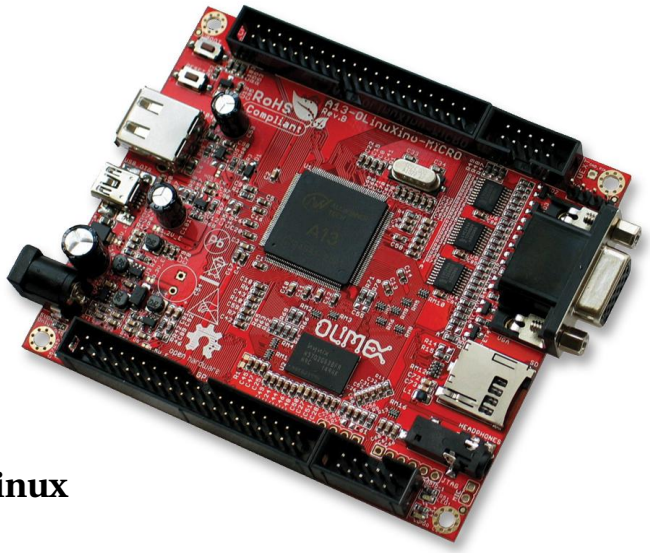
TP communication TCP/IP et système embarqué sous GNU/Linux

J.-M Friedt, 30 novembre 2016



Objectifs de ce TP :

- intérêt de travailler sur une plateforme exécutant GNU/Linux : portabilité du code entre PC et ARM
- configuration des interfaces réseau, routage des paquets
- serveur TCP/IP [a]
- problème d'*endianness* lors des transactions entre interlocuteurs sur réseau
- écoute des transactions sur une socket.

a. La façon la plus simple de tester un serveur TCP : `telnet IP port`

1 Configuration du réseau sous GNU/Linux

Configuration des interfaces : `ifconfig`.

- IP (*Internet Protocol*) : notion de routage des paquets
- des interfaces permettent à un "ordinateur" de communiquer
- des règles de routage définissent la destination de chaque paquet
- un temps maximal de vie de chaque paquet en limite la durée de routage
- adresse MAC : adresse physique de chaque interface ethernet, unique
- adresse IP : adresse virtuelle liée à la géographique
- relation MAC-IP : ARP
- relation IP-nom de domaine : DNS (`/etc/resolv.conf`)
- les services, le protocole utilisé et le port de connexion sont standardisés : `/etc/services`
- sous dimensionner le matériel rend **vulnérable aux attaques DoS**.

Nous allons utiliser une interface réseau virtuelle sur bus USB, nommée `usb`i ($i \geq 0$), pour communiquer entre le PC et la plateforme embarquée. Les interfaces ethernet sont quant à elles nommées `eth`i.

Liste des interfaces configurées : `ifconfig`. L'option `-a` permet d'afficher toutes les interfaces, même celles qui ne sont pas configurées. L'interface `localhost` avec IP 127.0.0.1 existe toujours.

La carte Olimexino A13 micro ne propose pas d'interface par défaut : seule la communication sur la console, au travers du port série, est active.

1. **Se connecter à la carte en lançant `minicom` sur l'interface `/dev/ttyUSB0`, débit de 115200, 8N1, pour atteindre la console.**
2. **Constater que l'environnement GNU/Linux ainsi accessible (sur la plateforme embarquée) se comporte de façon très similaire à l'interface utilisateur (en mode console) sur le PC. Nous allons nous convaincre que nous sommes sur la plateforme distante par `cat /proc/cpuinfo`.**
3. **Charger le module d'émulation d'une interface réseau sur bus USB par `modprobe g_ether`.**
4. **Assigner une adresse IP à cette interface par `ifconfig usb0 192.168.2.2`.**

Une fois la carte embarquée configurée, le PC a découvert la présence de cette nouvelle interface et nous informe des actions prises en conséquent (messages systèmes : `dmesg | tail`). Nous allons assigner une adresse IP à l'interface usb0 sur le PC : `ifconfig usb0 192.168.2.1`.

