Alessandro Olivero *Nunzio Onorati* *Leonardo Pio Petrillo*

01QZD

Laboratorio di Internet e Comunicazioni

Internet - Laboratorio 1

5. Connectivity check – Advanced ping

1) Cosa succede quando un terminale esegue un ping all'indirizzo di broadcast?

   Quando si esegue il comando ping verso l'indirizzo di broadcast viene richiesto di aggiungere il parametro -b al comando.

   ```
   ping 172.16.0.63 -b
   ```

   Il terminale richiedente invia a tutti gli altri il pacchetto icmp request, gli altri terminali lo ricevono e, se configurati per la risposta ognuno risponde con un pacchetto icmp reply.
   Di default il SO non è configurato per la risposta al broadcast in quanto può essere utilizzato per attaccare una rete (o un host) sovraccaricandola.
   All'interno della tabella arp del mittente non viene aggiunto alcun indirizzo mac in quanto il mittente non esegue alcuna arp request per un indirizzo IP che corrisponde a più host (indirizzo broadcast).
   Gli altri host, prima di rispondere eseguono un'arp request per identificare l'indirizzo mac del mittente. Esso non era noto a priori perché l'indirizzo del pacchetto ricevuto in broadcast non viene salvato nell'arp del destinatario.
   Una volta che il mittente avrà risposto all'arp request, il suo indirizzo mac sarà memorizzato nella tabella arp del destinatario ed esso provvederà a mandare un pacchetto di risposta. Il mittente iniziale calcola di conseguenza il RTT in base al primo pacchetto di risposta che riceve.

2) Cosa succede quando un terminale esegue un ping all'indirizzo di rete?

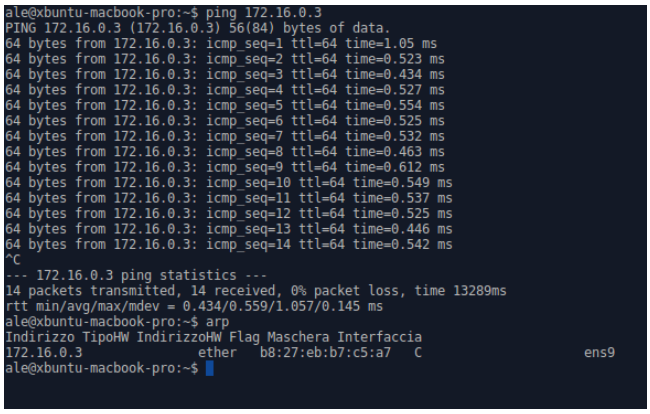
   Quando si esegue un ping all'indirizzo di rete il comportamento è identico a quello osservato al punto precedente.

Rete con indirizzo duplicato

Abbiamo configurato la rete per avere due host con lo stesso indirizzo IP (H1, H1')

1) Cosa succede quando H2 esegue un ping su H1?

   Inizialmente H2 esegue una arp request. Sia H1 che H1' rispondono alla richiesta. Nella tabella arp di H2 sarà memorizzato l'indirizzo mac dell'host che risponde più velocemente (ad esempio H1'), e dunque H2 inizierà a pingare H1'.



2) Cosa succede quando H1 e H1' incominciano a pingare H2 nello stesso momento? Controlla le tabelle di arp.

   Inizialmente H1 e H1' inviano ognuno una arp request verso H2, e salvano nelle rispettive tabelle arp l'indirizzo mac di H2. Nella tabella arp di H2 viene associato all'IP di H1 e H1' l'indirizzo mac del primo host di cui riceve l'arp request. Successivamente il valore viene sostituito quando anche in secondo host segue l'arp request.
   Assumiamo l'ultimo indirizzo mac che H2 ha memorizzato sia quello di H1'. Per ogni ICMP request ricevuta da H2, qualunque sia il mittente (H1 o H1'), H2 risponderà sempre ad H1', che riceverà dunque anche le risposte alle richieste di H1. Le risposte alle richieste di H1' sono coerenti al numero di sequenza, al contrario di quelle riferite ad H1.

Netmask sbagliate

Configuriamo H1 e H2 in modo tale che i rispettivi indirizzi IP abbiano netmask diverse. H1 risulta non appartenente alla stessa subnet di H2.

1) Cosa succede quando H1 esegue il comando di ping verso H2? Quali pacchetti invia H1? Quali pacchetti invia H2? Come cambiano le tabelle arp di H1 e di H2?

   Quando H1 vuole eseguire un ping verso H2 manda inizialmente una arp request. H2 riceve la richiesta, senza tuttavia fornire una risposta. Questo perché, se H2 rispondesse, H1 comincerebbe a mandare pacchetti ad H2, a cui esso non potrebbe rispondere, in quanto H1 viene visto da H2 al di fuori della rete in cui è collocato. Si verrebbe dunque a creare un collegamento logico unidirezionale, per cui sarebbe impossibile instaurare una comunicazione reciproca.
   Le tabelle arp di H1 e H2 rimangono dunque vuote.

```
354 967.098084    PcsCompu_e5:92:80    Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.66
355 967.098115    PcsCompu_e5:92:80    Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.66
356 968.121574    PcsCompu_e5:92:80    Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.66
357 968.121632    PcsCompu_e5:92:80    Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.66
358 969.145760    PcsCompu_e5:92:80    Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.66
359 969.145817    PcsCompu_e5:92:80    Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.66
```

2) Cosa succede quando H2 esegue il comando di ping verso H1? Quali pacchetti invia H2? Quali pacchetti invia H1? Come cambiano le tabelle arp di H1 e di H2?

   H2 prova a mandare una ICMP request verso H1. Tuttavia, al livello IP di H2 il pacchetto viene bloccato poiché l'IP di H1 non appartiene alla stessa rete di H2, e non sono disponibili rotte per raggiungerlo. Il controllo viene effettuato confrontando l'and bit a bit dell'indirizzo IP di H2 associato alla propria Netmask con quello dell'indirizzo ip di H1.

```
ale@xbuntu-macbook-pro:~$ arp
Indirizzo TipoHW IndirizzoHW Flag Maschera Interfaccia
172.16.0.3               ether   b8:27:eb:b7:c5:a7   C                     ens9
ale@xbuntu-macbook-pro:~$ ping 172.16.0.66
connect: Network is unreachable
ale@xbuntu-macbook-pro:~$ arp
Indirizzo TipoHW IndirizzoHW Flag Maschera Interfaccia
172.16.0.3               ether   b8:27:eb:b7:c5:a7   C                     ens9
ale@xbuntu-macbook-pro:~$
```

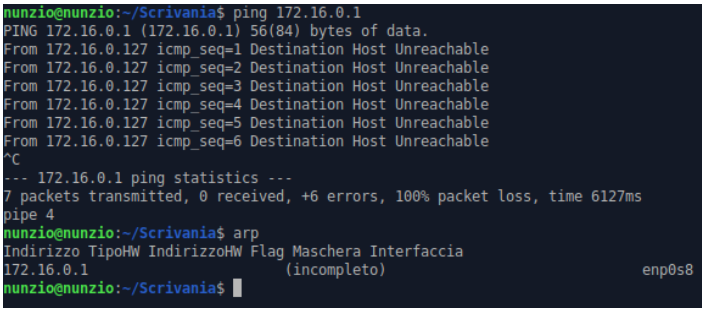
Conflitto con indirizzo di broadcast

Abbiamo configurato H1 e H2 in modo tale che H1 presenti l'indirizzo di broadcast della rete a cui appartiene H2:

- H1 con un indirizzo IP `172.16.0.127` con Netmask `255.255.255.0`
- H2 cin un indirizzo IP `172.16.0.1` con Netmask `255.255.255.128`

1) Cosa succede quando H1 pinga H2? Quali pacchetti sta trasmettendo H1? Quali pacchetti sta trasmettendo H2? Come cambiano le tabelle arp?

