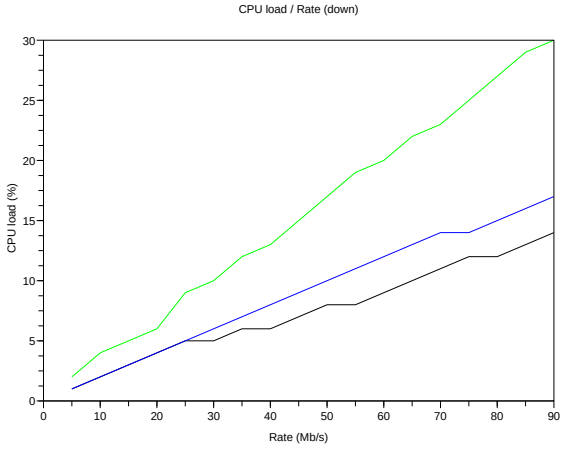

Figure 8 - CPU load when the flow enters in the tunnel

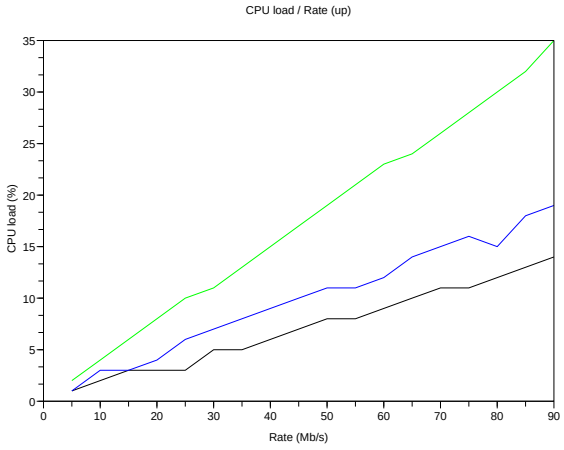

Figure 9 - CPU load when the flow exits the tunnel

It can be seen that the flow's increase causes an increasing of the CPU charge. This is proportional to the flow data rate. Of course, the OpenVPN curve is far above IPSec's one. This one slightly exceeds that of the direct connection's one.

All these measurements have been carried out without encryption. It arises the question of what encryption added to the CPU load. This is required in order to calibrate the equipment. We propose here to evaluate the CPU load generated by a symmetric encryption 128-bit Blowfish. We use the same experimental method as before to obtain the following curves:

OpenVPN encrypted

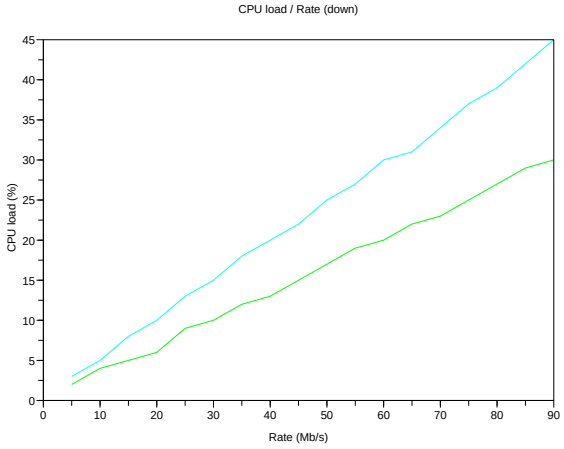

Figure 10 - CPU load when the flow enters in the secure tunnel

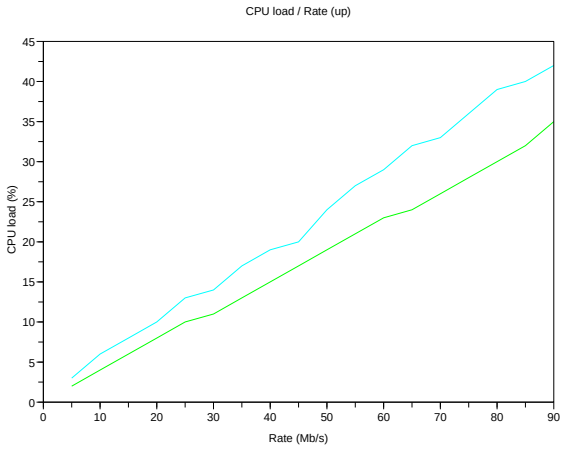

Figure 11 - CPU load when the flow exits the secure tunnel

We can see here that the curves remain linear but their slope increases from 45% down (transmission) and 24% up (reception). The addition of encryption is not negligible in the final installation of a VPN and to calibrate the ma-

chines performing the encryption. However the load is linear so it's a easy value to assess.

