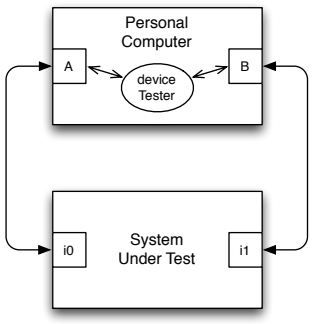
# DeviceTester: progetto di Gestione di Reti, a.a. 2011/12

Federico Mariti

## Indice



# 1 Scopo del progetto




Fornire una caratterizzazione delle prestazioni di un dispositivo di interconnessione di rete (System Under Test o SUT) in condizioni di traffico realistiche. In particolare è richiesta la determinazione del massimo throughput sostenibile, della latenza e del jitter del SUT. Il SUT dispone di due interfacce di rete che operano sia in ricezione che in invio, tali interfacce sono direttamente collegate con quelle di un personal computer sul quale viene eseguito il programma di test da realizzare. Quando il SUT riceve un pacchetto su di una interfaccia, se sussiste l'inoltro, questo è sull'interfaccia opposta. Il SUT viene sollecitato con del traffico letto da un file pcap [1], in questo modo viene approssimata una situazione realistica di traffico in cui si può venire a trovare il SUT. I pacchetti letti dal file pcap vengono inoltrati al SUT in modo sequenziale secondo l'ordine di lettura, sia alla stessa velocità della cattura che ad una velocità costante specificata dall'utente. I pacchetti contenuti nel traffico pcap vengono partizionati in due secondo un criterio scelto dall'utente, in modo da sollecitare il SUT su entrambe le interfacce.

# 2 Definizioni

L'Internet Engineering Task Force (IETF) ha descritto in diverse RFC le metriche e le metodologie per la misurazione delle prestazioni di un dispositivo di interconnessione di rete. L'RFC 1242 fornisce un insieme base di definizioni che riguardano il traffico osservabile all'ingresso del SUT. L'RFC 4689 riguarda lo studio dei sistemi di controllo del traffico e per questo motivo amplia e generalizza la 1242 con delle definizioni che considerano il traffico osservabile all'uscita del SUT.

**Throughput** The maximmum rate at which none of the offeres frames are dropped by the device. [3]

**Latency** For store and forward device. The time interval starting when the last bit of the input frame reaches the input port and ending when the first bit of the output frame is seen on the output port. [3]

**Forwarding Capacity** The number of packets per second that a device can be observed to transmit successfully to the correct egress interface in response to a specified offered load, while the device drops none of the offered packets. [6]

**Frowarding Delay** The time interval starting when the last bit od the input IP packet is offeres to the input port of the DUT/SUT and ending when the last bit of the output IP packet is received from the output port of the DUT/SUT. [6]

**Jitter** The absolute value of the difference between the Forwarding Delay of two consecutive received packets belonging to the same stream. [6]

L'RFC 2544 descrive una metodologia per misurare le definizioni nella RFC 1244, tale metodologia non è però applicabile al programma relizzato in quanto non prevede l'utizzo di più protocolli durante il test.

# 3 Misurazioni

Il modo in cui vengono effettuate le misurazioni è semplice, per ogni pacchetto viene registrato l'istante temporale d'invio e quello di ricezione, da tali due dati si calcolano le misure.

## 3.1 Latenza

La latenza di un pacchetto di test viene calcolata come la differenza tra l'istante temporale in cui il pacchetto viene ricevuto dal programma di test e l'istante in cui il pacchetto viene inviato dal programma di test. La misura calcolata risulata differente dalla *latency* [3] e dal *forwarding delay* [6], in quanto non è possibile non misurare il ritardo indotto dai componenti del sistema di test (quali il tempo di propagazione nel canale di comunicazione o il ritardo non nullo indotto dal personal computer). Chiamando *latenza di rete* l'intervallo di tempo che comincia quando il primo bit del frame viene visto sull'interfaccia di uscita del sistema di test, e finisce quando il primo bit dello stesso pacchetto viene ricevuto nell'interfaccia di ingresso del sistema di test, ciò che viene misurato è una *approssimazione della latenza della rete*, in quanto la misura finisce con l'arrivo dell'ultimo bit nell'interfaccia di ingresso del sistema di test.

