Progetto di Sistemi Virtuali: Architettura per comunicazioni multi-interfaccia con VDE 4

Stefano Traini

December 2017

1 Introduzione

Virtual Square è un progetto nato nel 2004 all'interno dell'Università di Bologna e ad oggi può essere definito come un laboratorio internazionale di progetti legati alla virtualità. Il suo nome, che in italiano viene tradotto come "Piazza Virtuale", racchiude la sua essenza, che è quella di mettere insieme diversi progetti virtuali unificando concetti e creando strumenti di interoperabilità.
Il suo logo, V^2, viene letto come Virtual Squared (Virtualità al quadrato) e racchiude un altro scopo del progetto Virtual Square, ossia quello di sfruttare la virtualità esistente per creare ulteriori servizi virtuali.
Uno dei progetti chiave che è possibile trovare all'interno di Virtual Square è *VDE* (Virtual Distributed Ethernet) che può essere definito come un "coltellino svizzero" di strumenti riguardanti il networking. Oggi giunto alla versione 4 è fruibile nel profilo GitHub del creatore e principale maintainer all'indirizzo github.com/rd235/vdeplug4, è composto da diversi strumenti virtuali di networking, come ad esempio switch e hub, che svolgono le stesse funzioni (e talvolta le ampliano) degli strumenti fisici da cui prendono il nome. Grazie a questo tool è possibile creare reti virtuali che svolgono le stesse funzioni delle reti tradizionali.
Un altro strumento utile contenuto in V^2 è *vdens*, disponibile sempre nel profilo GitHub del creatore di Virtual Square all'indirizzo github.com/rd235/vdens, che permette la creazione di User Namespaces connessi a reti VDE.
Questi descritti sono solo due dei numerosi tool messi a disposizione da Virtual Square, gli altri sono consultabili nella wiki del progetto all'indirizzzo wiki.v2.cs.unibo.it o nel profilo GitHub sopracitato github.com/rd235.
In questo progetto si sono utilizzati i tool virtuali messi a disposizione da Virtual Square per creare una rete VDE per far fronte ad una problematica presente soprattutto nei moderni dispositivi mobili: la perdita di pacchetti e la latenza presente nelle tradizionali comunicazioni wireless dovuti ai limiti fisici di tali tecnologie di comunicazione.
I moderni smartphone mettono a disposizione diverse interfacce di comunicazione e sono solitamente equipaggiati con un'interfaccia di rete **dati cellulare** e una scheda di rete **WiFi**. L'idea è quella di utilizzare **Multi**, uno dei dispositivi virtuali presente nella versione 4 di VDE, per permettere al dispositivo di scegliere ed utilizzare l'interfaccia che gli garantisca migliori prestazioni in termini di affidabilità e velocità di trasmissione oppure utilizzare entrambe le interfacce contemporaneamente.
Nella seconda sezione viene descritta l'architettura progettata per lo scopo prefissato, nella terza sezione vengono mostrati i test effettuati con i relativi risultati ottenuti e nell'ultima sezione vengono tratte alcune conclusioni e vengono forniti spunti per sviluppi futuri.

2 Architettura

Come è possibile vedere in Figura 1 l'architettura descritta in seguito non è stata realizzata su un dispositivo mobile ma è stata sviluppata su un laptop tradizionale con sistema operativo *Debian 9 Stretch* in vesrione *stable*, configurato in modo tale da svolgere le stesse funzioni di un moderno smartphone, equipaggiato con una scheda di rete WiFi integrata e una scheda di rete dati cellulare ottenuta collegando un dispositivo mobile *Android* in modalità *Tethering USB* che espone la propria intefaccia dati cellulare integrata al laptop a cui è connesso e che viene vista come una tradizionale interfaccia ethernet.

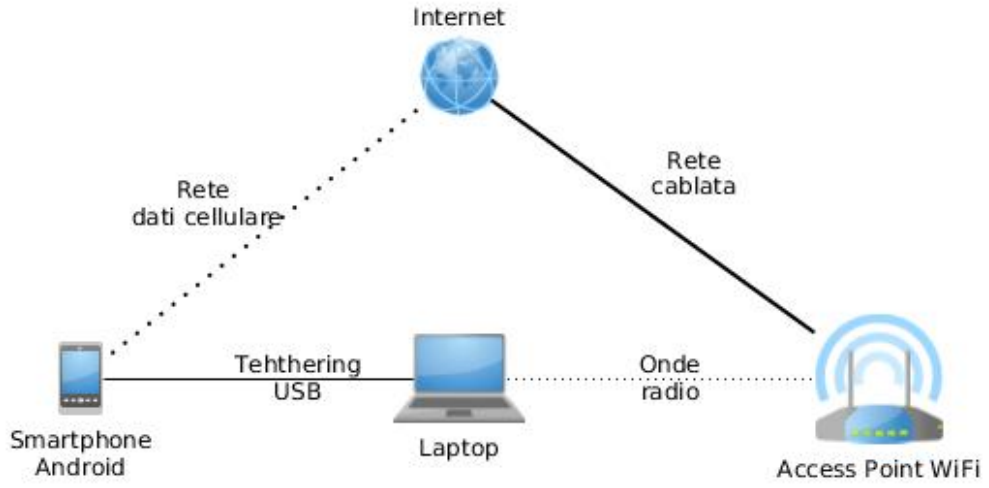

Figura 1: Configurazione delle interfacce.