To calibrate the machine performing the encryption, it would be interesting to know if the CPU load can worsen the performances for a given flow in terms of transition time. For this, the tool CPUkiller was created and used to simulate a CPU load higher than the charges previously obtained. Having found under other conditions that the RTT exploded for low flows, but for a high CPU load, the results obtained, namely a constant time are not published here. Because of the lack of time we haven't been to determine the reasons of these results.

4 Limitations of the study and perspectives

- Given the duration of measurements, we couldn't conduct them more precisely.
- We have kept the same hardware configurations for all our tests, but we changed machines for logistical reasons.
- Machines are not optimized for the routing and the operating system running in the background can affect the CPU load and latency.
- The transition time have not been performed for data rates exceeding 90Mb/s because of the difficulties of transmission like packet retransmission. A possible improvement is to modify the client/server software, to take into account the TCP retransmission, in order to automate the measures.
- For a better approach, we could also consider requirements for the encryption of the transmission.

5 Conclusions

We have seen that when using the Internet, the protocol transition times are negligible. Regarding transmission rates, the differences between IPSec and OpenVPN are not critical (less than 5%) but may support a decision rather than another. The determining factors in the choice of a VPN solution will be the ease of installation on machines in the company and the CPU load improved by each solution. For a company with only one site and mobile clients, SSL VPN seems to be the best solution. Indeed the mobile workers are often connected through a NAT router (ISP box for example) and can't use IPSec. For the same reasons, we see that for a multi-site it is essential to use an SSL VPN (OpenVPN type) for its mobile workers. But for a site-to-site link, the use of a IPSec VPN is preferable thanks to its performances and the fact that professional routers are able to manage IPSec networks.

6 References

[1]Le tunnel GRE, Christian CALECA http://www.stielec.ac-aix-marseille.fr/cours/caleca/vpn/le_principe.htm
[2]Le tunneling, Alain MILLE http://liris.cnrs.fr/alain.mille/enseignements/DESS/Le_tunneling.htm
[3]http://openvpn.net/
[4]http://www.winpcap.org/
[5]http://www.tcpdump.org/

Annex

client/serveur TCP

This software allows to determine protocol transition time between TCP and Ethernet