   Viene trasmessa una arp request da H1 a H2. L'arp di H2 dovrebbe fornire una risposta ad H1, tuttavia non lo fa perché H1 ha l'indirizzo IP di broadcast. Rispondere ad H1, dunque, significherebbe rispondere a tutti gli host appartenenti alla rete, cosa che non ha senso logico.



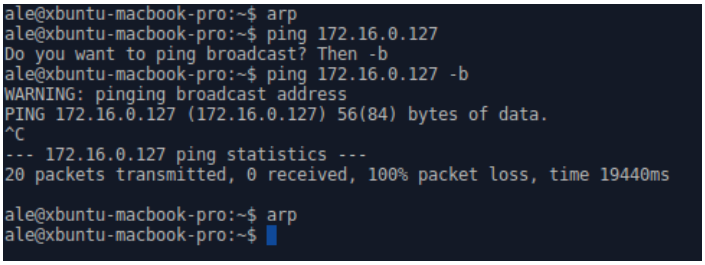
   Nella tabella arp di H1 viene inserita una entry corrispondente ad H2. Poichè H1 non riceve una risposta, il campo riservato all'indirizzo mac di H2 rimane incompleto.

   Per quanto riguarda invece la tabella arp di H2, essa rimane invariata.

```
703 1331.846004650 PcsCompu_e5:92:80      Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.127
704 1332.869213521 PcsCompu_e5:92:80      Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.127
705 1333.893413375 PcsCompu_e5:92:80      Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.127
706 1334.917459182 PcsCompu_e5:92:80      Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.127
707 1335.941780567 PcsCompu_e5:92:80      Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.127
708 1336.964805302 PcsCompu_e5:92:80      Broadcast           ARP         60 Who has 172.16.0.1? Tell 172.16.0.127
```

2) Cosa succede quando H2 pinga H1? Quali pacchetti sta trasmettendo H1? Quali pacchetti sta trasmettendo H2? Come cambiano le tabelle arp?

   H2 manda una ICMP request, senza effettuare prima una arp request, poiché l'indirizzo IP di H1 è l'indirizzo di broadcast della rete. Prima di inviare una risposta, H1 trasmette una arp request ad H2. Tuttavia, H2 non fornirà mai una risposta per gli stessi motivi evidenziati al punto 1).
   Anche in questo caso nella tabella arp di H1 viene inserita una entry relativa ad H2, e poiché H2 non fornisce una risposta alla arp request di H1, il campo riservato all'indirizzo mac di H2 rimane incompleto.



Alessandro Olivero *Nunzio Onorati* *Leonardo Pio Petrillo*

01QZD

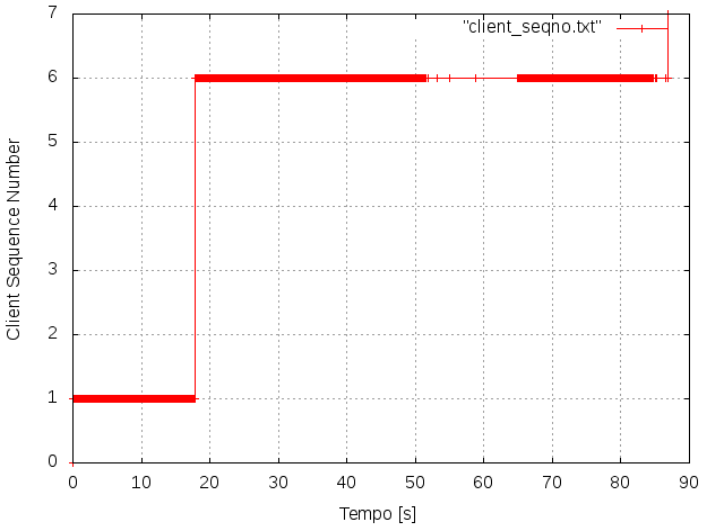
Laboratorio di Internet e Comunicazioni

Internet - Laboratorio 3

4. Analysis of the CHARGEN service

1) Numeri di sequenza del client

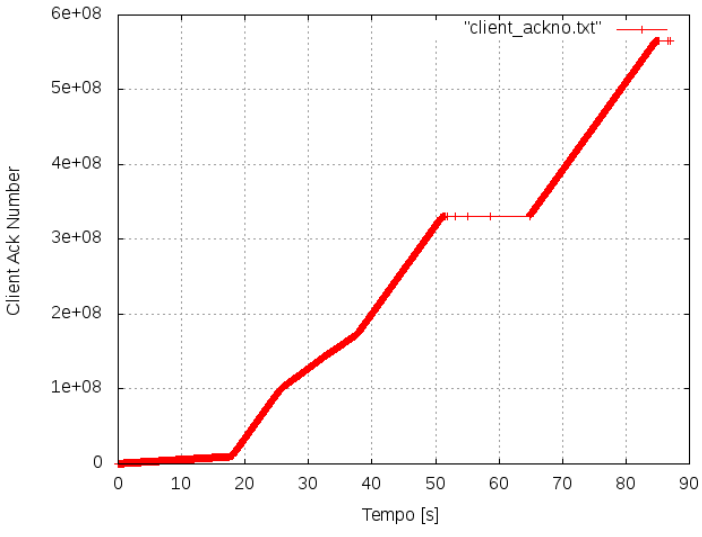
Il client inizialmente invia una SYN al server per inizializzare la connessione. Essa ha numero di sequenza relativo pari a 0. Dopo aver ricevuto la SYN ACK da parte del server, il client manda una ACK di risposta con numero di sequenza 1. Da quell'istante il client non trasmette più pacchetti, dunque il numero di sequenza si mantiene costante al valore unitario.
Dopo alcuni secondi, con il comando CTRL+C, si fa in modo che Telnet non effettui stampe su schermo dei pacchetti ricevuti. A questo punto il client manda una PSH ACK di lunghezza 5 byte al server al fine di velocizzare la trasmissione dei pacchetti. Di conseguenza, il numero di sequenza sale a 6. Esso si mantiene costante fino alla chiusura della connessione. In quel momento la FIN ACK mandata dal client causa un aumento del numero di sequenza che si assesta a 7.



