Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica

Reti

Tesi di Laurea Magistrale

Sviluppo infrastruttura per monitoraggio del traffico malevolo per l'utilizzo di Honeypot

Relatori:
Marco Mellia
Idilio Drago

Candidato:
Thomas Favale

Anno Accademico 2018/2019

Ringraziamenti

Il ringraziamento più grande va a mia mamma e a mio papà che hanno sempre saputo instancabilmente sostenermi e darmi la forza, soprattutto nei momenti più difficili, per raggiungere questo traguardo così importante per la mia vita. Grazie per avermi permesso di crescere. Grazie per aver creduto in me.

Un ringraziamento speciale va a mio fratello che fin da piccolo è sempre stato al mio fianco e con cui ho avuto l'immenso piacere di condividere i giochi, le fantasie, la vita.

Infine un doveroso ringraziamento va a tutti gli amici che ho conosciuto a Torino e ai Lolli che hanno riempito le mie giornate tra i corridoi del Politecnico e le spiagge di Leuca, rendendole indimenticabili.

Thomas Favale

Sommario

Questa tesi si propone di analizzare e spiegare i concetti generali di *Network Virtualization* per la creazione di un'infrastruttura di rete virtuale opportunamente mascherata all'esterno, con lo scopo di ospitare una serie di sistemi di sicurezza software quali gli *Honeypot*. L'obiettivo del lavoro è quello di creare un'infrastruttura di rete dedicata interamente alla cyber-security tale da poter ospitare una serie di *Honeypot* che hanno lo scopo di reperire informazioni sugli attacchi eseguiti sulla rete del *Politecnico di Torino*, impedendone l'individuazione della natura virtuale dell'ambiente.

Dall'analisi eseguita, la rete "privata" del *Politecnico di Torino* risulta essere protetta in ingresso dalla presenza di un Firewall che opera bloccando semplicemente tutte le connessioni non lecite in ingresso (per esempio verso host inesistenti, ma il cui indirizzo è allocato). Questi dati potrebbero risultare utili per scopi di ricerca, quindi si vuole poter reindirizzare il suddetto traffico verso una zona demilitarizzata (DMZ). In questa maniera la rete di produzione non è in alcun modo compromessa e contemporaneamente è possibile raccogliere dati utili. Dal punto di vista implementativo, una prima idea è stata quella di creare una serie di reti LAN fisiche con un evidente vantaggio prestazionale, ma poco efficiente dal punto di vista economico e di gestione, infatti le operazioni da svolgere richiedono una "ridotta" capacità di calcolo e tra l'altro la migrazione di una macchina da una rete verso un'altra risulta complicata e costosa dal momento che si è tenuti a "spegnere il servizio" e trasportare fisicamente la macchina nella posizione opportuna. In sostanza, tale idea non rappresenta un buon tradeoff tra efficienza, costi, flessibilità e semplicità, pertanto tale soluzione è da scartare. Un paradigma che negli ultimi anni si sta facendo strada è la virtualizzazione che oggigiorno, non si limita esclusivamente ai sistemi operativi, ma anche alle funzioni di rete, quindi di fatto il software sta gradualmente prendendo il posto dell'hardware. In virtù di tale tendenza, il fine è quello di eseguire su una stessa macchina fisica una serie di sistemi operativi, indipendenti l'uno dall'altro, su un hardware virtuale (emulato). Su questa tematica alcuni lavori erano già stati portati avanti in passato come per esempio il progetto *Honeyd*: per quanto possa sembrare uno strumento molto potente, presenta delle limitazioni in termini di flessibilità, infatti riesce a simulare una serie di topologie arbitrarie contenenti solo honeypot di natura predefinita. Il nostro obiettivo, invece, è quello di poter inserire in una rete arbitraria qualunque tipo di software. Conseguentemente è risultato necessario ricercare delle soluzioni differenti. I software che permettono la virtualizzazione attualmente sono molteplici e spaziano da soluzioni open-source a soluzioni proprietarie ma con un pattern di gestione ad alto livello comune: garantiscono prima di tutto la scalabilità (parametro principe per l'ingegnerizzazione di un data center) ed in secondo luogo una visione generale e completa delle virtual machines in esecuzione. Infatti una volta prodotto un data center di qualunque genere e dimensione, si vuole poterlo gestire semplicemente attraverso un'interfaccia coerente che ne nasconda la topologia fisica e permetta all'amministratore di avviare una qualunque *virtual machine* senza doversi occupare del server in cui è lanciata.

Ponendo la questione da un ulteriore punto di vista, ciò di cui è opportuno dotarsi è un sovrasistema operativo che sia in grado di gestire una serie di server che già contengono al proprio interno un sistema operativo insieme ad un agent, il quale può essere "pilotato" da remoto per l'esecuzione dei vari comandi provenienti da un Controller, che è fondamentalmente il software che fornisce l'interfaccia per l'interazione con l'intero data center. Il framework open-source più diffuso e su cui si basa l'intero progetto è *OpenStack*.

In base a quanto esposto sopra, l'idea è quella di utilizzare un numero relativamente ridotto di macchine fisiche grazie al paradigma della virtualizzazione, che permetterà il deploy di *Honeypot* sottoforma di *virtual machines*, quindi si è proceduto nel forwarding del traffico malevolo verso una rete virtualizzata composta da diversi router e LAN con topologia di arbitraria complessità,

che permettono al traffico di raggiungere delle particolari virtual machines, gli *Honeypot*. Il sistema virtualizzato in questione formerà una rete stub completamente scollegata da quella di produzione, costituendo solo un'appendice dell'intera maglia che dovrà gestire solamente il traffico forwardato dal firewall/router posto a monte. Si è implementata inizialmente un'architettura con un nodo centrale che rappresenta l'hypervisor ed uno secondario (il cui numero può essere incrementato in qualunque momento) che costituisce il compute node. Con lo scopo di accelerare il processo di testing, si è deciso di utilizzare un unico server per contenere l'intero sistema virtualizzato. Se ne sono valutate le prestazioni tramite analisi di latenza e throughput in presenza di differenti carichi di lavoro sul processore per apprezzare qualitativamente e quantitativamente la perdita di prestazione, così da intervenire in maniera mirata sui parametri dell'ambiente. L'obiettivo che si vuole raggiungere è quello di verificare che un attaccante esterno alla rete del *Politecnico di Torino* non sia in grado di comprendere la natura virtuale della rete in maniera banale e che sia costretto ad eseguire attività maggiormente invasive che potrebbero solamente giovare alla raccolta dati che si vuole effettuare. Per quanto riguada i test di latenza, i risultati sono discostanti rispetto a ciò che ci si sarebbe aspettato poiché, è stato riscontrato una generale riduzione del suo valore con l'aumento di carico sulla CPU: si può supporre che la causa sia la politica di *power saving* e di *throttling* del processore che, in presenza di bassi carichi opera a frequenze inferiori rispetto alle sue reali prestazioni per ridurre i consumi di energia. D'altro canto, per quanto riguarda il throughput tra *virtual machines*, quanto osservato è perfettamente in linea con le aspettative, infatti si può apprezzare una perdita progressiva di prestazione: dal momento che i pacchetti attraversanti sono elaborati proprio nel processore, un carico elevato comporta una riduzione della velocità di gestione degli stessi. Maggiore attenzione è stata prestata nello scenario *virtual machine - server fisico*, poiché rappresenta il più comune ed il più critico. A fronte di dati non ottimi in rapporto a quelli di una macchina fisica, si è provveduto ad un'analisi approfondita sul sistema virtualizzato per identificarne le cause: si è riscontrato una dimensione dei buffer sui router troppo piccoli che, a fronte dei test portati avanti, ha permesso di giungere alla conclusione che ne fossero la causa primaria. Tali dimensioni sono state progressivamente modificate per apprezzarne gli effetti benefici sulle prestazioni. Al termine si è constatato come in presenza di una rete a 100Mb/s i risultati sono perfettamente paragonabili con quelli di una macchina fisica. In definitiva è possibile ritenersi soddisfatti del risultato poiché i dati ottenuti dall'ambiente virtuale sono perfettamente comparabili con quelli di una rete fisica, pertanto si può ritenere che l'obiettivo preposto sia stato pienamente raggiunto.

Un ulteriore approfondimento è stato fatto in presenza di una rete a 1Gb/s, riscontrando prestazioni non al livello di quelle misurate sulle macchine fisiche a causa di un progressivo riempimento delle code. In conclusione si può ritenere che la perdita di prestazione riscontrata sia dovuta all'hardware della macchina fisica che, probabilmente, non è sufficientemente efficiente per eseguire le operazioni di trasferimento dei pacchetti dall'ambiente virtuale a quello fisico. Tale punto si presta perfettamente a lavori futuri ricercando opportuni tuning sul sistema OpenStack per limitare ancor di più le perdite di prestazione, a cui è possibile aggiungere uno studio e l'implementazione di topologie che possano essere il più simili possibili a quelle reali. Altrettanto interessante potrebbe risultare l'inserimento di una rete parallela a quella esposta agli hacker per il transito delle informazioni raccolte dagli Honeypot, così da trasferirle verso le apposite macchine che eseguiranno le analisi sugli stessi.

Il progetto portato avanti è stato molto interessante e stimolante, poiché nel corso della mia carriera universitaria ho avuto modo di studiare ed approfondire queste tematiche, ma non si è mai presentata l'opportunità di lavorare su un progetto reale che potesse coniugare al meglio i miei interessi ciò, mi ha permesso di essere fortemente spronato non solo nel raggiungere lo scopo prefissato, ma anche nel ricercare le soluzioni migliori e più efficienti possibili.

Contents

1 Introduzione

L'origine della rete Internet risale agli anni sessanta, durante la guerra fredda, come strumento di difesa e anti-spionaggio messo a punto dagli USA. Tale realtà divenne di pubblico dominio solo nel 1991 con l'*High performance computing act*, emanato sempre dagli USA, che dava la possibilità di ampliare e diffondere una rete Internet fino a quel momento di proprietà statale e destinata al mondo scientifico. Ciò significava che l'accesso alla stessa, fosse riservato a pochi utenti con il nobile scopo di conservare, catalogare e consultare documenti, infatti i primi nodi si basavano su un'architettura client/server e le applicazioni eseguite erano fondamentalmente programmi di FileTransport Protocol (FTP) e Telnet (il servizio di posta elettronica fu inventato solo successivamente nel 1971).

Ne consegue che il mondo che oggi conosciamo e più specificatamente la rete Internet, è frutto dello sviluppo avviato grazie alla sopracitata legge che ha permesso la diffusione dell'informatica in maniera più capillare nella società e di conseguenza la specializzazione degli utenti nell'utilizzo della rete stessa. Ciò ha comportato riscontri positivi sotto molti punti di vista (economico e sociale, in primo luogo), ma anche negativi con la nascita, per esempio, della criminalità informatica che tra i tanti, punta a porre a rischio i sistemi informativi di qualunque ente economicamente strategico (aziende, università, Stati). È in questo contesto che si pone il macro-progetto: come intercettare questo traffico malevolo per analizzarlo ed utilizzare le informazioni ricavate per migliorare i sistemi di sicurezza. L'obiettivo di questo lavoro, nello specifico, è quello di creare un'infrastruttura di rete dedicata interamente alla cyber-security, ospitando una serie di *Honeypot* che hanno lo scopo di attirare pirati informatici che, conseguentemente, lasceranno delle tracce sull'attività da loro svolta. Tali informazioni risultano estremamente utili per comprendere le nuove forme di attacchi informatici ed individuarne soluzioni sempre nuove e all'avanguardia. Non si ferma, dunque, esclusivamente all'installazione di una serie di macchine in rete, valutandone semplicemente le prestazioni, ma è stato svolto un lavoro mirato con l'obiettivo di produrre un sistema contemporaneamente flessibile, di facile gestione ed ampliamento: si è cercato di sfruttare il minor numero di macchine fisiche possibili in favore della virtualizzazione. Come emergerà successivamente in questa tesi, la scelta di creare un ambiente virtuale garantisce innumerevoli vantaggi per l'amministratore di rete o comunque per colui il quale sarà designato a costruire un'infrastruttura del genere, ma lascia un grande dubbio sulla possibilità che un attaccante esterno possa scoprire la natura della rete e quindi abbandoni il tentativo di hackeraggio. Pertanto oltre ai comuni test per verificarne il corretto funzionamento, ne sono stati eseguiti degli altri più mirati sulle prestazioni dell'ambiente virtuale in differenti condizioni di stress. I risultati ricavati risultano molto interessanti poiché permettono di comprendere non solo quanto la quantità di risorse a disposizione del calcolatore siano un importante fattore discriminante sulle prestazioni dello stesso, ma anche che un server può essere in grado di gestire un ambiente virtuale anche sotto stress, cammuffandone la sua natura.

Su questa tematica alcuni lavori erano già stati portati avanti in passato come per esempio in [1]. Si ritiene infatti che gli honeypot virtuali siano degli strumenti alquanto interessanti perché richiedono meno sistemi fisici, riducendo i costi di manutenzione. Utilizzando honeypot virtuali, è possibile popolare una rete con host che eseguono numerosi sistemi operativi. *Honeyd*, per esempio, è caratterizzato da diversi componenti: un database di configurazione, un central packet dispatcher, gestori di protocolli, un motore di personalità e un componente di routing opzionale. Per quanto lo strumento possa sembrare molto potente, presenta delle limitazioni in termini di flessibilità, infatti per quanto esposto nell'articolo, riesce a simulare una serie di topologie arbitrarie contenenti solo honeypot di natura predefinita. Il nostro obiettivo, invece, è quello di poter inserire in una rete arbitraria qualunque tipo di software sia per l'implementazione di Honeypot che possibilmente *Darknet*. Conseguentemente è risultato necessario ricercare delle

soluzioni differenti.

Quanto esposto in questa testi, si basa pesantemente sull'utilizzo di sistemi per la virtualizzazione quali OpenStack poiché, come detto, garantisce la flessibilità richiesta e permette un'espansione del progetto senza problematiche di scalabilità.

2 Background

Per meglio comprendere gli aspetti tecnici che caratterizzano questo lavoro, è opportuno presentare i concetti su cui esso si basa, partendo prima di tutto dal concetto di *sicurezza informatica*: l'insieme dei prodotti, dei servizi e dei comportamenti individuali che si prefiggono come scopo quello di proteggere i sistemi informativi di un ente, garantendo *confidenzialità*, *integrità* e *disponibilità*. Ciò può essere garantito solo dalla corretta applicazione dei concetti di *prevenzione*, *rilevamento* e *reazione*.

La *prevenzione* può essere definita come qualsiasi impresa che:

1. scoraggia gli intrusi;
2. rende le violazioni nel sistema non fattibili.

Il *rilevamento* è il processo di identificazione della presenza di azioni che danneggiano i sistemi:

1. riservatezza;
2. integrità;
3. disponibilità.

La *reazione* descrive l'esecuzione di misure reattive dopo aver rilevato azioni dannose. Idealmente, la reazione rafforza la prevenzione e migliora le future rilevazioni.

Come accennato, con l'avanzamento tecnologico i rischi sono aumentati vertiginosamente e conseguentemente la sicurezza informatica deve seguire questo trend e, per quanto si cerchi di tutelarsi da ogni forma d'attacco, col tempo ne sorgono sempre di nuovi, pertanto essa è un'operazione di tipo reattivo: evidenziato il problema, si ricerca la soluzione.

Le problematiche più evidenti sono le seguenti:

- i dati e l'elaborazione sono distribuiti: è necessario proteggere contemporaneamente più calcolatori;
- tramite la rete è possibile raggiungere ogni elaboratore connesso;
- i calcolatori sono dotati di capacità di elaborazione e memorizzazione autonoma;
- le comunicazioni sulla rete sono in broadcast dal momento che essa è caratterizzata da canali condivisi da più nodi e generalmente gestiti da terzi.

In definitiva, diventa sempre più difficile prevedere tutte le possibili forme di attacco, pertanto bisogna rimanere in "ascolto". Questo è uno degli obiettivi finali dell'intero progetto: rimanere in ascolto per trovare e studiare i nuovi pericoli per creare soluzioni ottime ed efficienti. Per raggiungere questo obiettivo è opportuno creare un'infrastruttura di rete che possa essere adibita a tale scopo, ovvero riservata allo studio degli attacchi informatici, rimanendo il più possibile scollegata dalla rete di produzione (rete ospitante gli elaboratori operativi del *Politecnico di Torino*), in cui poter inserire, in futuro, una serie di "macchine" che prendono il nome di Honeypot.

2.1 Honeypot

Come largamente descritto in [2], un *Honeypot* è un sistema usato come "trappola" con lo scopo di proteggere i sistemi informativi da attacchi di pirati informatici. In generale si cerca di esporre un dispositivo con lo scopo di far credere all'attaccante che possa permettergli l'accesso ad informazioni riservate, ma in realtà risulta essere isolato o addirittura privo di numerose funzionalità.

Il suo valore risiede nella natura e nella frequenza degli attacchi subiti. Particolare attenzione è da prestare nella gestione dello stesso, poiché un *Honeypot* non opportunamente configurato e protetto, potrebbe essere utilizzato per penetrare all'interno della rete di produzione.

La letteratura in merito a questo sistema di sicurezza, risulta essere molto vasta, poiché le potenzialità che offre sono notevoli. In questo capitolo saranno trattate le tematiche principali per comprenderne al meglio lo stato dell'arte e l'idea alla base del progetto.

La ricerca in ambito honeypot si compone principalmente di due pilastri:

1. lo sviluppo del software honeypot e la sua effettiva distribuzione;
2. l'analisi dei dati del log acquisiti in modo strutturato.

L'obiettivo di un Honeypot è quello di distrarre gli attaccanti dal loro obiettivo reale o di raccogliere informazioni sugli attaccanti e sui modelli di attacco, come l'insieme di host di destinazione popolari e la frequenza delle richieste-risposte. Tuttavia, vale la pena notare che gli Honeypot non sono da considerarsi un'implementazione per risolvere un certo problema, piuttosto uno strumento per comprenderli. Indipendentemente da dove e come vengono installati gli Honeypot, contribuiscono solo se esposti agli attacchi.

Oltre agli *Honeypot*, è possibile utilizzare una combinazione di *Firewall*, *Sistemi di Rilevamento delle Intrusioni* (IDS), *Antivirus* (AV), *Sistemi di Prevenzione delle Intrusioni* (IPS) e monitoraggio dei log per migliorare la sicurezza della rete. È importante differenziare questi dispositivi per utilizzarli al massimo del loro potenziale[1] [3]. Tradizionalmente, i firewall monitorano e controllano il traffico di rete in base a regole di sicurezza statiche predeterminate e quindi fungono da barriera tra reti sicure e insicure. Un IDS analizza interi pacchetti, sia l'*header* che il *payload*, alla ricerca di firme dannose: quando viene rilevato un evento noto viene generato un avviso. Gli IPS possono essere intesi come un'estensione di IDS che non solo emette avvisi, ma rifiuta attivamente i pacchetti e aggiunge dinamicamente nuove regole. Anche il software AV è basato sulla firma, ma funziona solo a livello di file e tenta di catturare infezioni da un trojan o da un virus.

I problemi vengono rilevati valutando statisticamente i file di registro per modelli di testo noti che indicano eventi importanti. Gli standard di sicurezza informatica prevengono gli attacchi definendo le linee guida sulla sicurezza per il software e l'infrastruttura, come una configurazione pulita, un kernel rinforzato e aggiornamenti recenti. Oggigiorno una chiara distinzione tra questi concetti diventa difficile, poiché le soluzioni di sicurezza sono ibride e combinano diversi concetti al fine di minimizzare il trade-off individuale.

Gli honeypot aggiungono poco valore alla prevenzione in quanto non proteggono dalle violazioni della sicurezza e possono inibire gli attacchi, poiché i pirati informatici potrebbero essere preoccupati di sprecare tempo e risorse, distratti dal loro obiettivo primario (sistemi di produzione).

In caso di rilevamento gli honeypot aggiungono un valore esteso. È molto difficile individuare attacchi ai sistemi di produzione perché gli attacchi semplicemente sono sommersi dalla grande quantità di attività produttive. Un Honeypot può semplificare il processo di rilevamento: non hanno attività di produzione, tutte le connessioni con l'Honeypot sono sospette per natura e pertanto rilevano un probe non autorizzato, una scansione o un attacco con quasi nessun falso positivo e negativo. Come suggerisce il nome, i sistemi di rilevamento delle intrusioni sono progettati per rilevare gli attacchi ma, la loro efficacia dipende in larga misura dalla qualità della

[1] In fase di progettazione e implementazione di un sistema di sicurezza per una rete di qualunque genere, è importante che i dispositivi provengano da vendor differenti per motivazioni alquanto banali: il software utilizzato da un singolo vendor potrebbe essere affetto da bug o falla che, se presente su tutti i dispositivi che fanno sicurezza, permetterebbe ad un attaccante di prevaricare facilmente le difese. Al contrario, la presenza di dispositivi eterogenei, garantisce una protezione più elevata.

firma. Gli amministratori possono essere sopraffatti dai falsi positivi, diventando "insensibili" e ignorando quei messaggi che rappresentano dei veri positivi. D'altra parte, i falsi negativi sono anche possibili se non esiste una firma appropriata quando vengono utilizzati nuovi *exploit*[2]. Pertanto la reazione agli attacchi può essere accelerata con l'aiuto di honeypot. L'analisi dell'attacco è sostanzialmente più semplice, poiché i dati di attacco non sono mescolati con i dati relativi all'attività di produzione e, contrariamente ai sistemi di produzione, gli honeypot possono essere completamente eliminati per l'analisi, il che consente un'analisi corretta e completa. Le informazioni possono quindi essere utilizzate per pulire i sistemi di produzione e comprendere l'exploit, che è il primo passo per correggere le vulnerabilità corrispondenti.

2.1.1 Classificazione

Gli Honeypot attualmente in utilizzo e sviluppati possono essere classificati in tre categorie a seconda del loro comportamento ed implementazione:

- Low Interaction Honeypot (LIHP):

  Simulano solo un piccolo insieme di servizi come SSH o FTP, non forniscono alcun accesso al sistema operativo all'utente malintenzionato. Producono risposte minime per consentire l'handshake del protocollo. Poiché la raccolta di informazioni è limitata sono utilizzati principalmente per la valutazione statistica. Tuttavia, sono sufficienti per riconoscere i picchi nel numero di richieste, ad esempio create da worms autonomi. I LIHP tendono ad essere considerati come Honeypot di produzione per la semplicità di gestione ed utilizzo.

- Medium Interaction Honeypot (MIHP):

  Sono leggermente più sofisticati di LIHP, tuttavia non forniscono ancora alcuna funzionalità del sistema operativo. I servizi simulati sono più elaborati e forniscono un livello più alto di interazione per l'attaccante, infatti MIHP produce risposte ragionevoli nella speranza di scatenare gli attacchi successivi. Per essere un obiettivo attraente, MIHP emula anche più servizi di LIHP. A causa della ridotta capacità di interazione, LIHP e MIHP hanno entrambe scarse possibilità di essere compromessi.

- High Interaction Honeypot (HIHP):

  Sono i più sofisticati e i più complessi da implementare, distribuire e mantenere, poiché forniscono all'attaccante un ambiente di sistema operativo reale non ristretto. Inoltre viene installato un enorme insieme di servizi. HIHP raccolgono la maggior quantità possibile di informazioni, tra cui registri di attacco completi, accesso ai dati, attraversamento di file tree, codici byte eseguiti ecc. A causa della complessità elevata, l'analisi tramite HIHP viene solitamente eseguita solo in ambiti di ricerca al fine di esaminare gli attacchi e sviluppare le opportune contromisure. Uno scopo secondario risulta senz'altro la comprensione dei motivi, dei comportamenti, gli strumenti e l'organizzazione della comunità black-hat[3].

Un'ulteriore classificazione può essere effettuata in base alla loro natura:

- Physical Honeypot;
- Virtual Honeypot.

[2] Bug o falla utilizzata dall'attaccante per eseguire operazioni malevoli.

[3] Hacker con intenti criminali: mantiene segrete le proprie conoscenze sulle vulnerabilità e gli exploit che trova riuscendo a inserirsi in un sistema o in una rete violando illegalmente sistemi informatici.

Evidentemente, un honeypot fisico è una vera macchina connessa in rete, mentre un honeypot virtuale viene simulato (virtualizzato) da una macchina host che inoltra il traffico di rete all'honeypot virtuale. Più honeypot virtuali possono essere simulati su un singolo host. Questo è proprio l'obiettivo che si cerca di raggiungere con questo lavoro, pertanto è opportuno approfondire il concetto di virtualizzazione per comprendere al meglio il funzionamento dell'ambiente sviluppato.

2.2 Virtualizzazione

Da quello che si può desumere dalla letteratura, la virtualizzazione risulta uno strumento estremamente efficiente e flessibile per la condivisione di risorse tra diversi sistemi operativi:

- CPU;
- Memoria;
- I/O (ad esempio NIC: scheda di rete).

L'obiettivo è di eseguire su una stessa macchina fisica una serie di sistemi operativi, indipendenti l'uno dall'altro, su un hardware virtuale (emulato). Oltre all'evidente potenzialità e flessibilità desumibile da una visione generale del paradigma, è possibile evidenziare ulteriori vantaggi scavando a fondo nelle funzionalità che l'emulazione può offrire: l'utente, infatti, oltre ad avere la possibilità di istanziare più sistemi in una singola macchina, può anche personalizzare a seconda delle necessità l'hardware sottostante modificandone le risorse da dedicare.

