Linux Network Testing

Introduzione agli strumenti per il testing di rete su Linux

Vittorio Palmisano

6, 13 Novembre 2007

Sommario

Outline

Testing di applicazioni di rete

Serve a:

- Validare un protocollo/algoritmo direttamente sul campo
- Scoprire problemi legati all'implementazione

Ambienti simulati vs ambienti emulati

- **simulare** significa riprodurre le prestazioni di un sistema reale (ritardo, banda, perdita di pacchetti, ecc.) usando funzionalità limitate e valide solo all'interno di quell'ambiente (es. NS-2).
- **emulare** significa riprodurre le funzionalità di un sistema reale (configurazioni, architetture, protocolli, ecc.) e limitarne le prestazioni usando strumenti per il controllo della QoS (Quality of Service).

Outline

ifconfig

- **ifconfig** permette di apportare modifiche a livello Data link (2) e IP (3)
- alcuni esempi:
  - Attivare una interfaccia di rete:
    ```
    # ifconfig eth0 192.168.0.1 netmask 255.255.255.0
    broadcast 192.168.0.255 up
    ```
  - Cambiare il valore di MTU:
    ```
    # ifconfig lo mtu 1500
    ```
  - Mostrare la configurazione corrente:
    ```
    # ifconfig -a
    ```

route

- **route** opera a livello IP e permette di impostare le regole di routing per i pacchetti
- alcuni esempi:
  - Impostare la route di default:
    ```
    # route add default gw 192.168.0.1
    ```
  - Impostare la route per il IP specifico:
    ```
    # route add -host 192.168.0.2 dev eth0
    ```
  - Mostrare la tabella di routing corrente:
    ```
    # route -n
    ```

Altri comandi utili

- **arp**: operazioni sulla tabella ARP (livello 2 e 3)
- **iwconfig**: interfacce wireless (livello 2)
- **ping**: invio messaggi ICMP/ping (livello 3)
- **tracepath** e **mtr**: invio messaggi ICMP/traceroute (livello 3)

Outline

iperf

- Serve a generare flussi TCP e UDP
- Funzionamento come receiver (server):
  `$ iperf -s -p 1234`
- Funzionamento come sender (client):
  `$ iperf -c 192.168.0.1 -p 1234 -t 600`
- Aggiungendo il parametro -**u** si ottengono flussi UDP

Outline

tcpdump

- Serve a catturare i pacchetti che passano attraverso una interfaccia di rete
- Necessita di una interfaccia di rete che supporti il *promiscuous mode*
- Alcuni esempi:
  - `# tcpdump -n -i eth0`
  - `# tcpdump -n -i eth0 'tcp and dst 192.168.0.1'`
  - `# tcpdump -n -i eth0 -X 'udp and dst port 1234 and src port 4321'`
  - `# tcpdump -n -i any -w outfile.cap`

getsockopt()

- Chiamata di sistema UNIX
- Serve a catturare le informazioni esportate del kernel per ogni socket aperto
- Esempio:

