Netfilter e iptables

”Allacciate le cinture”

Daniele Iamartino
<otacon22@poul.org>

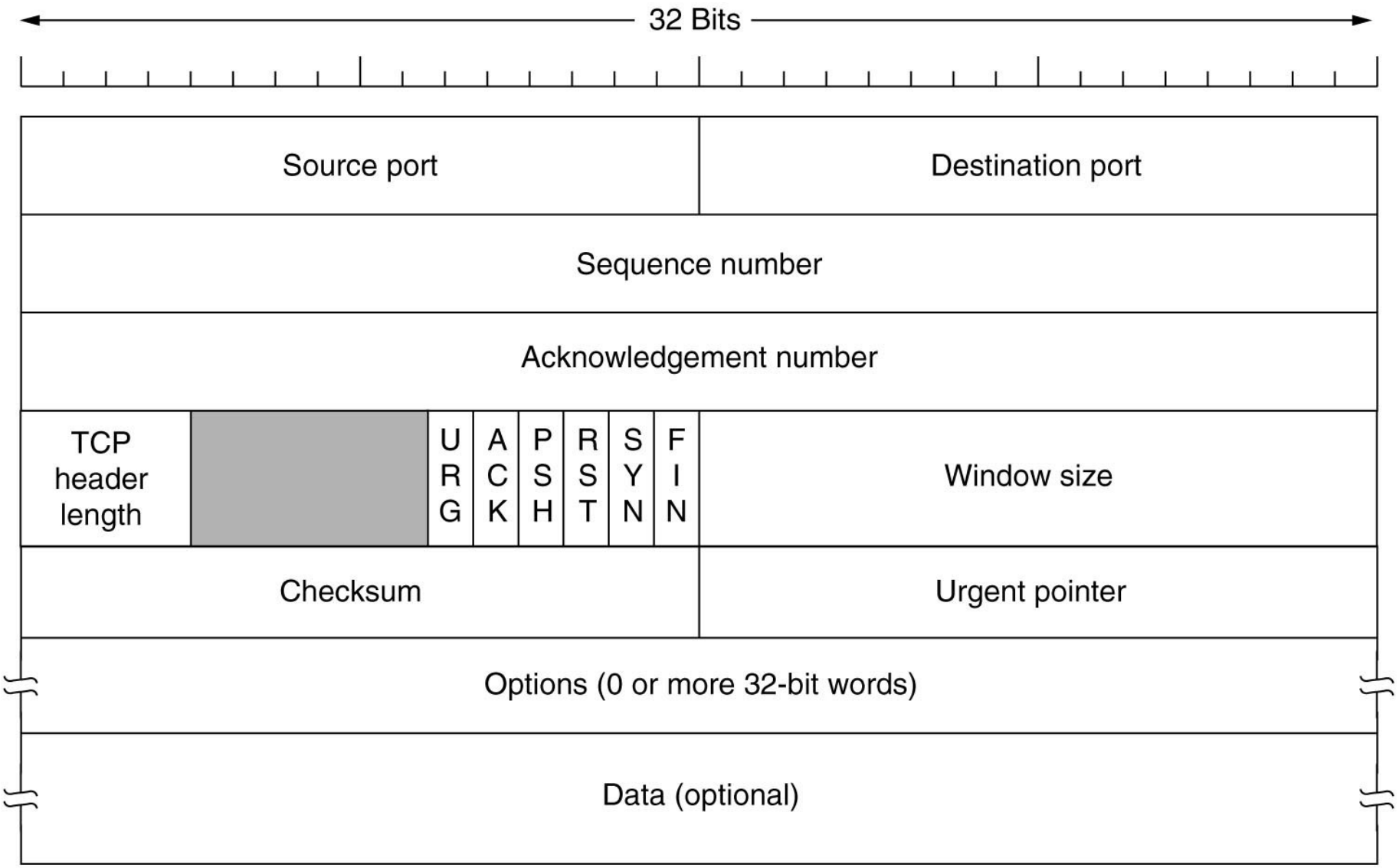
IPv4

+	Bits 0–3	4–7	8–15	16–18	19–31
0	Version	Internet Header length	Type of Service (now DiffServ and ECN)	Total Length	
32	Identification			Flags	Fragment Offset
64	Time to Live		Protocol	Header Checksum	
96	Source Address				
128	Destination Address				
160	Options (optional)				
160 o 192+	Data				

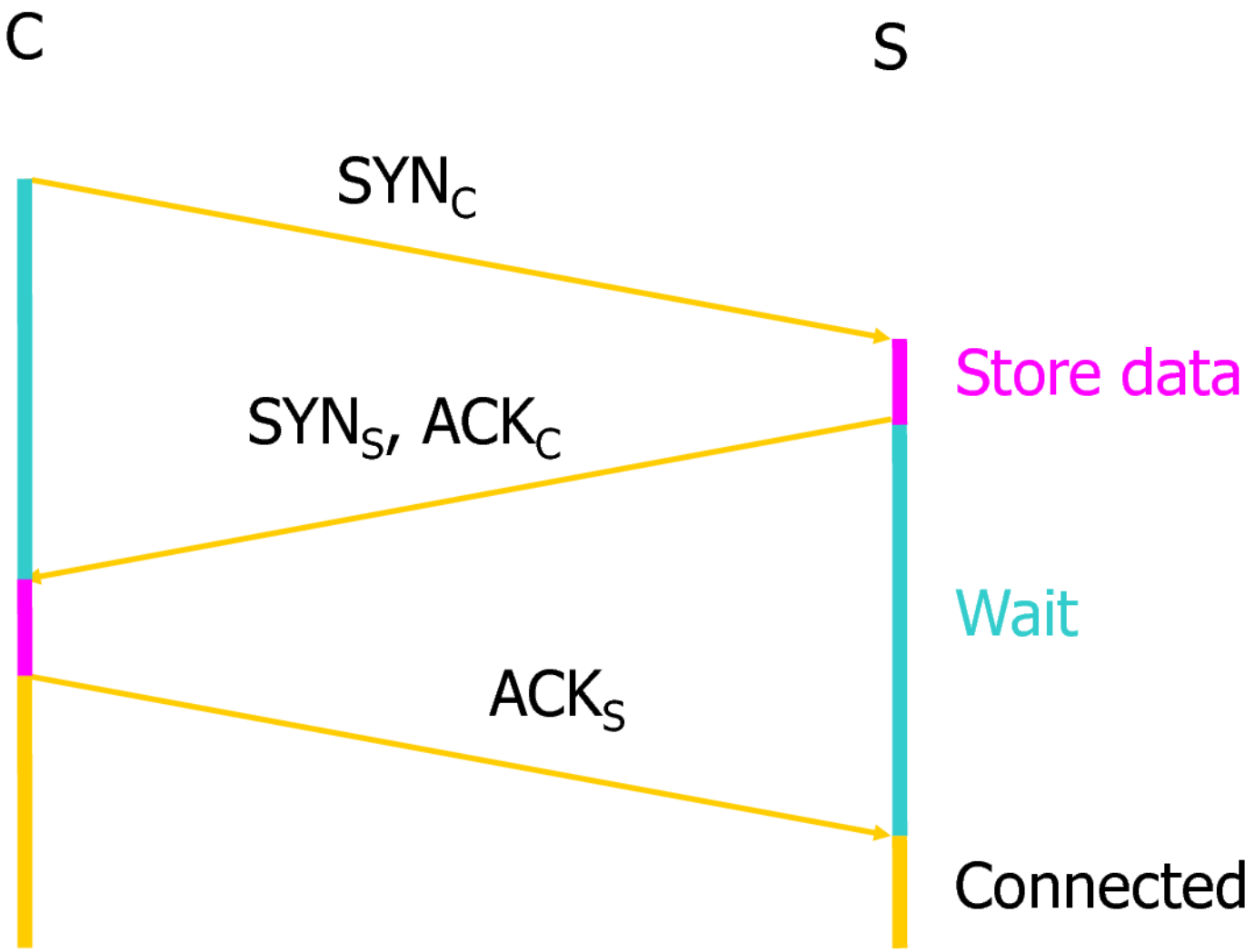
IPv6

+	Bits 0–3	4–11	12–15	16–23	24–31
0-31	Version	Traffic Class	Flow Label		
32-63	Payload Length			Next Header	Hop Limit
64 - 191	Source Address (128 bit)				
192 - 319	Destination Address (128 bit)				

TCP


32 Bits
Source port
Destination port
Sequence number
Acknowledgement number
TCP header length
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options (0 or more 32-bit words)
Data (optional)

