Unidad 14:
NAT y Firewall con netfilter

Curso de Introducción a la administración de servidores GNU/Linux

Centro de Formación Permanente
Universidad de Sevilla
Abril-Junio 2010

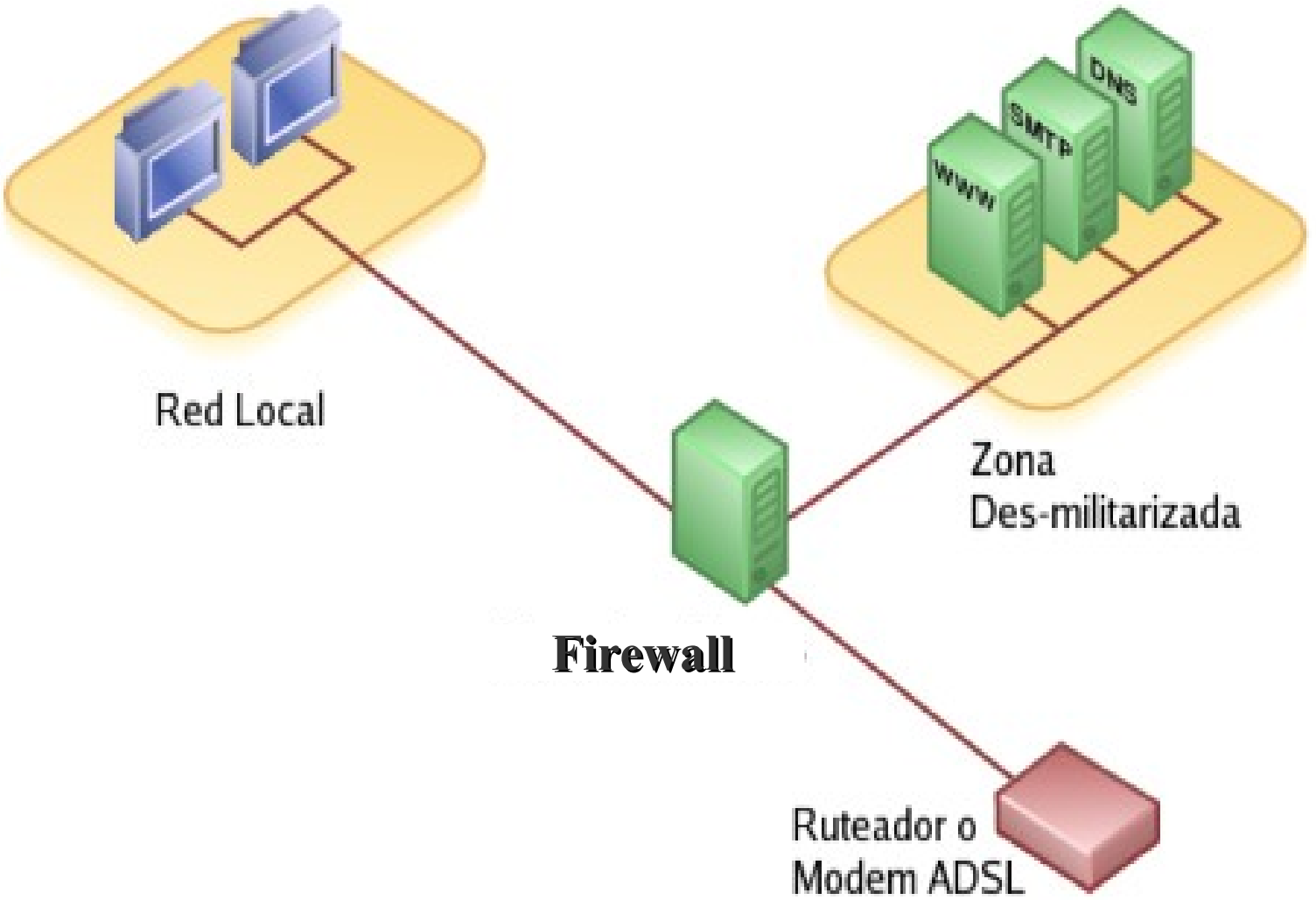
Contenidos

1. Introducción: Filtrado de Paquetes, NAT
2. Netfilter e IPtables
3. Filtrado de paquetes con netfilter
    1. Cabeceras IP y TCP, tcpdump, ethereal
    2. Routing: cómo trata el kernel los paquetes
    3. Construyendo filtros con IPtables
    4. Extensiones
4. NAT/Masquerading con netfilter
5. Otras opciones de firewall

1. Introducción

- *La red* no es segura, TCP/IP no lo es
- Conexiones cliente-servidor no encriptadas
- Servidores escuchando en todas las interfaces
- El usuario es anónimo, sólo se conoce la IP
- Un **firewall** nos ayuda a proteger nuestra LAN y a gestionar el acceso desde el exterior a los servicios 'abiertos al público'
- Pero con netfilter de Linux también se pueden hacer más cosas...

