ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA
SECONDA FACOLTÀ DI INGEGNERIA CON SEDE A CESENA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

SDN e sperimentazioni di VM migration

Elaborato in

"Reti di telecomunicazioni L-M"

Relatore

Franco Callegati

Presentata da

Giuliano Santandrea

Correlatore

Walter Cerroni

Sessione III
Anno Accademico 2011/2012

SOMMARIO

Introduzione

Le nuove teorie di rete come Software Defined Networking Network Function Virtualization, insieme alle teorie Cognitive/Autonomics consentono di abilitare scenari futuri “disruptive” di rete. Lo scopo di questa tesi è quello di esplorare questi scenari futuri e di capire il ruolo della migrazione di funzioni di rete, sotto forma di Virtual Machine. Si vuole affrontare la migrazione di Virtual Machine non solo dal punto di vista delle performance, ma anche come strumento di gestione delle risorse in uno scenario di rete d'accesso autonomica.

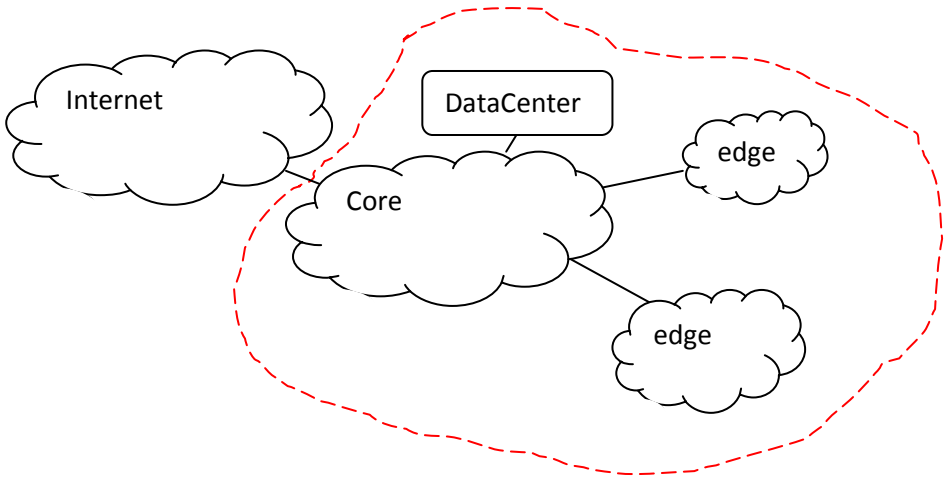
Capitolo 1

Situazione attuale della rete e prospettive

In questo capitolo si descrive la situazione attuale della rete e i problemi che ne derivano; vengono tratteggiati degli scenari che descrivono come dovrà essere la rete di domani, e quali vantaggi comporterà per l'operatore; vengono individuate le problematiche che nascono dall'introduzione del nuovo paradigma. Sono elencate quali teorie e aree di ricerca abiliteranno il cambiamento del paradigma di rete.

1.1 Situazione attuale e problematiche

La rete attuale è molto complessa, statica e sempre più difficile da gestire. Sono in atto molti cambiamenti che la rendono sempre più inadatta alle sfide future. È di fondamentale importanza per si occupa di reti di telecomunicazioni, come gli Operatori, capire quali sono i trend in atto nella rete ed elaborare degli scenari di rete futura, tenendo d'occhio le innovazioni tecnologiche e le nuove teorie di ricerca.



La rete di un operatore

1.1.1 Un'architettura di rete poco flessibile

Per far fronte alle differenti necessità delle reti di oggi è necessario un cambiamento nel'architettura delle reti odierne. Il modello convenzionale consiste in una rete gerarchica, dove tanti switch sono collegati fra loro a formare un albero: questa architettura non si adatta bene alle reti di DataCenter o dei Carrier (gestori di una infrastruttura di rete) odierni, e il motivo è che è cambiato il pattern di traffico. Inizialmente il traffico consisteva solamente nello scambio di pacchetti fra client e server, quindi di tipo nord-sud, ma ora non è più così. Le applicazioni sono distribuite fra un gran numero di server, e questi ultimi comunicano fra di loro scambiandosi grandi quantità di dati prima di rispondere all'utente con il risultato finale. Il traffico fra i server, detto anche M2M (Machine to Machine), è di tipo est-ovest ed è altamente dinamico, ovvero spesso non è possibile prevedere in anticipo quali saranno i flussi di traffico fra i vari nodi. Il motivo di tale dinamicità può risiedere nel comportamento stesso delle applicazioni distribuite: a seconda delle situazioni l'applicazione può richiedere flussi di traffico molto diversi fra di loro. Oppure i server contengono spesso istanze di applicazioni che possono essere spostate da un server all'altro e questo a sua volta causa un cambiamento del pattern di traffico.

1.1.2 I DataCenter

Le aziende inoltre hanno cominciato a dotarsi di grandi centri di calcolo, i DataCenter, o a usufruirne di servizi da altre aziende che dispongono di questi DataCenter. I Data Center sono dei centri di calcolo composti da un gran numero di server connessi a una rete privata ad alte prestazioni. Con le tecnologie di rete attuali i gestori dei Data Center faticano a ottenere il livello di flessibilità che vorrebbero nella gestione del Data Center e sentono la necessità di nuovi paradigmi.

Questi DataCenter spesso poi sono dislocati molto lontano l'uno dall'altro e si scambiano fra di loro grandi quantità di traffico, formando una rete wide-area-network. Questo traffico WAN deve passare attraverso la rete internet. A questo si aggiungono i problemi relativi all'*addressing* quando si devono operare delle *network consolidation* fra Data Center dislocati in punti

geografici differenti o si devono operare migrazioni di VM inter-DataCenter. Alcune soluzioni sviluppate per far fronte a questi problemi cercano di realizzare un "mac-over-IP", ovvero aggiungono uno ulteriore strato di rete L2 sopra l'IP, ma spesso queste soluzioni sono poco flessibili e alla prova dei fatti poco pratiche.

1.1.3 Aumento di complessità della rete

Con l'avvento di nuovi dispositivi come smartphone e tablet è aumentata per gli operatori la richiesta di servizi di traffico mobile, e in generale un aumento dei dispositivi che richiedono l'accesso alla rete (sia voce che rete internet). Si è verificato inoltre un aumento dei servizi di cloud: molti utenti o aziende sfruttano sempre di più servizi di cloud privati/pubblici. Tutti questi fattori hanno causato un aumento di complessità della rete.

La tecnologia di rete di oggi è un insieme di protocolli progettati per connettere gli host fra di loro, ma i protocolli sono definiti per risolvere solo un problema specifico e sono definiti in isolamento rispetto agli altri e vengono progettati e sono sviluppati senza delle astrazioni di rete ben definite e universali. Questa complessità eccessiva ha portato a una staticità della rete: per minimizzare il rischio di disservizi si cerca di modificare il meno possibile la rete attuale.

La virtualizzazione dei server ha comportato un aumento del numero di host e un cambiamento dei pattern di traffico.

Le reti attuali tendono sempre di più verso una rete dati, voce e video che sia IP-converged.

C'è difficoltà ad automatizzare la configurazione delle policy poiché i dispositivi sono poco flessibili. Attualmente la granularità con cui si possono impostare le policy è per singola applicazione o per sessione, ma spesso sono richieste policy più complesse e raffinate che sono difficili da impostare. Inoltre bisogna configurare manualmente ogni dispositivo di rete e se i dispositivi sono di vendor differenti s usano differenti tool/comandi. Si fa quindi molta fatica a applicare delle policy globali, ovvero a livello di rete, e che siano anche consistenti.

1.1.4 Difficoltà di gestione dei dispositivi di rete

Una rete scalabile è una rete in cui le prestazioni non degradano se la rete viene estesa: una piccola azienda dispone inizialmente di una rete modesta, proporzionata alle esigenza iniziali; poi l'azienda cresce, crea nuove sedi distaccate, si dota di DataCenter, e ha necessità di estendere la rete.

I dispositivi attuali sono configurabili attraverso terminale remoto: l'amministratore di rete si collega da remoto (es. via telnet) e invia dei comandi alla middle box per riconfigurarla. Il collegamento è di tipo 1-1 e il processo di configurazione richiede che l'amministratore inserisca i singoli comandi, che inoltre sono specifici del vendor che ha fabbricato il dispositivo.

A grandi dimensioni non è gestibile in una rete che scala dover fare tutto il management a livello manuale, ovvero a livello di configurazione dei singoli dispositivi.

1.1.5 Perdita del principio end-to-end

Inizialmente nelle reti valeva il principio dell'end to end, ovvero i pacchetti viaggiavano da un capo all'altro della rete senza subire grosse trasformazioni. Negli anni le reti hanno acquisito un numero sempre maggiore di funzionalità, e queste funzionalità sono diventate sempre più sofisticate (NAT, IDS, DPI,..). Poiché le reti attuali sono poco flessibili queste funzionalità sono state introdotte in nodi di rete specializzati, detti middleboxes, che sono piazzati in nodi intermedi della rete. I pacchetti, passando attraverso queste middleboxes, subiscono delle trasformazioni e quindi il principio dell' end-to-end è venuto meno. Il network processing che avviene in queste middlebox è diventato una parte cruciale delle reti di oggi. L'enorme aumento di complessità delle reti ha reso sempre più dispendioso, in termini di tempi e lavoro richiesto, lo sviluppo e il deployment di nuovi servizi nella rete. Si è quindi alla ricerca di alternative, ovvero di piattaforme che permettano di ospitare funzioni di rete.

1.1.6 Mancanza di una piattaforma di management universale

Vi è inoltre il problema di unitarietà di gestione degli apparati di management: le piattaforme di cui dispongono gli operatori per gestire le loro

reti sono tante e poco integrate fra di loro, dove ogni sistema di gestione si preoccupa di un determinato e specifico aspetto di gestione. Si sente quindi la necessità di avere una unitarietà di gestione delle reti.

1.2 Previsioni delle reti del futuro/ vision

Negli ultimi anni si è verificata la migrazione dell'intelligenza verso le parti più periferiche della rete, ovvero l'edge network. Quello che si aspetta è questa tendenza si accentui sempre di più.

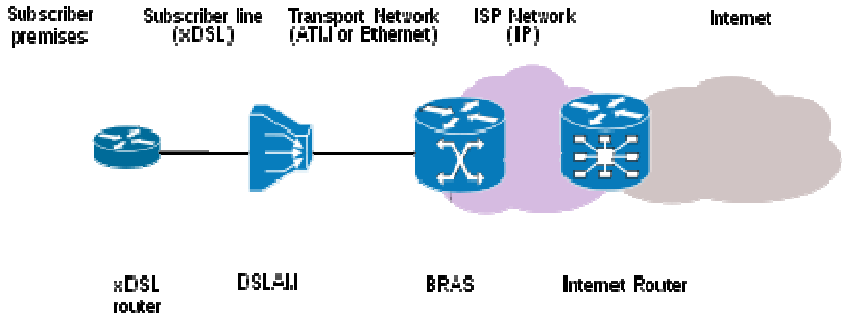
Le reti del futuro saranno composte da un gran numero di nodi, piccoli, poco costosi fisici o virtuali. Ci sarà un generale aumento della connettività, quindi una richiesta sempre crescente di maggiore robustezza dell'architettura di rete nel suo insieme.

Le reti diventeranno sempre più virtuali: le funzioni di rete saranno sempre più disaccoppiate dall'HW sottostante e questo aprirà un numero enorme di possibilità.

La rete di accesso sarà composta da un grandissimo numero di nodi commodity HW, ovvero HW generico, su cui si potranno allocare le funzioni di rete a piacimento per ottenere le funzionalità di rete desiderate a tutti i livelli della scala ISO/OSI (L3+). Si potrà allocare, spostare rimuovere funzioni di rete in tutti o quasi i nodi della rete.

Inoltre l'aumento della complessità delle reti renderà sempre più difficile una gestione manuale della rete. Saranno quindi presenti elementi autonomici-cognitivi, tali per cui la rete sarà dotata di un certo livello di autonomia e autogestione. Questo non significa che l'intervento umano non sarà più necessario ma sicuramente lo sarà in misura molto minore di oggi.

1.2.1 Customizzazione dei provisioning dei servizi



I dispositivi di rete d'accesso odierni (tratto da Wikipedia)

I nodi all'edge che si collegano agli utenti e forniscono il cosiddetto "ultimo miglio" (es. i DSLAM) attualmente funzionano unicamente da "imbuto", ovvero non hanno altre funzioni al di fuori di quella di fornire la connettività fisica agli utenti. Le funzioni di rete sono spesso negli strati più interni della rete, ad esempio nei BRAS, o negli switch e router collegati al BRAS. Si vuole spostare ancora di più l'intelligenza all'edge: in futuro è possibile che i nodi "ultimo miglio" contengano funzioni di rete, così da essere in grado di fornire una connessione internet customizzata per ciascun utente. Ad esempio un utente residenziale potrebbe ottenere una connessione internet a medie prestazioni con incluso un servizio di video on-demand, mentre un'azienda potrebbe ottenere una connessione internet con garanzie alte di prestazione, delle policy di utilizzo già impostate (es. firewall, proxy, ACL) e dei servizi di sicurezza aziendale (es. IDS). Questi servizi sarebbero forniti a livello di rete, ovvero l'azienda non dovrebbe dotarsi essa stessa di apparati che forniscono queste funzioni ma sarebbero già incluse nel servizio.

Un altro approccio potrebbe essere quello di inserire le funzionalità di rete addirittura all'interno dei nodi dell'utente, come il router degli utenti residenziali. Anche da questo frangente si aprono molteplici possibilità.

1.2.2 Ritorno all'end-to-end

Un futuro possibile e desiderato dagli operatori è che l'evoluzione della rete riporti la *rete core* al principio dell'end-to-end: la rete core deve tornare a essere un insieme di "tubi" ad altissime performance che si occupano solamente di trasportare i pacchetti senza operare delle trasformazioni su di essi. In una rete come questa un pacchetto partirebbe da una rete edge, attraverserebbe il core (che opererebbe solamente a livelli 1-3 dello stack ISO/OSI) e poi arriverebbe di nuovo a una rete edge, dove verrebbe nuovamente trattato anche ai livelli superiori (livelli OSI 4+, ovvero dal livello di trasporto in su). In questo modo nel core verrebbero a mancare quegli elementi che rallentano il flusso del traffico, portando ad un aumento delle performance. Nella rete all'edge invece si possono allocare le funzionalità, così che l'operatore può farne il provisioning agli utenti residenziali o business (banche, aziende, etc.). Lo spostamento delle funzionalità all'edge può risultare anche vantaggiosa in termini di qualità del servizio, poiché i nodi che forniscono le funzionalità sono più vicini agli utenti e questi ne beneficiano in termini di performance.

1.2.3 La metafora del computer

L'evoluzione che stiamo testimoniando nella rete si può paragonare a quello che è successo ai computer: si è passati da dei dispositivi di calcolo proprietari e acquistabili da pochi vendor, ai *computer* che hanno una design aperto e sono progettati per l'interoperabilità. I computer dispongono di un insieme di istruzioni primitive, l'Instruction Set Architecture (ISA), che è standard e consente di scrivere programmi eseguibili su processori differenti.

Allo stesso modo per i dispositivi di rete succederà la stessa cosa: gli apparati chiusi e proprietari che contengono all'interno firmware sviluppato internamente dai vendor verranno sempre di più sostituiti da dispositivi aperti. Attualmente l'interoperabilità di dispositivi di vendor differenti è limitata solo ai protocolli standard della rete internet e questo è un problema: molti vendor sviluppano protocolli proprietari che implementano funzionalità avanzate di rete, ma questi protocolli funzionano solamente fra dispositivi prodotti dalla stesso vendor.

1.2.4 Vantaggi economici

Un cambiamento di paradigma potrebbe portare notevoli vantaggi economici a chi si occupa di fare il provisioning dei servizi, come gli operatori. Attualmente il modello di business è sfavorevole per gli operatori, in quanti gli utenti usano la rete e le funzioni di rete senza pagare direttamente i servizi correlati ma pagando un generico costo di utilizzo del servizio. Poiché queste funzioni di rete sono contenute nelle middlebox gli operatori riescono ad agire poco su di esse, ma con degli apparati di rete più flessibili sarebbe possibile fare un provisioning dei servizi molto più specifico e flessibile: per ciascun utente si potrebbero allocare determinate funzioni di rete e quindi determinati servizi.

I vantaggi dal punto di vista economico per chi deve acquistare e mantenere gli apparati di rete, come ad esempio gli operatori o i gestori di DataCenter, si concretizzano in un calo dell'OPEX e nel CAPEX. Il CAPEX (capital expenditure), ovvero la spesa di investimento iniziale, sarà molto minore poiché l'acquisto di commodity HW è molto minore rispetto al costo di acquisto delle middlebox attuali. Le middleboxes sono dispositivi proprietari molto costosi, poiché le funzioni di rete che contengono sono il frutto di anni di ingegnerizzazione e segreto industriale, che hanno consentito ai vendor di realizzare HW ottimizzato per le specifiche funzionalità di rete. Tuttavia con commodity HW le funzionalità potrebbe essere realizzare il più possibile in SW, magari con sforzi collettivi, ad esempio di comunità open-source.

L'OPEX (operational expenditure), ovvero il costo di mantenimento e gestione degli apparati, sarà minore perché l'intervento umano sarà meno importante: l'aspetto autonomico-cognitivo ridurrà il numero di interventi umani necessari e quindi anche degli errori che ne possono derivare.

1.3 Le funzioni di rete da virtualizzare

Abbiamo detto che si vuole togliere le funzioni di rete dalle middlebox chiuse e proprietarie e virtualizzarle, ovvero renderle delle applicazioni in SW da poter instanziare a piacimento su nodi commodity HW. Per capire quali

funzioni di rete in futuro sarà possibile instanziare nella rete possiamo analizzare le funzioni che sono presenti all'interno delle middleboxes.

Queste funzioni sono:

- le funzioni di rete di base come il forwarding e il routing
- funzioni relative alla sicurezza, come applicazioni Intrusion Detection System (IDS), firewall, Deep Packet Inspection (DPI)
- delle specifiche sul comportamento della rete come policy di rete, access control (ACL)
- funzionalità di gestione del traffico (Traffic Engineering), bandwidth management
- funzionalità di risparmio energetico, ottimizzazione dello storage e processor utilization
- multicasting, NAT

1.4 Scenario di "network upload"

Questo nuovo paradigma abilita per gli operatori scenario innovativi di deployment di una rete (e dei suoi servizi). In una zona del mondo dove non è già presente una infrastruttura di rete, ad esempio un paese in via sviluppo, è possibile immaginare uno scenario di deployment innovativo e slegato dalle pratiche attuali. Questo scenario, che possiamo chiamare *network upload*, consiste nel progettare la nuova rete e i suoi servizi in una infrastruttura privata (ad esempio un DataCenter). Il fornitore di servizi può utilizzare questa infrastruttura privata per la fase di progettazione, prototipazione e testing dei servizi. Una volta che questi servizi sono stati sufficientemente testati, l'intera rete con i relativi servizi viene migrata nella zona dove si devono installare i servizi. La "progettazione interna" e "deployment al'esterno" in due passi distinti e separati fra di loro comporta un notevole risparmio in termini di costi.

Questo scenario è reso possibile in una infrastruttura fisica di commodity HW: le funzioni di rete sarebbero come delle VM che si possono migrare a piacimento.

1.5 Problematiche da affrontare con il nuovo paradigma

Un problema sarà nel dover creare le funzioni di rete. Sarà necessario "re-inventare la ruota"? Ovvero tutte le funzioni che sono state sviluppate dai vendor e inserite nelle loro middle box – e quindi segreto industriale- dovranno essere re-inventate da capo? Una ipotesi possibile è che si vengano a creare nuovi attori di mercato che svolgano il ruolo di SW vendor di funzioni di rete, oppure potrebbero crearsi dei gruppi di lavoro collettivo che sviluppano SW di rete opensource. Ne è un esempio l'Open Source Routing group, un nuovo progetto di Google e dell'Internet Systems Consortium che ha ripreso lo sviluppo di Quagga, un software di routing open source che implementa i protocolli OSPF e RIP.

Un altro problema è: quanto deve essere effettivamente "generico" l'HW di questi nodi di rete commodity per ospitare le funzionalità di rete con un buon livello di flessibilità? Un ipotesi estrema è che l'HW sia completamente generico, simile all'HW dei server nei DataCenter: questo in realtà non è conveniente a livelli di performance. Un'altra soluzione potrebbe essere che il nodo sia formato da una parte ottimizzata per certe funzionalità (come il piano dati degli switch realizzato in HW) e altamente performante, e un'altra parte, più generica, possa essere programmata per ospitare le funzioni di rete.

1.6 Le nuove teorie di ricerca

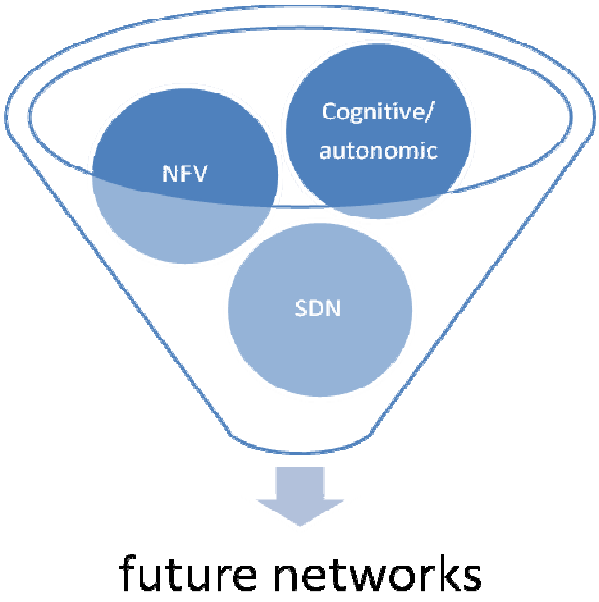
Pensiamo che la rete di domani prenderà spunto da una combinazione di tre aree di ricerca: Software Defined Networks, Network Function Virtualization, e cognitive-autonomics.

Software Defined Networks. Queste tre teorie sono concetti diversi, ma spesso si intersecano fra di loro. La vision di SDN è di disaccoppiare il piano di controllo di rete (logico) dal piano di forwarding (hardware) dai dispositivi di rete e spostare questo piano di controllo in un controller software.

