Massimiliano Leone
kosmiko@logorroici.org

"Quelli che il Traffic Shaping... attività varie ed eventuali..."

hackit 2005
Napoli, 17-06-2005

Il firewall di Linux

Si usera il termine Linux piuttosto che GNU/Linux, non solo per motivi “pratici”, ma anche tecnici.

Il firewall disponibile nei sistemi GNU/Linux è parte integrante del kernel stesso, e quindi la dicitura “Linux” non è del tutto errata...

NETFILTER

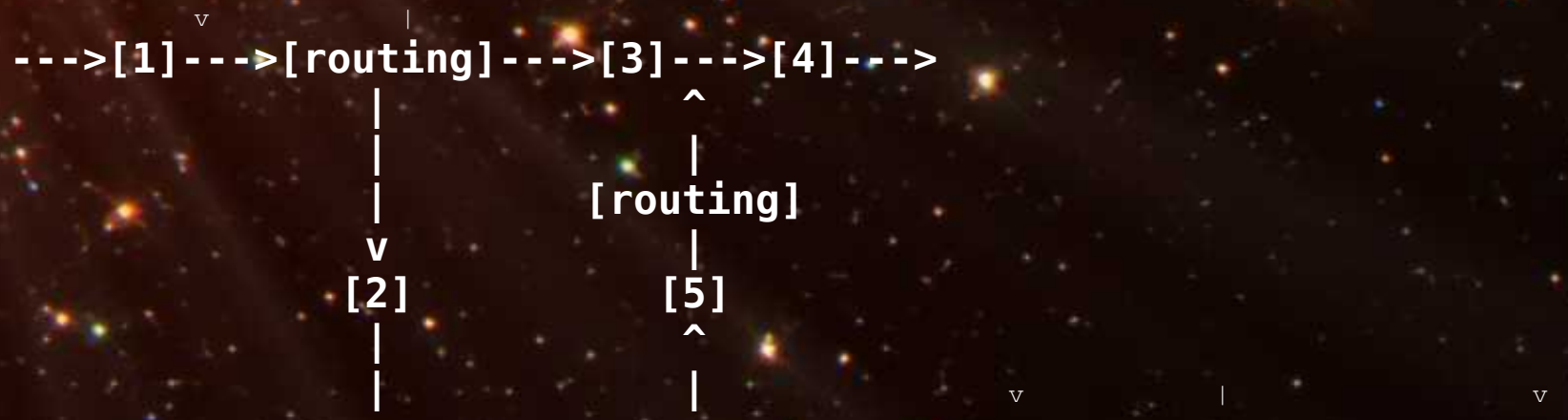
Netfilter è il nome di questo componente: come già detto, è parte del kernel, e quindi lavora in kernel-space (con i vari pro [in maggior numero...] e contro del caso...)

Può essere eseguito come modulo run-time, o esser compilato staticamente nel kernel.

Iptables è la controparte user-space di Netfilter, ed è ciò che serve per la configurazione delle varie ACL nel kernel.

Netfilter: implementazione (1)

Netfilter è strutturato in una in una serie di “punti di aggancio” (hook) collocati in vari livelli del protocol stack TCP/IP, come dalla illustrazione ascii:



(1) I pacchetti in entrata sono passati al 1° punto del framework netfilter, NF_IP_PRE_ROUTING [1]; entrano, in seguito, nella tabella di routing, dove si decide se i pacchetti sono destinati ad un'altra interfaccia di rete, ad un processo locale, o, eventualmente, ad essere scartati.

Netfilter: implementazione (2)

(2) Se destinato alla box stessa, i pacchetti sono processati da NF_IP_LOCAL_IN [2].

(3) Se invece destinato ad un'altra interfaccia, Netfilter invocherà NF_IP_FORWARD [3].

(4) Infine i pacchetti, prima di essere inviati di nuovo all'esterno, attraversano il punto finale, NF_IP_POST_ROUTING [4].

(5) NF_IP_LOCAL_OUT [5] è invece chiamato per i pacchetti creati localmente.

Netfilter: implementazione (3)

Se volutamente stabilito da qualche ACL, Netfilter “aggancerà” i pacchetti transitanti per quel particolare *hook* alla catena omonima (appunto, INPUT OUTPUT FORWARD PRE_ROUTING POST_ROUTING).