## 3.2 Jitter

Il jitter è stato definito come la variazione della latenza di due pacchetti contigui nello stream. Dato che la latenza è misurata come approssimazione della latenza di rete, la misura del jitter è caratterizzata dai seguenti quattro parametri: *A*) istante d'invio del primo pacchetto dal sistema di test *B*) istante di ricezione del primo pacchetto nel sistema di test *C*) istante d'invo del pacchetto successivo dal sistema di test *D*) istante di ricezione del pacchetto successivo nel sistema di test . Il jitter sarebbe perciò misurato come:

$$J = |(B - A) - (D - C)|$$

Esistono diversi metodi per misurare il jitter, come descritti in [7]. Si è utilizzato il metodo realtime (motivazioni in 4.1) che segue il seguente algoritmo:

(1) ricezione di un pacchetto

(1.1) non è disponibile la latenza del pacchetto precedente a quello corrente nello stream di test
  (1.1.1) calcola la latenza e memorizzala come primo valore
(1.2) else
  (1.2.1) il pacchetto corrente è quello atteso
    (1.2.1.1) calcola la latenza del pacchetto e memorizzala come secondo valore
    (1.2.1.2) calcola il jitter come la variazione delle ultime due latenze in valore assoluto
    (1.2.1.3) aggiorna i valori di massimo e minimo del jitter
    (1.2.1.4) imposta come primo valore di latenza il corrente secondo valore
  (1.2.2) else
    (1.2.2.1) calcola la latenza del pacchetto e memorizzala come primo valore

### 3.3 Throughput

Durante l'esecuzione del test vengono mantenuti il numero di pacchetti e dei bytes ricevuti correttamente e il numero dei pacchetti e dei bytes scartati in quanto non attesi.

$$elapsed\ time = end\ time - start\ time\ (sec)$$

$$Throughput = \frac{packets\ good}{elapsed\ time}\ (pps) \quad = \quad \frac{bytes\ good}{elapsed\ time}\ (bps)$$

## 4 Implementazione

### 4.1 Funzionamento generale

Il programma di test è costituito da un unico processo che analizza il file pcap specificato dall'utente, ne memorizza in memoria le informazioni sui pacchetti e quindi inoltra in modo sequenziale i pacchetti letti. L'inoltro sequenziale consiste nell'inviare il pacchetto corrente sulla relativa interfaccia ed attendere la ricezione dello stesso pacchetto sull'interfaccia opposta, non si passa all'inoltro del pacchetto successivo fintanto che non è stato ricevuto il pacchetto corrente, neanche se è trascorso l'intervallo di tempo che separa tali due pacchetti nel file pcap. A seconda del tipo di SUT da testare l'utente può definire un intervallo di timeout sulla ricezione, ed un numero di tentativi di rispedizione al verificarsi di tale evento. Il pacchetto è dato per perso se non viene ricevuto entro il numero massimo di ritrasmissioni-timeout.

Si considera come caso tipico d'utilizzo quello in cui viene stabilito un timeout in ricezione e nessun tentativo di rinvio. Un timeout è infatti utile per evitare l'attesa indefinita del programma di test durante la ricezione di un pacchetto. Nessun rinvio in caso di timeout consente di poter calcolare la latenza ed il jitter su di un pacchetto dato per perso ma ricevuto successivamente al timeout[1].

Il calcolo delle misurazioni viene fatto in real time durante l'esecuzione del test, dopo ciascuna ricezione. Non viene presa in considerazione una opzione di misurazione post test in quanto, nel caso di misurazioni real time, il tempo impiegato nei calcoli non influenza le misure successive, perché l'invio dei pacchetti è sequenziale.

### 4.2 Il problema dei timeout

Con l'uso di timeout nella ricezione si rende necessario gestire l'arrivo di un pacchetto che *non* risulta uguale a quello atteso, ovvero, distinguere tra le sequengti tre situazioni:

- il pacchetto ricevuto è quello atteso ma è stato corrotto,
- il pacchetto ricevuto è un pacchetto inviato precedentemente e considerato perso per via di un timeout nella relativa ricezione,
- come il caso precedente, ma è stato corrotto.

[1] ciò è garantito dal fatto che non si può avere l'arrivo fuori ordine di pacchetti

Si osserva che, data la natura sequenziale di inoltro e ricezione nel programma di test e nel SUT, non è possibile ricevere su una certa interfaccia un pacchetto considerato perso che sia stato inviato prima dell'ultimo pacchetto ricevuto correttamente sulla stessa interfaccia.

Potrebbe esistere un quarto caso nella ricezione di un pacchetto che non è uguale a quello atteso, ovvero nella situazione in cui il SUT riceva o invii pacchetti che non sono nella sequenza di test.