Network Function Virtualization consiste nel rendere sempre più virtuali le funzioni, le applicazioni e le risorse di rete.

Gli aspetti cognitive/autonomici consistono nell'introdurre negli elementi di un sistema caratteristiche di auto-adattamento e auto-configurazione, anche tramite autoapprendimento non supervisionato.

Nel capitolo seguente verranno approfondite queste tre aree di ricerca, con particolare attenzione a SDN.



Capitolo 2

Le nuove teorie di rete

In questo capitolo di descrivono nuove teorie di rete, fra cui Software Defined Network, Network Function Virtualization. Si descrive il concetto di sistema autonomico – cognitivo e come questa teoria si applichi alle reti di domani.

2.1 Software Defined Networks

Ancora non esiste una definizione universalmente accettata di Software Defined Networks (SDN), ma si possono trovare molti punti in comune fra le definizioni che si trovano in rete. Un punto di riferimento è il whitepaper della Open Networking Foundation (ONF), e inoltre su internet si trovano molti talk di esperti di rete e professori universitari che ne danno una definizione in base alla loro esperienza di ricerca. SDN è un concetto ancora acerbo, poiché necessita di una standardizzazione e i prototipi devono ancora raggiungere una maturità tecnologica.

2.1.1 Percezione di SDN nel mercato dei dispositivi di rete

Dal punto di vista del mercato bisogna far presente che c'è una gran confusione sul significato di SDN: ad esempio i Vendor di dispositivi di rete, nel tentativo di cavalcare l'hype che SDN sta generando negli ultimi anni, hanno utilizzato la definizione di SDN in modo molto generico adottandolo di volta in volta ai loro prodotti. Questo trend ha portato a far sì che SDN significasse praticamente qualunque cosa di innovativo nel campo delle reti, generando confusione e cinismo sull'utilizzo di questo termine. Una interpretazione è quella dell'ONF, che declina SDN dal punto di vista di OpenFlow, il loro protocollo di rete (descritto nel capitolo 2.4).

2.1.2 Il routing e il forwarding

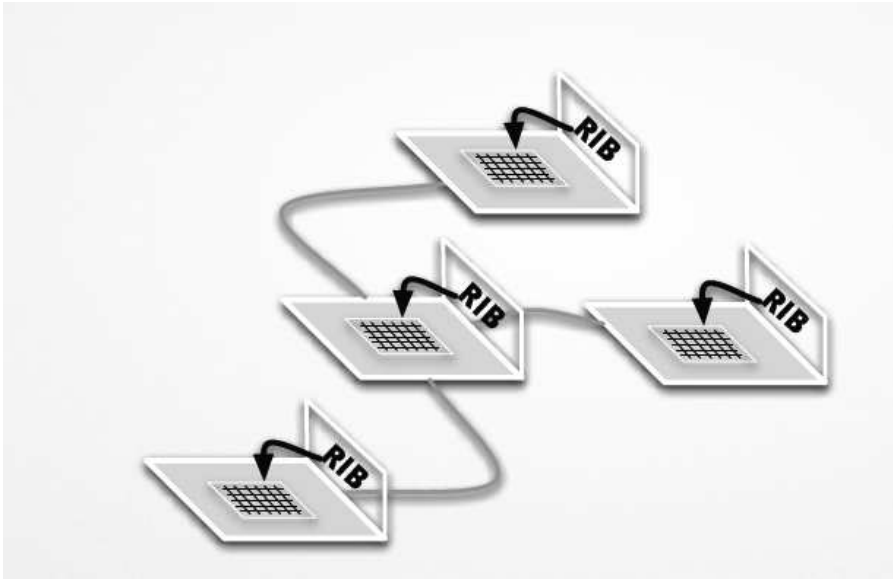
Prima di approfondire il concetto di SDN conviene capire come viene fatto oggi il routing e il forwarding in un'architettura di rete tradizionale.

I nodi di commutazione, cioè i nodi che usano i mezzi trasmissivi per creare canali di comunicazione, si possono dividere a livello funzionale in "piano di controllo" e "piano di forwarding" (piano dati).

Il piano di controllo contiene la funzione di instradamento, o routing, che è standard. La funzione di routing è quella parte che si occupa di calcolare e determinare le rotte dei pacchetti, ovvero quale percorso dovrà seguire un pacchetto nella rete per raggiungere la sua destinazione. Essa è composta da:

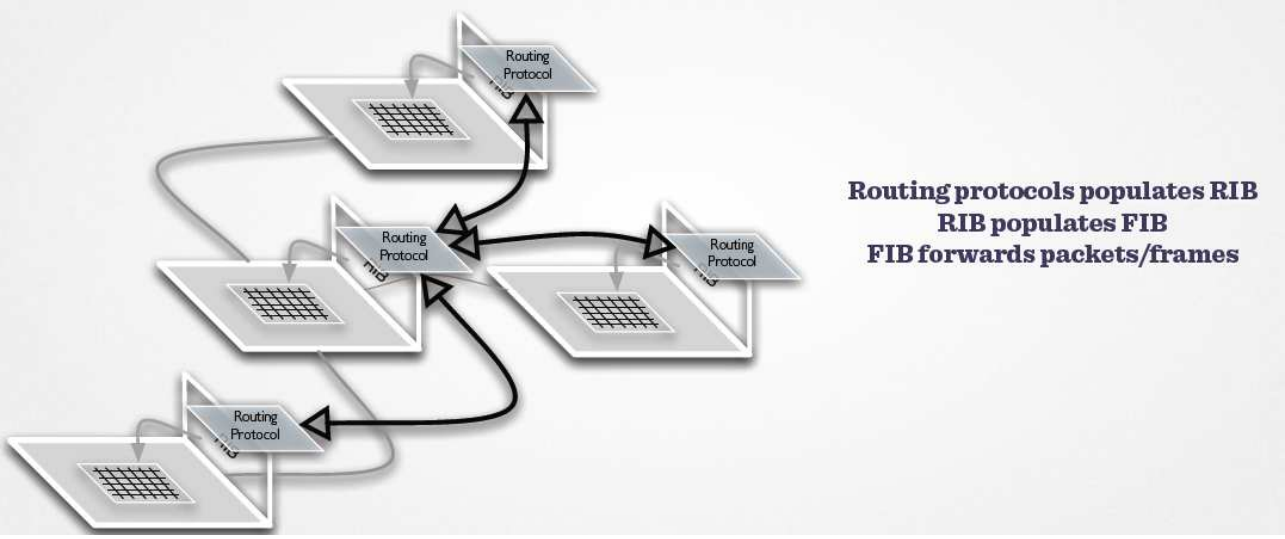
- Algoritmi di routing: algoritmi che calcolano le rotte da seguire a partire informazioni che hanno sulla topologia di rete
- Protocolli di routing: protocolli usati per lo scambio di informazioni relative alla topologia di rete con i nodi vicini
- La funzione di routing genera la tabella di routing (Routing Information Base) che contiene tutte le informazioni calcolate. Opzionalmente la tabella di routing viene compilata in tabelle di inoltro ad alte prestazioni: le Forwarding Information Base (FIB). La FIB è molto performante e solitamente realizzata con opportune tecniche in HW (TCAM).

Il piano dati consiste nell'azione da compiere sul pacchetto in arrivo. Questa azione, come l'inoltro del pacchetto a una porta specifica, viene eseguita in base alle tabelle (la RIB o la FIB).

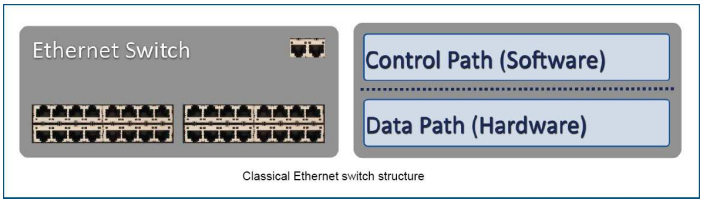


(immagine tratta da http://www.ipspace.net/)

Nella rete ogni nodo comunica con i vicini tramite i protocolli di routing, calcola la propria tabella di routing e la compila nella propria tabella di instradamento ad alte prestazioni. L'esecuzione degli algoritmi e il calcolo delle tabelle viene fatto in locale, in modo indipendente.

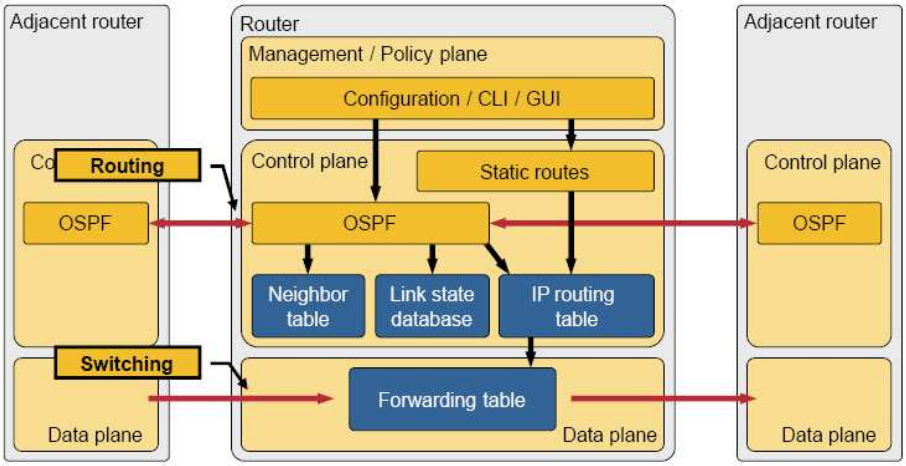


Le figure seguenti mostrano il modello degli apparati di rete (switch e router) tradizionali, dove il piano di switching, di controllo e di management è tutto incapsulato nello stesso apparato di rete.



(immagine tratta da http://www.koto.fr/le-sdn-donne-de-lintelligence-au-reseau/)

Nello switch è presente il piano dati e il piano di controllo. Il piano dati (data path) è ottimizzato per fare il forwarding il più veloce possibile, mentre il piano di controllo contiene il comportamento dello switch, sotto forma di semplici regole di forwarding.



Suddivisione funzionale di un router

Il router invece al suo interno contiene anche funzionalità di livelli superiori, sempre incapsulate all'interno del dispositivo, come il routing e il management.

L'aspetto da rimarcare è che sia nel router e che nello switch le funzionalità sono completamente contenute nel dispositivo di rete e agiscono in maniera autonoma rispetto alle funzionalità contenute negli altri nodi di rete. Questo ci porta al concetto di autonomous systems.

2.1.3 Gli autonomous systems

L'approccio tradizionale nella costruzione di una rete prende spunto dagli Autonomous Systems, ovvero un insieme di tanti elementi autonomi e

indipendenti fra loro. Il meccanismo di consegna dei messaggi fra un nodo sorgente e un nodo destinazione consiste nell'inoltro del messaggio attraverso vari hop intermedi, dove ogni hop ha un conoscenza parziale del percorso che seguirà il messaggio. Questo approccio permette grande scalabilità ed estensibilità. Se ogni nodo è indipendente si possono aggiungere nuovi nodi alla rete ed estendere la topologia senza dover andare a modificare i nodi già presenti.

Ma questo approccio ha dei limiti: l'identità dei nodi è strettamente legata alla loro locazione nella topologia fisica. Cambiare la locazione fisica della destinazione, ovvero cambiare la posizione del nodo destinazione all'interno della topologia, comporta per il meccanismo di consegna dei pacchetti anche un *cambio di identità* della destinazione.

È molto difficile implementare funzioni di rete come la QoS, raggruppamento logico o specificare aspetti relativi a flussi di pacchetti.

L'IETF, per venire incontro alle limitazioni relative al problema dell'identità logica, ha rilasciato alcuni standard come le virtual LAN o le VPN, ma sono comunque delle soluzioni isolate che risolvono specifiche necessità e inoltre aggiungono complessità alla configurazione di un apparato di rete.

2.1.4 Il paradigma Software Defined Networks

Il paradigma Software Defined Networks (SDN) propone di:

- rendere i nodi di rete programmabili
- introdurre delle astrazioni che permettono di generalizzare e modellare la rete e il suo comportamento a prescindere dalla specifica implementazione dei nodi
- accedere ai nodi tramite delle interfacce standard, simili alle API dei software

Questi obiettivi si ottengono operando una centralizzazione logica del controllo (si parla di controller centralizzato) e una apertura delle interfacce dei nodi della rete.

Il paradigma SDN consente di attuare lo step successivo, che è quello della virtualizzazione della rete, ovvero realizzare delle partizioni di rete virtuali della rete fisica. Ogni partizione poi potrà essere assegnata a un

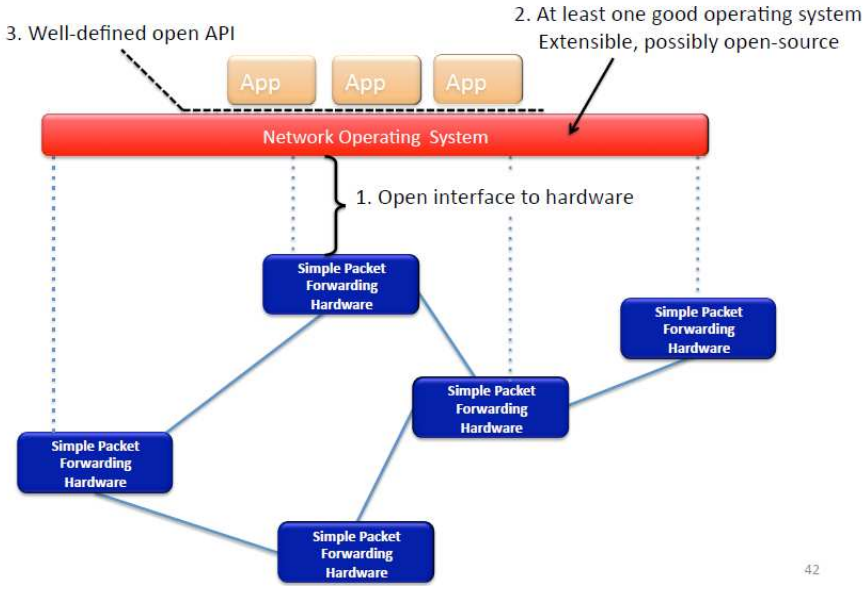
"controller" separato, e applicazioni che si appoggiano a controller differenti vedranno reti differenti.

2.1.5 Interpretazioni alternative di SDN

Vi sono diverse interpretazionidi cosa si intende per Software Defined Networking: di seguito vengono presentate tre possibili interpretazioni, quella di McKeown, quella di Shenker e quella di Heller, non necessariamente in conflitto fra di loro.

McKeown definisce SDN come un modo per fare il "re factoring" delle funzionalità di rete, ovvero un modo diverso di allocazione di queste nell'infrastruttura di rete fisica. Nelle reti di oggi le funzionalità sono allocate nelle middlebox, con SDN queste funzionalità vengono astratte e si separano dall'HW sottostante, "aprendo" le middlebox.

La figura sottostante mostra come queste funzionalità sono astrattate e si può parlare di un Network Operating System (concettualmente simile a un controller) funzionalmente separato dall'infrastruttura fisica.



(da Wikipedia)

Shenker invece sostiene che bisogna definire SDN in base alle astrazioni che la rete fornisce al software e di conseguenza alle persone che scrivono il software. Il piano di controllo, in software, necessita di astrazioni per consentire di risolvere problemi architetturali e abilitare un' "evoluzione" della rete. Secondo Shenker le reti attuali sono molto complesse e funzionano solo

perché c'è gente in grado di "gestire la complessità", ma in realtà si dovrebbe "estrarre la semplicità".

La transizione deve ripercorrere quella che si è verificata per i linguaggi di programmazione: con l'evolversi dei linguaggi di programmazione ci si è sempre di più slegati dall'HW e sono state aggiunte astrazioni come il sistema operativo (che fornisce l'astrazione dei file, della memoria virtuale, etc) o nuovi paradigmi di programmazione, come il garbage collection, i thread, gli oggetti, gli attori, gli agenti, etc

Si possono definire delle astrazioni per:

- il forwarding, così che il *forwarding behaviour* dei nodi possa essere definito da un controller
- lo stato distribuito: Network Operating System potrebbe fornire alle applicazioni una "global network view"
- le specifiche di comportamento di alto livello per la rete(delle policy, dei goal, dei task, etc.)

Infine Heller definisce SDN come un paradigma di rete che garantisce una maggiore flessibilità, e questa flessibilità si può esprimere attraverso delle dimensioni, come se fossero assi cartesiani.

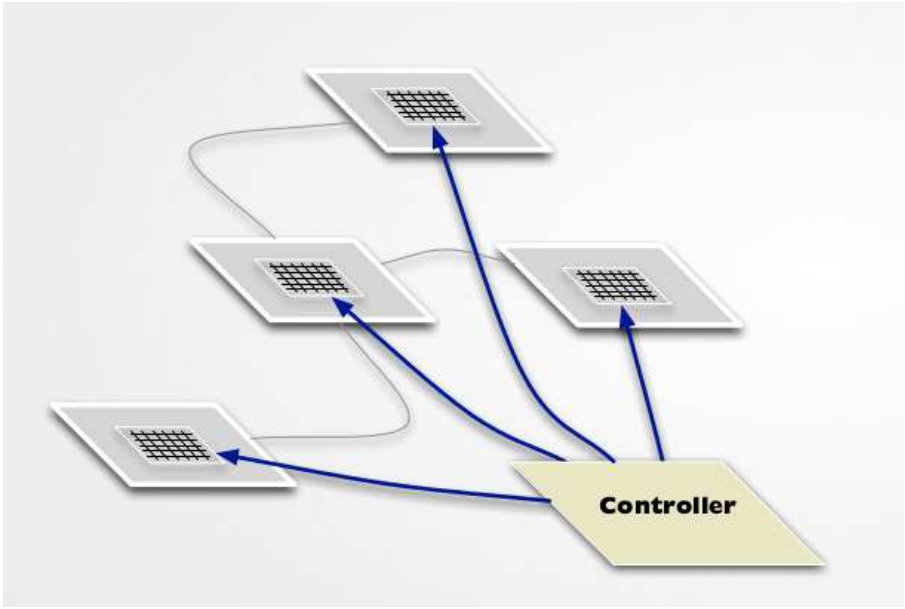
- Controllo centralizzato vs distribuito: possiamo avere un controller che si preoccupa di controllare un singolo nodo, un gruppo di nodi o un'intera rete. Inoltre il controller a sua volta potrebbe essere logicamente centralizzato, ma essere fisicamente distribuito su dei nodi di rete.
- Gestione del traffico a diversi livelli dettaglio: possiamo gestire singoli flussi di traffico specificando delle regole nel controller o interi gruppi di flussi utilizzando tecniche di aggregazione e regole wildcard
- Controllo reattivo o proattivo: il conrtroller della rete potrebbe essere reagire solamente a degli eventi o agire attivamente sulla rete, oppure una combinazione di entrambi
- Rete fisica o virtuale
- "Fully consistent" vs "eventually consistent": garantire l'efficienza e la consistenza allo stesso tempo dello stato di un sistema distribuito è un problema molto complesso, specialmente se il sistema e composto da

molti elementi. A seconda della situazione potrebbe essere sufficiente avere “eventual consistency”, ovvero la garanzia che prima o poi, senza sapere precisamente quando, la consistenza verrà raggiunta. Queste soluzioni a loro volta possono portare dei problemi se mal progettate, ad esempio il rischio di route flapping (situazioni in cui delle informazioni di raggiungibilità sono ripetutamente fornite e ritrattate).

Bisogna far presente che molte delle caratteristiche sopra elencate non sono mutualmente esclusive: ad esempio potremmo avere un controllo proattivo della rete, ma anche capacità di reagire agli eventi che si verificano nella rete.

2.1.6 La centralizzazione del controller

Un punto importante nel concetto delle Software Defined Networks consiste nel ripensare a come deve avvenire il controllo rispetto all’approccio tradizionale: il controllo della rete deve essere disaccoppiato dall’HW e posto in un controller SW logicamente centralizzato. Togliendo il vincolo che l’intelligenza del dispositivo debba risiedere nel dispositivo stesso posso spostarla anche al di fuori da esso: ad esempio posso avere un nodo che controlla altri nodi della rete. Questo approccio è profondamente disruptive, poiché non ci si rifà più agli autonomous systems: i nodi non decidono più in autonomia il loro comportamento, ma sono controllati da un’entità esterna.

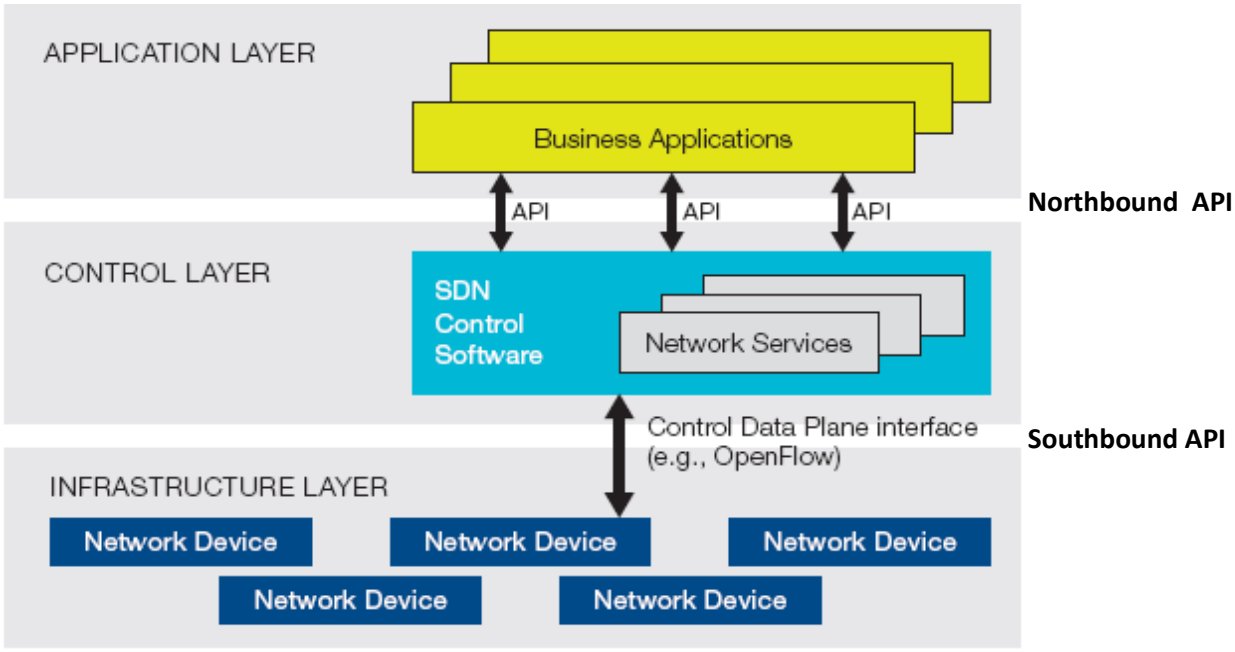


Il vantaggio di avere un controller SW centralizzato è quello di avere completo controllo sulla rete: modificando il SW posso fare in modo che la rete abbia un comportamento differente e ho un unico punto di controllo che mi permette di avere una visione dell'intera rete.