Il paradigma della virtualizzazione, viste le potenzialità precedentemente esposte, ha un pesante impiego all'interno dei data center: i calcolatori sono caratterizzati da un'elevata quantità di risorse sia dal punto di vista computazionale che di memorizzazione e non risultano essere sfruttati al pieno delle loro potenzialità. Per ovviare a questo "spreco" di risorse, la soluzione individuata è, ovviamente, quella di eseguire sulla stessa macchina, del software che compie operazioni in parallelo (ad esempio una serie di sistemi operativi), il che significa istanziare una serie di virtual machines che si suppone consumino una quantità sensibile di risorse in modo tale da ammortizzare i costi dell'hardware fisico.

I software che permettono la virtualizzazione attualmente sono molteplici e spaziano da soluzioni open-source a soluzioni proprietarie ma con un pattern di gestione ad alto livello comune: a differenza dei software di virtualizzazione più semplici e per uso "domestico", questi garantiscono prima di tutto la scalabilità (parametro principe per l'ingegnerizzazione di un data center) ed in secondo luogo una visione generale e completa delle virtual machines in esecuzione. Infatti una volta prodotto un data center di qualunque genere e dimensione, si vuole poterlo gestire semplicemente attraverso un'interfaccia coerente che ne nasconda la topologia fisica e permetta all'amministratore di avviare una qualunque virtual machine senza doversi occupare del server in cui è lanciata.

Ponendo la questione da un ulteriore punto di vista, ciò di cui è opportuno dotarsi è un sovrasistema operativo che sia in grado di gestire una serie di server che già contengono al proprio interno un sistema operativo insieme ad un agent, il quale può essere "pilotato" da remoto per l'esecuzione dei vari comandi provenienti da un Controller che è, fondamentalmente, il software che fornisce l'interfaccia per l'interazione con l'intero data center.

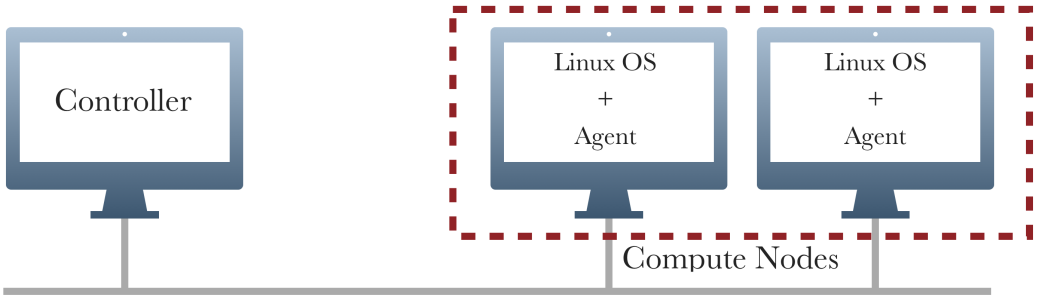

Figure 1: Architettura di un data center

Il framework open-source più diffuso e che permette operazioni di tale genere e su cui si basa l'intero progetto è OpenStack [4].

2.3 OpenStack

Lo sviluppo tecnologico finora raggiunto in ambito informatico ci permette di distinguere due macro "ere" dettate non solo dall'avanzamento in termini prestazionali dei componenti elettronici, ma anche dalla diffusione della tecnologia stessa all'utente finale che può sfruttare innumerevoli servizi provenienti principalmente dalla rete:

1. un'era "primordiale" in cui i servizi erano creati e messi a disposizione da un'unica macchina con il proprio sistema operativo (generalmente Unix/Linux) indipendente da qualunque altro calcolatore;

2. un'era "moderna" in cui i servizi sono distribuiti su un intero data center per ragioni di costo e scalabilità, innanzitutto, quindi l'orizzonte non è più la gestione del singolo server e del singolo sistema operativo, ma si ha conseguentemente la necessità di un tool per permette una gestione più ampia di tipo IaaS (Infrastructure-as-a-Service).

Nato nel 2010 dalla collaborazione tra NASA [5] e RackSpace [6], OpenStack è il tool open-source più largamente diffuso che permette il controllo di vasti pool di risorse di elaborazione, storage e di rete in un data center, gestito tramite un dashboard o tramite un'API.

2.3.1 Componenti

Dal punto di vista architetturale è organizzato a blocchi, ognuno di essi autonomo e dedicato ad un particolare servizio necessario al deploy della propria soluzione. Di seguito è riportata una mappa dei vari componenti e relativi servizi:

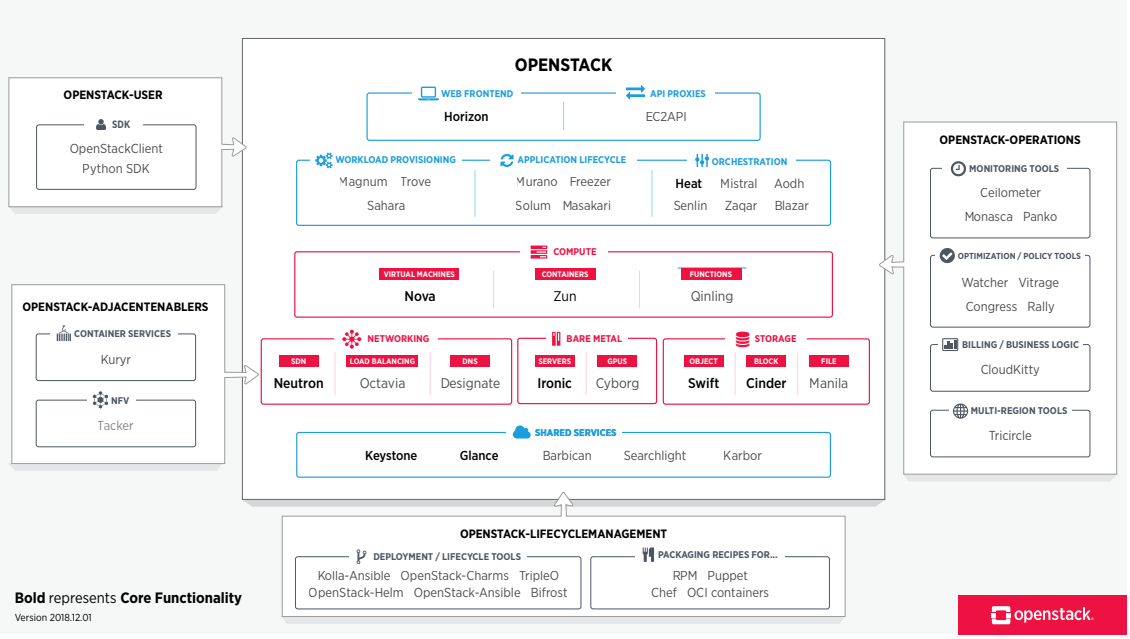

Figure 2: Architettua a blocchi di OpenStack (https://www.openstack.org/software/)

I servizi più importanti e fondamentali per il progetto che si vuole descrivere sono i seguenti:

- Compute:

  Nova: è un controller di cloud computing , che è la parte principale di un sistema IaaS. È progettato per gestire e automatizzare pool di risorse informatiche e può funzionare con varie tecnologie di virtualizzazione. La sua architettura permette una scalabilità orizzontale senza particolari requisiti hardware o software.

- Block Storage:

  Cynder: fornisce un catalogo e un repository per le immagini del disco virtuale che sono comunemente utilizzate dal blocco di Compute. È appropriato per scenari sensibili alle prestazioni come l'archiviazione di database, i file system espandibili o la fornitura di un server con accesso all'archiviazione a livello di blocco non elaborata. La gestione delle istantanee offre potenti funzionalità per il backup dei dati archiviati su volumi di storage a blocchi. Le istantanee possono essere ripristinate o utilizzate per creare un nuovo volume di archiviazione a blocchi.

- Dashboard:

  Horizon: fornisce un'interfaccia utente modulare basata su Web per tutti i servizi di OpenStack. Utilizzato per eseguire la maggior parte delle operazioni come l'avvio di un'istanza, l'assegnazione di un indirizzo IP e l'impostazione dei controlli di accesso. Una delle potenzialità "nascoste" è la possibilità di eseguire automazione nella gestione delle risorse.

- Identità:

  Keystone: fornisce autenticazione ed autorizzazione per tutti i servizi di OpenStack. Supporta molteplici forme di autenticazione tra cui credenziali standard per nome utente e

password, sistemi basati su token e accessi in stile AWS (Amazon Web Services https://aws.amazon.com/it/).

- Object Storage:

  Swift: è un sistema di storage ridondante scalabile. Gli oggetti e i file vengono scritti su più unità disco distribuite tra i server nel data center, con il software OpenStack responsabile di garantire la replica e l'integrità dei dati all'interno del cluster. I cluster di archiviazione si dimensionano orizzontalmente semplicemente aggiungendo nuovi server. In caso di guasto di un server o di un disco rigido, OpenStack replica il suo contenuto da altri nodi attivi in nuove posizioni nel cluster.

- Network:

  Newtron: fornisce "network connectivity as a service" tra i dispositivi di interfaccia gestiti da altri servizi OpenStack ed è il cuore pulsante di tutto l'aspetto "retistico" del sistema. Permette l'emulazione di reti e dispositivi di rete, tutto completamente virtualizzato. I modelli standard includono VLAN, gestione degli indirizzi IP, configurazione statica di indirizzi IP o tramite DHCP. Gli indirizzi IP flottanti consentono il reindirizzamento dinamico del traffico verso qualsiasi risorsa nell'infrastruttura IT, in modo che gli utenti possano reindirizzare il traffico durante la manutenzione o in caso di errore. L'amministratore è in grado di creare la propria rete, controllare il traffico e connettere server e dispositivi a una o più reti.

- Metering:

  Celiometer: fornisce un'infrastruttura per la raccolta di dati di misurazione (ad esempio sulla performance).

- Orchestration:

  Heat: offre la possibilità di configurare i servizi con un semplice file XML, che viene istanziato chiamando i moduli appropriati.

Ognuno dei blocchi, come spiegato in precedenza, opera autonomamente e il singolo "compito" da svolgere non dipende in alcun modo dagli altri ma, per permettere la fruibilità del servizio nella sua interezza, è necessario che ci siano delle comunicazioni tra gli stessi, in modo tale da orchestrare al meglio il funzionamento. Di seguito è riportata una mappa delle interazioni tra i diversi componenti:

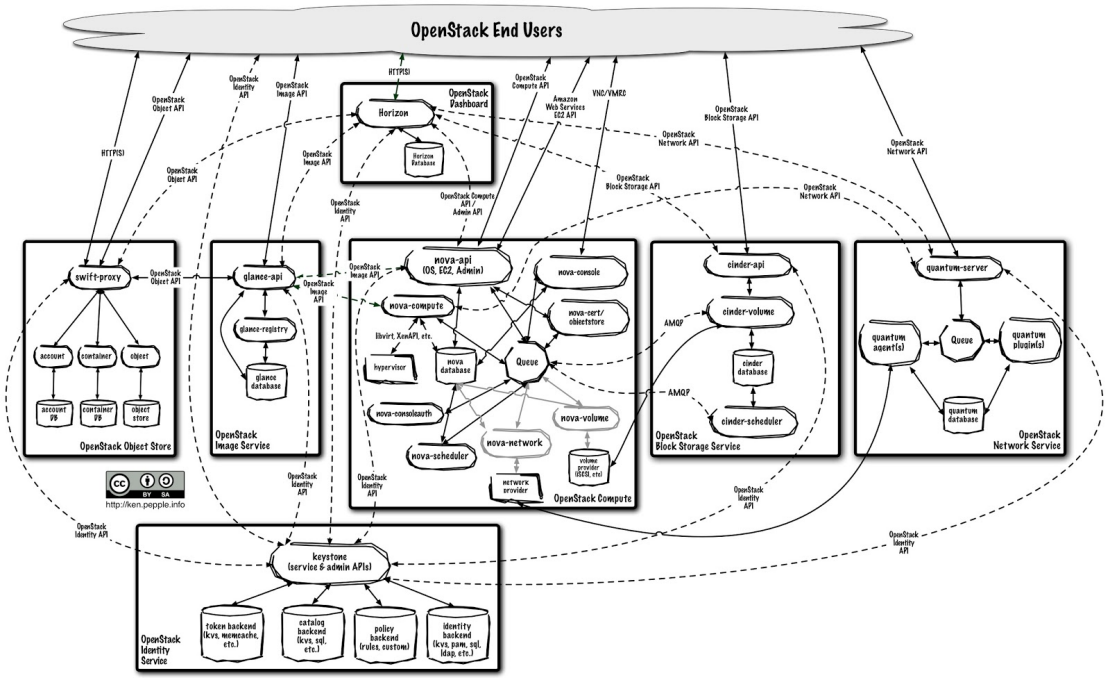

Figure 3: Interazioni tra blocchi operativi (http://ken.pepple.info)

Ogni macro blocco rappresenta uno dei componenti principali del sistema con le rispettive interazioni inter ed intra blocco che avvengono tramite un sistema di messaggi che vengono raccolti dai componenti *queue* (coda) in modalità *publish* and *subscribe*: mittenti e destinatari dialogano attraverso un *dispatcher* (la coda). Il mittente di un messaggio (detto publisher) si limita esclusivamente alla "pubblicazione" del proprio messaggio al dispatcher, mentre i destinatari si rivolgono a loro volta al dispatcher "abbonandosi" alla ricezione degli stessi. Il dispatcher quindi, inoltra ogni messaggio inviato da un publisher a tutti i subscriber interessati a quel messaggio.

2.3.2 Organizzazione dell'ambiente

Tutti i componenti di OpenStack sono da installare opportunamente sulle macchine che compongono il data center secondo la logica indicata nello schema seguente:

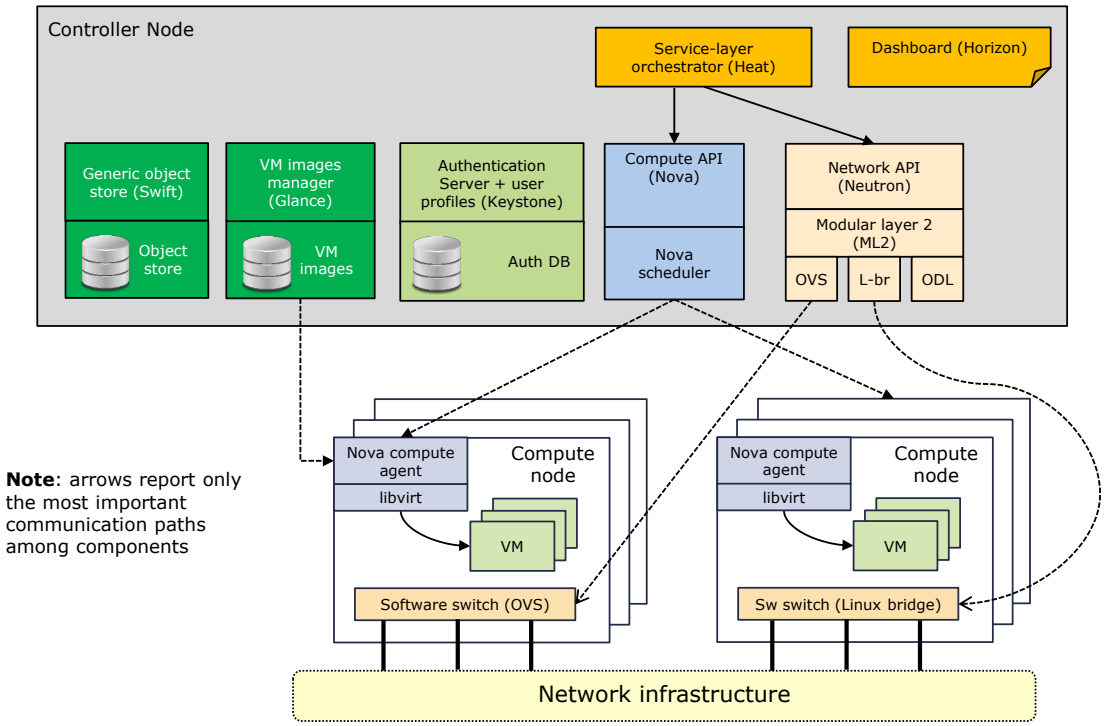

Figure 4: Organizzazione dei componenti sulle macchine fisiche (http://par.frisso.net)

Il data center sarà quindi composto da:

- una serie di server definiti Compute node, che contengono:
  - le singole virtual machines avviate;
  - l'agent che ne permette il collegamento con il modulo di controllo;
  - il software switch che permette la comunicazione tra le diverse virtual machines, il quale a sua volta può essere collegato alla infrastruttura reale;
- e il Controller node che possiede tutti i componenti server tra cui:
  - Nova, che espone un sotto-componente definito *scheduler* che ha come compito quello di stabilire quale server è designato all'avvio di una particolare *virtual machine*;
  - Keyston per l'autenticazione;
  - Glance per la gestione e la memorizzazione delle immagini (OSs);

ed ha lo scopo di gestire interamente i Compute node.

3 Architettura ed implementazione

In questa sezione verranno trattati i punti fondamentali su cui si basa il progetto e le idee utilizzate per l'implementazione e lo sviluppo dello stesso.

Il lavoro si concentra sulla creazione di una rete stub dedicata interamente alla cyber-security e agganciata alla rete del *Politecnico di Torino*, che possa ospitare al proprio interno una serie di macchine che costituiscono una rete di Honeypot.

La sfida principale risulta quella di riuscire a creare un'infrastruttura che possa garantire:

- in primo luogo un collegamento coerente delle macchine;
- una facile gestione;
- elevata flessibilità, con l'obiettivo lungimirante di poter espandere tale progetto per offrire al proprio interno ancora più servizi.

3.1 Architettura

Dalla documentazione analizzata in merito all'architettura della rete del *Politecnico di Torino*, si può estrapolare il seguente schema a blocchi:

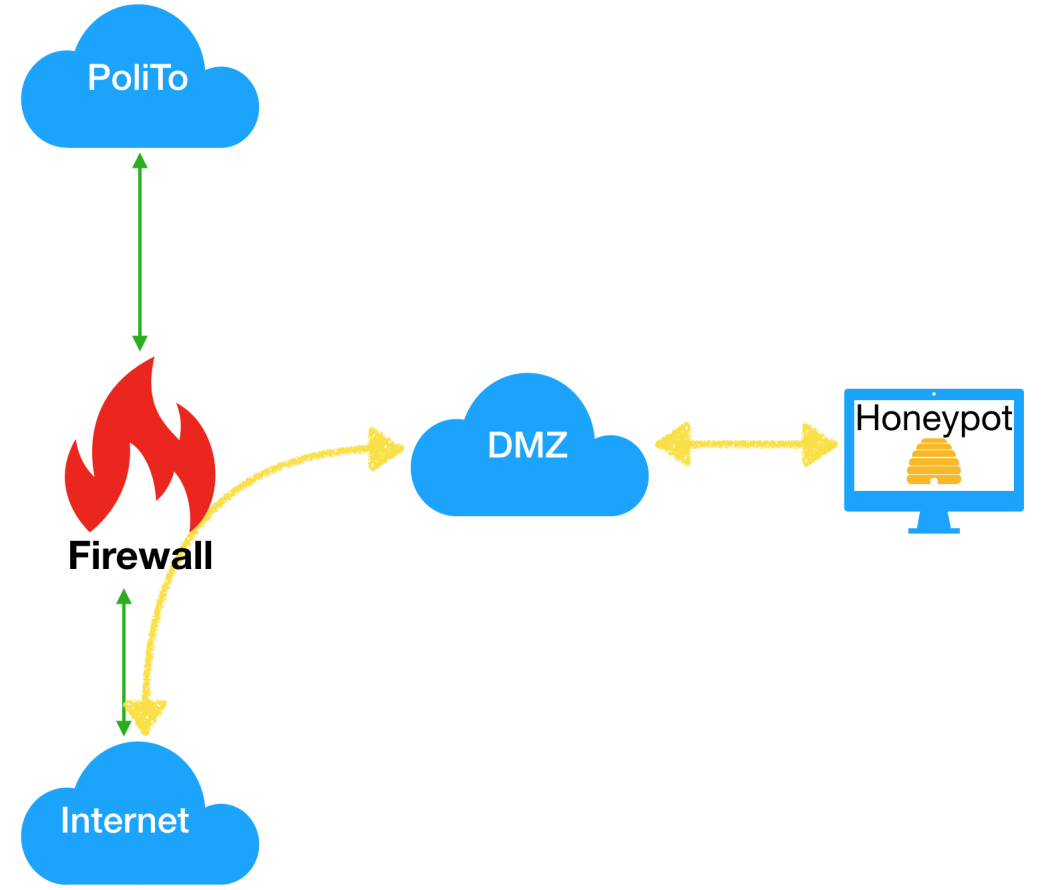

Figure 5: Rete Politecnico di Torino

La rete "privata" del *Politecnico di Torino* risulta essere protetta in ingresso dalla presenza di un Firewall che opera bloccando semplicemente tutte le connessioni non lecite in ingresso (per esempio verso host inesistenti, ma il cui indirizzo è allocato) e quindi scartando tali pacchetti. Questi, però, potrebbero risultare utili per scopi di ricerca, in modo da poter estrapolare dalla loro analisi informazioni sugli attaccanti e sulle strategie adottate per future attività nell'ambito della sicurezza informatica. Per questo motivo si vuole reindirizzare il suddetto traffico verso una zona demilitarizzata (DMZ) ossia una sottorete isolata che, nel nostro caso non offre servizi, ma è adibita all'ascolto dei pacchetti che altrimenti sarebbero droppati dal firewall, pertanto rappresenta il punto migliore in cui porre gli honeypot di cui si è largamente discusso precedentemente: in questa maniera ogni pacchetto indirizzato verso la rete del *Politecnico di Torino*, ma destinato ad un host o una rete allocata e non utilizzata, andrebbe reindirizzato verso la DMZ ed opportunamente trattato al proprio interno. In questa maniera la rete di produzione non è in alcun modo compromessa e contemporaneamente è possibile raccogliere dati utili.

Dal punto di vista implementativo, una prima idea è stata quella di creare una serie di reti LAN opportunamente interconnesse e con un apposito piano di indirizzamento, contenenti macchine fisiche dedicate all'esecuzione di software Honeypot. Il vantaggio di tale struttura è chiaramente dal punto di vista prestazionale: ogni calcolatore risulterà dotato di un "elevato"

quantitativo di risorse e quindi le operazioni da svolgere risultano estremamente veloci, d'altro canto, però, risulta poco efficiente dal punto di vista economico e di gestione, infatti le operazioni che ogni singola macchina deve svolgere sono alquanto semplici e pertanto richiedono una "ridotta" capacità di calcolo (ovviamente dipendente dalla tipologia di Honeypot selezionato), quindi le risorse dedicate ad ognuno di questi sistemi risultano in gran parte inutilizzate. Non meno importante è la questione della gestione, poiché l'amministratore è tenuto ad operare contemporaneamente su un numero elevato di macchine che naturalmente possono subire danni nel corso del tempo o risultare obsolete dopo un certo periodo operativo. Quindi si può evincere che anche la flessibilità è un parametro che viene meno in questo caso, poiché anche la migrazione di una macchina da una rete verso un'altra, risulta un'operazione complicata e costosa dal momento che si è tenuti a "spegnere il servizio" e trasportare fisicamente la macchina nella posizione opportuna.

In conclusione, tale idea non rappresenta un buon tradeoff tra efficienza, costi, flessibilità e semplicità, pertanto tale soluzione è da scartare.

In questi ultimi anni sta prendendo piede la virtualizzazione che oggigiorno non si limita esclusivamente ai sistemi operativi, ma anche alle funzioni di rete, quindi di fatto il software sta gradualmente prendendo il posto dell'hardware. L'idea è quella di astrarre le componenti hardware degli elaboratori al fine di renderle disponibili al software in forma di risorsa virtuale. L'insieme delle componenti hardware virtuali prende il nome di macchina virtuale e su di esse può essere installato qualunque forma di software.

L'idea è quella di utilizzare un numero relativamente ridotto di macchine fisiche grazie al paradigma della virtualizzazione, che permetterà il deploy di Honeypot sottoforma di virtual machines. La potenzialità risiede nella possibilità di istanziare un numero "indefinito" di LAN e di apparati di rete (compatibilmente con le risorse dei calcolatori impiegati) interconnessi tramite topologie arbitrariamente complesse, nonché di Honeypot. Questo garantisce un elevato livello di flessibilità e scalabilità, infatti permette una notevole estensione del progetto a qualunque attività futura semplicemente aggiungendo, a seconda delle necessità, ulteriori macchine fisiche che potranno mettere a disposizione la propria capacità di calcolo senza sforzi eccessivi di configurazione, poiché interamente gestiti dall'ambiente OpenStack.

Un ulteriore vantaggio che tale soluzione fornisce è derivante da una delle features di OpenStack ed in particolar modo da *Glance*: all'interno del sistema è possibile salvare una serie di "sistemi operativi" o più in generale immagini disco che possono essere semplicemente prelevate ed avviate in qualunque posizione della rete virtuale. In questo modo in presenza di macchine virtuali potenzialmente compromesse è possibile sostituirle con delle nuove semplicemente spegnendole ed attingendo dal database di *Glance*: in questo modo si può usufruire di un ambiente pulito e immediatamente pronto per il deploy.