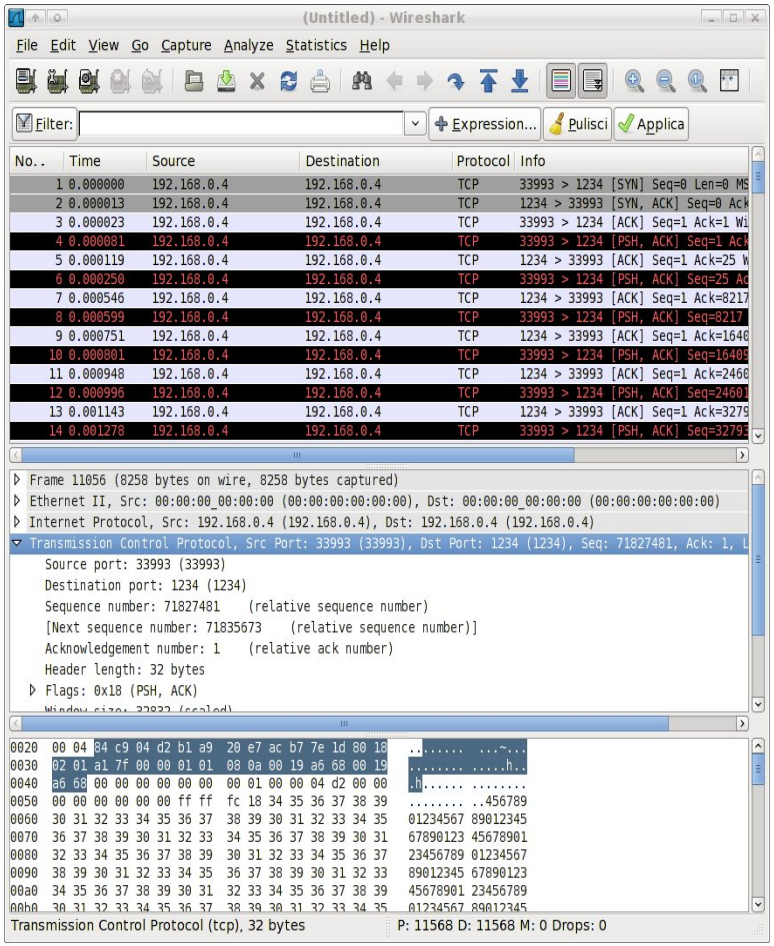
```
#include <netinet/tcp.h>
...
struct tcp_info info = {0};
int info_len = sizeof(struct tcp_info);
getsockopt(send_sock, SOL_TCP, TCP_INFO,
        (void*)&info, &info_len);
printf("%u %u %u\n", info.tcpi_snd_ssthresh,
        info.tcpi_snd_cwnd, info.tcpi_rtt);
```

Web100

- Serve a monitorare le variabili interne per ogni socket aperto (RTT, CWND, ecc..)
- Composto da:
  - una patch per il kernel Linux che abilita il monitor delle variabili interne
  - interfacce e librerie a livello applicativo che utilizzano le API
- L'interfaccia **gutil**

Altri monitor di rete

- **iptraf**
- **netstat**
  es: `# netstat -ntpae`
- **wireshark** (ex ethereal) e tshark
- **etherape**



Outline

Cosa è possibile fare

- limitare il **rate** con cui vengono spediti i pacchetti
- emulare i **ritardi** presenti in reti di grandi dimensioni (WAN)
- emulare la **perdita** di pacchetti secondo una percentuale desiderata
- emulare pacchetti **duplicati**
- emulare l'arrivo di pacchetti **fuori ordine**