⚠ Comme toute commande se trouvant dans `/sbin`, `ifconfig` est une commande d'administration qui nécessite les droits de `root`. Il faut donc, dans notre cas, préfixer l'exécution de cette commande de `sudo`.

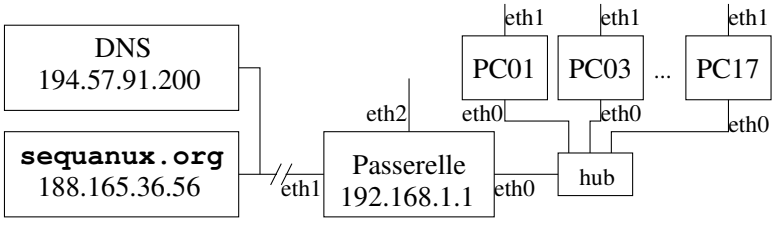


FIGURE 1 – Exemple d'architecture de réseau informatique.

Nous avons pris soin de sélectionner deux adresses IP sur le même sous-réseau, 192.168.1.x. Ainsi, Linux sait que tout paquet à destination de ce sous réseau doit partir sur l'interface `usb0`. Au contraire, tous les autres paquets (règle de `default gw`) doivent être routés vers l'interface connectée à internet [1], à savoir l'interface ethernet. Ces règles sont contenues dans la table de *routage*, visible par `route` (Fig. 1).

Si on veut accéder à une autre plage d'adresses (connexion internet), il faut définir une table de routage : `route add ...`. La passerelle (*default gw*) se définit par `route add default gw 192.168.0.1` par exemple.

Une fois la connexion TCP/IP [2] établie, nous sommes en mesure de remplacer la liaison asynchrone RS232 (lente) par une liaison rapide au travers du bus USB : la connexion se fait par `ssh root@192.168.2.2` (nous nous connectons en `root` en l'absence de présence d'autres comptes utilisateurs). Le mot de passe est lui aussi `root`. Par ailleurs, nous pouvons partager le contenu du disque dur du PC au travers du protocole NFS (*Network File System*) : la commande `mount -o nolock IP_distant:rép_distant rép_local` monte le répertoire distance du l'ordinateur d'IP_distant dans le répertoire local.

Exercice : monter, sur la carte Olinuxino A13, le répertoire `/home/etudiant/nfs_arma` du PC sur le répertoire local `/mnt`. Se convaincre de l'accès au disque dur du PC depuis la carte A13 en créant suir le PC un fichier dans ce répertoire (`touch toto /home/etudiant/nfs_arma`) et afficher le contenu du répertoire sur la carte A13 (`ls /mnt`).

Pour une connexion à internet, les serveurs de noms de domaines (DNS), chargés de traduire les noms en adresses IP, sont définis dans `/etc/resolv.conf`. La commande `nslookup` permet d'effectuer manuellement la résolution de nom, à savoir associer une adresse IP à un nom de site

```
$ nslookup sequanux.org
Name:   sequanux.org
Address: 188.165.36.56
```

2 Cross-compilation et exécution du binaire depuis la carte A13

L'objectif de la cross-compilation est de générer un binaire exécutable sur architecture ARM qui est le cœur de processeur contenu sur Olimexnio A13micro. La toolchain permettant cette compilation a été obtenue par l'outil `buildroot` selon la procédure décrite à `http://jmfriedt.free.fr/A13_v2.pdf`.

Une application est compilée sur PC au moyen de `arm-buildroot-linux-uclibcgnueabi-gcc` au lieu de `gcc`. Une fois l'exécutable généré, nous désirons y accéder depuis la carte A13 : nous allons copier l'exécutable du PC vers la carte Olinuxino A13 par `scp executable root@192.168.2.2:/tmp` (mot de passe : root). Au moyen de cette commande, le fichier sera copié sur la carte A13 dans le répertoire `/tmp`.

⚠Sous réserve d'accès au matériel, le *même* code source C pourra être compilé pour PC ou ARM et exécuté sur la plateforme adéquate.