Nella prima parte di questa sezione viene descritta l'architettura utilizzata per mostrare il funzionamento dei vari dispositivi virtuali che compongono l'architettura stessa, mentre nella seconda parte viene descritta quella che sarà l'architettura finale che potrà essere replicata su un tradizionale dispositivo mobile.
In questa sezione vengono volutamente mescolate parti descrittive e parti elencate in stile tutorial al fine di poter permettere al lettore di ricostruire l'intera architettura e di verificarne il corretto funzionamento.

2.1 Pre-requisiti e dipendenze

Per poter realizzare le seguenti architetture è necessario che la macchina su cui si intende utilizzare la medesima sia abilitata alla virtualizzazione (maggiori informazioni sono reperibili all'indirizzo http://wiki.v2.cs.unibo.it) e successivamente installare i tool di Virtual Square, precedentemente descritti, VDE versione 4 e vdens. Per poter procedere all'installazione è utile creare una cartella contenente i codici sorgente dei tool, scaricarli dal repository GitHub attraverso il programma Git e installarli. Tutti i comandi descritti vengono eseguiti sulla distribuzione Debian 9, ma sono disponibili (con le dovute modifiche) su qualsiasi distribuzione GNU/Linux con versione del kernel superiore alla 3.8 che supporta gli User Namespaces. Inoltre sono necessari, per poter eseguire alcuni comandi, i diritti di esecuzione da Super User. La notazione dei comandi inseriti rispecchia quella utilizzata

all'interno dei sistemi operativi GNU/Linux: il carattere "$" specifica che il comando può essere utilizzato da un normale utente, mentre il carattere "#" simboleggia che il comando debba essere eseguito da un utente con privilegi di SuperUser.

```
# apt-get update
# apt-get install git-core build-essential
$ git config --global user.name "Your name"
$ git config --global user.email yourEmail@example.com
$ mkdir virtualsquare
$cd /virtualsquare
virtualsquare$ git clone https://github.com/rd235/s2argv-execs
virtualsqaure/s2argv-execs$ autoreconf -if
virtualsqaure/s2argv-execs$ ./configure
virtualsqaure/s2argv-execs$ make
virtualsqaure/s2argv-execs# make install
virtualsqaure/s2argv-execs$ cd ..
virtualsquare$ git clone https://github.com/rd235/vdeplug4
virtualsquare$ cd /vdeplug4
virtualsqaure/vdeplug4$ autoreconf -if
virtualsqaure/vdeplug4$ ./configure
virtualsqaure/vdeplug4$ make
virtualsqaure/vdeplug4# make install
virtualsqaure/vdeplug4$ cd ..
virtualsquare$ git clone https://github.com/rd235/vdens
virtualsquare$ cd /vdens
virtualsqaure/vdens$ autoreconf -if
virtualsqaure/vdens$ ./configure
virtualsqaure/vdens$ make
virtualsqaure/vdens# make install
virtualsqaure/vdens# echo 1 > /proc/sys/kernel/unprivileged_userns_clone
$export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

Ulteriori dipendenze potrebbero essere richieste, per eventuali problematiche riferirsi alla documentazione ufficiale di Virtual Square http://wiki.v2.cs.unibo.it o al profilo ufficiale di GitHub del principale maintainer github.com/rd235

2.2 Architettura per i test

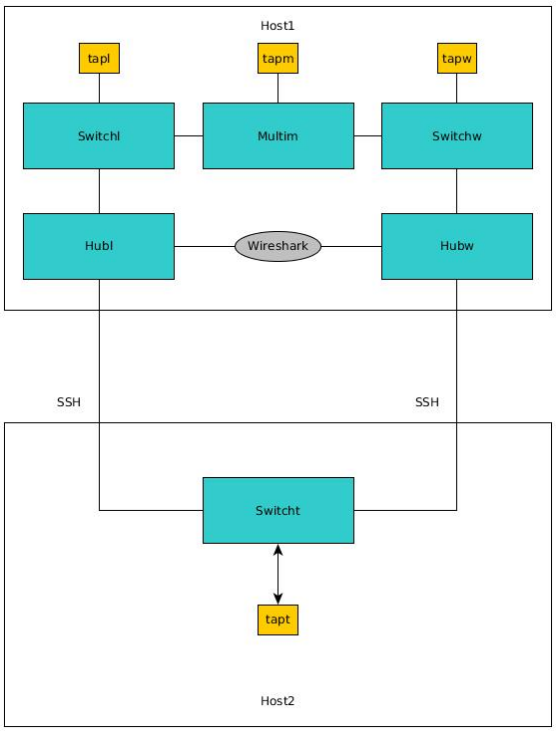

Figura 2: Architettura per i test.

Come è possibile vedere in Figura 2 l'architettura realizzata comprende l'utilizzo di queste componenti:

- 2 host;
- 4 interfacce tap;
- 3 VDE switch;
- 2 VDE hub;
- Wireshark;
- 1 VDE multi.

La descrizione ed il funzionamento dell'architettura e di tutte le sue componenti avvengono nei paragrafi seguenti.

2.2.1 Configurazione iniziale

Quando su un dispositivo tradizionale vengono collegate più interfacce di comunicazione esso agisce con una mutua esclusione su di esse, ossia preferisce un'interfaccia rispetto alle altre che viene definita come *default gateway*. La comunicazione può quindi entrare da qualsiasi interfaccia collegata ma

uscirà verso l'esterno sempre dall'interfaccia definita come predefinita. Dal momento che lo scopo di questo progetto è quello di utilizzare due interfacce distinte con traffico distinto è opportuno che ognuna di esse sia capace di occuparsi sia del traffico in entrata sia di quello in uscita, perciò bisogna creare due tabelle di indirizzamento, una per la rete dati cellulare (che, dal momento che per gli esperimenti si è utilizzata un'interfaccia LTE, da qui in poi sarà abbreviata con la sigla **LTE**) e l'altra per la rete WiFi.

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
$ echo 1 LTE >> /etc/iproute2/rt_tables
$ echo 2 WIFI >> /etc/iproute2/rt_tables

$ sudo ip route add default via <lte interface gateway>
    dev <lte interface name> table LTE
$ sudo ip rule add from <lte interface destination>/<mask>
    table LTE

$ sudo ip route add default via <wifi interface gateway>
    dev <wifi interface name> table WIFI
$ sudo ip rule add from <wifi interface destination>/<mask>
    table WIFI
```