El firewall en la LAN



Filtrado de Paquetes

- Se trabaja con los paquetes IP directamente
- El filtrado de paquetes consiste en examinar la información del paquete y tomar una decisión de que haremos con él
- Podemos dejar que siga su camino, simplemente descartarlo, u otras cosas...
- Es la base de un firewall
  - controla el tipo de tráfico entre redes
  - asegura el acceso a equipos y servicios
  - fuente de información para el admin de la red

NAT

- *Network Address Translation*
- Es el proceso de cambiar la dirección IP (de origen o de destino) de los paquetes IP al pasar de una red a otra
- Se usa generalmente en *intranets*, dónde se usa un direccionamiento privado (con IPs reservadas)

2. Netfilter

- Funcionalidades de netfilter:
  - filtrado de paquetes, sin estados
  - filtrado de paquetes, con estados
  - cualquier tipo de traducción de direcciones IP y puertos (NAT y NAPT)
  - estructura fléxible y extensible por el usuario
  - varios niveles de API's para programación
  - muchos módulos y plug-ins mantenidos

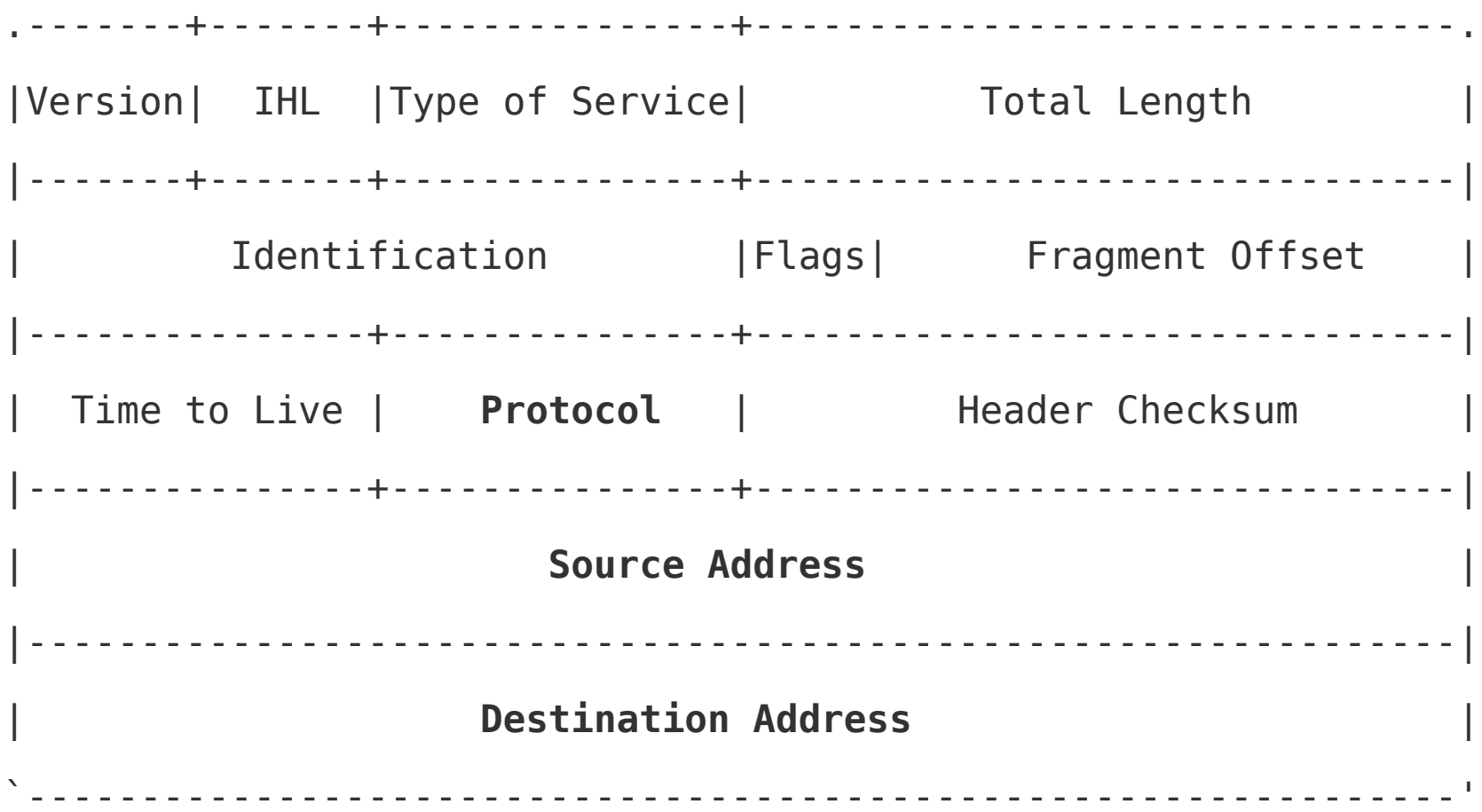
IPtables

- Para decidir que debe hacer con cada paquete, netfilter usa unas reglas definidas en una serie de tablas
- **IPtables** es el comando que el administrador usa para modificar esas tablas de filtrado y en consecuencia configurar netfilter
- En kernels 2.0 se usaba ipfwadm
- En kernels 2.2 se usaba ipchains
- En kernels 2.4 y 2.6 se usa iptables/netfilter
- Ojo, son parecidos pero no funcionan igual