Prenons l'exemple le plus simple

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main() {printf("Hello World, sqrt(2)=%f \n",sqrt(2));return(0);}
```

Deux observations s'imposent :

- quelque soit l'environnement d'exécution, la sortie standard (`stdout`) est redéfinie de façon appropriée par le système (redirection vers un terminal ou le port série).
- le programme n'est plus une boucle infinie mais rend la main au système d'exploitation en fin d'exécution.

Ce programme se compile par

```
arm-buildroot-linux-uclibcgnueabi-gcc -o executable source.c
```

Le résultat est bien un fichier exécutable sur architecture ARM et non PC. tandis que sur PC :

```
$ arm-buildroot-linux-uclibcgnueabi-gcc -o hello hello.c
$ file hello
hello: ELF 32-bit LSB executable, ARM, EABI5 ...
$ gcc -o hello hello.c
$ file hello
hello: ELF 32-bit LSB executable, Intel 80386 ...
```

3 Communication TPC/IP

Un serveur est à l'écoute des connexions des clients. Dans le cas qui va nous intéresser ici, nous choisirons de placer le serveur sur le système embarqué (A13) et le client est exécuté depuis le PC.

3.1 Programme en C (PC et A13)

```c
#include <sys/socket.h>
#include <resolv.h>
#include <strings.h>
#include <arpa/inet.h>

#define MY_PORT          9998
#define MAXBUF           1024

int main()
{int sockfd,longueur;
 struct sockaddr_in self;
 char buffer[MAXBUF];
 long s=0x61626364;

 sockfd = socket(AF_INET, SOCK_STREAM, 0);   // ICI LE TYPE DE SOCKET

 bzero(&self, sizeof(self));
 self.sin_family = AF_INET;
 self.sin_port = htons(MY_PORT);
 self.sin_addr.s_addr = INADDR_ANY;

 bind(sockfd, (struct sockaddr*)&self, sizeof(self));
 listen(sockfd, 20);

// while (1)
 {struct sockaddr_in client_addr;
  int clientfd,addrlen=sizeof(client_addr);

  clientfd = accept(sockfd, (struct sockaddr*)&client_addr, &addrlen);
  printf("%s:%d connected\n", inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));
  longueur=recv(clientfd, buffer, MAXBUF, 0);
  send(clientfd, buffer, longueur, 0);

  send(clientfd, &s, sizeof(s), 0);

  close(clientfd);
 }
 close(sockfd);return(0);   // Clean up (should never get here)
}
```

La séquence de commandes, extraites de ce programme, pour établir une transaction connectée de type TCP/IP, est donc

1. socket (protocole)
2. bind (port)
3. listen (attente bloquante)
4. accept (acquittement de la connexion du client)
5. send/recv (échange(s) de donnée(s) selon un protocole pré-établi)
6. close (fermeture du port permettant la transaction)

En cas d'erreur du client du type `Unable to connect to remote host: Connection refused`, tuer le serveur et le relancer : un délai est nécessaire entre la coupure d'une socket et la connexion d'un nouveau service.

Exercices :

1. **compiler le serveur écrit en C sur PC, l'exécuter et s'y connecter par** `telnet 127.0.0.1 9998`**. Que fait le serveur ? Analyser le résultat.**
2. **compiler le serveur pour ARM, l'exécuter sur la plateforme embarquée, et s'y connecter par** `telnet 192.168.2.2 9998`**. Que fait le serveur ?**

3.2 Programme en Java (PC)

```java
// from https://systembash.com/a-simple-java-tcp-server-and-tcp-client/
import java.io.*;
import java.net.*;

class TCPServer
{
 public static void main(String argv[]) throws Exception
  {String clientSentence;
   String capitalizedSentence;
   ServerSocket welcomeSocket = new ServerSocket(9998);
```