2.2.2 Host

L'Host 1 rappresenta il dispositivo mobile che necessita l'utilizzo di due interfacce che, come detto in precedenza, per i test è stato utilizzato un laptop equipaggiato con una interfaccia WiFi e una interfaccia LTE ottenuta collegando un dispositivo Android in modalità *tethering USB*.
L'Host2 rappresenta l'end-point con il quale si intende comunicare e nella sessione di testing è stata utilizzata una macchina virtuale messa a disposizione dal dipartimento di Informatica dell'Università di Bologna. Su entrambi gli host sono stati installati i tool VDE e vdens precedentemente descritti perché, come mostrato in Figura 2, la creazione di dispositivi virtuali avviene su entrambe le macchine.

2.2.3 Interfacce tap

Le interfacce **tap** sono delle interfacce virtuali che simulano il funzionamento di una interfaccia ethernet. Nell'architettura vengono utilizzate 4 tap.
L'interfaccia **tapl** (tap lte), con indirizzo 10.0.3.3/24, viene utilizzata per creare un collegamento tra lo *switchl* e la macchina fisica e proprio da questa interfaccia partiranno alcune comunicazioni riguardanti la rete LTE che verranno spiegate in seguito.
Analogamente alla precedente, **tapw** (tap WiFi), con indirizzo 10.0.3.4/24, collega l'Host 1 allo *switchw* e si occupa di comunicazioni riguardanti la rete WiFi.
L'interfaccia **tapm** (tap Multi), con indirizzo 10.0.3.2/24, si occupa di collegare il dispositivo *multim* all'Host 1. Da questa interfaccia partiranno e arriveranno i pacchetti della reale comunicazione.
La creazione delle tre interfacce presenti sull'Host 1 e l'assegnazione dell'indirizzo ip avvengono attraverso l'utilizzo dei comandi:

```
# ip tuntap add dev tapm mode tap user ${USER}
# ip tuntap add dev tapl mode tap user ${USER}
# ip tuntap add dev tapw mode tap user ${USER}
```

```
# ip addr add 10.0.3.2/24 dev tapm
# ip addr add 10.0.3.3/24 dev tapl
# ip addr add 10.0.3.4/24 dev tapw

# ip link set tapm up
# ip link set tapl up
# ip link set tapw up
```

La quarta interfaccia **tapt**, con indirizzo 10.0.3.1, viene creata sull'host remoto per collegare lo *switcht* alla macchina fisica. La creazione di questa interfaccia avviene da remoto collegandosi in *ssh* e digitando i comandi analoghi ai precedenti:

```
$ ssh user@remote.host
remote.host# ip tuntap add dev tapt mode tap user ${USER}
remote.host# ip addr add 10.0.3.2/24 dev tapt
remote.host# ip link set tapt up
```

Come è possibile vedere dai comandi è opportuno possedere i diritti di SuperUser per poter creare l'interfaccia tap, se non si dovessero possedere questi diritti sull'host remoto si consiglia di contattare l'amministratore.

2.2.4 VDE multi

VDE multi è un dispositivo virtuale che simula delle socket multiple, ossia i pacchetti in ingresso vengono duplicati e spediti su tutte le interfacce di rete che si collegano con la destinazione. In questa architettura il dispositivo VDE multi viene collegato alla rispettiva interfaccia tap e ai VDE switch dell'Host 1.

2.2.5 VDE switch

I VDE switch hanno il medesimo funzionamento degli switch tradizionali, si occupano dell'instradamento dei pacchetti indirizzandoli verso la destinazione giusta. In questa architettura gli switch nell'Host 1 vengono creati e vengono collegati alle rispettive tap, a multim e ai rispettivi hub.
Lo switch dell'Host 2 invece viene collegato agli hub attraverso un collegamento *ssh* proveniente dalle due interfacce utilizzate nell'Host 1.

2.2.6 VDE hub

I VDE hub replicano il funzionamento degli hub tradizionali nei quali un pacchetto viene spedito in multicast a tutti i dispositivi connessi ad eccezione del mittente. In questa architettura i VDE hub vengono collegati ai rispettivi VDE switch e a **Wireshark**, un software utilizzato per monitorare il traffico di rete.

2.3 Comandi per creare l'architettura

Una volta spiegato il funzionamento di ogni dispositivo è arrivato il momento di mostrare i comandi che permettono la creazione dell'architettura.
Prima di iniziare è opportuno specificare il funzionamento di tre comandi: *"cmd://"*, *"vdens -m"* e *"vde_plug"*.