Le applicazioni infine interagiscono con la rete attraverso il controller che dispone di una visione centralizzata a consistente della rete.

Bisogna evidenziare tuttavia come la centralizzazione del controllo non è la cosa fondamentale: la vera novità di SDN sta nello svincolare il controllo dalla topologia fisica dai nodi della rete. Il controllo infatti può essere logicamente centralizzato, ma fisicamente distribuito.

2.1.7 L'architettura di SDN



(immagine tratta da "Software-Defined Networking: The New Norm for Networks")

Dal punto di vista funzionale l'approccio SDN identifica 3 livelli: l'infrastruttura di rete di rete fisica, il piano di controllo e le applicazioni. Le applicazioni interagiscono con il controller (tramite northbound API), mentre il controller interagisce con i nodi della rete fisica (tramite le southbound API). In realtà Northbound e SouthBound API sono due termini che si rifanno a una implementazione specifica, quella del protocollo OpenFlow (protocollo che implementa SDN e descritto successivamente) ma si possono generalizzare a un contesto più ampio.

SDN è un approccio di progettazione o un'architettura di rete? Probabilmente è entrambe: una rete SDN richiede che i nodi della rete vengano progettati specificamente per essere controllati dal controller e la rete nel suo complesso deve avere un'architettura a 3 livelli.

2.1.8 Strato di controllo

Lo strato di controllo è quel software che si occupa di controllare la rete: conosce lo stato della rete, cioè lo stato di tutti i suoi nodi, e può modificarlo. Può essere un ecosistema di componenti SW, dove ognuno si occupa di gestire una determinata funzionalità. Questi componenti SW possono essere allocati in un singolo nodo, oppure possono essere distribuiti su più nodi, in quest'ultimo caso si parla di controller distribuito. La teoria SDN non pone vincoli alla natura del controller: a seconda dei contesti potrebbe essere più conveniente centralizzato o distribuito. Il controller deve poter comunicare con tutti i nodi della rete: il nodo di controllo potrebbe avere essere fisicamente collegato – tramite link diretti – a tutti gli altri nodi della rete, oppure potrebbe sfruttare una connettività multi-hop, cioè il nodo controller sfrutta nodi intermedi per raggiungere i nodi più lontani. Inoltre se il traffico di controllo è separato dal resto del traffico, oppure se la rete fisica di controllo è una rete separata rispetto alla rete di produzione, si parla di out-of-band control.

2.1.9 Visione logica della rete

Come si è detto, con SDN la rete è controllata da un unico controller centralizzato, e non è più un sistema complesso distribuito. Il controller deve fornire alle applicazioni una visione della rete che sia unica, globale (ovvero di tutta la rete che controlla) e consistente. Si astrae quindi dall'infrastruttura fisica della rete e si introduce un' "astrazione di rete".

Al contempo sorgono nuove domande:

- come può il controller ottenere (e mantenere) una visione consistente della rete?
- quali meccanismi deve attuare il controller per impostare uno stato nella rete, ad esempio modificare il comportamento di un nodo?

Per comunicare con la rete si può pensare a un protocollo di comunicazione, ed è quello che è stato fatto per il protocollo OpenFlow. Tramite un protocollo di comunicazione al controller vengono notificati specifici eventi (ad esempio la modifica dello stato di un nodo della rete) e il medesimo può inviare dei messaggi ai nodi della rete per modificare il loro stato.

2.1.10 Problematiche da affrontare

SDN introduce nuove problematiche: la separazione fra controller e applicazioni, la scalabilità di una rete SDN e la consistenza della rete. Di seguito sono descritti uno per uno.

2.1.10.1 Separazione fra controller e applicazioni

Non c'è una definizione precisa di dove finisca il controller e di dove comincino le applicazioni, ma il controller sicuramente deve essere quella parte che comunica con la rete fisica, che traduce la logica delle applicazioni nel comportamento che devono assumere i nodi della rete. A seconda delle implementazioni il controller potrebbe essere esso stesso una applicazione oppure sorta di sistema operativo di rete, una sorta di NetOS, a cui si interfacciano le applicazioni.

2.1.10.2 La scalabilità del controller SDN

Il controller di una rete SDN è in definitiva quella applicazione che si occupa di controllare la rete. Inizialmente conviene pensare al controller come centralizzato, quindi come una applicazione che gira su un nodo della rete, collegato direttamente o indirettamente a tutti gli altri nodi. All'aumentare delle dimensioni della rete tuttavia possono sorgere problemi di performance: un singolo controller potrebbe non essere sufficiente a gestire un'intera rete. A questo punto conviene pensare il controller sempre come logicamente centralizzato, ma fisicamente distribuito su più nodi della rete. La fattibilità di

rendere scalabile un controller è un problema ancora aperto e soggetto di ricerca.

Per reti di dimensioni limitate, ad esempio un campus universitario oppure una rete di una piccola azienda, un controller centralizzato potrebbe funzionare.

Per OpenFlow ad esempio è stato sviluppato HyperFlow, un control plane centralizzato che gira su HW distribuito.

2.1.10.3 Consistenza della rete

All'aumentare delle dimensioni della rete potrebbe essere complicato per un controller centralizzato mantenere la consistenza di una intera rete. Sono necessari ulteriori studi per capire se capire se questo sia possibile.

2.1.11 Il meccanismo di funzionamento di una rete SDN

L'unica implementazione di una rete SDN attualmente è il protocollo OpenFlow (descritto in un capitolo a parte). Prescindendo dalla specifica implementazione possiamo identificare dei meccanismi generali che devono essere presenti in una rete SDN.

In una rete SDN il controller riceve lo stato della rete, ovvero lo stato di ciascun nodo, e modifica il comportamento dei nodi per istruirli su come comportarsi in situazioni specifiche.

Il controller riceve dai nodi dei messaggi di notifica sugli eventi che accadono nei nodi stessi. Gli eventi potrebbero essere ad esempio:

- modifica dello stato del nodo, ad esempio rottura di un link con un altro nodo
- arrivo di pacchetti per i quali il nodo non sa come comportarsi

Le configurazioni che il controller impartisce alla rete possono essere ad esempio:

- istruire un nodo su come comportarsi in presenza di un determinato pacchetto
- inserimento di policy generiche

Si parla di comportamento reattivo se il controller reagisce a un evento compiendo una determinata azione. Ad esempio il controller reagisce alla richiesta di un nodo su come gestire un certo tipo di traffico sconosciuto dicendogli di scartarlo.

Si parla di comportamento proattivo se il controller di sua volontà e in assenza di eventi esterni agisce per modificare lo stato della rete.

2.1.12 Posizione dei Player (H/W e S/W Vendor, Operatori, Accademia, Enterprise) rispetto a SDN

Dopo una visione teorica di SDN conviene dare uno sguardo all'approccio che hanno i vari player nel mondo della rete su SDN.

I vendor (Cisco, Juniper, IBM, Dell, HP,..) che producono dispositivi di rete hanno un modello di business verticale, ovvero basato su uno stack di tecnologie proprietarie e spesso incompatibili con quelli di altri produttori. Per questa ragione vedono SDN come una minaccia in grado di rovinare il loro business, specialmente la versione più estrema di SDN che vede i dispositivi di rete come commodity HW a basso costo. Come alternativa a questa visione così "disruptive" i vendor propongono la loro "versione" di SDN, che consiste nel dotare le loro piattaforme di un certo livello di programmabilità, seppure molto limitata e poco integrabile. Sono inoltre molto restii a sviluppare soluzioni H/W compatibili con OpenFlow, infatti il numero di apparati che supporta OpenFlow e presente ora in commercio è ancora molto limitato. Le soluzioni SDN proposte dai vendor ora in commercio consistono in una "converged infrastructure" che integra OpenFlow ma aggiungendo soluzioni proprietarie; questo rende le loro soluzioni di rete tali da renderle poco compatibili anche con dispositivi OpenFlow-compatibili di altri produttori. La loro strategia non è passiva alla situazione attuale del mercato: molti vendor stanno infatti acquisendo delle startup che offrono soluzioni in questo campo, ad esempio Cisco ha recentemente acquisito la startup Insieme. In questo modo i vendor puntano ad acquisire il know how e a rimanere competitivi allargando la propria offerta alle nuove tipologie di richieste.

Grandi aziende come Google, Facebook, Amazon e Microsoft si sono costruiti da soli gli apparati di rete che supportano SDN perché i Vendor non li

fornivano in commercio. È diventata famosa la rete di Google che utilizza OpenFlow: in questo si tratta di una rete WAN che collega fra di loro i DataCenter Google e OpenFlow consente di operare del traffic engineering.

Le startup (es. Nicira, BigSwitch, Vello Systems, …) vedono SDN e in particolare OpenFlow come un mezzo per sviluppare soluzioni "green-field" legate al mondo delle reti, ovvero pensate e progettate con un approccio completamente innovativo e slegato dalle pratiche precedenti.

I fornitori di servizi di Cloud vedono l'SDN come una possibilità di sfruttare meglio i propri data center, aggiungendo flessibilità e maggiore facilità di gestione alle loro infrastrutture.

Le università e gli istituti di ricerca vedono OpenFlow e l'SDN come un modo per "aprire" le piattaforme di rete chiuse e proprietarie, in modo da avere più libertà di fare ricerca e sperimentazioni. Molti campus universitari recentemente hanno fatto richiesta di apparati che abbiano come requisito quello di supportare OpenFlow. Un vantaggio di SDN in questo senso sarà quello di portare innovazione nel campo delle reti.

I S/W Vendor (es. VMWare) sviluppano soluzioni di virtualizzazione per i Datacenter. Recentemente VMWare ha acquisito la startup Nicira, una startup pionere nel campo SDN/OpenFlow, che si era specializzata nella realizzazione di plugin SW in grado di "aprire" i dispositivi di rete proprietari.

I silicon vendor (IBM, ARM) vedono SDN come una ottima opportunità incrementare il loro mercato. Anche se queste aziende non producono sistemi completi (server) ma realizzano solo i componenti elettronici, vi sarebbe per loro un vantaggio indiretto, in quanto nodi di rete commodity HW richiederebbero la presenza di componenti generici , prodotti dai silicon vendor.

Gli operatori di rete ancora non hanno preso posizione sull'argomento SDN e stanno valutando le possibili alternative e scenari, dove SDN potrebbe essere utile all'edge e/oppure al core della rete dell'operatore. Vedono comunque tre declinazioni di SDN:

- quella proposta dai Vendor, dove si hanno dispositivi chiusi e proprietari dotati di una certa programmabilità e dotati di API proprietary

- quella di dispositivi semiprogrammabili dotati di interfaccia vendor independent
- Una visione di rete futuristica di nodi completamente commodity, e funzioni di rete completamente via SW

2.2 La virtualizzazione di rete

Come si è detto, SDN è un approccio che consente di abilitare la virtualizzazione di rete. Questo termine ha molteplici significati: si può intendere come virtualizzazione della topologia, delle applicazioni, dei servizi di rete, delle policy di rete. In generale si parla di virtualizzazione di funzioni di rete, o *Network Function Virtualization* (NFV), perché questo termine di fatto include tutti i casi sopra elencati. La definizione di riferimento di NFV è il whitepaper "*Network Function Virtualization*", un documento che stato redatto da un insieme di operatori, compreso Telecom Italia e descrive la NFV come la capacità di allocare delle funzionalità di rete a piacimento su nodi della rete come se fossero delle entità SW astratte, componibili e estensibili.

Si può fare un parallelo con le tecniche di virtualizzazione dei sistemi operativi: tramite virtualizzazione si riesce a condividere le risorse HW fra diversi sistemi operativi sotto forma di macchine virtuali. Ogni virtual machine (VM) crede di avere il pieno controllo sull'HW sottostante, ma in realtà fra l'HW e la VM è presente un HyperVisor, un componente SW che si pone in mezzo fra i due strati, regola l'accesso alle risorse e garantisce l'isolamento delle VM fra di loro.

Allo stesso modo si può pensare di avere sulla stessa rete fisica tante reti virtuali, o *slice*. Ogni *slice* è controllata da una diversa istanza di controller. In questo modo si possono avere diversi utenti che hanno una visione differente della rete, utilizzano protocolli di rete differenti o schemi di indirizzamento differenti.

I progressi fatti fino ad ora nella virtualizzazione dei sistemi operativi sono stati notevoli e hanno riscosso grande successo nei DataCenter permettendo l'evoluzione dei servizi di cloud. In questi Data Center sono

presenti un gran numero di nodi server connessi fra di loro e che contengono applicazioni distribuite. Le applicazioni distribuite consistono di Virtual Machine che risiedono su differenti nodi fisici della rete e interagiscono fra di loro. Attualmente nei DataCenter sono già utilizzate anche tecniche di virtualizzazione di rete, come gli switch virtualizzati (vSwitch).

Per le reti di telecomunicazione geografiche invece le tecniche di virtualizzazione devono essere molto più complesse: si richiede la capacità di istanziare, muovere o orchestrare dinamicamente degli insiemi di Virtual Machines (VM). Queste VM conterranno servizi di rete oppure applicazioni d'utente.

Tramite la virtualizzazione di rete è possibile operare un overprovisioning della connettività: spostando la connettività sul piano logico posso aumentarla di molto rispetto ai link della rete fisica sottostante. In altre parole posso aumentare il numero di connessioni (logiche) rispetto ai link disponibili (fisici). Questo è possibile perché su un router fisico ho tanti virtual router, ognuno connesso a una sua rete logica.

2.3 Teorie Autonomiche/Cognitive

Ci sono già delle aree di ricerca in cui sono studiate le teorie autonomiche/cognitive. Software defined radio ha studiato come ottimizzare lo spettro disponibile nelle comunicazioni wireless, introducendo capacità cognitive: lo studio riguardava come manipolare in modo autonomo lo spettro radio di una singola antenna per ridurre le interferenze con altre antenne.

Self Organizing Networks è un'altra area di ricerca che parte dale tecnologie attuali come LTE/4G e cerca di introdurre in queste l'aspetto di orchestrazione. I vendor forniscono dispositivi con algoritmi di ottimizzazione locale ma spesso questi meccanismi non sono in grado di "parlarsi" se i device stati realizzati da produttori differenti, e in ogni caso mancano sempre dei meccanismi di coordinazione a livello globale. Con le SON si vuole introdurre un meccanismo di ottimizzazione globale in modo che l'efficienza di queste antenne possa beneficiarne globalmente e non solo nelle comunicazioni punto-punto fra i singoli nodi.

In questa sede si vuole invece esplorare la teoria delle reti cognitive e autonomiche non limitandosi agli aspetti tecnici della ottimizzazione della banda, ma in un contesto più generale dove l'operatore deve fornire dei servizi agli utenti.

2.3.1 Comportamenti di un sistema autonomico

Un sistema autonomico realizza una auto ottimizzazione operando su più livelli:

1. Comportamento automatico: i nodi seguono delle semplici regole seguendo un comportamento reattivo
2. Comportamento autonomico: i nodi hanno dei meccanismi di autoapprendimento dall'ambiente e di risultati delle proprie azioni (si parla di "learning unsupervised") e questo porta in ogni nodo alla modifica delle regole presenti nel nodo o alla creazione di nuove regole. Si tratta di un meccanismo di ottimizzazione locale.
3. Comportamento di orchestrazione: l'orchestratore fornisce delle direttive generali ai tutti i nodi del sistema, in genere sotto forma di policy o linee guida. Si parla di ottimizzazione globale.

Il sistema autonomico realizza un trade-off fra le ottimizzazioni locali e quelle globali: quello che può accadere è che il comportamento autonomico globale porti a una leggera disottimizzazione dei singoli elementi, ma nel complesso risulta molto più importante ottimizzare a livello globale.

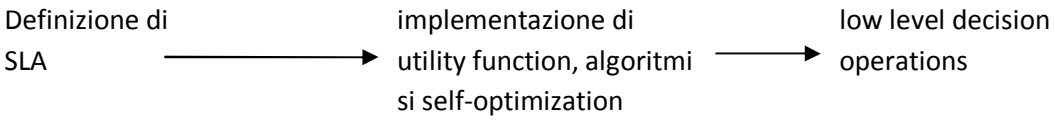
2.3.2 Utility functions nei sistemi autonomici

Le *utility function* sono un modo per ottenere l'auto-ottimizzazione dei singoli elementi in un sistema autonomico. Sono state utilizzate precedentemente anche in campo economico o in intelligenza. Una utility function fornisce la funzione obiettivo di un sistema autonomico.

L'utility function mappa ciascun stato (del sistema o del singolo elemento) in un valore scalare. Questo stato può essere descritto come un vettore di attributi.

Il sistema autonomico si troverà ad agire in un ambiente altamente dinamico, pertanto la ricomputazione di nuovi valori di utilità dalle *utility function* dovrà avvenire frequentemente. Spesso definire le utility function risulta un compito difficoltoso per un essere umano: è molto più conveniente

definire dei service level agreement (SLA) di alto livello. Il sistema autonomico poi dovrebbe essere sufficiente intelligente da tradurre questi requisiti astratti in opportune *utility function che*, congiuntamente agli algoritmi di ottimizzazione e a un'opportuna modellizzazione del sistema, forniscono il comportamento desiderato.



In un sistema autonomico a 2 livelli, ovvero dove sono presenti sia dei manager per le aree locali che l'ottimizzatore globale, le utility function possono essere usate per entrambi i livelli all'interno degli algoritmi di ottimizzazione.

Una utility function può essere a basso livello(service level-business terms) o ad alto livello(high-level business term), ovvero può specificare direttamente la quantità di risorse richieste (in un'ottica dove si deve ottimizzare la quantità di risorse) oppure può specificare l'utilità in termini di livello di servizio che contano per gli utenti o i fornitori del servizio, come un indicatore di Quality of Service (QoS).

L'utility function globale viene calcolata a partire da quelle locali e può essere calcolata in vari modi: come sommatoria, produttoria, somma pesata, etc. a seconda di come si modella il sistema.

In conclusione l'utility function deve riflettere l'aspetto che più ci interessa del sistema: questo significa che potrebbe essere una combinazione di più parametri del sistema, che nel complesso riflettono una determinata caratteristica.

2.3.3 Aspetti autonomici di DataCenter

Recenti studi si sono occupati di applicare approcci autonomici all'interno dei DataCenter. Molti di questi aspetti torneranno utili nello scenario di rete autonomica descritto nel capito 3.2 . L'aspetto autonomico in

un Data Center consiste nel gestire in maniera ottimale e autonoma le risorse facendo un'allocazione auto-ottimizzata basata su utility function.

Identifichiamo i seguenti ruoli funzionali in un DataCenter gestito con approccio autonomico:

- *Application environment*: all'interno del DC vi sono tanti application environment, che sono dei raggruppamenti di VM che sono collegati fra di loro a livello funzionale. Queste VM si occupano di fornire lo stesso servizio, oppure sono possedute dallo stesso tenant (l'utente del servizio fornito dal DC) , o cooperano per portare a termine uno stesso task. Sono insomma strettamente collegate fra di loro.
- *Application manager*: viene descritto come un componente che si occupa di gestire un singolo application environment.
- *Resource arbiter*: componente che a livello di DC si occupa di ottimizzare le risorse. Agisce indirettamente sui singoli application environment inviando delle direttive agli *Application manager.*

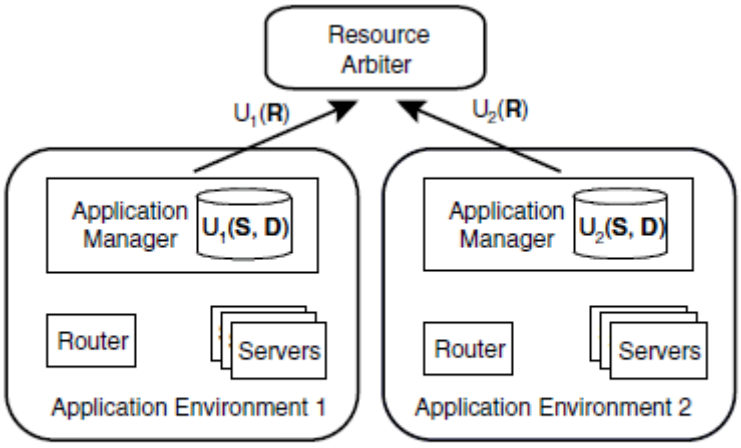


Figure 1. Data center architecture.

(immagine tratta dal paper "Utility function in autonomic systems")

Considerando lo scenario DC dal punto di vista autonomico si possono identificare due livelli di gestione autonomica: il Resource arbiter orchestra gli Application manager, mentre in ciascun Application environment l'application manager controlla le VM a lui preposte.

Di seguito si descrive il funzionamento del DC autonomico.

Un AE fornisce un distinto servizio; ogni AE possiede una *service-level utility function* che è stata ricavata dai SLA per quel AE. Il valore restituita dalla funzione di utilità indica qual è il grado di aderenza alle SLA per quel determinato AE. Gli AE comunicano i loro bisogni all'arbitro. I bisogni sono espressi in modo uniforme, si astrae in questo modo dalle specificità dei singoli servizi. Il resource arbiter non sa nulla del funzionamento interno degli AE. Gli elementi del sistema, in questo caso gli Application Environment, gestiscono autonomamente il proprio comportamento, in altre parole gli AE fanno un'ottimizzazione locale delle risorse che sono state garantite loro dal resource arbiter globale. Le operazioni di tuning che gli AE compiono per ottimizzare localmente le proprie risorse, senza entrare troppo nel dettaglio, consistono nel modificare dei parametri di controllo interni.