Un primo prototipo concettuale dell'architettura ideata è rappresentato dalla seguente figura:

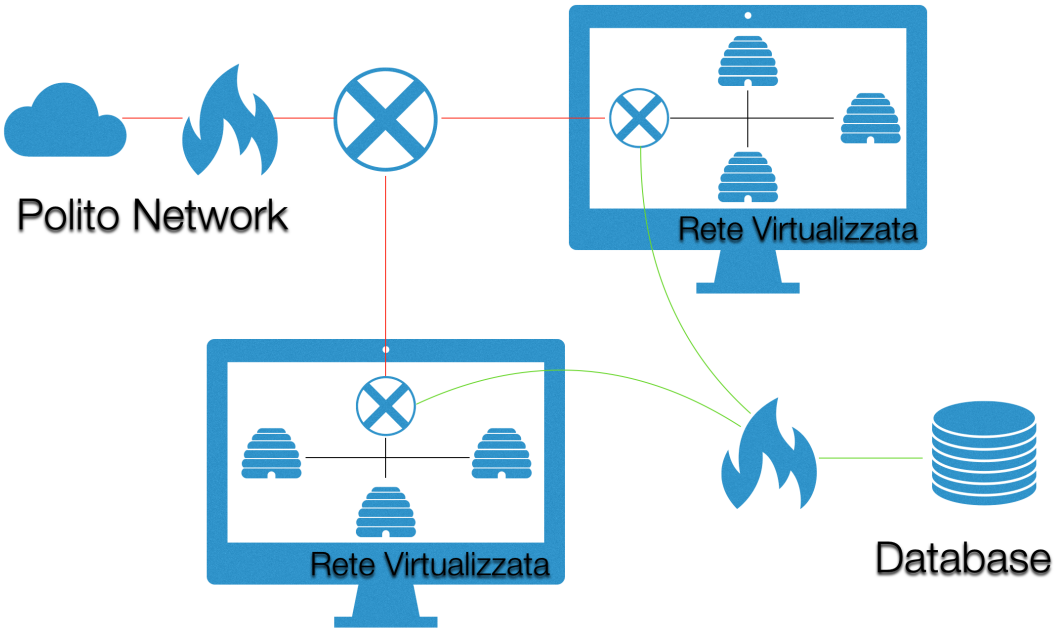

Figure 6: Architettura ideale ad alto livello

Si suppone che ci sia del traffico malevolo proveniente dalla rete Internet e diretto verso la rete del *Politecnico di Torino*. Tale traffico è costretto ad attraversare un firewall che deve possedere una regola al proprio interno che permetta il suo reindirizzamento verso un router non appartenente alla rete di produzione. Il suddetto router è tenuto a forwardare i pacchetti verso una o più macchine fisiche, a seconda dell'implementazione finale. In generale tali macchine fisiche contengono al proprio interno una rete virtualizzata composta da diversi router e LAN con topologia di arbitraria complessità, che permettono al traffico di raggiungere delle particolari virtual machines che sono gli Honeypot. Una possibile estensione è quella di inserire un ulteriore firewall in uscita dalle reti virtuali, in modo da poter salvare in un database esterno le informazioni ottenute dal deployment degli Honeypot.

Data l'architettura proposta, è opportuno valutare se le *virtual machines* siano in grado di gestire il traffico: la scheda di rete fisica potrebbe tranquillamente gestire pacchetti da una Gb Ethernet, ma a causa dei livelli di virtualizzazione presenti sopra di essa tali velocità potrebbero non essere ammissibili, soprattutto se in presenza di hardware non ad elevate prestazioni ma, trattandosi di una rete stub completamente scollegata da quella di produzione, costituirà solo un'appendice dell'intera maglia che dovrà gestire solamente il traffico forwardato dal firewall/router posto a monte tramite un apposita serie di entry statiche nella IP-Table. Il suddetto traffico sarà destinato di conseguenza agli Honeypot che, per definizione, non generano traffico poiché sono entità passive ed i soli pacchetti in transito deriveranno da possibili attacchi provenienti dall'esterno, quali per esempio port-scan o accessi remoti e relative risposte auto-generate. Di conseguenza si può tranquillamente affermare che, dal punto di vista teorico, l'architettura ideata è in grado di gestire il traffico che si prevede di ricevere.

In conclusione, l'obiettivo ideale è quello di implementare un'architettura software simile a quella rappresentata in Fig.4, quindi con un nodo centrale che rappresenta l'hypervisor ed una serie di nodi secondari che costituiscono i compute node. Dal punto di vista fisico ci si aspetta la soluzione rappresentata nella figura seguente:

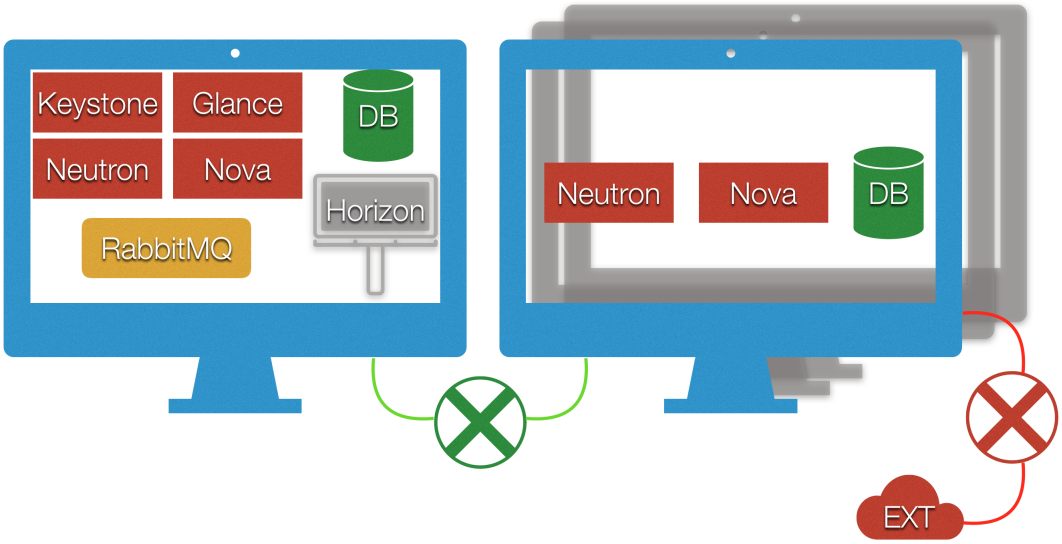

Figure 7: Organizzazione del progetto

3.2 Implementazione

In questa sezione si vuole descrivere il deploy dell'ambiente su un server reale, verso il quale è forwardato il traffico sospetto, ovvero quello indirizzato a reti allocate all'interno del pool del *Politecnico di Torino*, ma inutilizzate.

Il server in questione è dotato di un sistema operativo *Ubuntu Server 16.04* e delle seguenti risorse hardware:

- CPU: Intel(R) Xeon(R) CPU E5-2640 0 @ 2,50GHz x12;
- RAM: 31GB;
- Rete: I350 Gigabit Nework Connection (Intel Corporation) x2.

Per la descrizione dei singoli passaggi, si rimanda alla Sezione 7 e Sezione 8.

Il traffico relativo alle reti attualmente inutilizzate è forwardato verso il server sul quale è stato creato l'ambiente di OpenStack, pertanto è necessario far si che tale traffico sia indirizzato ulteriormente verso la rete virtuale. Questo avviene molto semplicemente aggiungendo opportunamente delle entry nella tabella di routing dell'Hypervisor:

```
route add -net $NET netmask $MASK gw $GATEWAY
```

La topologia implementata per gli esperimenti eseguiti, è la seguente:

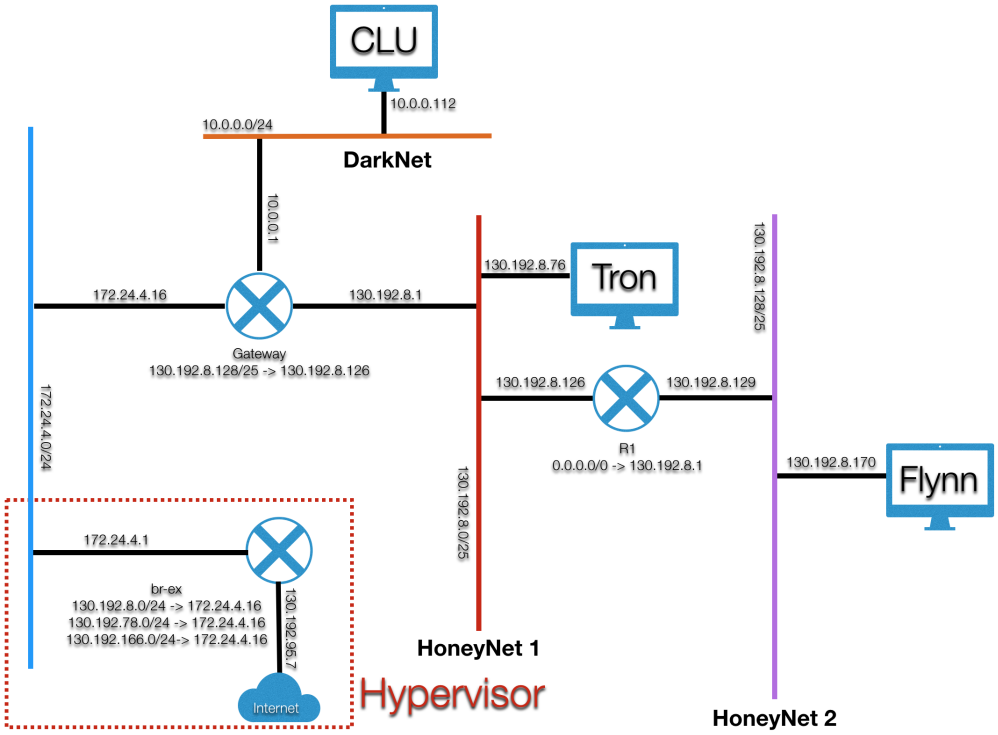

Figure 8: Topologia della rete Honeypot

Il traffico destinato alla rete virtuale, sarà forwardato verso l'interfaccia con IP *172.24.4.16* di *Gateway* che instraderà opportunamente i pacchetti (verso le reti *130.192.8.0/24*, *130.192.78.0/24* e *130.192.166.0/24*, di cui le ultime allocate, ma non ancora utilizzate): per semplicità sono state create due reti /25 a partire dalla rete 130.192.8.0/24 in cui sono stati istanziati alcuni host che sono in grado di scambiare qualunque tipo di pacchetto con la rete pubblica grazie ai loro indirizzi, raggiungibili dall'esterno e alle regole di sicurezza, disponibili su OpenStack, che non ne impediscono il passaggio in entrambe le direzioni:

☐	Direzione	Tipo etere	Protocollo IP	Intervallo porta	Prefisso IP remoto	Gruppo di sicurezza remoto	Description	Azioni
☐	Egress	IPv4	ICMP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola
☐	Egress	IPv4	TCP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola
☐	Egress	IPv4	TCP	1 - 65535	0.0.0.0/0	-	-	Elimina regola
☐	Egress	IPv4	UDP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola
☐	Egress	IPv4	UDP	1 - 65535	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	ICMP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	1 - 65535	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	25 (SMTP)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	53 (DNS)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	80 (HTTP)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	110 (POP3)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	143 (IMAP)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	389 (LDAP)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	443 (HTTPS)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	465 (SMTPS)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	993 (IMAPS)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	995 (POP3S)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	1433 (MS SQL)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	3306 (MYSQL)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	3389 (RDP)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	UDP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	UDP	1 - 65535	0.0.0.0/0	-	-	Elimina regola

Figure 9: Regole OpenStack Honeypot

Stesso discorso vale per quella che potrebbe essere la futura *DarkNet* che ha, al momento, un indirizzamento privato. Le macchine qui istanziate possiederanno un indirizzo privato, derivante dalla rete a cui sono collegate ed uno pubblico, in maniera tale da poter essere raggiungibili. Dal momento che tali host devono esclusivamente rimanere in ascolto, senza rispondere, sono state introdotte le seguenti regole in OpenStack che assolvono a tale compito:

☐	Direzione	Tipo etere	Protocollo IP	Intervallo porta	Prefisso IP remoto	Gruppo di sicurezza remoto	Description	Azioni
☐	Ingress	IPv4	ICMP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Elimina regola
☐	Ingress	IPv4	UDP	Qualsiasi	0.0.0.0/0	-	-	Elimina regola

Figure 10: Regole DarkHosts

Come si può evincere da Fig.8, *Gateway* funge da gateway per l'intera rete e grazie alle rotte statiche, permette il raggiungimento di tutte le reti attualmente istanziate. In futuro risulterà semplicemente necessario aggiungere opportune regole di instradamento per le nuove reti che si istanzieranno. Stesso discorso vale per *R1* che è gateway per la rete *HoneyNet 2* (*130.192.8.128/25*).

4 Testing Methodology

In questa sezione si discuterà dei test effettuati sull'architettura di cui sopra discusso per valutarne le prestazioni e verificare che un attaccante esterno non possa riconoscere la natura virtuale del proprio target.

In letteratura esistono alcuni esperimenti eseguiti sull'ambiente OpenStack e derivati, con l'obiettivo di valutare le prestazioni della rete virtuale in concomitanza con quella fisica. In [7] è indicata una linea guida per il testing che si basa su del lavoro già svolto in precedenza, come descritto in [8, 9].

Come Vmtp [10], si utilizza *Iperf* e *Ping* per valutare il *throughput* e la *latenza* della rete cloud con OpenStack, indicando le prestazioni quando l'intero cloud è soggetto a condizioni di carico variabili: per adempiere a tale scopo, sono stati creati dei programmi in linguaggio C con l'idea di poter essere eseguiti sulle singole macchine, virtuali e non, per generare traffico o creare carico sulla CPU. La scelta di tale linguaggio di programmazione è mirata alla riduzione dell'impatto del programma stesso sulla CPU, poiché è ragionevolmente snello e performante, in favore dei "job" che sono da eseguire. Sono state create due tipologie di programma, la prima che funge da *loader* e l'altra che funge, invece, da *packet generator*:

- i programmi di tipo *loader* ciclano su un vettore di 13 elementi che contiene i valori percentuali di carico da applicare alla CPU, in particolar modo: 0, 10, 20, 30, 40, 50, 60, 70, 80, 85, 90, 95, 100. Per carichi più elevati si è pensato di eseguire dei test più fini, con lo scopo di apprezzare meglio il comportamento in tali regioni, considerate critiche. Ad ogni iterazione è in grado di generare del load sulla CPU di tipo floating point, integer, bit manipulation e control flow, dividendo opportunamente il lavoro sui *core* disponibili. Per ricavare dei risultati che potessero essere il più affidabili possibili, il "job" è eseguito per un intervallo di tempo pari a 205 secondi (3 minuti e 25 secondi). Le suddette iterazioni sono eseguite, poi, un numero di volte pari agli *end-point* che si desidera contattare. Ciò avviene grazie all'invocazione della *system call* "*system()*", a cui è passato come parametro una stringa che segue il pattern:

  ```
  timeout 205 stress-ng -c 0 -l $LOAD
  ```

  Si richiede un timer di 205 secondi sul comando *shell* "*stress-ng*" che genera carico sulla CPU a seconda del valore di "*$LOAD*" (numero intero) passato come parametro (l'opzione *-c* permette di decidere a priori il numero di worker da lanciare, ma con l'opzione 0, si richiede un worker per *core*, il che è un test ragionevole).

- i programmi di tipo *packet generator* seguono un pattern molto simile poiché, dopo un'attesa di 10 secondi per garantire sincronizzazione ed assestamento hardware, eseguono il test per un intervallo di tempo pari a 180 secondi (3 minuti) ed attendono per i successivi 15 secondi. Ciò è ripetuto per 13 volte (numero di carichi sulla CPU) sul medesimo *end-point*, per poi, una volta esauriti i carichi, passare al successivo. Gli utility presi in considerazione sono:

  - *Ping* (Packet internet groper): utilizzato per verificare innanzitutto la raggiungibilità di un dispositivo di rete e per misurare le latenze di trasmissione di rete. Tecnicamente si invia un pacchetto *ICMP echo request* e si rimane in attesa di un pacchetto *ICMP echo reply*. Solitamente infatti la parte di sistema operativo dedicata alla gestione delle reti (stack di rete) è programmata per rispondere automaticamente con un pacchetto di tipo *echo reply* alla ricezione di un pacchetto di tipo *echo request*. Nei *packet generator* per la misura della latenza, si invoca la *system call* "*system()*", a cui è passato come parametro una stringa che segue il pattern:

```
timeout 180 ping $ADDRESS >> $FILE.txt
```

Si richiede un timer di 180 secondi sul comando *shell* "*ping*" che genera precisamente 180 pacchetti *ICMP echo request* che hanno come destinazione "*$ADDRESS*" secondo le impostazioni standard del sistema operativo in questione. L'output con i valori di latenza è rediretto verso un file che sarà successivamente analizzato.

- *Iperf*: utilizzato per la misurazione delle prestazioni della rete. Ha funzionalità client e server e può creare flussi di dati per misurare il *throughput* tra i due *end-point*. L'output Iperf tipico contiene un report con indicazione temporale della quantità di dati trasferiti e il throughput misurato. Su ogni *end-point* target è avviato un server *iperf* tramite:

  ```
  iperf -s
  ```

  che rimane in attesa di connessioni dall'esterno. Nei *packet generator*, quindi, per la misura del throughput, si invoca la *system call* "*system()*", a cui è passato come parametro una stringa che segue il pattern:

  ```
  iperf -c $ADDRESS >> $FILE.txt
  ```

  In questo caso, utilizzando le impostazioni standard, si genera un test di latenza che ha una durata pari a 10 secondi, pertanto per mantenere le tempistiche precedentemente esposte ed ottenere una quantità ragionevole di dati, la *system call* è eseguita 18 volte (180 secondi). Il server da contattare è indicato tramite "*$ADDRESS*" e l'output è rediretto verso un file che sarà successivamente analizzato.

Nelle seguenti sottosezioni si discuterà dei valori ottenuti, comparandoli con quanto ricavato su una macchina con risorse limitate (si veda la sezione 7 per le specifiche di PC1).

5 Risultati

5.1 Test di latenza su server

Loopback Il primo test eseguito sul server è stato effettuato sull'interfaccia di *loopback* (IP 172.0.0.1), in modo da disporre di una baseline per valutare la bontà dei valori ricavati.

Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

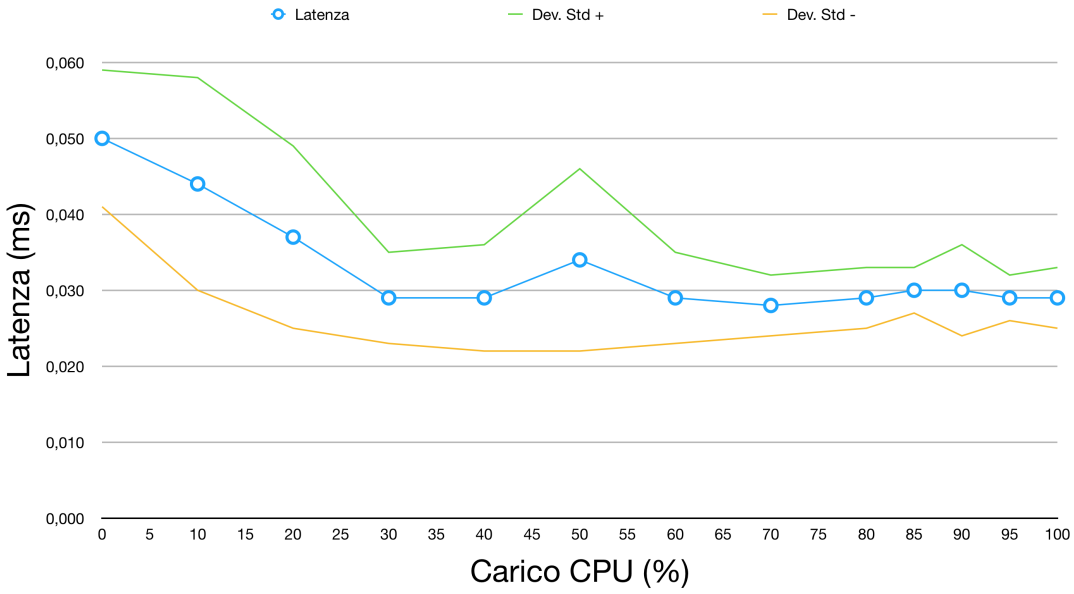

Figure 11: Latenza Loopback

Quanto ottenuto risulta differente rispetto alle aspettative: teoricamente la latenza sarebbe dovuta aumentare con l'aumentare del carico sulla CPU, ma ciò non avviene, infatti si può constatare come la curva si abbassi a partire da un valore iniziale di 0,050ms ad uno finale di 0,029ms. Il valore di latenza risulta quasi dimezzato tra un carico di circa 0% sulla CPU e di 100%. Tale fenomeno potrebbe essere dovuto all'azione di caching eseguita dalla CPU in concomitanza con le politiche di *power saving* e *throttling* che ne riducono le prestazioni in favore di un minore consumo di energia: per percentuali di carico basse, la CPU opera a frequenze più basse rispetto al suo normale funzionamento e si può supporre che non esegua caching, pertanto il codice da eseguire deve essere prelevato da posizioni relativametne "lontane" rispetto al punto di calcolo, comportando una riduzione delle prestazioni, al contrario, per percentuali più elevate, la frequenza operativa aumenta e si è costretti ad eseguire pesantemente caching, e quindi il codice sarà sempre reperibile nella cache della CPU, con conseguente aumento della prestazione. Ulteriori test sul comportamento della CPU sarebbero da svolgere per comprendere se la supposizione sia corretta o meno, ma esulano dal lavoro di cui si desidera discutere in questo documento. La stessa deviazione standard risulta anch'essa molto elevata sui primi test e più bassa sugli ultimi (ad eccezione del 90%). Per ottenere una spiegazione a tale fenomeno, si è valutato la CDF sui vari carichi. Il risultato è il seguente:

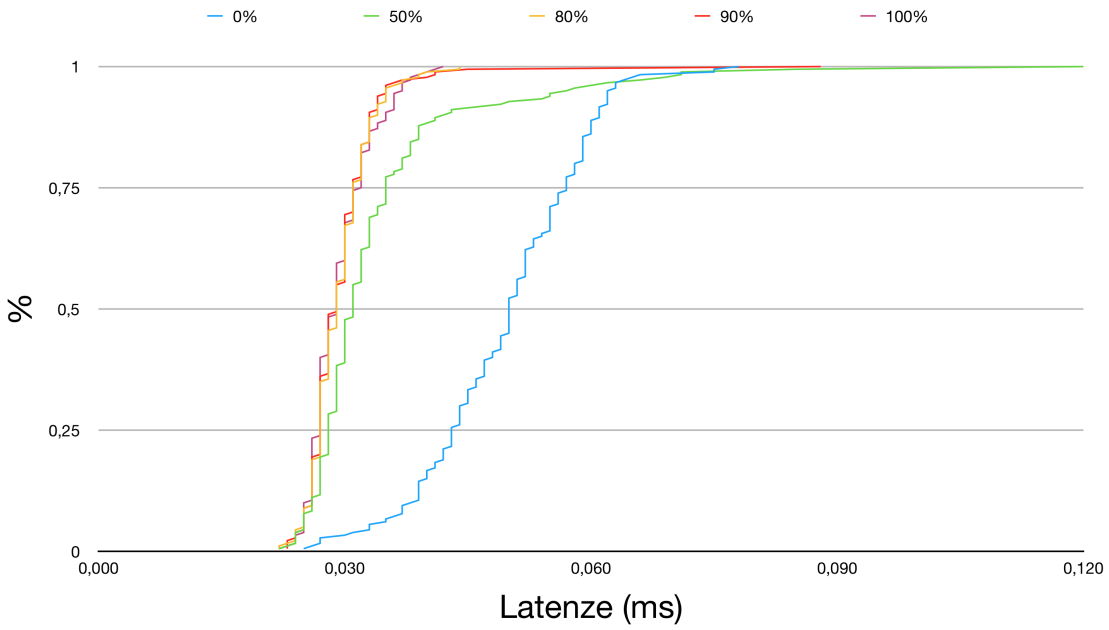

Figure 12: CDF loopback

Quello che si riesce ad evincere è che l'andamento della deviazione standard deriva da valori di latenza occasionali, ma eccessivamente elevati, come dimostrano le code, in particolare per il test con un carico del 50% sulla CPU.