- Client.c:

```
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <string.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <pcap.h>
#include <sys/time.h>
#include <unistd.h>
#include <semaphore.h>

//#define SERVEURNAME "10.8.0.1"
//#define SERVEURNAME "192.168.0.2"

int to_server_socket = -1;
int tuyaux[2];
int Compteur = 0;
sem_t Ecrit;
int chrono[2];
int stop=0;
pcap_t *adhandle;

void fin(int i)
{
        pcap_breakloop(adhandle);
        shutdown(to_server_socket,2);
        close(to_server_socket);
        pthread_join();
}


void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *pkt_data)
{
        struct tm *ltime;
        char timestr[16];
        time_t local_tv_sec;
        /* convert the timestamp to readable format */
        local_tv_sec = header->ts.tv_sec;
        chrono[0]=local_tv_sec;
        ltime=localtime(&local_tv_sec);
        strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);
//      printf("Paquet n %d: ",Compteur);
//      printf("%s,%.6d len:%d\n", timestr, (int)header->ts.tv_usec, header->len);
        chrono[1]=header->ts.tv_usec;
        sem_post(&Ecrit);
        Compteur ++;

}

void Capture(pcap_t *adhandle)
{
        /* start the capture */

        pcap_loop(adhandle, 0, packet_handler, NULL);
}




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

        char *server_name = "192.168.0.2";
        int vpn_actif=0;
        struct sockaddr_in serverSockAddr;
        struct hostent *serverHostEnt;
        long hostAddr;
        long status;
        char buffer[512];
        char bla[30];
        pcap_if_t *alldevs;
        pcap_if_t *d;
        int inum;
        int i=0;

        char errbuf[PCAP_ERRBUF_SIZE];
```

```
        struct timeval DateTCP;
        time_t local_tv_sec;
        struct tm *ltime;
        char timestr[16];
        u_int netmask=0xffffff;
        struct bpf_program fcode;
        int minim=1000000000;
        int maxim=0;
        int somme=0;
        int temps=0;
        int nb_pain_de_mie=0;
        int premier_paquet=1;

        sem_init(&Ecrit,0,0);
//      signal(SIGINT,fin);  // pour quand on fait le ctrl C

        bzero(&serverSockAddr,sizeof(serverSockAddr));


        if ((argc==2) && (!strcmp(argv[1],"-v"))) {  // pour les options du vpn
           server_name= "10.8.0.1";
            vpn_actif=1;
            printf("Options VPN activees\n");
        }else if ((argc==2) && (!strcmp(argv[1],"-s"))) {  // pour les options ipsec
           server_name= "10.0.0.1";
            vpn_actif=2;
            printf("Options IPSEC activees\n");
        }else{
            printf("Options pour lien direct activees\n");
            }
                hostAddr = inet_addr(server_name);
        if ( (long)hostAddr != (long)-1)
                bcopy(&hostAddr,&serverSockAddr.sin_addr,sizeof(hostAddr));
        else
        {
                serverHostEnt = gethostbyname(server_name);
                if (serverHostEnt == NULL)
                {
                        printf("Probleme gethost\n");
                        exit(0);
                }
                bcopy(serverHostEnt->h_addr,&serverSockAddr.sin_addr,serverHostEnt->h_length);
        }
        serverSockAddr.sin_port = htons(4242);
        serverSockAddr.sin_family = AF_INET;

        /* creation de la socket */
        if ( (to_server_socket = socket(AF_INET,SOCK_STREAM,0)) < 0)
        {
                printf("Probleme creation socket client\n");
                exit(0);
        }
        /* requete de connexion */
        if(connect( to_server_socket,(struct sockaddr *)&serverSockAddr,sizeof(serverSockAddr)) < 0 )
        {
                printf("Probleme demande de connexion\n");
                exit(0);
        }
        /* Retrieve the device list on the local machine */
        if (pcap_findalldevs(&alldevs, errbuf) == -1)
        {
                fprintf(stderr,"Error in pcap_findalldevs: %s\n", errbuf);
                exit(1);
        }

        /* Print the list */
        for(d=alldevs; d; d=d->next)
        {
        printf("%d. %s", ++i, d->name);
        if (d->description)
                printf(" (%s)\n", d->description);
        else
                printf(" (No description available)\n");
        }

        if(i==0)
        {
                printf("\nNo interfaces found! Make sure Pcap is installed.\n");
                exit(0);
        }

        printf("Enter the interface number (1-%d):",i);
        scanf("%d", &inum);

        if(inum < 1 || inum > i)
        {
                printf("\nInterface number out of range.\n");
                /* Free the device list */
                pcap_freealldevs(alldevs);
                exit(0);
        }

        /* Jump to the selected adapter */
        for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);
```

```
        /* Open the device */
        if ( (adhandle= pcap_open_live(d->name,          // name of the device
        65536,            // portion of the packet to capture 65536 guarantees that the whole packet will be
             captured on all the link layers
        1,    // promiscuous mode
        1000,             // read timeout
        errbuf            // error buffer
        ) ) == NULL)

        {
                fprintf(stderr,"\nUnable to open the adapter. %s is not supported by Pcap\n", d->name);
                /* Free the device list */
                pcap_freealldevs(alldevs);
                exit(0);
        }

        /* compile the filter */
        if (vpn_actif==1){
          if (pcap_compile(adhandle, &fcode, "(dst host 192.168.0.2) and (src host 192.168.0.1) and (ip proto
               \\udp) and (greater 90) and (less 200) and (dst port 4000)", 1, netmask) <0 )
          {
                fprintf(stderr,"\nUnable to compile the packet filter. Check the syntax.\n");
                /* Free the device list */
                pcap_freealldevs(alldevs);
                exit(0);
          }
       }else if (vpn_actif == 2){//
                if (pcap_compile(adhandle, &fcode, "(dst host 192.168.0.2) and (src host 192.168.0.1) and (
                     greater 90) and (less 200)", 1, netmask) <0 )
          {
                fprintf(stderr,"\nUnable to compile the packet filter. Check the syntax.\n");
                /* Free the device list */
                pcap_freealldevs(alldevs);
                exit(0);
                }
        }else{
                if (pcap_compile(adhandle, &fcode, "(dst host 192.168.0.2) and (src host 192.168.0.1) and (
                     ip proto \\tcp) and (greater 90) and (less 200)", 1, netmask) <0 )
          {
                fprintf(stderr,"\nUnable to compile the packet filter. Check the syntax.\n");
                /* Free the device list */
                pcap_freealldevs(alldevs);
                exit(0);
                }
                }

        /* set the filter */
        if (pcap_setfilter(adhandle, &fcode)<0)
        {
                fprintf(stderr,"\nError setting the filter.\n");
                /* Free the device list */
                pcap_freealldevs(alldevs);
                exit(0);
        }

        printf("\nlistening on %s...\n", d->name);

        /* At this point, we don't need any more the device list. Free it */
        pcap_freealldevs(alldevs);

        printf("lancement de la capture\n");
        pthread_t ThreadCapt;
        pthread_create(&ThreadCapt,NULL,Capture,adhandle);
                while (1)
                {
                        /* envoie de donne et reception */
                                printf("Entrer le texte:\n");
                                //scanf("%s", (char *)&bla);
                                usleep(200000);
//                              printf("texte entre: %s\n",&bla);
                        if (!strcmp(bla,"$quit")) {
                                /* fermeture de la connexion */
                                shutdown(to_server_socket,2);
                                close(to_server_socket);
                                pcap_breakloop(adhandle);
                                exit(0);
                        }
                        else
                        {
                                gettimeofday(&DateTCP,NULL);
                                local_tv_sec = DateTCP.tv_sec;
                                ltime=localtime(&local_tv_sec);
                                strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);
//                              printf("Envoi Data TCP  : ");
//                              printf("%s,%.6d\n", timestr,(int)DateTCP.tv_usec );

                                /* Envoi de deux paquets tcp*/
                                write(to_server_socket,"Vive_les_lasagnes",60);
                                sem_wait(&Ecrit); //On attend que pcap ait capture un paquet
                                temps=(-DateTCP.tv_sec+chrono[0])*1000000-DateTCP.tv_usec+chrono[1];
                                printf("Temps de transition (descente): %d us\n",temps);
                                if (premier_paquet==0) {
                                  nb_pain_de_mie++;
                                  if (temps<minim) {
                                    minim=temps;
```

```
                                        }
                                      if (temps>maxim) {
                                        maxim=temps;
                                        }
                                      somme+=temps;
                                        printf("Temps minimal: %d\n",minim);
                                        printf("Temps moyen:   %d\n",somme/nb_pain_de_mie);
                                        printf("Temps maximal: %d\n",maxim);
                                        printf("Nombre de paquets: %d\n",nb_pain_de_mie);
                                        printf("\n----------------\n\n");
                                    }
                                        premier_paquet=0;
                                }
                        }
                return 1;
}
```