Il primo permette il lancio di comandi VDE su un host remoto, il secondo abilita un namespace in modalità *multi*, ossia permette di collegare più interfacce al namespace, ed il terzo permette di collegare due dispotivi VDE oppure un dispositivo VDE con un'interfaccia tap. Il primo comando sarà utilizzato per creare un collegamento ssh utilizzando un'interfaccia fisica (attraverso l'attributo *bind*) e usare vde_plug per collegare gli hub e lo switch remoto. Il secondo comando serve per creare un namespace connesso con gli hub e poi avviare Wireshark per monitorare il traffico.
I comandi scritti di seguito dovranno essere eseguiti su diversi terminali poiché quando si crea un dispositivo VDE non è possibile digitare dei comandi sotto di esso, qindi si prega di fare attenzione quando viene specificato il terminale prima di ogni comando.

```
term0$ vde_plug tap://tapl switch:///tmp/switchl

term1$ vde_plug tap://tapw switch:///tmp/switchw

term2$ vde_plug tap://tapm multi:///tmp/multim

term3$ ssh user@remote.host
term3$ vde_plug tap://tapt switch:///tmp/switcht

term4$ vde_plug hub:///tmp/hubl vde:///tmp/switchl

term5$ vde_plug hub:///tmp/hubw vde:///tmp/switchw

term6$ vde_plug vde:///tmp/multim vde:///tmp/switchl

term7$ vde_plug vde:///tmp/multim vde:///tmp/switchw

term8$ vde_plug vde:///tmp/hubl cmd://'ssh
    -b <LTE interface ip address> user@remote.host
    vde_plug vde:///tmp/switcht'

term9$ vde_plug vde:///tmp/hubw cmd://'ssh
    -b <WiFi interface ip address> user@remote.host
    vde_plug vde:///tmp/switcht'

termA$ vdens -m vde:///tmp/hubl vde:///tmp/hubm
termA$ ip link set vde0 up
termA$ ip link set vde1 up
termA$ wireshark
```

2.3.1 Funzionamento dell'architettura

Una volta spiegato come realizzare l'architettura per il testing è possibile descriverne il funzionamento. L'Host 1 attraverso le interfacce *tapl* e *tapw* è in grado di mandare pacchetti verso l'Host 2 per verificare la qualità del canale di comunicazione e quindi vedere quale dei due sia il più indicato per comunicare.

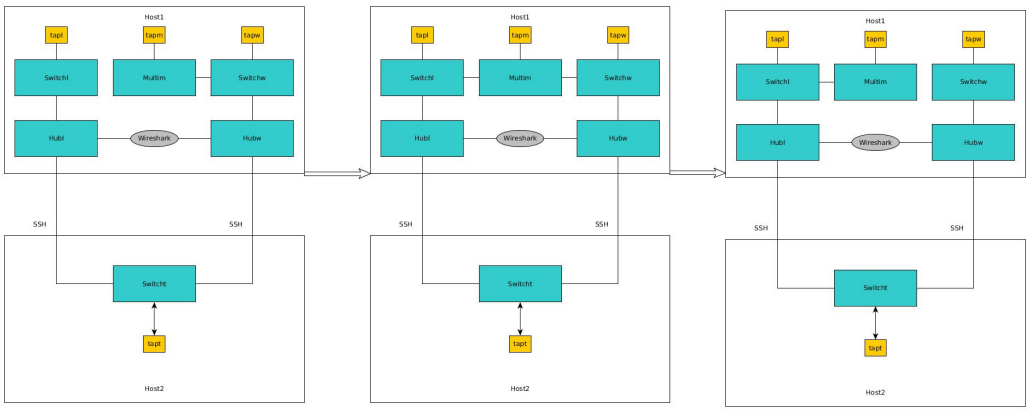

Figura 3: Funzionamento dell'architettura.

Come si mostrato nell'esempio di Figura2 , inizialmente il miglior canale di comunicazione è risultato essere quello WiFi, quindi i pacchetti inviati dalla *tapm* verso l'host remoto verranno inoltrati da *multim* verso *switchw*. Tuttavia, nel passo successivo dell'esempio il canale LTE risulta essere migliore del precedente, si collegano quindi *multim* e *switchl* con un vde_plug (term6). A questo punto *multim* riceverà i pacchetti dalla *tapm* e li inoltrerà verso *switchl* e *switchw*, gli Host riceveranno pacchetti duplicati poiché il dispositivo VDE multi si occupa di inviare i pacchetti su tutte le strade che raggiungono la destinazione. A questo punto, per far tornare la situazione alla normalità (non ricevere più duplicati), è sufficiente interrompere il processo che collega *multim* a *switchw* (term7) facendo cadere il collegamento tra di essi.