2) Ack Number del client

L'ack number cresce linearmente perchè, a parità di condizioni nel tempo, la capacità del client di smaltire i pacchetti si mantiene all'incirca costante.
Dall'istante di apertura della connessione, l'ack number comincia a crescere. Quando si dà il comando per terminare la stampa su schermo (all'istante t≈18s), la pendenza della retta aumenta considerevolmente poiché i pacchetti vengono trasmessi con frequenza molto più alta.

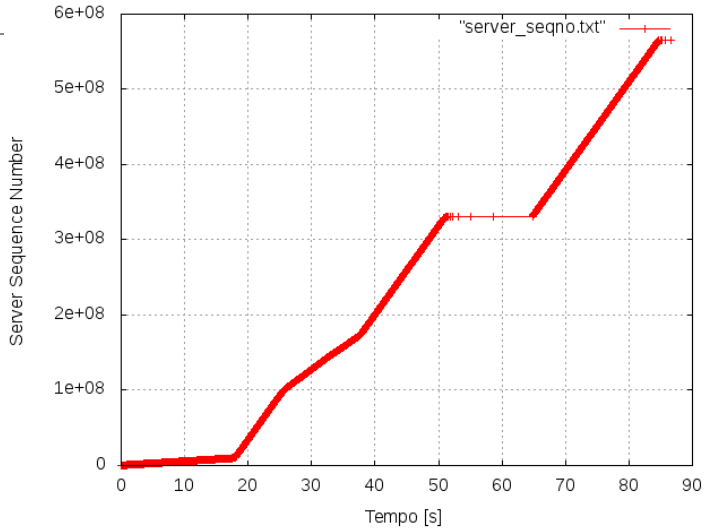
Successivamente, all'apertura di una seconda connessione, la velocità di trasmissione dei pacchetti diminuisce e



dunque la pendenza diminuisce, per poi ritornare al valore precedente nel momento in cui si chiude la seconda connessione.
Quando si apre la Telnet command mode, la trasmissione di pacchetti viene interrotta, vi sono solo più comunicazioni TCP ZeroWindow a intervalli regolari (con relativa risposta TCP Keep-Alive), e l'ack number si assesta a un valore pressoché costante. Alla chiusura della Telnet command mode, l'ack number torna a crescere e lo fa fino alla chiusura della connessione.

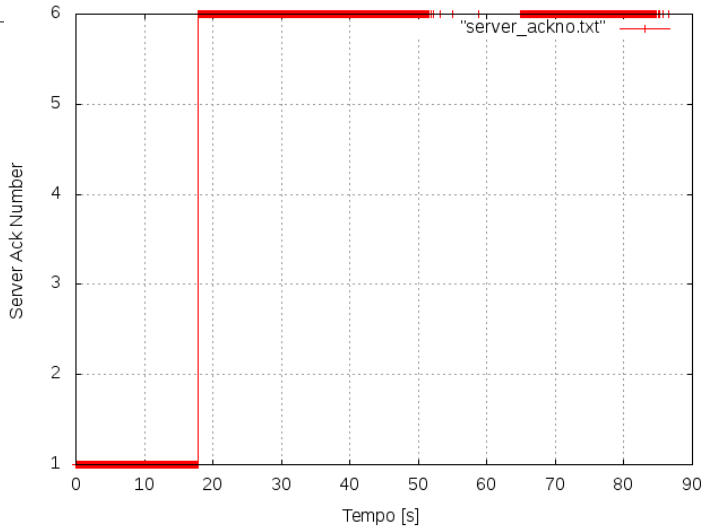
3) Server sequence number

L'avanzamento del sequence number del server è del tutto analogo a quello dell'ack number del client



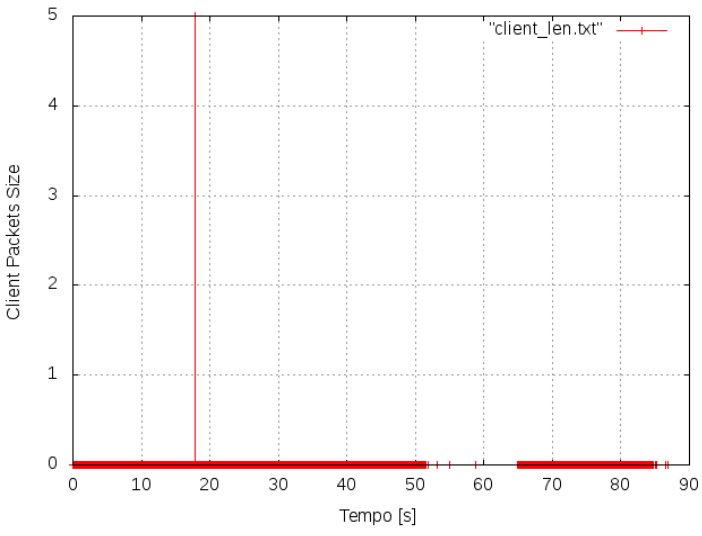
4) Server Ack Number

L'avanzamento del server ack number è identico a quello del client sequence number a differenza dell'ultimo pacchetto (quello di chiusura).



5) Client Packet Size

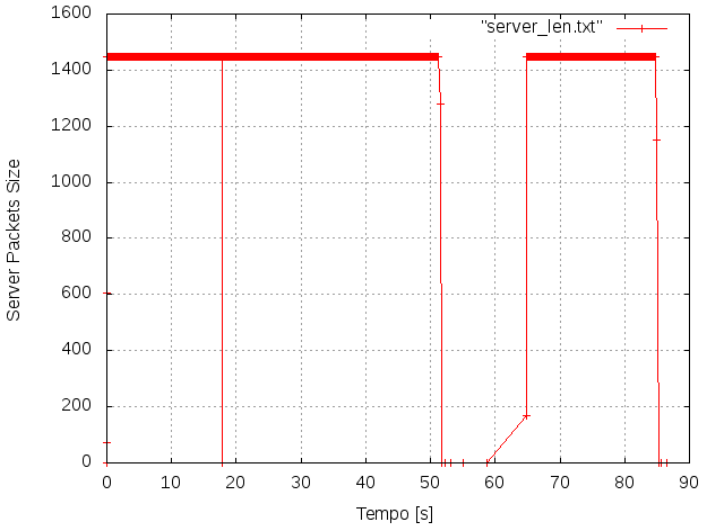
Lungo l'arco di tempo in cui è attiva la connessione, il client trasmette solo ACK, che sono pacchetti a dimensione nulla. Fa eccezione la PSH ACK (di lunghezza 5 byte) trasmessa dal client al server quando si decide di interrompere la stampa dei pacchetti ricevuti dal server.