```
    Socket connectionSocket = welcomeSocket.accept();
    BufferedReader inFromClient =
      new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
    DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
    clientSentence = inFromClient.readLine();
    System.out.println("Received: " + clientSentence);
    capitalizedSentence = clientSentence.toUpperCase() + '\n';
    outToClient.writeBytes(capitalizedSentence);
    outToClient.writeInt(0x61626364);
   }
}
```

1. **Compiler ce code au moyen de** `javac TCPServer.java`
2. **Exécuter ce code dans la machine virtuelle java au moyen de** `java TCPServer`
3. **Se connecter au serveur et constater le résultat de la transaction. Commenter.**

4 Accès au matériel depuis le serveur

Un système d'exploitation fournit un niveau d'abstraction élevé pour les périphériques supportés par des pilotes. Ainsi, les entrées sorties numériques (GPIO) auxquelles sont connectées des LEDs sont accessibles au travers du pseudo-système de fichiers `/sys`. Dans le cas particulier des GPIOs, une ressource devient accessible lorsqu'elle est exportée vers `/sys` au moyen d'une écriture dans `/sys/class/gpio/export`.

Une alternative, peu élégante mais qui nous permet de nous raccrocher à la programmation sur microcontroleur qui nous a occupé pour le moment consiste à accéder directement au port contrôlant les GPIOs. Une bibliothèque implémentant de telles fonctionnalités est disponible à `http://dl.cubieboard.org/software/libs/gpio.tar` : la broche qui commande la LED verte est, dans la nomenclature du processeur qui équipe la carte Olinuxino A13 micro, PG9. Un exemple trivial de clignotement de LED est donc :

```
#include <stdlib.h>
#include <stdio.h>
#include "gpio_lib.h"
#define PG9     SUNXI_GPG(9)

int main()
{int i,status=0;

 if(sunxi_gpio_init())                {printf("Failed init  \n");return -1;}
 if(sunxi_gpio_set_cfgpin(PG9,OUTPUT)) {printf("Failed config\n");return -1;}

 while (1) {
    sunxi_gpio_output(PG9,status); sleep(1);
    status^=0xff;
 }
 sunxi_gpio_cleanup();
 return 0;
}
```

qui se compile par le `Makefile` suivant :

```
CC=arm-buildroot-linux-uclibcgnueabi-gcc

all: gpio_sleep gpio_sigalarm

gpio_lib.o: gpio_lib.c gpio_lib.h
  $(CC) -c gpio_lib.c

gpio_sigalarm.o: gpio_sigalarm.c
  $(CC) -c gpio_sigalarm.c

gpio_sleep.o: gpio_sleep.c
  $(CC) -c gpio_sleep.c

gpio_sigalarm: gpio_sigalarm.o gpio_lib.o
  $(CC) -o gpio_sigalarm gpio_sigalarm.o gpio_lib.o

gpio_sleep: gpio_sleep.o gpio_lib.o
  $(CC) -o gpio_sleep gpio_sleep.o gpio_lib.o

clean:
  rm *.o gpio_sigalarm gpio_sleep
```

1. **Une fois le GPIO initialisé depuis le shell, modifier le serveur pour allumer ou éteindre la LED sur ordre du client.**
2. **Modifier le serveur pour accepter plusieurs connexions simultanément au moyen d'un gestionnaire de connexions placé dans un** `thread` **initialisé par** `pthread_create()` **auquel nous passerons le descripteur de fichier de**

la `socket` comme argument.

3. **Lorsque deux clients accèdent à la même ressource, il y a risque de conflit et donc d'incohérence de l'état du matériel. Protéger cet accès aux ressources par la méthode appropriée.**

5 Application de `telnet` aux protocoles de haut niveau

`telnet` est le client universel pour le protocole TCP/IP : il permet d'accéder à la majorité des services (cf `/etc/services`) supportés par ce protocole. À titre d'exemples, nous proposons ci-dessous des transactions vers serveur SMTP (mail), HTTP (web), et FTP (fichiers). Les implémentations des divers protocoles sont décrits dans les RFC, accessibles sur `https://www.ietf.org/rfc.html`.

5.1 SMTP RFC 772 (Sep. 1980) et 821 (Aug. 1982)

SMTP : port 25