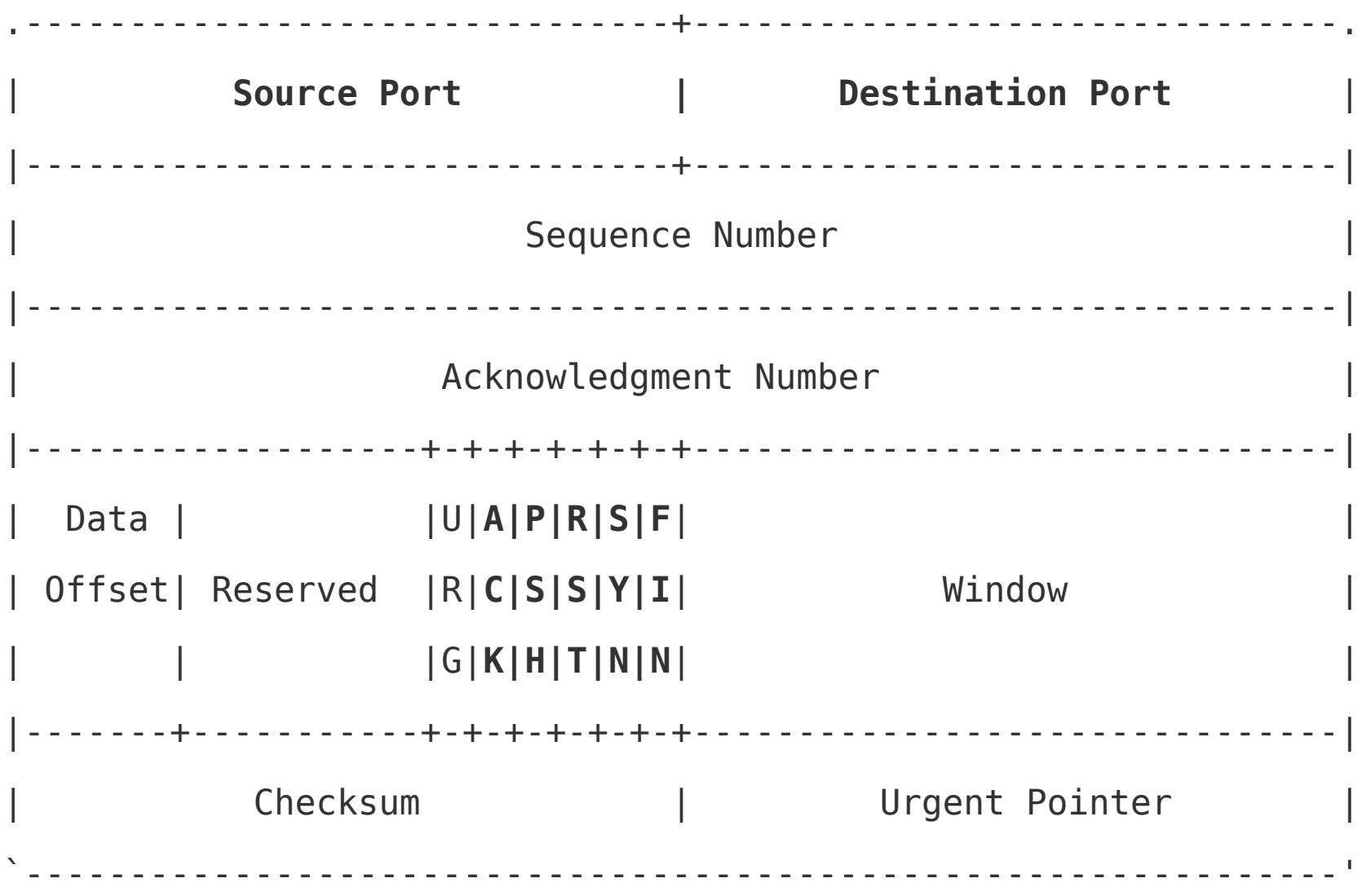
3. Filtrado de paquetes

- **Netfilter** es el software diseñado para hacer el filtrado de paquetes (y NAT) en Linux
- El filtrado de paquetes, al igual que el *routing*, se realiza desde el propio kernel
- Parte de netfilter se incluye ya en los kernels recientes, versiones 2.4 y 2.6
- netfilter examina paquete a paquete todo el tráfico que pasa por el equipo