TCP handshake



UDP

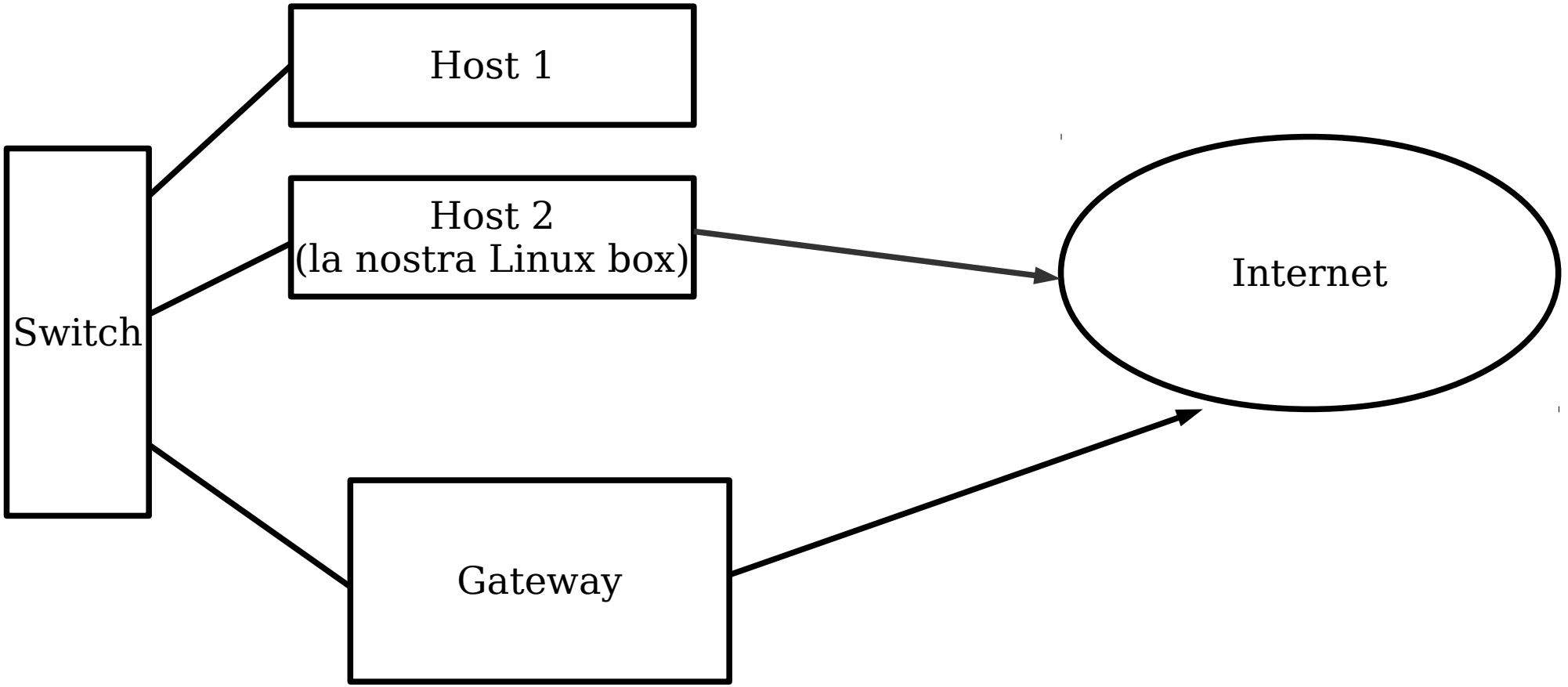
$\leftarrow$ 32 bit $\rightarrow$

Source Port (16 bit)	Destination Port (16 bit)
UDP Lenght (16 bit)	Checksum (16 bit)
Dati (facoltativi)	

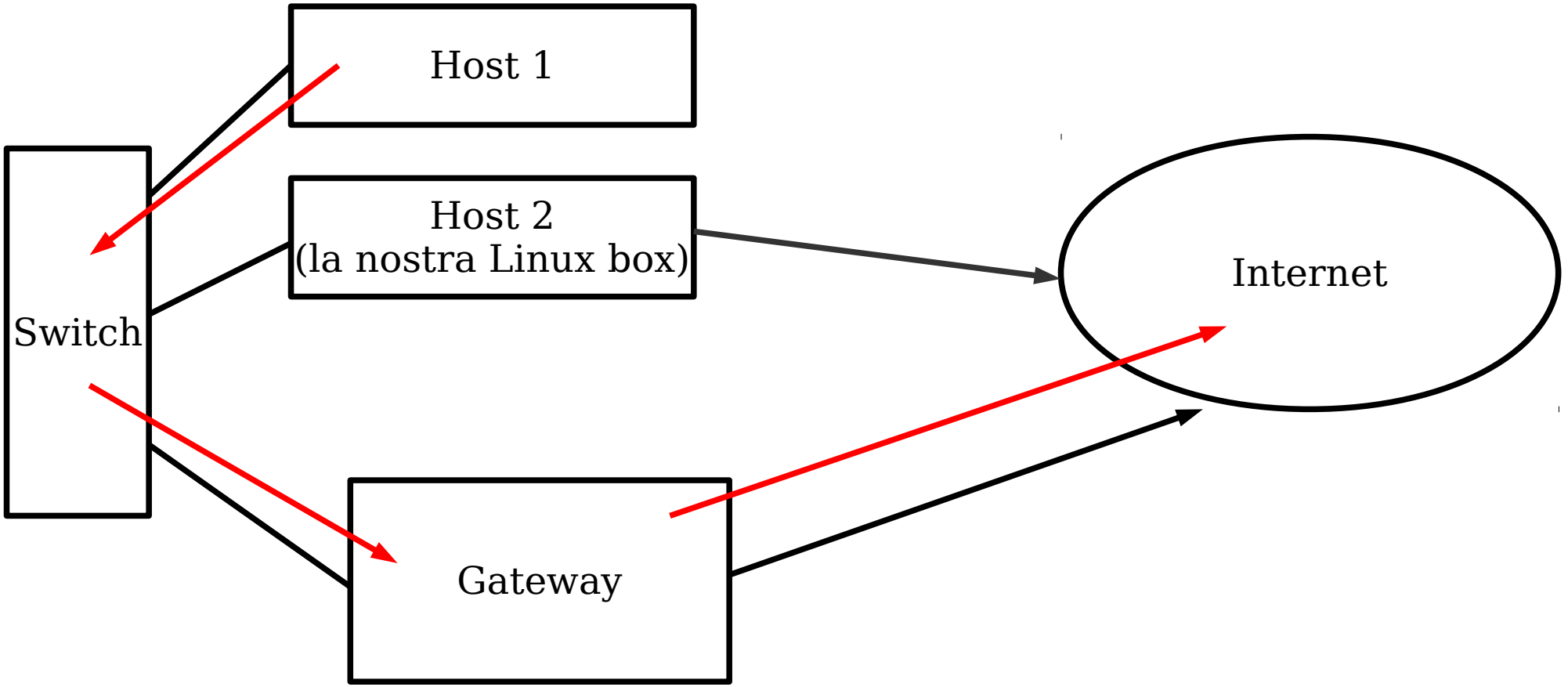
Routing su GNU/Linux

- La vostra macchina GNU/Linux è anche un router, lo sapevate?
- La nostra macchina possiede una serie di regole di routing che gli dicono come raggiungere certe destinazioni.
- Solitamente in una rete locale LAN abbiamo un unico router di frontiera (chiamato anche gateway) e gli host all'interno della LAN hanno una sola regola di (default) routing: “invia tutto al gateway”
- Qualunque macchina linux può potenzialmente operare da router e inoltrare il traffico per conto di altri.
- Per abilitare il “forwarding” di pacchetti ip dobbiamo abilitarlo con
  - **echo “1” > /proc/sys/net/ipv4/ip_forward**
  - In questo modo altri host potranno utilizzare noi come router (o gateway di default) e noi provvederemo a inoltrare il traffico che ci arriva in base alle nostre regole di routing.