Più granularmente: Netfilter utilizza tre tabelle per gestire i flussi di pacchetti in entrata e in uscita dalla box. La 1^a, generica, è la FILTER; la 2^a è la tabella di NAT, usata per effettuare, appunto, il NAT, se la box stessa è un gateway per una lan, ed infine la tabella MANGLE, dove i pacchetti sono marcati e/o modificati, per successivi utilizzi.

Netfilter: implementazione (4)

Attenzione! Non tutte le catene sono usate in tutte le tabelle.

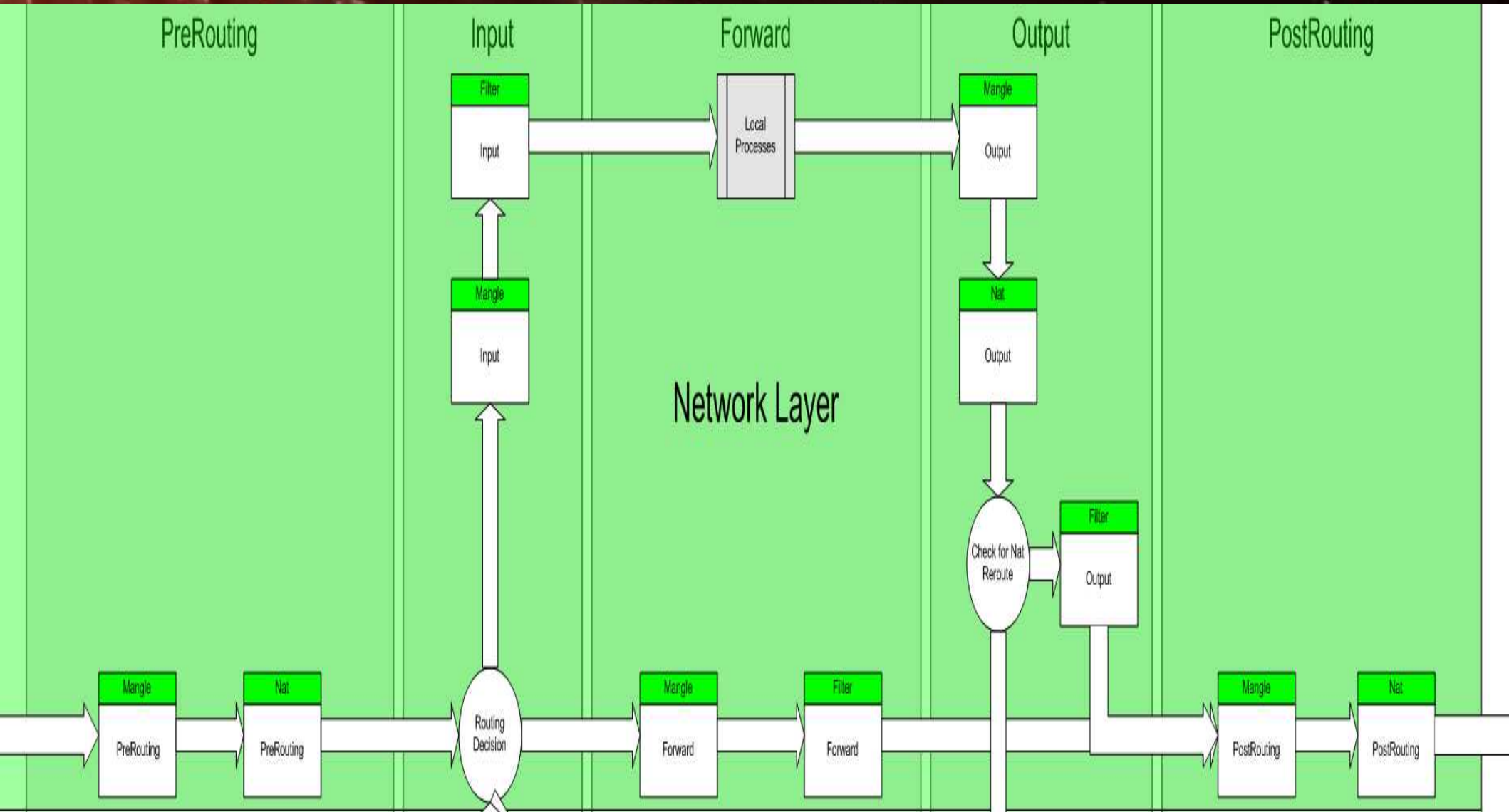
In particolar modo, quella di PRE_ROUTING e quella di POST_ROUTING, inutilizzate nella tabella di FILTER, sono invece di fondamentale importanza nella tabella di NAT, quando, rispettivamente, un pacchetto esterno sarà instradato verso la lan, dopo la modifica del campo destinazione, o quando, invece, un pacchetto interno alla lan sarà immesso verso l'interfaccia esterna, dopo la modifica del campo sorgente.