3.1 Cabecera IP



Cabecera TCP

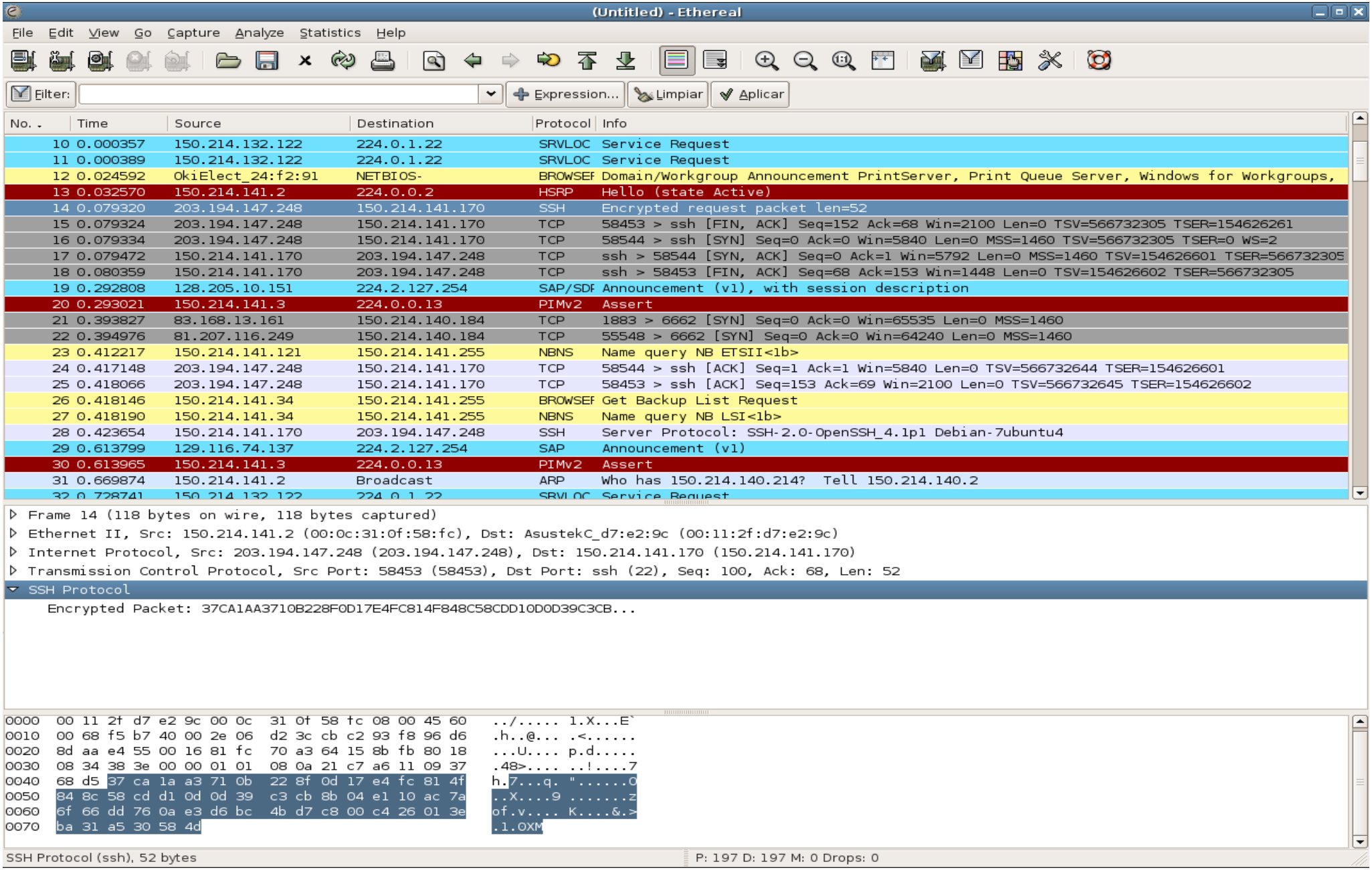


tcpdump

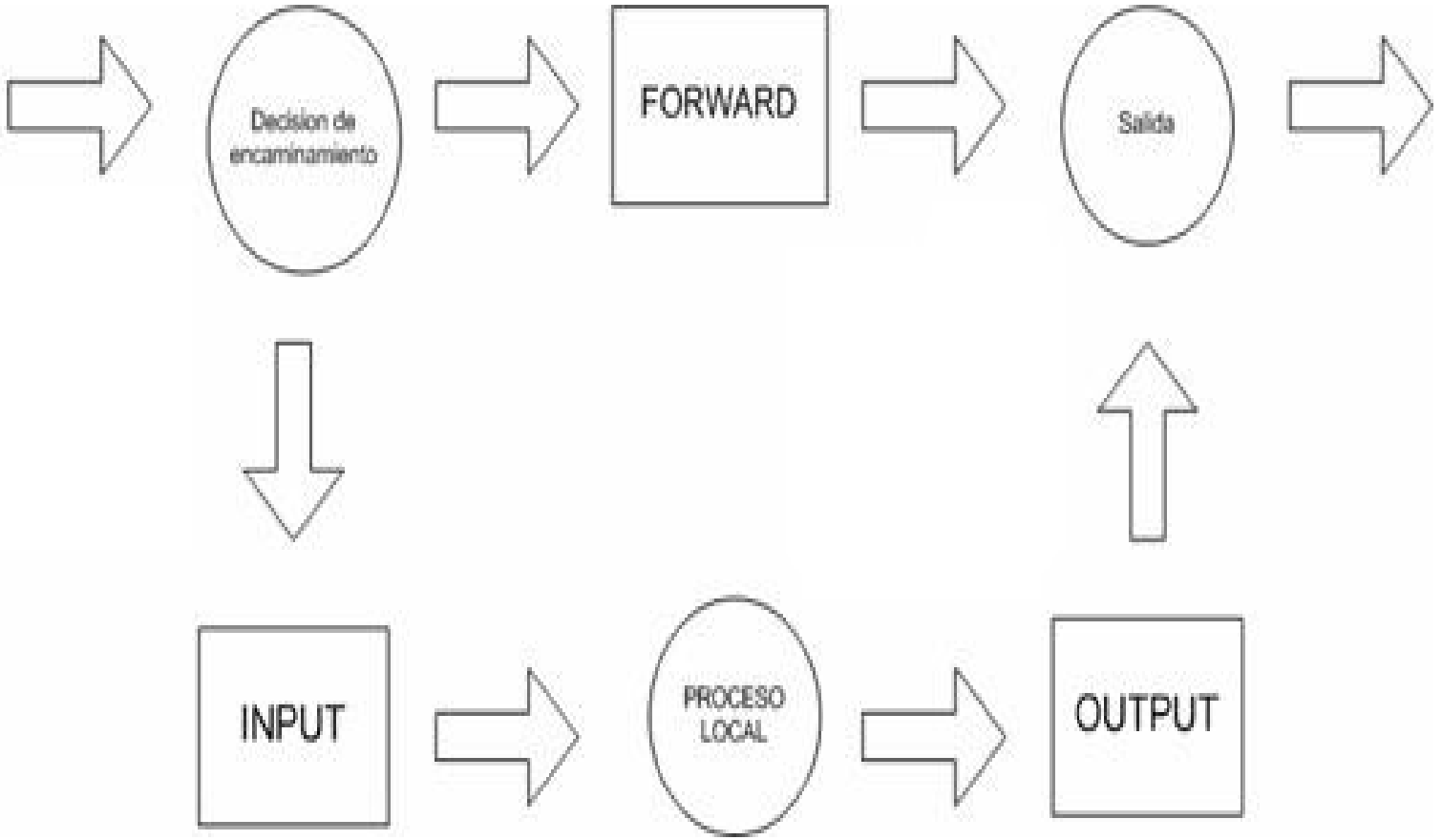
- La herramienta **tcpdump** nos permite poner un interfaz en *modo promiscuo* y mostrar 'en tiempo real' información de los paquetes que llegan y salen por esa interfaz.
- Formato: tcpdump [-n] [-i int] _reglas_, con _reglas_=_regla_ [and|or _reglas_] y _regla_= tcp, udp, icmp, dst host IP, src host IP, host IP, dst net IP/m, src net IP/m, net IP/m, dst port P, src port P, port P, portrange P1-P2...
- Cada regla puede llevar un “not” delante.

```
# tcpdump -n -i eth0 dst port 80 and
        not src host 150.214.141.170
```