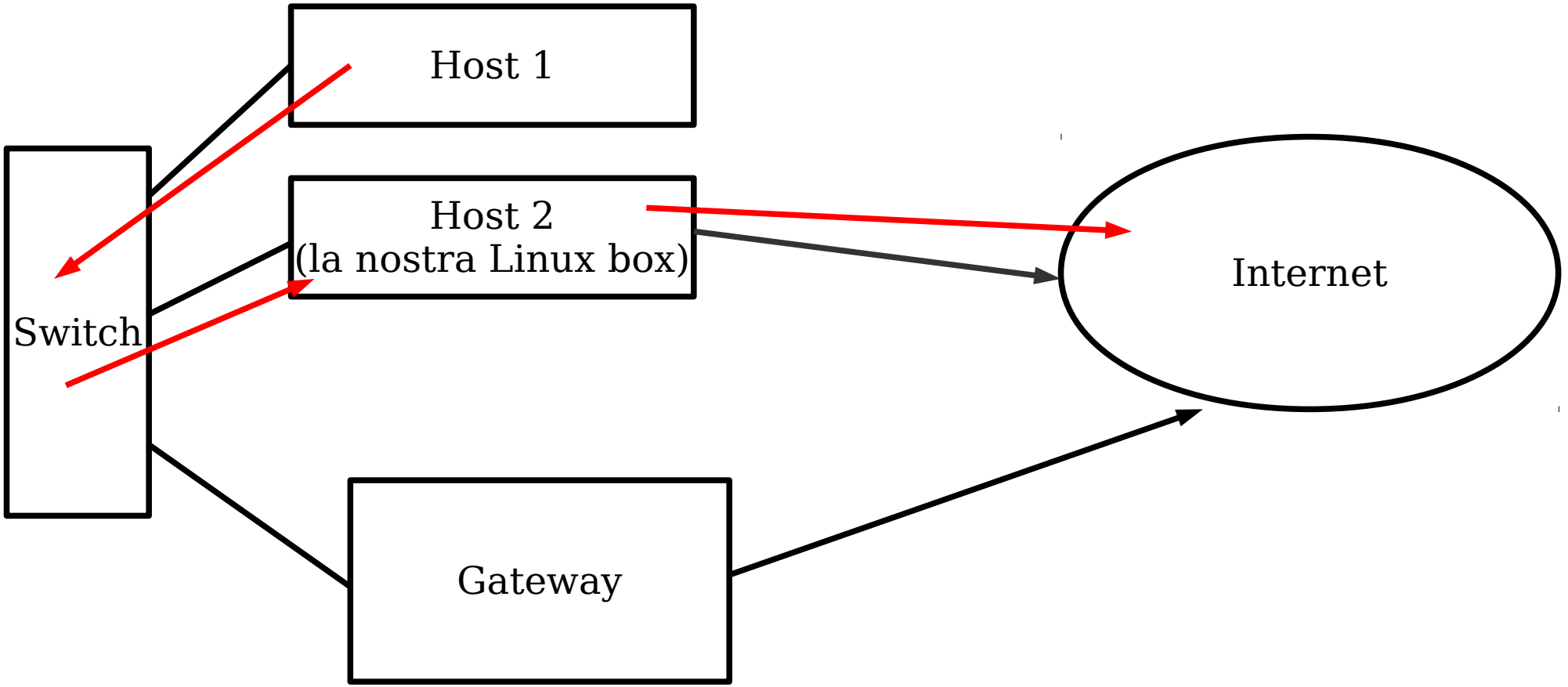
Routing su GNU/Linux



Routing su GNU/Linux



Routing su GNU/Linux



Cos'è un firewall

- Essenzialmente è un programma **che si occupa di decidere se accettare o meno i pacchetti** in transito su un host.
- Eventualmente effettua anche delle **modifiche** sui pacchetti o altre operazioni (anche se l'operazione principale è lo scarto o il passaggio).
- In GNU/Linux una parte del kernel, chiamata **Netfilter**, si occupa di effettuare queste operazioni quando i pacchetti transitano nel kernel.
- **iptables** è l'interfaccia utente che ci permette di configurare varie parti di netfilter. È il sostituto del vecchio ipchains e del vecchio ipfwadm.
- Netfilter introuduce funzionalità di **stateful** packet filtering (SPF), cioè è in grado di tenere traccia delle connessioni che lo attraversano e del loro stato.

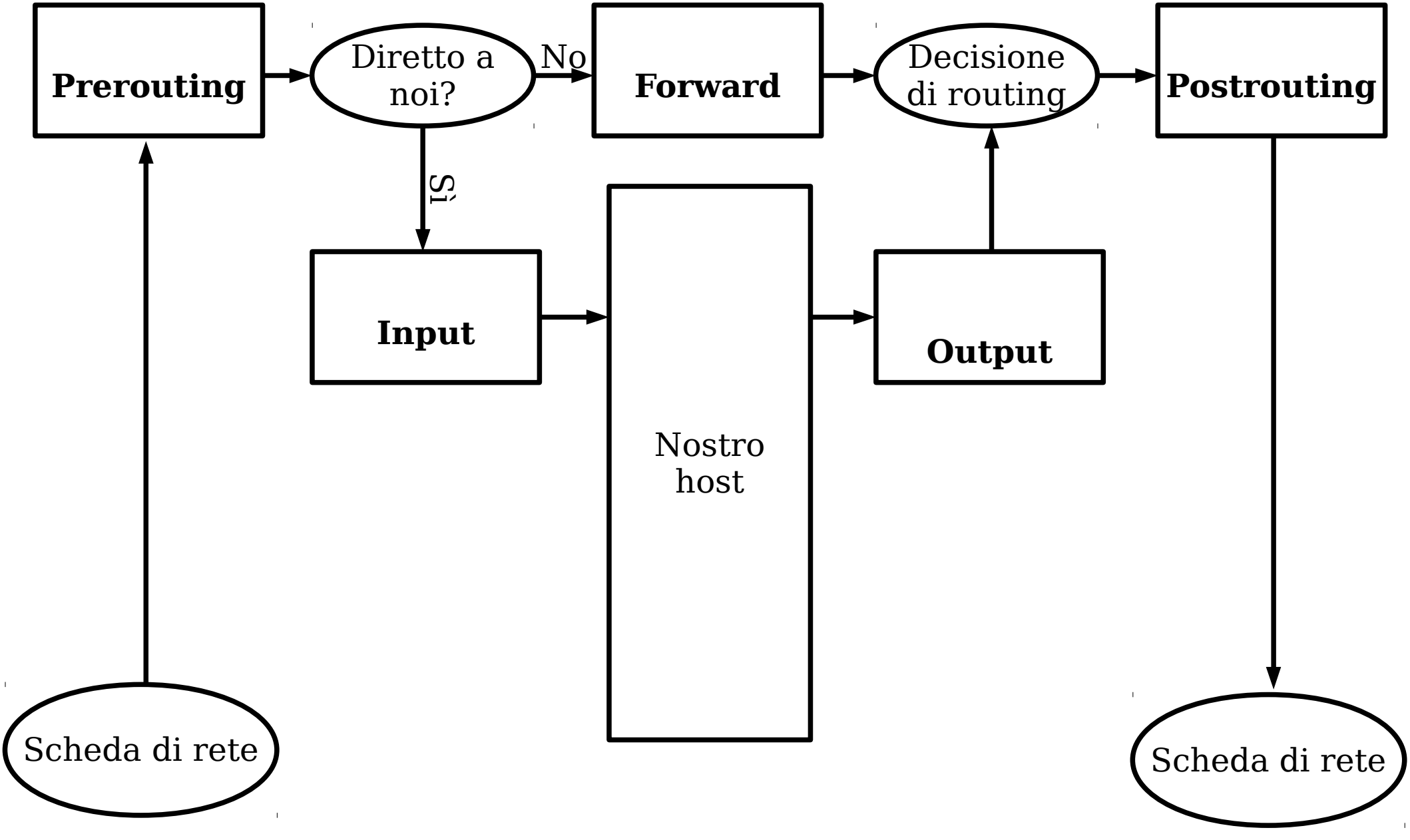
Perché dovrebbe interessarmi tutto ciò?

- Configurare un router/gateway/firewall di una rete locale
- Proteggere a valle di una rete (anche già protetta) il mio server
- Fare al volo cose come condividere la mia connesione wifi via ethernet o bloccare il traffico di certi utenti/programmi, controllare meglio il traffico sulla propria macchina
- Effettuare operazioni avanzate di amministrazione di rete e altre porcherie

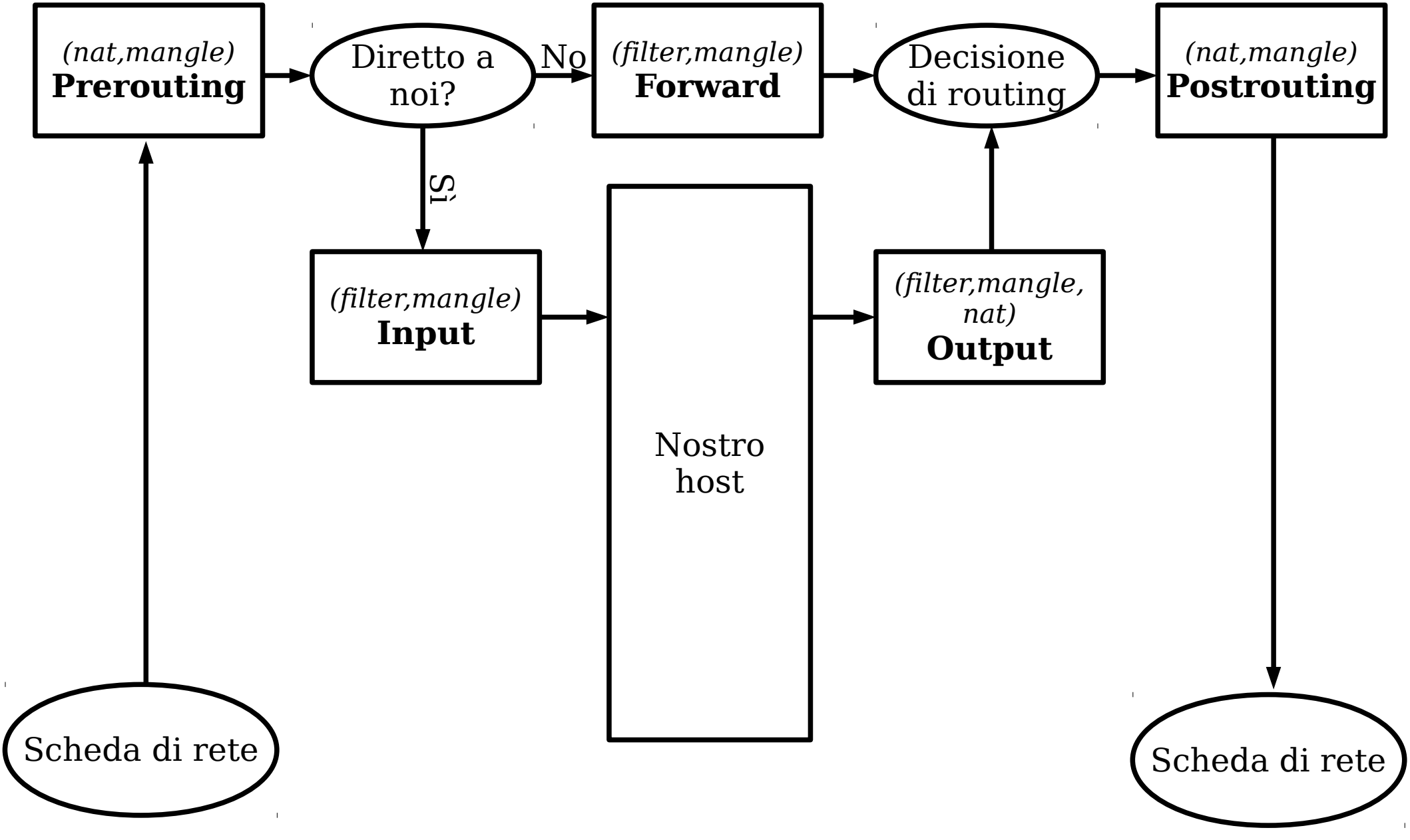
Struttura di netfilter

- Linux arriva già “con le batterie incluse” e comprende al suo interno un fantastico ed avanzatissimo filtro di pacchetti attrezzato in modo molto completo.
- Ci sono una serie di tabelle (**tables**) che contengono ciascuna diverse catene (**chains**) da attraversare a seconda del percorso che sta facendo il pacchetto.
- La tabella principale, relativa alle regole su cosa può entrare verso la nostra macchina e uscirne è la tabella ***filter***.
- Nelle prossime regole, se non lo vediamo scritto, ci stiamo implicitamente riferendo a quella.