6) Server Packet Size

Dall'istante in cui si apre la connessione il server incomincia a mandare pacchetti di lunghezza pari alla Maximum segment size (1448 bytes).

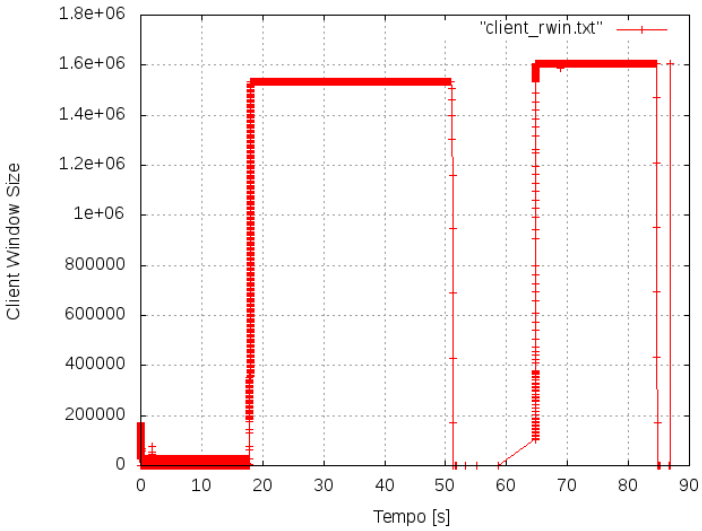
All'apertura della command mode, il server cessa la trasmissione di pacchetti, per poi riprenderla alla chiusura della stessa, fino al termine della connessione.



7) Finestra di ricezione del client

All'apertura della connessione la finestra del client si apre ad una certa ampiezza. Essa però si restringe immediatamente, in quanto il client, dovendo anche stampare i pacchetti in arrivo, non riesce a smaltire i dati ricevuti dal server.

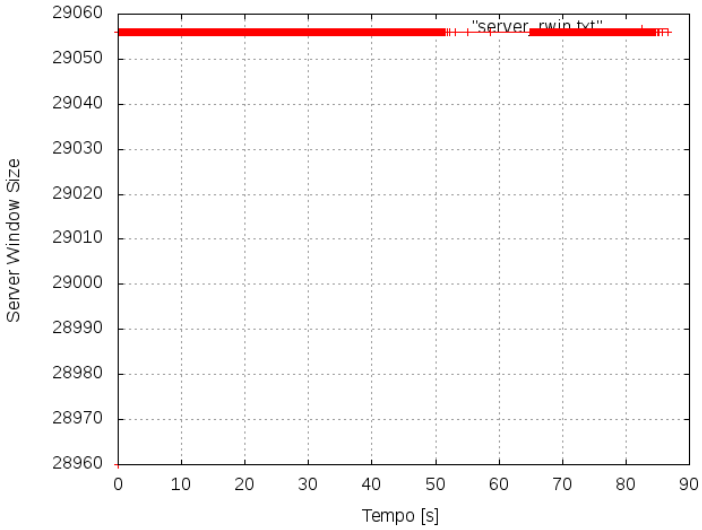
Successivamente la finestra va incontro a una nuova espansione nel momento in cui si ferma la stampa dei pacchetti.



Quando si apre la command mode il client non è più disposto a ricevere dei pacchetti e la finestra si chiude, per poi riaprirsi nel momento in cui si chiude la command mode. Per effettuare la chiusura della connessione, si entra nuovamente nella command mode e la finestra si azzera. L'ultima apertura della stessa è istantanea, ed è contestuale alla ricezione dell'ACK di chiusura mandata dal server.

8) Finestra di ricezione del server

La finestra di ricezione del server rimane costante durante tutta la cattura in quanto non ha bisogno né di ridurla né di aumentarla perchè non ha nessun problema (in capacità di calcolo) di smaltire i pacchetti di ACK.



Appendice

processa_file.sh

```
#!/bin/bash

sed -i "s/^/ /g" client_to_server.txt
sed -i "s/^/ /g" server_to_client.txt

cat client_to_server.txt | grep -v Source | tr -s ' ' | cut -d' ' -
f 3 > client.time.txt
cat client_to_server.txt | grep -v Source | cut -d'=' -f 2 | cut -
d' ' -f 1 > client.seqno.txt
cat client_to_server.txt | grep -v Source | cut -d'=' -f 3 | cut -
d' ' -f 1 > client.ackno.txt.tmp
cat client_to_server.txt | grep -v Source | cut -d'=' -f 4 | cut -
d' ' -f 1 > client.rwin.txt.tmp
cat client_to_server.txt | grep -v Source | cut -d'=' -f 5 | cut -
d' ' -f 1 > client.len.txt.tmp


cat server_to_client.txt | grep -v Source | tr -s ' ' | cut -d' ' -
f 3 > server.time.txt
cat server_to_client.txt | grep -v Source | cut -d'=' -f 2 | cut -
d' ' -f 1 > server.seqno.txt
cat server_to_client.txt | grep -v Source | cut -d'=' -f 3 | cut -
d' ' -f 1 > server.ackno.txt
cat server_to_client.txt | grep -v Source | cut -d'=' -f 4 | cut -
d' ' -f 1 > server.rwin.txt
cat server_to_client.txt | grep -v Source | cut -d'=' -f 5 | cut -
d' ' -f 1 > server.len.txt
```

```
# al primo pacchetto manca l'ack e quindi mi prende la lunghezza
sbagliata
echo 0 > client.len.txt
tail client.len.txt.tmp -n +2 >> client.len.txt

echo 0 > client.ackno.txt
tail client.ackno.txt.tmp -n +2 >> client.ackno.txt

echo 0 > client.rwin.txt.tmp
tail client.rwin.txt.tmp -n +2 >> client.rwin.txt

rm -rf *.tmp


paste client.time.txt client.seqno.txt > client_seqno.txt
paste client.time.txt client.ackno.txt > client_ackno.txt
paste client.time.txt client.rwin.txt > client_rwin.txt
paste client.time.txt client.len.txt > client_len.txt


paste server.time.txt server.seqno.txt > server_seqno.txt
paste server.time.txt server.ackno.txt > server_ackno.txt
paste server.time.txt server.rwin.txt > server_rwin.txt
paste server.time.txt server.len.txt > server_len.txt


gnuplot client_seqno.plt
gnuplot client_ackno.plt
gnuplot client_rwin.plt
gnuplot client_len.plt


gnuplot server_seqno.plt
gnuplot server_ackno.plt
gnuplot server_rwin.plt
gnuplot server_len.plt
```

---

client_seqno.plt

```
set xlabel "Tempo [s]"
set ylabel "Client Sequence Number"
set grid
set terminal png
set output "client_seqno.png"
plot "client_seqno.txt" with linespoint
```

Tutti gli altri file `plt` sono del tutto analoghi, abbiamo solamente cambiato la ylabel e i file di I/O