- Serveur.c:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <pcap.h>
#include <semaphore.h>

char buffer[512];
int ma_socket;
int client_socket;
int Compteur = 0;
sem_t Ecrit;
sem_t Arret;
int stop=0;
int chrono[2];
int bool=0;
pcap_t *adhandle;

void packet_handler(u_char *param, const struct pcap_pkthdr *header, const u_char *pkt_data)
{

        sem_wait(&Arret);
        struct tm *ltime;
        char timestr[16];
        time_t local_tv_sec;
        /* convert the timestamp to readable format */
        local_tv_sec = header->ts.tv_sec;
        chrono[0] = local_tv_sec;
        //ltime=localtime(&local_tv_sec);
        //strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);
        printf("Paquet n %d: ",Compteur);
        //printf("%s,%.6d len:%d\n", timestr, (int)header->ts.tv_usec, header->len);
        chrono[1]=header->ts.tv_usec;

        sem_post(&Ecrit);
        Compteur ++;

}

/* Fonction de fermeture de connexion en cas d'echec */
void fin(int i)
{
pcap_breakloop(adhandle);
shutdown(client_socket,2);
            close(client_socket);
shutdown(ma_socket,2);
    close(ma_socket);


}

void Capture(pcap_t *adhandle)
{
        /* start the capture */
        pcap_loop(adhandle, 0, packet_handler, NULL);
}