Netfilter: Percorso di un pacchetto nel sistema



Netfilter: Percorso di un pacchetto nel sistema



Struttura di netfilter

- Ogni **chain** (o catena) contiene una lista di regole che devono essere attraversate e che possono eseguire diverse azioni se il pacchetto le soddisfa.
- I nomi delle chain sono scritti in maiuscolo (per distinguerli dalle tables).
- Se nessuna regola corrisponde al pacchetto transistato, si applica come operazione la **policy di default**.

- Le tabelle importanti sono 3:
  - **filter** : Si occupa delle regole relative al traffico in ingresso/uscita/inoltro dalla nostra macchina sono le regole che solitamente non modificano i pacchetti.
  - **mangle**: È dove mettiamo solitamente le regole che modificano i pacchetti in qualche loro parte
  - **nat** : È la tabella dove mettiamo tutte le regole relative al funzionamento della NAT

Sintassi delle regole di iptables

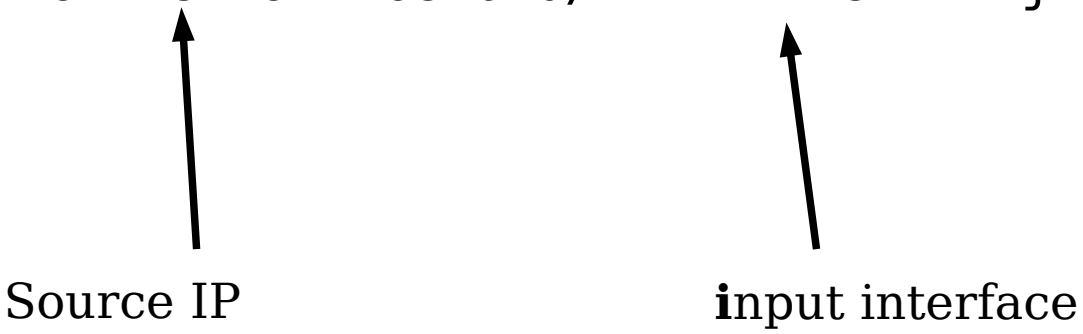
- Una regola è solitamente strutturata in due parti:
  - Le informazioni per “**riconoscere**” il tipo di pacchetto che ci interessa (**match**)
  - L'operazione da effettuare come conseguenza al match: il **target**
  - Anche qua vedremo che per distinguere tra match e target si usano nomi minuscoli o maiuscoli
- Partiamo subito con un esempio prima di tutto:
  - **`iptables -A INPUT --source 192.168.0.153 -j DROP`**
    - **-A** significa che vogliamo aggiungere questa regola (in coda alle altre) nella chain INPUT
    - **--source** indica che la condizione che il pacchetto Ipv4 deve avere come sorgente l'indirizzo 192.168.0.153
    - **-j DROP** specifica che se le precedenti condizioni di match sono soddifatte, l'operazione da effettuare è un **jump** al target DROP, cioè scartiamo il pacchetto.
    - Il pacchetto viene cestinato e non proseguirà nelle altre regole.

Sintassi delle regole di iptables

- Nel match possiamo praticamente sempre utilizzare la negazione mettendo un punto esclamativo “!” davanti ad una delle condizioni del match

- Esempio:

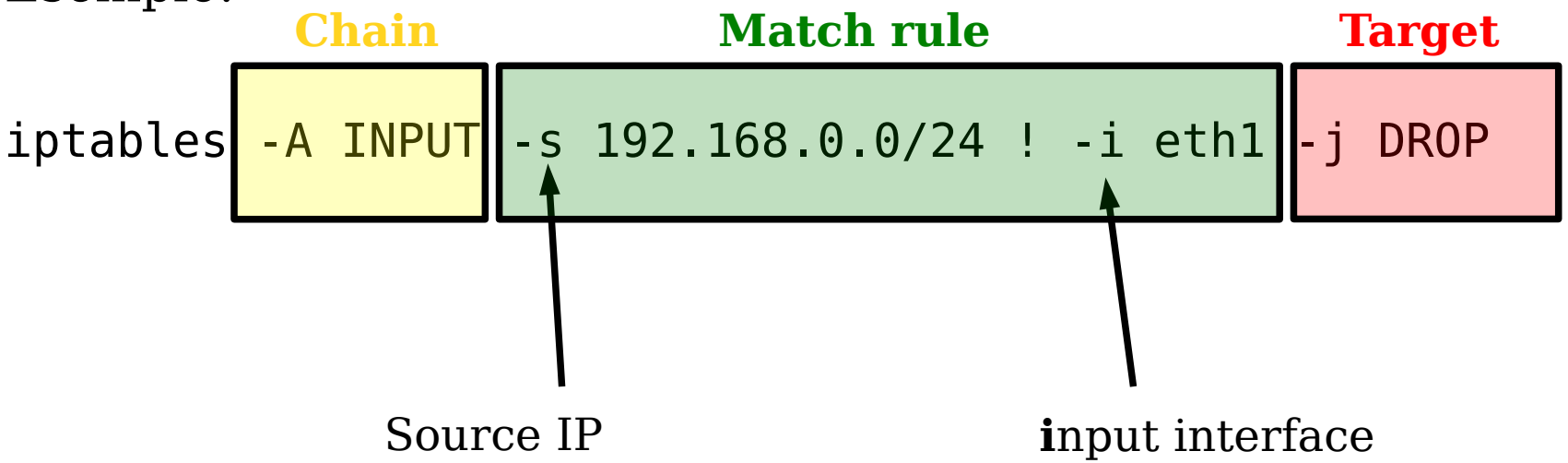
```
iptables -A INPUT -s 192.168.0.0/24 ! -i eth1 -j DROP
```



Sintassi delle regole di iptables

- Nel match possiamo praticamente sempre utilizzare la negazione mettendo un punto esclamativo “!” davanti ad una delle condizioni del match

- Esempio:



Sintassi delle regole di iptables

```
iptables -A [CHAIN] [Condizione sul pacchetto] -j [Operazione]

iptables -D [CHAIN] [Condizione sul pacchetto] -j [Operazione]

iptables -I [CHAIN] [Condizione sul pacchetto] -j [Operazione]

iptables -F [CHAIN]
```

(Flush delle regole)

```
iptables -t [Tabella] -A [CHAIN] .........
```

Cancellare una regola

- Possiamo riscriverla uguale mettendo -D al posto di -A

- Oppure possiamo trovare il numero della regola e fare:

  `iptables -t [`*`table`*`] -D [`*`CHAIN`*`] [`*`Numero regola`*`]`

- Per cancellare tutto possiamo usare -F (agisce solo all'interno di una specifica tabella, se non specifichiamo niente, con iptables -F viene cancellata tutta la tabella *filter*)

Sintassi delle regole di iptables

- Esistono altre cose oltre a -j (jump) (tipo -g) ma non ci addentreremo per ora (Non sono importanti). Se vi interessa roba avanzata RTFM.

- Abbiamo visto il target DROP. Il target duale è ACCEPT.

- Con **`-j ACCEPT`** diciamo:
  - Accetta il pacchetto in questa chain e passa alla chain successiva (se c'è).
    - Nel caso di INPUT non c'è una chain successiva (perché dopo viene passato al sistema operativo, quindi mettere ACCEPT in INPUT significa far entrare il pacchetto)
    - Se invece per esempio accettiamo in PREROUTING, non è detto che il pacchetto arrivi. Potrebbe venire bloccato in INPUT!