Virtual Machine - Virtual Machine Rimandando alla figura Fig.8, sono prese in considerazione le *virutal machines* "Flynn" (macchina di origine del test) e "Tron" (macchina target). Per eseguire tale test, si è lanciato il programma *loader* sul server fisico e il programma *packet generator* di "ping" su "Flynn". Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

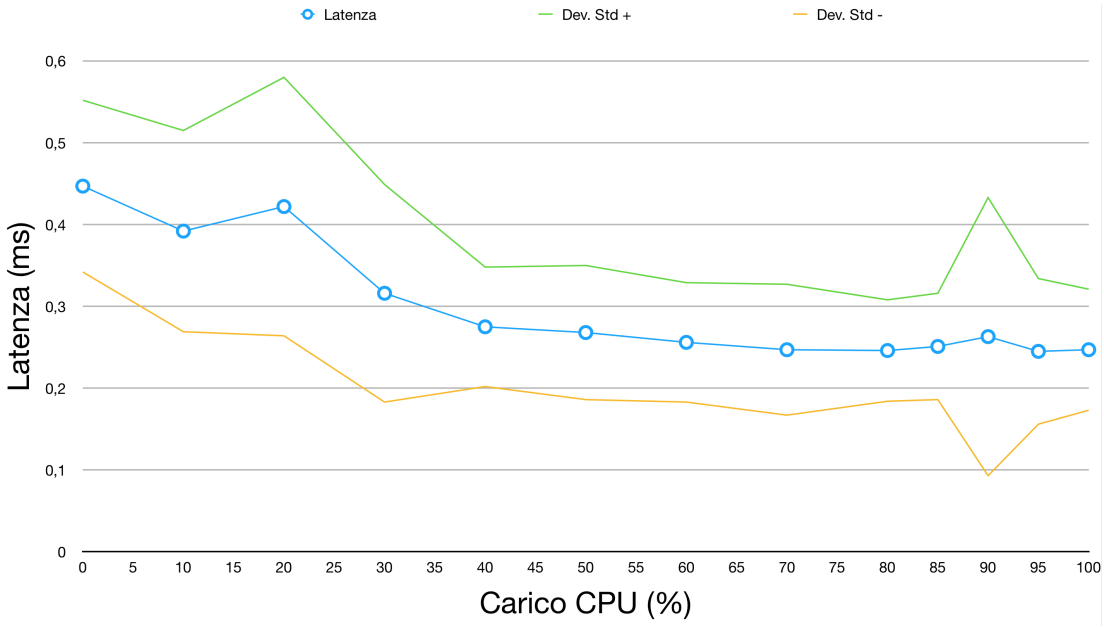

Figure 13: Latenza VM - VM

Anche qui, quanto ottenuto risulta differente rispetto alle aspettative, ma segue il trend del test eseguito sull'interfaccia di *loopback*, infatti si può constatare come la curva si abbassi a partire da un valore iniziale di 0,447ms ad uno finale di 0,247ms. Il valore di latenza risulta, anche in questo caso, quasi dimezzato tra un carico di circa 0% sulla CPU e di 100%. La stessa deviazione standard risulta anch'essa molto elevata sui primi test e più bassa sugli ultimi (ad eccezione sempre del 90%). Per ottenere una spiegazione a tale fenomeno, si è valutato la CDF sui vari carichi. Il risultato è il seguente:

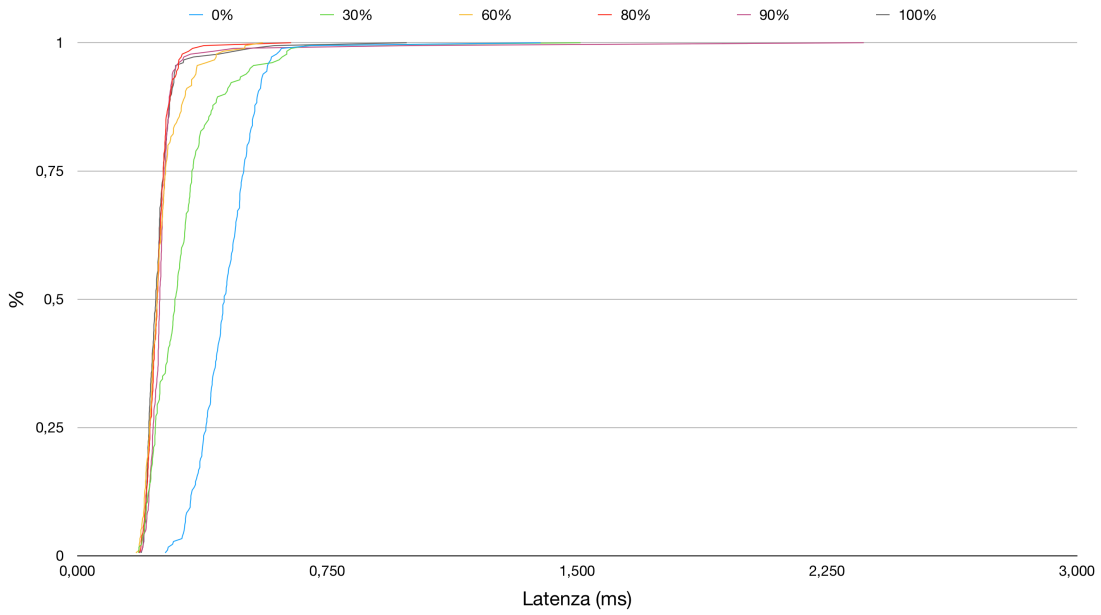

Figure 14: CDF VM - VM

In questo caso le curve risultano ulteriormente esasperate dal momento che possiedono delle code molto lunghe: ciò permetterebbe di avvalorare la tesi esposta per il test sull'interfaccia di *loopback*.

Da notare che i valori di latenza risultano già più elevati rispetto al primo test: ciò è ragionevole poiché i livelli software da attraversare sono numerosi, ma risultano accettabili, se non ottimi in confronto ad una rete fisica.

Virtual Machine - Hypervisor Rimandando alla figura Fig.8, sono prese in considerazione la *virutal machine* "Flynn" (macchina di origine del test) e l'Hypervisor (la macchina fisica ospitante, che funge da target). Per eseguire tale test, si è lanciato il programma *loader* sul server fisico (Hypervisor) e il programma *packet generator* di "ping" su "Flynn". Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

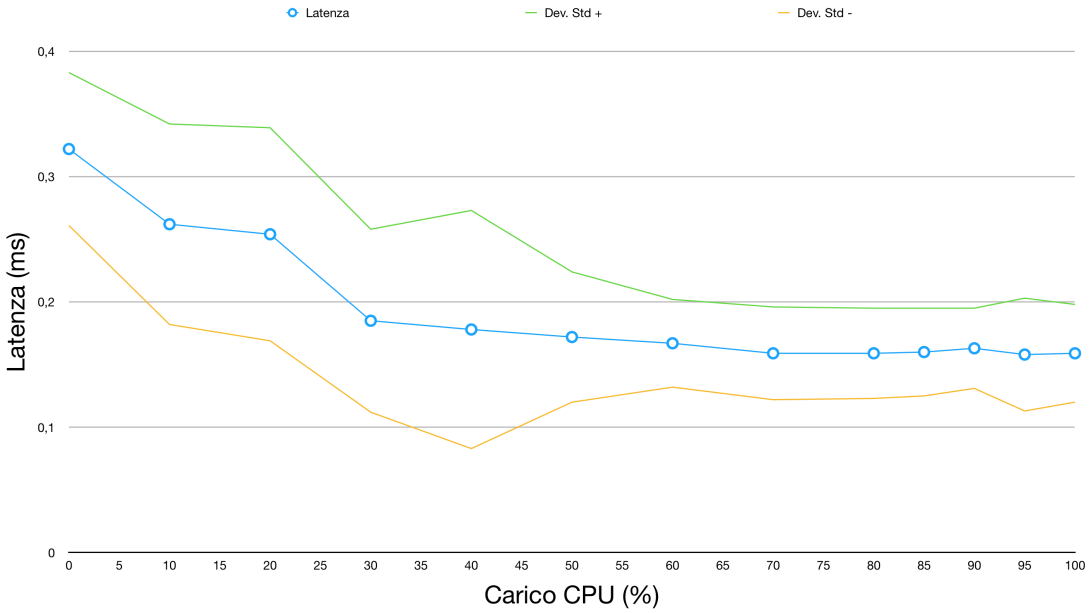

Figure 15: Latenza VM - Hypervisor

Anche qui, quanto ottenuto risulta differente rispetto alle aspettative, ma segue il trend del test eseguito sull'interfaccia di *loopback*, infatti si può constatare come la curva si abbassi a partire da un valore iniziale di 0,322ms ad uno finale di 0,159ms. Il valore di latenza risulta, anche in questo caso, quasi dimezzato tra un carico di circa 0% sulla CPU e di 100%. La stessa deviazione standard risulta anch'essa più elevata sui primi test e più bassa sugli ultimi (ad eccezione in questo caso del 40%). Si noti che i valori di latenza risultano inferiori rispetto al test precedente *virtual machine - virtual machine*: ciò dipende dal percorso dei pacchetti che, molto probabilmente non attraversano completamente l'architettura virtuale, ma passano direttamente all'host fisico. Per giustificare i valori ricavati sulla deviazione standard, si è valutato la CDF sui vari carichi. Il risultato è il seguente:

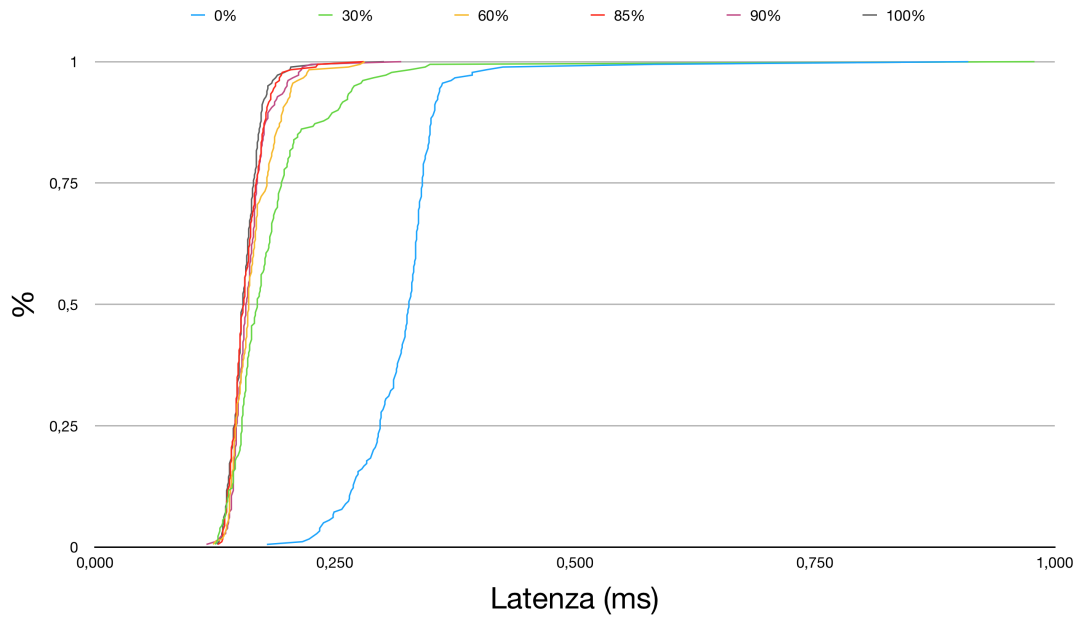

Figure 16: CDF VM - Hypervisor

Virtual Machine - Server in medesimo data center Rimandando alla figura Fig.8, sono prese in considerazione la *virutal machine* "Flynn" (macchina di origine del test) ed una macchina fisica situata nello stesso data center ("a05-bigdata.polito.it", che funge da target). Per eseguire tale test, si è lanciato il programma *loader* sull'Hypervisor e il programma *packet generator* di "ping" su "Flynn". Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

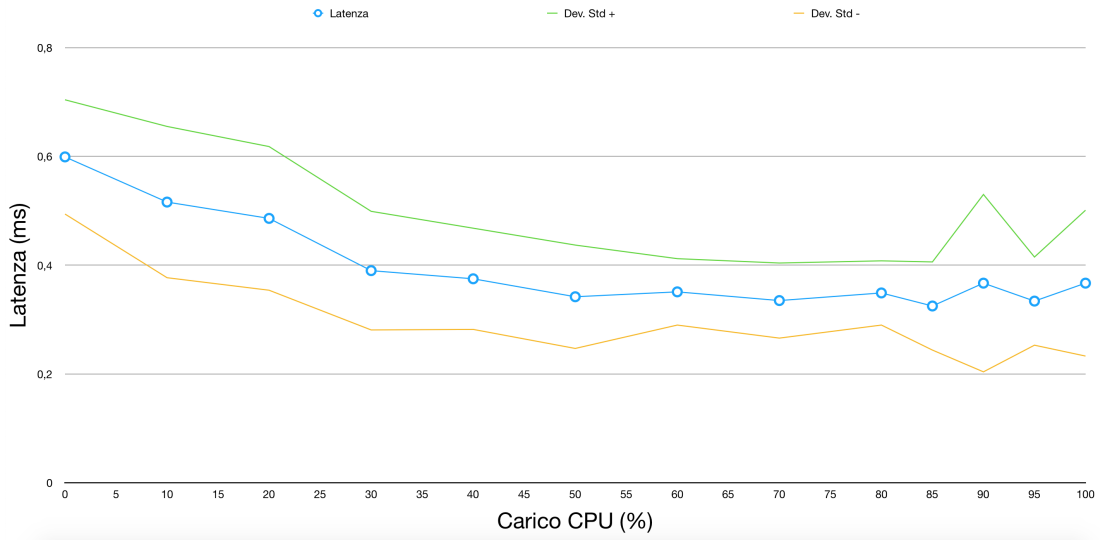

Figure 17: Latenza VM - a05-bigdata.polito.it

Anche qui si segue il trend del test eseguito sull'interfaccia di *loopback*, infatti si può constatare come la curva si abbassi a partire da un valore iniziale di 0,599ms ad uno finale di 0,367ms. In questo caso, però, il comportamento della deviazione standard è leggermente diverso, infatti risulta più "limitata" per bassi carichi di CPU, mentre tende ad aumentare verso il 100%. Per ottenere una spiegazione a tale fenomeno, si è valutato la CDF sui vari carichi. Il risultato è il seguente:

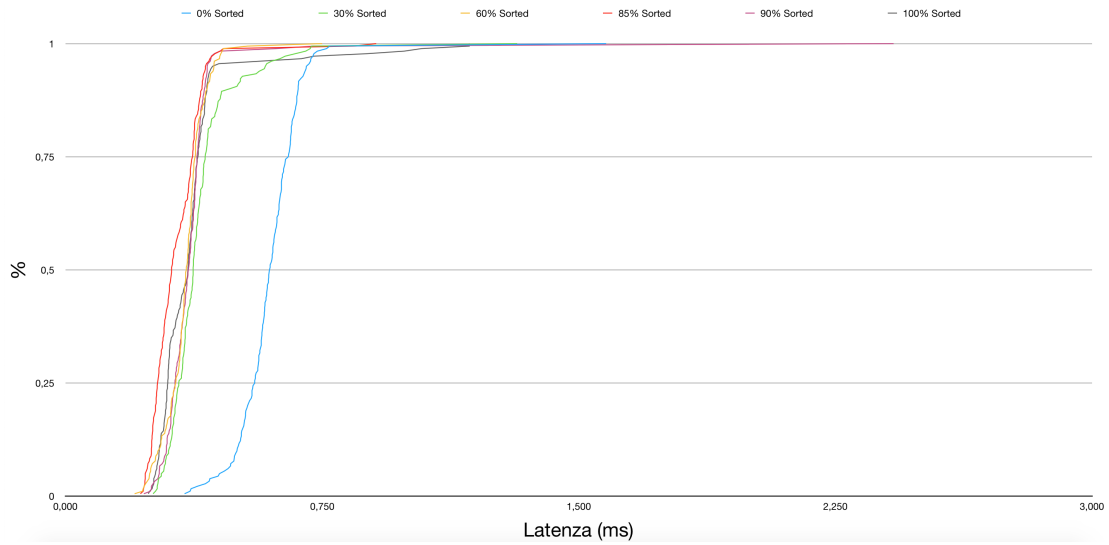

Figure 18: CDF VM - a05-bigdata.polito.it

Quello che si può dedurre è che uscendo dalla macchina, la latenza è influenzata da una serie di fattori esterni e non dai delay interni del server su cui gira l'ambiente di OpenStack.

Virtual Machine - Server remoto Rimandando alla figura Fig.8, sono prese in considerazione la *virutal machine* "Flynn" (macchina di origine del test) ed una macchina fisica situata all'interno della rete del *Politecnico di Torino* (che funge da target) e distante dall'Hypervisor. In questo test si vuole valutare la latenza in presenza di un attraversamento della rete fisica. Per eseguire tale test, si è lanciato il programma *loader* sull'Hypervisor e il programma *packet generator* di "ping" su "Flynn". Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

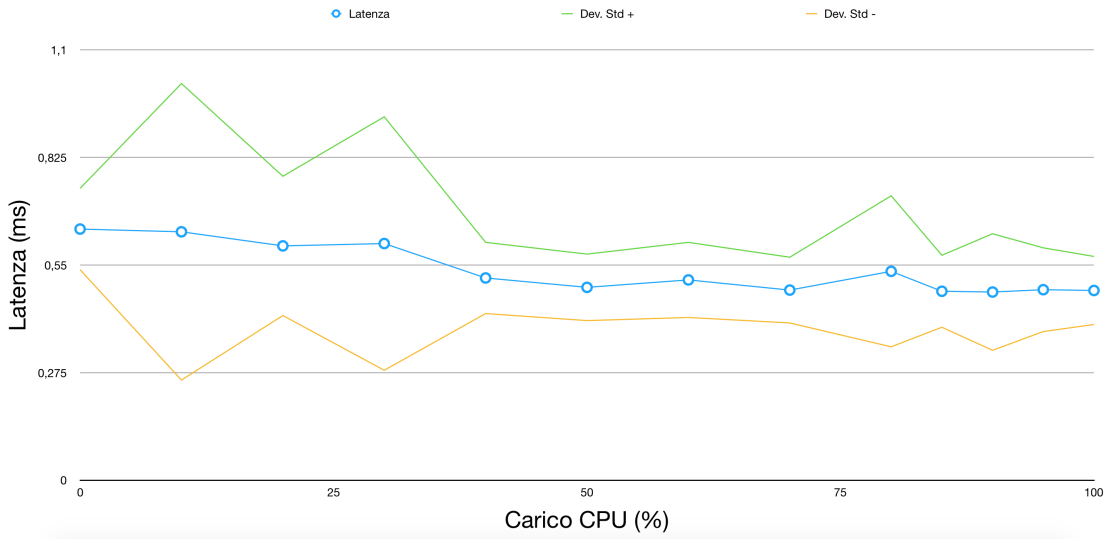

Figure 19: Latenza VM - server remoto

In questo test il risultato è leggermente differente rispetto ai precedenti, poiché sono introdotte una serie di variabili per via della presenza dell'infrastruttura fisica: anche qui si segue il trend del test eseguito sull'interfaccia di *loopback*, ma la curva risulta molto più piatta e stabile poiché non solo i router che instradano il traffico devono eseguire un'operazione di *longest-prefix-match* che è dispendiosa e vanifica in parte l'aumento di prestazione della CPU locale, ma è anche coinvolto l'attraversamento di notevoli distanze sulla rete fisica. Anche la deviazione standard risulta differente rispetto ai test precedenti, poiché è molto più elevata per percentuali basse di carico sulla CPU. Questo fenomeno è probabilmente dovuto alla latenza tipica di una rete cablata che risulta essere molto più stabile e si potrebbe supporre dovuto al caching eseguito dai router attraversati che, nelle prime fasi saranno costretti a fare un *lookup* direttamente in memoria e solo successivamente potranno disporre dell'instradamento direttamente in cache. Si è valutato la CDF sui vari carichi. Il risultato è il seguente:

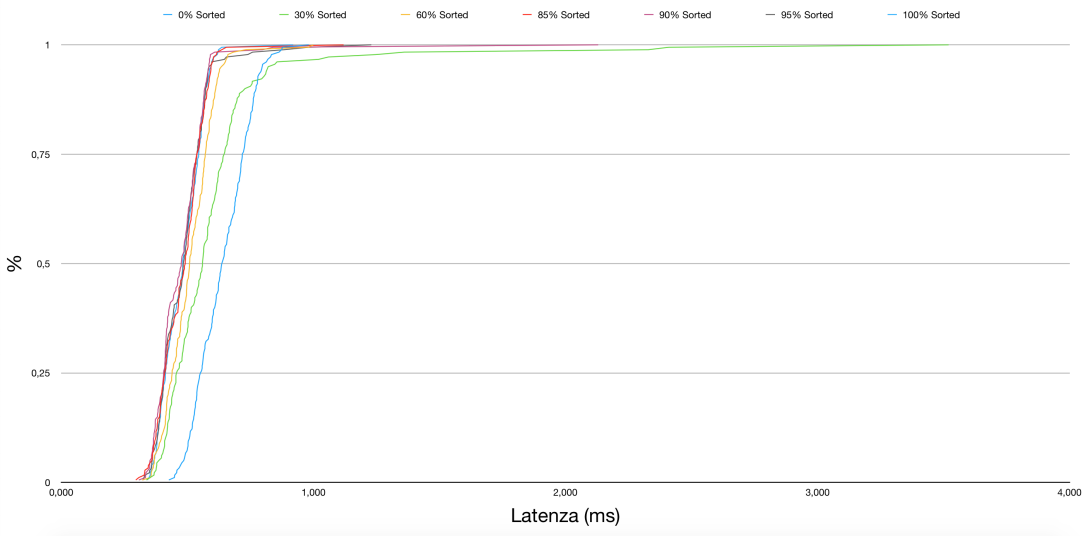

Figure 20: CDF VM - server remoto

Hypervisor - Server remoto Un test estremamente importante per effettaure comparazioni coinvolge l'Hypervisor ed il server remoto precedentemente utilizzato come target. Tale test permette di valutare la bontà e quanto differisce la latenza se la macchina in questione è l'Hypervisor o una *virtual machine* situata al suo interno. Per eseguire tale test, si è lanciato sia il programma *loader* che *packet generator* di "ping"sull'Hypervisor. Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

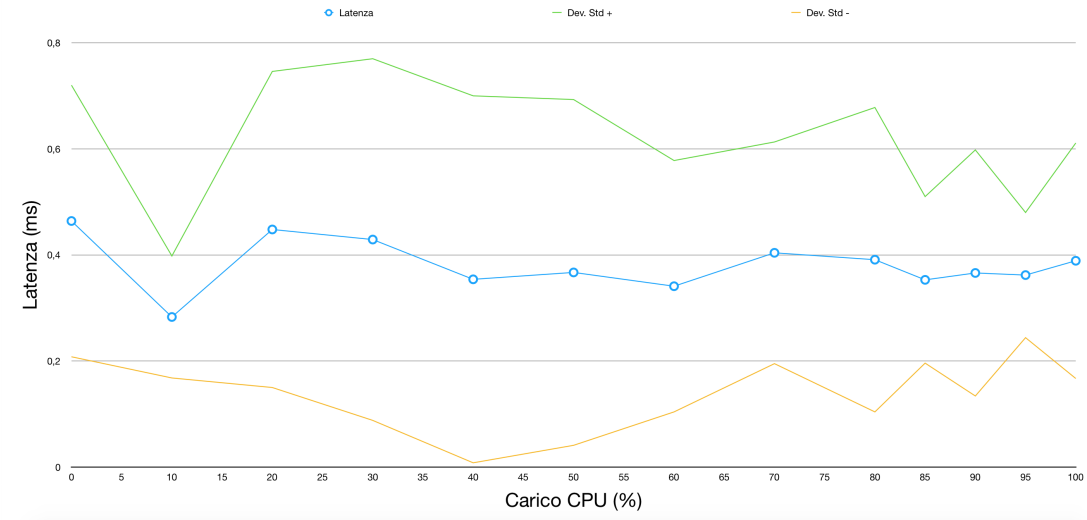

Figure 21: Hypervisor - Server remoto

In questo test, il risultato è coerente con le aspettative, infatti, a meno del valore ricavato al 10%, l'andamento è sostanzialmente costante attestandosi sugli 0,4ms. La deviazione standard

anche qui risulta molto più elevata per percentuali basse di carico sulla CPU, sicuramente non attribuibili alle macchine, ma alla rete. Si è valutato la CDF sui vari carichi. Il risultato è il seguente:

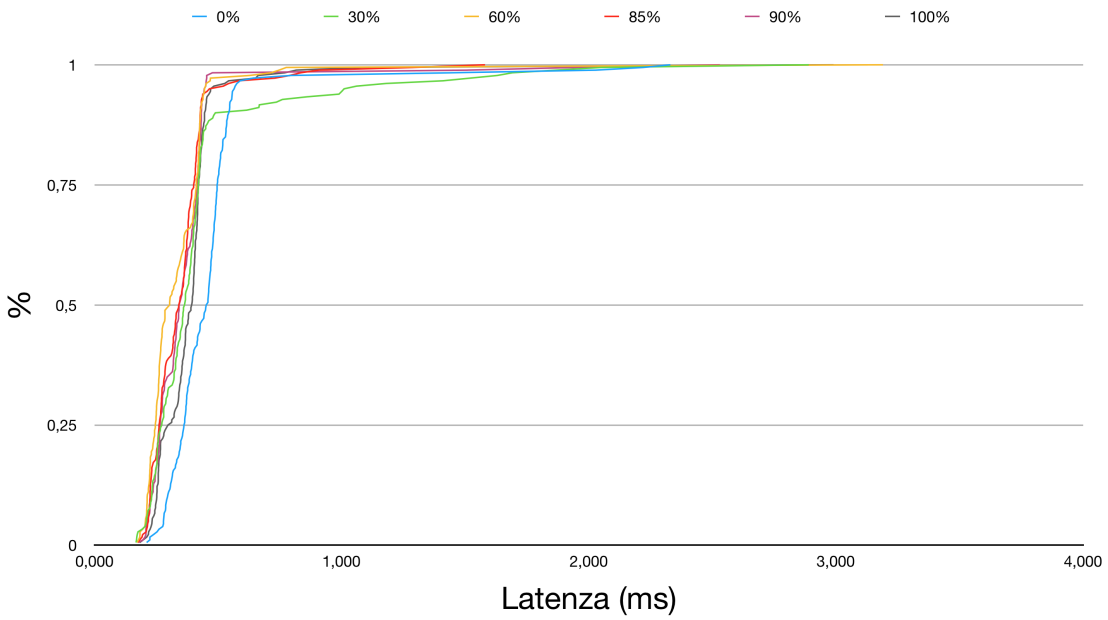

Figure 22: CDF Hypervisor - Server remoto

Risultati A questo punto si analizzeranno tutti i dati raccolti, comparandoli tra di loro per trarre una conclusione preliminare.