```
$ telnet localhost 25
Trying ::1...
Connected to localhost.
Escape character is '^]'.
220 rugged ESMTP Exim 4.84
HELO moi.ici
250 rugged Hello moi.ici [::1]
MAIL FROM: moi@ici.maison
250 OK
RCPT TO: jmfriedt@localhost
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
mail stupide
.
250 OK id=1YfXdU-0004Cl-PT
QUIT
221 rugged closing connection
Connection closed by foreign host.
```

```
You have mail in /var/mail/jmfriedt
$ mail
Mail version 8.1.2 01/15/2001.  Type ? for help.
"/var/mail/jmfriedt": 1 message 1 new
>N  1 moi@ici.maison     Fri May 08 08:59   15/448
Message 1:
From moi@ici.maison Fri May 08 08:59:12 2015
Envelope-to: jmfriedt@localhost
Delivery-date: Fri, 08 May 2015 08:59:12 +0200
From: moi@ici.maison
Date: Fri, 08 May 2015 08:59:12 +0200

mail stupide
```

5.2 HTTP (RFC 2068 (Jan. 1997) et 2616 (Jun. 1999))

Protocole supportant le World Wide Web [5, 6], port 80

```
$ telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET http://localhost/
Hello World

Ceci est un test.
Connection closed by foreign host.
```

ou pour accéder au monde extérieur

```
telnet proxy-www.univ-fcomte.fr 3128

GET http://jmfriedt.free.fr/ HTTP/1.1
```

5.3 FTP (RFC114 et successeurs, RFC542 (Aug. 1973), RFC959 (Oct. 1985)

FTP [1] requiert deux connexions – commande en 21 et transfert de fichiers :

commande ...

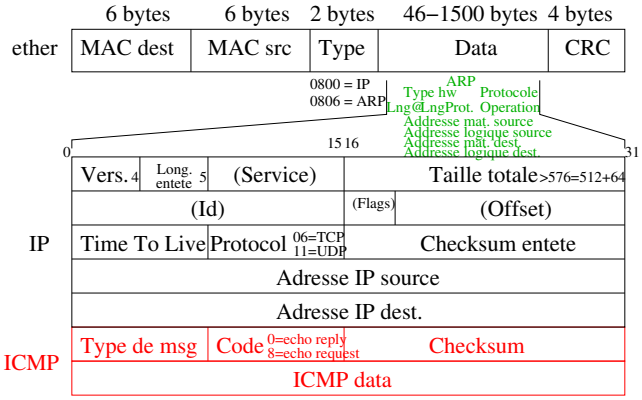
```
$ telnet localhost 21
USER anonymous
EPSV
229 Extended Passive mode
          OK (|||36982|)
LIST
RETR fichier
```

... et data

```
$ telnet localhost 36982
-rw-r--r-- 0    May 8 09:50 fichier
-rw-r--r-- 1002 May 8 09:47 fichier_de_test
Connection closed by foreign host.
$ telnet localhost 36982
ceci est un fichier de test
Connection closed by foreign host.
```

1. W.R. Stevens, *TCP/IP Illustrated Vol. 1*, Addison & Wesley (1994), chapitre 27 "FTP : File Transfer Protocol".

6 Raw IP et ICMP



Question : quel est l'identifiant du protocole ICMP dans l'entête IP ?

```
$ arp -a
? (192.168.0.1) at 00:13:d3:8d:d3:97 [ether] on eth0
$ arp -d 192.168.0.1
$ ping 192.168.0.11