Vedere la lista di regole

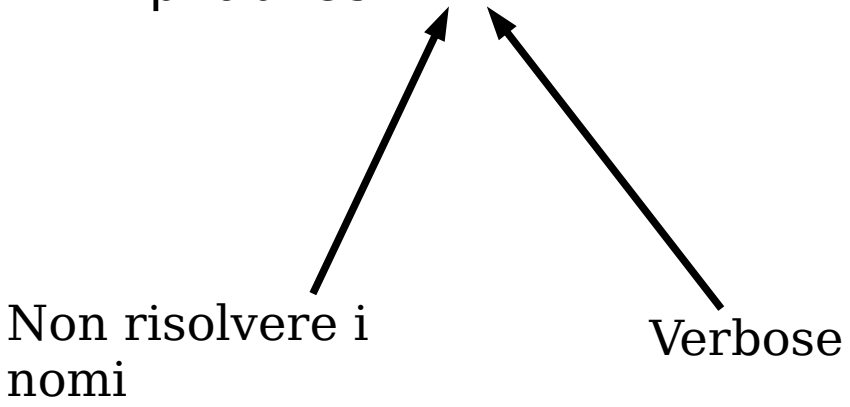
- Con il comando

  `iptables -L`

  Si ottiene la lista delle regole per la tabella filter (in caso di altre tabelle basta aggiungere -t come ad esempio `iptables -t nat -L`)

- Consiglio di usare

  `iptables -nvL`



  `iptables -nvL --line-numbers`

Match comuni

- Interfaccia di rete di **i**nput/**o**utput

  -i [*iface*]

  -o [*iface*]

  - Questi e altri match ovviamente non possiamo usarli sempre. Ad esempio non ha senso usare -i nella chain di OUTPUT.
  - Possiamo usare -i in INPUT, FORWARD, PREROUTING
  - Possiamo usare -o in OUTPUT, FORWARD, POSTROUTING

Match comuni

- Indirizzo sorgente/destinazione

  -s [*address or network address*]

  -d [*address or network address*]

  - L'indirizzo è il solito indirizzo Ipv4 (vedremo poi cosa dobbiamo usare per lavorare con ipv6) in formato decimale puntato. L'eventuale maschera di rete viene indicata in notazione CIDR
  - Esempio:

    iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT

    - Accettiamo tutti i pacchetti provenienti da qualunque host con indirizzo 192.168.1.*
    - (Se non mettiamo la maschera è di default 32, cioè esattamente **quell**'indirizzo)

Match comuni

- Possiamo fare dei match sul protocollo (di livello 4):

  `iptables -A INPUT -p udp -j ACCEPT`

  - I principali protocolli utili sono: tcp,udp,icmp, all (uguale a non specificare niente)

- **Per TCP e UDP possiamo specificare i numeri di porta**

  `iptables -A INPUT -p tcp --dport 80 -j ACCEPT`

  ↑

  Esempio classico dell'apertura della porta per un server http

- Possiamo fare il match anche sulla porta sorgente (molto raramente utile):

  `iptables -A INPUT -p tcp --sport 5000 -j ACCEPT`

Match comuni

- Per comodità è possibile scegliere anche un range di porte nel match:

  iptables -A INPUT -p tcp --dport 80:100 -j ACCEPT

- Oppure una lista di porte

  iptables -A INPUT -p tcp --dport 22,25,80 -j ACCEPT

Match malati

```
iptables -A INPUT -p tcp -m string --string “gattuso”
--algo bm -j DROP
```

È solo un esempio di cosa malata! Fate attenzione ad usare queste porcherie!

Match stateful

- **Nota**: I match che abbiamo visto fino ad ora sono praticamente tutti “integrati” all'interno di netfilter. Poi molti altri match e tutti quelli aggiuntivi necessitiamo quasi sempre di

  ```
  -m nome_modulo_match
  ```

  Per dire a iptables, e quindi a netfilter di caricare il modulo relativo a quel match

- Possiamo fare un match sullo stato della connessione associata ad un pacchetto. Questo è il tipico esempio di come sfruttuare il fatto che netfilter è uno **stateful packet filter**

  `iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT`

  - Questa regola indica che se il pacchetto che sto analizzando è parte di una connessione già stabilita (c'è già stato il three-way handshake nel caso di TCP o sono già stati inviati dati nel caso di UDP) allora accetto il pacchetto in ingresso

Targets comuni

- Ce ne sono decine (man iptables). Vi mostro quelli più utili e comuni

- **ACCEPT/DROP** : Li abbiamo già visti
- **REJECT**: Come DROP ma viene avvisato il mittente tramite qualche messaggio
  - Ad esempio **`-j REJECT --reject-with tcp-reset`**
- **`LOG:`** Salva una riga di log relativa al pacchetto che ha fatto il match la regola nei log di sistema (dmesg)

Targets

- Esiste anche la possibilità di definire una nostra chain di regole

  ```
  iptables -N MIACHAIN
  ```

- E saltarci dentro da altre chain come target

  ```
  iptables -A INPUT -p tcp -j MIACHAIN
  ```

- Non ci interessa per ora... Vi dico solo che esiste anche questa possibilità!

RST o DROP?

- *“We do not send resets. When a strange packet appears we just **drop it on the floor**”*

RST o DROP?

RST = “Che diavolo è questa roba? Non so cosa sia e non voglio averci niente a che fare, **annulla tutto**!”

- Normalmente, se non utilizzano un firewall, quando qualcuno tenta di stabilire una connessione TCP su una nostra porta chiusa, il nostro host **risponde con un RST** per segnalare che il servizio non c'è.
- Lo standard dice: si invia un RST per ogni connessione TCP malformata o non valida etc etc..
- La risposta di RST permette di capire subito quali porte sono aperte (quelle che non mandano RST).
- La risposta di RST permette di capire se un host è online (a volte si vuole nascondere questo fatto)
- In generale possiamo evitare di rispondere, facendo finta che l'host è spento, quindi perché perdere tempo a rispondere?

Blacklisting o whitelisting?

- “Vogliamo fare una festa. Facciamo la lista di tutti gli invitati o la lista di tutti i '*non-invitati*' ?” (cit.)
- Si fa praticamente sempre whitelisting. Se state facendo blacklisting state sbagliando qualcosa.
- All'interno delle varie **chain** è presente una *policy di default,* l'operazione da intraprendere se il pacchetto non corrisponde a nessuna regola.
- Le possibili policy di default sono due:
  - **DROP**
  - **ACCEPT**
- Quando viene inizializzato netfilter tutte le chains hanno policy ACCEPT. Solitamente ha senso impostare in DROP le chain di:
  - INPUT
  - FORWARD

Blacklisting o whitelisting?

- Solitamente la chain di INPUT ad esempio si imposta con policy di default **DROP**, poi si specificano tutte le eccezioni da accettare.
- È possibile mettere in DROP anche **OUTPUT** e quindi specificare poi una serie di regole riguardo al traffico che può uscire.
  - Questa scelta, specialmente per traffico in uscita dalla macchina locale, non viene però praticamente **mai fatta** (a parte per i super-paranoici o casi particolari), dato che dovreste pensare a tutti i possibili protocolli e porte che potrebbero uscire e aggiungere una regola per ciascuno.
  - Quindi solitamente la filosofia è **facciamo whitelisting sull'input e permettiamo tutto in output**.

Blacklisting o whitelisting?

- Alcune persone sfruttano l'ordine delle regole per fare delle policy di default

```
iptables -P INPUT ACCEPT
iptables -A INPUT -i eth0 -j ACCEPT
iptables -A INPUT -j DROP
```

- Per aggiungere **altre** regole oltre a queste, dopo bisogna usare -I di modo da metterle in testa, altrimenti il DROP che abbiamo messo non ci farà mai arrivare nella nuova regola aggiunta con -A
- Questo è certamente ammesso, ma se potete evitare di fare queste **porcherie** e usare invece le policy di default che sono lì apposta, fatelo!

- Questo **workaround** è più che altro utile nelle **chain definite dall'utente** dove non è possibile scegliere la policy di default

Interfaccia di loopback

- In linux esiste una interfaccia di rete “lo” che sta per loopback, che rappresenta l'interfaccia associata all'indirizzo 127.0.0.1 e tutta la rete 127.0.0.0/8
- Molti programmi di sistema utilizzano questa interfaccia per comunicare tra di loro.
- Iptables gestisce il traffico anche di questa interfaccia
- **Dobbiamo assicurarci di non bloccare NULLA in ingresso/uscita da questa interfaccia.**
- Quindi dovremo praticamente sempre avere

```
iptables -A INPUT -i lo -j ACCEPT
```

- Eventuali eccezioni sono come al solito per i super-paranoici o applicazioni super-hipster

ICMP

- **I**nternet **C**ontrol **M**essage **P**rotocol è un protocollo di servizio che si occupa di trasmettere informazioni riguardanti malfunzionamenti, informazioni di controllo o messaggi tra i vari componenti di una rete.
- Il tipo di pacchetto ICMP più famoso è senza dubbio la “echo request”, che viene inviata con il comando **ping** e serve a sapere se un certo host è attivo (e viene utilizzato per controllare se arrivano i pacchetti o ci sono problemi sulla rete)
- A meno che non abbiate specifiche esigenze (ad esempio nascondere il fatto che il vostro host è acceso) è considerata pratica **scorretta** oltre che “**maleducazione**” ignorare i messaggi ICMP
- Se volete essere gentili semplicemente accettate tutto:

```
iptables -A INPUT -p icmp -j ACCEPT
```

ICMP

- Se volete evitare di accettare **proprio tutto**, almeno accettate i tipi di messaggio icmp fondamentali: type 8,3,0