Di seguito è riportato un grafico raffigurante le curve ricavate dai test eseguiti, in modo da comparare i valori e verificarne la bontà:

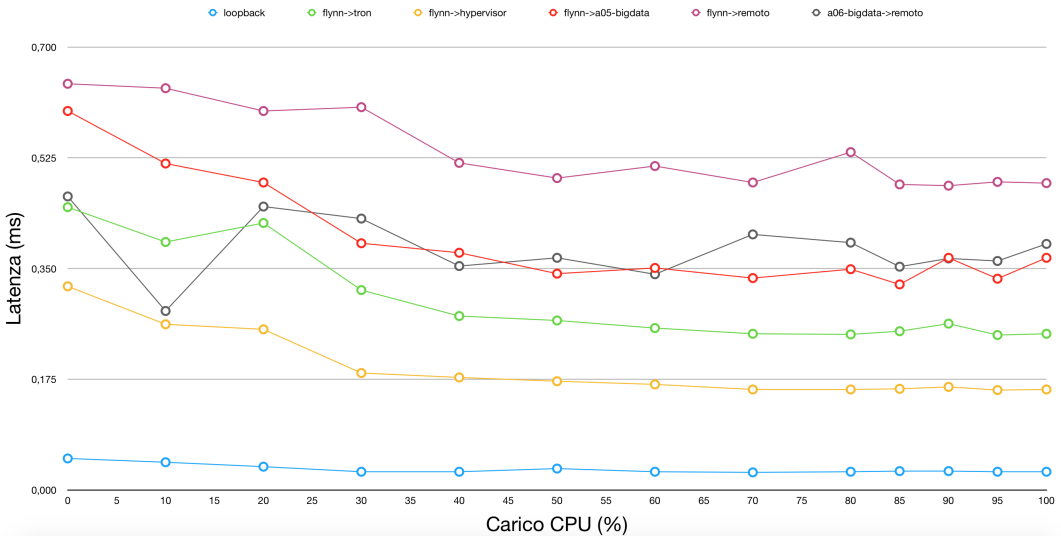

Figure 23: Grafico comparativo delle latenze

Una prima analisi si può fare prima di tutto sull'andamendo delle curve: come evidenziato precedentemente, hanno tutte un andamento decrescente, tendendo ad appiattirsi in condizioni di elevato carico sulla CPU. Si suppone che questo fenomeno sia dovuto al funzionamento della CPU: essa è da considerarsi "dormiente" in presenza di basso carico, quindi può "permettersi" di prelevare le informazioni dalla memoria, piuttosto che effettuare caching, cosa inammissibile, invece, in presenza di una condizione di stress, pertanto per ridurre i tempi di esecuzione, risulta più efficace salvare le informazioni localmente in cache per accedervi più velocemente.

Un aspetto da evidenziare è l'altezza stessa delle curve. Come ci si può aspettare, il test sull'interfaccia di *loopback* presenterà un valore di latenza minimo, poiché è un test locale che coinvolge esclusivamente lo stack protocollare del sistema operativo. Gli altri test presentano, invece, un grafico che tende ad alzarsi proporzionalmente con la "distanza" logico/fisica dell'host target. Un'eccezione, però, è rappresentata dal test *virtual machine - hypervisor*, i cui valori di latenza risultano "paradossalmente" inferiori rispetto al test *virutal machine - virtual machine*. Questo risultato, comunque, ha un suo perché: a livello di sistema operativo, OpenStack non istanzia *virual routers* o *virtual bridge* in corrispondenza 1:1 con l'architettura virtuale implementata, ma genera opportunamente dei TAP, ovvero simulatori di dispositivi di rete a livello di collegamento: invece di ricevere pacchetti da supporti fisici, li riceve da un programma dello spazio utente e invece di inviare pacchetti tramite i supporti fisici li scrive nel programma dello spazio dell'utente. Funziona con i pacchetti di livello 2 come i frame Ethernet, pertanto i pacchetti scambiati tra una qualunque *virtual machine* e l'Hypervisor non dovranno attraversare necessariamente la rete virtuale, indipendentemente dal livello di profondità dell'albero. Questo farebbe giungere quindi alla conclusione che ogni pacchetto che ha come sorgente e destinazione una *virtual machine*, deve prima di tutto essere inviato ad un TAP (quindi verso l'Hypervisor), per poi essere correttamente farwardato verso il programma nello spazio utente opportuno, ovvero il programma ospitante il destinatario.

Un ulteriore aspetto interessante è rappresentato dalla latenza riscontrata nei collegamenti tra *virutal machine - server remoto* e *Hypervisor - server remoto*. Come si può evincere dall'altezza delle due curve, la latenza che coinvolge l'host virtuale è maggiore rispetto a quella con gli host fisici, il che suggerisce la bontà del processo di virtualizzazione: sulla base di tali dati non si può

desumere che l'ambiente sia virtuale.

5.2 Test di latenza su personal computer

Questo ulteriore gruppo di test ripercorre gli stessi passi seguiti in precedenza, ma impiegando una macchina con una quantità di risorse notevolmente inferiori, con l'obiettivo di valutare quanto queste possano influenzare il corretto funzionamento dell'ambiente e poter comparare questi dati con quelli precedentemente ricavati.

L'architettura di riferimento, in questo caso è raffigurata in Fig.45, la cui implementazione è descritta nella Sezione 8.

Loopback Il primo test eseguito è stato effettuato sull'interfaccia di *loopback* (IP 172.0.0.1), in modo da disporre di una baseline per valutare la bontà dei valori ricavati.

Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

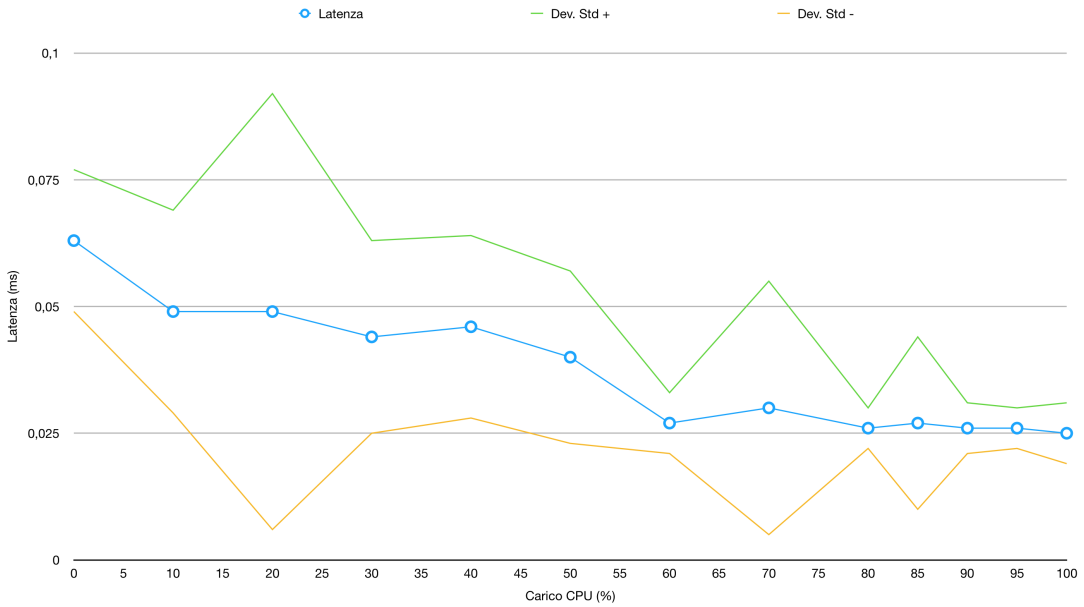

Figure 24: Latenza Loopback PC

Quanto ottenuto risulta in linea con quanto ricavato in precedenza, in termini di pendenza e forma della curva: come ci si aspettava dall'esperienza precedente, il grafico ha un andamento discendente con l'aumentare del carico sulla CPU, partendo da un valore iniziale di 0,063ms ad uno finale di 0,025ms. Anche i valori di deviazione standard risultano in linea, poiché si desume che il caching sia effettuato a partire da un carico di circa 50%: escludendo alcuni picchi al 60% e 80%, dovuti probabilmente a dei *cache-mis*, la deviazione standard tende a convergere verso il valore medio.

Virtual Machine - Virtual Machine In questo test sono prese in considerazione le *virutal machines* "VM1" (macchina di origine del test) e "VM3" (macchina target). Per eseguire tale

test, si è lanciato il programma *loader* sul PC fisico e il programma *packet generator* di "ping" su "VM1". Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

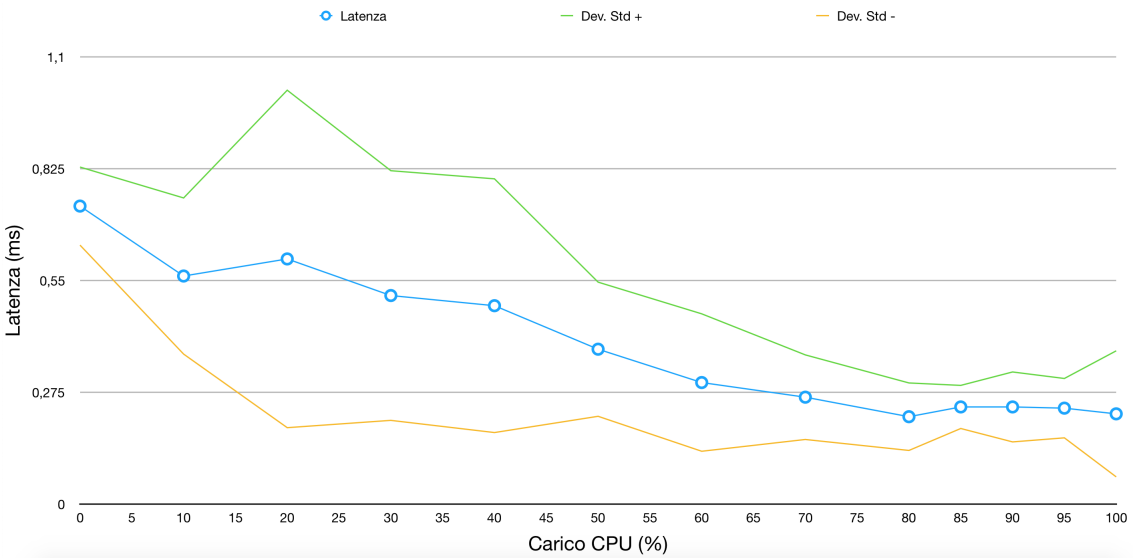

Figure 25: Latenza VM - VM PC

Quanto ottenuto risulta analogo al test eseguito in precedenza in termini di pendenza della curva ricavata: come ci si aspettava il grafico ha un andamento discendente con l'aumentare del carico sulla CPU, partendo da un valore iniziale di 0,795ms ad uno finale di 0,224ms. La differenza sostanziale è evidenziata dalla deviazione standard: essa è molto elevata sui primi punti e tende a ridursi, con un picco al 90% ed al 100% per via del carico elevato generato di per sè dal processo di virtualizzazione. Ciò e sintomo di una CPU non estremamente performante e che, purtroppo, entra in crisi quando i carichi di lavoro risultano eccessivi. Si ricava che l'attività di caching performa al meglio per carichi compresi tra il 50% e l'85%.

Virtual Machine - Hypervisor In questo test sono prese in considerazione la *virutal machine* "VM1" (macchina di origine del test) e l'Hypervisor (la macchina fisica ospitante, che funge da target). Per eseguire tale test, si è lanciato il programma *loader* sul server fisico (Hypervisor) e il programma *packet generator* di "ping" su "VM1". Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

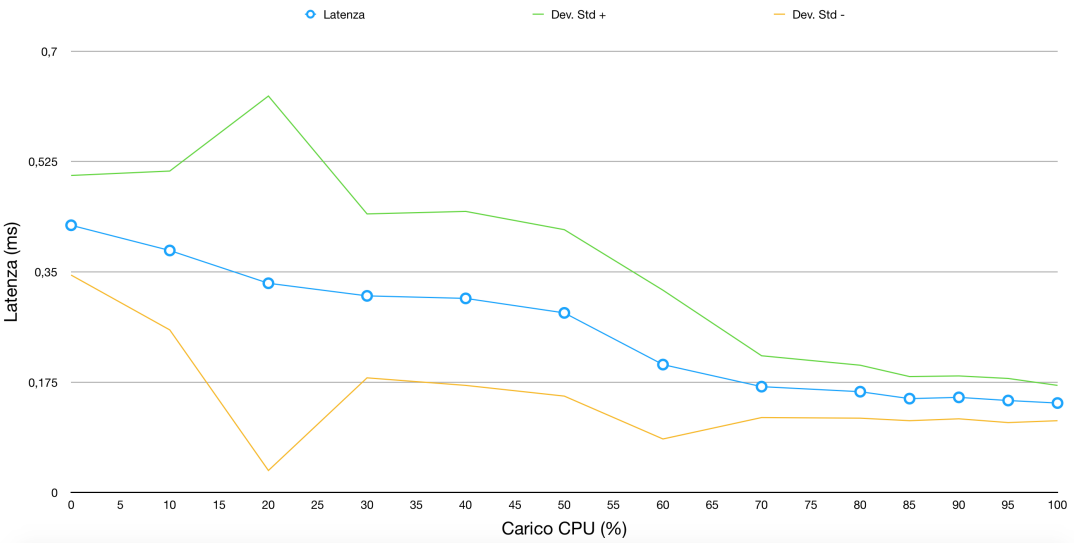

Figure 26: Latenza VM - Hypervisor PC

Quanto ottenuto risulta analogo al test eseguito in precedenza in termini di pendenza della curva ricavata: come ci si aspettava il grafico ha un andamento discendente con l'aumentare del carico sulla CPU, partendo da un valore iniziale di 0,424ms ad uno finale di 0,142ms. Al contrario del test *virtual machine - virtual machine* la deviazione standard tende a convergere verso il valore medio, avvalorando la tesi precedentemente esposta riguardo ai TAP: infatti si può constatare non solo come i valori di latenza siano più bassi, ma anche la deviazione starndard tende a convergere. Ciò, in definitiva, avviene poiché i pacchetti non devono attraversare tutta la rete virtuale, ma sono rapidamente forwardabili verso l'Hypervisor.

Virtual Machine - Server remoto In questo test sono prese in considerazione la *virutal machine* "VM1"(macchina di origine del test) ed il server DNS IPv4 di Google (IP 8.8.8.8, che funge da target), con l'obiettivo di verificare il comportamento nel momento in cui i pacchetti siano costretti ad attraversare la rete pubblica. Per eseguire tale test, si è lanciato il programma *loader* sull'Hypervisor e il programma *packet generator* di "ping" su "VM1". Per ogni percentuale di carico sono stati ricavati 180 valori di latenza, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

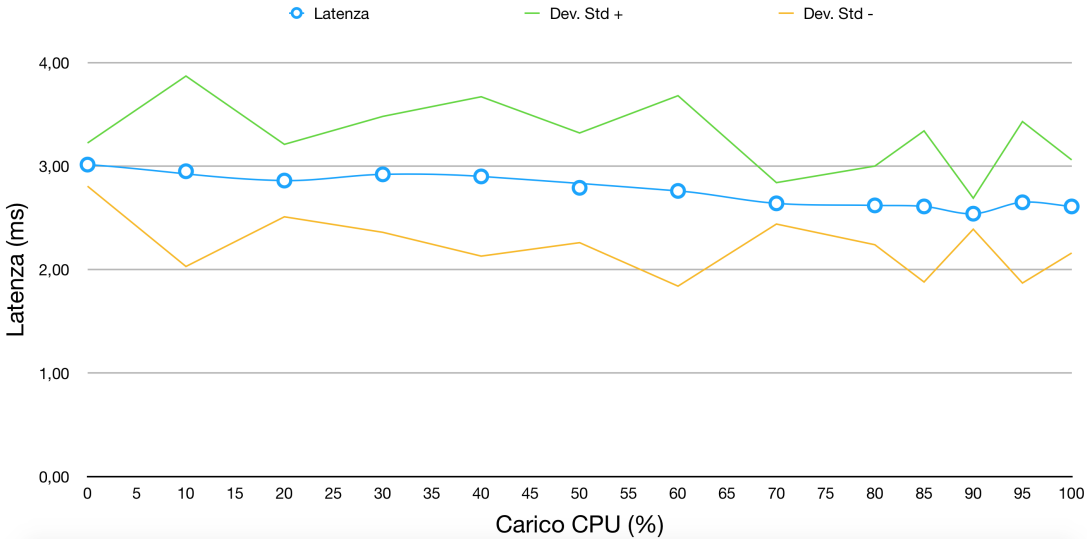

Figure 27: Latenza VM - a05-bigdata.polito.it

Come evidenziato nel test analogo, ma sul server, il risultato è leggermente differente rispetto ai precedenti, poiché sono introdotte una serie di variabili per via della presenza dell'infrastruttura fisica: anche qui si segue il trend del test eseguito sull'interfaccia di *loopback*, ma la curva risulta molto più piatta e stabile poiché i router che instradano il traffico devono eseguire un'operazione di *longest-prefix-match* che è dispendiosa e vanifica in parte l'aumento di prestazione della CPU locale. Anche la deviazione standard risulta differente rispetto ai test precedenti, poiché rimane pressochè costante su tutti i punti presi in considerazione, a parte qualche convergenza nei punti a 20%, 70% e 90% di carico. Questo fenomeno è probabilmente dovuto al caching eseguito dai router attraversati e dalla CPU locale.

5.3 Risultati

In questi ultimi test eseguiti i risultati ricavati risultano analoghi a quanto discusso in precedenza sul server, pertanto qui sarà effettuata un'analisi comparativa sulle prestazioni dei due sistemi, valutandone quanto le risorse influiscano sui valori ottenuti.

Di seguito è riportato un grafico che include le curve delle interfaccie di *loopback* e quelle dei collegamenti *virtual machine* - *virtual machine* e *virtual machine* - *Hypervisor*, che risultano i più interessanti da analizzare:

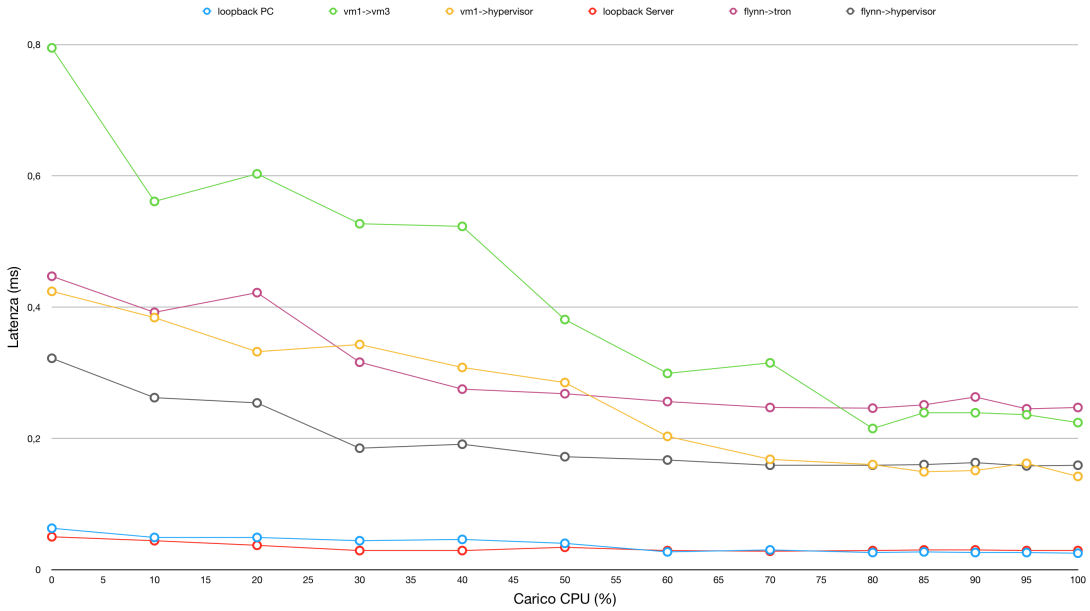

Figure 28: Grafico comparativo delle latenze su Server e PC

Le curve saranno analizzare a coppie:

- *loopback*: si può notare come le due curve quasi si sovrappongano, con la differenza che il server risulta leggermente più costante ma con latenze inferiori rispetto al PC su carichi bassi di CPU e più alte dall'80% di carico in su, nonostante la differenza sia minima.
- *virtual machine* - *virtual machine*: qui si può più facilmente notare come le prestazioni del server risultino più costanti rispetto a quelle del PC. La latenza del primo risulta molto inferiore rispetto a quella del PC, quindi suggerisce una maggiore efficienza ai bassi carichi, mentre per carichi maggiori di 80% il PC risulta più rapido, ma solo di circa 1ms.
- *virual machine* - *Hypervisor*: questo test è sicuramente il più interessante, poiché entrambe le curve sono più basse rispetto ai rispettivi test *virtual machine* - *virtual machine* e risultano pressochè parallele, ma con latenze inferiori da parte del server.

5.4 Throughput

In questa sezione si desidera valurare il throughput dell'intero sistema virtualizzato sul server, in presenza di differenti carichi di lavoro sulla CPU. Ci si aspetta una velocità della rete massima in presenza di carichi bassi, e più lenta con carichi alti. L'obiettivo è quello di verificare in che modo avviene il decadimento di prestazione.

Virtual Machine - Virtual Machine Rimandando alla figura Fig.8, sono prese in considerazione le *virutal machines* "Flynn" (macchina di origine del test) e "Tron" (macchina target). Per eseguire tale test, si è lanciato il programma *loader* sul server fisico e il programma *packet generator* di "iperf" su "Flynn". Per ogni percentuale di carico sono stati ricavati 18 valori di throughput, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

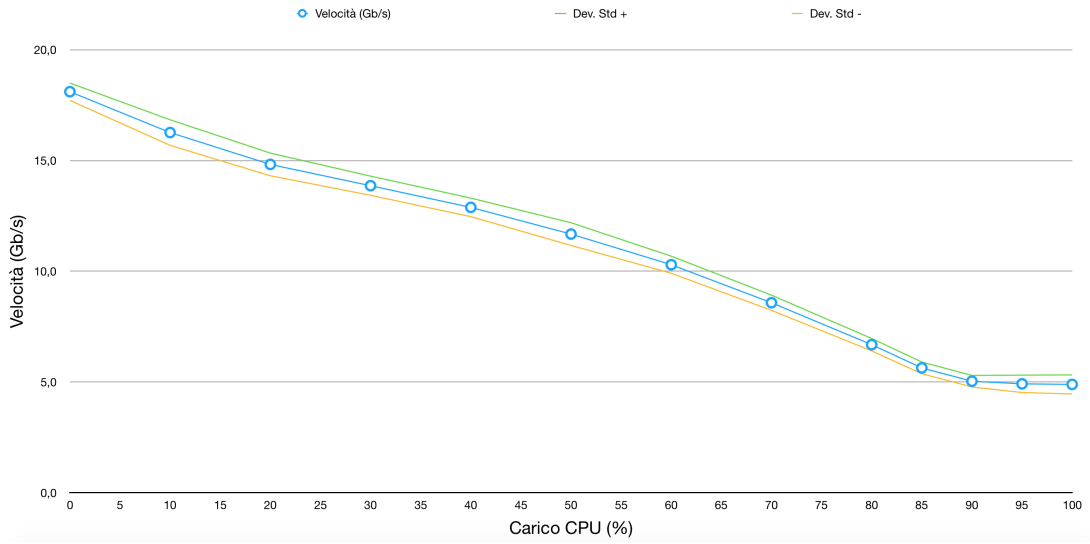

Figure 29: Performance VM - VM

Osservando la curva dei valori medi si può notare come questa decada quasi con un andamento lineare per poi stabilizzarsi tra il 90% ed il 100% di carico su valori tra i 4,89Gb/s e i 5,04Gb/s. Questo corrisponde all'andamento aspettato, poiché più la CPU è impegnata, meno rapide risulteranno tutte le attività: dal momento che i pacchetti sono maneggiati completamente dalla CPU, il degrado è una naturale conseguenza. A conferma di ciò, c'è la deviazione standard che segue l'andamento della curva delle medie, convergendo per valori di carico più elevati.

