

# INF157 - Utilisation des Réseaux

## Licence 3 Informatique

Arnaud Pecher (repris par Damien Magoni)

Bureau 322, Bâtiment A30, LaBRI  
Université de Bordeaux

Licence 3 Informatique - Bordeaux

- 1 Sockets
- 2 Aperçu des sockets en C
- 3 Sockets en Java
  - UDP
  - TCP

- 1 Sockets
- 2 Aperçu des sockets en C
- 3 Sockets en Java
  - UDP
  - TCP

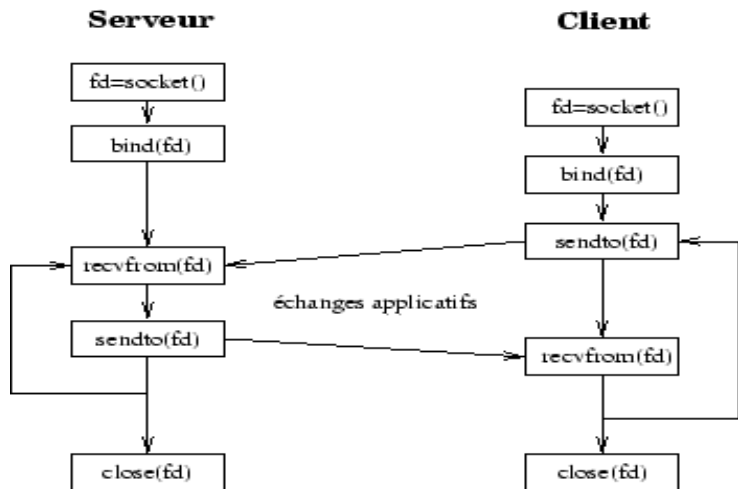
## sockets

Deux concepts :

- ensemble de primitives : *sockets de Berkeley* ;
- extrémité d'un canal de communication par lequel un processus peut émettre ou recevoir des données.

- 1 Sockets
- 2 Aperçu des sockets en C
- 3 Sockets en Java
  - UDP
  - TCP

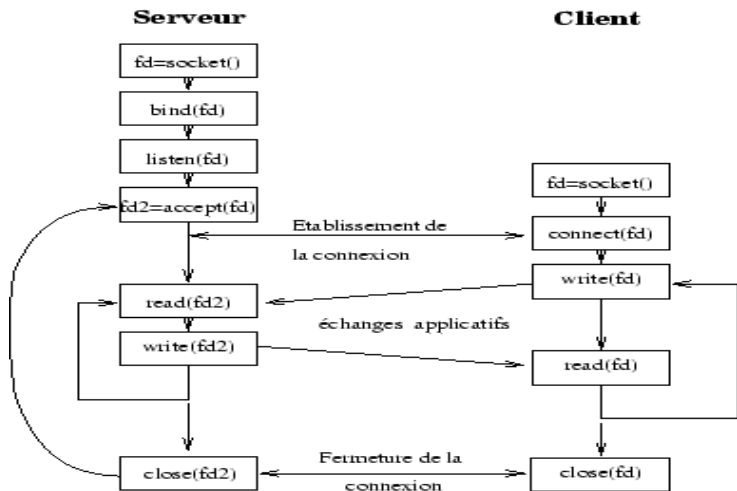
# Synopsis : mode non-connecté



©F. Laissus, 2005

- Socket = une connexion réseau est vue comme un fichier ;
- Appels système en mode non-connecté (UDP) :
  - *socket* : création d'un descripteur (de fichier) ;
  - *bind* : associer À une adresse locale ;
  - *sendto* : envoi de données ;
  - *recvfrom* : réception de données ;
  - *close* : libération du descripteur.

# Synopsis : mode connecté



©F. Laissus, 2005



Appels système en mode connecté (TCP) :

- *socket* : création d'un descripteur (de fichier) ;
- *bind* : associer À une adresse locale ;
- *listen* : volonté d'accepter des connexion ;
- *accept* : attente jusqu'À l'arrivée d'une demande de connexion ;
- *connect* : établir la connexion ;
- *send* : envoi de données ;
- *receive* : réception de données ;
- *close* : fermeture de connexion.