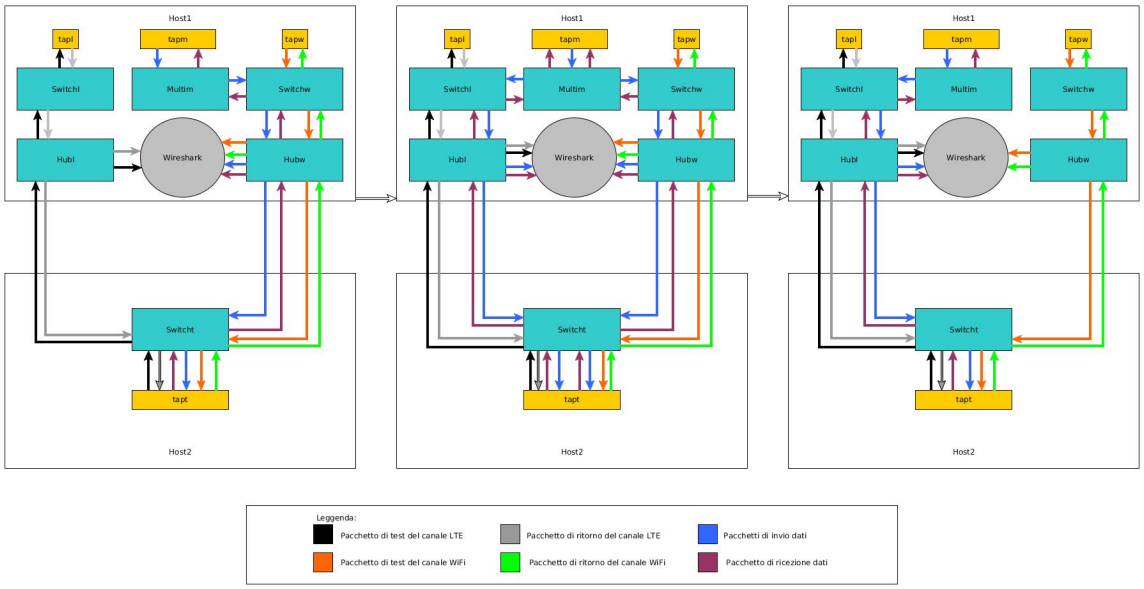

Figura 4: Andamento dei pacchetti nell'architettura.

La Figura 4 mostra l'andamento dei pacchetti all'interno dell'architettura.

2.4 Architettura finale

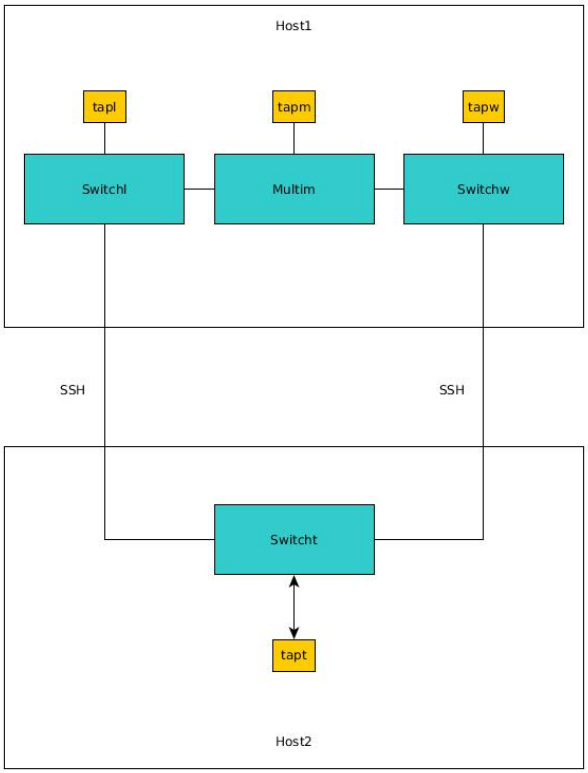

Figura 5: Architettura finale.

L'architettura finale che andrà ospitata sui dispostivi mobili non prevede l'utilizzo dei VDE hub e di Wireshark, infatti questi dispositivi sono necessari unicamente per raccogliere i dati e mostrare l'andamento dei pacchetti all'interno dell'architettura creata al fine di dimostrare il corretto funzionamento dell'architettura.
I comandi per creare l'architettura finale saranno quindi:

```
term0$ vde_plug tap://tapl switch:///tmp/switchl

term1$ vde_plug tap://tapw switch:///tmp/switchw

term2$ vde_plug tap://tapm multi:///tmp/multim

term3$ ssh user@remote.host
term3$ vde_plug tap://tapt switch:///tmp/switcht

term4$ vde_plug vde:///tmp/multim vde:///tmp/switchl

term5$ vde_plug vde:///tmp/multim vde:///tmp/switchw

term6$ vde_plug vde:///tmp/switchl cmd://'ssh
```

```
    -b <LTE interface ip address> user@remote.host
    vde_plug vde:///tmp/switcht'

term7$ vde_plug vde:///tmp/switchw cmd://'ssh
    -b <WiFi interface ip address> user@remote.host
    vde_plug vde:///tmp/switcht'
```

3 Test effettuati e risultati

Per verificare la corretta connessione tra gli host sono stati effettuati diversi test. Ognuno di essi prevedeva l'uso del comando:

```
ping -I <interface name> <destination ip address>
```

nell'Host 1 che invia dei pacchetti di tipo *ICMP*, specificando l'interfaccia, all'Host 2, il quale restituisce una risposta. Nell'architettura costruita l'indirizzo IP di destinazione è sempre il medesimo, ossia quello dell'interfaccia *tapt*: *10.0.3.1*
Per ogni test effettuato è stato verificato il corretto collegamento dell'interfaccia LTE con il comando:

```
ping -I tapl 10.0.3.1
```

Analogamente è stato testato il corretto funzionamento dell'interfaccia WiFi con il comando:

```
ping -I tapw 10.0.3.1
```

In fine è stato mostrato il corretto funzionamento dell'architettura creata con il comando:

```
ping -I tapm 10.0.3.1
```

I dati riguardanti i pacchetti *ICMP* scambiati tra gli host sono mostrati grazie al software Wireshark. Le immagini dei test hanno una uguale suddivisione:

- la parte sinistra è completamente occupata da Wireshark, dove si possono vedere i pacchetti che vengono scambiati;
- la parte in alto a destra è occupata da un terminale dove vengono lanciati i test;
- la parte in basso a destra è occupata da due terminali:
  - nel primo c'è il terminale contenente il processo che si occupa del collegamento tra *multim* e *switchl*;
  - nel secondo c'è il terminale contenente il processo che si occupa del collegamento tra *multim* e *switchw*.

