PARTE E

DHCP e
Traffic Shaping

Modulo 1: Protocollo DHCP

Configurazione della rete

- **Ogni nodo in rete ha bisogno di alcuni parametri per il funzionamento**
  - Indirizzo IP
  - NetMask
  - Indirizzo del server DNS
  - Indirizzo del gateway
- **Un nodo deve essere configurato per poter connettersi alla rete**
- **La configurazione può essere**
  - Statica
  - Dinamica

Attribuzione statica dei parametri

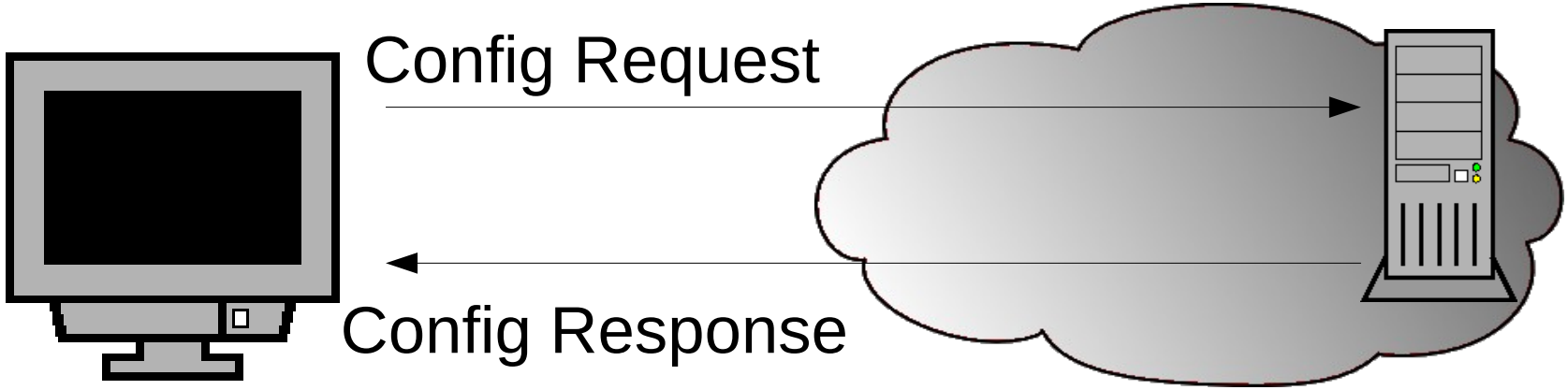
- **Gestire in modo statico le configurazioni su ogni nodo è problematico**
  - Manca un registro centrale degli indirizzi attribuiti
  - Si rischia di creare conflitti
  - Ogni modifica richiede di essere implementata su ogni nodo coinvolto
  - **La presenza di nodi mobili aggrava il problema**
  - Un portatile che si collega a reti diverse deve adattare le proprie configurazioni
  - Bisogna gestire diversi profili e scegliere il profilo giusto in ogni contesto

Attribuzione dinamica dei parametri

- **Il problema della configurazione della rete viene gestito in modo centralizzato**
- **La rete ha un repository centrale di informazioni**
  - Quali nodi sono connessi
  - Quali configurazioni dare ai nuovi nodi che si connettono
- **Questo consente anche di controllare gli accessi alla rete**
  - Se un nodo non è autorizzato ci si rifiuta di fornirgli dati sulla configurazione

Protocollo DHCP

- **Dynamic Host Configuration Protocol**
- **Protocollo per gestire la configurazione dei nodi alla loro connessione**
- **Idea di funzionamento:**
  - Il nodo (client) appena connesso non sa come deve configurarsi
  - Manda una richiesta generica “alla rete”
  - La rete invia i parametri di configurazione



Dettagli dell'interazione DHCP

- **Il client assume un indirizzo “provvisorio” per mandare le sue richieste** (From 0.0.0.0:68 To 255.255.255.255:67)
- **Client → server**
  - DHCP_DISCOVER
  - Richiede se ci sono indirizzi di rete disponibili
- **Server → Client**
  - DHCP_OFFER
  - Offre una possibile configurazione
- **Client → server**
  - DHCP_REQUEST
  - Richiede uno degli indirizzi offerti
- **Server → Client**
  - DHCP_ACK
  - Concede l'indirizzo richiesto

Validità della configurazione

- **Il server tiene traccia degli indirizzi che ha inviato (evita conflitti)**
- **Inoltre ogni indirizzo è dato in prestito solo temporaneamente**
  - Ogni indirizzo ha un lease time
- **Scaduto il lease time bisogna rinnovare la configurazione**

Installazione di un server DHCP

- **Sulla rete deve esserci un solo server DHCP**
  - Casi particolari possono vedere la presenza di più server con una rigida gerarchia di operazioni, ma a noi non interessa
- **La macchina server deve essere raggiungibile a livello H2N da ogni nodo della rete**
- **Installare il server DHCP su un nodo Debian/Ubuntu**
  - Nel nostro caso useremo DNSMasq
- **DNSMasq fa anche da server DNS**
  - Risoluzione nome → IP