- 1 Sockets
- 2 Aperçu des sockets en C
- 3 Sockets en Java**
  - UDP
  - TCP

# Programmer en Java

Installation sous linux/ubuntu :

- **Java 1.6** : `sudo apt-get install sun-java6-jdk`
- environnement **Eclipse** : `sudo apt-get install eclipse`

Utilisation en ligne de commande :

- **compilation** en bytecode : `javac *.java`
- **exécution** : `java uneClasse`

HelloWorld.java \_\_\_\_\_

```
public class HelloWorld{
    public static void main( String[] args ){
        System.out.println("Hello World!");
    }
}
```

Compilation & execution \_\_\_\_\_

```
> javac HelloWorld.java
> java HelloWorld
Hello World!
```

# La classe DatagramPacket

datagramme UDP

## DatagramPacket

Cette classe permet de créer des objets qui contiendront les données d'un datagramme UDP.

Deux constructeurs sont disponibles, un pour les paquets À recevoir, l'autre pour les paquets À envoyer :

- `public DatagramPacket(byte buffer[], int taille)` : construit un objet pour recevoir un datagramme ; le paramètre *taille* correspond À la taille maximale des datagrammes À recevoir.
- `public DatagramPacket(byte buffer[], int taille, InetAddress a, int p)` : construit un objet pour envoyer un datagramme, À la machine d'adresse *a* sur le port *p*.

# La classe DatagramSocket

socket UDP

## DatagramSocket

Cette classe permet de créer des sockets UDP pour l'envoi et la réception des datagrammes UDP.

- `public DatagramSocket (int port)` : crée un objet de type socket et l'attache au port UDP local passé en paramètre ;
- `public void send(DatagramPacket data)` : envoi le datagramme *data* ;
- `public void receive(DatagramPacket data)` : réception d'un datagramme, stocké dans *data*.

## Exemple de client UDP

---

```
InetAddress address = InetAddress.getByName("raoul.labri.fr");
int port = 4321;
String ch = "Le message"; int chl = ch.length();
byte[] message = new byte[chl]; ch.getBytes(0, chl, message, 0);

DatagramPacket dp = new DatagramPacket(message, chl, address, port);
DatagramSocket ds = new DatagramSocket();
ds.send(dp);
```

# La classe InetAddress

Adresse IP

## classe InetAddress

Permet de gérer les adresses IP et les noms associés.

Obtenir

- l'adresse de la machine locale : `public static InetAddress getLocalHost()`
- l'adresse d'une machine dont on connaît le nom : `public static InetAddress getByName(String hostname)`
- les adresses d'une machine dont on connaît le nom : `public static InetAddress[] getAllByName(String hostname)`

Méthodes :

- connaître le nom de la machine : `public String getHostName ()`
- récupérer les 4 octets de l'adresse : `public byte[] getAddress ()`

Exemple :

```
System.out.println("adresse IP du serveur du LaBRI : "+InetAddress.getByName("www.labri.fr"));
```

## Exemple de serveur UDP

---

```
byte[] buffer = new byte[1024]; String ch;
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
DatagramSocket ds = new DatagramSocket(4321);
while(true) {
    ds.receive(dp);
    ch = new String(buffer, 0, 0, dp.getLength());
    System.out.println("Paquet reçu : message = " + ch +
        " - expéditeur = " + dp.getAddress().getHostName() +
        " - port = " + dp.getPort());
}
```



## Exercice

écrire 2 classes `ClientUDP.java` et `ServeurUDP.java` permettant le transfert d'un message **caractère par caractère**.

# Solution ClientUDP

```
import java.net.*;
import java.io.*;

public class ClientUDP{

public static void main( String[] args ) throws Exception{
    InetAddress address = InetAddress.getByName("localhost");
    int port = 4321;
    String ch = "Hello Bob, ici Greg, tout va bien?"; int chl = ch.length();
    byte[] message = new byte[1];
    DatagramSocket ds = new DatagramSocket();
    for (int j=0; j<1000; j++){
        long start = System.currentTimeMillis();
        for (int i=0; i<chl; i++){
            message[0]=(byte) ch.charAt(i);
            DatagramPacket dp = new DatagramPacket(message,1,address, port);
            ds.send(dp);
        }
        long end = System.currentTimeMillis();
        System.out.println("UDP: Message envoye: temps d'emission = "+(end-start)+"ms");
    }
}
```

# Solution ServeurUDP

```
import java.net.*;
import java.io.*;

public class ServeurUDP{

    public static void main( String[] args ) throws Exception{

        byte[] buffer = new byte[1024]; String ch;
        DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
        DatagramSocket ds = new DatagramSocket(4321);
        System.out.println("Serveur UDP en attente");
        while(true) {
            ds.receive(dp);
            ch = new String(buffer, 0, 0, dp.getLength());
            System.out.print(ch);
        }
    }
}
```

## classe Socket

La classe Socket permet de gérer une connexion TCP.