Netfilter: implementazione (5)

MANGLE può utilizzare invece tutte le catene: si potrà quindi marcare un pacchetto in uscita (o in entrata) dalla propria macchina, e proprio ad essa destinata, o diretto alla lan interna (senza o dopo aver attraversato il kernel del gateway), o infine da questa proveniente.

MANGLE, tra l'altro, è utilissima per il traffic-shaping (di cui più avanti): grazie ad essa si possono infatti identificare i pacchetti in transito nella box sfruttando tutti i filtri che Netfilter ci mette a disposizione. Pacchetti che saranno poi accodati verso l'interfaccia di rete, con una certa quantità di banda loro assegnata, proprio sulla base di questi identificativi.

Netfilter: graphic



Netfilter: Stati (1)

Netfilter, inoltre, è un firewall *statefull*: è, ovvero, in grado di tenere traccia di una connessione e del "passato" di ogni pacchetto. Il Connection Tracking fornisce 4 stati:

(1) **NEW**: Il primo pacchetto relativo ad una nuova connessione (syn TCP o nuovo pacchetto UDP).

(2) **ESTABLISHED**: Pacchetti relativi a connessioni già stabilite, nelle quali è transitato almeno un pacchetto da entrambi i peer.

Netfilter: Stati (2)

(3) **RELATED**: Pacchetti in qualche modo correlati a connessioni esistenti ed *established*.
Es.: traffico di dati FTP o trasferimento DCC su IRC.

(4) **INVALID**: Pacchetti che non rientrano in alcuno dei suddetti stati: di solito vengono ignorati.

Esempio reale di un conntrack: permettere, su un host, tutto il traffico in uscita e, in entrata, solo il traffico correlato a quanto uscito, ovvero una tipica desktop-box.

Netfilter: targets

Netfilter, ancora, dispone di un certo quantitativo di targets. Il “target” dice, in sostanza, “cosa fare di quel pacchetto” al kernel. Esempi:

- ACCEPT: accetta in entrata (o uscita)
- DROP: ignora
- REJECT: rifiuta inviando RST
- DNAT/SNAT: effettua DNAT/SNAT
- ULOG: registra su file/database
- MARK: “marca” con 1 identificativo numerico
- TTL: modifica ttl
- MASQUERADE: effettua masquerading di più host
- ...varie ed eventuali ;-)

Netfilter: match

Netfilter, infine, fornisce un certo elevato numero di match per filtrare il traffico quanto più granularmente si desideri. Ad esempio:

- *lenght*: lunghezza pacchetto
- *owner*: pid/user/group generante quel pacchetto
- *string*: particolare stringa (es. porn o warez)
- *time*: ora di arrivo
- *psd*: pacchetto facente parte di un portscan
- l*ayer7*: pacchetto di livello applicazione (secondo osi)
- *osf*: fingerprinting passivo del sistema operativo
- ...vari ed eventuali #2 ;-)

Netfilter/Iptables: examples

Effettuo il masquerading della subnet 192.168.0.0 verso il device ppp0:

- *iptables -t nat -o ppp0 -s 192.168.0.* -j MASQUERADE*

Contrassegno con l'intero 1 tutti i pacchetti (level 7) di kazaa:

- *iptables -t mangle -A OUTPUT -m layer7 –l7proto kazaa -j MARK –set-mark 1*

Ignoro pacchetti in uscita contenenti la stringa “porn”:

- *iptables -A OUTPUT -m string porn -j DROP*

Traffic Shaping

Obiettivo:

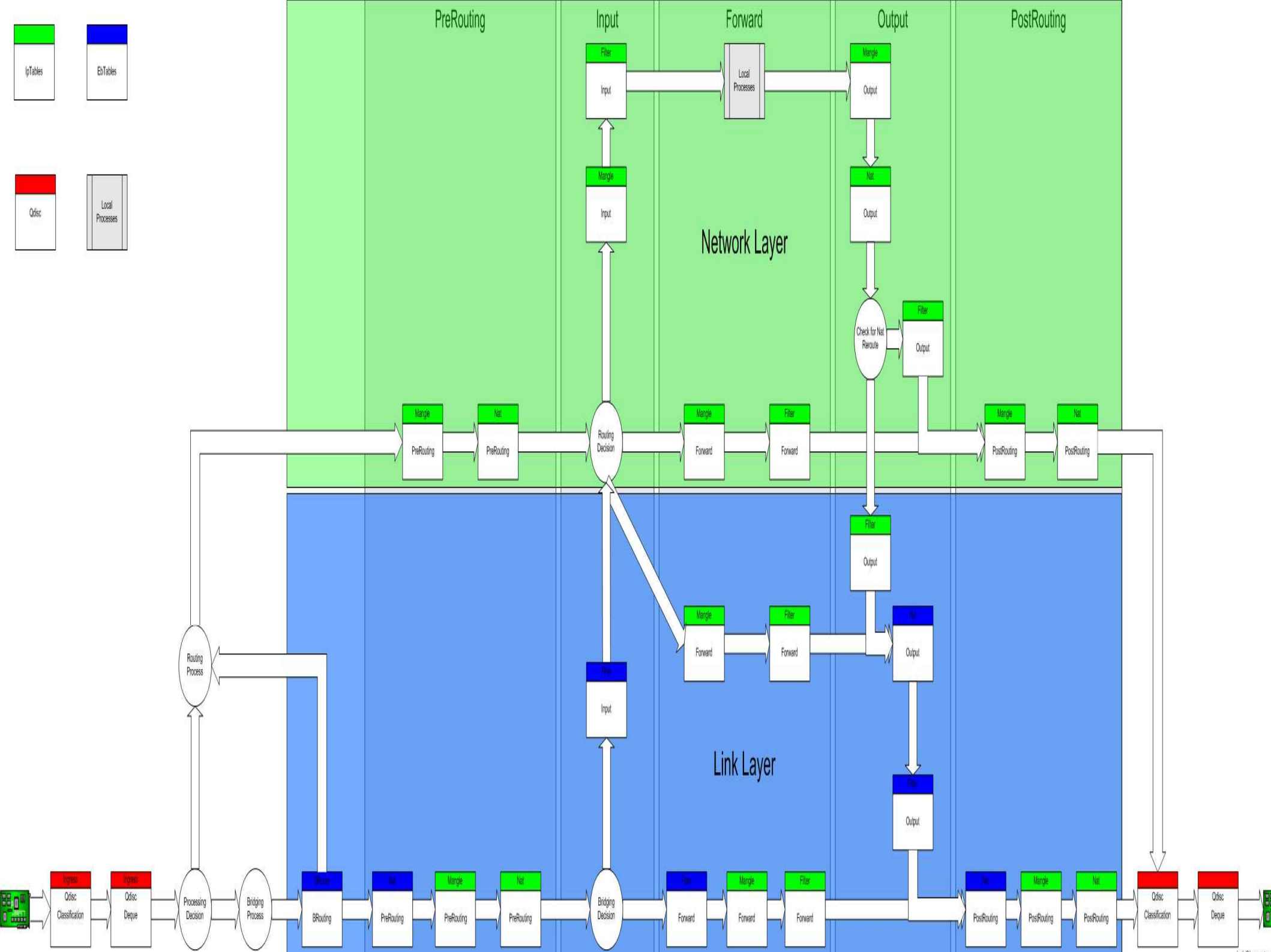
si vuole modellare il traffico in una certa rete, assegnando quindi non già una univoca ampiezza di banda a tutti gli host e/o protocolli, ma quantità differenti a seconda delle specifiche esigenze.

Traffic Shaping: theory (1)

Premessa non banale:

L'unico traffico modellabile è quello uscente da un'interfaccia, e non gia quello entrante.