Alessandro Olivero *Nunzio Onorati* *Leonardo Pio Petrillo*

01QZD

Laboratorio di Internet e Comunicazioni

Internet - Laboratorio 4

Port Scan - Using nmap

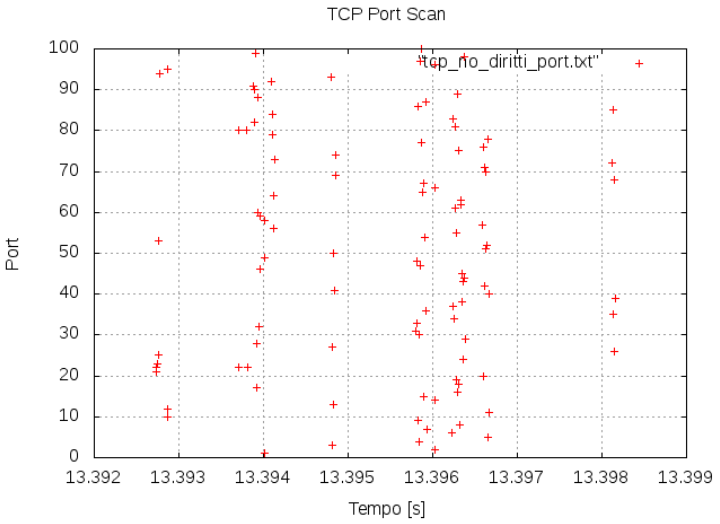
Abbiamo eseguito una Port Scan sulle prime 100 porte di un host in rete per vedere quali servizi erano attivi su quell'host

La Port Scan è stata eseguita in 4 modalità:
- TCP senza diritti di root
- UDP senza diritti di root
- TCP con diritti di root
- UDP con diritti di root

TCP senza diritti di root

```
nmap 192.168.2.15 -p 1-100 -v
```

Nmap prova ad effettuare una risoluzione DNS del client che dopo 13s di timeout fallisce.
A questo punto esegue la Port Scan che completa in pochi millisecondi.



Ogni porta viene contattata una sola volta e, sfruttando la connessione TCP capisce quali servizi sono attivi sull'host target.

Se sulla porta non sono presenti servizi attivi la conversazione consiste in due pacchetti: un [SYN] mandato dall'host locale e un [RST, ACK] mandato dall'host target.

Se sulla porta vi sono servizi attivi la connessione viene stabilita. In questo modo nmap sa che c'è un'applicazione pronta ad ascoltare e quindi può chiudere la connessione.

Si può notare che ogni connessione viene aperta da una porta differente perchè tante connessioni attivate dalla stessa porta potrebbero essere sospette.

UDP senza diritti di root

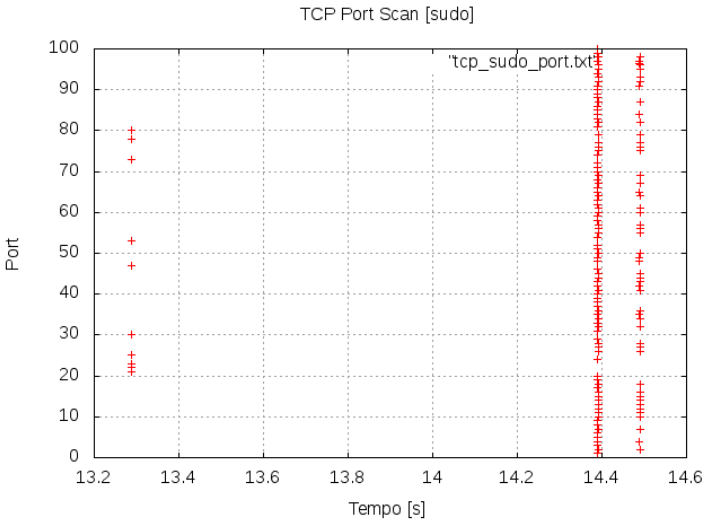
```
nmap 192.168.2.15 -p 1-100 -v -sU
```

Nmap ci avverte che per eseguire questa scan ha bisogno dei diritti di root, senza i quali non può osservare i pacchetti ICMP di ritorno.

TCP con diritti di root

```
sudo nmap 192.168.2.15 -p 1-100 -v
```

Come nel caso senza diritti di root, nmap prova una risoluzione DNS che fallisce dopo 13s.
A questo punto esegue la Port Scan.



Ogni porta viene contattata 1 o 2 volte.

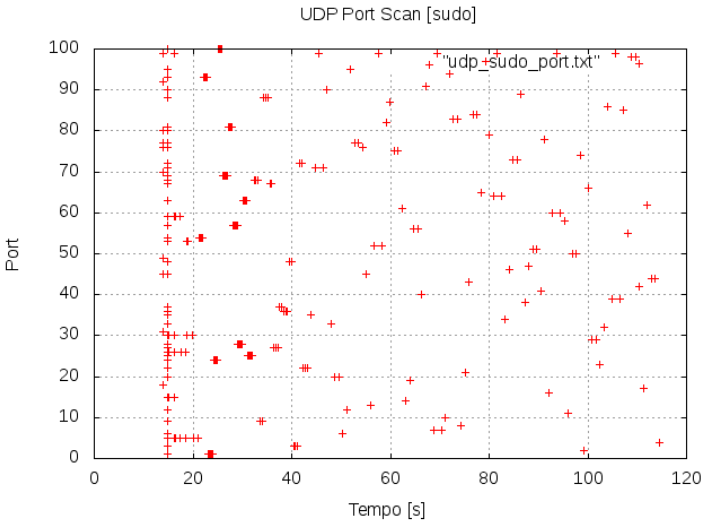
Se sulla porta non sono presenti servizi attivi, la conversazione si svolge esattamente come nel caso del TCP senza i diritti di root: un [SYN] mandato dall'host locale e un [RST, ACK] mandato dall'host target.

Invece se sulla porta vi sono servizi attivi, potendo intervenire su livelli più bassi rispetto al livello 4 del TCP, la conversazione è più breve e si compone di: [SYN], [SYN, ACK], [RST].
In questo modo la connessione non viene mai aperta, sono richiesti meno pacchetti per ottenere le informazioni e soprattutto vi sono meno probabilità che l'host target segnali la conversazione come sospetta in quanto non è mai stata aperta una connessione vera e propria. Ogni SYN viene sempre inviato dalla stessa porta.

Nmap fornisce inoltre l'indirizzo MAC dell'host target.

UDP con diritti di root

```
sudo nmap 192.168.2.15 -p 1-100 -v -sU
```



Se sulla porta sotto esame vi è un servizio attivo nmap presume che questo servizio, quando interpellato, risponda qualcosa.
Se non ottiene nessuna risposta non è possibile concludere nulla in quanto può essere andato perso il pacchetto UDP, può essere stato bloccato da un firewall, l'applicazione può aver deciso di non rispondere, è andato perso il pacchetto ICMP di risposta, ecc…

Se riceve un pacchetto ICMP Destination Unreachable (Port unreachable) allora significa che su quella porta non vi è nessuna applicazione UDP in ascolto.