### Emission/Réception de données :

- **réception** : `public InputStream getInputStream() throws IOException`
- **émission** : `public OutputStream getOutputStream() throws IOException`

### Informations sur la connexion :

- **adresse IP distante** : `public InetAddress getInetAddress()`
- **adresse IP locale** : `public InetAddress getLocalAddress()`
- **port distant** : `public int getPort()`
- **port local** : `public int getLocalPort()`

La méthode `close()` ferme la connexion et libère les ressources du système associées À la connexion.

# Exemple de client TCP

classe Socket

## Exemple de client TCP

---

```
try {
    Socket s = new Socket("serverName", 5432);
    OutputStream os = s.getOutputStream();
    InputStream is = s.getInputStream();
    os.write(("a"));
    System.out.println(is.read());
    s.close();
} catch (Exception e) {
    // Traitement d'erreur
}
```

# Exemple de serveur TCP

La classe ServerSocket

## Exemple de serveur TCP

---

```
try {
    ServerSocket ecoute = new ServerSocket (5432, 5);
    Socket service = (Socket) null;
    while(true) {
        service = ecoute.accept();
        OutputStream os = service.getOutputStream();
        InputStream is = service.getInputStream();
        os.write(is.read());
        service.close();
    }
} catch (Exception e) {
    // traitement d'erreur
}
```

## Exercice

écrire 2 classes `ClientTCP.java` et `ServeurTCP.java` permettant le transfert d'un message **caractère par caractère**.

# Solution ClientTCP

```
import java.net.*; import java.io.*;
public class ClientTCP{
public static void main( String[] args ) throws Exception{
    String message="Hello Bob, ici Greg, tout va bien?";
    long start = System.currentTimeMillis();
    for (int i=0; i<message.length(); i++)
    {
    Socket s = new Socket("raoul.labri.fr",5432);
    OutputStream os = s.getOutputStream();
    os.write(message.charAt(i));
    s.close();
    }
    long end = System.currentTimeMillis();
    System.out.println("TCP: Message envoye: temps d'emission = "+(end-start)+"ms");
}}
```



# Solution ServeurTCP

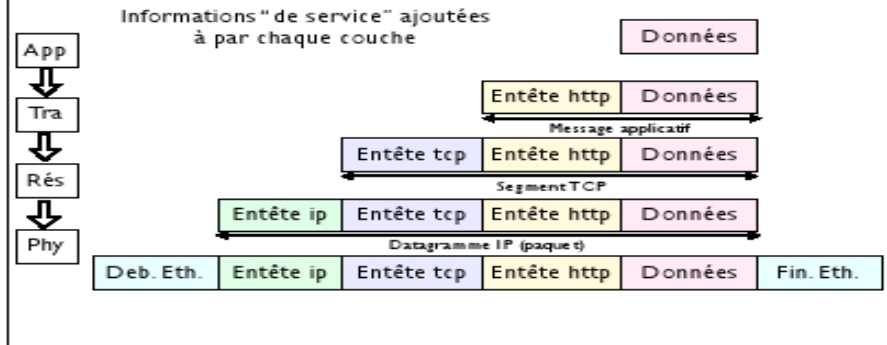
```
import java.net.*;
import java.io.*;
public class ServeurTCP{
public static void main( String[] args ) throws Exception{
    ServerSocket ecoute = new ServerSocket(5432,5);
    Socket service = (Socket)null;
    System.out.println("Serveur TCP en attente");
    while(true) {
        service = ecoute.accept();
        InputStream is = service.getInputStream();
        System.out.print((char) is.read());
        service.close();
    }
}}
```