Nelle sezioni successive si descrivono e si mostrano tutti i test e i riultati ottenuti.

3.1 Test a due vie

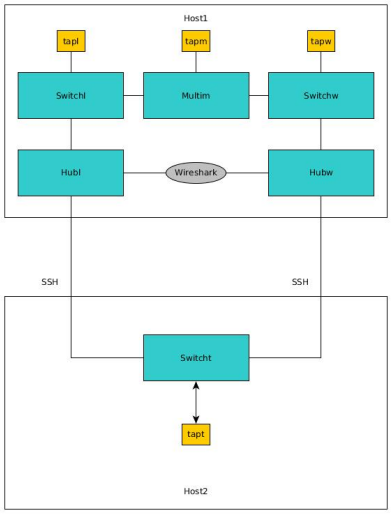

Figura 6: Architettura a due vie.

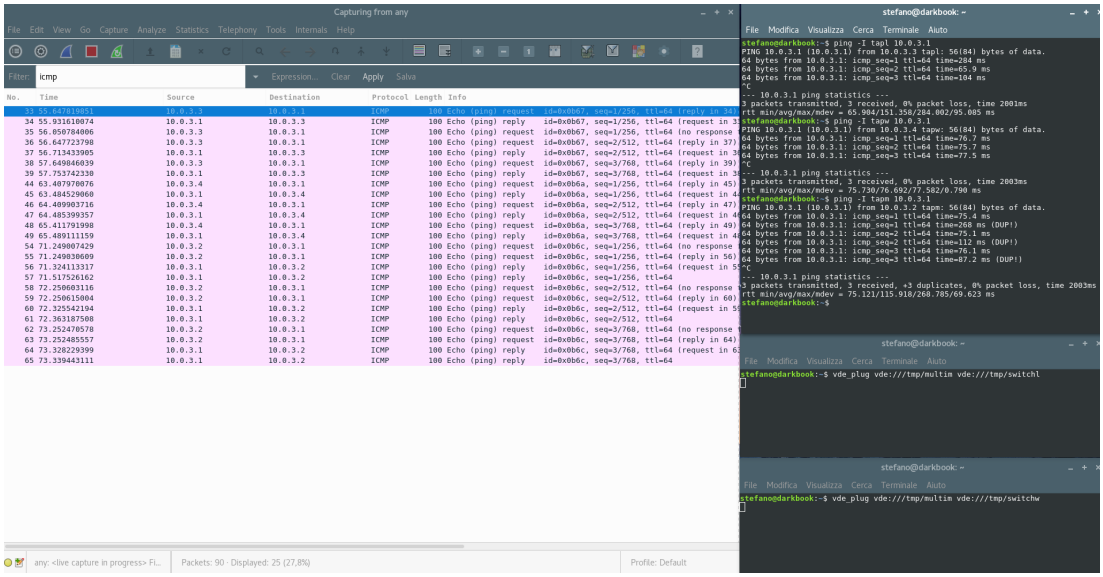

Figura 7: Test a due vie.

Come mostrato in Figura 6 l'architettura utilizzata è quella completa già descritta nella seconda sezione. Nella Figura 7 sono mostrati i test effettuati e come è possibile vedere tutto funziona correttamente. Da *tapl* si riesce a monitorare la qualità del collegamento LTE, da *tapw* si riesce a monitorare la qualità del canale WiFi e da *tapm* si riescono ad inviare pacchetti che vengono duplicati da *multim*.

3.2 Test via LTE

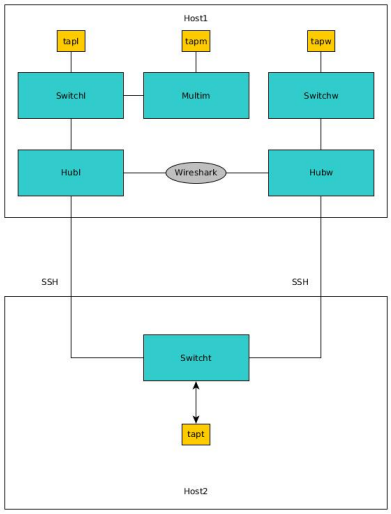

Figura 8: Architettura via LTE.