ethereal


(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Limpiar Aplicar
No. Time Source Destination Protocol Info
10 0.000357 150.214.132.122 224.0.1.22 SRVLOC Service Request
11 0.000389 150.214.132.122 224.0.1.22 SRVLOC Service Request
12 0.024592 OkiElect_24:f2:91 NETBIOS- BROWSEF Domain/Workgroup Announcement PrintServer, Print Queue Server, Windows for Workgroups,
13 0.032570 150.214.141.2 224.0.0.2 HSRP Hello (state Active)
14 0.079320 203.194.147.248 150.214.141.170 SSH Encrypted request packet len=52
15 0.079324 203.194.147.248 150.214.141.170 TCP 58453 > ssh [FIN, ACK] Seq=152 Ack=68 Win=2100 Len=0 TSV=566732305 TSER=154626261
16 0.079334 203.194.147.248 150.214.141.170 TCP 58544 > ssh [SYN] Seq=0 Ack=0 Win=5840 Len=0 MSS=1460 TSV=566732305 TSER=0 WS=2
17 0.079472 150.214.141.170 203.194.147.248 TCP ssh > 58544 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 TSV=154626601 TSER=566732305
18 0.080359 150.214.141.170 203.194.147.248 TCP ssh > 58453 [FIN, ACK] Seq=68 Ack=153 Win=1448 Len=0 TSV=154626602 TSER=566732305
19 0.292808 128.205.10.151 224.2.127.254 SAP/SDF Announcement (v1), with session description
20 0.293021 150.214.141.3 224.0.0.13 PIMv2 Assert
21 0.393827 83.168.13.161 150.214.140.184 TCP 1883 > 6662 [SYN] Seq=0 Ack=0 Win=65535 Len=0 MSS=1460
22 0.394976 81.207.116.249 150.214.140.184 TCP 55548 > 6662 [SYN] Seq=0 Ack=0 Win=64240 Len=0 MSS=1460
23 0.412217 150.214.141.121 150.214.141.255 NBNS Name query NB ETSII<1b>
24 0.417148 203.194.147.248 150.214.141.170 TCP 58544 > ssh [ACK] Seq=1 Ack=1 Win=5840 Len=0 TSV=566732644 TSER=154626601
25 0.418066 203.194.147.248 150.214.141.170 TCP 58453 > ssh [ACK] Seq=153 Ack=69 Win=2100 Len=0 TSV=566732645 TSER=154626602
26 0.418146 150.214.141.34 150.214.141.255 BROWSEF Get Backup List Request
27 0.418190 150.214.141.34 150.214.141.255 NBNS Name query NB LSI<1b>
28 0.423654 150.214.141.170 203.194.147.248 SSH Server Protocol: SSH-2.0-OpenSSH_4.1p1 Debian-7ubuntu4
29 0.613799 129.116.74.137 224.2.127.254 SAP Announcement (v1)
30 0.613965 150.214.141.3 224.0.0.13 PIMv2 Assert
31 0.669874 150.214.141.2 Broadcast ARP Who has 150.214.140.214? Tell 150.214.140.2
Frame 14 (118 bytes on wire, 118 bytes captured)
Ethernet II, Src: 150.214.141.2 (00:0c:31:0f:58:fc), Dst: AsustekC_d7:e2:9c (00:11:2f:d7:e2:9c)
Internet Protocol, Src: 203.194.147.248 (203.194.147.248), Dst: 150.214.141.170 (150.214.141.170)
Transmission Control Protocol, Src Port: 58453 (58453), Dst Port: ssh (22), Seq: 100, Ack: 68, Len: 52
SSH Protocol
Encrypted Packet: 37CA1AA3710B228F0D17E4FC814F848C58CDD10D0D39C3CB...
0000 00 11 2f d7 e2 9c 00 0c 31 0f 58 fc 08 00 45 60 ../..... 1.X...E`
0010 00 68 f5 b7 40 00 2e 06 d2 3c cb c2 93 f8 96 d6 .h..@... .<......
0020 8d aa e4 55 00 16 81 fc 70 a3 64 15 8b fb 80 18 ...U.... p.d.....
0030 08 34 38 3e 00 00 01 01 08 0a 21 c7 a6 11 09 37 .48>.... ..!....7
0040 68 d5 37 ca 1a a3 71 0b 22 8f 0d 17 e4 fc 81 4f h.7...q. "......O
0050 84 8c 58 cd d1 0d 0d 39 c3 cb 8b 04 e1 10 ac 7a ..X....9 ....z
0060 6f 66 dd 76 0a e3 d6 bc 4b d7 c8 00 c4 26 01 3e of.v.... K....&.>
0070 ba 31 a5 30 58 4d .1.0XM
SSH Protocol (ssh), 52 bytes
P: 197 D: 197 M: 0 Drops: 0