Virtual Machine - Hypervisor Rimandando alla figura Fig.8, sono prese in considerazione la *virutal machine* "Flynn" (macchina di origine del test) e l'Hypervisor (la macchina fisica ospitante, che funge da target). Per eseguire tale test, si è lanciato il programma *loader* sul server fisico (Hypervisor) e il programma *packet generator* di "iperf" su "Flynn". Per ogni percentuale di carico sono stati ricavati 18 valori di throughput, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

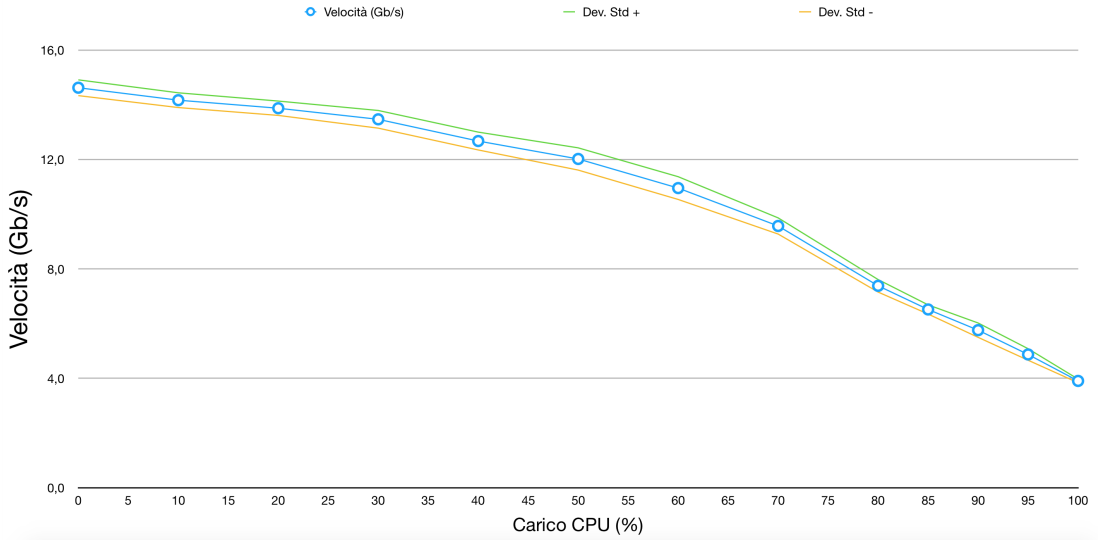

Figure 30: PerformanceVM - Hypervisor

In questo caso la curva ottenuta ha un andamento leggermente diverso, ma sempre in linea con le aspettative: si riescono ad individuare due rette, la prima con pendenza inferiore tra 0% e 70% e una seconda con pendenza maggiore tra 70% e 100%. Ciò suggerisce che il decadimento della prestazione è più lento nella prima parte rispetto alla seconda. La prestazione massima corrisponde a 14,6Gb/s, un valore di 9.57Gb/s sulla soglia di 70% di carico, fino a scendere a 3,90Gb/s con il carico di 100%. Come si può evincere, il decadimento è notevole, ma comunque giustificato per via dell'impegno sulla CPU.

Risultati dell'ambiente virtuale A questo punto si analizzeranno i dati fin qui raccolti, comparandoli tra di loro per trarre una conclusione preliminare.

Di seguito è riportato un grafico raffigurante le curve ricavate dai test eseguiti, in modo da confrontare i valori e verificarne la bontà:

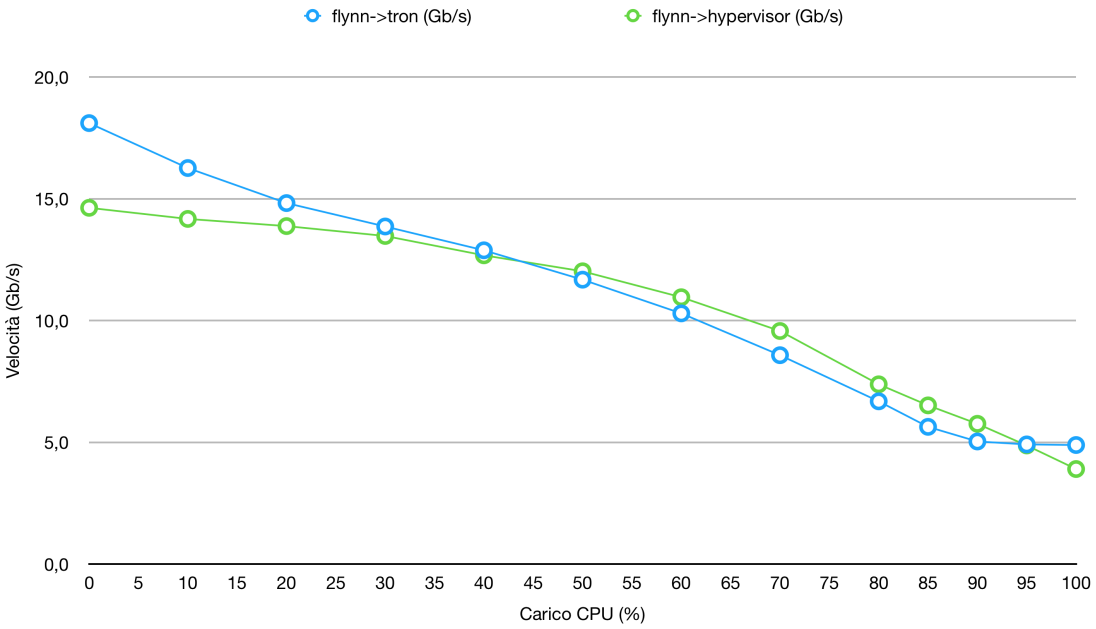

Figure 31: Grafico comparativo del Throughput

Ciò che risulta molto interessante da notare è come per percentuali di carico sulla CPU comprese tra 40% e 95% i test evidenziano performance analoghe (anche se in favore sempre del collegamento *virtual machine - Hypervisor*), a differenza del marcato scarto evidenziato nel test sulle latenze. In fine si può evidenziare come per carichi bassi il collegamento *virtual machine - virtual machine* sia più performante, mentre la situazione si inverte per carichi più elevati.

Virtual Machine - Server remoto Questo test è estremamente importante, poiché permetterà di trarre le conclusioni definitive in merito al raggiungimento degli obiettivi prefissati, pertanto si è pensato di trattarlo separatamente rispetto agli altri.

Rimandando alla figura Fig.8, sono prese in considerazione la *virutal machine* "Flynn" (macchina di origine del test) ed una macchina fisica situata all'interno della rete del Politecnico di Torino (che funge da target) e distante dall'Hypervisor. Per eseguire tale test, si è lanciato il programma *loader* sull'Hypervisor e il programma *packet generator* di "iperf" su "Flynn". Per ogni percentuale di carico sono stati ricavati 18 valori di throughput, di cui si è calcolato un valore medio e la deviazione standard. Tali risultati sono stati raccolti e graficati, ottenendo il seguente risultato:

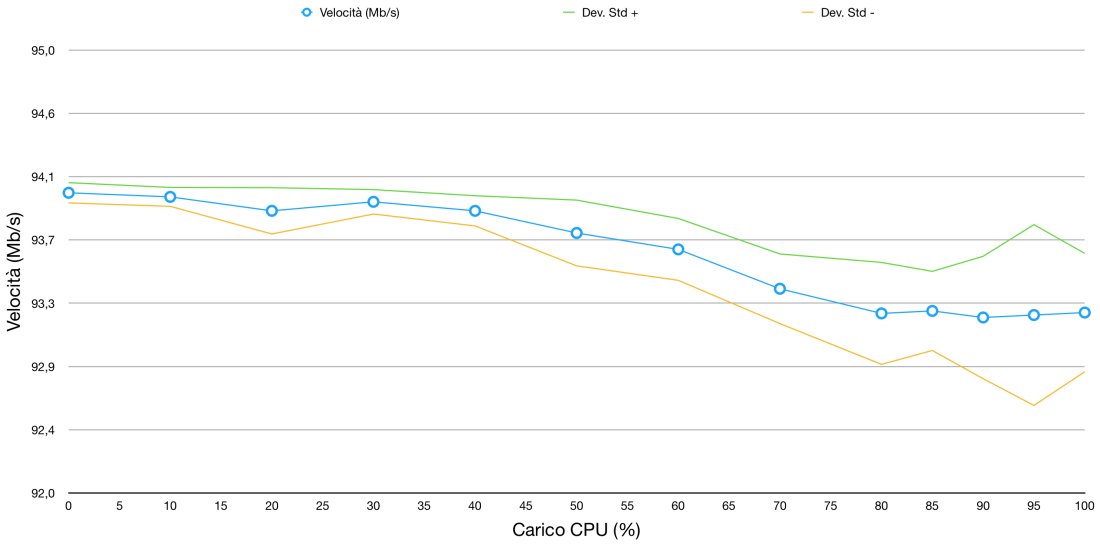

Figure 32: PerformanceVM - Server remoto

Sorprendentemente anche in questo caso si conferma la presenza di un andamento discendente a partire da un valore massimo di 94,0Mb/s fino ad un minimo di 93,6Mb/s, il che non è all'altezza della velocità che ci si sarebbe aspettati sapendo che è attraversata una rete a 100Mb/s per raggiungere la destinazione. Come si può osservare dal grafico seguente, le prestazioni della *virutal machine* risultano inferiori rispetto a quelle ottenute utilizzando come sorgente l'hypervisor (che raggiunge velocità costanti intorno ai 94,4Mb/s):

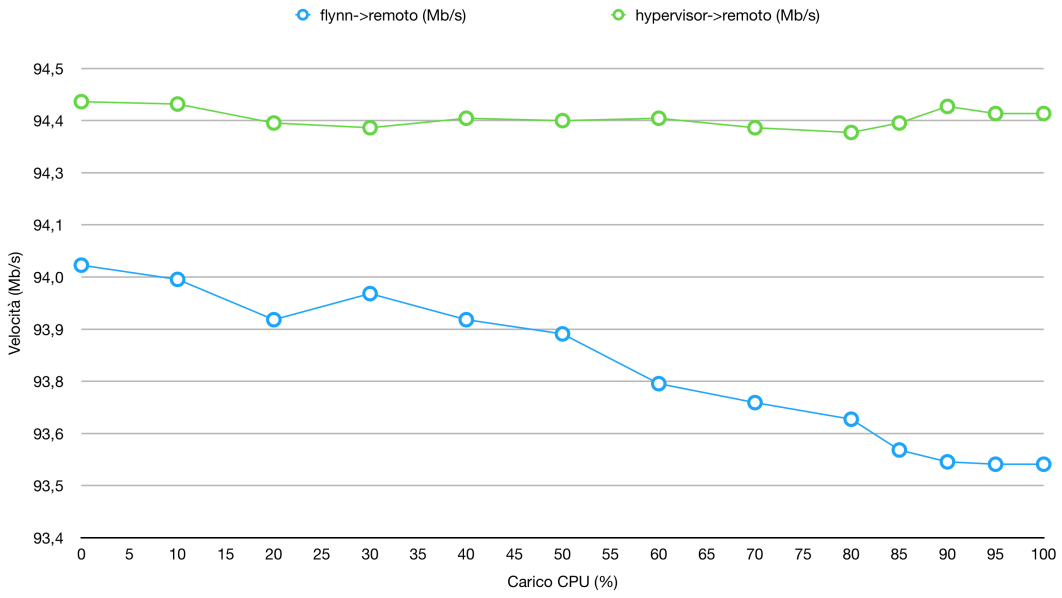

Figure 33: Performance VM rispetto ad Hypervisor

Di conseguenza, teoricamente era atteso un andamento più costante ed indipendente dal carico di CPU applicato, vista la notevole differenza di prestazione tra la rete virtuale e fisica. Una spiegazione a tale fenomeno è maggiormente comprensibile se si analizzano i pacchetti in transito sulle interfacce a monte ed a valle di "*br-ex*" con i diversi carichi sulla CPU tramite comandi del tipo:

```
sudo tcpdump -i br-ex port not 22 and host $TARGET_HOST -v -w $FILE_BR.pcap

sudo tcpdump -i bond0 port not 22 and host $TARGET_HOST -v -w $FILE_BOND.pcap
```

Ciò che si riesce a constatare è che le due tracce sono perfettamente identiche, quindi è possibile ritenere che il *virtual router* in questione non sia la causa della perdita di prestazione rispetto alla rete fisica. Risulta più probabile che all'interno di OpenStack sia presente uno o più componenti che siano dotati di code troppo piccole e, conseguentemente, non in grado di contenere tutti i dati in transito vista la presenza del "collo di bottiglia". Ad avvalorare ciò ci sono una serie di pacchetti ritrasmessi, di cui nelle tracce mancano quelli inviati originariamente: questo lascia intendere che la congestione, e quindi la perdita di pacchetti, si verifichi a monte. Conseguentemente l'aumento di carico comporta un rallentamento delle operazioni e quindi la perdita di pacchetti diventa più frequente e conseguentemente più costosa.

A questo punto si è provveduto ad un'analisi più profonda su tutto l'aspetto "retistico" dell'architettura, cercando di ottenere informazioni su tutti gli apparati istanziati. Dopo un'accurata documentazione sulle funzionalità offerte dalla CLI di OpenStack (OpenStackClient) [11] si è provveduto all'installazione della stessa [12]. Il primo passo è stato quello di ottenere gli ID interni di tutti gli apparati di rete istanziati:

```
openstack router list
+--------------------------------------+----------+--------+
| ID                                   | Name     | Status |
+--------------------------------------+----------+--------+
| 23722bb9-2b07-41de-9969-818ee4f987be | Gateway  | ACTIVE |
| 685bfe9c-d6f0-456a-9c50-9dcd7788ece7 | router1  | ACTIVE |
| 6b6d15cd-9eed-4abc-bf1d-24de6e91f14f | R1       | ACTIVE |
+--------------------------------------+----------+--------+
```

Tali ID sono stati utilizzati per accedere alle proprietà dei singoli router, verificandone i parametri di default.

La configurazione di default di R1, ad esempio, è la seguente:

```
sudo ip netns exec qrouter-6b6d15cd-9eed-4abc-bf1d-24de6e91f14f ifconfig
lo  Link encap:Local Loopback
    inet addr:127.0.0.1  Mask:255.0.0.0
    inet6 addr: ::1/128 Scope:Host
    UP LOOPBACK RUNNING  MTU:65536  Metric:1
    RX packets:0 errors:0 dropped:0 overruns:0 frame:0
    TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1
    RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
qr-03e198c7-f2  Link encap:Ethernet  HWaddr fa:16:3e:f0:80:3f
                inet addr:130.192.8.126  Bcast:130.192.8.127  Mask:255.255.255.128
                inet6 addr: fe80::f816:3eff:fef0:803f/64 Scope:Link
                UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
                RX packets:151002668 errors:0 dropped:0 overruns:0 frame:0
                TX packets:130088671 errors:0 dropped:0 overruns:0 carrier:0
                collisions:0 txqueuelen:1
                RX bytes:7224765585 (7.2 GB)  TX bytes:6144110401749 (6.1 TB)
qr-39e6f3ea-1d  Link encap:Ethernet  HWaddr fa:16:3e:4e:79:97
                inet addr:130.192.8.129  Bcast:130.192.8.255  Mask:255.255.255.128
                inet6 addr: fe80::f816:3eff:fe4e:7997/64 Scope:Link
                UP BROADCAST RUNNING MULTICAST  MTU:1450  Metric:1
                RX packets:118209429 errors:0 dropped:0 overruns:0 frame:0
                TX packets:137693126 errors:0 dropped:0 overruns:0 carrier:0
                collisions:0 txqueuelen:1
                RX bytes:6141462263667 (6.1 TB)  TX bytes:8505572918 (8.5 GB)
```

Ciò che è risultato alquanto anomalo, è il parametro "*txqueuelen*" che rappresenta il numero di pacchetti bufferizzabili per l'invio. In questo caso il valore è pari a 1, ovvero può essere salvato nel buffer un solo pacchetto alla volta. Per avere conferma di tale "anomalia", si è controllato il valore del suddetto parametro sull'interfaccia fisica dell'hypervisor: qui il valore è pari a 1000. Analizzando la configurazione, qui omessa per semplicità, dei router "Gateway" e "br-ex" (ovvero quelli attraversati dai pacchetti per raggiungere l'host target) si è constatato che anche in questi apparati l'impostazione di default per il "txqueuelen" è pari a 1. Una prima supposizione a cui si può giungere è che un buffer così ridotto potrebbe essere la causa della perdita di prestazione precedentemente riscontrata nei confronti della macchina fisica. Per trovare un fondamento a tale ipotesi e fornire una risposta adeguata al fenomeno riscontrato, è stato eseguito un test di throughput con l'utility "iperf" senza porre carichi sulla CPU ma aumentando progressivamente la dimensione del "txqueuelen" su "br-ex", "Gateway" ed "R1" rispettivamente tramite:

```
sudo ifconfig br-ex txqueuelen $VALUE
sudo ip netns exec qrouter-$ROUTER_ID ifconfig $INTERFACE_ID txqueuelen $VALUE
```

dove "$VALUE" rappresenta la dimensione da attribuire al buffer (come numero di pacchetti), "$ROUTER_ID" rappresenta l'ID interno del router ed "$INTERFACE_ID" è l'identificativo dell'interfaccia del router interessato (acquisibile tramite il comando indicato per il router R1 in precedenza). Sono stati scelti i seguenti valori per "txqueuelen": 1, 2, 5, 10, 20, 100, 250, 500, 1000, con lo scopo di apprezzare al meglio l'andamento della curva ottenuta dalla media delle 18 misure di throughput effettuate per ogni caso selezionato:

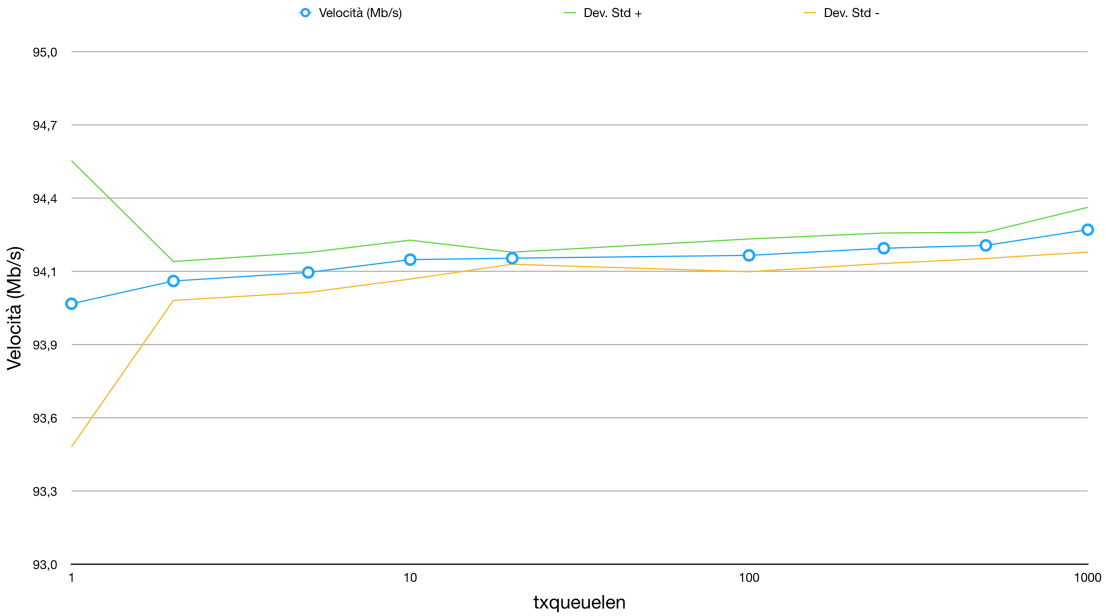

Figure 34: Throughput con incremento del buffer ed assenza di carico sulla CPU

Il grafico ottenuto permette di confermare quanto supposto, infatti con un buffer troppo piccolo si rileva non solo una bassa prestazione, ma anche un'elevata variabilità (indicata dai valori di deviazione standard) dovuti ad un eccessivo numero di ritrasmissioni. Aumentando progressivamente la dimensione di "txqueuelen", si riesce ad apprezzare come i valori non solo risultino più stabili, ma soprattuto si ha un incremento del throughput che raggiunge quasi lo stesso valore ottenuto dall'hypervisor con il valore di 1000. Per questo motivo, ciò è da considerarsi un risultato ottimo dal momento che si è anche individuato il motivo per il quale i valori ottenuti inizialmente fossero più bassi del normale.

Alla luce di quanto scoperto, è stato eseguito il medesimo test ma applicando un carico del 100% sulla CPU, così da apprezzare il comportamento anche in questa situazione estrema:

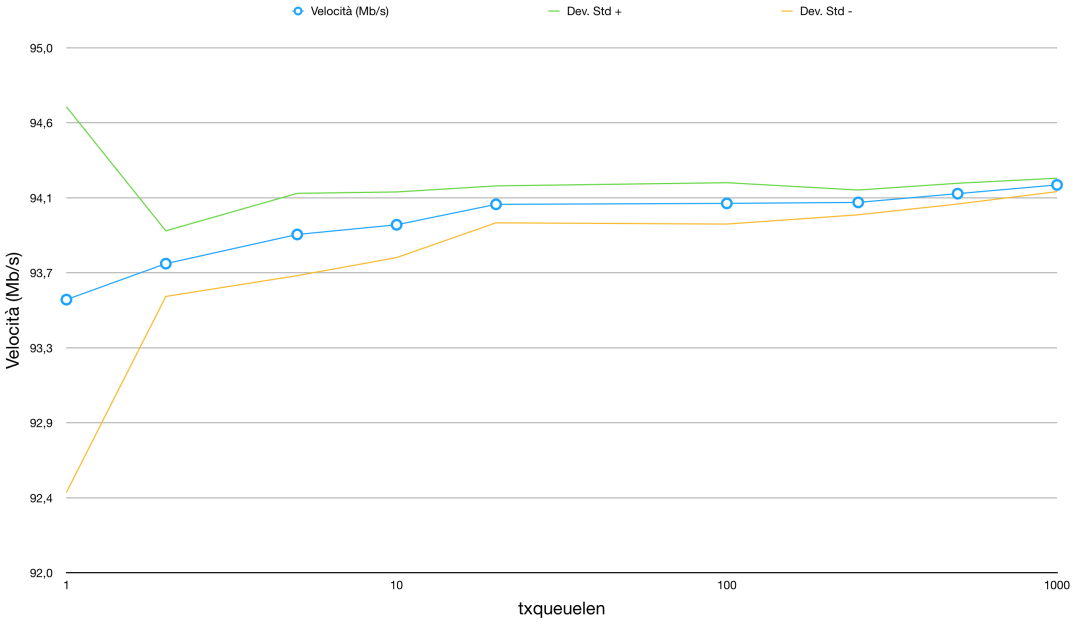

Figure 35: Throughput con incremento del buffer con carico del 100% sulla CPU

Come ci si aspettava dal risultato del test precedente, anche qui le prestazioni aumentano in concomitanza con l'incremento della dimensione del buffer, ottenendo inizialmente dei valori poco stabili, mentre successivamente, con l'aumento di "txqueuelen", questi tendono ad avvicinarsi al rispettivo valore medio. Nonostante tale modifica, il throughput risulta ovviamente inferiore rispetto al test precedente semplicemente perché la CPU è eccessivamente occupata e quindi le operazioni di trasferimento dei pacchetti nell'ambiente virtuale sono rallentate rispetto alla condizione di assenza di carico.

Quando detto è rappresentato nel grafico seguente:

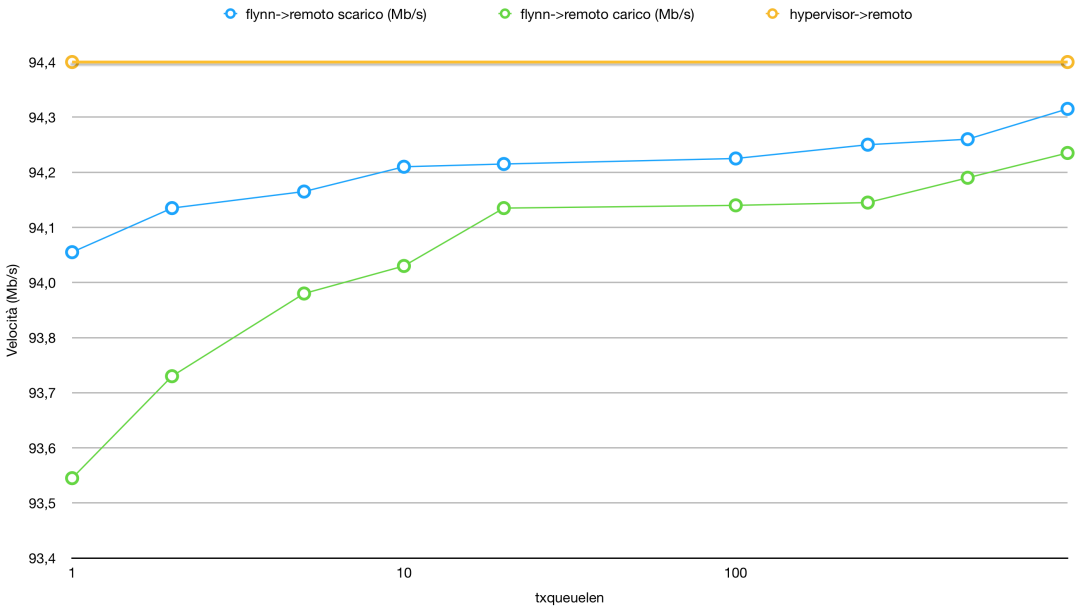

Figure 36: Rapporto throughput con macchina fisica

In aggiunta a quanto osservato e discusso fin qui, è stato analizzato il comportamento dell'ambiente virtuale nell'interfacciamento con una rete a 1Gb/s: è stato selezionato un server del Politecnico di Torino appartenente allo stesso cluster e connesso tramite una Gb Ethernet. Sul server in questione è stato avviato "iperf" in modalità "ascolto", mentre su "Flynn" sono stati eseguiti i test seguendo il pattern già ampiamente discusso in precedenza con l'aumento progressivo della "txqueuelen": in questo caso è stato selezionato solo un sottoinsieme delle dimensioni proposte, per rendere le variazioni maggiormente apprezzabili. La prima verifica è stata eseguita in assenza di carico sulla CPU e con buffer di dimensione 1, 2, 5, 10, 20, 500, 1000:

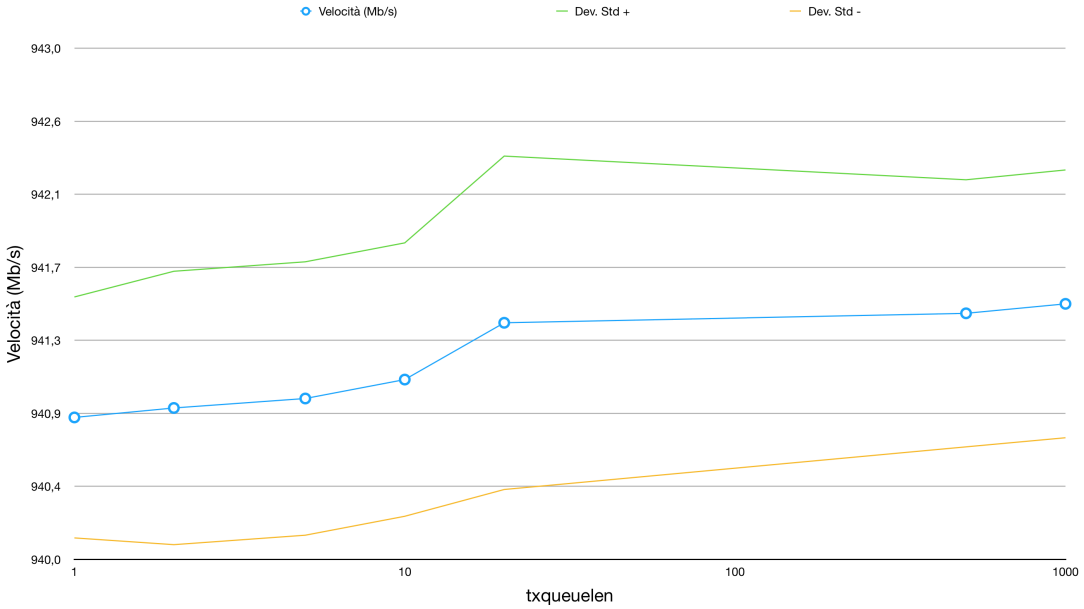

Figure 37: Throughput con incremento del buffer ed assenza di carico sulla CPU

Ciò che è molto interessante da notare è innanzitutto come per una finestra compresa tra i valori 1 e 20 si abbia un incremento prestazionale a partire da 940,8Mb/s con andamento parabolico, per poi attestarsi con un andamento quasi asintotico sui 941,5Mb/s. In aggiunta si può constatare come la deviazione standard sia costantemente compresa tra gli 0,7Mb/s e gli 0,8Mb/s.