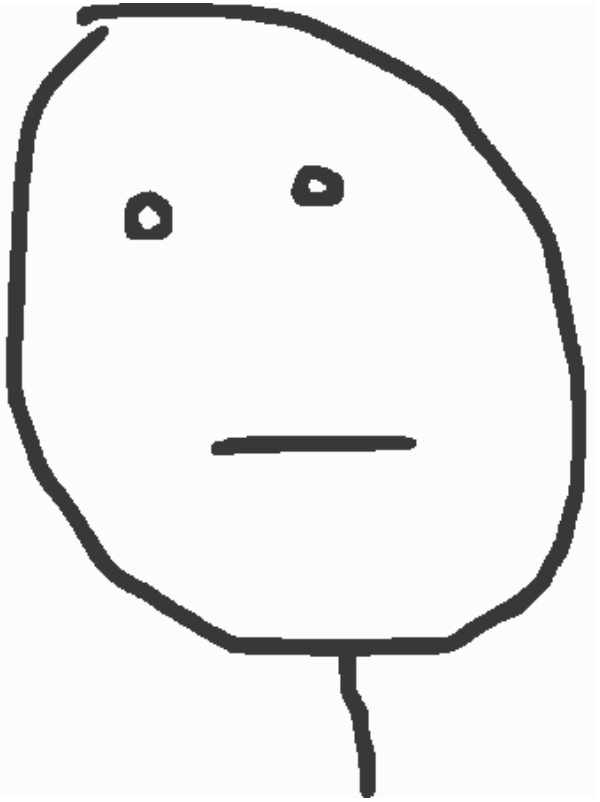
```
iptables -A INPUT -p icmp --icmp-type 8
iptables -A INPUT -p icmp --icmp-type 3
iptables -A INPUT -p icmp --icmp-type 0
```

  - Questi messaggi sono rispettivamente:
    - Echo request (type 8)
    - Destination Unreachable (type 3)
    - Echo reply (type 0)

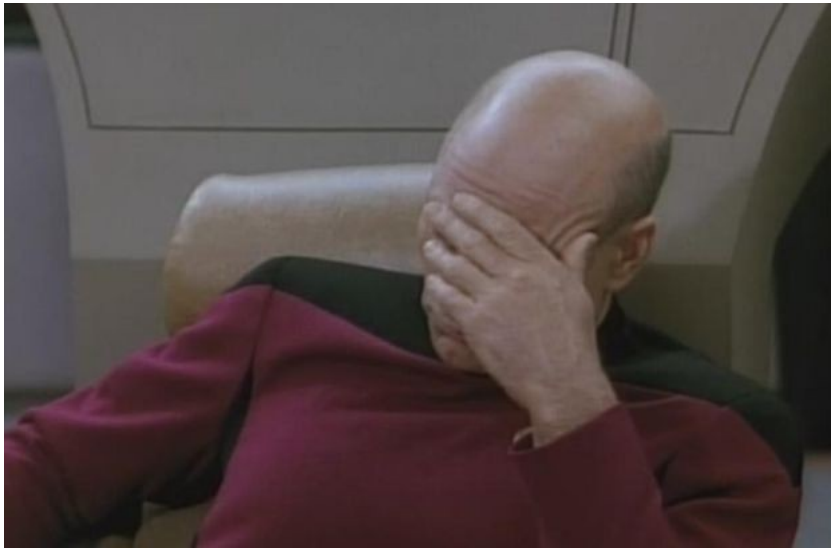
Chiudersi fuori di casa

Scena tipica

- Fate **molta attenzione** se state configurando una macchina remota (tipo via ssh) quando impostate le policy di default!
- Caso tipico:
  - “Oh cavolo, non ho impostato nessun firewall su questa macchina… beh… per prima cosa policy di default **DROP**!”
  - “Perché non risponde più il server? WTF?”
  - …
  - Tramite cosa stavate comunicando? (-.-')



Chiudersi fuori di casa



- Ricordatevi sempre di impostare le policy di default come **ultimo** passo, dopo esservi assicurati di aver messo una regola per accettare le connessioni nuove e stabilite per la vostra shell (ssh solitamente)

Chiudersi fuori di casa

- Ricordiamoci anche di fare attenzione ad usare il comando di **flush** (**-F**) perché non reimposta le policy di default.

```
iptables -A INPUT -m state –state RELATED,ESTABLISHED -j
ACCEPT
iptables -A INPUT -p tcp –dport 22 -j ACCEPT
iptables -P INPUT DROP
```

Più tardi...

iptables -F INPUT

…

Connection lost.

Sicurezza: Scrivere regole specifiche

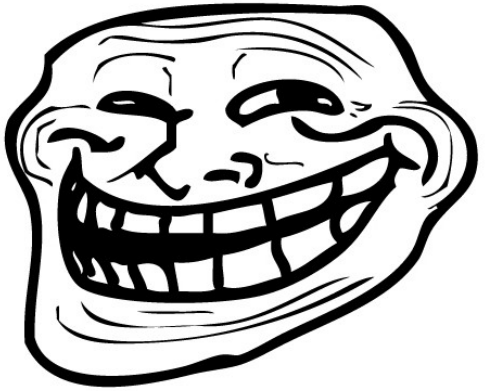
- Giusto come reminder di sicurezza, tentate di essere specifici quanto serve nella dichiarazione delle regole.
- Ammettiamo di avere il nostro server collegato a:
  - eth0: Collegamento a internet
  - eth1: Lan interna con indirizzi 192.168.2.0/24
- Vogliamo permettere alla LAN interna di inviare pacchetti TCP su qualunque porta del server:

```
iptables -A INPUT -s 192.168.2.0/24 -j ACCEPT
```

- Diciamo poi che invece da internet si può accedere solo ad alcune porte e mettiamo la policy di default DROP.

Scrivere regole specifiche

- Dall'interfaccia eth0 arriva un pacchetto con sorgente 192.168.2.5



Problem?

- Cosa succede?
- **Il pacchetto viene accettato...**

- Nota a margine: In realtà se avete abilitato il meccanismo di reverse-path filtering nel kernel il pacchetto viene scartato, ma meglio non farci affidamento (tra l'altro spesso è “spento” questo meccanismo)

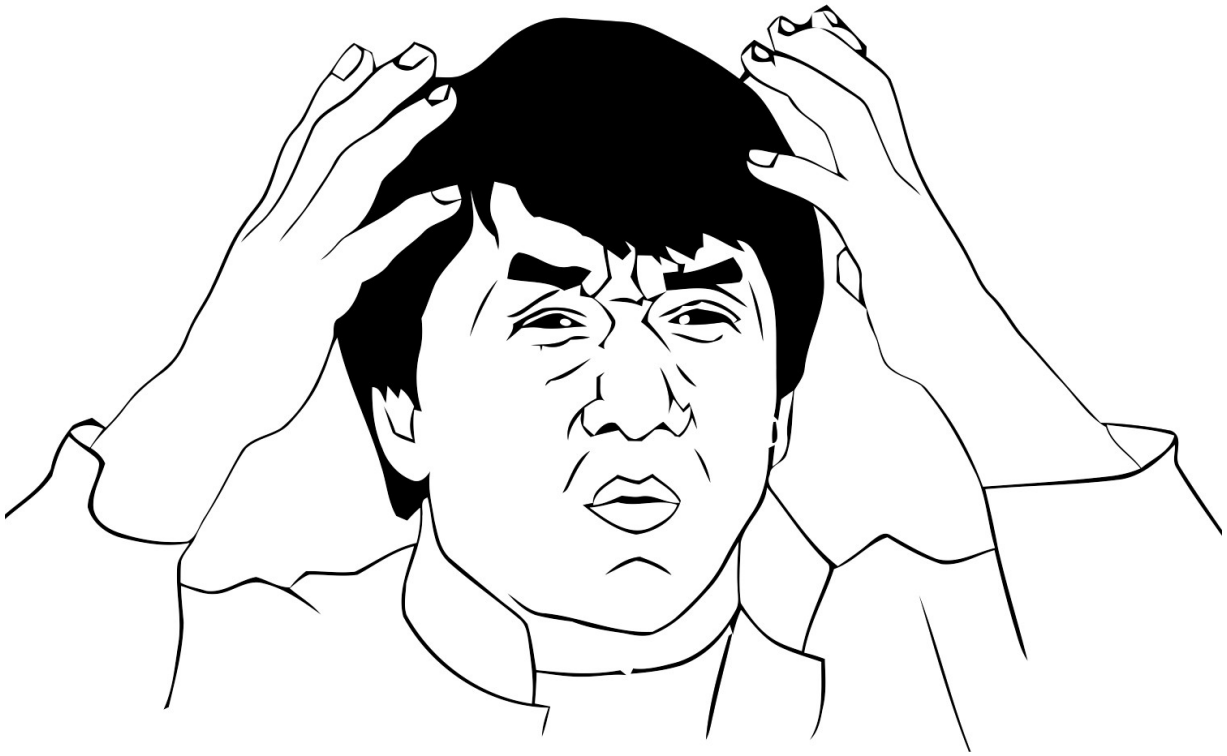
Scrivere regole specifiche

Ma chi è quel mona che imposta l'IP e chiude urlando senza specificare l'interfaccia?

...

Scrivere regole specifiche

- ...alcuni router “domestici” in vendita fanno proprio questo tipo di errore...



Scrivere regole specifiche

`iptables -A INPUT -s 127.0.0.0/8 -j ACCEPT`

È **decisamente diverso** da:

iptables -A INPUT -i lo -s 127.0.0.0/8 -j ACCEPT

Qualcuno sa dire perché?