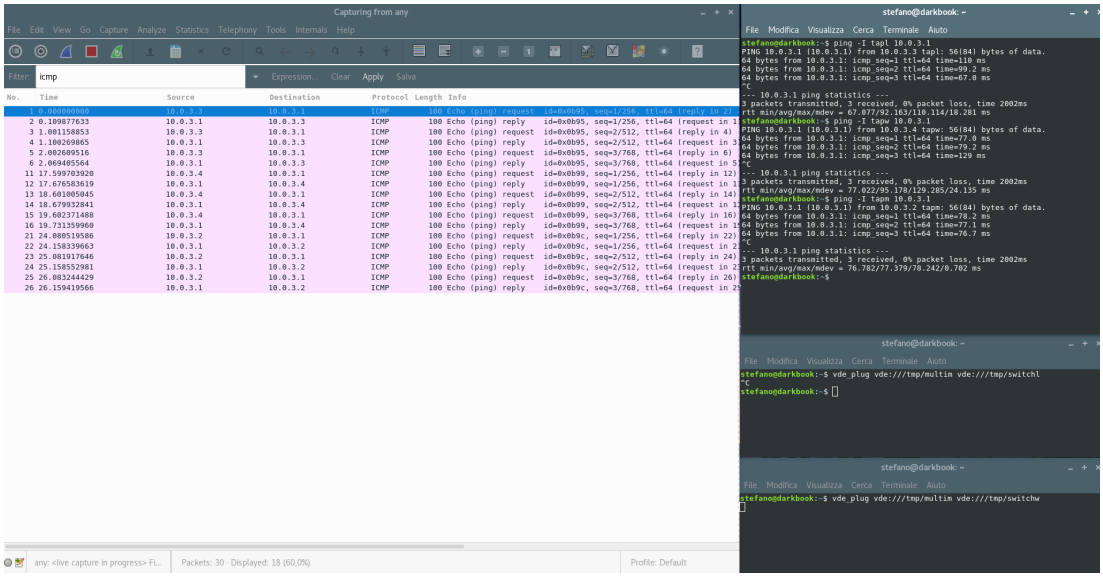

Figura 9: Test via LTE.

Come mostrato in Figura 8 l'architettura utilizzata per l'invio di pacchetti dati comprende il solo utilizzo dell'interfaccia LTE mentre rimangono le connessioni per verificare la qualità del canale. Nella Figura 9 sono mostrati i test effettuati e anche in questo caso è possibile vedere tutto funziona correttamente. Sempre dalla Figura 9 si nota che è stato interrotto il processo che collega *multim* e *switchw*. Da *tapl* si riesce a monitorare la qualità del collegamento LTE, da *tapw* si riesce a monitorare la qualità del canale WiFi e da *tapm* si riescono ad inviare i dati all'Host 2 utilizzando il collegamento tra *multim* e *switchl*.

3.3 Test via WiFi

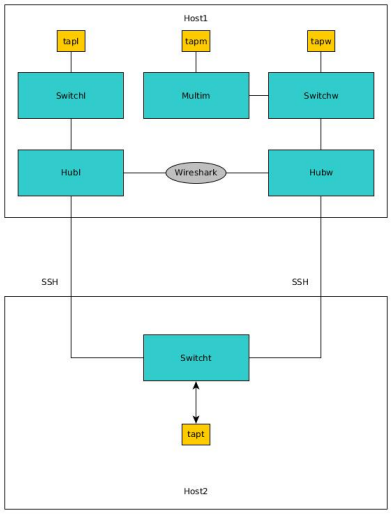

Figura 10: Architettura via WiFi.

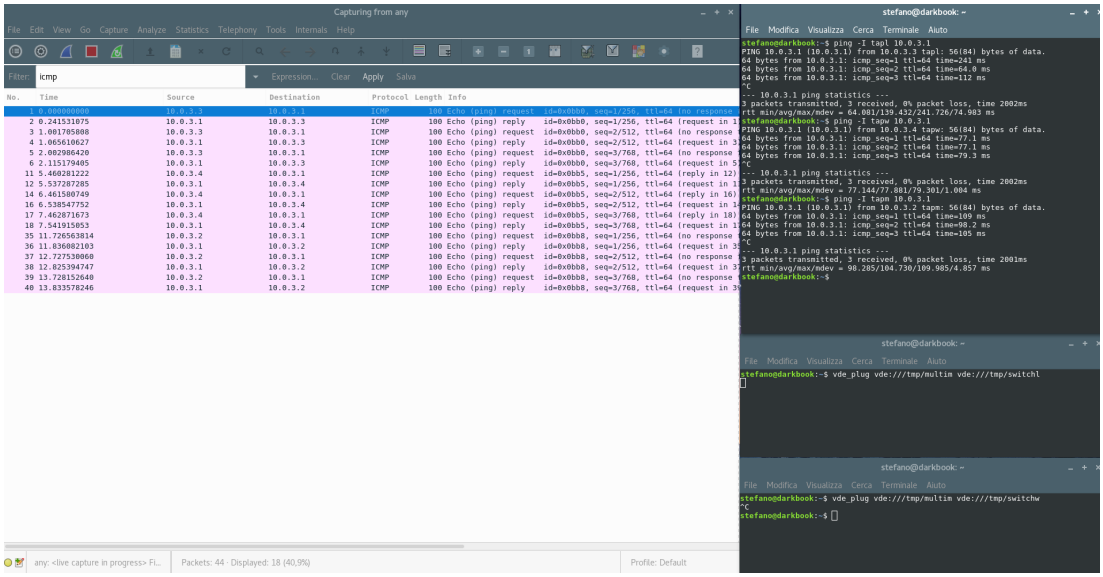

Figura 11: Test via WiFi.

Come mostrato in Figura 10 l'architettura utilizzata per l'invio di pacchetti dati comprende questa volta il solo utilizzo dell'interfaccia WiFi mentre rimangono le connessioni per verificare la qualità del canale. Nella Figura 11 sono mostrati i test effettuati e come nei casi precedenti è possibile vedere tutto funziona correttamente. Sempre dalla Figura 11 questa volta si nota che è stato interrotto il processo che collega *multim* e *switchl*. Come in precedenza da *tapl* si riesce a monitorare la qualità del collegamento LTE, da *tapw* si riesce a monitorare la qualità del canale WiFi e da *tapm* si riescono ad inviare i dati all'Host 2 utilizzando, questa volta, il collegamento tra *multim* e *switchw*.