Nel caso di studio preso in esame l'utility function per un singolo AE tiene conto di due parametri: la qualità di servizio (S) e la quantità di richieste (D). S è misurato in termini di tempo medio di risposta per alla richiesta di caricamento di una pagina web (una richiesta è considerata come una transazione), mentre D è la quantità di richieste per unità di tempo.

Capitolo 3

Sperimentazioni

In questo capitolo di descrivono le sperimentazioni che sono state condotte sulla migrazione di Virtual Machine. La prima sperimentazione si concentra sull'algoritmo di migrazione in se per individuare le relazioni fra le performance e i fattori di influenza della migrazione, mentre la seconda sperimentazione studia la migrazione di Virtual Machine come strumento di gestione delle risorse in un contesto di rete d'accesso autonomica.

3.1 Analisi delle performance di migrazione di VM

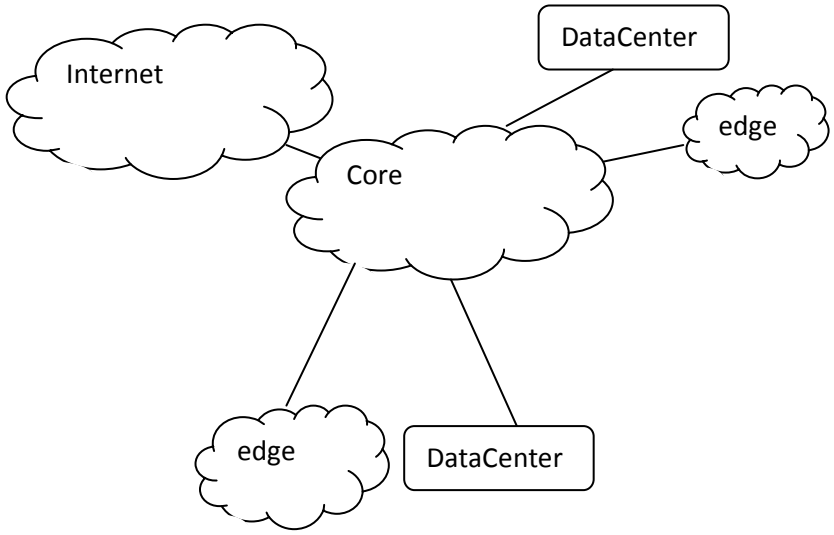
Per poter operare la virtualizzazione delle funzioni di rete è necessario disporre di tecnologie che consentano di migrare tali funzioni di rete. Attualmente gli apparati non consentono di istanziare o migrare funzioni di rete, quindi ci siamo concentrati su un altro campo: la virtualizzazione di VM nei Datacenter. Abbiamo cercato di capire come funzionano queste tecnologie e come si possano adattare ad un contesto di rete.

Nei DataCenter la virtualizzazione dei sistemi operativi all'interno di VM è una tecnologia matura e sono largamente in uso soluzioni che consentono di virtualizzare sistemi operativi nei nodi server dei Data Center e persino di migrare le VM intra-DC, ovvero fra nodi dello stesso DataCenter. Queste soluzioni sono ottimizzate per questo particolare contesto e l'obiettivo della sperimentazione è stato anche quello di capire se queste tecnologie consentano l'utilizzo anche in un ambiente differente da quello della rete privata e ad alte prestazioni dei datacenter. La differenza sarebbe che il nuovo ambiente da considerare è quello della rete d'accesso, ovvero la parte periferica di una rete di telecomunicazione geografica, dove si hanno minori garanzie di latenza e banda.

3.1.1 Contesto

Per questa prima sperimentazione si ipotizza il seguente contesto: l'operatore dispone delle risorse fisiche di rete o le affitta da un Physical Infrastructure Provider (PIP). Su queste risorse può allocare e gestire delle

risorse virtuali sia di rete (Virtual Routers/Switch) sia di processing (IT servers nei Data Centre o nel Cloud). Per semplificare ci riferiamo al dominio di un singolo Operatore. Le risorse sono dislocate in punti geografici diversi, secondo criteri strategici: ad esempio due DataCenter sono vicini rispettivamente a due reti d'accesso gestite dall'operatore.



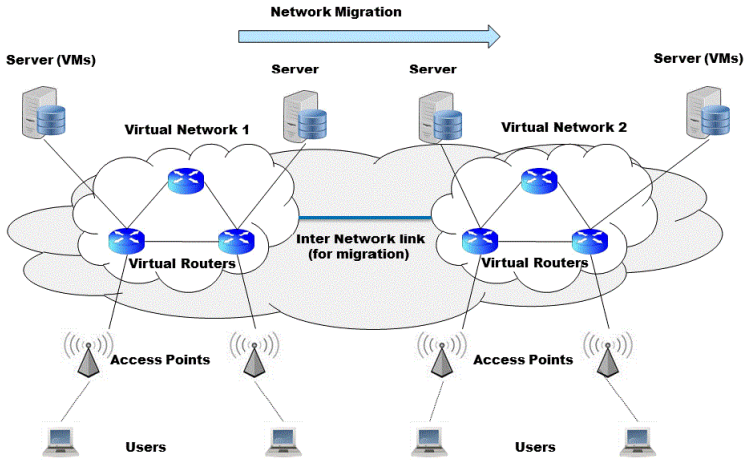
Ad un Cliente viene fatto il provisioning di un insieme di risorse virtuali di rete e IT: in particolare un gruppo di VM che virtualizzano le risorse IT (server) su cui girano le applicazioni del Cliente e un gruppo di VM che virtualizzano le risorse di rete (Virtual Routers/Switch), ovvero una *slice* di rete.

Queste risorse virtuali, VM server e slice di rete, devono essere in grado di adattarsi dinamicamente ai cambiamenti che si possono verificare (guasti, attacchi informatici, o ad esempio necessità dell'Operatore di effettuare azioni di traffic engineering, o hot spot mitigation). Tra gli eventi possibili che richiedono un adattamento delle risorse virtuali/fisiche consideriamo in particolare uno spostamento di un Cliente, verso altri network attachment point, nella rete di accesso.

Questo rende necessario che la VM su cui gira l'applicazione si sposti da un server verso un altro server magari in un DataCenter più vicino al Cliente per migliorare performance e che la slice di rete virtuale sia adatti conseguentemente.

In generale si potrebbero avere migrazioni di una o più VM attraverso reti WAN (quindi inter-DataCenter). Questo potrebbe avere un impatto sull'applicazione (QoE) che sta usando l'Utente.

La sperimentazione ha richiesto di capire come simulare al meglio le prestazioni della migrazione delle VM in funzione delle prestazioni della rete che supporta la migrazione (throughput, latency) e studiare quale modello matematico adottare.



3.1.2 Obiettivo della sperimentazione

L'obiettivo di questa sperimentazione è stato quello di esplorare e capire la fattibilità, attraverso specifiche simulazioni, dello spostamento "seamless", ovvero senza ricadute di prestazioni, degli insiemi di VM che virtualizzano le risorse di rete (slice) e IT – e capire l'impatto sulle applicazioni del cliente. Si può pensare ad esempio ad applicazioni di multi media streaming, gaming, applicazioni che per funzionare richiedono alte performance di rete. In particolare si vuole capire cosa cambia se la migrazione delle VM server è inter-data centre. Questo è importante perché ci fa valutare l'impatto della rete nella migrazione, ad esempio latenza, jitter, etc.

In questa fase ragioniamo in un'ottica di controller ideale, ovvero trascuriamo le specifiche problematiche per focalizzarci sulla visione d'insieme. Fare l'ipotesi di una rete SDN con un controller ideale permette di non considerare tutti i problemi di un controller reale, come la scalabilità e la latenza.

Per questo scenario è stata descritta una timeline e di seguito un modello matematico. La timeline elenca quali sono gli step temporali e descrive in ciascuno step come si comporta ciascun componente in gioco: la VM, l'utente, la rete fisica e quella virtuale.

3.1.3 Time Line

Di seguito è descritta la timeline:

- Nella fase iniziale avviene la fase di provisioning: l'operatore si accorda con l'utente su quali risorse (IT e di rete) fornirgli in base alle sue esigenze. In questo caso l'operatore si accorda con l'utente per allocare un'applicazione server, sotto forma di virtual machine, in un nodo all'interno della sua rete. L'utente potrà poi usufruire dell'applicazione collegandosi a quel nodo di rete che contiene la VM. Un esempio di applicazione potrebbe essere un server di un gioco online, a cui l'utente si connette tramite un'applicazione client. Prima della fase di provisioning delle risorse virtuali al Cliente, l'Operatore deve stabilire l'allocazione ottimale delle risorse virtuali sulle risorse fisiche, per minimizzare i costi e ottimizzare le performance. Un criterio potrebbe essere che la VM venga allocata in un nodo il più possibile vicino all'utente. Deve essere inoltre creato uno "strato di connettività" fra l'applicazione e l'utente, ovvero una *slice* di rete dedicata. È stato dimostrato che in generale il problema di fare il mapping ottimale di risorse di rete virtuali su un substrato di rete fisico può essere NP-Hard. Potrebbe essere necessario usare delle euristiche per approssimare una soluzione.

- A provisioning avvenuto, dopo un po' di tempo si verifica un evento, non programmato: il Cliente si sposta e cambia il network attachment point,

collegandosi a una nuova rete; l'evento rende necessario che la VM server si sposti verso un altro server più vicino al Cliente per migliorare la performance, opzionalmente in un altro DataCenter. La slice di rete virtuale di deve adattare conseguentemente.

Approfondiamo questo secondo passo:

- La rete si accorge dell'evento "l'utente si è spostato". Ad accorgersene potrebbe essere la nuova rete di accesso, cioè la rete a cui si è collegato l'utente che si è spostato, oppure la rete del Data Center, in ogni caso è un meccanismo reattivo. Può essere utile ragionare considerando due specifiche implementazioni di reti SDN: OpenFlow e ClickOS (una breve descrizione di ClickOS è presente in appendice).
    a. Se è una rete OpenFlow: il comportamento reattivo deriva dal comportamento classico di uno switch OF. Lo switch della nuova rete si vede arrivare un pacchetto sconosciuto e chiede aiuto al controller.
    b. Se è una rete con ClickOS: l'istanza di click router deve essere programmata per reagire all'evento di un pacchetto sconosciuto. Su un server XEN sono eseguite più istanze di Click router. L'evento verrà gestito da una delle istanze esistenti. Potrebbe essere creata una nuova istanza di click router per gestire la nuova utenza .

- Il controller deve ristabilire la connettività fra il client, che si è spostato, e la VM (non ancora migrata) che gira su un server IT. In altre parole, deve essere riadattata la slice di rete, cambiando il mapping logico-fisico.

- L'operatore identifica il server destinazione, ovvero il nodo dove verrà migrata la VM. Il criterio potrebbe essere sulla vicinanza dei nodi, sul livello di carico dei nodi o sullo stato di congestione della rete. Inoltre si pensa a come dovrà essere modificata nuovamente la slice di rete, che

dovrà collegare il client alla nuovo server verso il quale verrà migrata la VM.

- Occorre identificare il percorso ottimo (tunnel) per fare la migrazione delle VMs

- Si preparano i nodi per la migrazione. Per le risorse IT è necessario inizializzare il container, ovvero il nodo che ospiterà la VM. Per le risorse di rete si deve modificare lo stato per creare la slice (ma senza ancora usarla); ad esempio:
  i. In OpenFlow modifico lo stato degli switch inserendo flow entries
  ii. In ClickOS alloco delle istanze di ClickRouter nei nodi intermedi

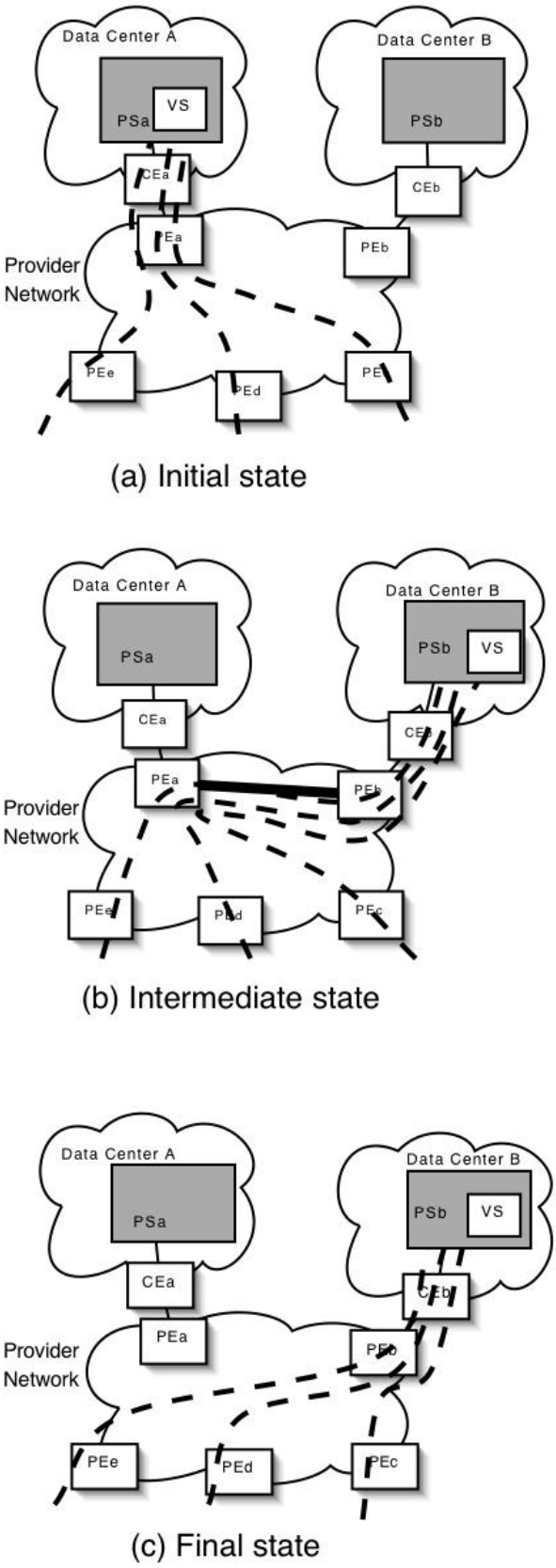
- Si inizia la migrazione della VM server

- Si finalizza la migrazione

- Si ristabilisce la connettività, ovvero si modifica la slice di rete nel modo precedentemente calcolato

La figura seguente mostra i vari step dello scenario:

Capito



(immagine tratta da "Migration across WANs: A Robust Cooperative Context Aware Approach")

3.1.4 La migrazione: analisi del modello matematico

Si possono identificare due parametri fondamentali che indicano le performance di migrazione:

- Total migration time (MT): è il tempo totale di migrazione, dal momento in cui si decide di migrare a quando la VM riprende a funzionare sul nuovo host fisico.
- Downtime (DT): periodo di tempo durante il quale la VM non è in funzione e non fornisce il suo servizio. Se la VM sta comunicando con degli altri host (ad esempio sta offrendo un servizio a un utente) un periodo lungo di downtime può causare disservizi agli utenti.

3.1.5 Discussione generale degli algoritmi di migrazione

Si può fare una classificazione generale delle tipologie di migrazione considerando le diverse caratteristiche della migrazione:

- in base a chi controlla il processo di migrazione
  - autonoma: la VM inizia, gestisce e compie autonomamente la migrazione
  - managed: la migrazione viene controllata da un attore esterno, ad esempio un terzo nodo
- in base allo sfruttamento di risorse
  - l'algoritmo di migrazione all'interno della VM sfrutta tutte le risorse disponibili nel nodo ed è vincolato solo dallo scheduler. Potrebbe creare problemi di performance alle applicazioni nella VM. Questa tecnica è chiamata anche "migrazione non adattativa" perché è indipendente dal carico di lavoro della VM.
  - adattativa: l'algoritmo di migrazione non sfrutta pienamente le risorse disponibili ma "lascia respiro" alla VM che viene migrata. La migrazione infatti è un procedimento che richiede molte risorse computazionali e di banda e non utilizzare tutte le risorse disponibili può significare metterci più tempo a completare la migrazione; questo però consente alla VM di eseguire le sue applicazioni con una certa qualità di servizio e contemporaneamente eseguire la migrazione. È detta " migrazione adattativa" perché l'algoritmo controlla periodicamente il carico di lavoro della VM e si adatta di conseguenza
- in base alla topologia di rete, riferendosi ai DataCenter
  - intra-datacenter: fra nodi dello stesso DataCenter
  - inter-datacenter: fra nodi di DataCenter diversi
- in base alla memoria disco

- si fa uso di un network-FileSystem, quindi nel DataCenter è presente un nodo NAS che contiene la memoria disco delle VM; viene migrato solo il contenuto della RAM (e i registri della CPU) da un server all'altro del DataCenter
- la memoria disco è nello stesso nodo della VM e si trasferisce la memoria disco insieme al contenuto della RAM

- in base al meccanismo di migrazione:
  - *Pure stop-and-copy*: mette in pausa la VM originale e copia l'intera memoria nel nodo di destinazione, poi la VM viene fatta riprendere nella nuova destinazione. Questa tecnica minimizza il *total migration time* ma soffre di un alto downtime.
  - *Pure on-demand copy*: la VM viene messa in pausa per breve tempo, solo per copiare le parti della VM essenziali per funzionare nella nuova destinazione. La parte restante viene copiata "on-demand", ovvero viene copiata solo quando l'applicazione richiede di accedere a parti della memoria che non sono presenti nel nodo destinazione perché non ancora copiate. Questa tecnica soffre di un alto *total migration time*, perchè con questo criterio potrebbe volerci molto tempo a migrare l'intera memoria nella nuova destinazione.
  - *Iterative pre-copy:* si basa sul concetto di copiare iterativamente delle "fotografie" della memoria della VM nella nuova destinazione, mentre questa continua a funzionare nel nodo originale. Quando lo stato della memoria del nodo originale e quello del nodo destinazione sono sufficientemente simili, viene messa in pausa VM nel nodo originale, viene migrata completamente e fatta riprendere nel nuovo nodo.

La tecnica analizzata in questa sperimentazione sarà quella di iterative pre-copy. È stato studiato l'algoritmo di migrazione della piattaforma di virtualizzazione XEN: è stato scelto algoritmo perché è open source e quindi si può visionare il funzionamento interno. Per lo studio dell'algoritmo è stato considerato il caso classico di una migrazione di VM all'interno di una rete LAN in presenza di NAS. Secondo la classificazione predente l'algoritmo è non adattativo.

3.1.6 Limitazioni delle tecniche attuali di migrazione

Le tecniche attuali di migrazione di VM suppongono che la migrazione avvenga intra-DC, ovvero fra due host dello stesso DC, in una rete di livello 2 e ad alta velocità. Non c'è quindi il problema di gestire l'indirizzo IP della VM, in quanto la rete è di livello 2 e dall'esterno la VM mantiene lo stesso indirizzo IP. La migrazione di VM "in the wild", ovvero il trasferimento da un DC a un altro passando al di fuori di una rete DC è molto difficile poiché entrano in gioco fattori come banda, latenza, perdita dei pacchetti. Alcuni recenti prototipi consentono una migrazione inter-DC, anche se per garantire il loro funzionamento richiedono alla rete dei grossi vincoli di banda e latenza: ad esempio recentemente è stata presentata la soluzione di migrazione vMotion di VMWare che richiede 622 Mbps di banda e non meno di 5ms di latenza round trip time.

Un'altra limitazione di molte delle attuali tecniche di migrazione di VM intra-DC è che richiedono che la VM sia collegata a un Network Access Storage (NAS), ovvero un nodo che contiene la memoria disco della VM, mentre sull'host dove gira la VM è presente solamente la memoria RAM. L'host che funziona da NAS inoltre si deve trovare nello stesso DC, di conseguenza la migrazione in presenza di un NAS si riduce alla copia della RAM e dei registri della CPU.

3.1.7 Dirty bitmap

Per tenere conto di quali pagine di memoria sono state modificate dalla VM durante il suo normale funzionamento l'algoritmo di migrazione utilizza una dirty bitmap, ovvero una struttura di memoria che tiene conto delle pagine che sono state modificate rispetto a un determinato istante. La dirty bitmap è uno strumento utilizzato dal sistema operativo per fare il paging della memoria. Man mano che le pagine di memoria vengono modificate per la normale attività della cpu il sistema operativo aggiorna la dirty bitmap aggiungendo le pagine modificate. Queste pagine sono dette *dirty,* ovvero inconsistenti, perché la versione di quella pagina nella memoria primaria è cambiata rispetto a quella presente in memoria secondaria e il SO prima o poi dovrà rendere la memoria secondaria nuovamente consistente. Il meccanismo di migrazione tiene sottocchio la dirty bitmap per sapere quali pagine sono state modificate e

quindi sono nuovamente da migrare. Si possono compiere due operazioni sulla dirty bitmap:

- peek: recupero il valore della dirty bitmap
- clean: recupero il valore della dirty bitmap e la resetto

3.1.8 Gli step di migrazione

La migrazione di VM con l'algoritmo di *iterative pre-copy* si può dividere nelle seguenti fasi:

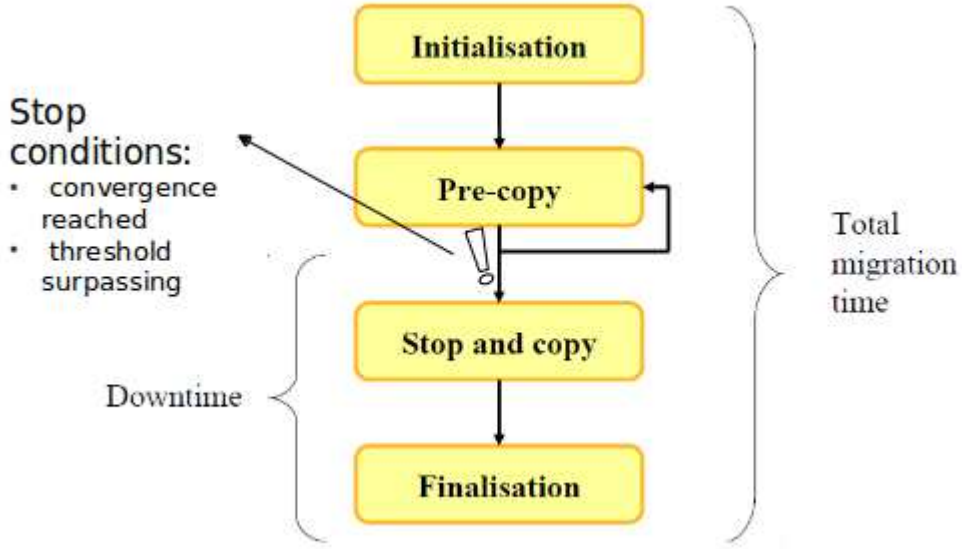
1. *initialization*: viene scelto un target container dove verrà migrata la VM
2. *reservation*: vengono riservate le risorse nel target container
3. *iterative precopy*: le pagine che sono diventate *dirty* durante l'iterazione di precopy precedente vengono trasferite nell'altro host; si ripete questo step finchè non scatta una "stop condition". Nella prima iterazione di precopy si cerca di trasferire tutte le pagine della memoria (per l'algoritmo è come se fossero tutte *dirty*).
4. l'ultimo step di iterative pre-copy, detto s*top-and-copy*, mette in pausa la VM e trasferisce l'ultima "fotografia" della memoria
5. *commitment*: l'host destinazione comunica che ha ricevuto con successo una copia consistente della VM
6. *activation*: le risorse fisiche vengono ricollegate alla VM nell'host destinazione

L'idea è quella di una convergenza iterativa: le pagine di memoria che sono state "sporcate" dall'applicazione nella VM rispetto alla copia precedente, ovvero il "delta" di memoria, vengono re-inviate. Si fa l'assunzione che prima o poi il numero di pagine modificate diventi una quantità sufficientemente piccola. Più questa quantità sarà piccola, minore sarà la quantità di memoria da trasferire nell'ultima iterazione -quando la VM è in pausa- e quindi minore sarà il downtime.