Lo stesso test è stato eseguito imponendo un carico massimo sulla CPU, ottenendo il seguente risultato:

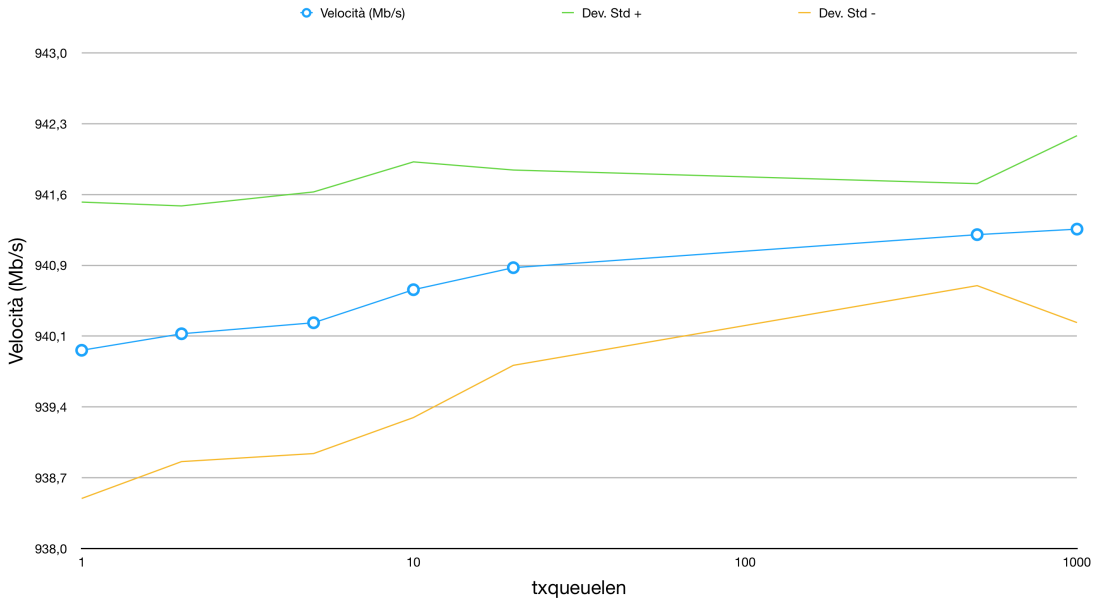

Figure 38: Throughput con incremento del buffer con carico del 100% sulla CPU

Come si può notare dal grafico, l'incremento di prestazione è limitato rispetto al caso precedente per via del carico sul processore che ne limita le prestazioni, dunque è da considerarsi un risultato coerente con le aspettative. Ovviamente se paragonati questi dati con quelli ricavati in precedenza, si può notare come la curva sia più bassa, ma segue abbastanza fedelmente l'andamento riscontrato:

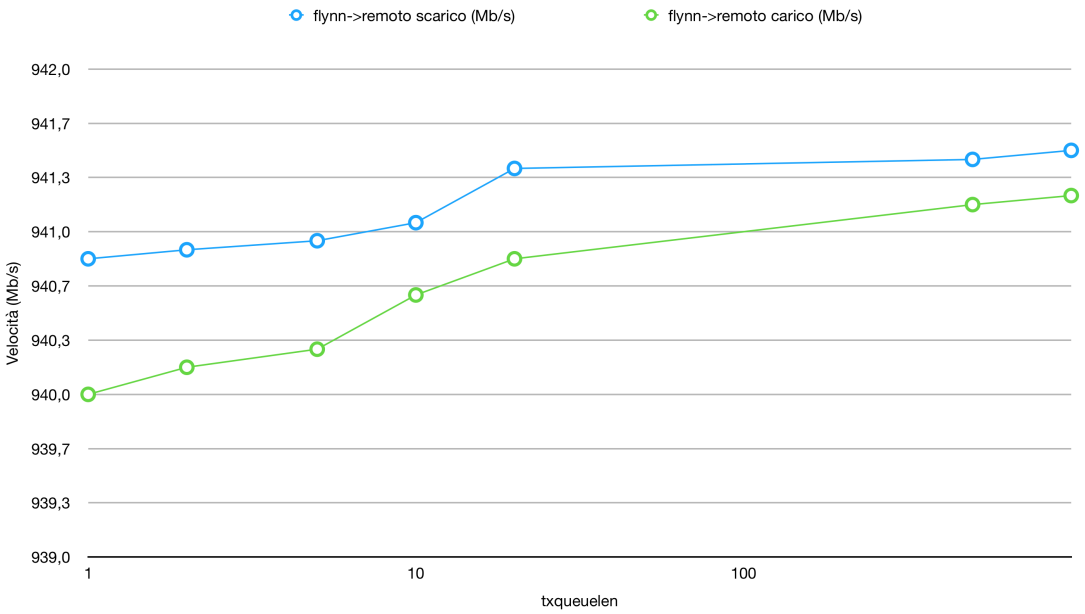

Figure 39: Rapporto throughput su Gb Ethernet

Ciò che emerge dai dati analizzati è che in presenza di una connessione a 1Gb/s le prestazioni non sono al livello di quelle misurate sulle macchine fisiche: queste riescono a trasferire dati ad una velocità pressochè costate che si attesta sui 992Mb/s, al contrario di quanto riscontrato sulla 100Mb/s. Di conseguenza si è cercato di scavare ancora più a fondo per comprenderne le cause. Il test eseguito fonde il throughput e la latenza, così da verificare il comportamento dei pacchetti *ICMP echo request* (e quindi anche gli *ICMP echo reply*) in presenza di un transfer e mantenendo tutte le "txqueuelen" ad un valore pari a 1000. Il seguente comando è stato lanciato sulla *virtual machine* "Flynn":

```
sudo ping -i 0.5 $ADDRESS
```

il che permette di eseguire "ping" verso l'host con indirizzo "$ADDRESS" con una frequeza di un pacchetto ogni 0,5s, per ottenere una stima che possa essere il più veritiera possibile. Dopo un'attesa di 10s è stato avviato il seguente comando sulla stessa macchina:

```
while true; do iperf -c $ADDRESS; sleep 10; done
```

che esegue un test di latenza di durata pari a 10s, per poi attenderne altri 10s prima di eseguirlo nuovamente. L'idea è quella di creare degli intervalli di rete "scarica" e "carica".

In assenza di carico sulla CPU si è osservato il seguente fenomeno:

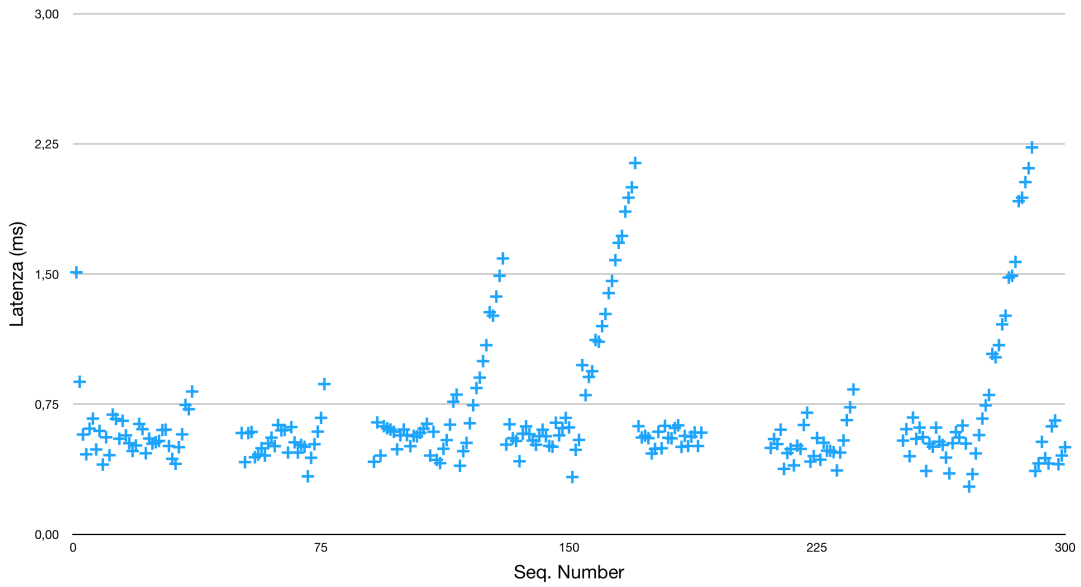

Figure 40: Latenza su sequence number a 0% CPU

Sono osservabili una serie di pacchetti *ICMP* caratterizzati da una latenza che aumenta progressivamente nel momento in cui è eseguito il test di throughput. Ciò lascia intendere che all'interno di Newtron ci sia un progressivo riempimento delle code, i cui pacchetti non riescono ad essere smaltiti efficacemente. La situazione è notevolmente più evidente nel caso in cui la CPU fosse occupata al 100%:

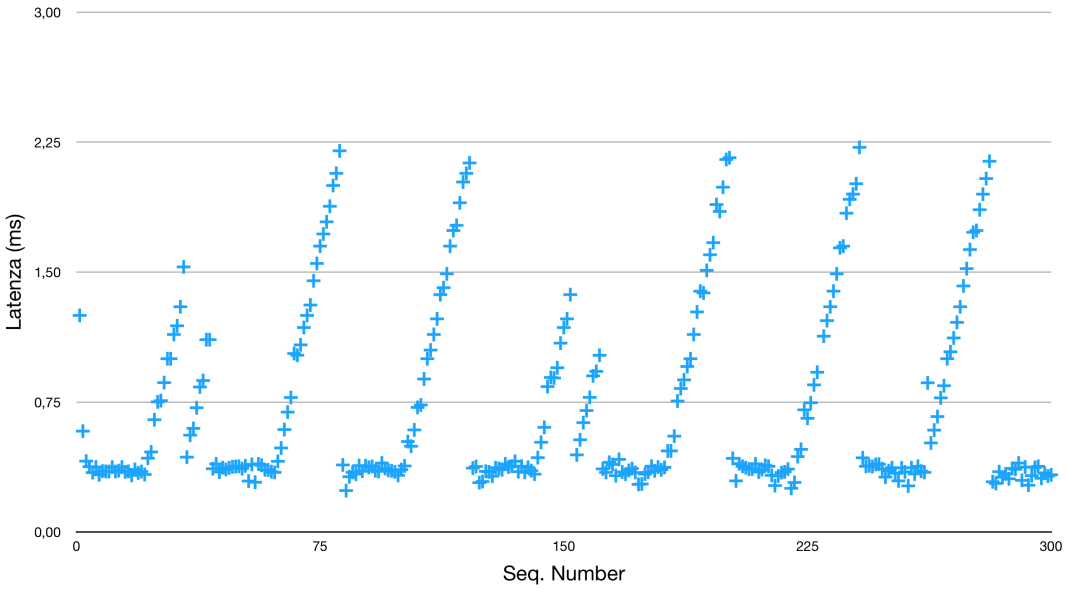

Figure 41: Latenza su sequence number a 100% CPU

In conclusione tale test permette di ritenere che la perdita di prestazione riscontrata in presenza di una connessione a 1Gb/s sia dovuta all'hardware della macchina fisica che, probabilmente, non è sufficientemente efficiente per eseguire le operazioni di trasferimento dei pacchetti dall'ambiente virtuale a quello fisico. Ci si può comunque ritenere soddisfatti dei risultati ottenuti poiché i comportamenti e i dati riscontrati sono perfettamente in linea con quelli di un ambiente privo di virtualizzazione, quindi un attaccante esterno non sarebbe in grado di comprenderne la natura in maniera banale.

6 Conclusioni

Il presente lavoro di tesi ha cercato di proporre un'architettura di rete in grado di ospitare dei sistemi di sicurezza, gli Honeypot, opportunamente connessi in rete e raggiungibili dall'esterno, con lo scopo di raccogliere dati sui più moderni attacchi informatici per studiarli e ricavare nuove misure di protezione. Le specifiche sono state abbastanza chiare e vincolanti, dal momento che si è ricercato un sistema contemporaneamente scalabile, flessibile e di facile gestione: una soluzione basata completamente su macchine fisiche, dopo diverse analisi, è stata immediatamente scartata, poiché non riusciva a garantire le proprietà desiderate, dunque ci si è spostati verso una soluzione basata sul concetto della virtualizzazione che riesce a coniugarle al meglio. Definita la strada da intraprendere, si è ricercato innanzitutto il tool più adatto ad assolvere il compito prestabilito, ovvero il software di virtualizzazione OpenStack che costituisce un sovrasistema operativo in grado di gestire una serie di server. OpenStack permette la creazione di un'arbitraria topologia di rete con l'impiego di *virtual routers* e *virtual machines*, quindi si è provveduto all'implementazione di una tipica rete aziendale con alcuni server dotati del sistema operativo Linux. Una volta costruito e settato il testbed è stato necessario eseguire una serie di verifiche mirate per comprendere contemporaeamente le potenzialità e i punti deboli che la soluzione scelta evidenzia.

I test eseguiti hanno lo scopo di valutare tramite le prestazioni della rete virtuale in sé e del suo intefacciamento con il mondo reale, se un attaccante esterno sia in grado o meno di soprire la natura emulata dell'ambiente ancor prima di prendere il controllo di una o più *virtual machines* o dell'hypervisor stesso. Si è proceduto nella misura dei valori di latenza e throughput tra *virtual machines* e macchine fisiche in diverse condizioni di carico sulla CPU, così da osservarne l'andamento. Nel caso della latenza, i risultati sono discostanti rispetto a ciò che ci si sarebbe aspettato poiché, è stato riscontrato una generale riduzione del suo valore con l'aumento di carico sulla CPU, sia sulle *virtual machines* che sulle macchine fisiche: non sono state eseguite analisi mirate per comprenderne il motivo, ma si può supporre che la causa sia la politica di *power saving* e di *throttling* eseguita dal processore che, in presenza di bassi carichi opera a frequenze inferiori rispetto alle sue reali prestazioni per ridurre i consumi di energia. D'altro canto, per quanto riguarda il throughput tra *virtual machines*, quanto osservato è perfettamente in linea con le aspettative, dal momento che si può apprezzare una perdita progressiva di prestazione in concomitanza con l'incremento di lavoro sulla CPU: dal momento che i pacchetti attraversanti sono elaborati proprio nel processore, un carico elevato comporta una riduzione della velocità di gestione degli stessi. Maggiore attenzione è stata prestata nello scenario *virtual machine - server fisico*, poiché rappresenta il più comune ed il più critico. A fronte di dati non ottimi in rapporto a quelli di una macchina fisica, si è provveduto ad un'analisi approfondita sul sistema virtualizzato per identificarne le cause: si è riscontrato una dimensione dei buffer sui router troppo piccoli che, a fronte dei test portati avanti, ha permesso di giungere alla conclusione che ne fossero la causa primaria. Tali dimensioni sono state progressivamente modificate per apprezzarne gli effetti benefici sulle prestazioni. Al termine si è constatato come in presenza di una rete a 100Mb/s i risultati sono perfettamente paragonabili con quelli di una macchina fisica, al contrario della connessione a 1Gb/s. Ulteriori approfondimenti hanno permesso di affermare che nonostante le modifiche apportate, l'hardware non sia in grado di eseguire efficientemente le operazioni di interfacciamento tra rete virtuale e rete fisica, probabilmente perché troppo vecchio e quindi non sufficientemente prestazionale.

In definitiva è possibile ritenersi soddisfatti del risultato raggiunto poiché i dati ottenuti dall'ambiente virtuale sono perfettamente comparabili con quelli di una rete fisica, pertanto un hacker esterno alla rete del *Politecnico di Torino* non sarà in grado di riconoscere la natura emulata della rete creata semplicemente tramite analisi prestazionali, ma sarà costretto ad effettuare

attività maggiormente invasive che potrebbero solamente giovare alla raccolta dati che si desidera effettuare.

Il lavoro compiuto si è concentrato principalmente sulla ricerca di una soluzione alla necessità di raccogliere dati su attacchi informatici nella maniera più efficiente possibile e, conseguentemente, sull'analisi prestazionale per valutarne la bontà, ma ulteriori estensioni potrebbero essere studiate ed implementate per ottenere topologie che possano essere il più simili possibili a quelle reali, ricercando opportuni tuning sul sistema OpenStack per limitare ancor di più le perdite di prestazione in presenza di elevati carichi sulla CPU. Altrettanto interessante potrebbe risultare l'inserimento di una rete parallela a quella esposta agli hacker per il transito delle informazioni raccolte dagli Honeypot, così da trasferirle verso le apposite macchine che eseguiranno le analisi sugli stessi.

7 Appendice 1 - Installazione e configurazione in multi-node

L'implementazione di tale progetto parte da un foglio completamente bianco su cui andare a disegnare tutto ciò che è effettivamente necessario ed importante inserire.

Il primo passo è stato quello di ricreare in laboratorio l'ambiente fisico descritto. Gli strumenti utilizzati sono:

- Due PC privi di sistema operativo:
  - Specifiche PC1:
    - CPU: Intel(R) Core(TM) i5-7500 @ 3,40GHz x4
    - RAM: 7,7GB
    - Disco: 1TB
    - Ethernet interface: RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (Realteck)
  - Specifiche PC2:
    - CPU: Intel(R) Core(TM) 2 Quad Q8400 @ 2,66GHz x4
    - RAM: 1,9GB
    - Disco: 1TB
    - Ethernet interface: 82567LM-3 Gigabit Network Connection (Intel)
- Un gigabit switch ethernet per interconnettere le due macchine.

La scelta del sistema operativo è un obbligo per ovvi motivi in quanto permette la gestione delle risorse hardware e software della macchina, fornendo servizi di base agli applicativi installati, ma non necessariamente di semplice esecuzione.

Figure 42: Servizi del Sisstema Operativo

Alcuni parametri sono da tenere in considerazione:

- Costo;
- Funzionalità;
- Compatibilità.

Per ovvi motivi di costo si è preferito sfruttare delle soluzioni open-source, pertanto la scelta è stata ristretta a due sistemi operativi:

- CentOS;
- Ubuntu.

Vista la documentazione, la soluzione più utilizzata coinvolgeva l'utilizzo del sistema *CentOS* a *64bit*. Durante il processo di installazione si è incorsi in errori su una delle due macchine poiché dotata di una quantità di risorse insufficienti, pertanto si è pensato di optare per una soluzione in cui la macchina più performante era dotata di CentOS x64, mentre l'altra di CentOS *x86*

(sistema operativo a 32bit, che risulta più leggero, ma garantisce prestazioni, dal punto di vista della potenza di calcolo, inferiori). Il problema di base riscontrato risiede nella non disponibilità dei pacchetti necessari all'installazione di OpenStack per il sistema a 32bit. Ciò decreta, di conseguenza, l'abbandono di CentOS quale sistema operativo di base per tale progetto. Un aspetto degno di nota è che, nonostante il qui descritto sia semplicemente un prototipo su cui eseguire test, l'ultima soluzione identificata, supponendo fosse stata valida, sarebbe stata scartata comunque, dal momento che non vi sarebbe stata uniformità tra i sistemi delle macchine, il che avrebbe rappresentato una soluzione non propriamente "pulita".

La naturale conseguenza è stata quella di adottare il release più recente (al momento della creazione del progetto) di *Ubuntu* ossia il *18.04.1 LTS x64* (unica versione disponibile per il release prescelto) che è risultato essere pienamente compatibile con le macchine in uso. In fase di installazione si è preferito optare per una soluzione minimale, in maniera tale da evitare l'aggiunta di servizi non necessari che avrebbero semplicemente sottratto risorse ai processi più importanti del prototipo.

Lo step successivo è stato quello della gestione dell'indirizzamento: è opportuno far si che le due macchine possano "vivere" nella medesima LAN e poter di conseguenza comunicare tra di loro in maniera diretta. Tale obiettivo è raggiungibile tramite un'opportuna configurazione del sistema operativo, che per semplicità, è omessa.

La descrizione ora si sposta sul cuore del progetto, ovvero l'installazione di OpenStack su entrambe le macchine [13]. All'operazione pratica precede un ragionamento logico sul ruolo che le due macchine devono ricoprire:

- Controller node;
- Compute node.

Una valida idea potrebbe essere quella di sfruttare la più elevata capacità di calcolo di una delle due macchine per eseguire la virtualizzazione vera e propria, designando quella più "debole" al compito di gestione delle risorse e allocazione delle macchine (hypervisor). Il risultato è stato, purtroppo, negativo per motivi alquanto banali: per quanto l'idea fosse valida, le risorse a disposizione del potenziale hypervisor erano troppo ridotte, pertanto durante il processo di installazione, l'avvio dei diversi componenti falliva. Si ricorda infatti che nonostante l'hypervisor abbia il solo compito di gestire il data center, comunque necessita che tutti i componenti di OpenStack siano avviati al suo interno. Tale problema definisce in via definitiva l'architettura del prototipo:

- Hypervisor è ospitato dalla macchina col maggior numero di risorse;
- Compute node è rappresentato dal pc con meno risorse

entrambi appartenenti alla medesima LAN. Di seguito sono riportati gli step per eseguire l'installazione di OpenStack e l'impostazione dell'ambiente: essi risultano simili su entrambe le macchine, quindi si provvederà a specificare quali siano le singole differenze.

Il primo passo eseguito è stato quello di innalzare il livello di privilegio dell'utente corrente, fornendo i diritti di "*superuser*" tramite un comando del tipo:

```
sudo usermod -a -G sudo hduser
```

per cui "*usermod*" è il nome dell'utente a cui si desidera innalzare il privilegio: l'idea è quella di aggiungerlo al gruppo *sudo*, ovvero il gruppo degli amministratori nel sistema operativo Ubuntu.

Successivamente è opportuno settare una chiave SSH su ognuno dei nodi per eseguirne l'accesso. Il comando utilizzato è il seguente:

```
mkdir ~/.ssh
chmod 700 ~/.ssh
echo ''ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCyYjfgyPazTvGpd8OaAvtU2
utL8W6gWC4JdRS1J95GhNNfQd657yO6s1AH5KYQWktcE6FO/xNUC2reEXSGC7ezy+sG
O1kj9Limv5vrvNHvF1+wts0Cmyx61D2nQw35/Qz8BvpdJANL7VwP/cFI/p3yhvx2lsn
jFE3hN8xRB2LtLUopUSVdBwACOVUmH2G+2BWMJDjVINd2DPqRIA4Zhy09KJ3O1Joabr
0XpQL0yt/I9x8BVHdAx6l9U0tMg9dj5+tAjZvMAFfye3PJcYwwsfJoFxC8w/SLtqlFX
7Ehw++8RtvomvuipLdmWCy+T9hIkl+gHYE4cS3OIqXH7f49jdJf openstack.local''
 > ~/.ssh/authorized_keys
```

Per semplicità e rapidità di installazione si è optato per l'installazione di un ambiente OpenStack tramite l'utilizzo di DevStack: una serie di script estensibili utilizzati per creare rapidamente un ambiente OpenStack completo basato sulle ultime versioni, da Git Master. È utilizzato in modo interattivo come ambiente di sviluppo e come base per gran parte dei test funzionali del progetto OpenStack. L'unico svantaggio sostanziale è rappresentato dalla perdita del lavoro svolto nel caso in cui la macchina dovesse essere spenta. In questo caso si suppone che l'ambiente di sviluppo sia posto su un pc continuamente alimentato e dedicato esclusivamente allo scopo di eseguire il software OpenStack senza che questo sia in alcun modo spento.

