Politecnico di Torino

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE
Corso di Laurea Magistrale in Ingegneria Informatica

Analisi delle tecnologie per misurazioni di perdite e ritardo nelle reti a commutazione di pacchetto

Candidato:
Stefano Righetti

Relatore:
Prof. Fulvio Risso

Torino, anno accademico 2017/2018

Ringraziamenti

Prima di tutto, vorrei ringraziare sinceramente il Professor Fulvio Risso. Di tanto in tanto può capitare di incontrare persone che tengono a darti una mano semplicemente perché sanno quanto questo possa significare per te. In tutti questi anni è sempre stato presente per un consiglio o una parola gentile. In Lui ho sempre trovato un esempio di professionalità e cortesia e spero davvero che un giorno potrò restituire, almeno in parte, quanto ricevuto.

In questi ultimi anni, lungo il percorso che mi ha portato verso il raggiungimento di questo risultato, ho avuto il grande privilegio di incontrare persone eccezionali, provenienti da diverse parti d'Italia e aventi background molto diversi tra di loro. Ognuna di esse mi ha arricchito personalmente ed a ciascuna di loro devo una parte di questo successo. Ringrazio quindi i miei amici Vincenzo, Sandro, Fabrizio, Giovanni, Enrico e Dario.

Non può certamente mancare un ringraziamento alla mia famiglia, in primis a mia Madre e mio Fratello, che so quanto mi vogliano bene e quanto tengano alla mia felicità. A entrambi voglio dire di guardare sempre al futuro con coraggio ed ottimismo. Grazie a mia Nonna, a Silvia, Elena, Sara e Matteo. Grazie per essere sempre presenti nei momenti più importanti.

Un pensiero speciale va a mio Nonno. So con certezza quanto sarebbe orgoglioso di me. E' sempre stato una grande fonte di ispirazione ed un mio grande rammarico è quello di non potergli stringere la mano guardandolo negli occhi il giorno della mia Laurea. Anche questa sua mancanza è un'ulteriore lezione di vita che mi insegna che il tempo passa e non sempre è possibile tornare in dietro o fare meglio.

Porto sempre con me le ultime parole che mi ha detto quando chiedendogli un consiglio su una decisione che dovevo prendere lui mi rispose: "Ricordati sempre che ciò che fai lo fai solo per te stesso". Il significato non era naturalmente quello di pensare in modo egoista ma quello di non fare nulla nella vita se non ciò che vogliamo davvero per noi stessi.