Le iterazioni di precopy sono ripetute finché non scatta una "stop condition" che cerca di intercettare la convergenza dell'algoritmo: questa condizione può essere che l'ultimo "delta" sia sufficientemente piccolo. Il

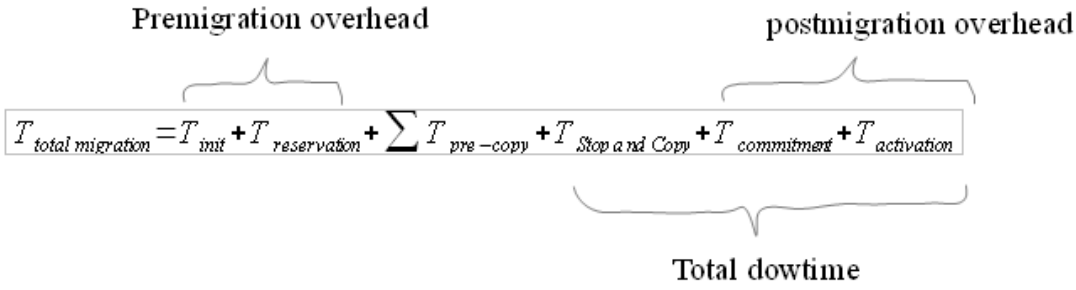
verificarsi di questa condizione dipende molto dal carico di lavoro della VM, cioè dal PDR. Dato che può capitare che questa condizione non si verifichi mai, ad esempio per un improvviso aumento del PDR che allontana dalla convergenza, è necessario fissare delle condizioni aggiuntive, ovvero delle soglie che consentano di non protrarre per un tempo indefinito la migrazione. Queste soglie si possono fissare su aspetti differenti (tempo, memoria trasferita, iterazioni fatte, etc) e sono una caratteristica peculiare dell'algoritmo di migrazione: possono esserci algoritmi con differenti stop condition.

Inoltre, come si vedrà, in realtà l'algoritmo opera un'ottimizzazione in tutte le iterazioni a parte l'ultima per cui non tutte le pagine marcate come da trasferire vengono effettivamente trasferite.



3.1.9 Indicatori di performance

Possiamo ora definire gli indicatori di performance in funzione degli step di migrazione:



$$TotalDowntime = Stop\text{-}and\text{-}copy + \underbrace{Commitment + Activation}_{Post\text{-}migrationOverhead}$$

3.1.10 Parametri che influenzano la simulazione

Possiamo suddividere in due tipologie i fattori che influenzano la migrazione:

- statici: inevitabili nel processo di migrazione e indipendenti dalla VM e dal collegamento. Sono quelle operazioni in più rispetto al trasferimento di dati vero e proprio attraverso il collegamento, ovvero gli step di pre- e post-migration. Sono l'inizializzazione del container nell'host destinazione, mirroring dei block device, l'operazione di mantenere libere le risorse, il ri-attaccare le periferiche alla VM migrata e notificare all'esterno (es. i client) se è cambiato l'indirizzo.
- dinamici: sono legati alle caratteristiche della VM (dimensione) e delle applicazioni (carico di lavoro) e sono la dimensione della VM e il carico di lavoro. Influenzano in particolare le fasi di trasferimento dei dati (pre-copy, stop & copy).

Un indicatore del carico di lavoro è il Page dirty rate (PDR), cioè la velocità con la quale le pagine di memoria vengono modificate. Un PDR alto significa che dovrà essere trasferita molta memoria nelle fasi di iterative precopy e stop-and-copy.

Un altro parametro di influenza è la banda di collegamento fisico: se ne utilizza solo una parte detta throughput (o goodput). Se si trascura il PDR dalla formula precedente il si vede che il tempo di migrazione è approssimativamente inversamente proporzionale alla banda.

3.1.11 Stop conditions

Le stop condition sono caratteristiche dell'algoritmo di migrazione e quindi sono strettamente legate alla piattaforma di migrazione utilizzata. Per l'algoritmo di XEN è ragionevole pensare che siano state ricavate dai programmatori in modo sperimentale, facendo tante prove di migrazione. Hanno un effetto significativo sulle performance e sono la causa di trend non lineari sul total migration time (MT) e total downtime (DT). In genere puntano a minimizzare l'utilizzo della banda, la quantità di memoria trasferita e al contempo il downtime.

Le stop condition di XEN sono:

1. nell'ultima iterazione di precopy sono state trasferite meno di 50 pagine
2. sono state eseguite 29 iterazioni
3. è stata trasferita una quantità di memoria maggiore di 3 volte la RAM allocata alla VM

Per fare un confronto con altri algoritmi, nella soluzione di VMWare vMotion le stop condition sono:

- la memoria *dirty* è sufficientemente piccola da poter far partire lo stop-and-copy e farlo durare al più X ms dove X è fissato
- non è stato fatto sufficiente progresso nel precedente step di pre-copy, dove questo progresso è misurato come la differenza fra la dimensione della *dirty memory* allo step attuale e allo step precedente.

3.1.12 Stima dei tempi di migrazione

Si può fare una stima grossolana dei tempi di migrazione. Una VM di dimensione VMdim che viene migrata attraverso un collegamento di banda B ci mette un tempo:

T [s] = VMdim [Byte] / B [Byte/s]

Questa stima può essere corretta se la VM viene messa in pausa e trasferita in blocco nel nodo destinazione, quindi può andare bene per stimare

una migrazione di tipo *pure stop-and-copy*. Tuttavia la formula non tiene conto dei tempi necessari a preparare i nodi, a mettere in pausa la VM, e a far ripartire la VM nel nodo destinazione (fasi di initialization, reservation, commitment e activation). Inoltre questa stima non si può applicare per una migrazione di tipo iterative pre-copy, che è il nostro caso. Mentre la memoria viene trasferita la VM continua a funzionare e la memoria si modifica, pertanto deve essere ri-trasferita. La modifica della memoria è strettamente dipendente dalle applicazioni che girano sulla VM. Per stimare questi tempi in maniera precisa è necessario riprodurre l'intera migrazione in modo da simulare il comportamento dell'algoritmo ad ogni istante di tempo. Pertanto è stato studiato un algoritmo di migrazione realmente utilizzato e ne è stato ricavato un modello da simulare in Scilab, un clone del software di simulazione Matlab.

3.1.13 Upper bound e lower bound

Conviene stabilire dei bound precisi nei quali si ha la garanzia che rientreranno i valori di total migration time (MT) e total downtime (DT).

I casi limite per il total migration time sono:

- il MT più basso si ha con la CPU in idle, e quindi non vengono modificate pagine di memoria. La fase di iterative precopy termina subito dopo la prima iterazione poiché la differenza di memoria è nulla. Il tempo richiesto è quello necessario a inviare una volta tutta la memoria RAM più i pre- e post-migration overhead.
- il MT più alto si ha se la memoria viene modificata a una velocità che si avvicina alla capacità di banda. Il caso peggiore è se scatta la stop condition 3: in questo caso significa che è stata trasferita una quantità maggiore di 3*VM di memoria. A questo punto, per come è fatto l'algoritmo, viene eseguito un ulteriore passo di iterative copy e poi lo stop-and-copy, in entrambe trasferendo la quantità massima possibile di memoria, quindi viene trasferita 2*VM.

$$Overheads + \frac{VMSize}{LinkSpeed} \leq TotalMigrationTime \leq Overheads + \frac{5 * VMSize - 1 * page}{LinkSpeed}$$

Casi limite per il downtime:

- il downtime è minimo se il PDR è basso, ovvero CPU in idle: nello stop-and-copy non viene copiata memoria. Il downtime è quindi solo il post-migration overhead
- il downtime è massimo se il PDR alto, cioè la memoria è modificata molto velocemente. Nello stop-and-copy viene copiata l'intera memoria.

$$Post\text{-}migrationOverhead \leq TotalDowntime \leq Post\text{-}migrationOverhead + \frac{VMSize}{LinkSpeed}$$

I bound trovati formano un intervallo di variabilità troppo elevato e non permettono una stima sufficientemente precisa dei tempi di migrazione. Bisogna quindi ricorrere ad algoritmi che simulano la migrazione per essere in grado di predire meglio le performance.

3.1.14 Algoritmo di migrazione

L'algoritmo studiato è quello della piattaforma di virtualizzazione XEN. Il meccanismo di migrazione è scritto in codice C: si vuole simulare il comportamento dell'algoritmo e in particolare l'aspetto della tempistica. Quelle parti dell'algoritmo che compiono delle azioni e che impiegano del tempo a essere eseguite vengono sostituite nella nostra simulazione da delle variabili tempo che vengono incrementate di una certa quantità.

In particolare:

- dove nel codice XEN si esegue il trasferimento di un batch di dati, nella nostra simulazione una variabile tempo viene incrementata in base alla dimensione del batch e alla banda del collegamento.
- dove nel codice XEN viene controllata la dirty bitmap, nella nostra simulazione questa operazione viene simulata tramite i due metodi sim_peek e sim_clean. Questi metodi simulano il "dirtying" della memoria e poi eseguono il recupero del valore della dirty bitmap (sim_peek senza modificare la dirty bitmap, sim_clean resettando la dirty bitmap)

Alla fine la simulazione restituisce il tempo totale di migrazione (total_migration_time) e il tempo totale di downtime (total_downtime).

Di seguito l'algoritmo di simulazione è mostrato in pseudocodice:

```
      sent_this_iter=0
      skipped_this_iter=0
      while(n<p2m_size){
            skipped_in_batch=0
            if(!last_iter)
                  to_skip= sim_peek() //peek shadow map
            for(batch=0; batch<1024 && N<p2m_size;N++){
                  n_in_tsend = n â^ to_send
                  n_in_tskip = n â^ to_skip
                  if(!last_iter && n_in_tsend && n_in_tskip){
                        skipped_this_iter++
                        skipped_in_batch++
                  }
                  ok_to_send = (n_in_tsend && !n_in_tskip ) ||
                              (n_in_tsend && last_iter)
                  if(!ok_to_send)
                        continue
                  batch ++
            }
            bytes_to_send = batch * page_size
            batch_send_time = bytes_to_send / link_speed
            migration_time += batch_send_time
            if(last_iter)
                  downtime += batch_send_time
            sent_this_iter += batch
      }
      total_sent += sent_this_iter
      if(last_iter)
            break
      if(iter>MAX || total_sent > 3* VMdim || sent_this_iter+skipped_this_iter<50)
            last_iter = true
      to_send = sim_clean
}
total_migration_time = pre_time + migration_time + post_time
total_downtime = downtime + post_time
```

Si veda l'appendice per uno schema temporale dell'algoritmo.

3.1.15 Ottimizzazioni

L'algoritmo sopra descritto utilizza delle particolari ottimizzazioni: in un passo di iterative pre-copy le pagine che sono diventate nuovamente *dirtied* nell'attuale passo di iterazione vengono saltate durante lo "scanning" delle pagine (non vengono incluse nel batch da trasferire) e quindi non vengono trasmesse. Queste pagine verranno riconsiderate all'iterazione successiva. L'algoritmo sfrutta quindi il principio di località temporale e ipotizza che se la pagina è stata modificata più volte in un così breve periodo è probabile che venga modificata ancora e quindi non vale la pena trasmetterla adesso se poi si dovrà trasmetterla di nuovo.

Un'altra caratteristica è che l'algoritmo invia i dati in batch di 1024 pagine per volta, quindi in pacchetti di circa 1024*4KB= 4MB (la dimensione della pagina si suppone di 4KB).

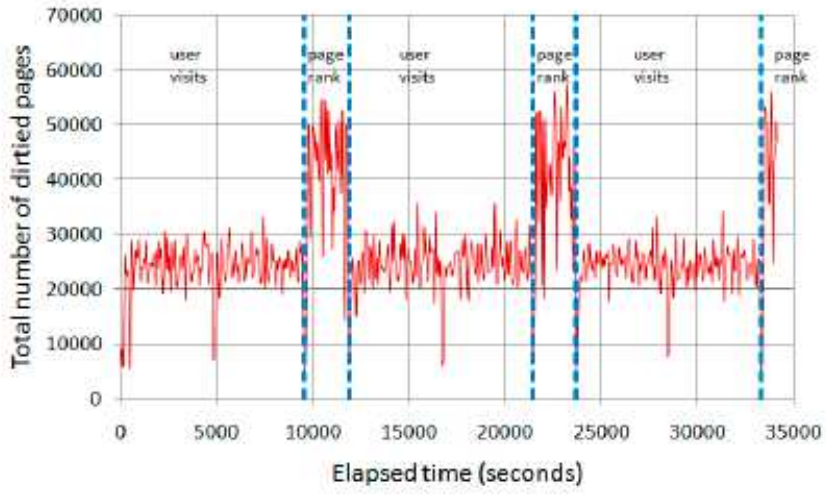
3.1.16 Simulare il carico di lavoro

Simulare il carico di lavoro è uno dei problemi più importanti se si vuole predirre le performance di migrazione. Come si è detto la parte del carico di lavoro che influenza la migrazione non è l'attività della CPU in sé, ma la velocità con cui vengono modificate le pagine di memoria, e questa velocità è detta page dirty rate (PDR) e si misura in pagine/secondo.

Per la simulazione si possono usare due modelli:

- **AVG model:** si assume un page dirty rate costante. È utile se il PDR di una VM è più o meno stabile.
- **HIST model**: adatto a simulare situazioni più realistiche dove il carico di lavoro è una funzione del tempo e questa funzione si può approssimare deterministicamente. L'approssimazione può essere ottenuta facendo il log nel tempo delle pagine *dirtied* dell'applicazione reale e si ottiene così uno storico; si eseguono tanti "run" dell'applicazione e si fa la media dei log ottenuti e si ottiene così la funzione nel tempo del PDR.

La figura seguente mostra un'applicazione di esempio, MapReduce, per cui è stato ottenuta la funzione del tempo del carico di lavoro della VM:



Nella nostra simulazione è stato usato l'AVG model.

3.1.17 Analisi delle relazioni fra i parametri di influenza e le performance

Le stop condition causano **trend non lineari** nella relazione fra i parametri di influenza (banda, page dirty rate,latenza) e le performance ottenute. Nonostante questo si possono individuare localmente degli intervalli in cui si manifestano in media relazioni lineari.

Se il PDR è molto minore della banda l'algoritmo ha buone performance (basso DT, basso MT) e a scattare è la stop condition 1 . Inoltre se il PDR è molto basso il DT è circa costante ed è uguale al lower bond, perché la banda è sufficiente a trasferire la memoria nelle fasi di iterative precopy e pochissima memoria nello stop-and-copy.

Al crescere del PDR scattano invece le stop condition 2 o 3; l'algoritmo entra nella fase di stop and copy con più memoria da trasferire, portando a un DT maggiore. Il DT quindi cresce circa proporzionalmente all'aumentare del PDR fino ad arrivare all'upper bound (che comporta nello stop-and-copy l'invio dell'intera memoria). Anche il MT cresce, fino ad arrivare all'upper bound fissata dalla stop condition 3: infatti l'iterative copy viene interrotta dopo che è stata trasferita una quantità massima di memoria.

Per PDR molto alte a scattare è la stop condition 2. Questo avviene perché l'algoritmo è ottimizzato per saltare le pagine di memoria che cambiano in fretta (e precisamente una pagina viene saltata se è *dirtied* nello stesso passo di iterazione e l'iterazione corrente non è l'ultima iterazione). Saltando molte pagine i passi di iterazione si completano più in fretta, fino ad arrivare al caso estremo in cui in ciascun passo di iterazione non vengono trasferite pagine. Questo porta a far scattare la stop condition 2 prima delle altre, e quindi il MT scende fino ad arrivare al suo caso limite, il lower bound in cui nelle 29 iterazioni sono state trasferite poche (o nessuna) pagine e si deve inviare l'intera memoria nell'ultima iterazione. Infine anche nella fase di stop-and-copy viene inviata nuovamente molta o quasi tutta la memoria in quanto, avendo PDR alto, molte pagine sono nuovamente *dirtied* anche nell'ultima iterazione di precopy.

3.1.18 Risultati di simulazione

Di seguito sono mostrati i risultati di simulazione della migrazione di una VM di 10MB. A causa della complessità dell'algoritmo di simulazione dell'algoritmo è risultato proibitivo simulare la migrazione di una VM di dimensioni a quelle reali (dell'ordine del GB), ma ciononostante la simulazione non è priva di significato: una VM di piccole dimensioni può essere assimilata a una funzione di rete di cui si è parlato nel capitolo 2, che viene migrata da un Data Center a un altro per fornire un qualche servizio di rete all'utente.

Il workload, ovvero il PDR, è fisso durante l'intera migrazione. Sono state eseguite simulazioni con PDR di 5000, 10000 e 20000 pagine/secondo e con uno throuput di collegamento di 100Mbps,1Gbps, 10 Gbps. I grafici mostrano Il total migration time ottenuto e, per ogni run, quale stop condition è scattata.

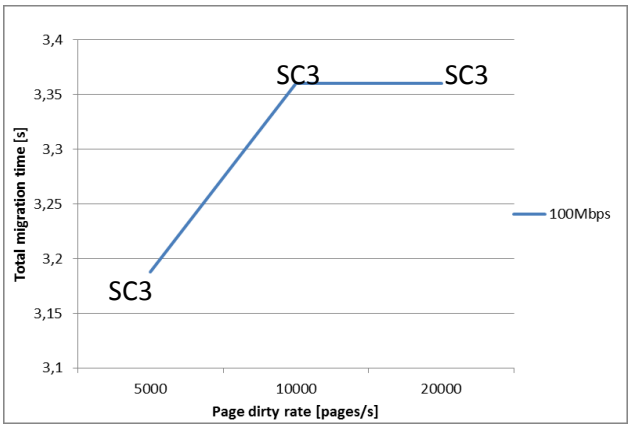


Fig. 3. VM migration: total migration time (100 Mbps)

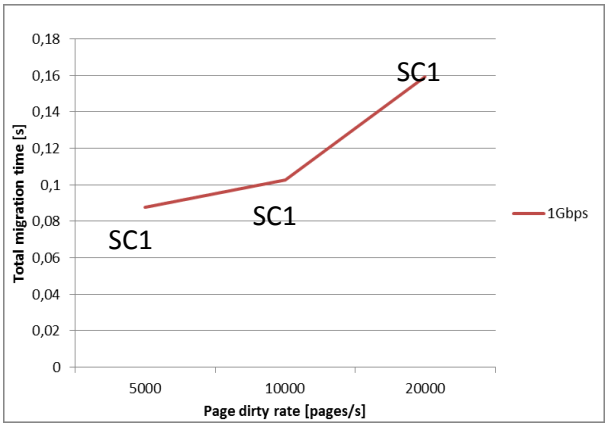


Fig. 4. VM migration: total migration time (1 Gbps)

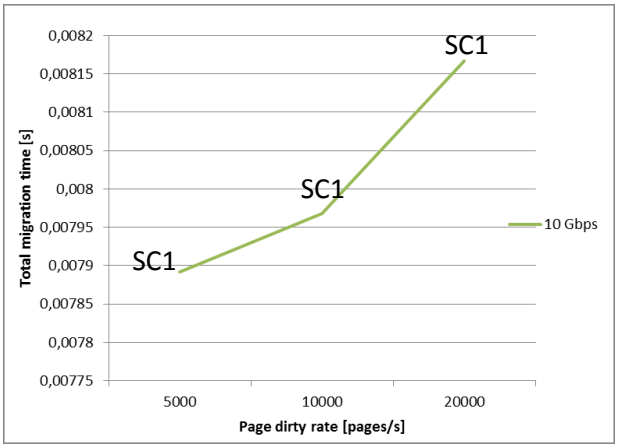


Fig. 5. VM migration: total migration time (10 Gbps)

Le simulazioni mostrano alti page dirty rate corrispondono a un tempo maggiore di migrazione: per quasi tutti i run la migrazione riesce a raggiungere la convergenza e scatta la stop condition 1. Per bassa banda di collegamento e un alto page dirty rate le stop condition forzano l'iterative copy a fermarsi: nella simulazione con 100 Mpbs scatta la stop condition numero 3, ovvero sono stati trasferiti troppi byte sul collegamento e il total migration time raggiunge un upper bound. Per B=100 Mbps e PDR=5000 pages/s non è raggiunto l'upper bound perché il PDR non è così alto da far trasferire la quantità massima di memoria nella fase di stop-and-copy, anche se è scattata la stop condition 3. Negli altri due casi (B=100 Mbps e PDR=10000 pages/s e 20000 pages/s) invece il PDR è così alto, e la banda così bassa, che nella fase di stop-and-copy viene trasferita l'intera memoria della VM (10 MB) e si ottiene l'upper bound.