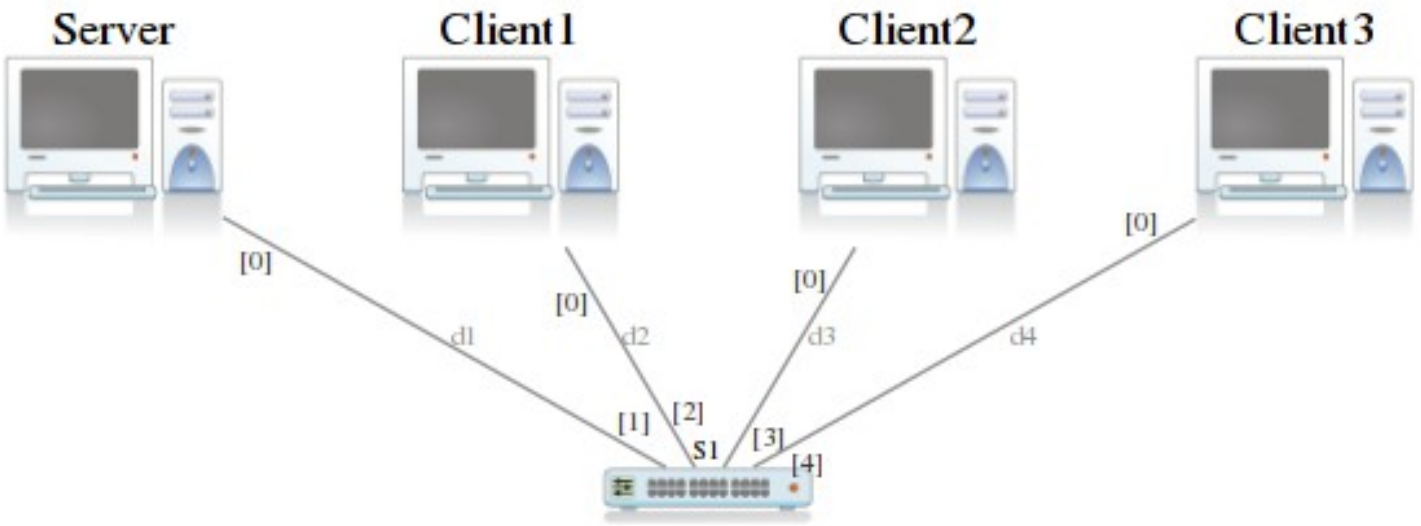
Configurazione

- **Elementi generali**
- **Pool di indirizzi da assegnare liberamente**
  - Ogni client che si connette riceve un indirizzo libero scelto da questo insieme
- **Indirizzi riservati da assegnare ad un nodo ben preciso**
  - Si usa l'indirizzo MAC per riconoscere un nodo

Configurazione

- **Elementi generali**
  - Server DNS
  - Gateway
  - Server WINS
  - Lease time
- **Pool di indirizzi da assegnare liberamente**
  - Ogni client che si connette riceve un indirizzo libero scelto da questo insieme
- **Indirizzi riservati da assegnare ad un nodo ben preciso**
  - Si usa l'indirizzo MAC per riconoscere un nodo

Scenario di riferimento



- **Server → DHCP server, gateway**
- **Client1, Client2 → client con indirizzi da pool**
- **Client3 → client con indirizzo fisso**

Client 3

- **Per creare un nodo con indirizzo fisso**
  - Non necessario in nodi fisici
  - Necessario con marionnet
- **Uso del file /etc/network/interfaces**
  - Configurazione dell’interfaccia di rete
  - hwaddress ether 02:04:06:11:22:33
- **Può servire anche in contesti reali**

Generico client

- File /etc/network/interfaces

```
auto eth0
iface eth0 inet dhcp
```

- Solo su Client 3 (Serve MAC address fisso)

```
auto eth0
iface eth0 inet dhcp
      hwaddress ether 02:04:06:11:22:33
```

Server

- File /etc/network/interfaces

```
auto eth0
iface eth0 inet static
        address 192.168.1.254
```

- Perché serve indirizzo statico sul server DHCP?

Server

- File /etc/dnsmaq.conf

```
# don't look for other nameservers
no-resolv
# read /etc/ethers file
read-ethers
# network interface for DHCP
interface=eth0
# network domain name
domain=reti.org
#...
```

Server

- File /etc/dnsmaq.conf

```
# ...
# Some DHCP options
# 3 = default GW
# 6 = DNS server
dhcp-option=3,192.168.1.254
dhcp-option=6,192.168.1.254
# or...
# dhcp-option=option:router,192.168.1.254
# dhcp-option=option:dns-server,192.168.1.254
```

Server

- File /etc/dnsmaq.conf

```
# ...
# dhcp range (min-max IP), include lease time
dhcp-range=192.168.1.10,192.168.1.15,1h
# static config, not /etc/ethers + /etc/hosts
dhcp-host=02:04:06:11:22:33,client3,192.168.1.3,1h
# override address (can also use /etc/hosts)
address=/www.hackerz.com/192.168.1.1
```

Server

- File /etc/hosts

```
192.168.1.254    server server.reti.org
192.168.1.3      client3 client3.reti.org
```

- File /etc/ethers

```
02:04:06:11:22:33   192.168.1.3
```

- Avviare il server all'accensione

```
systemctl enable dnsmaq
```

- Lanciare il server

```
service dnsmasq start
```

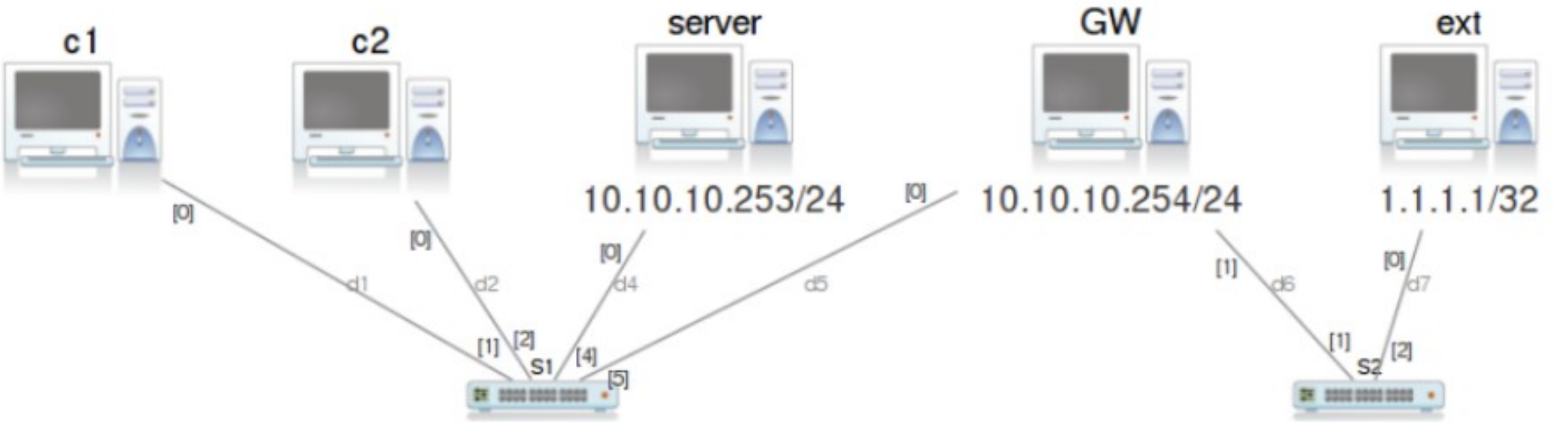
Verifiche DHCP e DNS

- **Su ogni client: configurazione DHCP**
  - Verificare indirizzo IP

    `ip addr show dev eth0`
  - Verificare regole routing

    `ip route show dev eth0`
- **Verificare DNS**
  - `cat /etc/resolv.conf`
  - `nslookup client3`
  - `nslookup client3.reti.org`
  - `nslookup www.hackerz.com`