Scrivere regole specifiche

Volendo fare i pignoli anche

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
```

È diverso da

```
iptables -A INPUT -p tcp --dport 22 -m state --state
NEW -j ACCEPT
```

Ma questo errore (forse) è meno grave del precedente

Esempio di config essenziale

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 8 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 3 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 0 -j ACCEPT
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -P INPUT DROP

iptables -P OUTPUT ACCEPT

iptables -P FORWARD DROP #Se non vogliamo fare da router a nessuno...
```

DoS: Denial of Service attacks

(Mahou Sensei Negima 154 pag 7)

(*) "iptables -A INPUT -p tcp --syn -m limit --limit 100/second -J ACCEPT" = it's the bucket filter for Linux

iptables -A INPUT -p tcp --syn -m limit --limit 100/second -j ACCEPT

Match “limit”

- Un metodo grezzo per evitare attacchi DoS di tipo SYN Flood è l'utilizzo del match limit. Vediamo prima come funziona.
- Esempio:

  iptables -A INPUT -p udp -j LOG #logga traffico UDP

  iptables -A INPUT -p udp -m limit --limit 1/second -j LOG
- Nel secondo caso viene effettuata l'azione al massimo 1 volta al secondo, non per ogni pacchetto che la soddisfa.
- La limitazione viene effettuata con il classico *token bucket filter*

- Gli attacchi SYN Flood si basano sul fatto che inviando troppe richieste TCP di connessione (pacchetti con flag SYN) troppo velocemente, si sovraccarica il sistema. Possiamo risolvere limitando il numero di richieste di connessione possibili per ogni secondo

```
iptables -I INPUT -p tcp --dport 80 -m state --state NEW -m limit --limit 100/second -j ACCEPT
```

Match “limit”

- Un altro esempio

  ```
  iptables -A INPUT -p icmp -m limit --limit 2/second -j ACCEPT
  ```

  (policy di INPUT: DROP)

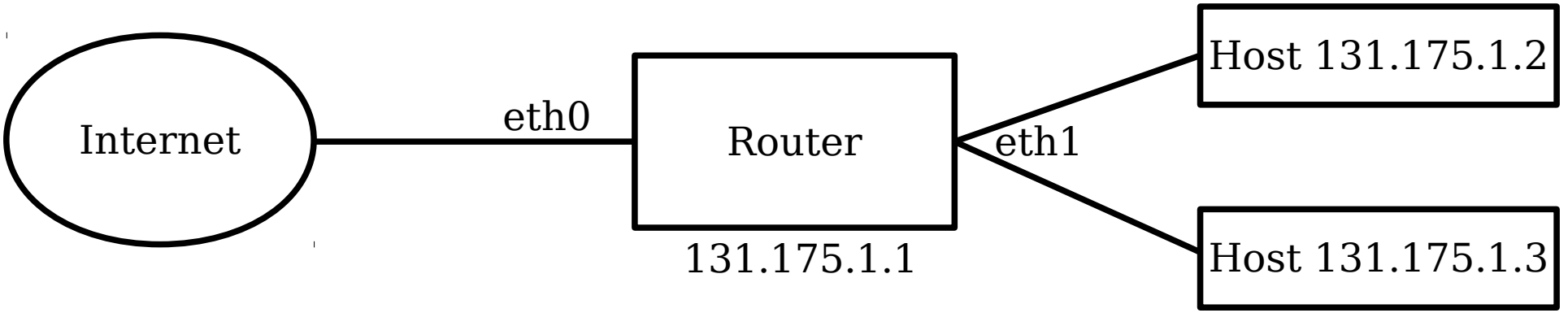
- Poi proviamo a fare

  ```
  ping -i 0.1 127.0.0.1
  ```

  - (Invia una richiesta icmp ogni 0.1 secondi)

  - Questo è solo un esempio, sarebbe meglio filtrare anche in base alla sorgente.
  - Il match hashlimit è generalmente migliore come performance

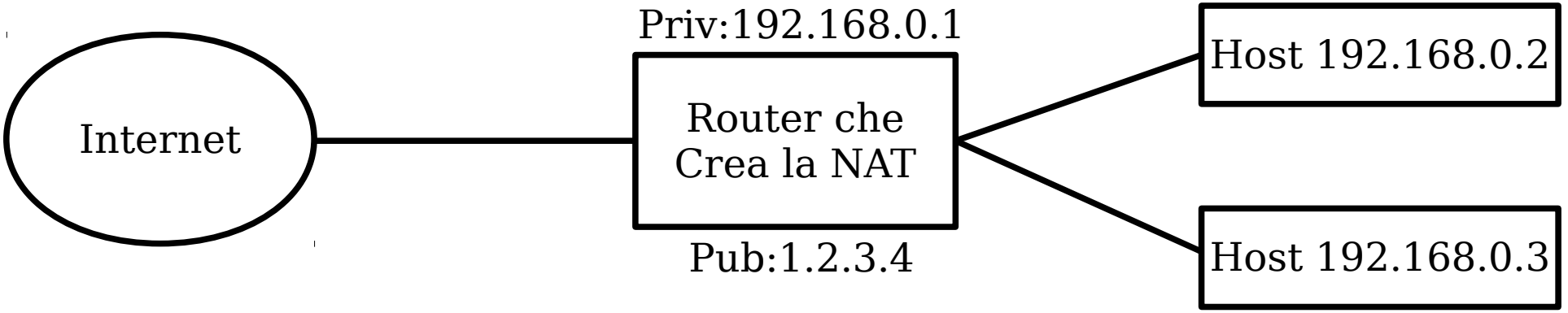
Fare un router/firewall



- `echo 1 > /proc/sys/net/ipv4/ip_forward`
- `iptables -P FORWARD DROP`
- `iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 80 -j ACCEPT`
- `Iptables -A FORWARD -i eth0 -o eth1 -m state --state RELATED,ESTABLISHED -j ACCEPT`

Network Address Translation

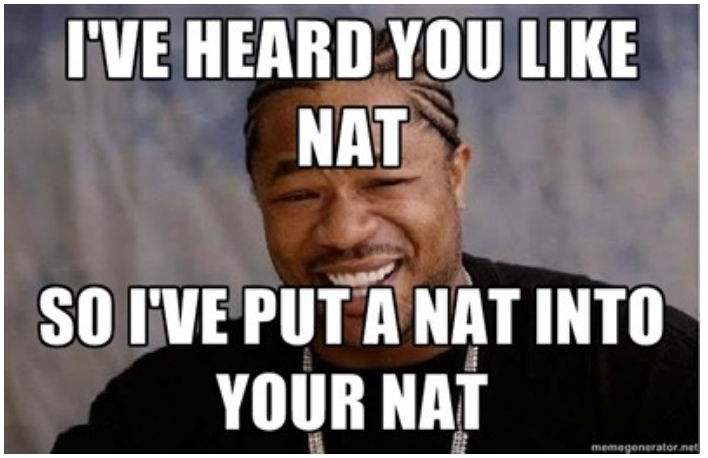
- Come funziona?



- Quando un host, ad esempio 192.168.0.2 vuole connettersi ad un server esterno, ad esempio 5.6.7.8, invia i pacchetti al router di frontiera (gateway) che prende l'**indirizzo sorgente** e lo sostituisce con il proprio (1.2.3.4). Il router sostituisce poi la mia porta TCP sorgente con una sua porta libera e si annota questo scambio in una tabella di connessioni stabilite
- Quando arrivano le risposte dal server (solo risposte relative ad una connessione **già stabilita**), il router capisce chi è il destinatario nella rete locale e sostiuisce l'indirizzo di destinazione (1.2.3.4) con l'indirizzo privato 192.168.0.2 e con la porta destinazione corretta.

NAT: un abominio

- La NAT sarà tanto bella e comoda ma **<u>è un workaround</u>** (temporaneo fino ad IPv6). Quando è possibile non andrebbe usata. Specialmente bisognerebbe evitare di fare **<u>doppie NAT</u>** o altre cose simili..



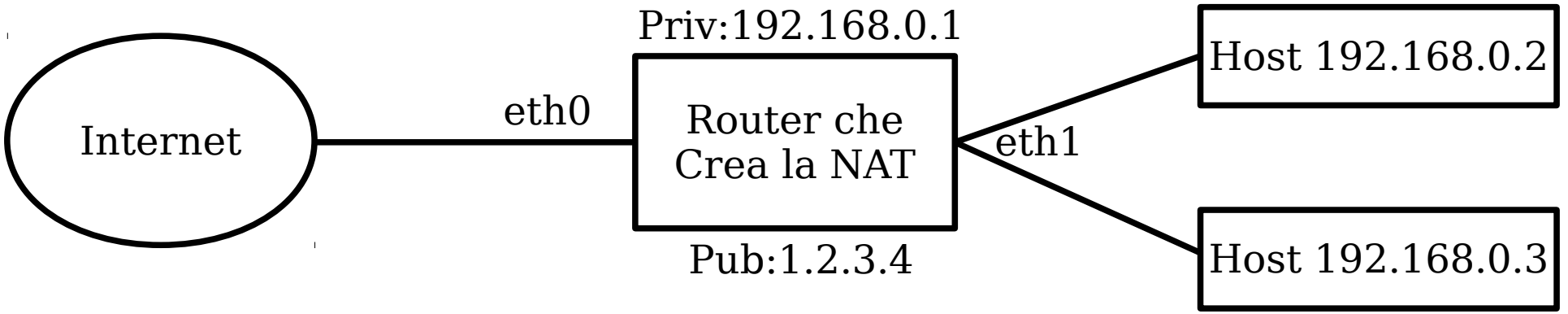
- **NAT viola il modello gerarchico di IP**
- **I processi su Internet non sono obbligati ad utilizzare TCP e UDP**
- **Il numero di connessioni contemporanee diminuisce**
- **NAPT trasforma Internet da una rete ad assenza di connessione**
- **La NAT da sola <u>non è un meccanismo firewalling.</u>**

Approfondimenti: http://otacon22.wordpress.com/2011/12/22/network-address-translation/

NAT: un abominio

- Se avete una valanga di IP pubblici o siete già dentro ad una NAT, evitate di farne “**yet another one**”, grazie.
- Purtroppo molte volte c'è poco da fare (abbiamo solo un indirizzo IP pubblico e tanti host da collegare) e siamo costretti ad usarla.
- Vediamo allora come fare...