Outline

Il pacchetto iproute2

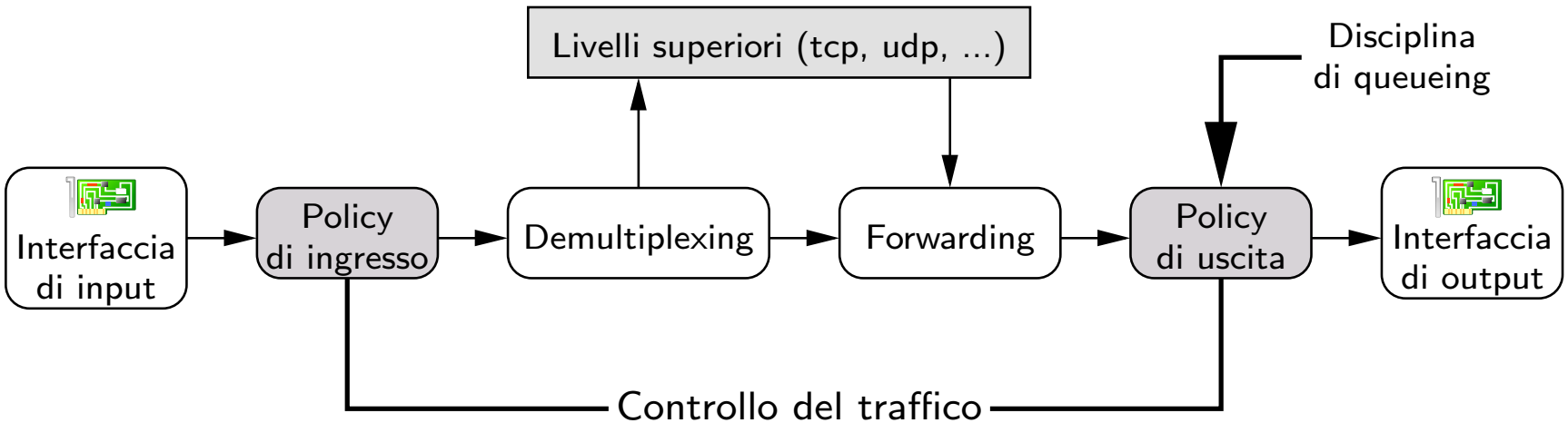
- Composto da una serie di programmi userspace che permettono di gestire i parametri di QoS esportati dal Kernel
- Necessita dei moduli del kernel attivabili dal menu:

```
Networking --->
    Networking options --->
        QoS and/or fair queueing --->
```

- Permette di:
  - impostare delle policy di routing avanzate, con possibilità di gestire reti multiple e load balancing del traffico
  - configurare tunnel IP e GRE
  - configurare il protocollo IPSEC
  - impostare delle discipline di **queueing** e gestire la **banda** disponibile
  - classificare in maniera avanzata i pacchetti in base al contenuto o all'instradamento

Controllo del traffico

Schema di elaborazione dei pacchetti in Linux (**iptables**):



Discipline di queueing: il comando tc

- La caratteristica più importante utilizzata è quella che permette di limitare la banda **in uscita** e/o applicare ritardi
- La modalità con cui devono essere spediti i pacchetti che attraversano una interfaccia di rete si indica tramite il parametro **qdisc** (*queueing discipline*)
- Tra le possibili qdisc ci sono:
  - **pfifo_fast**: semplice coda FIFO, dispone di tre 'bande', ciascuna con priorità diversa. I pacchetti inseriti nella banda 0 hanno priorità di invio maggiore su quelli inseriti nella banda 1 e così via per le bande 1 e 2.
  - **tbf**: è un meccanismo di controllo che utilizza l'algoritmo *Token Bucket Filter* per schedulare l'invio dei dati

tbf: parametri di configurazione (1/2)

- **limit**: il massimo numero di bytes che possono essere lasciati in coda in attesa
- **latency**: indica il limite di tempo che un pacchetto può attendere in coda prima che venga scartato (alternativo a limit)
- **burst** o **buffer** o **maxburst**: dimensione in byte del bucket, è il numero massimo di byte che possono essere inviati allo *stesso istante*
- **rate**: il rate desiderato a cui deve essere limitato il traffico in uscita

tbf: parametri di configurazione (2/2)