3.2 Routing


Decision de encaminamiento
FORWARD
Salida
INPUT
PROCESO LOCAL
OUTPUT

El paso de los paquetes

- Si llega un paquete se mira destino: *routing*
- Si el destino es el propio equipo, se pasa a la *chain* de INPUT, y si continua su camino, llegará al proceso local que corresponda
- Si el destino es alcanzable por otra interfaz del equipo y el *forwarding* está activo, se pasa a la chain de FORWARD, si es aceptada saldrá por la interfaz correspondiente.
- Los paquetes generados localmente pasan directamente al chain OUTPUT y, si lo superan, finalmente al interfaz de salida.

3.3 Construyendo filtros con IPtables

- Contamos con 3 chains iniciales (INPUT, FORWARD y OUTPUT) y podemos añadir más
- Cada chain tendrá una serie de reglas (*rules*) y una política por defecto (*policy*)
- Los paquetes al entrar en una *chain* se irán evaluando por las reglas y si alguna se cumple se tomará la acción (*target*) que diga la regla.
- Si ninguna regla se aplica, se usará la política por defecto

Iptables: *targets* frecuentes

- ACCEPT: acepta el paquete, sale del *chain*
- DROP: elimina el paquete, sale del *chain*
- LOG: guarda un log, pero continua evaluando
- REJECT: rechaza el paquete, enviando un 'mensaje de error' al origen, sale del *chain*
- <*chain*>: se puede especificar otro *chain* y se evaluarán entonces las reglas de ese.

Iptables: manejando chains

- Comandos para operaciones con chains:

-P: fija política por defecto en la chain

-L: lista de chains y reglas

-F: elimina todas las reglas de una chain

-Z: reset a cero de contadores

-N: nueva *chain de usuario*

-X: elimina *chain de usuario* vacía

- Ejemplos:

iptables -P FORWARD DROP

iptables -F OUTPUT

Iptables: manejando reglas

- Comandos para manejar reglas de una chain:

-A: añade una regla al final de la chain

-I: inserta una regla en una posición determinada

-D: borra una regla, por posición o especificándola

-R: reemplaza una regla en una posición por otra

- Ejemplos:

iptables -A INPUT <regla> (añade al final)

iptables -I INPUT 2 <regla> (añade en 2ª pos)

iptables -D OUTPUT 3 (borra la 3ª regla)

Iptables: definiendo reglas

- La regla define las condiciones que debe cumplir un paquete y la acción a tomar. Se forman con los siguientes parámetros:

-p [!] protocolo: tcp, udp, icmp o all

-s [!] IP[/máscara]: dirección/red origen

-d [!] IP[/máscara]: dirección/red destino

-j target: marca la acción/target a ejecutar

-i [!] interfaz: interfaz de entrada del paquete

-o [!] interfaz: interfaz de salida del paquete

-m extensión: usa una extensión para la regla

Iptables: un primer ejemplo

- Para probar, queremos filtrar en nuestro equipo el tráfico ICMP (ping) a nuestra IP eth.
- La chain implicada es la de INPUT y la acción será DROP (o REJECT)
- El protocolo es ICMP. El origen cualquiera. El destino 10.1.15.120. El interfaz de entrada será “eth0”, si sólo hay ese.
- El comando quedaría:

```
# iptables -A INPUT -p icmp -d 10.1.15.120 -i eth0 -j DROP
```

Iptables: un primer ejemplo

```
# iptables -L -n (muestra IPs, no resuelve DNS)
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            10.1.15.120

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

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

- Comprobemos si funciona.
- ¿Desde nuestro equipo responde el ping?
- ¿Y desde otro equipo? ¿Y por otra interfaz?
- ¿Cómo filtramos el icmp completamente?

Iptables: segundo ejemplo

- Queremos filtrar ahora el ping al localhost (sólo se llega desde nuestro propio equipo). La regla sería la siguiente...

 # iptables -D INPUT 1

 # iptables -A INPUT -p icmp -s 127.0.0.1 -i lo -j DROP

- Comprobemos si filtra, ping al 127.0.0.1
- ¿Por qué funciona? ¿Los procesos locales no iban directos al OUTPUT? ¿Qué estoy filtrando?

Iptables: segundo ejemplo

