Diagnostic réseau

1. Outils de diagnostics en couche réseau

a. ping

La célèbre commande **ping** rend toujours d'immenses services. Elle permet bien entendu de tester la connectivité IP de bout en bout, de tester la résolution DNS native, mais aussi d'obtenir des informations plus subtiles, comme par exemple l'indication qu'une route est inaccessible.

La commande **ping** exploite le protocole ICMP (*Internet Control Message Protocol*).

Exemple de réponse au ping

Dans cet exemple, la réponse au ping est différente selon que la route existe et que la machine cible du ping est indisponible, ou que la route est inconnue.

```
A:~$ route
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
192.168.200.0   *               255.255.255.0   U     1      0        0 eth0
A:~$ ping 172.17.18.19
connect: Network is unreachable
A:~$ route add -net 172.17.0.0 netmask 255.255.0.0 gw 192.168.200.254
A:~$ ping 172.17.18.19
PING 172.17.18.19 (172.17.18.19) 56(84) bytes of data.
From 172.17.18.19 icmp_seq=1 Destination Host Unreachable
From 172.17.18.18 icmp_seq=2 Destination Host Unreachable
A:~$
```

b. Indicateurs de la commande route

La commande **route**, utilisée pour configurer des routes statiques, fournit également des éléments de diagnostics. Elle permet de savoir quels sont les réseaux locaux ou distants (accessibles par une passerelle), ou encore de voir qu'une route est rejetée par le noyau. Ces informations sont données par les indicateurs de la commande route.

Commande route : principaux indicateurs	
U	Up : la route est active et exploitable.
H	Host : la cible est un hôte (et non un réseau).
G	Gateway : la cible est accessible par une passerelle.
D	Dynamic : la route a été configurée par un protocole de routage.
!	Le noyau a rejeté la route.

Exemple d'indications de la commande route

Toutes les routes sont actives et exploitables.

```
[root@beta ~]# route
Table de routage IP du noyau
Destination     Passerelle      Genmask         Indic Metric Ref    Use Iface
10.1.2.3        192.168.200.200 255.255.255.255 UGH   0      0        0 eth0
192.168.199.0   *               255.255.255.0   U     0      0        0 eth1
192.168.200.0   *               255.255.255.0   U     0      0        0 eth0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth1
```

```
default         192.168.200.254 0.0.0.0         UG    0      0        0 eth0
[root@beta ~]#
```

c. traceroute

La commande **traceroute** comme la commande **ping** permet de tester la connectivité avec un système distant, mais en donnant l'ensemble des routeurs qui permettent d'acheminer le paquet. En cas de problème de connectivité, on peut donc déterminer à quel endroit le paquet est bloqué ou s'est perdu.

Exemple d'utilisation de la commande traceroute

Dans cet exemple, on constate que pour atteindre la machine 192.168.199.10, il faut d'abord passer par le routeur 10.8.0.1.

```
tata@stotion:~$ traceroute 192.168.199.10
traceroute to 192.168.199.10 (192.168.199.10), 30 hops max, 60 byte packets
 1  10.8.0.1 (10.8.0.1)  44.928 ms  50.972 ms  51.015 ms
 2  192.168.199.10 (192.168.199.10)  51.056 ms  51.112 ms  51.149 ms
tata@stotion:~$
```

2. Outils de diagnostics en couches transport et application

a. netstat

La commande **netstat** permet d'observer les connexions établies avec le système local. Ces connexions peuvent être de type TCP, UDP, ou socket. Les connexions TCP et UDP sont en général établies avec des systèmes distants, alors que les sockets sont des fichiers de type particuliers qui servent de point d'échange entre des composants applicatifs sans passer par le réseau. Par exemple, le serveur d'affichage X qui était à l'origine une application client-serveur utilisée en réseau utilise désormais un socket pour les communications entre le client X et son serveur situés sur la même machine.

Dans un but de diagnostic du fonctionnement réseau, on s'intéressera principalement aux connexions TCP et UDP.

Syntaxe de la commande netstat pour voir les connexions actives

```
netstat -n
```

Où l'option -n, facultative, empêche la résolution inverse sur les adresses IP et sur les numéros de ports. L'affichage est plus rapide.