Indice

Acronimi

PNPM Packet Network Performance Monitoring

PNPM SLA Packet Network Performance Monitoring Service Level Agreement

HPCAP High-Performance Capture

DMA Direct Memory Access

DNA Direct NIC Access

DNS Domain Name System

DHCP Dynamic Host Configuration Protocol

NUMA Non Uniform Memory Access

API Application Programming Interface

NAPI New API

NIC Network Interface Card

CPU Central Processing Unit

HW hardware

SW software

TI Telecom Italia

CSLD Command, Status, Length, Data

OTS Off-The-Shelf

COTS Commodity Off-The-Shelf

SO Sistema Operativo

pps pacchetti per secondo

HPCAP High-performance Packet CAPture

ISP Internet Service Provider

PC Personal Computer

I/O Input/Output

SLA Service Level Agreement

IP SLA Internet Protocol Service Level Agreement

PCI Peripheral Component Interconnect

RSS Receive Side Scaling

HTTP Hypertext Tranfer Protocol

MOS Mean Opinion Score

ToS Type of Service

QoS Quality of Service

VLAN Virtual Local Area Network

PTP Precision Time Protocol

MAC Media Access Control

KPT Kernel Level Polling Thread

SPMC Single Producer, Multiple Consumer

UTC Coordinated Universal Time

GPS Global Positioning System

NTP Network Time Protocol

LAN Local Area Network

BMC Best Master Clock

ACL Access Control List

DS Diff Service

DMTW Delay Medio Two-Way

IP Internet Protocol

TCP Transmission Control Protocol

IPTV Internet Protocol Television

MPLS Multi Protocol Label Switching

RTP Real-Time Transport Protocol

MCP Marking Cycle Period

MP Marking Period

CB Coloured Bit

IDS Intrusion Detection System

FTP File Transfer Protocol

WAN Wide Area Network

ERP Entreprise Relationship Management

CRM Customer Relationship Management

MRP Material Requirement Planning

ATS Average Timestamp

AD Average Delay

TPJM Two Point Jitter medio

OPJM One Point Jitter medio

RTDM Round Trip Delay medio

RTT Round Trip Time

VoIP Voice over IP

DMOW Delay Medio one-way

DMTW Delay Medio two-way

Elenco delle figure

2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 2.10 2.11 2.12 2.13 3.4 3.5 3.6 3.7 3.8 3.9 4.1 4.2 4.3 4.4 5.4 5.5 5.6 5.7 3.1 3.2 3.3 6.2 6.1 6.3 6.5 6.6 6.4 6.7 4.6 5.3

Capitolo 1

Introduzione

Questo capitolo presenta una panoramica sugli aspetti centrali di questa tesi, ne introduce le motivazioni e quali sono i suoi obiettivi. Quindi riporta l'organizzazione generale dei successivi capitoli fornendo una breve descrizione per ciascuno di essi.

1.1 Scenario e motivazioni

L'interesse dei principali Internet Service Provider (ISP) di aumentare e differenziare il numero di servizi offerti ed il costante incremento di utenti rende fondamentale disporre di strumenti efficienti che permettano di monitorare e valutare l'effettivo livello di servizio offerto. Questo aspetto è infatti alla base di molti rapporti tra gli ISP ed i propri clienti.
Negli ultimi anni la comunità scientifica e le principali aziende nel settore hanno compiuto numerosi sforzi per ottenere tecnologie di rete in grado di effettuare operazioni di monitoraggio e processamento del traffico con prestazioni sempre più elevate, sia da un punto di vista hardware (HW) che software (SW).
Nel contesto dell'analisi delle prestazioni di rete è necessario caratterizzare metriche quali la packet loss il delay ed il jitter prestando particolare attenzione agli apparati attraversati dal traffico dati dalla sorgente verso la destinazione. Infatti, sia la perdita di pacchetti, sia i ritardi che si possono accumulare nelle operazioni di cattura e processamento, sono spesso legati all'architettura ed al funzionamento degli apparati di rete.
Uno degli aspetti centrali trattati in questa tesi riguarda appunto l'analisi di diverse tecnologie nate con l'obiettivo di processare in SW elevate quantità di traffico. Sfruttando le potenzialità di tali tecnologie è possibile ottenere misurazioni di qualità attraverso una gestione ottimizzata del traffico all'in-

terno del dispositivo in cui esso si trova ad essere processato.

Lo sviluppo della tesi vede la collaborazione con Telecom Italia[TI]. L'azienda in questione ha sviluppato una nuova metodologia per il monitoraggio del traffico di rete, denominata Packet Network Performance Monitoring (PNPM)[PNPM]. Nei riguardi di Telecom Italia il lavoro svolto nella tesi si è concentrato su due fronti. In primo luogo nello sviluppo di un nuovo protocollo di rete, chiamato PNPM_SLA. Quindi successivamente nell'implementazione di tale protocollo in sonde attive[1] e dedicate[2] e nell'analisi delle prestazioni ottenute. PNPM_SLA può quindi essere definito come un protocollo per il monitoraggio delle prestazioni di rete che opera in modo coerente con quanto previsto dalla metodologia PNPM.

1.2 Obiettivi della tesi

I principali obiettivi di questa tesi sono due e possono essere così riassunti;

Prima parte: analisi teorica di driver di input-output accelerato e sincronizzazione; In seguito ad uno studio delle caratteristiche dei sistemi Off-The-Shelf (OTS) e di alcuni dei principali acceleratori di Input/Output (I/O) di pacchetto, è stato possibile creare il background necessario al fine di comprendere le problematiche alla base delle attività di monitoraggio, cattura e processamento del traffico all'interno delle reti a commutazione di pacchetto. In particolare questa prima parte della tesi si limita ad una analisi di tali tecnologie senza portare però ad una sua effettiva implementazione in fase di sviluppo del protocollo PNPM_SLA. L'obiettivo infatti è quello di suggerire che, al fine di ottenere misure di precisione, anche tali strumenti potrebbero portare a significativi miglioramenti rispetto ai risultati attualmente raggiungibili.