Calcolando l'upper bound teorico si ottiene:

$MT_{upperbound}$= (5*VMsize-1*pagesize)/B = (5* 10M – 4K)/ (100M/8) = 4 sec

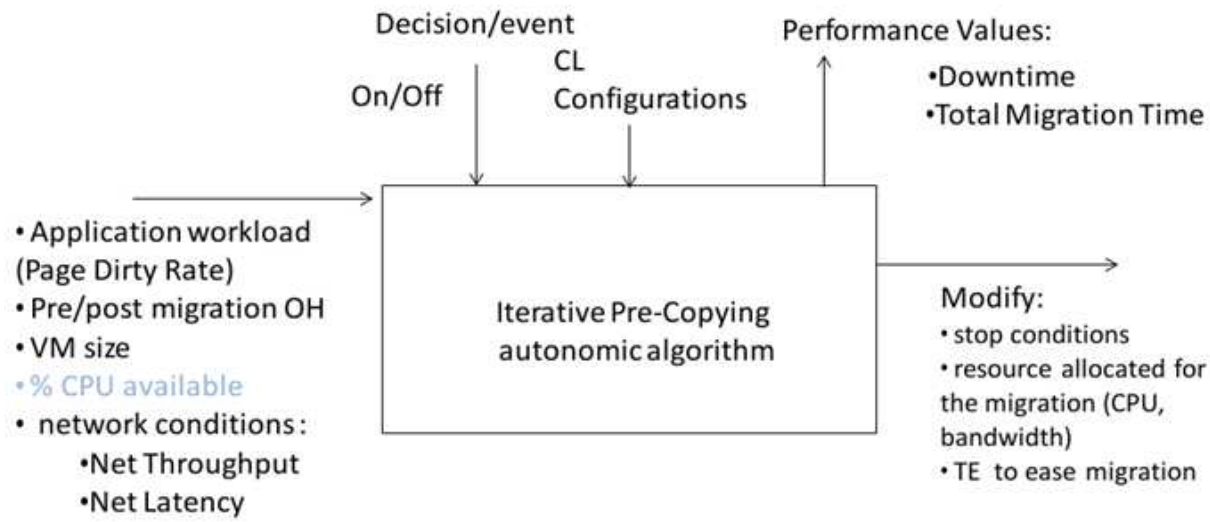
Risulta che è maggiore dell'upper bound effettivo: questo si può spiegare con il fatto che la formula restituisce un valore che è per l'appunto teorico e l'algoritmo di migrazione, per come è progettato e anche in virtù delle sue ottimizzazioni, non lo raggiungerà mai.

Infine bisogna notare come la simulazione non tenga conto dei pre/post overhead, impostati a zero nelle simulazioni fatte. Come si è detto i tempi di pre/post overhead sono statici e non dipendono dalla dimensione dal carico della VM. In alcuni casi, come per bassi PDR e alta banda, questi valori sono predominanti nel bilancio complessivo delle performance di migrazione. Per ottenere un valore significativo di questi è necessario eseguire delle prove sperimentali e poi aggiungere questi dati nella simulazione.

3.1.19 Verso un algoritmo autonomico

Nei capitoli precedenti è stato studiato un algoritmo di migrazione di VM e si è visto come molte caratteristiche fossero strettamente legate alla specifica implementazione. Inoltre si è visto come le performance di migrazione siano dipendenti da diversi fattori, come la dimensione della VM, le caratteristiche delle applicazioni che girano sulla VM e la rete. In futuro è ragionevole pensare che gli algoritmi di migrazione saranno in grado di adattarsi alle condizioni ambientali: avremo quindi degli algoritmi di migrazione di funzioni di rete all'edge con caratteristiche autonomiche.

Per una migrazione inter-DataCenter la VM deve essere migrata passando attraverso una rete geografica d'accesso: la migrazione si dovrà adattare a differenti condizioni di banda, latenza, applicazioni. Per operare questo adattamento l'algoritmo può modificare alcuni suoi parametri interni, come le stop condition, le risorse allocate per fare la migrazione, come CPU o la banda disponibili, oppure potrebbe modificare anche parte dell'ambiente: ad esempio potrebbe operare del Traffic Engineering sulla rete che deve consentire la migrazione per migliorare la velocità di trasferimento o diminuire la latenza. Di seguito è mostrato uno schema black-box di un algoritmo di



migrazione autonomico che include le caratteristiche sopra descritte. L'adattamento autonomico potrebbe avvenire durante la stessa migrazione e quindi per migliorare istantaneamente le performance di migrazione oppure potrebbe avvenire fra migrazioni successive, dove ogni migrazione cerca di avvenire in maniera più performante della precedente.

3.2 Scenari di reti autonomiche/cognitive

In questo seconda sperimentazione la migrazione di VM è stata affrontata da un punto di vista più di gestione delle risorse. Partendo dalla vision che è stata descritta nel capitolo 1, ovvero si è voluto spostare il più possibile l'intelligenza all'edge e introdurre aspetti cognitivi nella rete, sono stati elaborati degli scenari che illustrassero meglio questi concetti. Su questi scenari sono poi state fatte delle simulazioni.

Come si è detto nei capitoli precedenti le recenti teorie Software Defined Network e Network Function Virtualization saranno abilitatrici di paradigmi di rete innovativi: SDN per quel che riguarda il disaccoppiamento del controllo dall'H/W di rete e NFV per quel che riguarda la possibilità di avere delle "istanze" S/W di funzionalità di rete che possono essere allocate nei nodi della rete o possono essere spostate a piacimento. Rimanendo su un piano astratto possiamo limitarci a identificare queste funzioni di rete come delle entità atomiche elementari che possono essere composte e che quindi possono persino cooperare fra di loro.

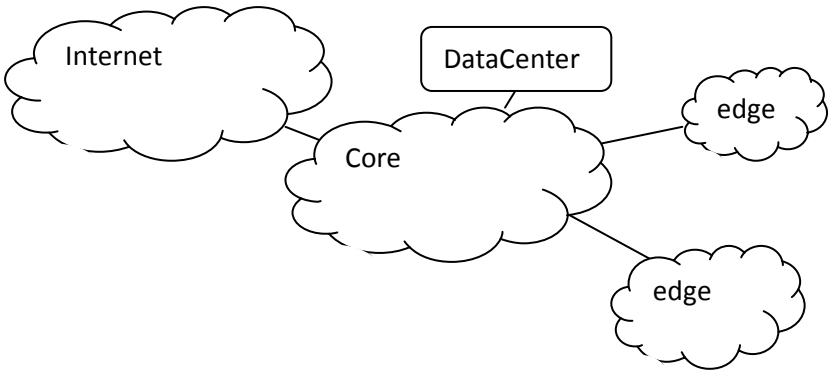
3.2.1 I CENode e le CENet

È utile definire gli elementi che caratterizzano i nostri scenari:

- definiamo il Cognitive Edge Node (CENode) come un nodo della rete all'edge. In particolare è quel nodo dell'operatore che fornisce il servizio agli utenti (in un'interpretazione più estesa è un nodo qualunque dell'edge network).

- definiamo poi il Cognitive Edge Network (CENet) un'intera rete all'edge dotata di aspetti cognitivi, comprensiva dei punti di accesso, degli utenti e delle risorse di storage e processing. Ricordiamo che l'intera rete dell'operatore è composta da tante edge network collegate fra di loro tramite una core network, sempre dell'operatore. La core network a sua volta è collegata alla rete Internet.

3.2.2 Fasi di evoluzione della rete

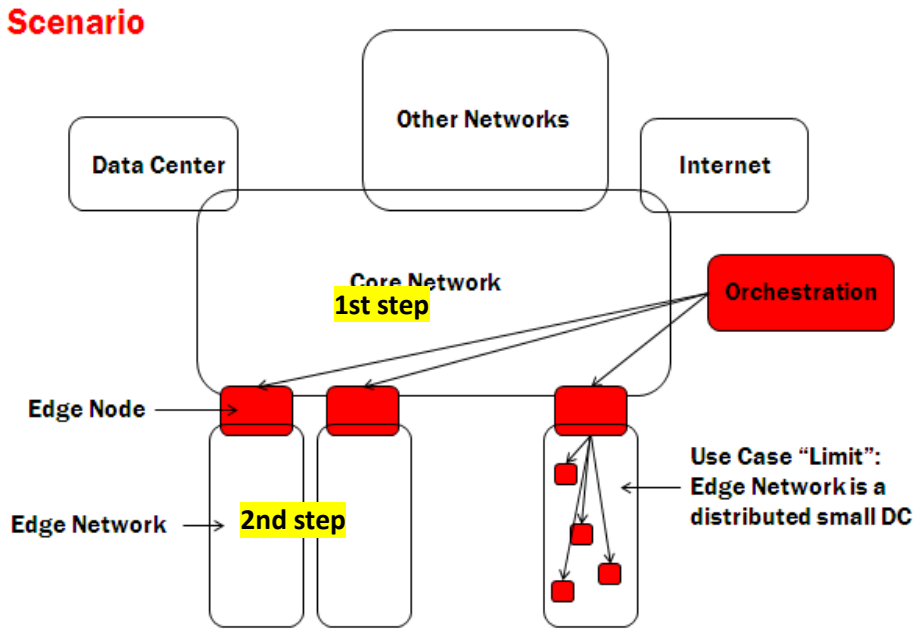


Si possono individuare due fasi di evoluzione della rete rispetto a quella attuale:

1. si verificherà uno spostamento dell'intelligenza all'edge, portando le funzionalità di rete nei nodi della rete di accesso. Questi nodi edge disporrano di risorse di storage e di processing e forniranno dei servizi ai nodi degli utenti. I nodi all'edge saranno capaci di coordinarsi fra di loro tramite interazioni locali, ma sarà presente anche un orchestratore in grado di dirigere il comportamento complessivo delle edge network fornendo delle direttive.

2. Un ulteriore passo è distribuire le funzionalità sull'ambiente, ovvero non ci sarà più una distinzione netta fra i nodi che forniscono il servizio e quelli che ne usufruiscono (questo sarà possibile con un generale aumento delle capacità di strage e processing dei dispositivi). Si introduce quindi un livello ulteriore di orchestrazione, dove tutti i nodi utente di una rete di accesso vengono orchestrati dagli edge node della rispettiva CENet, similmente a come si orchestrano oggi i server all'interno di un DataCenter.

   La figura seguente vuole descrivere l'aspetto di orchestrazione delle due fasi di evoluzione della rete e in particolare l'aspetto di ricorsività dell'orchestrazione:



E' utile trovare esempi concreti per descrivere questi due scenari/fasi di evoluzione della rete: di seguito si descrivono degli ipotetici "kiosk nodes" come esempio illustrativo del primo scenario e degli "ultrasmart devices" per il secondo scenario.

Possiamo identificare dei "kiosk nodes" come dei nodi (o gruppi di nodi fisicamente vicini) dotati di un discreto quantitativo di risorse (storage/processing) e che sono un punto di accesso per i nodi utenti. Questi nodi quindi forniscono il servizio di accesso alla rete, ma non solo quello, e saranno localizzati in punti strategici dove possano usurfruirne una grande

quantità di utenti. Come esempio concreto si consideri un aeroporto dove è presente un access point in grado di fornire una connessione a internet e anche servizi dedicati, come informazioni sui terminal o su scioperi del personale.

Nel secondo scenario che abbiamo immaginato invece ogni nodo utente possiederà delle risorse (di processing, storage, rete…) tali da poter fornire i servizi, e quindi rivestirà un ruolo attivo nell'ecosistema di rete. Si consideri come caso reale la possibilità di fare una telefonata sfruttando altre utenze come hop intermedi: si ha così la possibilità di comunicare anche in zone dove magari la copertura di rete non è sufficiente, oppure in questo modo si potrà alleviare carico di lavoro ai punti di accesso dell'operatore in zone molto affollate. Può darsi che i cellulari del futuro (che identifichiamo come "ultrasmart devices") possiedano una potenza tale da non richiedere la presenza di gran parte dei CENode dell'operatore, oppure che ci sia la necessità di tanti piccoli "hub" intermedi, magari persino di proprietà degli utenti. In ogni caso non ci sarà più la distinzione netta fra i nodi di servizio dell'operatore e i nodi degli utenti che usufruiscono del servizio.

Ci sono dei punti di contatto fra la rete d'accesso dell'operatore e i Data Center per quel che riguarda l'aspetto autonomico, che è stato descritto al capitolo 2: si può fare una similitudine fra il componente di "edge orchestration" e il "Resource arbiter". L'application manager invece è simile all'edge node. Un application environment infine a livello funzionale è molto simile una delle edge networks presenti nella rete dell'operatore.

3.2.3 La blackboard

Concentriamoci sul primo scenario che è stato ipotizzato, ovvero quello di una rete dell'operatore composta da tante CENet, ciascuna gestita da un CENode e con un orchestratore a operare una ottimizzazione globale delle risorse: possiamo ora andare a esplorare quali potranno essere i meccanismi di orchestrazione. Prendiamo una singola CENet: si vogliono identificare le necessità degli utenti, i servizi, i meccanismi di orchestrazione.

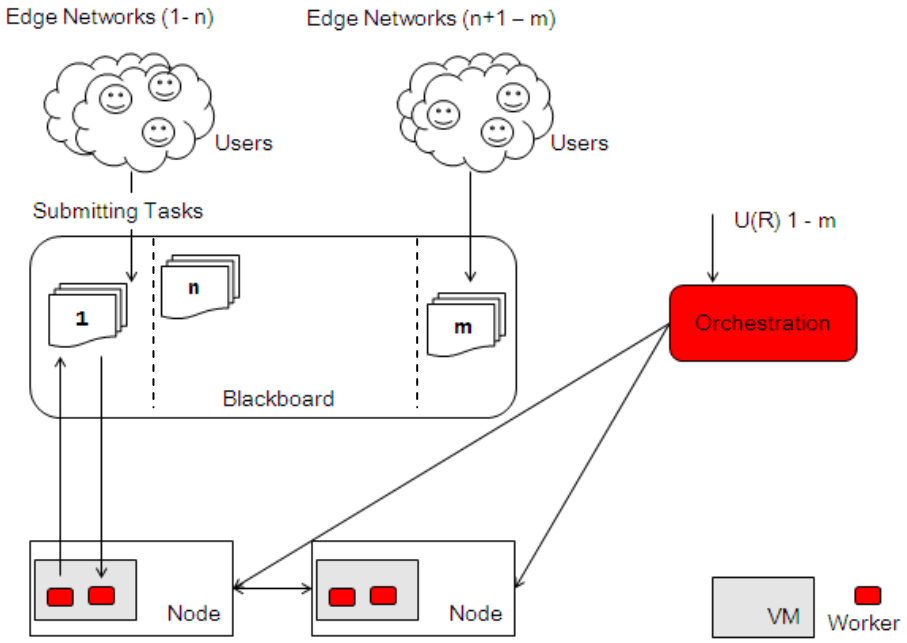
Lo scenario qui illustrato identifica gli utenti, i CENode, le VM all'interno dei CENode e l'orchestratore. Possiamo vedere CENode come dei

“container , che contengono delle unità computazionali che possiamo identificare come VM. A differenza della rete attuale in cui i nodi della rete si vedono assegnare i compiti dall’esterno, i CENode sono in grado di assegnarsi autonomamente dei compiti e poi eseguirli. Questo comportamento avviene perché nei CEN sono state installate delle regole locali tali per cui quando un CENode è sufficientemente “libero”, ovvero dispone di risorse inutilizzate, va in cerca di compiti utili da svolgere. Una *blackboard*, che è la nostra astrazione di spazio condiviso, contiene i compiti da svolgere. Gli utenti quando vogliono richiedere un servizio inseriscono la richiesta nella *blackboard* e i CENode recuperano il compito da svolgere dalla *blackboard*. La blackboard non è visibile all’utente, è uno strumento del CENode che gli consente di immagazzinare e recuperare le richieste. È stata pensata una blackboard per ogni edge network e quindi il nostro non è uno spazio totalmente condiviso: un’altra scelta più radicale sarebbe potuta essere quella di pensare a una blackboard condivisa fra tutti i CENode, così che un CENode di una certa CENet potrebbe decidere di prendersi carico di una richiesta di servizio di un utente di un’altra CENet.

Le richieste di funzionalità di rete quindi vengono eseguite dalle VM contenute nel CEN. L’orchestratore controlla qual è il carico di lavoro di ciascun CEN e può decidere di spostare le VM in un altro CEN: questi interventi devono avvenire se portano dei benifici a livello di qualità del servizio per l’utente o di consumo energetico per l’operatore.

Possiamo identificare due comportamenti autonomici:

- Le regole locali installate nei CENode che fanno avere loro un comportamento di tipo *pull*, ovvero recuperano autonomamente le richieste. Nella teoria dei sistemi autonomici si parla di comportamento automatico. I singoli nodi devono ottimizzare utility function locali.
- Le regole globali installate nell’edge orchestrator, comportamento di orchestrazione. L’orchestratore deve ottimizzare una utility function globale.



I problemi che sorgono sono molteplici: come gestire la migrazione delle risorse magari anche fra CENet differenti, quali sono le utility function locali e globali, etc. Questi problemi verranno approfonditi nei capitoli successivi.

Nell'ottica di studiare in generale l'integrazione delle teorie autonomiche nelle reti dell'operatore in scenari a lungo termine non si vuole identificare con precisione quali sono i task, le utility function, etc.

Non si vuole infatti entrare troppo nei dettagli specifici delle tecnologie: questo approccio non si adatta bene nell'elaborazione di scenari "long-term", perché potrebbe fornire una versione riduttiva e limitante di uno scenario futuro, precludendo delle possibilità a priori.

3.2.4 Sistema complesso adattativo

Lo scenario illustrato ha delle affinità con i "sistemi complessi adattativi"(SCA). Un SCA è un sistema che comprende un gran numero di elementi che interagiscono fra di loro in modo non lineare: la definizione di interazione non lineare dipende molto dal caso in esame, ma in generale si può dire che l'interazione risultante non rispetta il principio di sovrapposizione

degli effetti e nessuno dei componenti del sistema può essere studiato separatamente dagli altri. Complessivamente un SCA mostra delle dinamiche caotiche: si possono verificare delle repentine transizioni di fase, ovvero il sistema può passare velocemente da una configurazione all'altra.

In un SCA reale di grandi dimensioni si presentano delle *dinamiche auto organizzanti*, ovvero dei comportamenti che tendono a portare il sistema in una determinata configurazione: questi comportamenti e appaiono senza un orchestratore centrale, ma dipendono solo dalle interazioni locali fra i singoli elementi. Si può fare un paragone con i SCA presenti in natura: le lucciole sono in grado di uniformare fra di loro la frequenza di oscillazione fra lo stato di luce e di buio. Ciascuna lucciola si adatta semplicemente alla frequenza (e la fase) delle lucciole vicine e, dopo un primo periodo di caoticità, tutte le lucciole raggiungono uno stato uniforme. Gli stormi di uccelli sono in grado di mantenere una certa formazione senza la presenza di un controllo centrale, ma solo tramite regole locali. Ogni uccello infatti cerca di mantenere costante la distanza con gli uccelli vicini e questo è sufficiente per mantenere la formazione globale.

Inoltre la dinamica del sistema è molto dipendente dalle condizioni iniziali. A seconda dello stato iniziale il sistema può raggiungere differenti stadi di equilibrio finali.

Nel nostro scenario il numero di elementi del sistema è molto piccolo (considero 2-3 edge network) quindi si può considerare come un esempio semplice di SCA. Nel nostro scenario per ottenere l'auto-organizzazione aggiungiamo un orchestratore centrale.

3.2.5 Descrizione scenario per la simulazione

Riprendiamo alcuni concetti descritti precedentemente per definire uno scenario che possiamo simulare.

Supponiamo di avere tanti edge node, ogni edge node è attestato a una edge network e si occupa di servire le richieste di servizi degli utenti di quella edge network. L'edge node nelle reti attuali viene chiamato BRAS ed è un apparato che ha unicamente il ruolo di "imbuto" per le connessioni di rete e non ha capacità di processing e storage e non dispone nemmeno di funzionalità

avanzate di rete. Possiamo però supporre che questo cambi e nelle reti del futuro il BRAS venga sostituito da un dispositivo con grosse capacità di processing e storage. Possiamo vedere anche uno scenario intermedio in cui questo edge node è un nodo assistito da un DataCenter in grado di fornirgli la potenza computazionale richiesta per svolgere i suoi compiti. In ogni caso su questo particolare aspetto facciamo un'astrazione e parlaremo semplicemente di edge node.

Il comportamento orchestrativo consiste nel redistribuire le risorse: queste vengono spostate da un edge node in surplus di risorse verso un nodo con carenza di risorse. All'atto pratico in uno scenario futuro di rete reale non è necessario che queste risorse vengano realmente spostate: un nodo potrebbe semplicemente "delegare" un certo compito a un altro nodo perché si trova momentaneamente in difficoltà. In ogni caso in questa sede non ce occupiamo in quanto ci interessa analizzare solo il comportamento autonomico finalizzato alla ottimizzazione delle risorse.

In questo scenario non si considera la latenza, in quanto stiamo astraendo da questo dettaglio fisico.

Si astrae inoltre anche dalla specificità dei singoli servizi. Il nostro scenario non considera quindi tutti quei casi in cui un nodo non dispone di una determinata funzionalità e la richiede a un altro nodo.

Ogni edge network ha diverse classi di utenti: possiamo dividerla in utenti che richiedono funzionalità di tipo "*gold*", "*silver*" o "*bronze*", ovvero classifichiamo in una scala gerarchica le diverse priorità/qualità di servizio. L'obiettivo di ogni CENode è quello di soddisfare molto bene le richieste servizio di tipo gold, abbastanza bene quelle di tipo silver e in modalità "best effort" quelle di tipo bronze.

Ogni CENode ha una utility function che, ad ogni istante di tempo, misura qual è lo "stato di salute" suo e della CENet, ovvero quanto bene il CENode riesce a svolgere il suo compito di soddisfare le richieste degli utenti.

Se il CENode si ritrova ad avere ancora molte risorse da soddisfare e ha poche risorse a disposizione si trova in difficoltà e la sua utility function sarà bassa. Se invece il CENode riesce a stare al passo con le richieste di servizi degli utenti allora avrà un'alta utility function.