Le porte vengono contattate più volte (entro un numero massimo) finché non viene ottenuta una risposta. Questa è una grande differenza rispetto alla scan effettuata tramite TCP (protocollo affidabile) rispetto ad UDP (protocollo non affidabile).
La velocità di questa scan è molto bassa in quanto l'host target invia solamente un pacchetto ICMP al secondo circa. Nmap capendo questo meccanismo rallenta l'invio dei pacchetti.

Considerazioni

Con qualsiasi protocollo si effettui la Port Scan l'ordine delle porte è casuale in quanto l'host target non deve capire di esserne bersaglio.

L'operazione effettuata tramite TCP è molto più veloce rispetto a quella effettuata tramite UDP in quanto si sfrutta la caratteristica del TCP di essere un protocollo affidabile.

Se il TCP viene utilizzato con i diritti di root si aumenta l'efficienza e soprattutto la capacità di rimanere "nascosto" di nmap.
La connessione non stabilita non viene registrata a differenza di una connessione stabilita e chiusa senza effettivo passaggio di dati.

Alessandro Olivero *Nunzio Onorati* *Leonardo Pio Petrillo*

01QZD

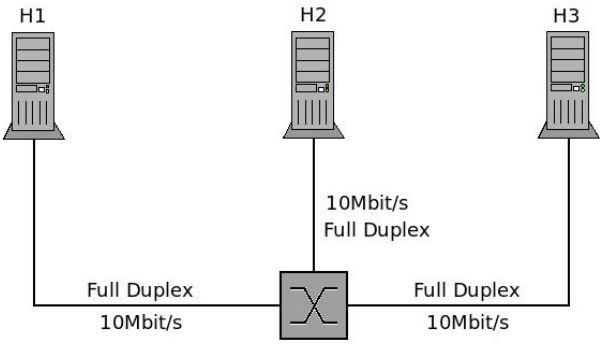
Laboratorio di Internet e Comunicazioni

Internet - Laboratorio 6

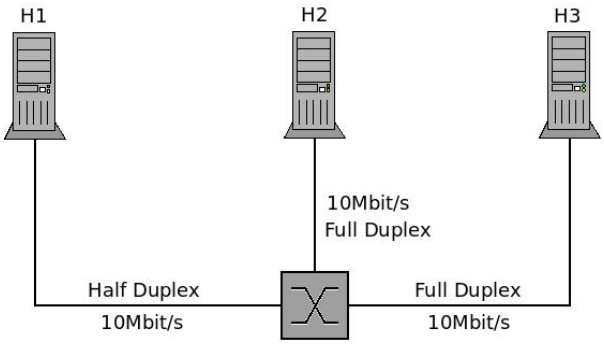
Performance test on Ethernet links

Configurazione di rete

Tipo A



Tipo B



A - Rete tipo A

Protocollo	AVG TCP goodput		Collision probability		Loss at the application Layer	
	Pred	Obs	Pred	Obs	Pred	Obs
TCP	9,500	9,441	0%	0%	0%	0%
UDP	9,600	9,572	0%	0%	0%	0%

La rete è occupata solamente dalla comunicazione tra H1 e H2 che quindi raggiunge il massimo goodput consentito dal protocollo (circa 9,5 per il TCP e circa 9,6 per l'UDP).

Essendo tutti i canali in full duplex non vi possono essere collisioni.

Non vi è perdita a livello applicazione in quanto non vi sono pacchetti persi nelle code dello switch in quanto riceve a 10Mbit/s e trasmette alla stessa velocità, le code non possono dunque riempirsi.

B - Rete tipo B

Protocollo	AVG TCP goodput		Collision probability		Loss at the application Layer	
	Pred	Obs	Pred	Obs	Pred	Obs
TCP	< 9,5 (dovuto a trasmissione degli ack)	8,236	Bassa ma presente (sempre dovuta agli ack)	19,723%	0%	0%
UDP	9,600	9,571	0%	0%	0%	0%

Per quanto riguarda la comunicazione attraverso UDP non cambia nulla dal punto A in quanto non viene trasmesso nulla in senso contrario e quindi il canale in half duplex non ha effetti (non vi sono collisioni).

Invece per la connessione TCP vi sono dei cambiamenti: gli ACK devo essere mandati al trasmettitore sul canale half duplex causando possibile collisione con i pacchetti in arrivo.
Il goodput è quindi minore rispetto a quello permesso dal protocollo in condizioni ottimali.

Non vi sono comunque perdite a livello applicazione perchè il protocollo TCP prevede ritrasmissione dei pacchetti in caso di perdite.

C - Rete tipo A

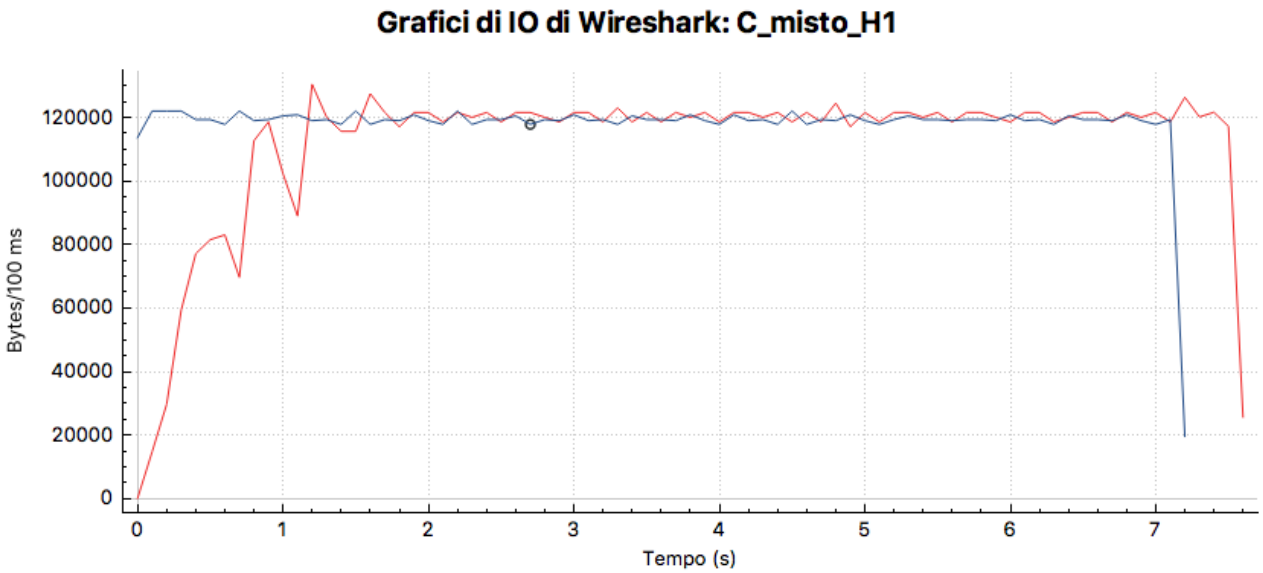
Protocollo	AVG TCP goodput		Collision probability		Loss at the application Layer	
	Pred	Obs	Pred	Obs	Pred	Obs
TCP - H1	< 9,5 (ci sono sia i pacchetti che gli ack su ogni canale)	9,143	0%	0%	0%	0%
TCP - H2		9,180	0%	0%	0%	0%
UDP - H1	9,600	9,572	0%	0%	0%	0%
UDP - H2	9,600	9,571	0%	0%	0%	0%