Uno dei problemi affrontati è legato all'analisi di specifici intervalli temporali, o per essere più precisi, vengono analizzate le attività a cui un pacchetto di rete viene sottoposto in tali intervalli. Ad esempio, l'intervallo che intercorre tra quando un pacchetto raggiunge la Network Interface Card (NIC)

[1] Per *sonda attiva* si intende un apparato di rete, quale ad esempio un normale Personal Computer (PC), dedicato sia alla cattura di traffico di rete, sia alla generazione, da cui l'aggettivo *attivo*.

[2] Per *sonda dedicata* si intende una sonda che abbia come principale attività quella di eseguire il protocollo PNPM_SLA

dell'adattatore di rete e quando effettivamente il medesimo pacchetto viene processato dall'applicazione dedicata, è scandito da un numero elevato di operazioni che coinvolgono sia l'hardware che il software.
Un secondo problema riguarda il tempo trascorso tra quando, a livello applicativo, un pacchetto viene trasmesso e quando tale pacchetto raggiunge realmente l'interfaccia di rete per essere instradato. Tutti gli acceleratori di I/O di pacchetto presentati nella sezione 2.3 cercano di ridurre il più possibile tali intervalli. La differenza tra quanto svolto dalla maggior parte di essi e quanto invece è l'oggetto principale di questa prima parte della tesi può essere così spiegato: *l'obiettivo principe di un generico acceleratore di pacchetti è massimizzare il numero pacchetti da catturare e processare nell'unità di tempo. Quindi l'attenzione è concentrata su elevatissime quantità di traffico*[3]*. Questa tesi invece, sfruttando le potenzialità di tali tecnologie, ha l'obiettivo di caratterizzare attraverso una gestione ottimizzata del timestamping, gli effettivi istanti di ricezione e trasmissione dei pacchetti, indipendentemente dalla quantità di traffico in gioco.*
Va inoltre sottolineata la predominanza del software come strumento di ricerca dei risultati desiderati.

Infine, vengono presentati i due principali protocolli di sincronizzazione, Network Time Protocol (NTP) e Precision Time Protocol (PTP), ed il modo in cui gli apparati di rete possono sfruttare tali protocolli per raggiungere un certo livello di sincronismo.

Seconda parte; Il secondo obiettivo affrontato riguarda invece direttamente le esigenze di Telecom Italia in merito alla necessità di sviluppare un proprio protocollo per il monitoraggio delle prestazioni di rete. Tale risultato permetterebbe infatti a Telecom Italia (TI) di rendersi parzialmente o completamente indipendente da protocolli aventi finalità simili ma di proprietà di terze parti. Questo scenario non solo porterebbe ad un risparmio in termini economici ma anche alla possibilità di avere una piena conoscenza sul modo in cui determinati risultati vengono ottenuti ed ovviamente alla capacità di apportare modifiche e miglioramenti al proprio protocollo. Infatti la metodologia PNPM, già sviluppata ed operativa, definisce alcune linee guida da seguire nelle attività di monitoraggio del traffico, senza però specificare come queste debbano essere implementate. Nell'intenzione di fornire una struttura a tale metodologia si è deciso di progettare un nuovo protocollo di rete, chiamato appunto Packet Network Performance Monitoring Service

[3]Su una interfaccia Ethernet a 10GBit/s è possibile catturare oltre 14,88 Mpps[LRN1].

Level Agreement (PNPM SLA), al fine di ricreare un ambiente di analisi organizzato secondo le specifiche di PNPM.

1.3 Struttura della tesi

Di seguito viene riportata, capitolo per capitolo, una breve descrizione di ciascuno di essi.