Mettersi alla prova



Modulo 2:
Traffic Shaping

Traffic shaping

- **Il traffic shaping riguarda la gestione dei flussi di traffico.**
  - Si ha un link con capacità trasmissiva limitata
  - Si vuole ottimizzare la scelta dei pacchetti inviati.
- **Usi tipici:**
  - prioritizzazione del traffico
  - minimizzazione della latenza
  - gestione della banda
  - garanzia “fairness” tra diversi servizi

Traffic shaping su Linux

- **I datagrammi IP vengono accodati prima di essere spediti**
  - ogni interfaccia di rete dispone di (almeno) una coda dei datagrammi da inviare in rete
  - ogni coda è gestita da una queuing discipline (qdisc)
- **le qdisc possono essere classless, o classful:**
  - classles: possono riordinare, ritardare o eliminare dei pacchetti in coda
  - classful: sono dei “contenitori” per ulteriori qdisc (classless o classful). Consentono di organizzare gerarchicamente le qdisc
- **I filtri si applicano a una qdisc classful e servono per selezionare a quale qdisc “figlia” inoltrare il traffico di una qdisc classful “padre”**
- **La configurazione delle qdisc viene effettuata tramite la suite iproute 2, in particolare con il comando tc. In queste slide parleremo sempre di traffico in uscita da una interfaccia.**

classless qdisc

Gestiscono il traffico utilizzando solo *riordinamento*, *ritardo* ed *eliminazione* dei pacchetti, esistono diverse tipologie di code a seconda dello scenario delle policy da definire e dello scenario:

- **`pfifo_fast` : default**
- **`tbf` : Token Bucket Filter**
- **`sfq` : Stochasting Fairness Queuing**
- **`red` : Random Early Detection**
- **`codel, fq_codel` : Controlled Delay**
- **…**

classful qdisc

Classful qdisc disponibili su tc:

- `prio` : Priority, tc-prio(8)
- `cbq` : Class Based Queueing, tc-cbq(8)
- `htb` : Hierarchical Token Bucket, tc-htb(8)

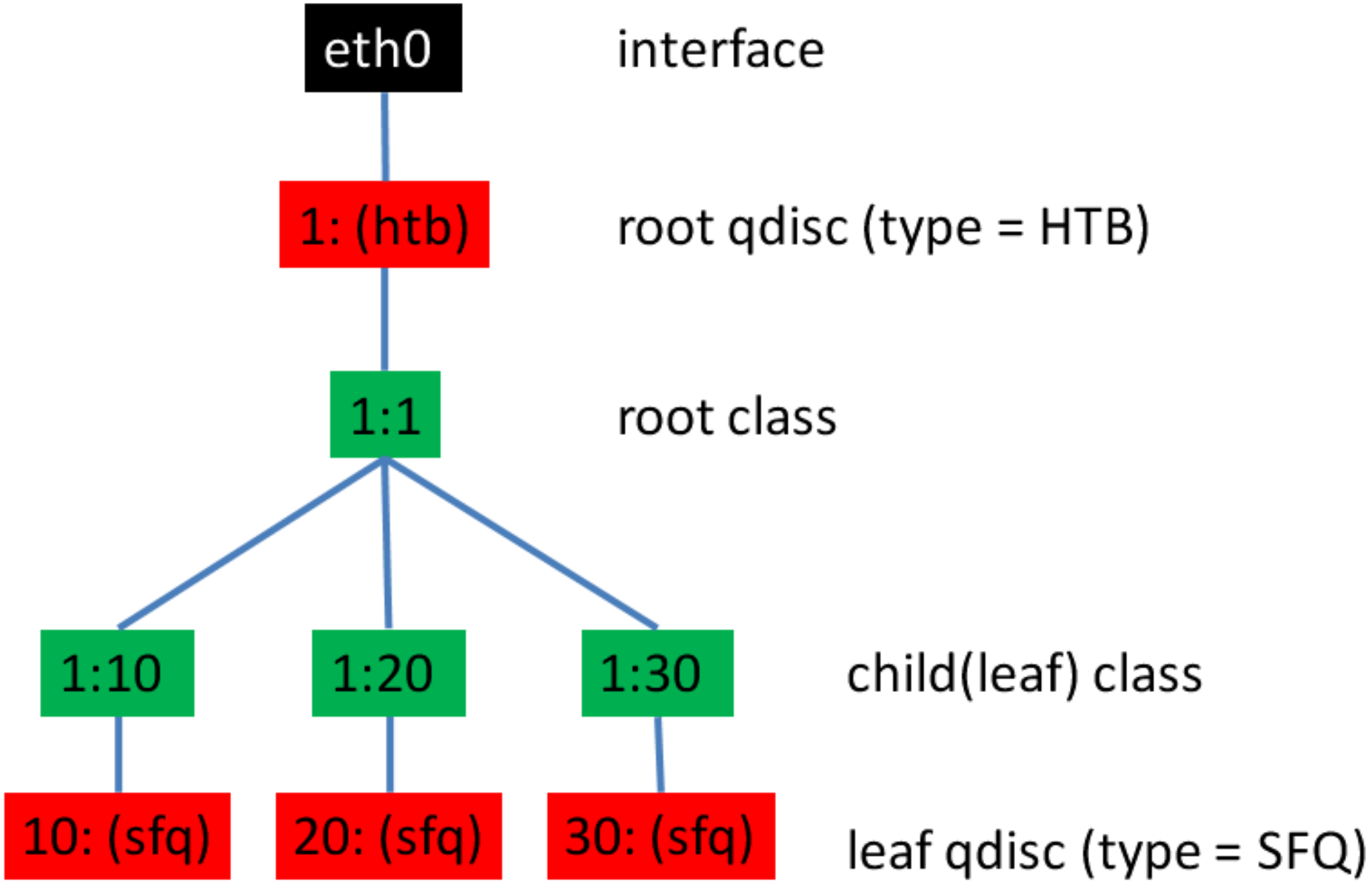
Gerarchia di qdisc e classi

Le qdisc e le classi sono identificate da handle tramite una notazione di due byte, nella forma **major**:**minor**