### 4.2.1 Nessuna gestione dei timeout

Si suppone che un evento di timeout sia sempre corrispondente ad un evento di perdita. Allora quando si riceve un pacchetto che non coincide con quello atteso si interpreta tale situazione come corruzione del pacchetto, nonostante sia possibile che quello ricevuto sia un pacchetto inviato precedentemente e *considerato* perso. Tale implementazione funziona bene quando il SUT filtra dei pacchetti e l'utente del programma di test conosce bene la latenza media del SUT, in modo da impostare un intervallo di timeout adeguatamente alto. Viceversa tale implementanzione funziona male se si imposta un intervallo di timeout troppo basso, nel senso che se avviene una ricezione di pacchetto corretto ma dato per perso le ricezioni future non sono sincronizzate con i confronti effettuati perciò i pacchetti verranno scartati anche se corretti.

### 4.2.2 Una gestione dei timeout

Un evento di timeout è interpretato come *presunta* perdita. Si definisce come sequenza di test la sequenza ordinata dei pacchetti letti dal file pcap, l'ordinamento è ovviamente sul tempo di cattura specificato nel file pcap.

$$S = (\ p_0,\ p_1,\ p_2,\ p_3,\ \ldots,\ p_n\ )$$

Ogni pacchetto è associato ad una delle due interfaccie in base ad un criterio stabilito dall'utente. In ogni istante, durante il test, esiste il pacchetto correntemente inviato ed atteso, $p_k$, gli ultimi pacchetti correttamente ricevuti sulle due interfacce A e B, $p_i$ e $p_j$. Ne segue che per ciasuna interfaccia esiste una sottosequenza di S i cui pacchetti sono presunti persi e correntemente attesi, ad esempio, supponendo che il criterio di partizionamento della sequenza di test sia "pari-dispari":

$$A_{persi} = (\ p_{i+1},\ p_{i+3},\ \ldots\ , p_{k-2}\ )\ , \quad A_{attesi} = (\ p_{i+1},\ p_{i+3},\ \ldots,\ p_{k-2},\ p_k\ )$$

$$B_{persi} = (\ p_{j+1},\ p_{j+3},\ \ldots,\ p_{k-1}\ )\ , \quad B_{attesi} = B_{persi}$$

Se sull'interfaccia A si riceve un pacchetto $p_z$:

- $p_z \in A_{persi}$ allora i pacchetti in $\{\ p_{i+1},\ \ldots,\ p_{z-1}\ \}$ sono considerati definitivamente persi, metre si ha $A_{attesi} = (\ p_{z+1},\ p_{z+3},\ \ldots,\ p_{k-2},\ p_k\ )$. Si effettuano le misure possibili su $p_z$ e si effettua nuovamente la ricezione per $p_k$.
- $p_z = p_k$ allora $A_{persi}$ sono effettivamente persi. Si effettuano le misure possibili su $p_k$ e si procede con il prossimo invio.
- $p_z \notin A_{attesi}$ allora $p_z$ è la corruzione di uno dei pacchetti in $A_{attesi}$, non è possbile stabilire quale. Ci si comporta presumendo perso $p_k$, quindi $A_{persi} = A_{attesi}$ e si passa all'invio successivo.

Si osserva infine che nel caso in cui il SUT riceva ed inoltri pacchetti che sono fouri dalla sequenza di test, tale gestione della ricezione conteggia questi pacchetti come scartati e fa proseguire oltre, il pacchetto che era atteso verrà riconosciuto e conteggiato in futuro.

### 4.2.3 Imlpementazione della gestione descritta

Vengono presentate le variabili ed i tipi di dato utili per definire il comportamento dopo la ricezione:

```
struct packet {
  usec_t                delta;
  unsigned int          len;
  unsigned char *       data;
  unsigned char         interface;
  struct timeval        test_send_ts;
```

```
  struct packet *       next;
};
```

Listing 1: Tipo di dato per descrivere un pacchetto nella sequenza di test

```
  usec_t                latency[5]              = {
    0 /*current*/, 0 /*maximum*/, USEC_T_MAX /*minimum*/, 0 /*sum*/, 0 /*square*/
  };
  usec_t                jitter[5]               = { 0, 0, USEC_T_MAX, 0, 0 };
  usec_t                latency_previous        = 0;
  struct packet *       itr                     = list.head;
  struct packet *       last_correct[2]         = { NULL, NULL };
  unsigned char         out_intf                = itr->interface;
  unsigned char         in_intf                 = (out_intf + 1) % 2;
  unsigned long         pkt_good                = 0;
  unsigned long         pkt_disc                = 0;
  struct timeval        send_ts;
  const u_char *        bytes;
  struct pcap_pkthdr    h;
```