4 Analyse d'une trame Ethernet pas-À-pas

5 Avec des outils de capture

- 4 Analyse d'une trame Ethernet pas-À-pas
- 5 Avec des outils de capture

## Encapsulation

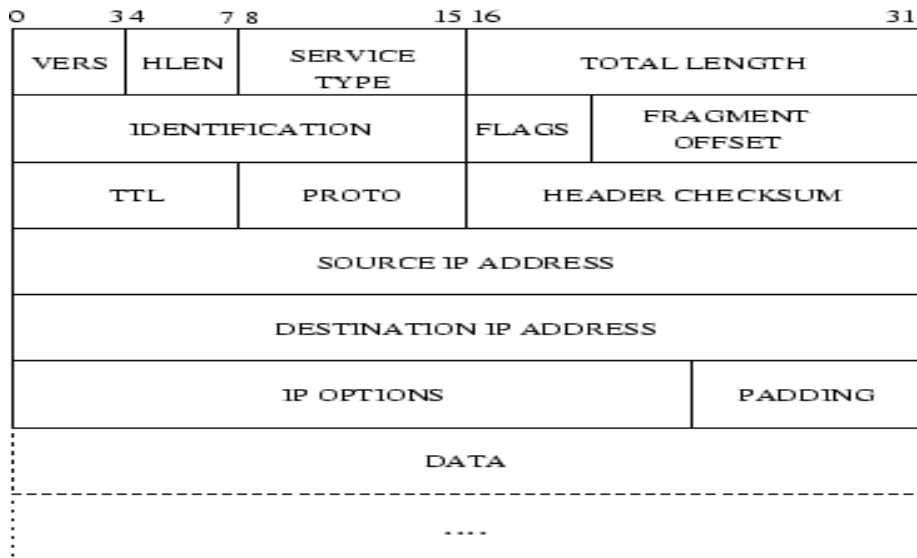


©Laissus (2004)

La trame de données est formée successivement de

- 8 octets : préambule
- 6 octets : adresse de destination
- 6 octets : adresse source
- 2 octets : type (en pratique) / longueur (norme)
- jusqu'à 1500 octets de données
- octets de remplissage
- 4 octets : total de contrôle

# Datagramme IPv4 : rappel de la structure



# Entête TCP : rappel de la structure

Transmission Control Protocol																														
0	3	4	7	8	11	12	15	16	19	20	23	24	27	28	31															
1 Source port																Destination port														
2 Sequence Number																														
3 Acknowledgment Number																														
4 Data Offset		Reserved					CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window															
5 Checksum																Urgent pointer														
6 Options																							Padding							
7 Data																														

Source : <http://www.linux-france.org/>

©2001-2005 Oskar Andreasson

# Exercice : interprétation d'une trame

## Interprétation d'une trame

Soit la capture d'une trame en hexadécimal suivante. Détailler les caractéristiques de la trame, en interprétant les entêtes encapsulés.

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

*Indications :*

- le préambule Ethernet n'a pas été capturé ;
- le type d'une trame IP est 0800 ;
- le type d'une trame TCP est 06, d'une trame UDP est 12.



# Solution : interprétation de l'entête Ethernet

## Entête Ethernet :

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

- 00 09 5b 9a c4 94 : adresse MAC destination ;
- 00 90 4b 16 5d 24 : adresse MAC source ;
- 08 00 : suite de la trame : IP/TCP

# Solution : interprétation de l'entête IP

Détermination de la taille de l'entête et de la version (4 ou 6) :

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

- $0x45 = 01000101$  ;
- version :  $0100 = 4$  (IPv4) ;
- longueur entête :  $0101 = 5$  mots de 32 bits (20 octets) ;

# Solution : interprétation de l'entête IP (2)

## Interprétation de l'entête IPv4 : octets 2 À 4 :

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

- service type : 0x00 ;
- longueur totale : 0x0034 = 52 octets ;

### Remarques :

- 52 octets +14 octets entete Ethernet = taille totale de la trame 66 octets ;

# Solution : interprétation de l'entête IP (3)

## Interprétation de l'entête IPv4 : octets 5 À 8 :

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

- identification : 0x68db ;
- flags (3 bits) : 0x4000 = 0100 0000 0000 0000
  - bit DF (don't fragment) = vrai ;
  - bit MF (more fragment) = faux ;
- fragment offset : 0x4000 = 0100 0000 0000 0000 = 0 : numéro 0 (pas de fragmentation) ;

# Solution : interprétation de l'entête IP (4)

## Interprétation de l'entête IPv4 : octets 9 À 12 :

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

- TTL (nombre de sauts restants) : 0x40=64 ;
- PROTO : 0x06=6=TCP ;
- Header Checksum : 0x74dd

# Solution : interprétation de l'entête IP (fin)

Interprétation de l'entête IPv4 : les 8 derniers octets (13 À 20) :

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

- adresse IPv4 de la source : 0xc0a80002=**192.168.0.2** ;
- adresse IPv4 du destinataire : 0x93d2088f=**147.210.8.143**.