Observation des processus responsables de connexions réseau

```
netstat -p
```

Exemple d'utilisation de la commande netstat

Observons ici la commande netstat appelée toutes les secondes pour surveiller la connexion avec une application du système local. L'exemple propose un script contenant une boucle infinie et dont toutes les commandes sont placées sur une seule ligne. Si on a un besoin répété de cette commande sous cette forme, on aura intérêt à créer un fichier de script.

```
while true ; do clear ; netstat -an | head -20 ; sleep 1 ; done
```

On pourra sortir de cette boucle par la combinaison des touches [Ctrl] C.

b. nc

La commande **nc** ou **netcat** est un outil qui permet de lire ou écrire des données au travers de connexions réseau. Par exemple, si on est confronté à une application quelconque qui fonctionne en TCP sur le port 1234 et qu'on ne dispose d'aucun outil de diagnostic, **nc** permet d'établir une connexion sur le port TCP/1234, d'envoyer des données brutes, et d'observer la réponse du serveur.

Syntaxe de la commande nc

`nc -u adresse_ip port`

Commande nc : options et paramètres	
-u	Facultatif. Précise que l'on souhaite travailler en UDP. Si ce paramètre est omis, toutes les requêtes sont faites en TCP.
adresse_ip	L'adresse IP de la machine avec laquelle on souhaite communiquer.
port	Le port par lequel on souhaite s'adresser à la machine distante.

Exemple d'utilisation de nc pour interroger un serveur web

Dans cet exemple, le serveur interrogé répond bien en html au code http (GET /) qui lui demande d'afficher sa page d'accueil par défaut. On voit bien ici que l'utilisation de nc à des fins de diagnostic nécessite une connaissance précise des protocoles sous-jacents.

```
toto@ubuntu:~$ nc 172.17.6.26 80
GET /
<html><body><h1>It works!</h1></body></html>
toto@ubuntu:~$
```

3. Diagnostics et informations en couche application

a. lsof

La commande **lsof** permet d'établir la liste des fichiers ouverts par des processus sur un système. **lsof** exécutée sans options affiche simplement l'ensemble des fichiers appartenant à tous les processus actifs.

Affichage des fichiers ouverts par la commande xeyes

Les colonnes les plus directement utiles sont PID, USER et NAME.

```
A# lsof | grep xeyes
COMMAND PID  USER  FD  TYPE DEVICE SIZE/OFF NODE    NAME
xeyes   9584 toto  cwd DIR  8,3    12288    7258113 /tmp
xeyes   9584 toto  rtd DIR  8,3     4096          2 /
xeyes   9584 toto  txt REG  8,3    20416    2366803 /usr/bin/xeyes
xeyes   9584 toto  mem REG  8,3    22568    2362738 /usr/lib/libXfixes.so.3.1.0
xeyes   9584 toto  mem REG  8,3    39232    1966225 /usr/lib/libXcursor.so.1.0.2
xeyes   9584 toto  mem REG  8,3  1170770    6463538 /usr/lib/locale/fr_FR.utf8/
LC_COLLATE
xeyes   9584 toto  mem REG  8,3    19008    2146517 /lib/libuuid.so.1.3.0
xeyes   9584 toto  mem REG  8,3    22560    2364984 /usr/lib/libXdmcp.so.6.0.0
xeyes   9584 toto  mem REG  8,3    14488    2364981 /usr/lib/libXau.so.6.0.0
xeyes   9584 toto  mem REG  8,3    97904    2364446 /usr/lib/libICE.so.6.3.0
```

b. Journaux sur /var/log/syslog & /var/log/messages

Les fichiers **/var/log/syslog** sur les distributions d'origine Debian et **/var/log/messages** sur les distributions d'origine Red Hat concentrent l'essentiel des remontés de journaux toutes applications confondues. Ils sont alimentés par le démon **syslogd** pour Red Hat ou **rsyslogd** pour Debian et s'incrémentent à chaque événement subit ou provoqué par une application compatible **syslog**. Ainsi, les événements associés au réseau, qu'ils proviennent d'une application client-serveur ou de la gestion du réseau par le système elle-même seront probablement mentionnés dans ces fichiers journaux.

Sur les systèmes d'origine Debian, un fichier /var/log/deamon.log est spécifiquement réservé aux journaux d'activité des services.