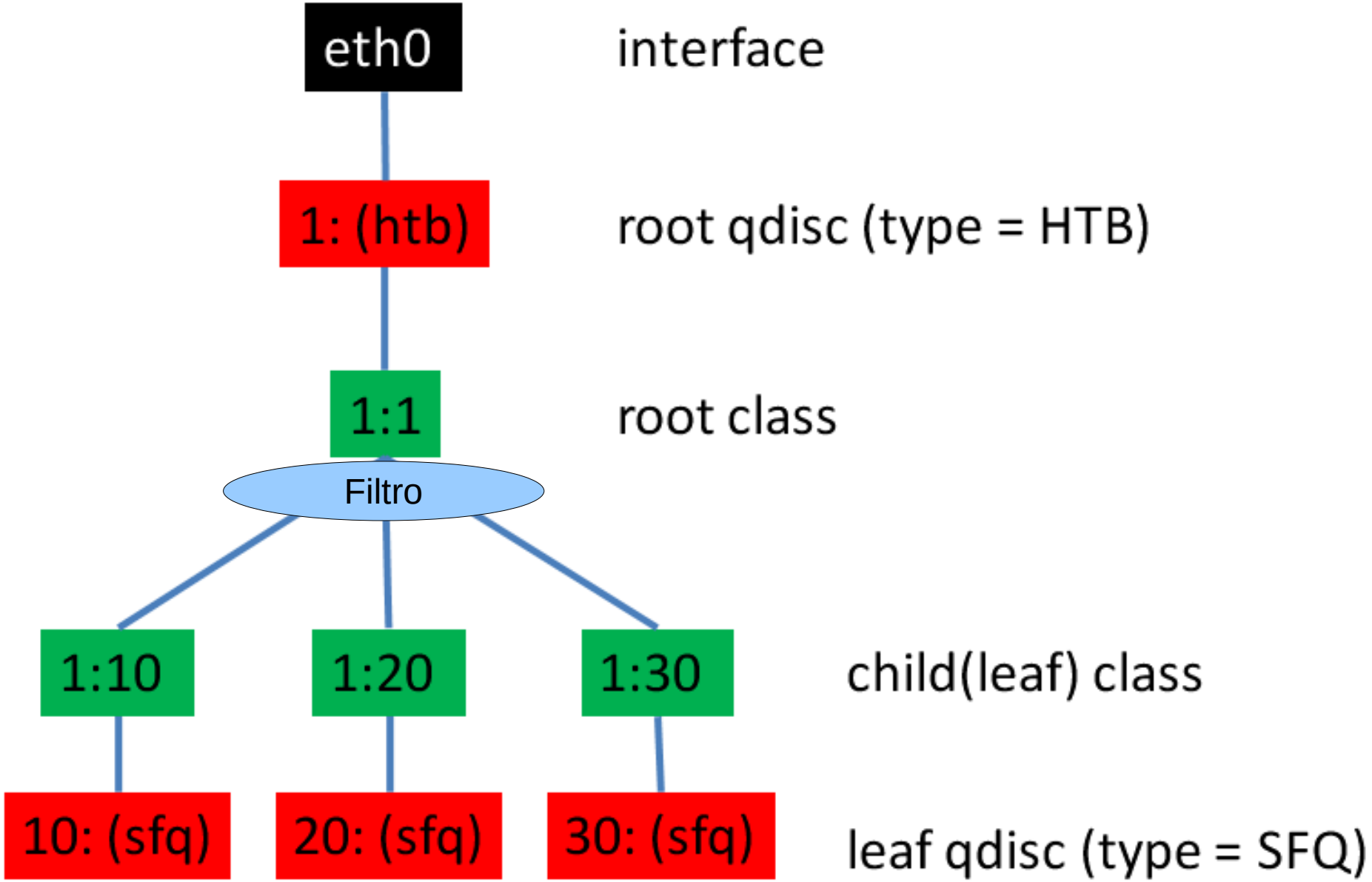
- **major** può essere scelto arbitrariamente, e viene solitamente associato a un qdisc. Il major delle classi figlie sarà solitamente lo stesso del qdisc padre
- **minor** è uguale a 0 per la qdisc radice, per gli altri nodi ha valori diversi da 0 e univoci nella gerarchia

La root qdisc è identificata per convenzione con la notazione 1: (cioè 1:0)

Esempio di gerarchia



Esempio di gerarchia



Traffic shaping su Linux: *tc*

tc = traffic control

Visualizzare le regole [e le statistiche] qdisc dell'interfaccia:

```
# tc [-s] qdisc show dev <iface>
```

Impostazione di default su kernel linux relativamente recenti

```
qdisc pfifo_fast 0: root refcnt 2 bands 3
  priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

Traffic shaping su Linux: *tc*

tc = traffic control

Visualizzare le regole [e le statistiche] qdisc dell'interfaccia:

```
# tc [-s] qdisc show dev <iface>
```

Impostazione di default su kernel linux relativamente recenti

```
qdisc pfifo_fast 0: root refcnt 2 bands 3
  priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

Handler:

- È una qdisc radice → il minor è 0 e viene omesso
- È la qdisc di default → il major è 0 (valore riservato)

Traffic shaping su Linux: *tc*

tc = traffic control

Visualizzare le regole [e le statistiche] qdisc dell'interfaccia:

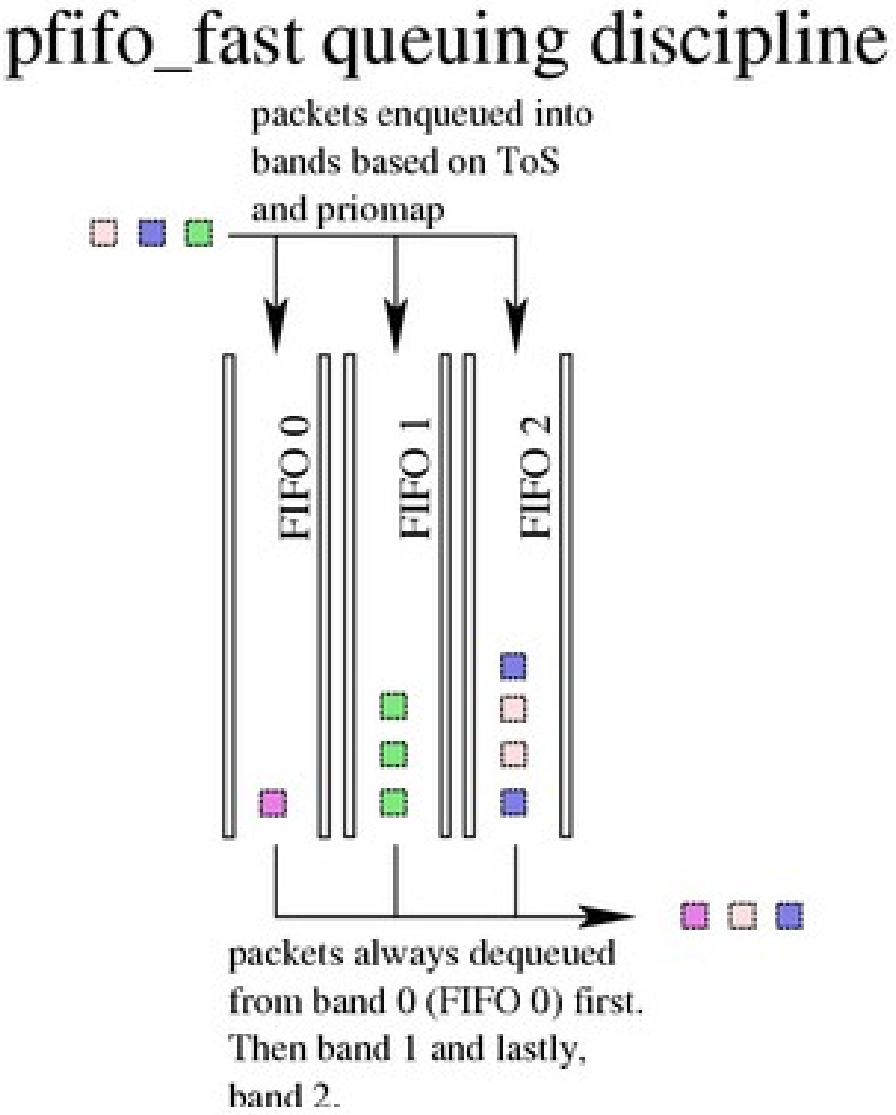
```
# tc [-s] qdisc show dev <iface>
```

Impostazione di default su kernel linux relativamente recenti

```
qdisc pfifo_fast 0: root refcnt 2 bands 3
  priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

Nome della queue discipline attualmente impostata

pfifo_fast


pfifo_fast queuing discipline
packets enqueued into
bands based on ToS
and priomap
FIFO 0
FIFO 1
FIFO 2
packets always dequeued
from band 0 (FIFO 0) first.
Then band 1 and lastly,
band 2.

Traffic shaping su Linux: tc

tc = traffic control

Visualizzare le regole [e le statistiche] qdisc dell'interfaccia:

```
# tc [-s] qdisc show dev <iface>
```

Impostazione di default su kernel linux relativamente recenti

```
qdisc pfifo_fast 0: root refcnt 2 bands 3
  priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

Numero di bande (3 nella configurazione di default)

Traffic shaping su Linux: *tc*

tc = traffic control

Visualizzare le regole [e le statistiche] qdisc dell'interfaccia:

```
# tc [-s] qdisc show dev <iface>
```