Listing 2: Variabili usate per gestire la ricezione

```
  gettimeofday(&send_ts, NULL);
  pcap_inject(live[out_intf], itr->data, itr->len);
  ... bytes = pcap_next(live[in_intf], &h) ...
```

Listing 3: L'inoltro e la ricezione

In generale siamo in una situazione in cui è stato inviato un paccheto sull'interfaccia `out_intf`, di tale interfaccia è noto l'ultimo pacchetto ricevuto `last_correct[out_intf]`; può esistere una sottosequenza non vuota di pacchetti tra `last_correct[out_intf]` e il pacchetto correntemente atteso che sono dati per persi (non ancora ricevuti). Il comportamento dopo una corretta ricezione è il seguente:

(1) `bytes` $\neq$ `NULL`

(1.1) `bytes` = `itr->data`
il pacchetto ricevuto è quello atteso

- `pkt_good ++`
- aggiorna `latency[]`
- *se* `latency_previous` $\neq$ `0` *e* ( `last_correct[0]->next` = `itr` *oppure* `last_correct[1]->next` = `itr` ) *allora* aggiorna `jitter[]`
- `latency_previous := latency[0]`
- `last_correct[out_intf] := itr`
- `itr := itr->next`

(1.2) esle

(1.2.1) $\exists$ `(struct packet *)pkt`$\in$ { `last_correct[out_intf]`,...,`itr`}`.bytes=pkt->data`
il pacchetto ricevuto è uno di quelli dati per persi

- `pkt_good ++`
- aggiorna `latency[]`
- *se* `latency_previous` $\neq$ `0` *e* ( `last_correct[0]->next` = `itr` *oppure* `last_correct[1]->next` = `itr` ) *allora* aggiorna `jitter[]`
- `latency_previous := latency[0]`
- `last_correct[out_intf] := pkt`

(1.2.2) else
il pacchetto ricevuto è corrotto, non è possibile stabilire se sia quello corrente o quelli considerati persi

- `pkt_disc ++`

- `itr := itr->next`

(2) else
il pacchetto inviato viene dato per perso

- `itr := itr->next`

# 5 Organizzazione del codice

Intestazioni:

**util.h** macro per la gestione degli errori,

**deviceTester.h** funzionalità usate dal programma di test, dichiarate pubblicamente in modo da poter effettuare verifa e validazione:

- analisi del file pcap e memorizzazione delle informazioni necessarie al test,
- regole di partizionamento dei pacchetti,
- comparazione di due pacchetti,
- simulazione di ritardi e perdita di pacchetti durante l'inivo e la ricezione nel programma di test.

**types.h**

**timeUtils.h**

Funzionalità:

**deviceTester_impl.c** implementazione delle funzionalità dichiarate nell'intestazione `deviceTester.h`,

**deviceTester.c** esecuzione del test e calcolo delle misurazioni.

Altro:

**printPcap.c** programma di utilità che utilizza le funzionalità pubblicate in `deviceTester.h` per l'analisi di un file pcap,

**tests.c** test sulle funzionalità usate dal programma di test.

# A Compilazione ed esecuzione

## A.1 Prerequisiti

- Libreria Pcap [1]
- compatibilità standard `POSIX` [2]
- Intestazioni BSD
  - `net/ethernet.h`,
  - `netinet/ip.h`

## A.2 Compilazione, installazione e rimozione

Compilazione dei sorgenti e generazione dei programmi `deviceTester`, `tests`, `printPcap`

```
$ make
```

Installazione del programma di test, la directory di installazione predefinita è `/usr/local/bin`:

```
$ make install INSTALL_DIR=$HOME/bin/
```

Rimozione del programma di test:

```
$ make install INSTALL_DIR=$HOME/bin/
```

## A.3 Casi d'uso

Si suppone che il sistema di test disponga di due interfacce ethernet `eth0` e `eth1` collegate direttamente con il sistema da testare. Il comportamento predefinito del programma è caratterizzato dall'inoltro dei pacchetti in modo unidirezionale, con la velocità specificata nel file pcap, esiste il timeout in ricezione di 1 secondo, nessuna rispedizione in caso di timeout, i risultati vengono forniti al termine del programma con un formato leggibile dall'utente. Tale comportamento può venire modificato ne seguenti punti:

- impostazione del valore di timeout,
- impostazione del numero di tentativi di spedizione,
- esecuzione con velocità costante,
- impostazione di una regola di partizionamento diversa da quella unidirezionale,
- modalità *trial* (esecuzione di una sequenza di test), le misure sono presentate anche in formato compatto nel file /tmp/dt-trialoutput-ddmmyyyy-MMHH,
- simulazione dei ritardi delle perdite di un SUT,
- impostazione del formato della stampa delle misure per la lettura di un utente o di un programma,
- impostazine della modalità prolissa, stampa le misure ogni secondo nel formato scelto.

Esecuzione di un test con intervalli temporali tra pacchetti fissati a 10 milli secondi, partizionamento dei pacchetti in base al valore pari o dispari dell'indirizzo ip:

```
# deviceTester -f $HOME/pcaps/lab.pcap -A eth0 -B eth1 -F 10000 -p ip
```

Come il precedente ma con il risultato in forma parsabile da un programma e il partizionamento in base al valore pari o dispari dell'indirizzo mac:

```
# deviceTester -f $HOME/pcaps/lab.pcap -A eth0 -B eth1 -o script -F 10000 -p mac
```

Esecuzione di un trial di 50 test intervallati da una pausa di 10 secondi:

```
# deviceTester -f $HOME/pcaps/lab.pcap -A eth0 -B eth1 -T 50:10
```

Esecuzione di un test con la simulazione di un SUT che perde/filtra pacchetti con probabilità 0.15, ha un ritardo (casuale) tra i 10 msec e i 150 msec:

```
# deviceTester -f $HOME/pcaps/lab.pcap -A eth0 -B eth1 -s 15:10:150
```

[*DA FARE*]Simulazione di un SUT come il precedente, ma con anche il filtro sui messaggi ARP e ICMP:

```
# deviceTester -f $HOME/pcaps/lab.pcap -A eth0 -B eth1 -s 15:10:150:arp.icmp
```

Differnza tra i due formati di stampa:

- utente:

  ```
  elapsed time: 1041754 usec
  packet: 4192 received 0 discarded 0 lost
  bytes: 2862962 received 0 discarded 0 lost
  throughput cur: 4023.982629 pps 2748213.109813 bps
  throughput avg: 4023.982629 pps 2748213.109813 bps
  latency (usec): 163.112118 cur 163.112118 avg 49702.235476 sdev 27636 max 28 min
  jitter (usec): 194.090649 cur 194.090649 avg 65811.163989 sdev 27606 max 0 min
  ```

- script:

  ```
  4023.982629 2748213.109813 163.112118 49702.235476 27636 28 194.090649
  65811.163989 27606 0
  ```

Tutte le opzioni:

```
Usage: deviceTester -f pcapfile -A dev -B dev [options]
Options: -f file                    pcap file
         -A dev, -B dev             device A and device B
         -h|--help                  print this help
         -t|--timeout msec          receive timeout
         -r|--resend-attemps num    num of resend attemps when to occurs
         -F|--fixed-deltas msec     interpacket elapsed time
         -p|--partition-rule str    str in {none,ip,mac,random,evenAndOdd}
         -s|--simulate a:b:c        simulate mode: simulate network delay
                                    and sut packet filtering/lost
                                    a=lost% b=sutMaxDelay c=sutMinDelay
         -T|--trial num:sec         execute trial test: num=tests,
                                    sec=sleep time
         -v|--verbose               verbose mode
         -o|--output-mode str       str in {human,script}
Default values: timeout=1000, resend-attemps=0, fixed-deltas=0,
                partition-rule="none"
```

## References


[1] TCPDUMP & LibPcap http://www.tcpdump.org/

[2] IEEE Std 1003.1-2008 http://pubs.opengroup.org/onlinepubs/9699919799/

[3] RFC 1242 *Benchmarking Terminology for Network Interconnection Devices*

[4] RFC 2545 *Benchmarking Methodology for Network Interconnect Devices*

[5] RFC 2285 *Benchmarking Terminology for LAN Switching Devices*

[6] RFC 4689 *Terminology for Benchmarking Network-layer Traffic Control Mechanisms*

[7] Spirent Communications. *White Paper. Measuring Jitter Accurately, 2007.* http://www.spirent.com/White-Papers/Broadband/PAB/Measuring_Jitter_Accurately_WhitePaper