L'immagine seguente è esplicativa


IpTables
EbTables
Qdisc
Local Processes
PreRouting
Input
Forward
Output
PostRouting
Network Layer
Link Layer
Filter
Input
Local Processes
Mangle
Output
Nat
Output
Mangle
Input
Check for Nat Reroute
Filter
Output
Mangle
PreRouting
Nat
PreRouting
Routing Decision
Mangle
Forward
Filter
Forward
Mangle
PostRouting
Nat
PostRouting
Filter
Output
Mangle
Forward
Filter
Forward
Output
Filter
Input
Routing Process
Output
Ingress
Qdisc Classification
Ingress
Qdisc Deque
Processing Decision
Bridging Process
BRouting
Nat
PreRouting
Mangle
PreRouting
Nat
PreRouting
Bridging Decision
Filter
Forward
Mangle
Forward
Filter
Forward
Nat
PostRouting
Mangle
PostRouting
Nat
PostRouting
Qdisc Classification
Qdisc Deque

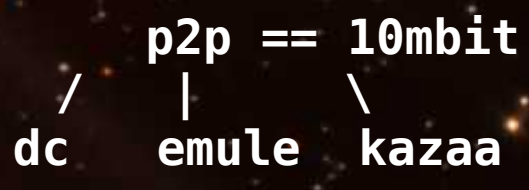
Traffic Shaping: theory (2)

Esiste in realtà una modalità per limitare il traffico in entrata su un device, ma permette un controllo assai limitato: si riduce, infatti, a gestire indistintamente la quantità di dati passante per quel network device, poiché il flusso di dati non “attraversa” ancora lo stack tcp/ip del kernel.

Il traffico in uscita, viceversa, è controllabile più granularmente dal sistema, data una discreta disponibilità di filtri (da netfilter, di cui sopra) e di algoritmi di accodamento dei pacchetti.

Traffic Shaping: theory (3)

Suddividere il traffico secondo certi criteri, significa, altresi, create un certo numero di code (queues), le quali possono avere, a loro volta, code figlie, come nell'esempio:



Traffic Shaping: theory (4)

Esistono anche una modalità di traffic-shaping dove non sono previste distinzioni di code, ovvero la modalità *classless* (contro la precedente *classfull*): non sarà peraltro trattata.

Altrettanto, non saranno trattati tutti gli algoritmi di scheduling dei pacchetti, ma solo i due più diffusi:
HTB e SFQ

Traffic Shaping: theory (5)

HTB è un algoritmo classfull che permette, cosa assai interessante, di effettuare il “prestito” tra le code.

Esempio: alloco 10 mbit per una sottorete di 5 host, e assegno ad ognuno 2 mbit. Peraltro, quando sono connessi solo 2 dei 5 host, è ragionevole voler usare anche gli altri 10-4 = 6 mbit.

Ed è proprio cio che permette di fare HTB

Traffic Shaping: theory (6)

Come tutte le strutture ad albero, questa suddivisione in classi ha, infine, delle classi figlie (dette foglie “leaf”). In queste “foglie” vengono alfine accodati i pacchetti dal kernel, che per default, utilizza una politica FIFO.

Tuttavia, se ne potrebbero preferire altre migliori: tra queste spicca per buon compromesso tra poca avidità di risorse e discreta efficienza proprio SFQ, che utilizza un algoritmo stocatisco con media pesata mobile.

Traffic Shaping: bottom-up example

```
tc qdisc add dev imq0 root handle 1 htb default 200
tc class add dev imq0 parent 1 classid 2 htb rate 10mbit

tc class add dev imq0 parent 2 classid 20 htb rate 5mbit

tc class add dev imq0 parent 20 classid 22 htb rate 2mbit ceil
    3mbit
iptables -t mangle -A POSTROUTING -m layer7 --l7proto ssh -j MARK
    --set-mark 22
tc filter add dev imq0 protocol ip parent 1:0 prio 1 handle 22
    fw classid 22
tc qdisc add dev imq0 parent 1:22 handle 22 sfq


tc class add dev imq0 parent 20 classid 23 htb rate 2mbit ceil
    3mbit
iptables -t mangle -A POSTROUTING -m layer7 --l7proto telnet -j
MARK --set-mark 23
tc filter add dev imq0 protocol ip parent 1:0 prio 1 handle 23
    fw classid 23
tc qdisc add dev imq0 parent 1:23 handle 23 sfq
```

Traffic Shaping: example (2)

tc è il tool userspace che permette di configurare le code nel kernel, cosi come la banda da assegnare loro.

- 1: creo la classe root, di tipo htb, con coda default 200;
- 2: classe di id 2 con banda 10 mbit
- 3: classe 20 con 5 mbit
- 4: classe 22 con banda 2 mbit, e che può chiedere 3 mbit in prestito (parametro ceil) alla coda padre (parent)
- 5: con iptables marco i pacchetti ssh con id 22
- 6: con tc aggiungo un filtro alla coda 22, che impone al kernel di inviare in essa i pacchetti marcati con id 22
- 7: aggiungo alla coda 22 (foglia) uno scheduler SFQ

IMQ ?