# tcpdump  -i eth0 -XX                      # XX = show ethernet header
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
16:06:33.462038 ARP, Request who-has 192.168.0.1 tell 192.168.0.11, length 28
        0x0000:  ffff ffff ffff cc7e e75f cf6e 0806 0001  .......~._.n....
        0x0010:  0800 0604 0001 cc7e e75f cf6e c0a8 000b  .......~._.n....
        0x0020:  0000 0000 0000 c0a8 0001                 ..........
16:06:33.462590 ARP, Reply 192.168.0.1 is-at 00:13:d3:8d:d3:97 (oui Unknown), length 46
        0x0000:  cc7e e75f cf6e 0013 d38d d397 0806 0001  .~._.n..........
        0x0010:  0800 0604 0002 0013 d38d d397 c0a8 0001  ................
        0x0020:  cc7e e75f cf6e c0a8 000b 0000 0000 0000  .~._.n..........
        0x0030:  0000 0000 0000 0000 0000 0000            ............
16:06:33.462607 IP 192.168.0.11 > 192.168.1.1: ICMP echo request, id 1906, seq 1, length 64
        0x0000:  0013 d38d d397 cc7e e75f cf6e 0800 4500  .......~._.n..E.
        0x0010:  0054 1191 4000 4001 a6bb c0a8 000b c0a8  .T..@.@.........
        0x0020:  0101 0800 ff64 0772 0001 e9c2 4c55 c90c  .....d.r....LU..
        0x0030:  0700 0809 0a0b 0c0d 0e0f 1011 1213 1415  ................
        0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  ...........!"#$%
        0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
        0x0060:  3637                                     67
```

Une bonne compréhension du réseau est nécessaire pour les applications

- de sécurité : exemple de firewall ou routeurs dédiés. Filtrage des paquets, gestion en un temps minimum
- d'instrumentation contrôlée à distance. Même sans se soucier des sécurité des données, il faut savoir éviter des attaques de type DoS.
- l'ensemble des couches n'est souvent pas nécessaire (souhaitable) pour une application embarquée : maîtriser l'ensemble de la chaîne de communication pour n'en garder que les éléments utiles (p.ex raw-IP).