Si può quindi immaginare una tabella dove a ogni istante di tempo in un CENode vengono soddisfatte un certo numero di richieste G, S o B, mentre altre rimangono "inevase" nella blackboard. Il CENode assolve alle richieste in ordine di priorità.

Tabella per il CENode i:

	Gold "inevase"	Silver "inevase"	Bronze "inevase"	Utility function
t0	..	..	..	..
t1	..	..	..	..
t2	..	..	..	..
..		..	..	..
..	..	..	..	..

3.2.6 Criteri di modellazione della simulazione

Nel realizzare la simulazione è stato necessario fissare alcuni aspetti rimasti non definiti a livello di descrizione dello scenario. Di seguito vengono descritti questi aspetti.

Un aspetto da modellare è quante iterazioni ci vogliono per eseguire completamente una richiesta di servizio, dove un'iterazione corrisponde a un intervallo di tempo:

- ogni task viene completato dal CENode in una sola iterazione
- i task hanno durata differente e alcuni richiedono più iterazioni.

Per la nostra simulazione è stato scelta la prima opzione, in quanto è più semplice da simulare e per il momento è sufficiente per lo scopo della simulazione. La seconda opzione è stata lasciata come lavoro futuro, ed è realizzabile modificando la simulazione attuale.

Modellazione del comportamento dell'edge node a ogni iterazione: a ogni iterazione l'edge node recupera un certo numero di richieste dalla blackboard e comincia a evaderle, a seconda di quante VM libere dispone. Avendo "in pancia" più VM può eseguire in contemporanea più task. Il CENode deve assolvere alle richieste di servizio secondo una certa policy. Alla fine dell'iterazione l'edge node invia all'orchestrator il valore attuale della sua utility function.

Modellazione del comportamento dell'orchestrator: in base alle utility function ricevute dai CENode calcula la sua utility function globale e/o opera un algoritmo di ottimizzazione e redistribuzione delle risorse. Per la nostra simulazione è stato scelto un algoritmo di tipo "first-fit", ovvero l'orchestratore opera una redistribuzione delle risorse andando a prendere la risorse dal primo nodo "adatto". Un nodo "adatto" da cui prendere le risorse è un nodo con una alta utility function.

Come modellare le utility function locali e globali: l'utility function di un CENode deve indicare quanto bene il CENode riesce a soddisfare le richieste di servizio della propria CENet. Dipende quindi dal numero di richieste di servizio ancora inevase, ovvero pendenti nella blackboard. Nell'utility function si potrebbero considerare, se presenti, anche quelle che sono eseguite per metà all'interno delle delle VM del CENode, ma abbiamo fatto l'ipotesi che una richiesta venga completata in una iterazione.

Di seguito è mostrata l'utility function locale utilizzata nella simulazione:

utility = 1/(a*k_gold+b*k_silver+c*k_bronze+1)

L'utility function è inversamente proporzionale alla dimensione della blackboard, calcolata come somma pesata delle richieste ancora "inevase" suddivise in classi di servizio. Le variabili a,b,c sono i pesi della somma pesata e il loro scopo è quello di far sì che nel bilancio finale lo stato di salute del CENode (il valore dell'utility function) sia influenzato in maniera differente dalle differenti classi di servizio. Nel caso ottimo in cui la blackboard sia vuota, cioè il nodo è riuscito a completare in un certo istante tutte le richieste di servizio, k_gold+ k_silver+ k_bronze vale 0 e quindi res vale 1.

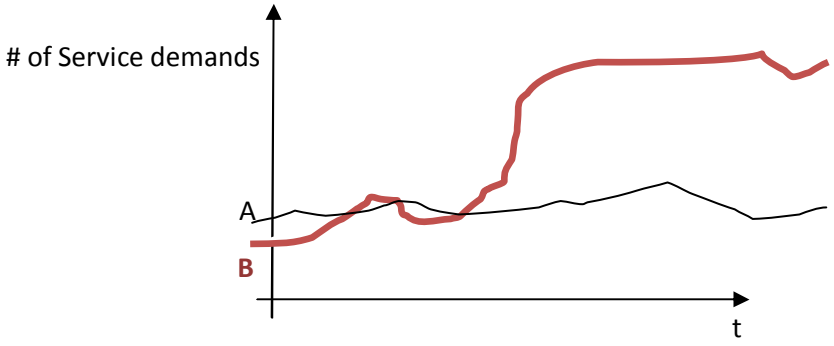
Per l'orchestratore invece si è scelto di non calcolare alcuna utility function globale: l'orchestrazione avviene unicamente mediante la redistribuzione di tipo first-fit, come descritto sopra.

Modellazione della generazione delle richieste: gli utenti di una CENet i a ogni iterazione aggiungono un certo numero di richieste alla blackboard i-ma.

Questo numero è casuale e potrebbe essere modellato secondo una certa distribuzione di probabilità. Nella nostra simulazione per semplicità si è scelto di generare in modo random le nuove richieste, con una distribuzione di probabilità uniforme in un certo intervallo. Per simulare un picco di richieste si è scelto di aggiungere un *delta* prefissato al valore casuale generato.

Scelta del numero di CENet da simulare: vogliamo simulare uno scenario in cui due CENode, A e B, servono le rispettive CENet, N_a e N_b.

Modellazione di una situazione specifica che richieda un'orchestrazione: A e B possiedono un quantitativo di risorse più che sufficiente da servire le richieste dei loro utenti, di conseguenza le utility function di A e B hanno entrambe un valore alto. A un certo punto in N_b aumenta il numero di richieste di servizio e B si trova in deficit di risorse. L'orchestratore nota che l'utility function di B diminuisce, ovvero sta peggio, e opera una redistribuzione delle risorse, trasferendo una certa quantità di risorse da A a B. Questo quantitativo deve essere tale riportare a un valore accettabile l'utility function di B, ma allo stesso tempo non abbassare troppo l'utility function di A.



Modellazione della modalità di evasione delle richieste: abbiamo già accennato che le richieste di servizio sono suddivise in "classi di servizio" (Gold, Silver, Bronze). Inizialmente si era pensato a una policy del tipo:

- il CENode evade per prime le richieste di servizio Gold e Silver, in percentuale 60% e 40%

- se il CENode ha ancora delle risorse disponibili (VM a cui non è stato ancora assegnato un compito) evade il più possibile delle richieste

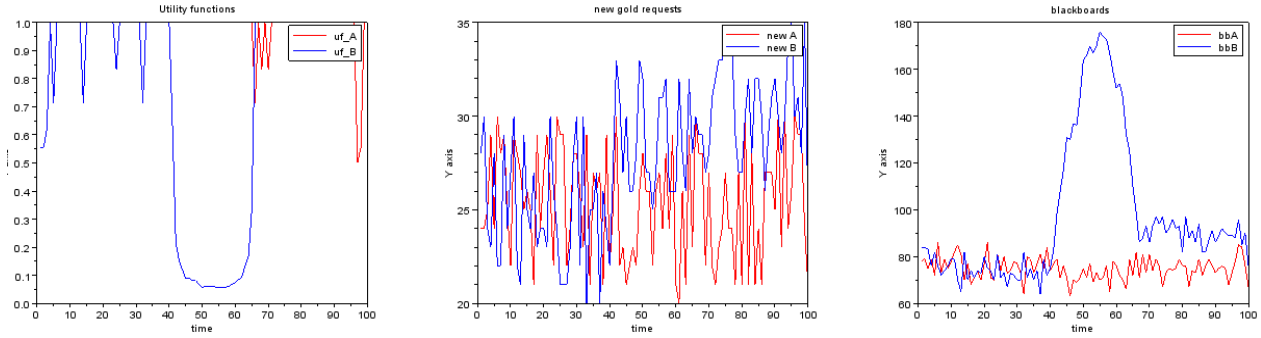
L'approccio sopra è troppo complesso per le finalità della nostra simulazione ed è stato preferito un approccio più elementare: prima tutti i gold, poi tutti i silver poi tutti i bronze. Questo approccio però ha la limitazione di non evidenziare la natura best-effort dei servizi bronze rispetto agli altri due. L'implementazione della policy sopra descritta è stata lasciata come lavoro futuro.

Modellazione della periodicità dell'orchestrazione globale: l'orchestratore deve operare periodicamente una redistribuzione delle risorse. La redistribuzione non deve avvenire troppo spesso per non interferire nell'attività dei CENode, ma neanche troppo raramente per evitare situazioni di crisi dei CENode. In questa simulazione si è scelto di far intervenire l'orchestratore ogni 10 unità di tempo.

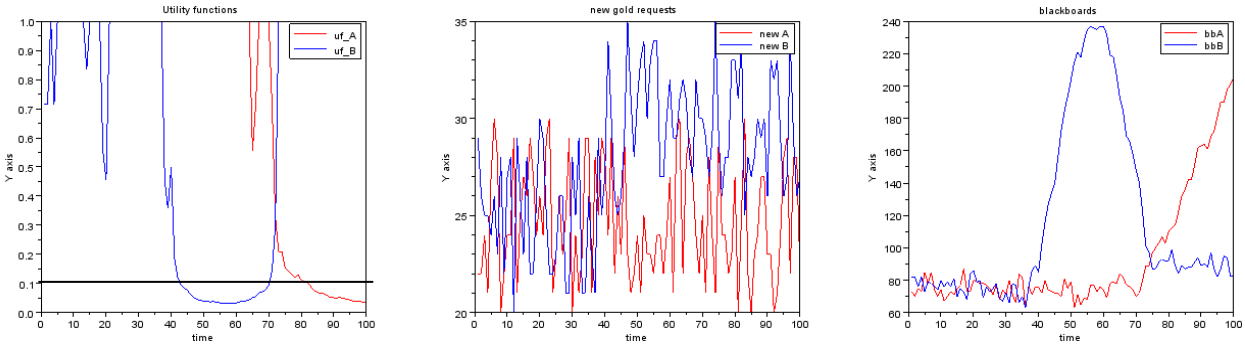
3.2.7 Risultati di simulazione

Sono state fatte quindi delle simulazioni in cui si può vedere come avviene la dinamica autonomica. A destra si vedono le utility function dei CENode A e B, al centro sono mostrate le nuove richieste di tipo gold ad ogni istante per A e B (la generazione casuale avviene allo stesso modo per tutti i tipi di richieste), mentre a destra è mostrata la dimensione delle blackboard alla fine di ogni iterazione, ovvero le richieste ancora inevase. Inizialmente A possiede un quantitativo di risorse tale da rimanere sempre con l'utility function al massimo (uf_A=1), mentre in B ogni tanto l'utility function diminuisce, ma senza mai scendere troppo. All'istante 40 avviene improvvisamente l'aumento di richieste in N_b : B non riesce più a star dietro alle richieste, aumentano le richieste inevase e l'utility function di B scende sotto la soglia critica (uf_B<0.1). L'orchestratore, che fa un controllo periodico (ogni 10 unità di tempo), decide di operare una ottimizzazione globale, spostando delle risorse da B ad A. In questo caso l'orchestratore interviene due volte (in 40 e in 50) e B riesce riportare il numero di richieste inevase a un

numero accettabile. A ha ora meno risorse a disposizione e infatti la sua utility function (quella rossa) non è più costantemente al massimo.



Nella progettazione di una rete autonomica reale l'orchestratore deve essere realizzato con molta cura: si deve evitare che l'ottimizzazione globale danneggi gli elementi del sistema che sono in "buona salute". Nel grafico sotto è mostrato un esempio in cui la redistribuzione delle risorse riesce a far recuperare l'utility function di B al prezzo di far perdere la stabilità al nodo A.



L'algoritmo di orchestrazione è lo stesso del precedente: sono state fatte molte run e si visto che in media l'algoritmo si comporta bene, ma in certi casi limite rischia di causare dei danni collaterali. In questo caso possiamo vedere che l'orchestratore interviene per ben tre volte (in 40,50 e 60), togliendo troppe

risorse al nodo A e causando il malfunzionamento. E' necessario quindi progettare e testare accuratamente l'algoritmo prima di metterlo in produzione in un sistema reale.

Infine si può notare che ci sono delle similitudini nelle finalità di questo scenario con i modelli di ottimizzazione della catena di produzione di una azienda, del cui studio si occupano gli ingegneri gestionali. In entrambe lo scopo finale è quello di ottimizzare la gestione e l'utilizzo di risorse.

Capitolo 4
Conclusioni e sviluppi futuri

4.1 Conclusioni

In questo elaborato sono stati studiate nuove teorie di rete, come Software Defined Network e Network Function Virtualization, insieme alle teorie Cognitive/Autonomics che consentono di abilitare scenari "disruptive" di rete futura. Si è visto che è molto importante capire come funzionano le tecnologie di migrazione di VM come punto di partenza per operare la virtualizzazione e migrazione di funzioni di rete.

La sperimentazione sull'algoritmo di migrazione ha consentito di evidenziare alcuni degli aspetti che influenzano maggiormente la migrazione: la dimensione della VM, il carico di lavoro (identificato in particolare nel page dirty rate), la banda di collegamento, la percentuale di CPU utilizzata. Sono stati identificati i parametri di performance: il total migration time e il downtime.

Se le migrazioni nelle reti LAN sono ben comprese le migrazioni attraverso la reti d'accesso pongono nuove sfide da affrontare: i vincoli di banda e latenza sono problemi aperti e richiedono ulteriori studi.

La sperimentazione sulle reti edge autonomiche ha consentito di esplorare quali aspetti entrano in gioco nella gestione delle risorse. L'approccio è innovativo: utilizzare le VM come mezzo per istanziare e migrare funzioni di rete all'edge. La simulazione ha consentito di evidenziare che questo approccio necessita di meccanismi autonomici, simili a quelli che si stanno cominciando ad applicare adesso nei DataCenter.

4.2 Lavori futuri

I risultati ottenuti nella simulazione di migrazione di VM possono essere confrontati con i risultati sperimentali di una vera migrazione. Questo consentirà si verificare con quanta accuratezza l'algoritmo è effettivamente in grado di predire le performance di migrazione. Per ottenere dei risultati ottimali di simulazione inoltre si può utilizzare il modello HIST e un page dirty rate campionato dall'applicazione che gira sulla VM. Ad esempio se si vuole simulare la migrazione di una funzionalità di rete si può campionare il page dirty rate di un'istanza di Click, un'applicazione software che esegue il routing. La simulazione inoltre potrebbe tenere in conto della latenza, un fattore di influenza molto importante nella migrazione attraverso la rete d'accesso.

Per lo scenario di rete autonomica la simulazione può essere ampliata e resa più complessa:

- si possono aggiungere "complex task" che per essere completati dai CENode richiedono tempi maggiori
- si possono realizzare delle blackboard totalmente condivise, che consentano ai CENode di operare politiche di "work stealing"
- si può scegliere una diversa politica di consumo delle richieste da parte del CENode, che tenga in conto la natura "best effort" dei task di tipo "bronze"
- si può operare una politica di orchestrazione più efficiente, ad esempio di tipo best-fit

APPENDICE A
OPENFLOW

a.1 OpenFlow

OpenFlow è attualmente la più famosa tecnologia che vuole immettere nelle reti attuali i concetti SDN. Tramite OpenFlow è possibile spostare il piano di controllo fuori dai dispositivi di forwarding e metterlo in un SW logicamente centralizzato.

OpenFlow consiste di un protocollo di comunicazione fra un controller e gli apparati di rete e di un modello di forwarding per questi ultimi. Il protocollo è una interfaccia standard di comunicazione fra il piano di controllo e il piano di forwarding di una architettura SDN: tramite OpenFlow un controller, che è programmato in SW, può comunicare con gli apparati di rete e modificare il loro comportamento, e in particolare va ad agire sul loro piano di forwarding. Questi apparati di rete possono essere switch o router, fisici o virtuali, ma tutti devono implementare il protocollo OpenFlow per poter essere gestiti dal controller. OpenFlow può essere paragonato a un'ISA di un computer, poiché questo protocollo definisce delle primitive con cui il controller esterno può modificare il forwarding plane dei dispositivi di forwarding. OpenFlow utilizza il concetto di flusso per identificare il traffico di rete: tramite delle regole (match rules) si possono identificare e raggruppare tipologie di traffico differenti, con differente granularità. Il routing tradizionale IP non consente questa flessibilità, poichè è basato unicamente sulle tabelle di routing IP.

a.2 Funzionamento di OpenFlow

Uno switch OpenFlow consiste di una parte data-path, composta da una o più *flow tables* e una parte di controllo. La parte di controllo all'interno del dispositivo si occupa unicamente di creare un canale sicuro con un controller remoto. In questo modo lo switch OpenFlow riceve i comandi dal controller e invia ad esso informazioni di stato. Il protocollo OpenFlow definisce lo standard con cui vengono modificate le entry delle flow table dello switch e come vengono inviati i messaggi di stato.

Ogni entry della flow table è composta da una coppia <pattern-azione>:

- Pattern: è un filtro sul pacchetto. Un pattern identifica un insieme di pacchetti, ovvero un flusso di rete. Il pattern è sui campi degli header del pacchetto.
- Azione: l'azione/i da eseguire una volta che il pacchetto coincide con il pattern. L'azione può essere ad esempio "inoltra alla porta i", "scarta il pacchetto", "spedisci il pacchetto al controller", etc.

Un esempio di flow table di uno switch:

OpenFlow-enabled Network Device

Flow Table comparable to an instruction set

MAC src	MAC dst	IP Src	IP Dst	TCP dport	...	Action	Count
*	10:20:.	*	*	*	*	port 1	250
*	*	*	5.6.7.8	*	*	port 2	300
*	*	*	*	25	*	drop	892
*	*	*	192.*	*	*	local	120
*	*	*	*	*	*	controller	11

Quando uno switch OF riceve un pacchetto fa un confronto uno a uno con le entry della flow table finchè non incontra una entry con un pattern che matcha con il pacchetto. Se ne trova una esegue l'azione associata, altrimenti chiede aiuto al controller inviandogli parte del pacchetto (l'header). Il controller a quel punto viene in soccorso dello switch istruendolo su come comportarsi: ad esempio invia un messaggio allo switch che lo istruisce di aggiungere una nuova entry (quindi modifica la flow table) oppure non aggiunge alcuna entry e istruisce lo switch a scartare il pacchetto.

a.3 OpenFlow per la ricerca e l'innovazione

Martin Casado, l'inventore di OpenFlow, quando ha creato questa tecnologia aveva pensato a OpenFlow come un mezzo per semplificare la ricerca nel campo delle reti. Tramite OF si possono programmare i dispositivi di rete per gestire separatamente tipologie diverse di traffico di rete: si può sfruttare una "production network", ovvero una rete normalmente già utilizzata per le attività di tutti i giorni (ad esempio la rete interna aziendale, oppure la rete di un campus universitario) per fare delle sperimentazioni di rete

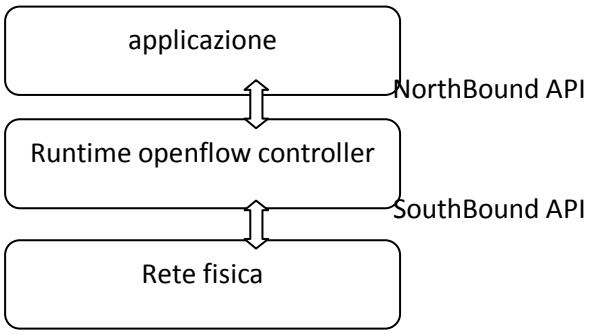
semplicemente partizionando le due tipologie di traffico. I due tipi di traffico non interferiscono e inoltre non ho perdite di performance dei dispositivi di rete, perché si possono scegliere quante risorse di rete allocare a ciascuna tipologia.

a.4 Limiti e problemi irrisolti

I detrattori di OpenFlow criticano il fatto che il modello di forwarding proposto per i nodi di rete è limitante. Tramite OpenFlow dovrebbe essere possibile tradurre il "forwarding behaviour" voluto all'interno delle *flow entry* contenute nella *flow table* dei nodi, ma secondo alcuni rimane comunque troppo complesso implementare un determinato "forwarding behaviour" specificato dalle applicazioni.

Poi ci sono alcune problematiche che rimangono ancora in sospeso, come la mancanza di una NorthBound API ben definita e il fatto che in OpenFlow non è stato considerato il piano di management. Di seguito si cerca di capire meglio queste problematiche.

Il runtime controller è uno strato SW che si pone come intermezzo fra lo strato di applicazione e la rete fisica composta dai dispositivi OpenFlow-compatibili. Le applicazioni che necessitano una visione logica/astratta della rete sfruttano questo "runtime controller", ma le diverse implementazioni dei runtime controller OpenFlow non forniscono una API generica con delle primitive ben definite. Di conseguenza ogni implementazione di controller fornisce dei metodi differenti e le applicazioni vanno realizzate dovendo considerare la specifica implementazione.