Nota sul parametro burst o buffer

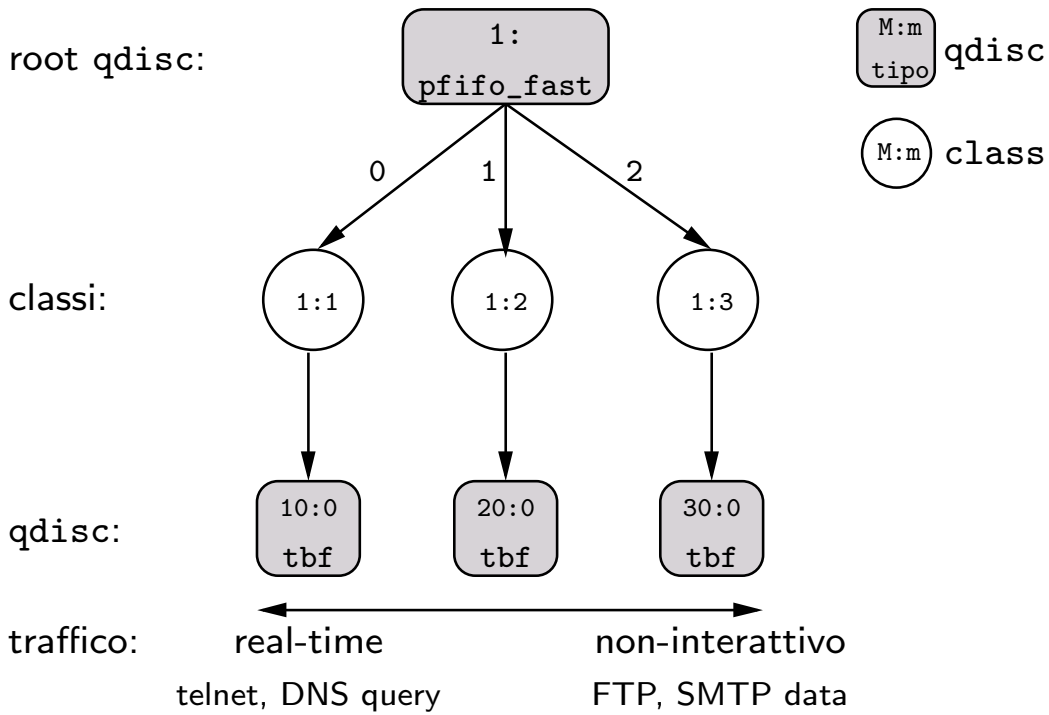
Un istante in kernel mode equivale a 1 **jiffies** ($1\,jiffies = \frac{1}{HZ}$, dove HZ dipende dalla configurazione del Kernel e può valere 10, 100 oppure 1000). Quindi, se $HZ = 100$, per ottenere un flusso di $1\,Mbyte/s$ è necessario impostare:

$burst = \frac{1\,Mbyte/s}{100\,s^{-1}} = 10Kbyte$

Tale valore deve essere superiore alla minima dimensione dei pacchetti (MTU) che è pari di solito a 1500 byte.

Classi e filtri (1/2)

- le **classi** servono per differenziare il traffico e applicare una diversa disciplina a seconda del tipo di dati
- gli elementi **filter** permettono di specificare delle condizioni secondo cui viene effettuato il routing



Classi e filtri (2/2)

- Sia le qdisc che le classi possiedono un **identificativo** formato da due numeri **major:minor**
- L'identificativo viene passato come argomento del parametro **handle**
- L'handle permette di identificare l'elemento stesso e il suo genitore
- I qdisc hanno sempre minor number 0
- Ogni interfaccia di rete possiede di default almeno un qdisc con handle pari a 1: (**root qdisc**) di tipo pfifo_fast
- Per le classi il major number di un elemento deve essere lo stesso dell'elemento genitore,
- Il minor number deve essere unico per le classi che si trovano sotto lo stesso qdisc

Un esempio

Impostare un limite di 20 Kbyte/s (160 Kbit/s) al traffico che passa attraverso l'interfaccia lo (loopback)

```
# tc qdisc add dev lo root tbf rate 160kbit buffer
1600 limit 3000
# tc -s qdisc
# tc qdisc del dev lo root
```

Il modulo netem

- Aggiunge un nuovo qdisc a quelli visti prima
- Serve a generare ritardi, loss, packet re-ordering
- Destinato soprattutto al testing di reti e di protocolli