Dans nos exemples, l'argument de `socket` définit le protocole de transport (`SOCK_DGRAM` (UDP), `SOCK_STREAM` (TCP)).

`SOCK_RAW` ⇒ entête de 20 octets contenant IP source et destination, suivi de la charge à communiquer (routage)

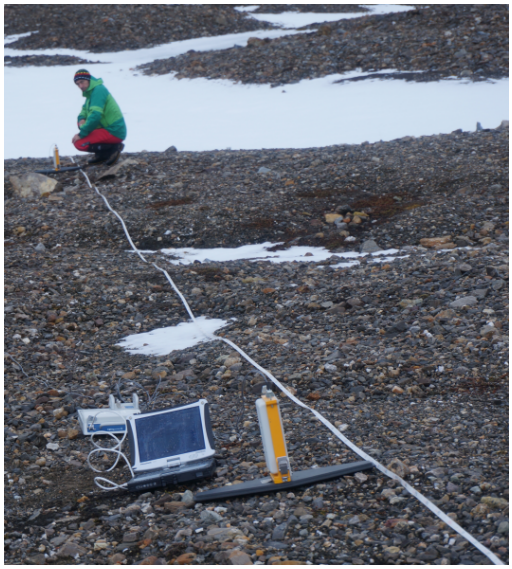
```
45 00 00 34 19 a9 00 00 3c ff 66 20 7f 00 00 01    <- source = 127.0.0.1
7f 00 00 01 30 30 30 30 30 30 30 30 30 30 30 30    <- dest = 127.0.0.1
```

7 Raw Ethernet

L'accès à la couche Ethernet brute [2] – sans indication d'adresse de départ et de fin – ne permet qu'une liaison point à point, avec un débit néanmoins élevé tel que l'offre la couche matérielle associée.

2. `http://sourceforge.net/projects/proexgprcontrol/`, A. Hugeat, J.-M Friedt, *A low cost approach to acoustic filters acting as GPR cooperative targets for passive sensing*, IWAGPR 2015

Pour système embarqué léger : raw-ethernet (Malå ProEx), pas de routage (liaison point à point PC-RADAR). Les paquets sont alors de simples trames ethernet (adresse MAC de source et de destination), sans notion de routage (absence d'adresse IP).



6 octets	6 octets	2 octets	
Adresse MAC destination	Adresse MAC source	Protocole	

Données

Données

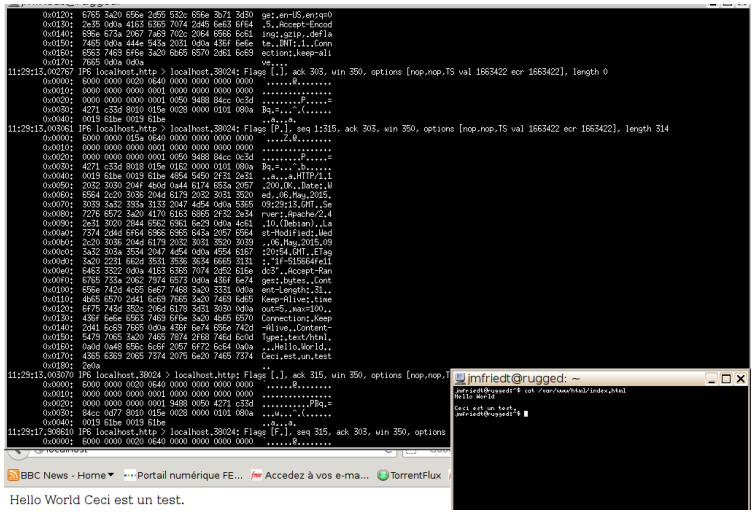
...

...

Données	2 octets Indicateur de fin de transaction	2 octets Compteur du nombre de paquets

8 À l'écoute des paquets ... tcpdump

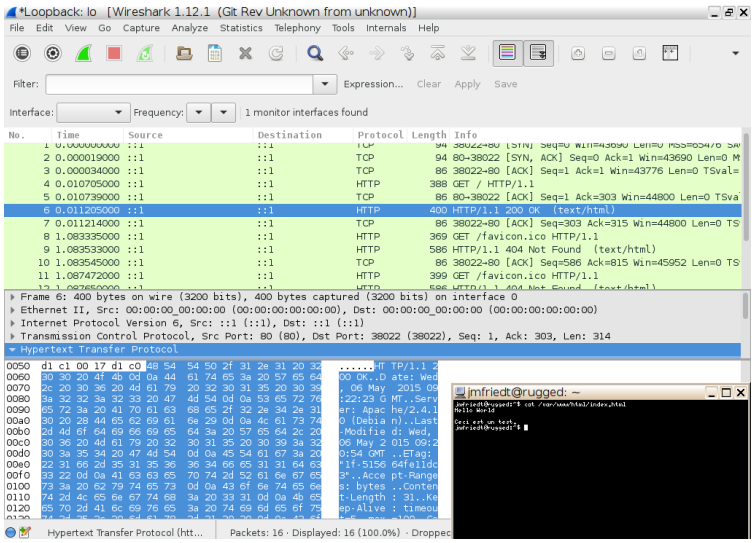
Exemple : `tcpdump port 80 -i lo -X`



Exercice : lancer une écoute sur `lo` par `tcpdump`, et engager une transaction ftp. Que constate-t-on en tappant son login/password ?

9 À l'écoute des paquets ... wireshark

Ex-`ethereal`, wireshark propose des fonctionnalités proches de celles de `tcpdump`, avec en plus une interface graphique pour l'analyse des paquets.



10 Outils TCP/IP

- traceroute