Cos'è il device imq?

E' un device di rete virtuale, che ovvia proprio al problema (di cui sopra) circa il fatto che si può controllare solo il traffico in uscita.

Ovvero: se non posso usare eth0 perchè è il mio device di entrata, semplicemente redirigo il traffico entrante da eth0 su imq0

```
iptables -t mangle -A PREROUTING -j IMQ --todev 0
```

Documentazione

www.netfilter.org

www.lartc.org

www.linuximq.net

www.docum.org

www.google.it

K-shaper (1)

I comandi appena mostrati, tuttavia, hanno una sintassi alquanto complessa, e questa complessità aumenta, anche di molto, in uno scenario reale, ad esempio con un albero di code con 3 livelli di genealogia e circa 15 protocolli da controllare.

K-shaper (2)

(0)L'idea è quindi di creare un tool che permetta di progettare un sistema di questo tipo, utilizzando un linguaggio semantico capace di:

(1) rappresentare l'albero delle code
(2) esprimere, sinteticamente ma altrettanto dettagliatamente, le caratteristiche di ogni ramo e di ogni foglia

...e XML è quanto di più adatto si voglia.

k-shaper (3)

Tramite XML possiamo quindi creare il modello del nostro sistema e, in seguito, estrapolare da esso i dati necessari per ottenere i comandi di iptables e tc da eseguire.

Di questo si occuperà uno script perl, che:

(1) validerà il file xml
(2) effettuerà il parsing dei dati
(3) “costruirà” le stringhe dei comandi
(4) eventualmente eseguirà i suddetti comandi

k-shaper (4)

La scelta del Perl si basa sul fatto che:

(1) la validazione è fatta sia sul modello dello Schema (di cui più avanti), sia con espressioni regolari, le quali, per antonomasia, si sposano assai bene con questo linguaggio
(2) l'esecuzione di comandi esterni, ovvero l'interazione con il sistema ospite, è anch'esso un punto di forza di Perl rispetto ad altri linguaggi

k-shaper (5)

Lo Schema, infine, fornisce, a sua volta, un modello per il file d'istanza XML, dichiarando quali proprietà (cardinalità, ordinalità, tipo, valori ammessi...) deve rispecchiare ogni caratteristica delle foglie.

E, infine, dei buon editor xml (leggasi “capaci di interpretare lo Schema) permettono un editing “intelligente”, fornendo automaticamente le scelte possibili per quel particolare ramo (o foglia).