L'altro problema è quello del piano di management: attualmente esistono già delle applicazioni di management, molto complesse e specializzate in qualche particolare aspetto di gestione degli apparati di rete. Non esistono

ancora dei meccanismi che consentano di integrare le applicazioni o il controller con queste piattaforme di management.

a.5 Confronto con MPLS

Il protocollo MPLS (Multi Protocol label switching) è un meccanismo per gestire il trasporto dati in base a delle etichette. MPLS è agnostico rispetto ai protocolli e quindi può trasportare pacchetti IP, ATM, ethernet frames, etc. MPLS è a metà fra il livello data-link (livello OSI 2) e il livello di rete (livello OSI 3). Gli switch vengono configurati per effettuare il forwarding dei pacchetti a determinate porte in base alla loro etichetta. MPLS si focalizza soprattutto sul fornire un *tunnel* ad alte prestazioni da un punto di ingresso a un punto di uscita di una rete di trasporto: aggrega dei flussi di traffico per rendere il forwarding più efficace. Se nell'IP il forwarding era legato al concetto di destinazione finale, avendo una visione globale della topologia (instradamento end-to-end), nell'MPLS si cerca invece di disaccoppiare il forwarding dalla destinazione finale. Anche OpenFlow slega il forwarding dalla destinazione finale, ma il focus non è più sull'idea di tunnel ad alte prestazioni, quanto piuttosto sul fornire una maggiore flessibilità, associando una certa logica di forwarding un certo flusso.

a.6 L'ONF

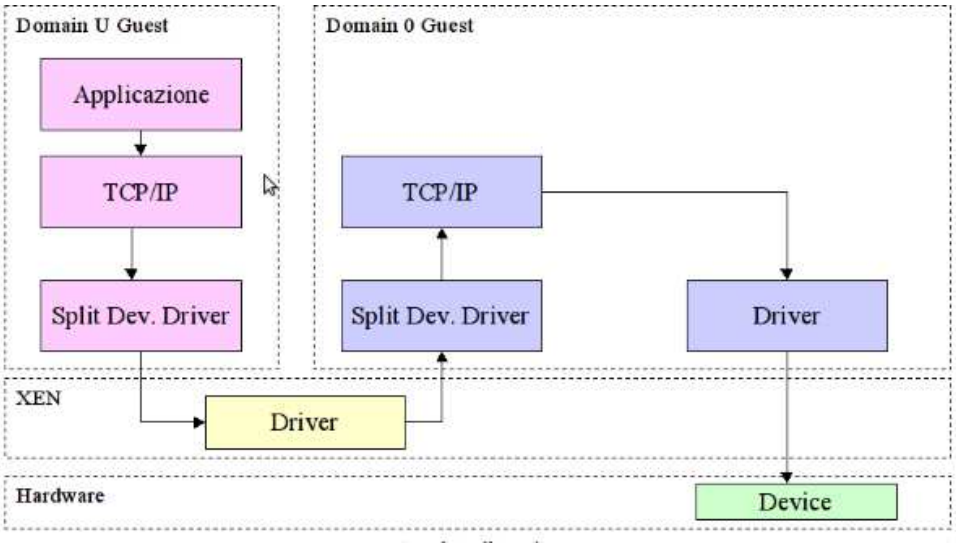
L'ONF (Open Networking Foundation) è l'organizzazione che si occupa di definire gli standard OpenFlow e di spingere sull'adozione di questa tecnologia. È un'organizzazione non-profit fondata da Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and Yahoo! a cui nel tempo si sono aggiunte decine di altre aziende. Allo stato attuale OpenFlow non è ancora uno standard riconosciuto alla pari degli standard ITU/T e l'ONF ha solo un ruolo di forum di discussione e di definizione dello standard.

APPENDICE B
XEN

XEN è un monitor di VM open source (licenza GPL) per piattaforme x86. Non esegue una emulazione completa dell'architettura x86 ma fa una paravirtualizzazione, ovvero consente alle VM di girare direttamente sull'HW limitandosi a controllare l'accesso alle risorse fisiche. Questo approccio consente alle VM di avere alte prestazioni. Gli OS destinati a girare su macchina virtuale (guest) necessitano di un kernel opportunamente modificato (xenizzato), a meno che l'HW non supporti particolari tecnologie di virtualizzazionev(Intel VT-X, AMD AMD-V,…). XEN fa una divisione in domini per gestire le VM: dom-0(il dominio privilegiato) è un'istanza di VM creata dall'hypervisor: viene avviata per prima ed è quella che fa partire tutte le altre VM in domini meno privilegiati (dom-u); dom-0 controlla l'accesso alle risorse da parte dei dom-u tramite opportuni meccanismi (shared ring, front/back-end driver,..).

I driver in XEN hanno una split-architecture: in dom-0 è presente il back-end driver, mentre nei domini dei guest-OS (dom-U) ci sono i front-end driver. Quando un programma in un SO guest in dom-U esegue un operazione che richiede certe system call, ad esempio un'operazione di I/O la richiesta deve passare attraverso il front-end driver, poi attraverso uno strumento di XEN (chiamato shared ring) e quindi al back-end driver nel dom-0. Quest'ultimo si interfaccia, con un suo driver, alla periferica.

Per i driver di rete si parla di netfront e netback.


Domain U Guest
Applicazione
TCP/IP
Split Dev. Driver
Domain 0 Guest
TCP/IP
Split Dev. Driver
Driver
XEN
Driver
Hardware
Device

APPENDICE C
LA PAGINA DI MEMORIA

Una pagina di memoria è un blocco contiguo di memoria virtuale, è la più piccola quantità di memoria nelle operazioni di:

- paging, ovvero i trasferimenti fra la memoria principale(esempio RAM) e la memoria secondaria (es. HD)
- allocazione di memoria fatta dal SO per conto di un programma

La dimensione delle pagine di memoria è decisa in base all'architettura del processore. La dimensione è stata tradizionalmente fissa a 4KB. Le architetture attuali consentono ai SO di gestire dimensioni multiple delle pagine, alcune anche simultaneamente.

Appendice C – LA PAGINA DI MEMORIA

APPENDICE D

Device file

Nei sistemi *nix (es. Linux) un device file un'interfaccia verso i driver di una periferica che viene presentata dal file system come un semplice file. Questo consente ai programmi di interagire con i driver usando delle chiamate di sistema I/O.

I block device in particolare sono relativi a periferiche che gestiscono i dati come blocchi di memoria e sono quindi legati a periferiche come l'HD, CD flash drive, etc..

Quando viene fatta la migrazione della VM è necessario ricollegarla alle periferiche del nuovo container, agendo su di questi file (si fa il mirroring dei block device).

Appendice D – Device file

APPENDICE E

MapReduce

Map-Reduce è un modello di computazione per processare un grande insieme di dati ed è usato per la computazione parallela su cluster o supercomputer.

Map-Reduce si divide in due parti: la funzione Map e la funzione Reduce. “Map” prende dei dati in input, li filtra, li trasforma e restituisce un set di risultati intermedi. Questi risultati solitamente sono divisi per valore di hash. “Reduce” processa o combina tutti i risultati intermedi associati con lo stesso valore di hash. Il processamento avviene in parallelo su un cluster di macchine a larga scala.

Appendice E – MapReduce

APPENDICE F

Prototipi di ricerca

Quando avviene un cambio di paradigma la letteratura scientifica "esplode" e nella ricerca spuntano moltissimi prototipi, anche molto diversi fra loro, che cercano di catturare un particolare aspetto delle novità introdotte dal nuovo paradigma. Per la rete adesso sta accadendo la stessa cosa e con l'avvento delle teorie SDN e NFV sono apparse una "pletora" di soluzioni diverse, prototipi di ricerca, soluzioni proprietarie o proposte standard, come OpenFlow. Di seguito verranno analizzati alcuni di questi prototipi senza entrare troppo nel dettaglio.

f.1 RouteBricks

RouteBricks è un prototipo di architettura di router SW altamente scalabile, progettato per funzionare su commodity server HW. Le prestazioni raggiunte non sono equiparabili a quelle dei router ad alte prestazioni dei vendor, ma sono comunque interessanti. Le performance di RouteBricks sono state valutate su 3 funzionalità di packet processing: IP routing, packet forwarding e IPSec encryption. Con RouteBricks si vuole avere una infrastruttura di rete capace ad alte performance e contemporaneamente programmabile.

Erano possibili 3 approcci:

- partire dai dispositivi ad alte prestazioni attuali in commercio e cercare di "aprirli" per renderli programmabili. Un esempio è la startup Nicira che ha sviluppato dei plugin in grado di rendere programmabili i router Juniper
- usare dei network processors (es. FPGA), ma sono difficili da programmare a causa della necessità di fare una programmazione a basso livello
- progettare un router interamente SW, che funzioni su standard HW. RouteBricks sceglie questo approccio sfruttando tecniche di parallelizzazione.

RouteBricks è quindi un'infrattura di routing parallelo *decentralizzata* che gira su un cluster di server con una *load-balanced interconnection*. Si ricerca il parallelismo della funzionalità di routing **fra più server** e **fra più core** dello stesso server**.**

La funzionalità di routing si può suddividere in *packet processing* (es. es. ricerca e classificazione delle routes) e *switching* (l'inoltro dei pacchetti dalle porte di input a quelle di output). Ogni server si prende carico di parte del packet processing, e in più partecipa allo switching che avviene in modo distribuito a livello di cluster.

La scelta di usare dei server general purpose e di parallelizzare la router functionality non permette di avere le stesse garanzie *strict* di performance dei router high-end tradizionali, specie per alti carichi di lavoro: questo obbliga a fornire garanzie di performance più "rilassate", specialmente per il *packet reordering* e la *latenza*.

f.2 ClickOS

Con ClickOS si è voluto realizzare un router in SW, con le funzionalità di routing di base, che sia una base di partenza per poi pensare di implementare funzionalità tipiche delle middleboxes, ad esempio NAT e firewall.

Sfruttando le tecniche SDN si vuole che le funzionalità di rete godano delle proprietà di estensibilità (possibilità di modificare/aggiungere funzionalità), dynamic instantion (utile in contesti adattativi, quando avviene una cambiamento nella rete), isolation (necessario in caso network slicing/overlay per motivi di performance e sicurezza).

ClickOS è un NET OS minimale: è basato sul Click software router ed è progettato per funzionare su piattaforma XEN.

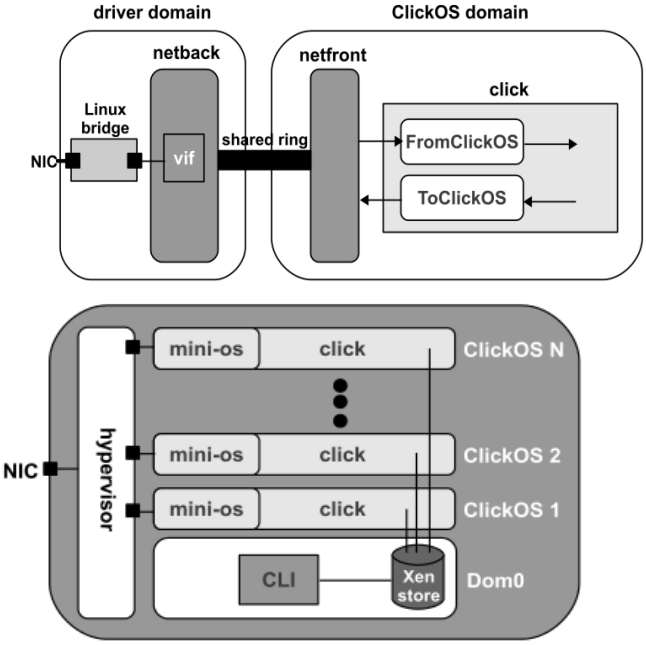
Permette quindi di avere:

- estensibilità: è interamente SW, quindi facilmente modificabile

- isolation: posso avere più VM ClickOS su XEN e più istanze “Click” sulla stessa VM, separate fra loro
- dynamic instantiation: grazie a un basso footprint (~20MB) è possibile una veloce migrazione e creazione di nuove istanze SW

Le VM ClickOS consistono di un OS minimale (mini-os, codice sorgente fornito da XEN) più leggero e performante di un kernel Linux completo, sopra il quale vengono eseguite come normali thread una o più istanze di Click 2.0.1.

In dom-0 vi è ClickOS CLI, da cui si possono impartire i comandi per il controllo delle VM/istanze Click.



La struttura e i meccanismi di controllo di XEN sono “pesanti”, quindi vengono usate diverse tecniche per incrementare le performance: ottimizzazione dei tempi di accesso alla memoria condivisa (tramite riutilizzo dei grant di XEN), la receive function per il netfront driver viene fatta a polling invece che a interrupt, ottimizzazione dello XEN shared ring (“*fast refill of request entry for netback driver”*).

f.3 Confronto fra ClickOS e RouteBricks

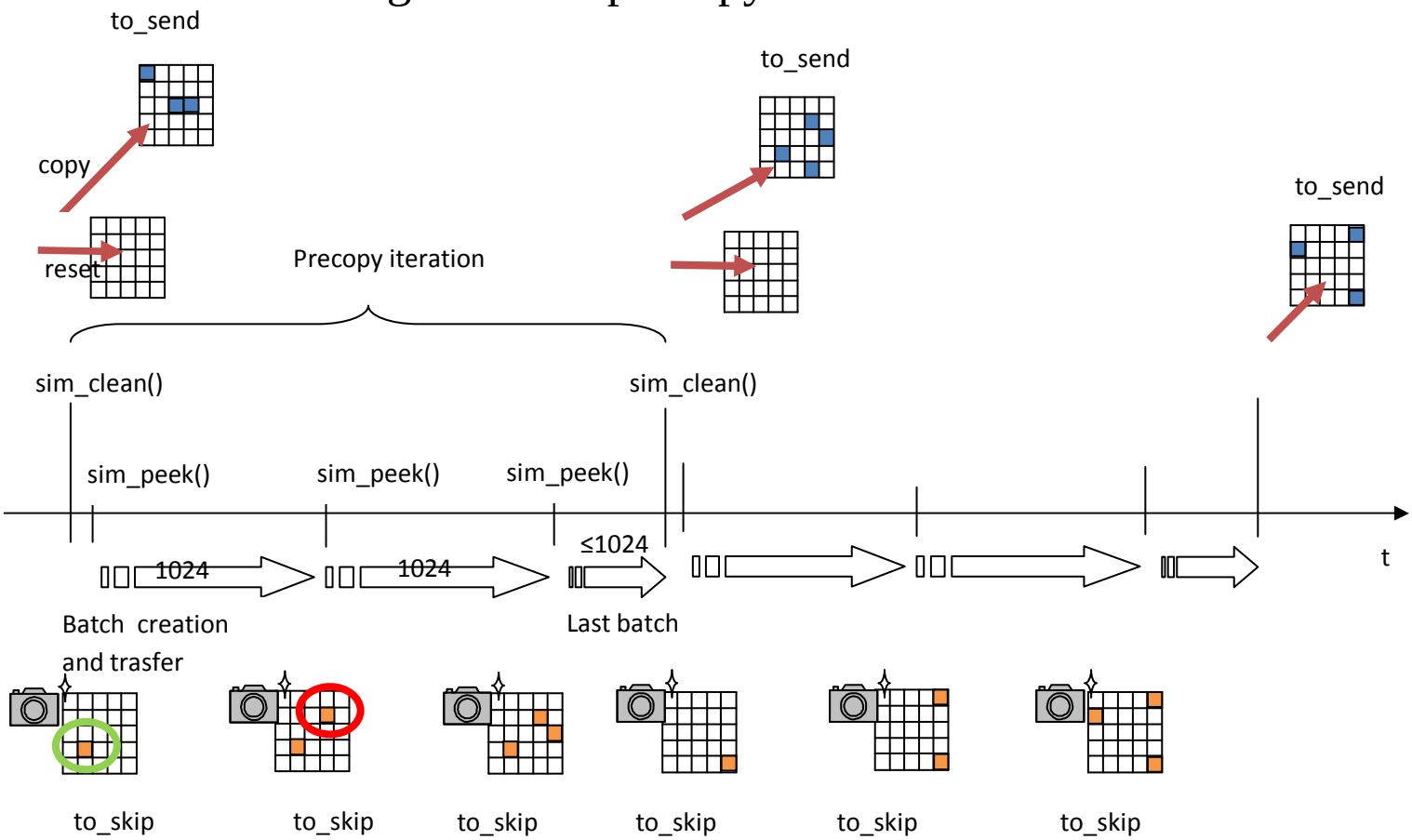
Entrambi i progetti hanno lo scopo di creare dei software router e partono dal software Click.

Il progetto ClickOS vuole ottenere tanti piccoli router leggeri, facili da istanziare e migrare, e si basa su un server XEN.

RouteBricks punta invece sulla parallelizzazione spinta e l'utilizzo di commodity server per realizzare **un solo** SW router e capire se le sue prestazioni possono avvicinarsi a quelle dei vendor router.

APPENDICE G

Algoritmo di precopy



In ogni iterazione di precopy viene scandagliata l'intera memoria, in ordine di pagina, trasferendo solo le pagine "dirty", ovvero presenti in to_send, e non "skipped", ovvero non presenti in to_skip.

All'inizio di ogni iterazione di precopy viene identificato un nuovo insieme di pagine da trasferire (to_send).

Più volte in una iterazione di precopy, e in particolare prima di trasferire un nuovo batch, viene identificato un nuvo insieme di pagine da saltare (to_skip). Se la pagina in esame è tra in to_skip non verrà inclusa nel batch, ma verrà trasferita in futuro.

Nell'esempio sopra: la pagina *dirty* cerchiata in verde è nelle zona finale della memoria mentre l'algoritmo è ancora all'inizio delgli indirizzi di memoria. Questa pagina che è fotografata al primo peek() verrà quindi saltata, ovvero non verrà inclusa nel batch da trasferire. Questo perché sono ancora in tempo a saltarla, ovvero lo sweeping della memoria non ha ancora raggiunto il punto dove si trova la pagina in esame. Nella prossima iterazione questa pagina verrà considerata nuovamente per il trasferimento.

La pagina *dirty* cerchiata in rosso invece è già stata presa in esame e quindi il fatto che sia dirty è ininfluente in questa iterazione. Nella prossima iterazione questa pagina verrà considerata nuovamente per il trasferimento.

Appendice G – Algoritmo di precopy

FONTI BIBLIOGRAFICHE

Callegati Franco, Cerroni Walter, Campi Aldo. Application scenarios for cognitive transport service in next-generation networks. *Communications Magazine, IEEE. March 2012. Page(s): 62 – 69*

Antonio Manzalini, Noel Crespi, Roberto Minerva, Uddin Shan Muhammad Emad, Capitolo ”Software Networks At the Edge” *IGI Book: Evolution of Cognitive Networks and Self-Adaptive Communication Systems*

Antonio Manzalini, Roberto Minerva, Franco Callegati, Walter Cerroni, Aldo Campi “Clouds of Virtual Machines at the Edge”, sottomesso a IEEE Com Mag Future Carriers Networks;

Antonio Manzalini, Giuliano Santandrea. Cognitive Edge Networks, towards distributed data centers at the edge.

Senthil Nathan, Purushottam Kulkarni, and Umesh Bellur. Resource Availability Based Performance Benchmarking of Virtual Machine Migrations. *International Conference on performance engineering, ICPE 2013*

Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, Guru Parulkar. Can the production network be the testbed? *OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation Article No. 1-6*

Bennani, Mohamed N. , Menascé, Daniel A. Resource Allocation for Autonomic Data Centers using Analytic Performance Models. *ICAC 2005. Proceedings. Second International Conference on Autonomic Computing, 2005.*

Zdravko Bozakov. Towards virtual routers as a service. *6th GI/ITG KuVS Workshop on Future Internet. 11/2010*

Zdravko Bozakov. Architecture and Algorithms for Virtual Routers as a Service. *ACM/IEEE International Workshop on Quality of Service, IWQoS 2011*

Christopher Clark,Keir Fraser,Steven Hand,Jacob Gorm Hansen,Eric Jul,Christian Limpach,Ian Pratt,Andrew Warfield. Live migration of virtual machines *NSDI'05 Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2. Pages 273-286*

Timothy Wood, K. K. Ramakrishnan, Prashant Shenoy, Jacobus van der Merwe. CloudNet: dynamic pooling of cloud resources by live WAN migration of virtual machines. *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments Pages 121-132*

K. K. Ramakrishnan, Prashant Shenoy, Jacobus Van der Merwe. Live Data Center Migration across WANs: A Robust Cooperative Context Aware Approach. *Proceedings of the 2007 SIGCOMM workshop on Internet network management. Pages 262-267*

Semplificare le reti di domani. Antonio Manzalini. Notiziario tecnico Telecom Italia, numero 1/2012.

Amin Tootoonchian, Yashar Ganjali. HyperFlow: A Distributed Control Plane for OpenFlow. *Proceedings of the 2010 internet network management conference on Research on enterprise networking*
http://static.usenix.org/event/inm10/tech/full_papers/Tootoonchian.pdf

ONS 2012 - OpenFlow/SDN Introduction with Brandon Heller
http://www.youtube.com/watch?v=aq_IrsONfZw

Software Defined Networks, Wikipedia.
http://en.wikipedia.org/wiki/Software-defined_networking

Questioning SDN Cynicism
http://nerdtwilight.wordpress.com/2012/09/24/questioning-sdn-cynicism/

Open Networking Foundation. Software Defined Networking: the new norm for networks. *ONF White Paper. April 13 2012*

Network Function Virtualization explained
http://wikibon.org/wiki/v/Network_Function_Virtualization_or_NFV_Explained

NFV Whitepaper
http://www.tid.es/es/Documents/NFV_White_PaperV2.pdf

William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi. Utility Functions in Autonomic Systems. *Proceedings. International Conference on Autonomic Computing, 2004*

Yanjue Xu (Tokyo Univ, Japan), Yuji Sekiya (Tokyo Univ, Japan). Scheme of Resource Optimization using VM Migration for Federated Cloud. *APAN 32, 2011*

Greg Ferro. OpenFlow and Software Defined Networking. "What is OpenFlow" Webinar.
http://demo.ipspace.net/get/OpenFlow

Akoush Sherif, Sohan Ripduman; Rice Andrew C.; Moore Andrew W. ,Hopper Andrew. Predicting the Performance of Virtual Machine migration. *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010. Page(s): 37 - 46*

Mohamed Ahmed, Felipe Huici and Armin Jahanpanah. Enabling Dynamic Network Processing with ClickOS. *ACM SIGCOMM Computer Communication Review - Special october issue SIGCOMM '12 archive*
Volume 42 Issue 4, October 2012 Pages 293-294

Mihai Dobrescu and Norbert Egi, Katerina Argyraki, Byung-Gon Chun,Kevin Fall, Gianluca Iannaccone, Allan Knies, Maziar Manesh, Sylvia Ratnasamy.

RouteBricks: Exploiting parallelism to scale software routers. *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*
Pages 15-28

RINGRAZIAMENTI

Vorrei ringraziare la mia famiglia per il sostegno durante la redazione della tesi, il mio tutor di stage Antonio e i tutti i suoi colleghi per avermi introdotto a un campo di ricerca molto interessante e avvincente e avermi dato sempre nuovi stimoli, idee e argomenti di discussione interessanti. Ringrazio gli altri ragazzi che ho conosciuto in Telecom per il tempo trascorso in compagnia. Vorrei ringraziare i miei compagni di convitto con cui me la sono “spassata” durante i sei mesi di stage a Torino, e in particolare l’ ”acqua santa” di Marco che tira sempre su il morale e l’ottima ricotta fatta in casa di Peppe. Vorrei ringraziare i miei compagni di studio per aver condiviso tante esperienze durante tutti i cinque anni di università. Ringrazio la Chiara con cui ho passato molti esami difficili e completato con successo lavori di gruppo fatti con fusi orari differenti.

Vorrei ringraziare i miei cugini e in particolare Andrea per la sua allegria contagiosa e che non vede l’ora di realizzare con me un mini dirigibile telecomandato con Arduino.