# Solution : interprétation de l'entête TCP

## Interprétation de l'entête TCP : les premiers octets (de 1 À 12) :

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

- port source :  $0x8002 = 32770$  ;
- port destination :  $0x03e1 = 993$  (imaps) ;
- numéro de séquence :  $0xcd9cf713 = 3449616147$  ;
- numéro d'acquittement :  $0x496fbd4c = 1232059724$ .

# Solution : interprétation de l'entête TCP (fin)

## Interprétation de l'entête TCP : les drapeaux (octet 14) :

Adresse (hexa)	8 octets	8 octets
0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0
0040	c4 ef	

Drapeaux : 0x10 = 00010000

- URG (urgent) : faux ; ACK (acquiescement) : vrai ;
- PSH (push) : faux ; RST (réinitialisation) : faux ;
- SYN (demande de connexion) : faux ; FIN (interruption) : faux.

Conclusion : la trame capturée est une trame d'acquiescement du protocole TCP, entre la machine 192.168.0.2 (port 32770) et le serveur courrier 147.210.8.143 en crypté (port imaps 993).

Remarque : le serveur 147.210.8.143 est iona.labri.fr (serveur du LaBRI).



- 4 Analyse d'une trame Ethernet pas-À-pas
- 5 Avec des outils de capture

## tcpdump

Utilitaire de capture de trames très complet.

Site : [www.tcpdump.org](http://www.tcpdump.org)

Exemple (extrait de Linux Magazine France 25)

```
# tcpdump -X -s 0 dst host 193.252.19.209 and port 110 and tcp
```

```
20:51:50.690745 193.253.217.180.42034 > 193.252.19.209.pop3: P 0:11(11)
```

```
ack 79 win 5808 (DF)
```

```
0x0000 4500 003f 0000 4000 4006 c939 c1fd d9b4 E..?..@.@..9....
0x0010 c1fc 13d1 a432 006e b862 482d 520a 6b11 .....2.n.bH-R.k.
0x0020 8018 16b0 c3cf 0000 0101 080a 00a9 c278 .....x
0x0030 0bfa 1f75 7573 6572 206f 6b6b 690d 0a ...uuser.bidon..
```

```
20:51:50.745300 193.253.217.180.42034 > 193.252.19.209.pop3: P
```

```
11:25(14) ack 84 win 5808 (DF)
```

```
0x0000 4500 0042 0000 4000 4006 c936 c1fd d9b4 E..B..@.@..6....
0x0010 c1fc 13d1 a432 006e b862 4838 520a 6b16 .....2.n.bH8R.k.
0x0020 8018 16b0 33d0 0000 0101 080a 00a9 c27e ....3.....~
0x0030 0bfa 1f7a 7061 7373 2079 746b 6b76 7875 ...zpass.coucou
0x0040 0d0a
```

## wireshark

Analyseur en mode graphique de protocoles réseaux.

- site : [www.wireshark.com](http://www.wireshark.com)
- multi-plateforme : linux, windows

# Wireshark (1)

## Trame - Ethernet/IP/TCP

The screenshot shows the Wireshark interface with a packet capture of an IMAPS connection. The packet list pane shows three packets:

No.	Time	Source	Destination	Protocol	Info
1	0.0001	192.168.0.1	147.210.8.	SSL	Application Data
2	0.0611	147.210.8.	192.168.0.1	SSL	Application Data
3	0.0611	192.168.0.1	147.210.8.	TCP	32770 > imap [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215

The details pane for Frame 3 (66 bytes on wire, 66 bytes captured) shows the following information:

- Arrival Time: Sep 29, 2005 17:48:41.890944000
- Time delta from previous packet: 0.000037000 seconds
- Time since reference or first frame: 0.061928000 seconds
- Frame Number: 3
- Packet Length: 66 bytes
- Capture Length: 66 bytes
- Protocols in frame: eth:ip:tcp
- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imap (993), Seq: 53, Ack: 109, Len: 0

The packet bytes pane shows the raw data in hexadecimal and ASCII:

0000	00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00	..[.....K.]\$.E.
0010	00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2	.4h.@.@.t.....
0020	08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10	.....Io.L.
0030	3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0	>.....9.
0040	c4 ef	..