Visualisation des événements relatifs aux cartes réseau

Les journaux représentent les principales sources d'information en cas de dysfonctionnement applicatif.

```
toto@ubuntu:/tmp$ grep eth /var/log/syslog | head
Jun 24 19:21:08 ubuntu NetworkManager: <info>  Activation (eth1)
starting connection 'Auto orange'
Jun 24 19:21:08 ubuntu NetworkManager: <info>  (eth1): device state change: 3 -> 4
(reason 0)
Jun 24 19:21:08 ubuntu NetworkManager: <info>  Activation (eth1) Stage 1 of 5
(Device Prepare) scheduled...
Jun 24 19:21:08 ubuntu NetworkManager: <info>  Activation (eth1) Stage 1 of 5
(Device Prepare) started...
Jun 24 19:21:08 ubuntu NetworkManager: <info>  Activation (eth1) Stage 2 of 5
(Device Configure) scheduled...
Jun 24 19:21:08 ubuntu NetworkManager: <info>  Activation (eth1) Stage 1 of 5
(Device Prepare) complete.
Jun 24 19:21:08 ubuntu NetworkManager: <info>  Activation (eth1) Stage 2 of 5
(Device Configure) starting...
Jun 24 19:21:08 ubuntu NetworkManager: <info>  (eth1): device state change: 4 -> 5
(reason 0)
Jun 24 19:21:08 ubuntu NetworkManager: <info>  Activation (eth1/wireless):
connection 'Auto orange' requires no security.  No secrets needed.
Jun 24 19:21:08 ubuntu NetworkManager: <info>  Activation (eth1) Stage 2 of 5
(Device Configure) complete.
```

4. Libpcap et les captures de paquets

a. La bibliothèque libpcap

Pour récupérer des informations précises sur le fonctionnement réseau d'une application, il arrive que l'on doive capturer directement l'ensemble des éléments qui passent sur le réseau. Les outils pour y parvenir sont nombreux sur tous les systèmes. En environnement Linux, ces outils s'appuient pour la plupart sur la bibliothèque **libpcap** qui fournit une interface de bas niveau normalisée pour la capture de paquets. **libpcap** a été créée à partir des premiers développements d'une commande de capture appelée **tcpdump**. Elle fut par la suite exploitée par de nombreux logiciels d'analyse réseau dont le célèbre **wireshark**.

b. tcpdump

tcpdump est un outil qui envoie sur la sortie standard (l'écran) une information résumée des captures réalisées par la carte réseau. **tcpdump** travaillant en temps réel (moyennant le temps de traitement par le programme), il est utile pour surveiller directement l'activité réseau d'une machine. Si on dirige les captures vers un fichier, alors les informations complètes des paquets capturées sont conservées et utilisables par d'autres outils compatibles avec le format **libpcap**.

Syntaxe de la commande tcpdump

`tcpdump -w fichier -i interface -s fenêtre -n filtre`

tcpdump : options et paramètres	
-w *fichier*	Facultatif : pour envoyer le résultat de la capture vers un fichier au format libpcap.
-i *interface*	Facultatif : pour réaliser la capture depuis une interface précise.
-s *fenêtre*	Facultatif : pour limiter la taille des trames capturées. Surtout utilisé avec le paramètre 0 (pas de limite).
-n	Facultatif : ne pas remplacer les valeurs numériques par des expressions littérales.
filtre	Détermine le trafic à capturer. Mots-clés principaux : host, port, src, dest.

Exemple d'utilisation de tcpdump

L'exemple ci-dessous nous montre des éléments de trafic capturés à la volée par tcpdump. Notez que la brièveté des informations proposées (ici des échanges liés au Spanning Tree Protocol entre commutateurs) ne permet pas d'analyse profonde, mais surtout de constater de visu la nature des informations échangées.

```
root@serveur:~$ tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth6, link-type EN10MB (Ethernet), capture size 96 bytes
10:07:59.961927
10:08:00.019503 STP 802.1d, Config, Flags [none], bridge-id
8007.00:25:46:b4:3c:80.800c, length 43
10:08:02.034712 STP 802.1d, Config, Flags [none], bridge-id
8007.00:25:46:b4:3c:80.800c, length 43
^C
3 packets captured
3 packets received by filter
0 packets dropped by kernel
root@serveur:~$
```