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

    int port = 4242;
        pcap_if_t *alldevs;
        pcap_if_t *d;
        int inum;
        int i=0;
        char errbuf[PCAP_ERRBUF_SIZE];
        struct timeval DateTCP;
        struct timeval DateTCP2;
        time_t local_tv_sec;
        struct tm *ltime;
        char timestr[16];
```

```
    u_int netmask=0xffffff;
    struct bpf_program fcode;
    sem_init(&Ecrit,0,0);
    sem_init(&Arret,0,0);
    int Tps;
    bool=0;
    gettimeofday(&DateTCP2,NULL);
    int VPN=0;
    int min=1000000000;
    int max=0;
    int somme=0;
    int nb_packet=0;
    sem_post(&Arret);



    if ((argc==2) && (!strcmp(argv[1],"-v"))){
            VPN=1;
    }else if(argc==1){
            VPN=0;
    }else if ((argc==2) && (!strcmp(argv[1],"-s"))){
            VPN=2;
    }else{
            printf("Erreur dans les arguments\n");
            exit(0);
    }
    printf("VPN %d\n",VPN);



struct sockaddr_in mon_address, client_address;
int mon_address_longueur, lg;

            /* Retrieve the device list on the local machine */
    if (pcap_findalldevs(&alldevs, errbuf) == -1)
    {
            fprintf(stderr,"Error in pcap_findalldevs: %s\n", errbuf);
            exit(1);
    }

    /* Print the list */
    for(d=alldevs; d; d=d->next)
    {
    printf("%d. %s", ++i, d->name);
    if (d->description)
            printf(" (%s)\n", d->description);
    else
            printf(" (No description available)\n");
    }

    if(i==0)
    {
            printf("\nNo interfaces found! Make sure Pcap is installed.\n");
            exit(0);
    }

    printf("Enter the interface number (1-%d):",i);
    scanf("%d", &inum);

    if(inum < 1 || inum > i)
    {
            printf("\nInterface number out of range.\n");
            /* Free the device list */
            pcap_freealldevs(alldevs);
            exit(0);
    }

    /* Jump to the selected adapter */
    for(d=alldevs, i=0; i< inum-1 ;d=d->next, i++);

    /* Open the device */
    if ( (adhandle= pcap_open_live(d->name,          // name of the device
    65536,            // portion of the packet to capture 65536 guarantees that the whole packet will be
         captured on all the link layers
    1,    // promiscuous mode
    1000,             // read timeout
    errbuf            // error buffer
    ) ) == NULL)

    {
            fprintf(stderr,"\nUnable to open the adapter. %s is not supported by Pcap\n", d->name);
            /* Free the device list */
            pcap_freealldevs(alldevs);
            exit(0);
    }

    /* compile the filter */
    if (VPN==0){
    if (pcap_compile(adhandle, &fcode, "(greater 75) and (less 200) and (dst host 192.168.0.2) and (src
         host 192.168.0.1) and (ip proto \\tcp)", 1, netmask) <0 )
    {
            fprintf(stderr,"\nUnable to compile the packet filter. Check the syntax.\n");
            /* Free the device list */
            pcap_freealldevs(alldevs);
            exit(0);
```

```
    }
    }else if (VPN ==1){
    if (pcap_compile(adhandle, &fcode, "(greater 95) and (less 200) and (dst host 192.168.0.2) and (src
        host 192.168.0.1) and (ip proto \\udp) and (dst port 4000)", 1, netmask) <0 )
    {
            fprintf(stderr,"\nUnable to compile the packet filter. Check the syntax.\n");
            /* Free the device list */
            pcap_freealldevs(alldevs);
            exit(0);
            }
    }else if (VPN ==2){
    if (pcap_compile(adhandle, &fcode, "(greater 90) and (less 200) and (dst host 192.168.0.2) and (src
        host 192.168.0.1)", 1, netmask) <0 ) //
    {
            fprintf(stderr,"\nUnable to compile the packet filter. Check the syntax.\n");
            /* Free the device list */
            pcap_freealldevs(alldevs);
            exit(0);
    }
    }


    /* set the filter */
    if (pcap_setfilter(adhandle, &fcode)<0)
    {
            fprintf(stderr,"\nError setting the filter.\n");
            /* Free the device list */
            pcap_freealldevs(alldevs);
            exit(0);
    }

    printf("\nlistening on %s...\n", d->name);