# Wireshark (2)

## Entête Ethernet : adresses MAC

The screenshot shows the Wireshark interface with the following details:

- Packet List:**
  - 1 0.0001 192.168.0.1:147.210.8. SSL Application Data
  - 2 0.0611 147.210.8. 192.168.0.1: SSL Application Data
  - 3 0.0611 192.168.0.1:147.210.8. TCP 32770 > imaps [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215
- Packet 3 Details:**
  - Frame 3 (66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0)
  - Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
    - Destination: 00:09:5b:9a:c4:94 (Netgear\_9a:c4:94)
    - Source: 00:90:4b:16:5d:24 (GemtekTe\_16:5d:24)
    - Type: IP (0x0800)
  - Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imaps (993), Seq: 53, Ack: 109, Len: 0
- Packet Bytes:**

```
0000 00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00  ....[.....K.]$.E.
0010 00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2  .4h.@.@.t.....
0020 08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10  .........Io.L..
0030 3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0  >.....9.
0040 c4 ef  ..
```

# Wireshark (3)

Entête Ethernet : adresse MAC - point d'accès WIFI

The screenshot shows the Wireshark interface with the following details:

- Packet List:**
  - 1 0.0000 | 192.168.0.1 | 147.210.8. | SSL Application Data
  - 2 0.0611 | 147.210.8. | 192.168.0.1 | SSL Application Data
  - 3 0.0611 | 192.168.0.1 | 147.210.8. | TCP 32770 > imaps [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215
- Packet 3 Details:**
  - Frame 3 (66 bytes on wire (66 bytes captured))
  - Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
    - Destination: 00:09:5b:9a:c4:94 (Netgear 9a:c4:94)
    - Source: 00:90:4b:16:5d:24 (GemtekTe\_16:5d:24)
    - Type: IP (0x0800)
  - Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imaps (993), Seq: 53, Ack: 109, Len: 0
- Packet Bytes:**

0000	00 09 5b 9a c4 94	00 90 4b 16 5d 24 08 00 45 00	..... [..... K.]\$.E.
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2	.4h.@.@. t.....
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10	..... ..Io.L..
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0	>..... ..9.
0040	c4 ef		..

# Wireshark (4)

## Entête Ethernet : type de datagramme encapsulé

Wireshark interface showing network traffic analysis. The packet list pane displays three packets:

- 1 0.0001 192.168.0.1:147.210.8. SSL Application Data
- 2 0.0611 147.210.8. 192.168.0.1:147.210.8. SSL Application Data
- 3 0.0611 192.168.0.1:147.210.8. TCP 32770 > imaps [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215

The packet details pane for the selected packet (Frame 3) shows:

- Frame 3 (66 bytes on wire (66 bytes captured) on interface 0)
- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Destination: 00:09:5b:9a:c4:94 (Netgear\_9a:c4:94)
- Source: 00:90:4b:16:5d:24 (GemtekTe\_16:5d:24)
- Type: IP (0x0800)
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imaps (993), Seq: 53, Ack: 109, Len: 0

The packet bytes pane shows the raw hex and ASCII data:

```
0000 00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00 .....K.]$.E.
0010 00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2 .4h.@.@.t.....
0020 08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10 .....Io.L...
0030 3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0 >.....9.
0040 c4 ef ..
```

# Wireshark (5)

## Entête IP

The screenshot shows the Wireshark interface with a packet list and packet details pane. The packet list shows three packets:

No.	Time	Source	Destination	Protocol	Info
1	0.000	192.168.0.1	147.210.8.	SSL	Application Data
2	0.061	147.210.8.	192.168.0.1	SSL	Application Data
3	0.061	192.168.0.1	147.210.8.	TCP	32770 > imap [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215

The packet details pane for packet 3 shows the following structure:

- Frame 3 (66 bytes on wire (66 bytes captured))
- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - Total Length: 52
  - Identification: 0x68db (26843)
  - Flags: 0x04 (Don't Fragment)
  - Fragment offset: 0
  - Time to live: 64
  - Protocol: TCP (0x06)
  - Header checksum: 0x74dd (correct)
  - Source: 192.168.0.2 (192.168.0.2)
  - Destination: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imap (993), Seq: 53, Ack: 109, Len: 0