3.4 Test a zero vie

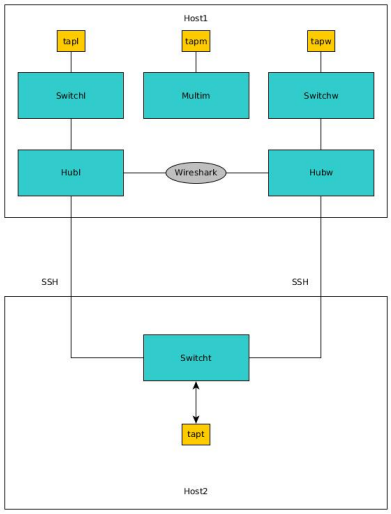

Figura 12: Architettura a zero vie.

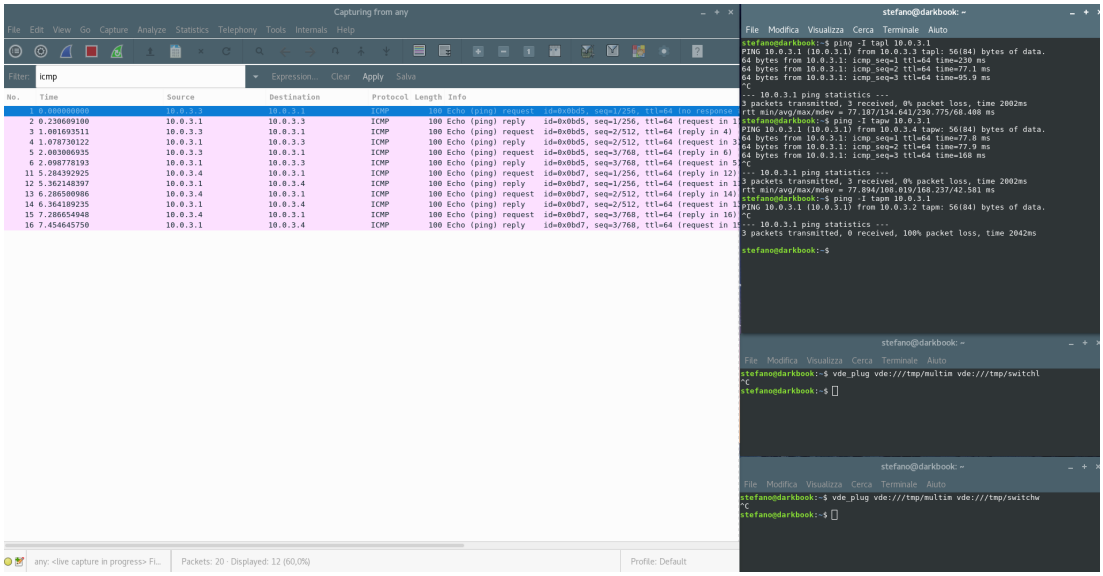

Figura 13: Test a zero vie.

Come mostrato in Figura 12 l'architettura utilizzata per l'invio di pacchetti dati non comprende alcuna connessione tra *multim* e gli switch mentre rimangono le connessioni per verificare la qualità del canale. Nella Figura 13 sono mostrati i test effettuati e come in tutti i casi visti fino ad ora tutto funziona correttamente. Sempre dalla Figura 13 questa volta si nota che è stato interrotto sia il processo che collega *multim* e *switchl*, sia quello che collega *multim* e *switchl* lasciando di fatto isolato *multim*. Come in precedenza da *tapl* si riesce a monitorare la qualità del collegamento LTE, da *tapw* si riesce a monitorare la qualità del canale WiFi, ma questa volta da *tapm* non si riescono ad inviare pacchetti verso la destinazione.

3.5 Risultati dei test

I risultati ottenuti dai test sono assolutamente soddisfacenti, infatti si è riuscito a dimostrare che l'architettura descritta in questo documento è efficace per lo scopo prefissato. Ricapitolando quanto detto in questa sezione, si è riusciti a mostrare che quando l'interfaccia che si vuole utilizzare per la comunicazione dei dati è connessa alla destinazione con più vie, allora i pacchetti vengono duplicati e quindi non si preferisce una via a discapito dell'altra. Si è dimostrata anche la facilità del cambio di via di comunicazione poiché è sufficiente interrompere un processo e attivarne un altro.

4 Conclusioni e sviluppi futuri

In conclusione si è mostrato come l'architettura proposta possa facilmente e rapidamente cambiare interfaccia di comunicazione ad un dispositivo a seconda delle necessità. Infatti come spiegato nella sezione 2, e poi mostrato nella sezione 3, è sufficiente avviare un processo ed interromperne un altro per cambiare canale di comunicazione senza perdita di pacchetti. L'unica controindicazione riguardante questa architettura è la ricezione di pacchetti duplicati nel momento in cui sono attive entrambe le interfacce, tuttavia, dipendentemente dalle necessità dell'utilizzatore finale, questo può essere solamente un piccolo prezzo da pagare per ottenere un'affidabilità e una qualità di comunicazione decisamente migliori e per ridurre la perdita di pacchetti.
Uno degli sviluppi futuri riguardante questa architettura può essere quello della costruzione di un demone che possa valutare la qualità dei canali di comunicazione e automaticamente effettuare le operazioni di cambio del canale come mostrato nelle sezioni precedenti.
Un ulteriore sviluppo futuro potrebbe essere quello di realizzare questa architettura su un dispositivo mobile Android andando così a sfruttare realmente tutti i benefici che essa comporta.