Cap1. Introduzione alla tesi, al suo contesto operativo ed alla definizione delle principali sfide affrontate;

Cap2. Descrizione dello stato dell'arte. Il capitolo è fondamentale al fine di formare il background necessario per la comprensione da un lato delle motivazioni che portano a determinate problematiche, dall'altro a possibili soluzioni che si potrebbero adottare per la loro risoluzione;

Cap3. Presentazione del protocollo [IPSLA] che é ad oggi il protocollo più diffuso per le attività di monitoraggio delle prestazioni di rete;

Cap4. Descrizione della metodologia PNPM così come brevettata da Telecom Italia;

Cap5. Descrizione di PNPM SLA. Questo è il capitolo centrale di questa tesi. Viene presentato sia l'algoritmo alla base del protocollo sia la struttura dei pacchetti di rete generati artificialmente. Vengono poi analizzate alcune problematiche legate alla precisione ed alla accuratezza relative alla gestione del timestamping. Ed infine, sono esposte alcune delle strategie adottate nell'attuale implementazione per tentare di ridurne gli effetti;

Cap6. Nel sesto capitolo, di particolare importanza in quanto descrive ciò che effettivamente è stato implementato, vengono presentati nell'ordine:

- Il modello di rete PNPM_SLA;
- Gli apparati presi in esame durante le prove;
- L'ambiente e l'architettura utilizzata nei test di laboratorio;
- La descrizione dei test e dei rispettivi risultati raggiunti.

Cap7. Nel settimo ed ultimo capitolo sono quindi riportate le conclusioni ed i possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

2.1 Sistemi Off-The-Shelf: HW e SW commodity

2.1.1 Introduzione

I sistemi OTS o Commodity Off-The-Shelf (COTS) si contrappongono ai sistemi custom, ovvero quei sistemi sviluppati secondo esigenze ben specifiche. La scelta di adottare o meno apparati OTS è fondamentale in quanto influisce sull'entità degli investimenti e definisce i vincoli tecnologici.
I sistemi OTS sono caratterizzati dal condividere un'architettura comune a molti apparati di fascia simile. Contengono tipicamente slot standard Peripheral Component Interconnect (PCI) per future espansioni che permettono di ottenere elevati livelli di compatibilità.
Queste caratteristiche sono molto importanti in quanto permettono di produrre o acquistare elevati volumi di hardware o software riducendone fortemente i costi, naturalmente a fronte di prestazioni spesso più contenute.
Un'ulteriore motivazione per cui è importante discutere su OTS è invece indipendente da fattori economici. Infatti, se è pur vero che fino a pochi anni fa il principale motivo per orientare un'attività di ricerca ad apparati OTS era primariamente economica, è altrettanto vero che oggi grazie all'evoluzione raggiunta dai moderni device(ad esempio: Central Processing Unit (CPU) multi-core, code Receive Side Scaling (RSS), nuovi adattatori di rete(NIC), acceleratori di I/O di pacchetto,etc.) questi nuovi apparati permettono di ottenere ottime prestazioni.
Un'ultima osservazione da fare è la seguente: sia lo sviluppo degli apparati che compongono la rete sia l'esigenza di servizi sempre più evoluti stanno portando il software a ricoprire un ruolo fondamentale per ottenere le

prestazioni desiderate. Non è più sufficiente fare affidamento esclusivamente sull'evoluzione dell'hardware.

2.1.2 Schede di rete programmabili

Negli ultimi anni le NIC si sono trasformate da semplici dispositivi di interfacciamento tra la rete ed il calcolatore in sistemi molto più complessi, in grado di avere un importante impatto sulle prestazioni nelle fasi di cattura e processamento del traffico di rete.
Vengono quindi riportate sinteticamente alcune delle principali evoluzioni adottate dalle NIC più recenti:

- Un primo esempio è rappresentato dalla tecnologia RSS sviluppata in Intel ed in Microsoft. RSS permette alle NIC di distribuire il carico del traffico ricevuto all'interno di un ambiente multi-core sfruttandone quindi meglio le potenzialità;

- Un secondo miglioramento si ottiene grazie a filtri hardware che possono essere programmati direttamente nella scheda di rete. Questa operazione di filtraggio, chiamata Flow Director, permette di filtrare i pacchetti in base a differenti criteri: indirizzo sorgente/destinazione e porte, Type of Service (ToS), protocolli di livello 3 e 4, e valori di Virtual Local Area Network (VLAN) ed Ethertype;

- Un ulteriore funzionalità offerta da alcune schede Intel ad 1GbE e 10GbE è la capacità di ottenere il timestamping in hardware con elevate precisioni [Intel 82580].