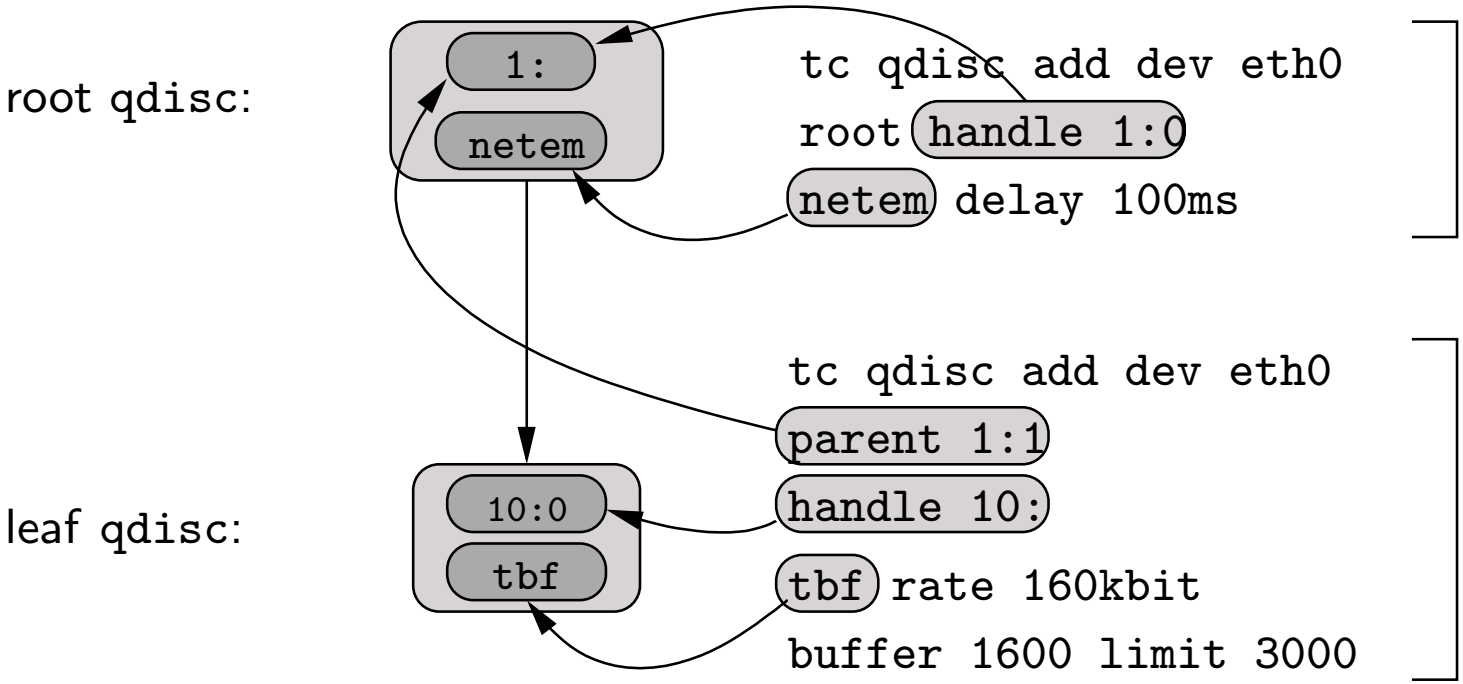
Aggiungere semplice ritardo sull'interfaccia di rete lo

```
# tc qdisc add dev lo root netem delay 100ms
```

Il ritardo applicabile è limitato come risoluzione dalla frequenza del timer delle interruzioni (dato dalla variabile HZ definita nel file include/asm/param.h del sorgente del kernel Linux). Su architetture Intel (indicate come i386 nel kernel) tale risoluzione è pari a 10 ms (oppure 1 ms se si utilizza HZ = 1000).

Esempio: ritardo e limitazione di banda

```
# tc qdisc add dev lo root handle 1:0 netem delay
100ms
# tc qdisc add dev lo parent 1:1 handle 10:  tbf rate
160kbit buffer 1600 limit 3000
```



Riferimenti

[c3lab] http://c3lab.poliba.it/index.php/MetodiDiControllo

[web100] http://www.web100.org/

[tc] http://lartc.org

[netem] http://linux-net.osdl.org/index.php/Netem