Cet exemple plus précis envoie vers un fichier au format libpcap les requêtes http vers un serveur à l'adresse IP 192.168.50.24.

```
root@serveur:~$ tcpdump -w fichier.cap -i eth0 -s 0 -n port 80 and host
192.168.50.24
root@serveur:~$
```

c. Wireshark

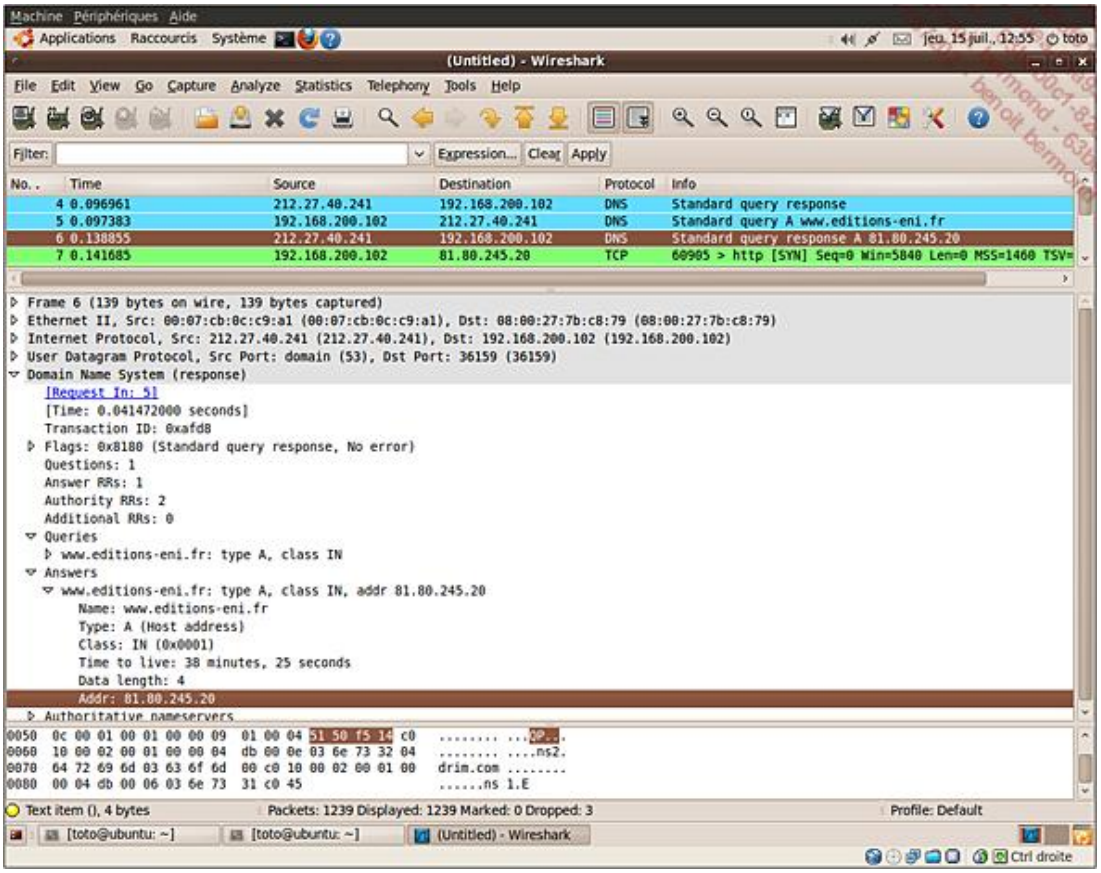
Wireshark (anciennement **ethereal**) est une application de capture de trames multiplate-forme disponible notamment sur les environnements Windows et Linux. **Wireshark** s'appuie sur la bibliothèque **libpcap** et permet de sauvegarder les données capturées à ce format ou d'exploiter des captures faites par d'autres utilitaires. **Wireshark** propose pour chacune des captures un découpage selon les couches du modèle OSI des informations capturées, ce qui est à la fois pratique et très pédagogique.

Procédure standard de capture avec wireshark

- Lancez l'applicatif Wireshark.
- Dans le menu Capture, choisissez Interfaces.
- Repérez la carte réseau à laquelle est associée votre adresse IP.
- Cliquez sur Start pour lancer la capture.
- Visualisez les paquets en cours de captures.
- Arrêtez la capture en cliquant sur Stop dans le menu Capture.

Exemple de capture de paquets avec wireshark

Notez l'écran divisé horizontalement en trois panneaux : le paquet à analyser, les détails couche par couche, et la valeur hexadécimale des informations capturées. Ici, on voit qu'il s'agit d'une requête DNS de type A pour la résolution du nom start.ubuntu.com.



Sur un réseau encombré, on risque d'être noyé sous une avalanche de paquets capturés qui n'ont pas forcément de rapport avec ce que l'on cherche. On gagnera en visibilité en appliquant un filtre d'affichage (champ **Filter** sur l'écran principal). Cette opération a l'avantage d'être réversible (bouton **Clear**).