2.1.3 Stack di rete del sistema operativo

Di seguito viene presentato il percorso che un generico pacchetto segue normalmente da quando viene intercettato dalla NIC a quando viene processato dall'applicazione a livello utente. Questa descrizione è utile per comprendere la complessità alla base del processo di cattura ed elaborazione dei pacchetti e permette di individuare sia alcune delle cause che possono portare alla perdita di pacchetto sia ai principali punti in cui il pacchetto può subire dei ritardi lungo tale percorso.
Per queste ragioni i paragrafi successivi si concentrano sull'analizzare alcune strategie per ottimizzare tale percorso e su come sia possibile ridurre la packet loss.

Cattura in *modalità standard.* Nelle normali condizioni operative quando una NIC intercetta un pacchetto, viene letto l'indirizzo Media Access Control (MAC) di destinazione riportato nel campo *Destination MAC Address* del frame relativo. Se tale indirizzo non corrisponde o al proprio indirizzo MAC o all'indirizzo MAC broadcast il pacchetto viene semplicemente scartato. Se viceversa c'è corrispondenza viene copiato all'interno di un buffer. A questo punto la NIC informa il kernel che un nuovo pacchetto è stato ricevuto ed è pronto per essere processato. Quindi il frame è trasmesso dalla NIC al kernel, il quale lo processa eseguendo un controllo di errore ed eliminando le intestazioni del pacchetto non più necessarie per il suo proseguimento. Dal kernel, il pacchetto è trasferito in un buffer per essere poi utilizzato dalle applicazioni di livello utente. L'applicazione che richiede il pacchetto eseguirà una system call e pulirà successivamente la coda del buffer.

Cattura in *modalità promiscua.* In un contesto dove un utente vuole catturare tutto il traffico visto da una certa NIC, indipendentemente dall'indirizzo MAC di destinazione o dalla sua correttezza, la NIC deve essere settata in modalità promiscua. In sostanza, questa operazione permette alla NIC di accettare tutti i pacchetti intercettati. Come in modalità standard, i pacchetti una volta intercettati dalla NIC, sono copiati in un buffer e poi trasferiti al kernel.
Siccome, in modalità promiscua, l'obiettivo è catturare tutti i pacchetti esattamente come questi sono visti in rete, non è più ammissibile che il kernel li scarti o ne elimini delle parti. Per questa ragione il pacchetto viene trasferito inalterato all'interfaccia dell'applicazione di cattura. Questa interfaccia è spesso creata attraverso l'uso di una libreria quale, ad esempio, [libpcap] . Libpcap mette a disposizione una Application Programming Interface (API) per la cattura dei dati attraverso la creazione di un dispositivo virtuale per applicazioni come [TCP DUMP] e [SNORT] per la lettura dei pacchetti. Libpcap è responsabile di realizzare la system call per ottenere il pacchetto ed infine pulire la coda.

Ora che i principali step logici seguiti da un pacchetto lungo la sua ascesa verso lo user-level sono definiti è possibile identificare le aree in cui è più probabile che si verifichi una perdita di tale pacchetto piuttosto che un suo rallentamento.

- Se il buffer utilizzato dalla NIC per contenere i nuovi pacchetti in attesa di essere trasferiti al kernel è saturo l'interfaccia di rete sarà costretta

a scartare i nuovi arrivi. Ciò può avvenire sia se la NIC sta ricevendo i pacchetti a velocità troppo elevata per le sue capacità sia se il kernel è impegnato in altre operazioni e non può quindi scaricare il buffer dell'adattatore;

- Una seconda causa di packet loss è dovuta alla possibilità che il buffer dedicato sia pieno quando il kernel tenta di passare la copia del pacchetto all'interfaccia di libpcap. Questo può accadere se l'applicazione che sta usando libpcap non è in grado di processare con sufficiente velocità i pacchetti.

Settando quindi la NIC in modalità promiscua si incrementa il numero di pacchetti che il kernel deve gestire causando così un incremento sul carico elaborativo della CPU. Questo incremento può quindi paradossalmente portare ad una potenziale riduzione della capacità del sistema di catturare traffico di rete.