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000 00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00  ...[.....K.]$.E.
0010 00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2  .4h.@.@.t.....
0020 08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10  ..Io.L..
0030 3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0  >.....9.
0040 c4 ef  ..
```



# Wireshark (6)

Entête IP : champ longueur

Wireshark interface showing packet details for an IP packet. The packet list shows three packets:

- 1 0.0001 192.168.0.1:147.210.8. SSL Application Data
- 2 0.0611 147.210.8. 192.168.0.1: SSL Application Data
- 3 0.0611 192.168.0.1:147.210.8. TCP 32770 > imaps [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215

The selected packet (No. 3) is expanded to show details:

- Frame 3 (66 bytes on wire (66 bytes captured))
- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - Total Length: 52
  - Identification: 0x68db (26843)
  - Flags: 0x04 (Don't Fragment)
  - Fragment offset: 0
  - Time to live: 64
  - Protocol: TCP (0x06)
  - Header checksum: 0x74dd (correct)
  - Source: 192.168.0.2 (192.168.0.2)
  - Destination: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imaps (993), Seq: 53, Ack: 109, Len: 0

The packet bytes pane shows the raw hex and ASCII data:

```
0000 00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00  ... [..... K.]$.E.
0010 00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2  .4h.@.@. t.....
0020 08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10  ..Io.L...
0030 3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0  >.....9.
0040 c4 ef ..
```

# Wireshark (7)

## Entête IP : champ identification

The screenshot shows the Wireshark interface with a packet capture of an SSL application data segment. The packet list pane shows three packets, with the third packet selected. The packet details pane shows the following structure:

- Frame 3 (66 bytes on wire, 66 bytes captured)
- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - Total Length: 52
  - Identification: 0x68db (26843)
  - Flags: 0x04 (Don't Fragment)
  - Fragment offset: 0
  - Time to live: 64
  - Protocol: TCP (0x06)
  - Header checksum: 0x74dd (correct)
  - Source: 192.168.0.2 (192.168.0.2)
  - Destination: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imap5 (993), Seq: 53, Ack: 109, Len: 0

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000  00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00  ....K.]$.E.
0010  00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2  .4h.@.t.....
0020  08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10  ....Io.L...
0030  3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0  >.....9.
0040  c4 ef  ..
```

# Wireshark (8)

## Entête IP : champ TTL

The screenshot shows the Wireshark interface with a packet capture of an SSL Application Data packet. The packet list pane shows three packets, with the third packet selected. The packet details pane shows the following information:

- Frame 3 (66 bytes on wire, 66 bytes captured)
- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - Total Length: 52
  - Identification: 0x68db (26843)
  - Flags: 0x04 (Don't Fragment)
  - Fragment offset: 0
  - Time to live: 64
- Protocol: TCP (0x06)
  - Header checksum: 0x74dd (correct)
  - Source: 192.168.0.2 (192.168.0.2)
  - Destination: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imap5 (993), Seq: 53, Ack: 109, Len: 0

The packet bytes pane shows the raw data of the packet, with the TTL field (0x40) highlighted in blue. The hex data is: 00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00 ... [..... K.]\$.E. 0010 00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2 .4h.@. t..... 0020 08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10 ..... ..Io.L.. 0030 3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0 >..... ..9. 0040 c4 ef ..

# Wireshark (9)

## Entête IP : type de paquet encapsulé

Wireshark interface showing network traffic analysis. The packet list pane displays three packets:

- 1 0.0000 | 192.168.0.2 | 147.210.8. | SSL Application Data
- 2 0.0611 | 147.210.8. | 192.168.0.2 | SSL Application Data
- 3 0.0611 | 192.168.0.2 | 147.210.8. | TCP 32770 > imaps [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215

The packet details pane for the selected packet (Frame 3) shows:

- Frame 3 (66 bytes on wire, 66 bytes captured)
- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - Total Length: 52
  - Identification: 0x68db (26843)
  - Flags: 0x04 (Don't Fragment)
  - Fragment offset: 0
  - Time to live: 64
- Protocol: TCP (0x06)
  - Header checksum: 0x74dd (correct)
  - Source: 192.168.0.2 (192.168.0.2)
  - Destination: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imaps (993), Seq: 53, Ack: 109, Len: 0

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000  00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00  ....K.]$.E.
0010  00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2  .4h.@. t.....
0020  08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10  ....Io.L...
0030  3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0  >.....9.
0040  c4 ef  ..
```

# Wireshark (10)

## Entête IP : somme de contrôle

Wireshark interface showing packet details for an IP header. The packet list shows three packets: two SSL Application Data and one TCP. The packet details pane is expanded to show the Internet Protocol section, which includes fields for Version (4), Header length (20 bytes), DSCP, Total Length (52), Identification (0x68db), Flags (0x04), Fragment offset (0), Time to live (64), and Protocol (TCP). The Transmission Control Protocol section is also visible below. The packet bytes pane shows the raw hex and ASCII data for the first 16 bytes of the packet.

No.	Time	Source	Destination	Protocol	Info
1	0.0001	192.168.0.1	147.210.8.1	SSL	Application Data
2	0.0611	147.210.8.1	192.168.0.1	SSL	Application Data
3	0.0611	192.168.0.1	147.210.8.1	TCP	32770 > imap [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215

Frame 3 (66 bytes on wire (66 bytes captured))

- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - Total Length: 52
  - Identification: 0x68db (26843)
  - Flags: 0x04 (Don't Fragment)
  - Fragment offset: 0
  - Time to live: 64
  - Protocol: TCP (0x06)
  - Header checksum: 0x74dd (correct)
  - Source: 192.168.0.2 (192.168.0.2)
  - Destination: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imap (993), Seq: 53, Ack: 109, Len: 0