    /* At this point, we don't need any more the device list. Free it */
    pcap_freealldevs(alldevs);


bzero(&mon_address,sizeof(mon_address));
mon_address.sin_port = htons(4242);
mon_address.sin_family = AF_INET;
mon_address.sin_addr.s_addr = htonl(INADDR_ANY);

/* creation de socket */
if ((ma_socket = socket(AF_INET,SOCK_STREAM,0))== -1)
{
    printf("Probleme lors de la creation du socket\n");
    exit(0);
}
signal(SIGINT,fin);

/* bind serveur - socket */
if (bind(ma_socket,(struct sockaddr *)&mon_address,sizeof(mon_address))==-1){
    printf("Port occupe\n");
    exit(0);
}

/* ecoute sur la socket */
listen(ma_socket,0);
printf("Serveur lance sur le port %d \nEn attente de connexion...\n",port);

/* accepte la connexion */
mon_address_longueur = sizeof(client_address);
    client_socket = accept(ma_socket,
                           (struct sockaddr *)&client_address,
                           &mon_address_longueur);
    printf("lancement de la capture\n");
    pthread_t ThreadCapt;
    pthread_create(&ThreadCapt,NULL,Capture,adhandle);
    while(1)
            {
                    DateTCP=DateTCP2;

                    sem_wait(&Ecrit);
                    lg = read(client_socket,buffer, 512);
                    gettimeofday(&DateTCP2,NULL);
                    sem_post(&Arret);
                    //local_tv_sec = DateTCP.tv_sec;
                    //ltime=localtime(&local_tv_sec);
                    //strftime( timestr, sizeof timestr, "%H:%M:%S", ltime);
                    //printf("Reception Data TCP  : ");
                    //printf("%s,%.6d\n", timestr,(int)DateTCP.tv_usec );
                    Tps = (DateTCP.tv_sec - chrono[0])*1000000 + (DateTCP.tv_usec - chrono[1]);
                    if (bool==0){
                            bool = 1;
                    }else{
                    if (min>Tps)
                            min = Tps;
                    if (max<Tps)
                            max = Tps;
                    somme+=Tps;
                    nb_packet++;
                    printf("Temps de transition (montee): %d us\n",Tps);
                    printf("Min %d/Max %d/Avg %d\n",min,max,somme/nb_packet);
                    }
```

```
                        if (lg <= 0)
                                break;
                        printf("Le serveur a recu: %s\n\n----------\n\n",buffer);
                }
                pcap_breakloop(adhandle);
            shutdown(client_socket,2);
            close(client_socket);
        shutdown(ma_socket,2);
            close(ma_socket);

    }
```

CPUkiller

This software create a determined CPU load.

```
#include <semaphore.h>
#include <stdlib.h>
#include <signal.h>
#include <stdio.h>
#include <pthread.h>

pthread_t * TabTh;
int NbTh;
sem_t Attend;

void fin(){
        int i;
        for (i=0;i<NbTh;i++){
                pthread_cancel(&(TabTh[i]));
        }
        sem_post(&Attend);
}

void Charge(int Ch){
        int a;
        int i;
        int dodo;
        struct timeval Avant;
        struct timeval Apres;
        while (1){
                gettimeofday(&Avant,NULL);
                for(i=0;i<=8000000;i++);
                gettimeofday(&Apres,NULL);
                a=(Apres.tv_sec-Avant.tv_sec)*1000000+Apres.tv_usec-Avant.tv_usec;
                dodo = a*100/Ch-a;
                if (dodo<0)
                        dodo=0;
                usleep(dodo);
        }
}

int main(int argc, char * argv[]){
        int i;
        sem_init(&Attend,0,0);
        NbTh=argc-1;
        TabTh=malloc(NbTh*sizeof(pthread_t));
        for (i=0;i<NbTh;i++){
                pthread_create(&(TabTh[i]),NULL,Charge,atoi(argv[i+1]));
        }
        signal(SIGINT,fin);
        sem_wait(&Attend);
        return 1;
}
```

charge_cpu.sh

This script measures the average CPU load.

```
#!/bin/sh
somme=0
n=0
while [ 1 ]
do
  n=$((n+1))
  rep=$(vmstat 1 2)
  set $rep
  rep=${60}
  somme=$((somme+$rep))
  printf "Moyenne: $((somme/$n)) % du CPU occupe\r"
done
```