Nella connessione TCP il goodput è minore di 9,5 perchè su entrambi i canali passano anche gli ACK dell'altra comunicazione occupando parte del canale. Cosa che non succede nella comunicazione UDP dove i due flussi non si disturbano a vicenda (non vi sono gli ACK).

Essendo la rete interamente in full duplex non ci possono essere collisioni.

Inoltre non ci sono perdite a livello applicazione perchè le velocità di trasmissione non permettono alle code dello switch di riempirsi.

AVG TCP goodput		AVG UDP goodput	
Pred	**Obs**	**Pred**	**Obs**
9,500	8,905	< 9,6 (canale condiviso tra UDP e ack del TCP)	9,307

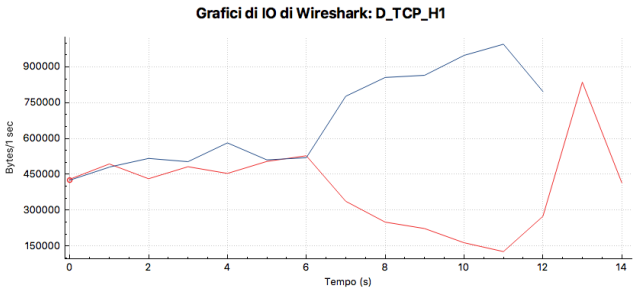


Nel grafico sono mostrati in rosso i bytes trasmessi dalla connessione TCP, in blu quelli trasmessi in UDP.

Il flusso UDP non riesce a raggiungere i 9,6Mbit/s perchè condivide il canale con gli ACK della connessione TCP.

D - Rete tipo B

Protocollo	AVG TCP goodput		Collision probability		Loss at the application Layer	
	Pred	**Obs**	**Pred**	**Obs**	**Pred**	**Obs**
TCP - H1	< 9,5/2 perchè il canale tra H1 e lo switch è condiviso dalle due comunicazioni + gli ack	3,848	Possibile la collisione	33,992%	0%	0%
TCP - H2		5,206	0%	0%	0%	0%
UDP - H1	circa 9,6 / 2	4,536	Possibile la collisione	9,483%	alta probabilità di perdita	24,268%
UDP - H2	circa 9,6 / 2	3,595	0%	0%	alta probabilità di perdita	24,658%

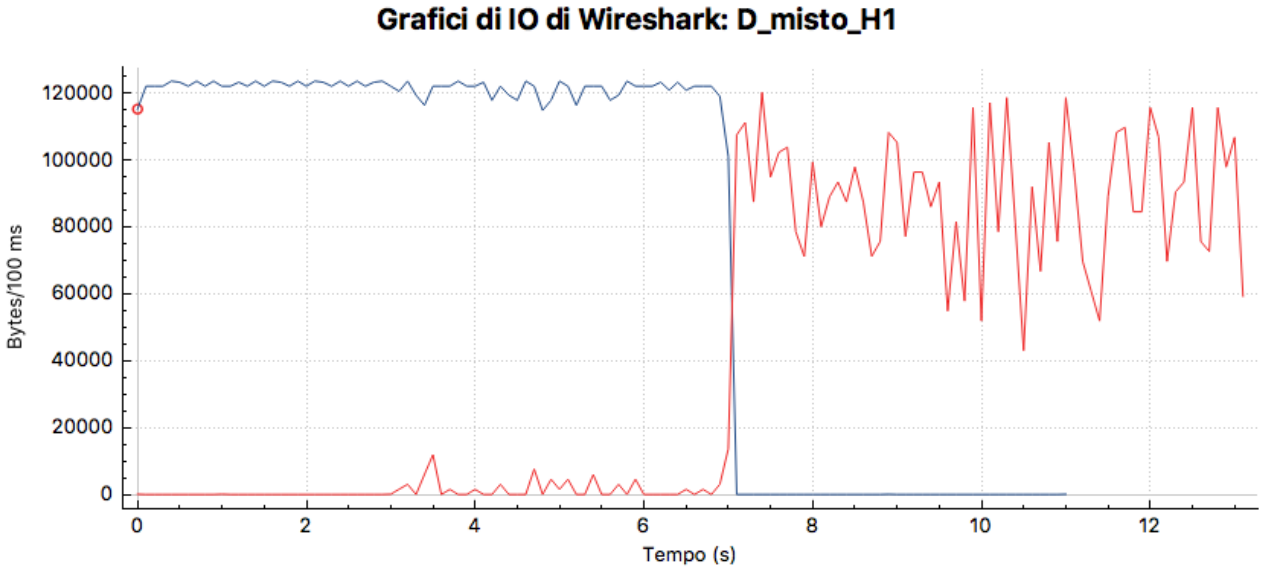


Le due connessioni TCP dovrebbero condividere in maniera uguale il canale tra H1 e lo switch, mentre non si disturbano (tranne che per gli ACK) tra lo switch e H2.

Come si vede dal grafico ad un certo punto (t ≈ 6s) una delle due connessioni prende il sopravvento sull'alta creando la differenza di goodput osservato.

Sia in TCP che in UDP è possibile la collisione tra H1 e lo switch (canale HD), in UDP questo ha effetto sulle perdite a livello applicazione (che sono molto elevate).
Va considerato che i pacchetti UDP sono frazionati in più pacchetti IP e, quindi, anche se uno solo dei pacchetti IP che lo compongono va perso, il pacchetto UDP non giunge correttamente a destinazione.
A causa delle ritrasmissioni dovute alle collisioni le code dello switch si riempiono e molti pacchetti vengono scartati.

AVG TCP goodput		**AVG UDP goodput**		
Pred	**Obs**	**Pred**	**Obs**	
< 9,5 perchè ridotto dalle collisioni con i pacchetti UDP inviati dall'altro host e con gli ack	5,160	< 9,6 perché il canale è condiviso con gli ack	9,483	Il canale HD è dalla parte del TCP, il FD dalla parte dell'UDP



La connessione UDP (in blu) riesce a fermare quasi del tutto quella TCP (in rosso) che può completarsi solo dopo l'istante in cui il flusso UDP cessa.
Questo comportamento spiega la grande diversità di goodput osservato: quello dell'UDP è molto vicino a quello ideale, mentre quello TCP è poco più della metà (per 7 secondi circa la velocità è prossima allo 0, mentre per i 7 secondi successivi è prossima a quella ideale).