```
    # iptables -F
    # tcpdump -n -i lo icmp
20:06:05.805823 IP 127.0.0.1 > 127.0.0.1: ICMP echo request, id 49698, seq 1, length 64
20:06:05.805878 IP 127.0.0.1 > 127.0.0.1: ICMP echo reply, id 49698, seq 1, length 64
```

- Quitemos el firewall y veamos que está pasando por la interfaz de “loopback”.
- **tcpdump** nos muestra el tráfico del loopback (-i lo), en este caso sólo el tráfico icmp
- Haciendo un ping vemos que realmente hay una salida (el “echo request”) y una entrada (el “echo reply”)
- Si filtramos en INPUT, eliminamos la respuesta! ¿Y en OUTPUT, como sería?

Iptables: consideraciones

- Lo más seguro es cerrar todo (policy DROP) e ir abriendo lo imprescindible, probando servicio a servicio
- Si es un firewall dedicado, en el gateway por ejemplo, mucho cuidado sobre todo con lo que se deja entrar en la red
- Si hay multiples interfaces, comprobar que el tráfico llega/sale por el interfaz correcto
- ¿Realmente necesito el FORWARD? Recuerda activarlo
- Script del firewall para arranque automático

3.4 Extensiones

- Hay multitud de extensiones mantenidas, se cargan de forma automática los módulos.
- Extensión TCP (-p tcp):

  --**sport** [!] <rango_puertos>: puerto/s de origen

  --**dport** [!] <rango_puertos>: puerto/s de destino

  --**tcp-flags** [!] <lista_flags> <flags>: comprueba que de la <lista_flags> están activos <flags>

  [!] --**syn**: una abreviatura de “--tcp-flags SYN,RST,ACK SYN”, paquetes de inicio de conexión del protocolo TCP.

  Ejemplo: -p tcp --syn --dport 20:25 -s 10.1.15.120

  [inicios de conexión desde 10.1.15.120 a tcp 20 al 25]

Extensiones

- Extensión UDP (-p udp):

  --**sport** [!] <rango_puertos>: puerto/s de origen

  --**dport** [!] <rango_puertos>: puerto/s de destino

- Extensión ICMP (-p icmp):

  --**icmp-type** [!] <tipo>: selecciona el tipo de icmp

  Ver # iptables -p icmp –help

- Extensión MAC (-m mac):

  --**mac-source** [!] <eth_addr>: comprueba la MAC address de origen (en INPUT)

Extensiones

- Extensión STATE (-m state)

  [!] --**state** <estado>: comprueba el estado de la conexión al que se refiere el paquete. Puede ser:

  NEW: creando una nueva conexión

  ESTABLISHED: paquete que pertenece a una conexión ya abierta

  RELATED: relacionado con otra conexión ya establecida, pero no es la misma conexión (ej: conexión de datos en FTP o respuesta al PING)

  INVALID: paquetes no identificados

  Ejemplo: # iptables -A FORWARD -i ppp+ -m state ! --state NEW -j DROP

Extensiones

- Extensión LIMIT (-m limit)

  --**limit** <numero>[/<tiempo>]: máx. nº de coincidencias por segundo, minuto, hora o día.

  Se usa para no guardan demasiados LOG o para evitar ciertos tipos de ataques o escaneos de puertos.

  - Ejemplo:

  # iptables -A FORWARD -j LOG -m limit --limit 10/m

  - Ejemplo:

  # iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit --limit 1/s -j ACCEPT

4. NAT con netfilter

- Hay dos tipos fundamentales:
  - SNAT: Source NAT, cambiamos 'de dónde viene' el paquete. Se realiza después del *routing*. Masquerading es un tipo específico de SNAT.
  - DNAT: Destination NAT, cambiamos 'adonde va' el paquete. Se hace siempre antes del routing. Este tipo incluye el NAPT (forward de puertos), balanceo de carga y proxy transparente.
- Tabla aparte en IPtables, “iptables -t nat” y dos chains: PREROUTING, POSTROUTING.
- Necesita que el kernel haga FORWARD:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Reglas para hacer NAT

- Es imprescindible el '-t nat'
- Los parámetros fundamentales siguen siendo válidos:

  -p, -s, -d, --sport, --dport, etc...

- Nuevas chains

  -A PREROUTING, -A POSTROUTING

- Nuevos targets

  -j SNAT, -j DNAT, -j MASQUERADE

Haciendo SNAT

- El SNAT ocurre en el POSTROUTING, se evalúa justo antes de la salida del paquete, así que el routing y el resto de chains verán el paquete antes del cambio.

  -j SNAT [-o <int>] --to-source <nueva-IP>

- Ejemplo, NAT para una intranet, a través de un gateway conectado a internet:

```
# iptables -t nat -A POSTROUTING -s 10.1.15.120/22
  -o eth0 -j SNAT --to-source 150.214.141.120
```

Haciendo Masquerading

- Masquerade es un tipo específico de SNAT, dónde no hace falta especificar la <nueva-IP>, sino que la coje del interfaz por el que sale, se usa con módems, nunca con direcciones IP estáticas.

  -j MASQUERADE -o <int>

- Ejemplo, conectar la intranet a internet por un módem instalado en el Linux:

```
# iptables -t nat -A POSTROUTING -s 192.168.10.0/24
  -o ppp0 -j MASQUERADE
```

Haciendo DNAT

- El DNAT ocurre en el PREROUTING, se evalúa justo a la entrada del paquete, así que el routing y el resto de chains verán la dirección de destino ya cambiada.

  -j DNAT [-i <int>] --to-destination <nueva-IP[:puerto]>

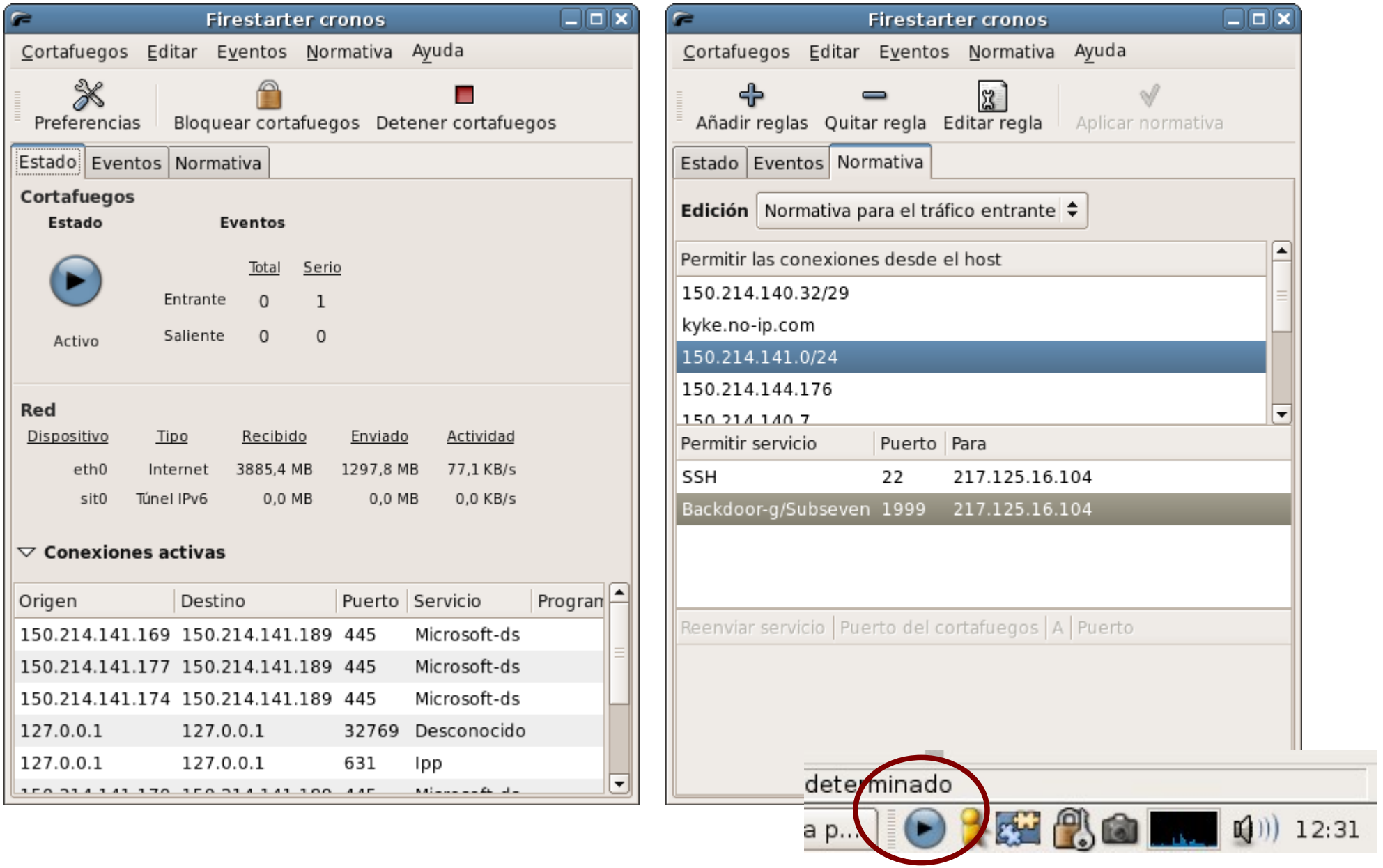
- Ejemplo, DNAT para acceder desde el exterior a un servidor web colocado en un equipo de la intranet en el puerto 8080:

```
# iptables -t nat -A PREROUTING -i ppp0 -p tcp
  --dport 80 -j DNAT --to-destination
  10.1.15.99:8080
```

5. Otras opciones: firestarter

- Firestarter es un software de firewall sencillo pero potente. Posee una interfaz gráfica y permite las restricciones más habituales, a nivel de IPs y puertos, para tráficos entrante y saliente.
- Todo esto lo implementa con netfilter.
- Resulta interesante en servidores pequeños.
- Además proporciona otros servicios: un asistente, la “conexión compartida a internet” (NAT), hace de servidor DHCP y funciones de priorización del tráfico (TdS)

firestarter



shorewall

- Es una solución intermedia, a partir de una descripción de las redes y de los servicios y restricciones que queremos implementar en el equipo, genera una serie de reglas para el netfilter del kernel.
- La configuración (en /etc/shorewall) se basa en varios ficheros de texto donde en cada uno se detallan interfaces, redes, modulos, políticas, servicios, redirecciones, etc... esto se procesa y se generan las reglas de netfilter.
- Es muy potente y flexible.