```
0000  00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00  ....[....K.]$.E.
0010  00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2  .4h.@.@.t.....
0020  08 8f 80 02 03 e1 cd 9c  f7 13 49 6f bd 4c 80 10  ....Io.L...
0030  3e 96 8d 1c 00 00 01 01  08 0a 00 0e ff e0 39 d0  >.....9.
0040  c4 ef  ..
```

# Wireshark (11)

Entête IP : adresse IP source

The screenshot shows the Wireshark interface with the following details:

- Filter:** `3 0.061! 192.168.0.1`
- Packet List:**
  - 1 0.000! 192.168.0.1:147.210.8. SSL Application Data
  - 2 0.061! 147.210.8. 192.168.0.1: SSL Application Data
  - 3 0.061! 192.168.0.1:147.210.8. TCP 32770 > imaps [ACK] Seq=53 Ack=109 Win=16022 Len=0 TSV=983008 TSER=969983215
- Packet 3 Details:**
  - Frame 3 (66 bytes on wire, 66 bytes captured)
  - Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
  - Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
    - Version: 4
    - Header length: 20 bytes
    - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    - Total Length: 52
    - Identification: 0x68db (26843)
    - Flags: 0x04 (Don't Fragment)
    - Fragment offset: 0
    - Time to live: 64
    - Protocol: TCP (0x06)
    - Header checksum: 0x74dd (correct)
    - Source: 192.168.0.2 (192.168.0.2)
    - Destination: 147.210.8.143 (147.210.8.143)
  - Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imaps (993), Seq: 53, Ack: 109, Len: 0
- Packet Bytes:**

0000	00 09 5b 9a c4 94 00 90	4b 16 5d 24 08 00 45 00	..... K.]\$.E.
0010	00 34 68 db 40 00 40 06	74 dd c0 a8 00 02 93 d2	.4h.@.@. t....
0020	08 8f 80 02 03 e1 cd 9c	f7 13 49 6f bd 4c 80 10	..... ..Io.L..
0030	3e 96 8d 1c 00 00 01 01	08 0a 00 0e ff e0 39 d0	>..... ..9.
0040	c4 ef		..

# Wireshark (12)

## Paquet TCP - protocole IMAP

Wireshark interface showing a captured packet (No. 3) of type TCP. The packet details pane shows the structure of the packet:

- Frame 3 (66 bytes on wire, 66 bytes captured)
- Ethernet II, Src: 00:90:4b:16:5d:24, Dst: 00:09:5b:9a:c4:94
- Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 147.210.8.143 (147.210.8.143)
  - Version: 4
  - Header length: 20 bytes
  - Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  - Total Length: 52
  - Identification: 0x68db (26843)
  - Flags: 0x04 (Don't Fragment)
  - Fragment offset: 0
  - Time to live: 64
  - Protocol: TCP (0x06)
  - Header checksum: 0x74dd (correct)
  - Source: 192.168.0.2 (192.168.0.2)
  - Destination: 147.210.8.143 (147.210.8.143)
- Transmission Control Protocol, Src Port: 32770 (32770), Dst Port: imap (993), Seq: 53, Ack: 109, Len: 0

The packet bytes pane shows the raw data:

```
0000 00 09 5b 9a c4 94 00 90 4b 16 5d 24 08 00 45 00  ....[....K.]$.E.
0010 00 34 68 db 40 00 40 06 74 dd c0 a8 00 02 93 d2  .4h.@.@.t.....
0020 08 8f 80 02 03 e1 cd 9c f7 13 49 6f bd 4c 80 10  .........Io.L.
0030 3e 96 8d 1c 00 00 01 01 08 0a 00 0e ff e0 39 d0  >.....9.
0040 c4 ef  ..
```