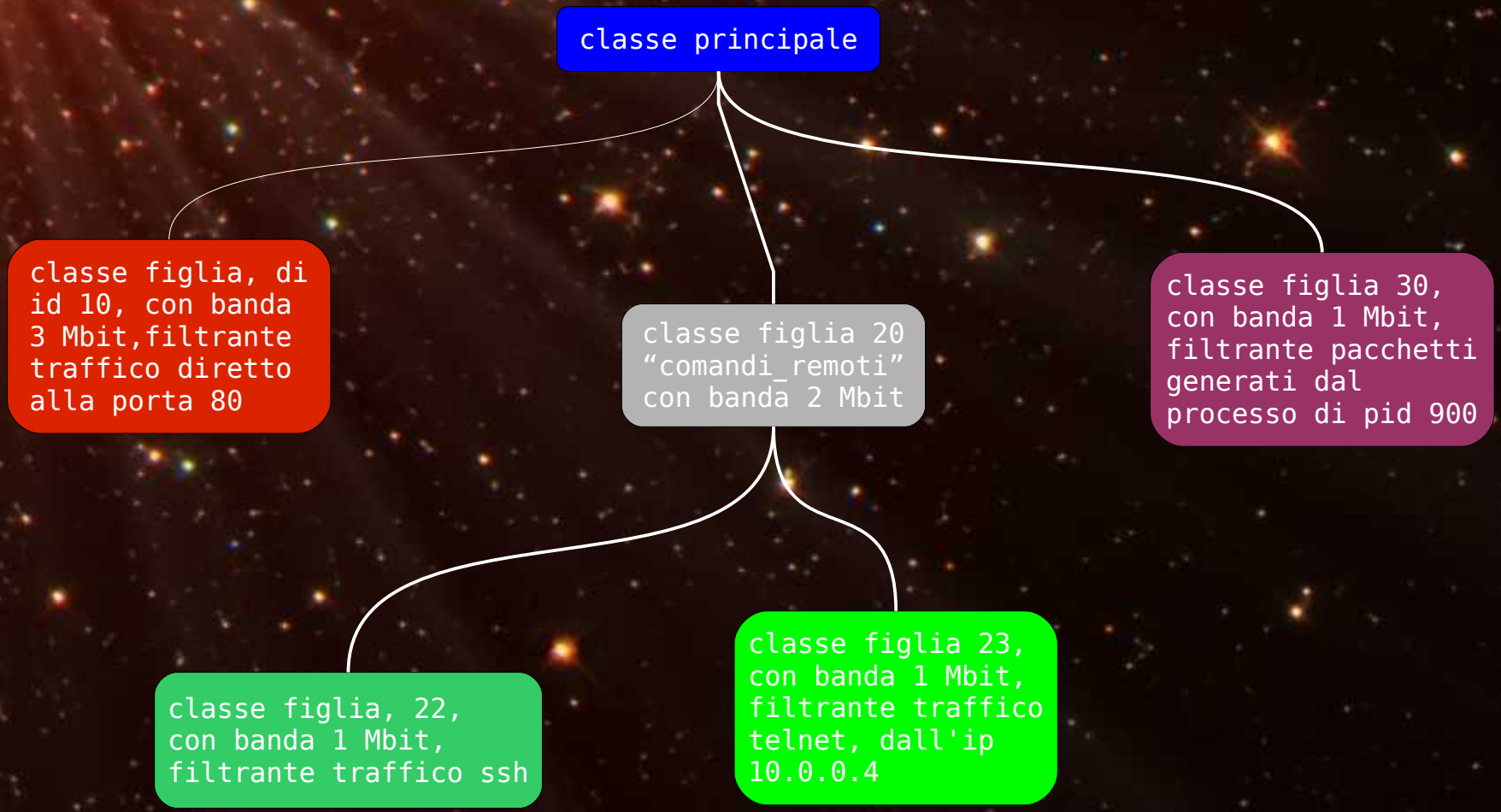
E ciò permette, alfine, di creare il nostro modello senza conoscere nulla (o quasi) di XML.

k-shaper (7)

k-shaper è il nome di questo tool:
seguirà un esempio di funzionamento.

La successiva slide mostra
uno scenario con 2 livelli di genealogia
e 5 protocolli (dal livello 3 al livello 7).

k-shaper (8)


classe principale
classe figlia, di id 10, con banda 3 Mbit,filtrante traffico diretto alla porta 80
classe figlia 20 “comandi_remoti” con banda 2 Mbit
classe figlia 30, con banda 1 Mbit, filtrante pacchetti generati dal processo di pid 900
classe figlia, 22, con banda 1 Mbit, filtrante traffico ssh
classe figlia 23, con banda 1 Mbit, filtrante traffico telnet, dall'ip 10.0.0.4

k-shaper (9)

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="config.xsd">
  <queue direction="both">
    <root default="100">
      <class>
        <id name="first">2</id>
        <rate unit="mbit">10</rate>
        <class>
              <id name="http">10</id>
              <rate unit="mbit">3</rate>
              <filter>
                    <port target="destination" protocol="tcp">80</port>
              </filter>
        </class>
        <class>
              <id name="remote_commands">20</id>
              <rate unit="mbit">2</rate>
              <class>
                    <id name="ssh">22</id>
                    <rate unit="mbit">1</rate>
                    <filter>
                          <application>ssh</application>
                    </filter>
              </class>
              <class>
                    <id name="telnet">23</id>
                    <rate unit="mbit">1</rate>
                    <filter>
                          <ip target="source">10.0.0.4</ip>
                          <application>telnet</application>
                    </filter>
              </class>
        </class>
        <class>
              <id name="local_p2p">30</id>
              <rate unit="mbit">1</rate>
              <filter>
                    <pid>900</pid>
              </filter>
        </class>
         <class>
              <id name="default">100</id>
              <rate unit="kbit">1</rate>
        </class>
      </class>
    </root>
  </queue>
</config>
```

K-shaper (10)

http://k-shaper.sf.net

per ora c'è un .tar e .deb,
di k-shaper e di una libreria xml
e la doc in sxi (openoffice < 1.1.3) e pdf

è rilasciato sotto licenza GPL

K-shaper (11)

A.A.A. cercasi collaboratori ;-)

License