Creare una NAT



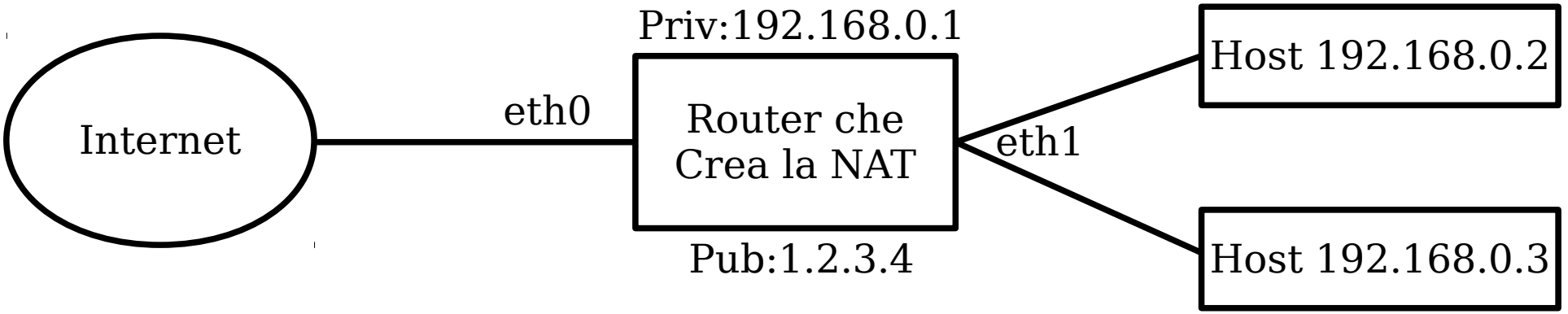

```
iptables -A FORWARD -s 192.168.0.0/24 -i eth1 -o eth0 -j ACCEPT

iptables -A FORWARD -i eth0 -o eth1 -m state --state
RELATED,ESTABLISHED -j ACCEPT

iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth0 -j SNAT --to-
source 1.2.3.4

echo “1” > /proc/sys/net/ipv4/ip_forward
```

Creare una NAT



Come facciamo se vogliamo mettere su un servizio (ad. es. un server web) sull'host 192.168.0.3?

```
iptables -A FORWARD -s 192.168.0.0/24 -i eth1 -o eth0 -j ACCEPT

iptables -A FORWARD -i eth0 -o eth1 -m state --state
RELATED,ESTABLISHED -j ACCEPT

iptables -A FORWARD -i eth0 -o eth1 -p tcp --dport 80 -j ACCEPT

iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth0 -j SNAT --to-
source 1.2.3.4

iptables -t nat -A PREROUTING -d 1.2.3.4 -i eth0 -j DNAT --to-
destination 192.168.0.3:80
echo “1” > /proc/sys/net/ipv4/ip_forward
```

IPv6

- Breve parentesi:
  - Linux supporta nativamente Ipv6 fin dai primi standard.
  - Le assegnazioni di blocchi IPv4 da parte della IANA sono state esaurite nel 2011.
  - In tutti i grandi datacenter internazionali IPv6 è già attivo e utilizzato.
- In netfilter ipv6 è già supportato. Per chiarezza il tool userspace per gestire le regole relative al traffico IPv6 è chiamato "**ip6tables**"
- Funziona esattamente allo stesso modo di iptables normale. La principale differenza è che non esiste la NAT
- Una nota: Alcune funzionalità di ICMPv6 sono obbligatorie (Neighbor Discovery Protocol, type 135 e 136) e se le bloccate avrete problemi di connessione.

```
ip6tables -A INPUT -p ipv6-icmp -j ACCEPT
```

Rendere le regole persistenti

- Problema: Abbiamo tutte le nostre belle regole di iptables, però come facciamo a salvarle tra un riavvio e l'altro (al riavvio il sistema operativo cancella tutte le regole e ripristina tutte le chain vuote)

- `iptables-save` e `iptables-restore` fanno al caso nostro.

- `iptables-save > /etc/firewall_rules`
- `iptables-restore < /etc/firewall_rules`

Rendere le regole persistenti

- C'è il vantaggio che sul restore se ci sono degli errori viene fatto Flush e ripristinata la policy ACCEPT = Evitiamo di chiuderci fuori
- Attenzione a non sbagliare le regole e chiuderci fuori **per sempre**..
- Ovviamente se abbiamo modificato i valori di alcuni file nella /proc non verranno salvate queste informazioni nel file e dovremo provvedere noi
- Su alcune distribuzioni esiste il servizio /etc/init.d/iptables

Come faccio io™.

- Metodo “mutanda di latta” che funziona più o meno ovunque:
  - Il file /etc/rc.local viene eseguito dopo aver lanciato tutti i demoni dei vari runlevel.
  - Aggiungiamo in questo file un paio di righe tipo:

    ```
    iptables-restore < /etc/firewall
    ```

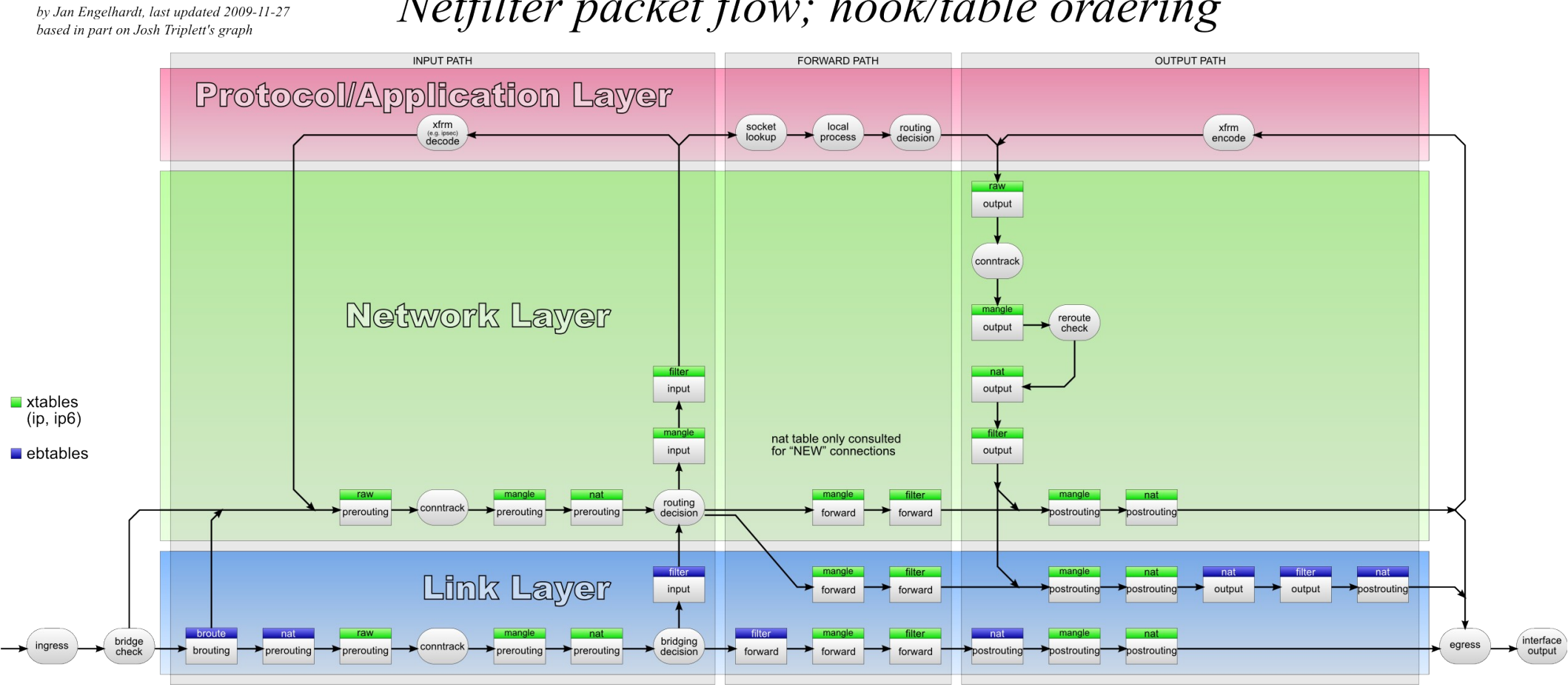
    ```
    echo “1” > /proc/sys/net/ipv4/ip_forward
    ```

  - … e ovviamente dobbiamo salvare la configurazione in /etc/firewall

Altri spunti

- Esistono i marcatori che vi permettono di “pitturare” i pacchetti che vi piacciono e farci altre cose più avanti nella traversata delle chain.
- C'è la possibilità di passare i pacchetti con -j QUEUE ad un programma userspace per analizzarli e fare una sorta di IDS
- ...
- ???
- PROFIT!

Abbiamo solo grattato la superficie di netfilter! ;)



http://inai.de/

Riferimenti

- Per **qualunque** domanda non esitate a scrivermi! otacon22@poul.org . Please non floddatemi o userò -m limit!

- `man iptables`

- Googlate: “Iptables filtering HOWTO”. Un po' vecchia ma ancora abbastanza valida come guida