A questo punto si è proceduto al download dei file necessari dal repository ufficiale:

```
git clone https://git.openstack.org/openstack-dev/devstack
cd devstack
```

Particolare attenzione è da prestare al primo dei due comandi, il quale potrebbe lanciare un errore in fase di download dovuto all'impossibilità di raggiungere la risorsa specificata. A ciò si può facilmente ovviare tramite l'utilizzo del seguente comando prima del download:

```
git config --global url.''https://''.insteadOf git://
```

Ciò è generalmente causato dal malfunzionamento delle url di tipo "*git://*". É opportuno istruire il sistema ad utilizzare invece "*https://*". Dopo aver scaricato tutti i file necessari, è opportuno eseguire la configurazione dell'ambiente. Questo è un procedimento che richiede particolare attenzione, affinché il tutto possa funzionare correttamente al termine dell'installazione (processo che necessita di una considerevole quantità di tempo per essere eseguito).

7.1 Hypervisor

Navigando nei direttori, e recandosi in

```
/home/devstack/samples
```

è possibile trovare dei file di configurazione da customizzare a seconda delle necessità.

Si suppone che il calcolatore sia stato settato con indirizzo IP 192.168.42.11 (poiché nella configurazione a più nodi i primi 10 o più IP nella sottorete privata sono solitamente riservati) sulla scheda di rete *enp1s0*.

Partendo dal presupposto che controller esegue tutti i servizi OpenStack, si procede alla configurazione del file "*local.conf*" [14] inserendo del codice come segue:

```
HOST_IP=192.168.42.11
FLAT_INTERFACE=enp1s0
FIXED_RANGE=10.4.128.0/20
FIXED_NETWORK_SIZE=4096
FLOATING_RANGE=192.168.42.128/25
```

```
MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=tron
DATABASE_PASSWORD=tron
RABBIT_PASSWORD=tron
SERVICE_PASSWORD=tron
```

Innanzitutto si identifica l'IP del nodo corrente e la sua interfaccia verso la rete, in maniera da definire il percorso dei pacchetti uscenti. Nella configurazione è indicato come "*FLOATING_RANGE*" il pool di IP da utilizzare per la rete che fungerà da ponte dall'ambiente virtuale e quello fisico, mentre con "*FIXED_RANGE*" l'insieme di IP che potranno essere utilizzati per raggiungere le macchine virtuali dall'esterno (feature non utilizzata nel progetto, poiché si è seguita una strada differente). Si indica il numero di host che si desidera considerare come compute node e le password per i servizi di base del sistema, ovvero il *database*, *RabbitMQ* e le password di accesso: per semplicità si è usato la parola chiave "*tron*".

A questo punto si provvede alla copia del file modificato insieme a *local.sh* (reperibile nello stesso direttorio) nel livello superiore (*/home/devstack*). Si può ora avviare OpenStack tramite il comando:

```
./stack.sh
```

segue un flusso di attività. Una volta completato, sarà presentato su terminale un riepilogo del lavoro di *stack.sh*, inclusi gli URL, gli account e le password pertinenti. Il file di registro più recente è disponibile in stack.sh.log.

Potenzialmente si potrebbe già iniziare ad operare, ma si preferisce passare ora alla configurazione del compute node.

7.2 Compute node

Navigando nei direttori, e recandosi in

```
/home/devstack/samples
```

è possibile trovare dei file di configurazione, come descritto in precedenza.

Si suppone che il calcolatore sia stato settato con indirizzo IP 192.168.42.12 sulla scheda di rete *enp0s25*.

Partendo dal presupposto che i nodi di calcolo eseguono solo i servizi di lavoro OpenStack, si procede alla configurazione del file "*local.conf*" inserendo del codice come segue:

```
HOST_IP=192.168.42.12 # change this per compute node
FLAT_INTERFACE=enp0s25
FIXED_RANGE=10.4.128.0/20
FIXED_NETWORK_SIZE=4096
FLOATING_RANGE=192.168.42.128/25
MULTI_HOST=1
LOGFILE=/opt/stack/logs/stack.sh.log
ADMIN_PASSWORD=tron
DATABASE_PASSWORD=tron
RABBIT_PASSWORD=tron
SERVICE_PASSWORD=tron
DATABASE_TYPE=mysql
SERVICE_HOST=192.168.42.11
```

```
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
ENABLED_SERVICES=n-cpu,q-agt,n-api-meta,c-vol,placement-client
NOVA_VNC_ENABLED=True
NOVNCPROXY_URL=''http://$SERVICE_HOST:6080/vnc_auto.html''
VNCSERVER_LISTEN=$HOST_IP
VNCSERVER_PROXYCLIENT_ADDRESS=$VNCSERVER_LISTEN
```

Tale configurazione è analoga, a meno dell'IP e dell'interfaccia di rete, per tutti i futuri compute node inseriti nel sistema.

Oltre alle informazioni già inserite anche nell'Hypervisor, qui si indica innanzitutto la posizione del nodo di controllo (inserendo l'indirizzo IP), insieme alla posizione dei servizi, tramite URL.

A questo punto si provvede alla copia del file modificato insieme a *local.sh* (reperibile nello stesso direttorio) nel livello superiore (*/home/devstack*). Si può ora avviare OpenStack tramite il comando:

```
./stack.sh
```

Anche in questo caso è eseguito un flusso di attività. Una volta completato, sarà mostrato un riepilogo del lavoro di *stack.sh*, inclusi gli URL, gli account e le password. Il file di registro più recente è disponibile in stack.sh.log.

Dalla documentazione ufficiale, emerge che a partire dalla versione di Ocata, Nova richiede una distribuzione di Cells v2[4] [15]. I servizi del nodo di calcolo devono essere associati a una cella prima che possano essere utilizzati. Dopo che ogni nodo di calcolo è stato aggiunto, è opportuno verificare che compaia nell'elenco di:

```
nova service-list --binary nova-compute
```

Il servizio di calcolo viene registrato nel database delle celle in modo asincrono, pertanto potrebbe richiedere il polling.

7.3 Accoppiamento dei nodi

Una volta visualizzati i servizi del nodo di calcolo, eseguire lo script:

```
./tools/discover_hosts.sh
```

dal nodo di controllo per mappare gli host di calcolo sulla singola cella.

Il servizio di calcolo in esecuzione sul nodo di controllo primario verrà rilevato automaticamente quando il nodo di controllo è compilato, quindi questo deve essere eseguito solo per i sottonodi.

A questo punto l'ambiente di sviluppo è completamente avviato ed è possibile operare sul controller collegandosi all'interfaccia web disponibile all'indirizzo:

```
http://192.168.42.11/dashboard
```

[4] storicamente, Nova è dipesa da un singolo database logico e una coda di messaggi su cui tutti i nodi dipendono per la comunicazione e la persistenza dei dati. Questo diventa un problema per i deployer in quanto il ridimensionamento e la tolleranza di errore per questi sistemi è difficile. Cells è una funzione che viene utilizzata da alcune distribuzioni di grandi dimensioni per partizionare i nodi di calcolo in gruppi più piccoli, accoppiato con un database e una coda. Questo sembra essere un sistema di risorse ben voluto e di facile comprensione, ma la sua implementazione presenta problemi di manutenzione e correttezza

8 Appendice 2 - Installazione e configurazione in single-node

A seguito di alcuni test sulla verifica del corretto funzionamento del sistema, è stato riscontrato che la parte computazionale era svolta comunque per la maggior parte sul nodo di controllo, per via delle elevate risorse a disposizione, mentre l'effettivo compute node eseguiva esclusivamente operazioni di Storage, pertanto si è deciso, anche per avere una maggior velocità di setup nei vari test eseguiti, di accantonare nella creazione del prototipo il secondo nodo. Si è proceduto, quindi ad una soluzione di tipo single-node, ma configurata in maniera tale da permettere una successiva estensione ad ulteriori compute node.

L'installazione dei componenti segue quanto già esposto nella sezione 7.1 con l'unica differenza risiedente nell'utilizzo di un IP publico per la macchina fisica che risulta 130.192.91.9. Il risultato che si desidera ottenere è rappresentato nella seguente figura:

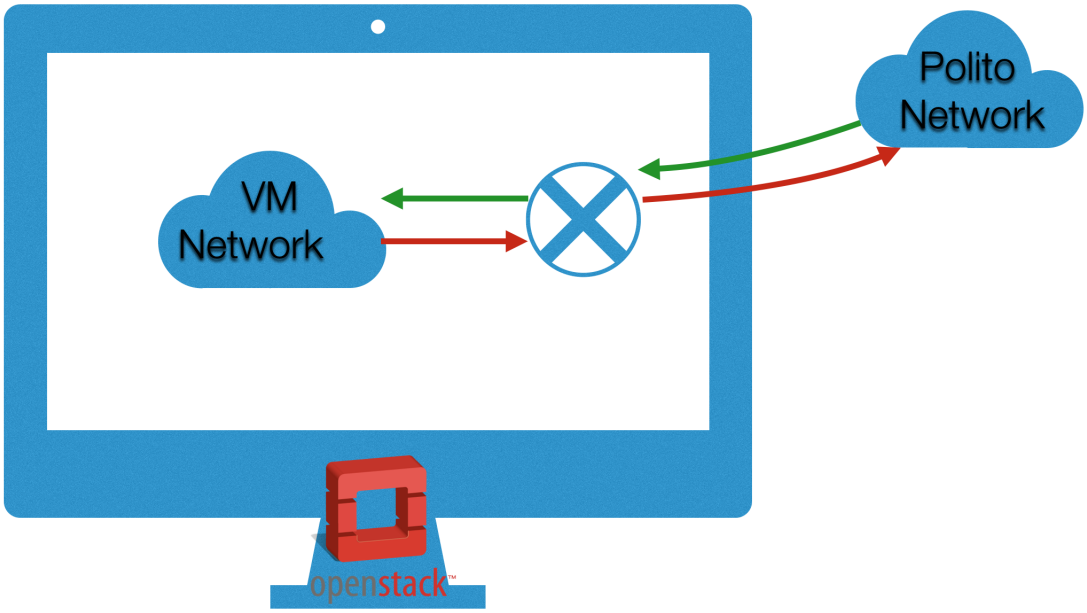

Figure 43: Connessione tra rete fisica e virtuale

Chiaramente il PC è collegato alla rete del Politecnico di Torino con un indirizzo pubblico appartenente al range in possesso dell'Università. Questo permette il raggiungimento della rete Internet. All'interno del sistema è istanziato OpenStack che necessita di scambiare pacchetti con la rete esterna tramite un router virtuale. Entrando più nello specifico, quello che si vuole ottenere è quanto raffigurato di seguito:

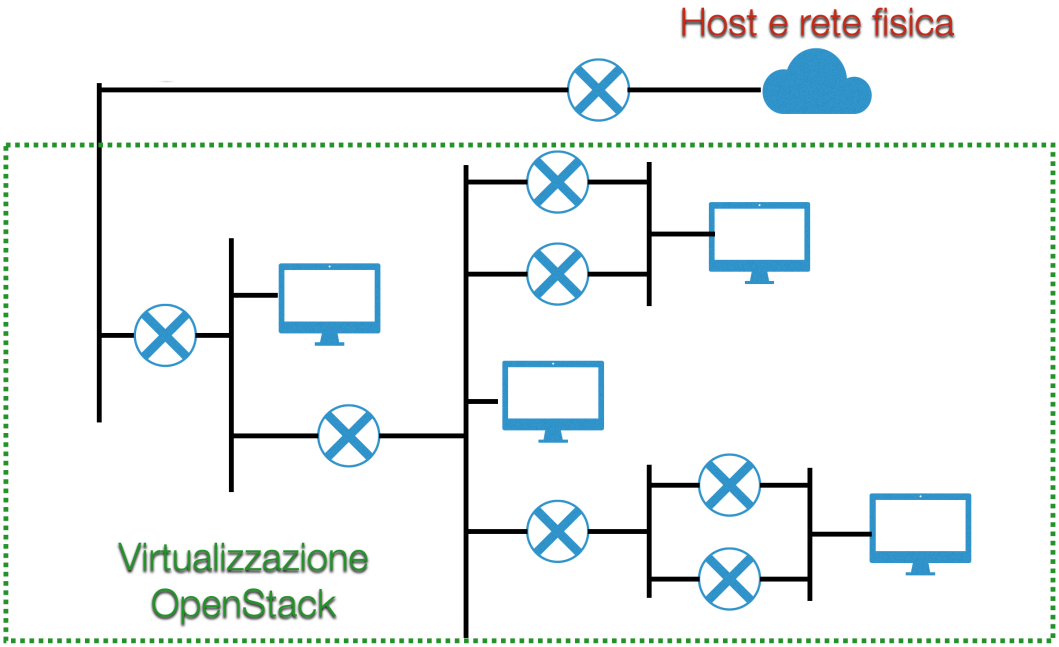

Figure 44: Topologia generale della rete virtuale

Si è creata una topologia di arbitraria complessità con una serie di virtual machines interconnesse tra di loro tramite dei *virtual routers*. Tale rete si vuole che sia collegata ad un router, anch'esso virtuale che permetta il raggiungimento della rete fisica. Per raggiungere tale obiettivo, è necessario introdurre nell'IP table dello stesso una rotta statica che forzi il forwarding dei pacchetti verso la rete virtualizzata. Entrando più nel dettaglio, di seguito è rappresentata la vera topologia ideata per il progetto:

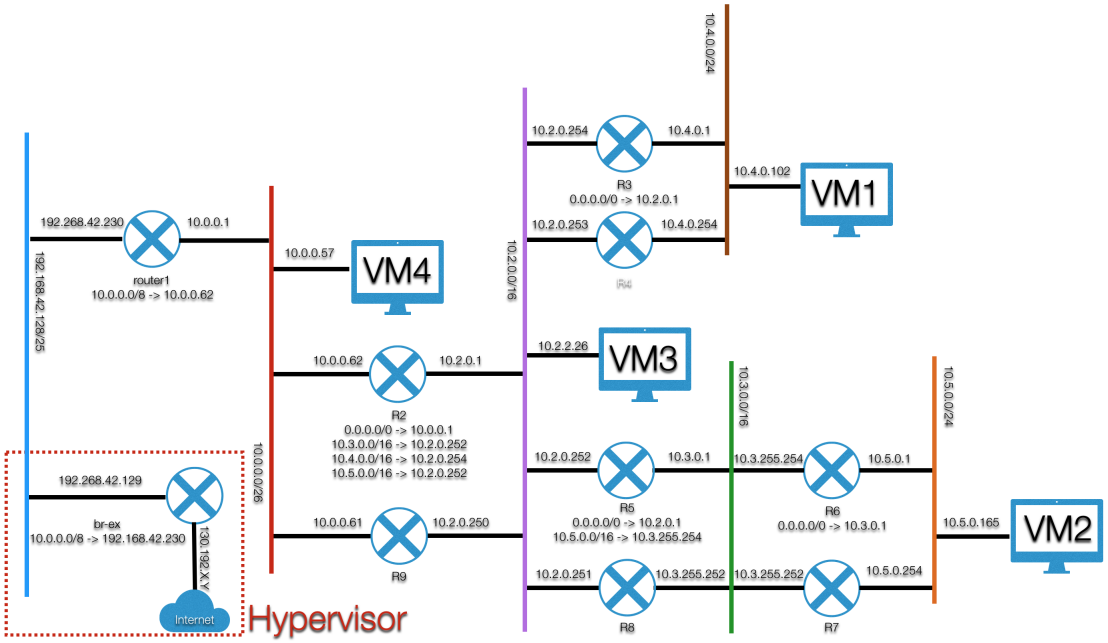

Figure 45: Topologia con piano di indirizzamento e tabelle di routing

Partendo dalla zona tratteggiata (Hypervisor, ovvero la macchina fisica), essa è connessa alla rete con un indirizzo del tipo 130.192.X.Y (la cui conoscenza al momento non è rilevante) che è annunciato su una delle interfacce del router virtuale denominato *"br-ex"*, all'interno del quale è da inserirsi la rotta statica:

`10.0.0.0/8 -> 192.168.42.230`

tramite il comando:

`route add -net $NET netmask $MASK gw $GATEWAY`

al quale è opportuno eseguire le seguenti sostituzioni:

- $NET -> 10.0.0.0: per indicare quale rete deve essere annunciata;
- $MASK -> 255.0.0.0: per indicare la rispettiva subnet mask;
- $GATEWAY -> 192.168.42.230: per indicare verso quale interfaccia deve essere forwardato il traffico.

Il motivo per cui risulta opportuno eseguire tale operazione è intuibile osservando la precedente figura e rivedendo la configurazione effettuata all'atto dell'installazione di OpenStack: il parametro "*FLOATING_RANGE*" rappresenta il range di indirizzi utilizzato da OpenStack per istanziare automaticamente una rete "*pubblica*", ossia che possa connettersi alla rete fisica, assieme ad un primo *virtual router* ("*router1*"). Nel range indicato è presente l'indirizzo 192.168.42.230 (appartenente a 192.168.42.128/25) che corrisponde a quello di una delle interfacce di "*router1*", dunque quello che si vuole ottenere è il forward dei pacchetti verso questo router che si occuperà successivamente di consegnarlo all'end-point corretto.

Tutto ciò che è posto a valle di "*router1*" è configurabile tramite l'opportuna interfaccia web fornita da OpenStack all'indirizzo:

http://130.192.91.9/dashboard

Sono state istanziate le seguenti reti virtuali, rappresentate con delle barre verticali nella precedente immagine:

- 10.0.0.0/26;
- 10.2.0.0/16;
- 10.3.0.0/16;
- 10.4.0.0/24;
- 10.5.0.0/24;

scelte in maniera completamente arbitraria dal punto di vista del piano di indirizzamento, ma sempre con un occhio alla flessibilità ed estensibilità del progetto stesso, garantendo una più facile gestione delle rotte, come spiegato di seguito, inoltre, l'idea generale che è dietro la topologia proposta, è quella di simulare un ambiente simile a quello di un data center, con una serie di virtual machines interconnesse tramite dei router ridondanti.

In un ambiente reale e di produzione, l'utilizzo di sistemi ridondanti è una necessità a cui non è possibile prescindere: le macchine, in quanto tali, sono soggette a *failures* di varia natura e, se non vi è ridondanza, possono rivelarsi dei *single-point-of-failure*, ovvero causerebbero il partizionamento della rete e conseguentemente un *denial-of-service*. Nelle reti fisiche sono utilizzabili dei particolari algoritmi quali l'HSRP [16] (*Hot Standby Router Protocol*, ideato da Cisco) che garantisce la *fault-tolerance* tra più router nella scelta di un *default gateway* (largamente descritto in [17]). Il protocollo è progettato per l'uso su LAN multi-accesso, multicast o broadcast (ad es. Ethernet). L'HSRP non è inteso come una sostituzione dei meccanismi di individuazione di router dinamici esistenti, e tali protocolli dovrebbero invece essere utilizzati laddove possibile. Si presume che tutti i router che partecipano all'HSRP eseguano protocolli di routing IP appropriati e dispongano di un insieme coerente di percorsi. Usando HSRP, un insieme di router cooperano per dare l'illusione di un singolo router virtuale agli host sulla LAN. Questo set è noto come gruppo HSRP o gruppo standby. Un singolo router eletto dal gruppo è responsabile per l'inoltro dei pacchetti che gli host inviano al router virtuale. Questo router è noto come router attivo. Un altro router viene scelto come router di standby. Nel caso in cui il router attivo non funzioni, lo standby assume le funzioni di inoltro dei pacchetti del router attivo. Sebbene un numero arbitrario di router possa eseguire HSRP, solo il router attivo inoltra i pacchetti inviati al router virtuale.

Nell'ambiente virtuale creato con OpenStack, questo protocollo non è implementato. Si è deciso, quindi, di utilizzare solo uno dei due router tramite l'inserimento di apposite rotte statiche (indicate puntualmente in figura), mentre i router secondari rimangono passivi e non saranno mai utilizzati. Essi risponderanno se contattati da altre macchine, ma non concorreranno al forward dei pacchetti verso le destinazioni, ma è comunque una condizione accettabile per il progetto, dal momento che il tutto risulta un ambiente virtuale con lo scopo semplicemente di ricreare un ambiente reale.

L'ultimo step è quello dell'avvio di alcune *virtual machines*. Si è deciso di stanziarne 4 per semplicità. Il sistema operativo scelto è stato un *CirrOS* (salvato nel componente Cinder), ovvero un piccolo OS specializzato nell'esecuzione in ambienti cloud. È estremamente leggero e garantisce le funzionalità base di un OS e di debug per un data center. Ciò ha permesso il debug ed il testing del prototipo. OpenStack si occupa autonomamente non solo dell'assegnazione di un indirizzo IP tramite i DHCP client istanziati per ogni rete, ma invisibili all'amministratore,

ma anche di settare il default gateway a seconda dei router presenti sulla rete: il router con parte host dell'IP pari ad 1, sarà il gateway. Lo schema dei gateway è rappresentato di seguito:

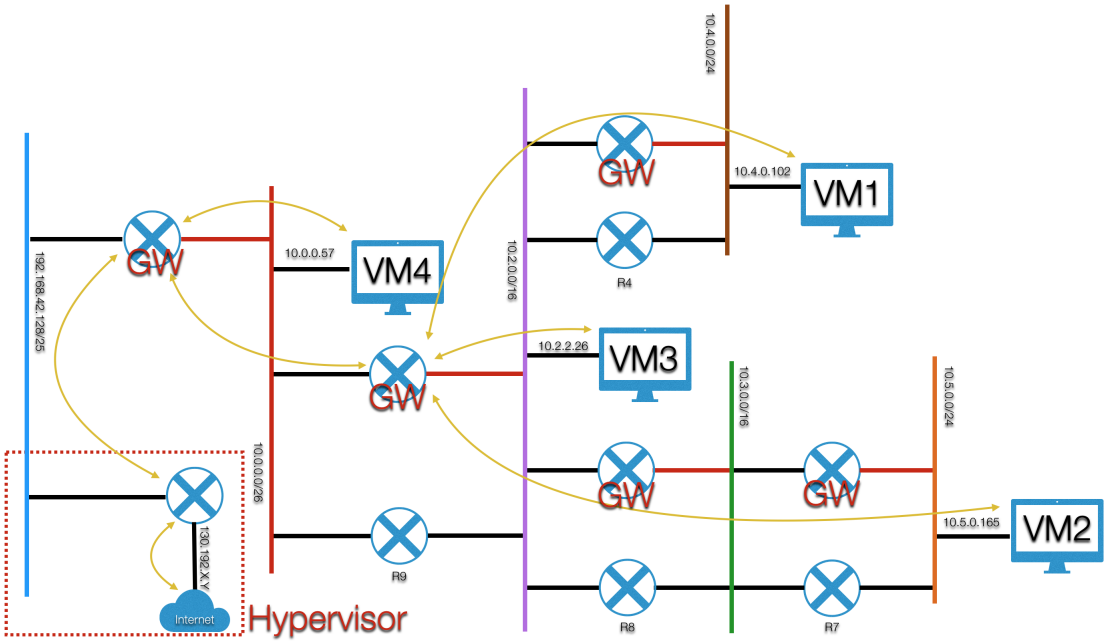

Figure 46: Topologia dei default gateway

References

[1] N. Provos, "A virtual honeypot framework," *USENIX*, Aug. 2004.

[2] M. Nawrocki, M. Wahlisch, T. C. Schmidt, C. Keil, and J. Schonfelder, "A survey on honeypot software and data analysis,"

[3] A. Mairh, D. Barik, K. Verma, and D. Jena, "Honeypot in network security: a survey," vol. International Conference on Communication, pp. 600 – 605, Computing & Security. ACM, 2011.

[4] N. Rackspace Cloud, "Openstack documentation," Aug. 2018.

[5] NASA, "Openstack cloud computing platform," 2018.

[6] Rackspace, "Rackspace openstack, from cloud pioneer to industry innovator," 2018.

[7] B. Karacali and J. M. Tracey, "Experiences evaluating OpenStack network data plane performance and scalability," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, IEEE, apr 2016.

[8] G. Almasi, M. Banikazemi, B. Karacali, M. Silva, and J. Tracey, "Open-stack networking: It's time to talk performance," 2015. OpenStack Summit - Vancouver.

[9] R. Shea, F. Wang, H. Wang, and J. Liu, "A deep investigation into network performance in virtual machine based cloud environments," 2014. Proceedings of IEEE INFOCOM.

[10] "Vmtp,"

[11] OpenStack.org, "Openstackclient," Aug. 2018.

[12] OpenStack.org, "Install the openstack command-line clients," Jan. 2018.

[13] OpenStack.org, "Multi -node lab," Jan. 2019.

[14] OpenStack.org, "Configuration," Sept. 2018. Sezione: local.conf.

[15] OpenStack.org, "Cells," Feb. 2019.

[16] Cisco, "Configuring hsrp," 2018. Understanding HSRP HSRP Versions Multiple HSRP.

[17] D. Li, B. A. Cole, P. Morton, and T. Li, "Cisco Hot Standby Router Protocol (HSRP)." RFC 2281, Mar. 1998.