```
jmfriedt@vm1:~$ traceroute www.whitehouse.gov
traceroute to www.whitehouse.gov (23.214.186.191), 30 hops max, 60 byte packets
 1  10.10.0.254 (10.10.0.254)  0.377 ms  0.370 ms  0.362 ms
 2  vss-5b-6k.fr.eu (46.105.123.252)  0.954 ms  0.952 ms  0.947 ms
 3  rbx-g1-a9.fr.eu (178.33.100.29)  2.509 ms  2.509 ms  2.505 ms
 4  * * *
 5  ldn-5-6k.uk.eu (213.251.128.18)  48.180 ms * *
 6  * * *
 7  ae10.mpr2.lhr2.uk.zip.zayo.com (64.125.31.194)  6.492 ms  8.722 ms  7.263 ms
 8  ae5.mpr1.lhr15.uk.zip.zayo.com (64.125.21.10)  4.519 ms  4.515 ms  4.496 ms
 9  94.31.61.250.IPYX-074083-001-ZYO.above.net (94.31.61.250)  4.504 ms  4.500 ms  4.495 ms
10  a23-214-186-191.deploy.static.akamaitechnologies.com (23.214.186.191)  4.493 ms  4.488 ms  4.793 ms
```

- nslookup

```
$ nslookup www.femto-st.fr
Server:         130.67.15.198
Address:        130.67.15.198#53

Non-authoritative answer:
Name:   www.femto-st.fr
Address: 195.83.19.10
```

11 Outils HTTP

- Le couteau suisse : nc (netcat)

```
echo -e "GET http://jmfriedt.free.fr HTTP/1.0\n\n" | nc jmfriedt.free.fr 80
```

- wget – méthode GET (réponse : 0 :47 1 :57 2 :25)

```
dest=Vesoul
wget -q -O- "http://reiseauskunft.bahn.de/bin/query.exe/fn?revia=yes&existOptimizePrice=1&country=FRA&\
dbkanal_007=L01_S01_D001_KIN0001_qf-bahn_LZ003&ignoreTypeCheck=yes&S=Besancon+Franche+Comte+TGV&REQ0\
JourneyStopsS0A=7&Z=${dest}&REQ0JourneyStopsZ0A=7&trip-type=single&date=Me%2C+07.01.15&\
time=08%3A37&timesel=depart&returnTimesel=depart&optimize=0&infant-number=0&tariffTravellerType.1=E&\
tariffTravellerReductionClass.1=0&tariffTravellerAge.1=&qf-trav-bday-1=&tariffClass=2&\
start=1&qf.bahn.button.suchen=" | grep -A1 uratio | grep ^[0-9] | tr '\n' ' '
```

- curl – méthode POST

```
month=12
year=2014
curl -s --cookie-jar test --data \
"anzahlprofile=1&einheit=meter&vonkurz=LFPG&nachkurz=LYR+&landetime=&zwpunkte=&adresse=&profil=profilangeben&\
day=15&month=$month&year=$annee&flightnumber=XX001&von=PARIS%2C+FRANCE++++++++++++++++++&\
uptime=00%3A30&obentime=04%3A00&flughoehe0=10000&downtime=00%3A30&nach=LONGYEARBYEN%2C+NORWAY++++++++++&send=Send+request" \
http://www.helmholtz-muenchen.de/epcard/eng_flugoutput.php | grep Sv
```

12 Conclusion et perspectives :

Divers niveaux d'abstraction des transactions sur internet (formalisme OSI) ont été explorés, de IP à TCP/IP aux protocoles de haut-niveau (SMTP, FTP, HTTP). Nous avons investigué les outils de débuggage de transaction sur réseaux (tcpdump, wireshark). Les perspectives portent sur la collecte automatique d'informations sur internet (*data mining* [3, 4]) et instrumentation (Linux embarqué).

Références

[1] K. Hafner & M. Lyon, *Where Wizards Stay Up Late*, Simon & Schuster (1998)

[2] W.R. Stevens, *TCP/IP Illustrated Vol. 1*, Addison & Wesley (1994)

[3] T. Segaran & J. Hammerbacher, *Beautiful Data – The Stories Behind Elegant Data Solutions*, O'Reilly Media (2009) (systématiser/automatiser la collecte de données sur internet)

[4] J.-M Friedt, *Cartographier le bout du monde*, GNU/Linux Magazine France 185 (Sept. 2015)

[5] J. Gillies & R. Cailliau, *How the Web Was Born : The Story of the World Wide Web*, Oxford Univ. Press (2000)

[6] T. Berners-Lee, *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web*, HarperBusiness (2000)