E - Rete tipo A

Protocollo	AVG TCP goodput		Collision probability		Loss at the application Layer	
	Pred	**Obs**	**Pred**	**Obs**	**Pred**	**Obs**
TCP - H1	9,5 / 2 perchè il canale tra H2 e lo switch è condiviso tra le due comunicazioni	4,707	0%	0%	0%	0%
TCP - H2		4,754	0%	0%	0%	0%
UDP - H1	molto < (9,6 / 2) perchè il canale è condiviso dai due flussi	0,331	0%	0%	Alta probabilità di perdita	95,605%
UDP - H2		0,589	0%	0%	Alta probabilità di perdita	92,139%

Essendo l'intera rete in full duplex non è possibile osservare collisioni.
Il collo di bottiglia in questo esperimento è il canale tra lo switch ed H3.

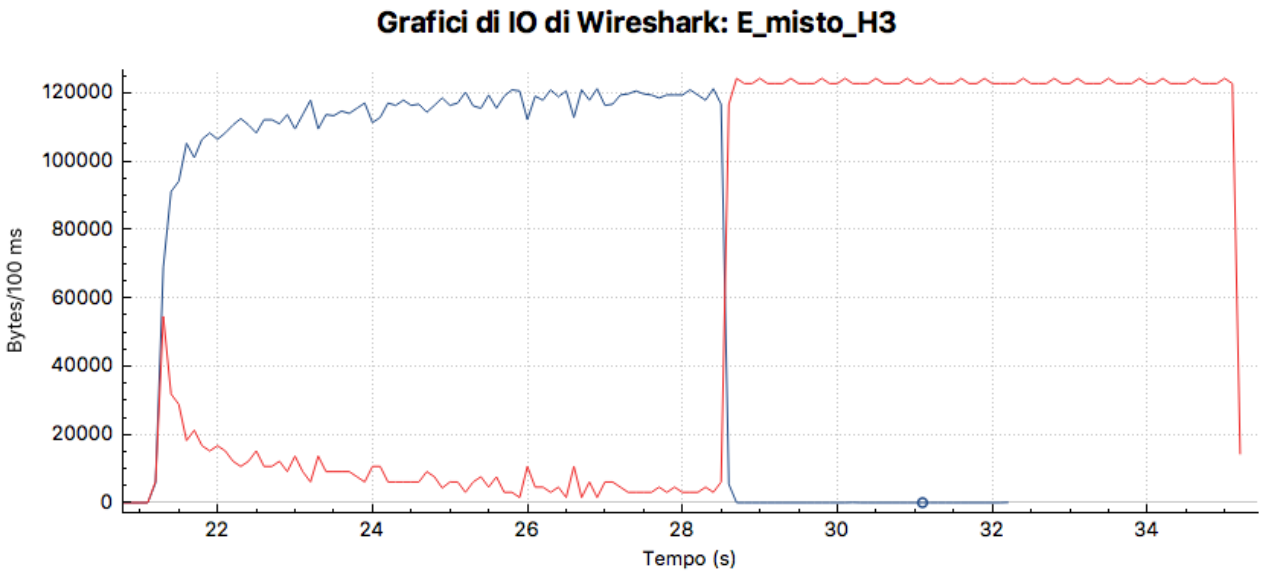
Per quanto riguarda le comunicazioni in TCP queste si dividono equamente il canale e non si osservano (ovviamente) perdite a livello applicazione.

Invece in UDP il comportamento è molto diverso. I due flussi UDP che si spostano indisturbati tra H1 e lo switch e tra H2 e lo switch riempiono le code di quest'ultimo causando la coda di molti pacchetti (circa la metà).

pacchetti ricevuti ≈ 20Mbit/s
pacchetti inviati ≈ 10Mbit/s

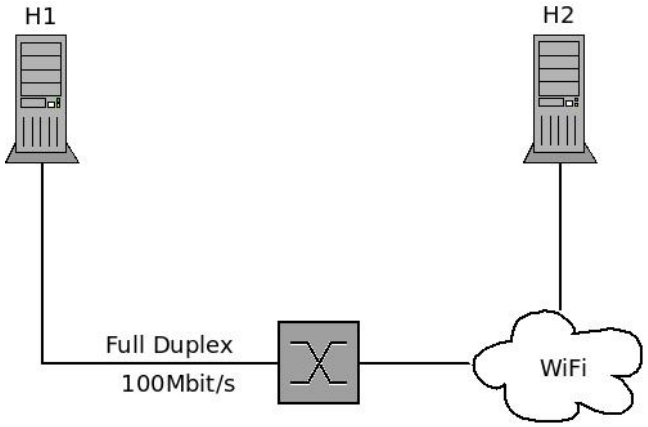
La perdita a livello applicazione è ancora più alta in quanto anche un solo pacchetto IP componente il pacchetto UDP perso causa la perdita dell'intero pacchetto.

AVG TCP goodput		AVG UDP goodput		
Pred	**Obs**	**Pred**	**Obs**	
circa 9,5 / 2 perchè durante la trasmissione dell'UDP la velocità è circa 0, quando UDP finisce trasmette a tutta velocità per un tempo circa uguale	4,820	< di 9,6 perchè parte del canale è comunque occupato dal flusso TCP	6,800	Il valore dell'UDP è basso in quanto il TCP regge abbastanza bene all'inizio



Nei primi istanti il flusso UDP e quello TCP condividono il canale ma molto presto quello UDP prende il sopravvento.
La velocità del flusso TCP è circa la metà di quella ideale (dovuta alla velocità prossima a zero nella prima parte e prossima all'idealità nella seconda).
Il flusso UDP invece agisce abbastanza indisturbato tranne che per i primi secondi.

Singolo flusso TCP su canale WiFi



Il canale WiFi è stato impostato per avere una velocità di trasferimento pari a 54 Mbit/s.
Per abbassare gli errori di misura, i dati trasmessi sono stati aumentati rispetto al valore di default a 20000 pacchetti da 1500 bytes.

La velocità effettiva calcolata grazie ad `nttcp` varia ad ogni prova con valori variabili tra 7.24 Mbit/s e 12.80 Mbit/s.

Questa varietà di risultati è dovuta alla natura del canale WiFi. Infatti il canale è in half duplex e quindi soggetto a collisioni. È inoltre soggetto a disturbi dovuti ad altre reti WiFi nelle vicinanze.

Anche in condizioni ottime il goodput è molto minore alla velocità indicata in quanto il protocollo WiFi prevede molta più intestazione ai pacchetti rispetto alla normale intestazione ethernet.