Impostazione di default su kernel linux relativamente recenti

```
qdisc pfifo_fast 0: root refcnt 2 bands 3
  priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

Mappa delle che indica a quale banda assegnare ogni pacchetto, a seconda del valore del campo TOS dell'header IP

TOS={0,4,8-15} → banda 1, TOS={1-3,5} → banda 2, TOS={6,7} → banda 0

Traffic shaping su Linux: tc

Impostare la root qdisc (o classe):

 # tc qdisc add dev *<iface>* root [handle <handle>] *<qdisc-type> <qdisc-options>*

Aggiungere una classe a un handle esistente:

 # tc qdisc add dev *<iface> parent <class-handle> <qdisc-type> <qdisc-options>*

Aggiungere un filtro:

 # tc filter add dev *<iface>* protocol *<proto>* parent *<qdisc-handle>* u32 match *<proto-rule>* flowid *<class-handle>*

Traffic shaping su Linux: *tc*

Eliminare la qdisc root:

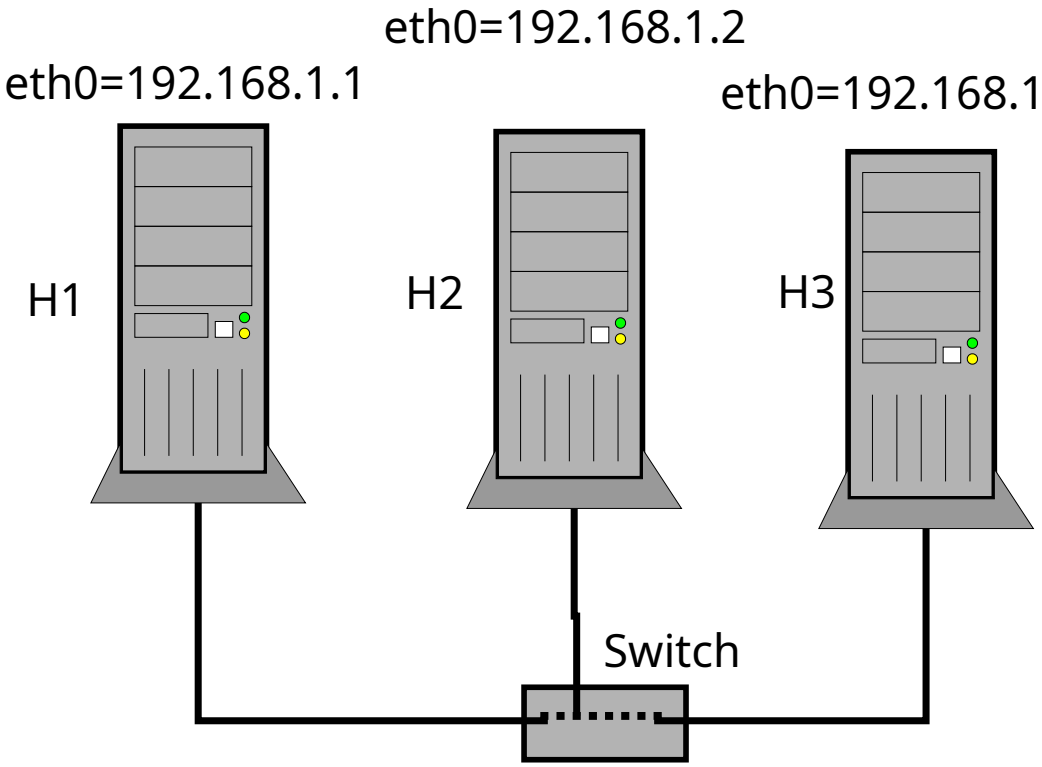
 # tc qdisc del dev *<iface>* root

Eliminare la qdisc corrispondente ad un handle:

 # tc qdisc del dev <iface> handle <handle>

Modulo 3: Esercitazione

Esercitazione 1



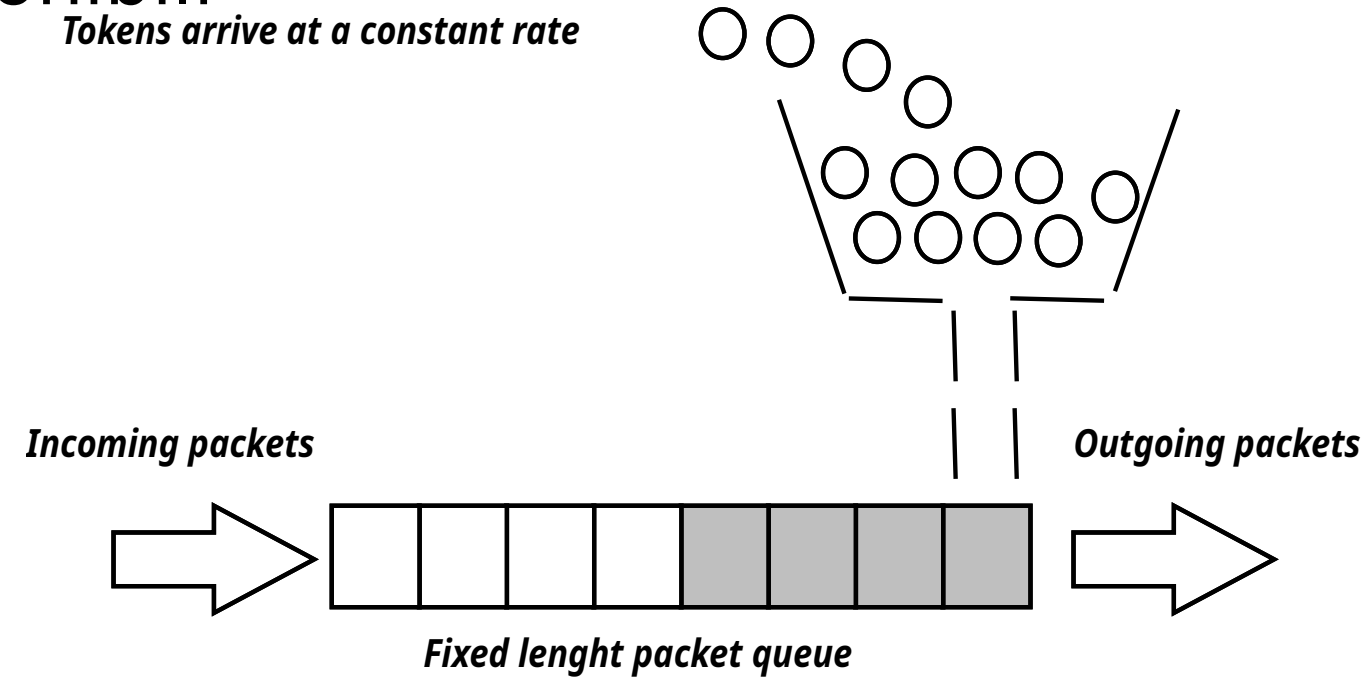
Obiettivi:

1. impostare una qdisc tbf (token bucket filter) su H1 con rate 220kb/s, burst 1539b, late 48ms. Verificarne il funzionamento tramite l'invio di un file verso H2
2. fare in modo che la limitazione precedentemente impostata valga solo per il traffico *verso* H2

Classless Queiung Discipline: Token bucket filter

TBF è una classless qdisc che modella il traffico come un secchio (*bucket*) di monetine (*token*)

- il secchio si riempie a ritmo costante
- spedire dati consuma dei token
- non si inviano dati se non ci sono abbastanza token disponibili

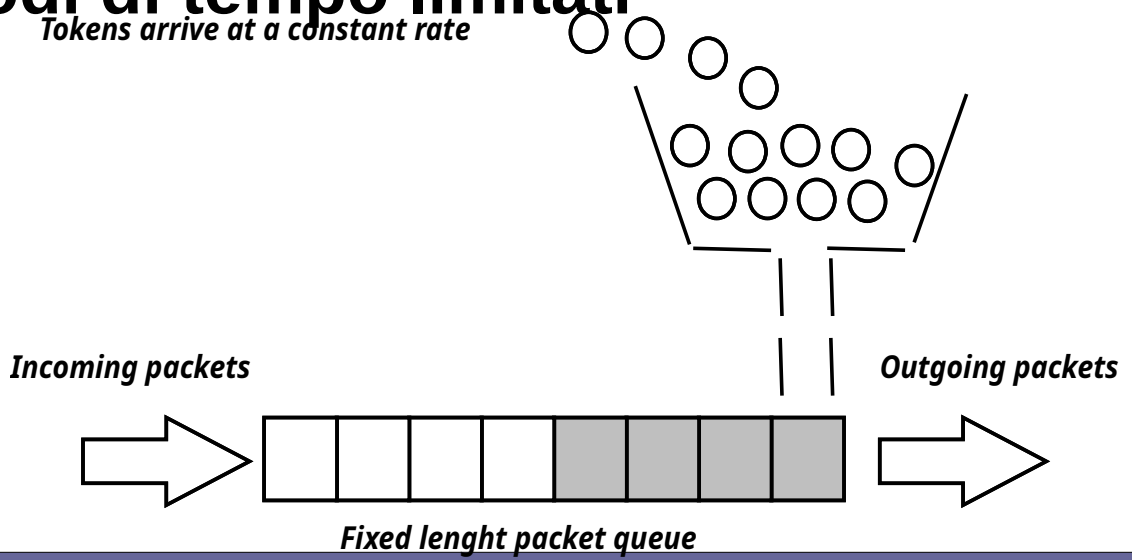


Classless Queiung Discipline: Token bucket filter

Il TBF manda dati a ritmo costante. E' abbastanza preciso e CPU friendly → *la presenza di virtualizzazione tuttavia ne altera le prestazioni in modo sensibile*

Parametri:

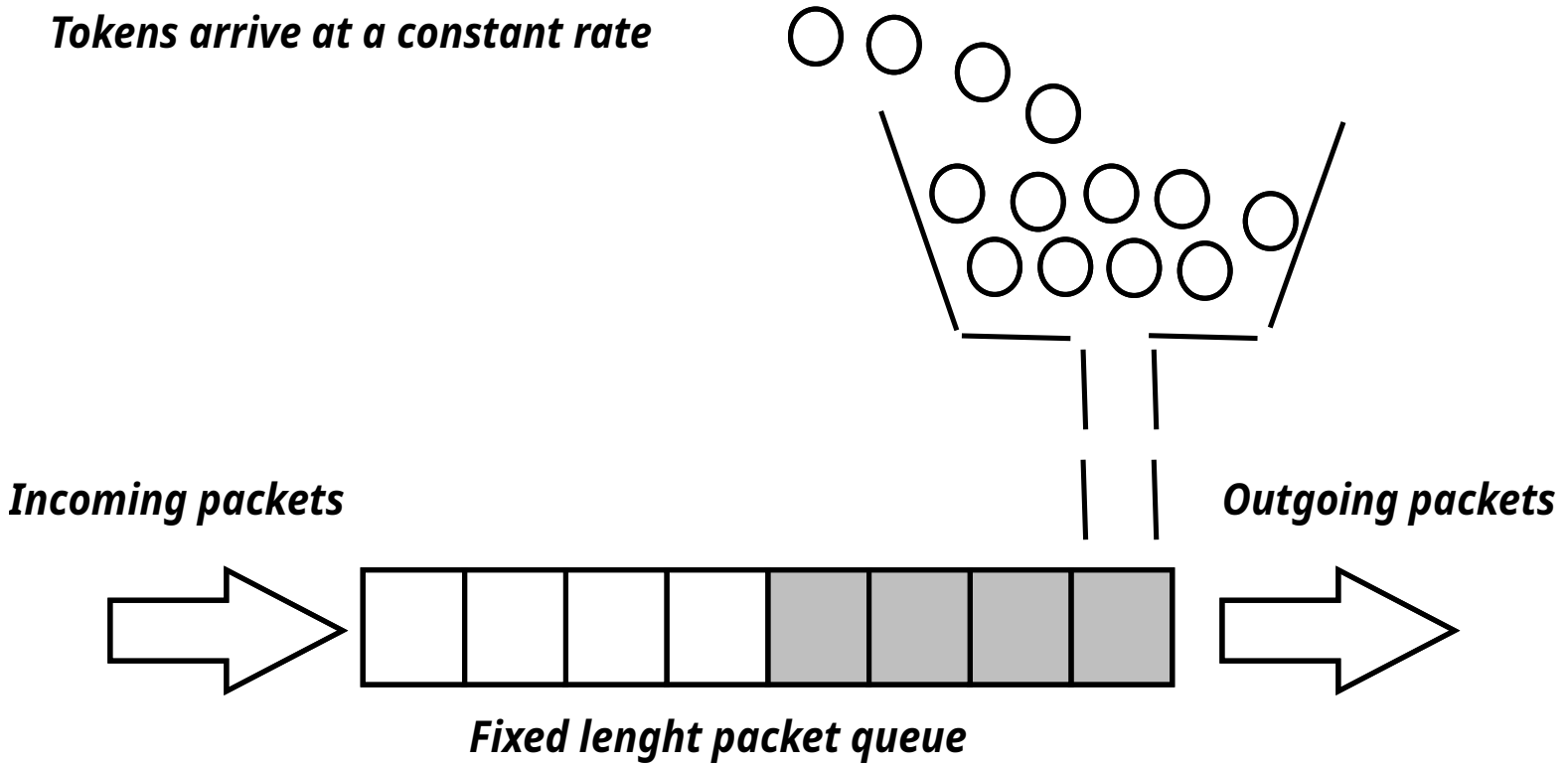
- **rate: il ritmo a cui riempio il secchio → quantità di pacchetti inviabili a ritmo costante**
- **burst: la dimensione del secchio → quantità di pacchetti inviabili a ritmo maggiore a quello stabilito, per periodi di tempo limitati**



Classless Queiung Discipline: Token bucket filter

Configurare una qdisc di tipo TBF:

```
# tc qdisc add dev <iface> \
  {root,parent <handle>} tbf \
  rate <rate> burst <burst> latency <latency>
```



Classless qdisc

Verifichiamo la banda della connessione provando a copiare un file da H1 a H2. Per creare un file usiamo dd:

H1 : **# dd if=/dev/zero of=file.bin bs=1M count=1**

Ora possiamo trasferire il file su H2:

```
H2: nc -l -p 8080 > /dev/null
H1: time sh -c “cat file.bin | nc 192.168.1.2 8080 -q1”
```

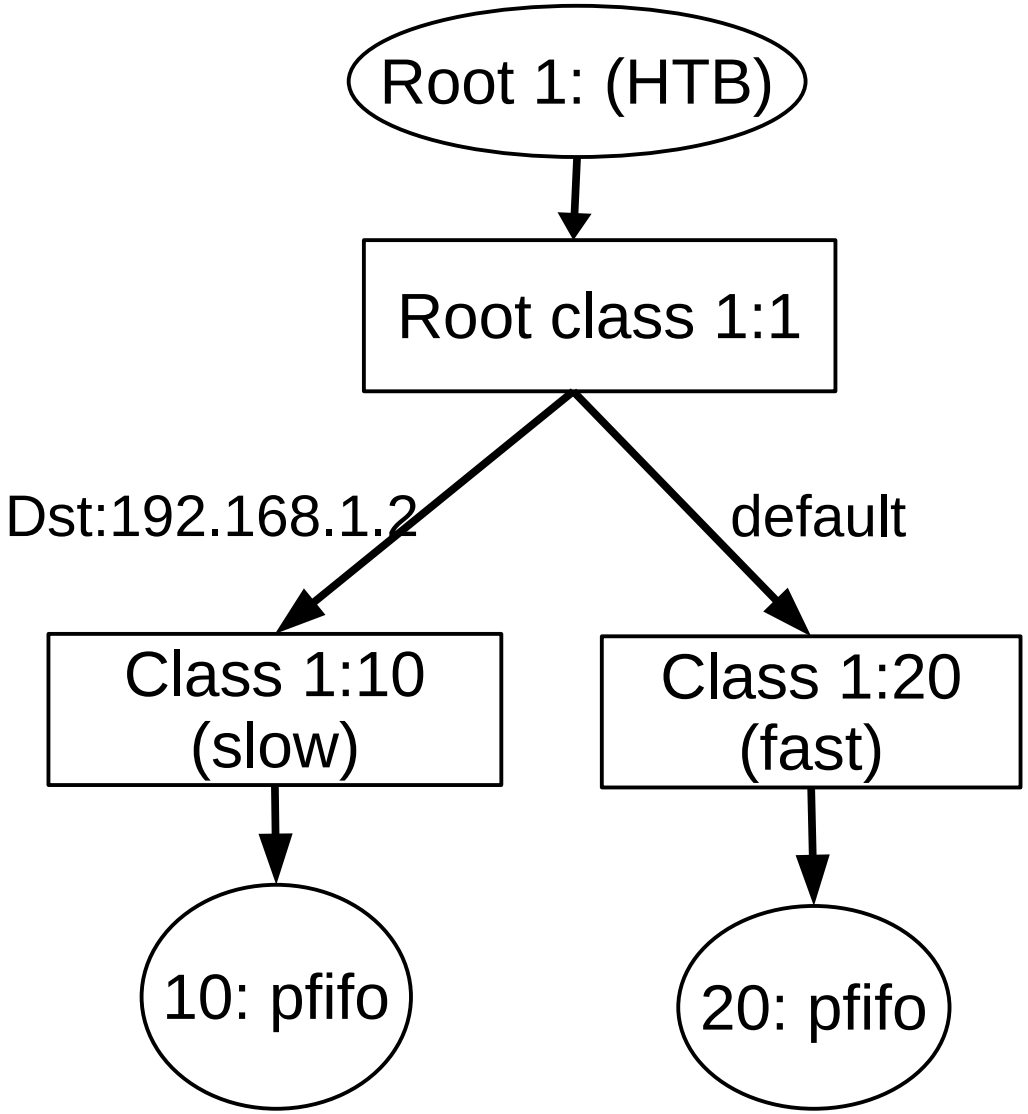
Classless qdisc

Configurazione qdisc tbf su eth0 di H1:

```
# tc qdisc add dev eth0 root tbf rate 1Mbit
  latency 50ms burst 1539
```

Torniamo a trasferire il file e guardiamo cosa cambia nei tempi di trasferimento

Classful qdisc



Soluzione

Eliminiamo la classe tbf impostata in precedenza sulla root qdisc:

```
# tc qdisc del root dev eth0
```

Inseriamo la qdisc HTB nella root della gerarchia:

```
# tc qdisc add dev eth0 root handle 1: htb default 20
```

Colleghiamo alla qdisc root appena creata la classe htb:

```
# tc class add dev eth0 parent 1: classid 1:1 htb
  rate 100Mbit burst 15k
```

Aggiungiamo alla classe 1:1 le due classi figlie

```
# tc class add dev eth0 parent 1:1 classid 1:10 htb
  rate 1Mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:20 htb
  rate 20Mbit ceil 50Mbit burst 15k
```

Soluzione

Aggiungiamo alla classe 1:10 la qdisc 10:

```
# tc qdisc add dev eth0 parent 1:10 handle 10:
  pfifo limit 50
```

Aggiungiamo alla classe 1:20 la qdisc 20:

```
# tc qdisc add dev eth0 parent 1:20 handle 20:
  pfifo limit 50
```

Aggiungiamo il filtro per redirigere I pacchetti con ip destinazione 192.168.0.2 alla classe 1:10 (quella più lenta):

```
# tc filter add dev eth0 protocol ip parent 1:0
  prio 1 u32 match ip dst 192.168.1.2 flowid
  1:10
```

Test

Trasferire lo stesso file verso 192.168.1.2 e 192.168.1.3, verificare la differenza dei tempi di trasferimento

Visualizzare e statistiche di pacchetti trasferiti sulle varie code con il comando:

```
# tc -s qdisc show dev eth0
```