Evoluzioni

I sistemi operativi più utilizzati [OSPU] forniscono uno stack di rete general-purpose [1] che privilegia le proprietà di compatibilità piuttosto che le performance. Ad esempio, lo stack di rete adottato in Linux, nei kernel precedenti alla versione 2.6, segue un meccanismo noto come *interrupt-driven.*
In pratica, ogni volta che un nuovo pacchetto viene catturato dall'adattatore, viene agganciato ad un descrittore all'interno di un buffer circolare posto nella NIC. Il descrittore del pacchetto contiene le informazioni che riguardano l'indirizzo della regione di memoria dove il pacchetto sarà copiato via trasferimento Direct Memory Access (DMA)[2], operazione che tipicamente può portare alla successiva esecuzione di un interrupt [3] per ottenere il

[1] Per dispositivi general purpose si intendono dispositivi non dedicati ad un solo possibile utilizzo, ma apparati versatili, progettati cioè tenendo presente una molteplicità di possibili destinazioni d'uso.

[2] In informatica ed elettronica il DMA (Direct Memory Access, accesso diretto alla memoria) è un meccanismo che permette ad alcuni sottosistemi hardware di un computer (periferiche) di accedere direttamente alla memoria di sistema per scambiarsi dati, oppure leggere o scrivere, senza chiamare in causa la CPU per ogni byte trasferito tramite l'usuale meccanismo dell'interrupt, ma generando un singolo interrupt per blocco trasferito.

[3] Un interrupt o interruzione è un segnale asincrono che indica la necessità di risorse da parte di una periferica finalizzata ad una particolare richiesta di servizio, un evento sincrono che consente l'interruzione di un processo qualora si verifichino determinate condizioni (gestione dei processi) oppure più in generale una particolare richiesta al sistema operativo da parte di un processo in esecuzione.

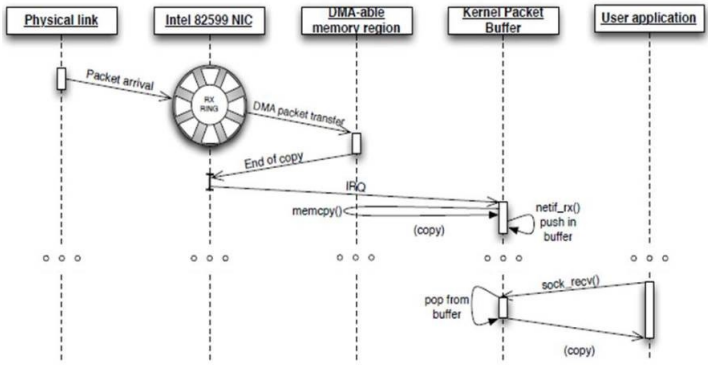


Figura 2.1: Stack di Rete Standard.

pacchetto relativo. Quando il frame deve invece essere trasmesso, il trasferimento DMA è fatto nella direzione opposta e la linea fisica di interrupt è rilasciata una volta che il trasferimento è stato completato in modo tale che nuovi pacchetti possano essere trasmessi. A questo punto viene sollevata una richiesta di interrupt(IRQ) diretta a notificare al kernel che un pacchetto è pronto per essere processato. Quindi la corrispondente routine software di interrupt copia il pacchetto dall'area di memoria nella quale il trasferimento DMA ha lasciato il pacchetto, all'interno una struttura locale del kernel, chiamata *sk_buff*. Una volta che viene fatta la copia del pacchetto, il corrispondente descrittore viene rilasciato cosi che la NIC possa riutilizzarlo per la ricezione di nuovi frame. Infine, la struttura sk_buff, contenente il pacchetto appena copiato viene resa accessibile all'applicazione utente.
Quanto appena descritto può essere rappresentato graficamente dalla figura 2.1. Il concetto chiave in questo schema di *packet I/O* è la necessità di sollevare una richiesta interrupt ogni volta che un pacchetto è ricevuto o è trasferito, modello interrupt-based appunto, meccanismo che sovraccarica l'host quando il carico di rete è elevato[PMSR1] .

Con l'obiettivo di superare questo problema, la maggior parte dei driver di rete ad alta velocità 2.3 fanno uso di un approccio New API (NAPI) [NAPI] . Questa caratteristica è stata incorporata a partire dal kernel 2.6 per migliorare il processamento dei pacchetti in ambienti ad alta velocità.
NAPI contribuisce a migliorare la velocità nella cattura dei pacchetti grazie a due principi:

- Interrupt mitigation. Come appena definito, il traffico ricevuto ad alta velocità utilizzando i tradizionali schemi genera numerosi interrupt

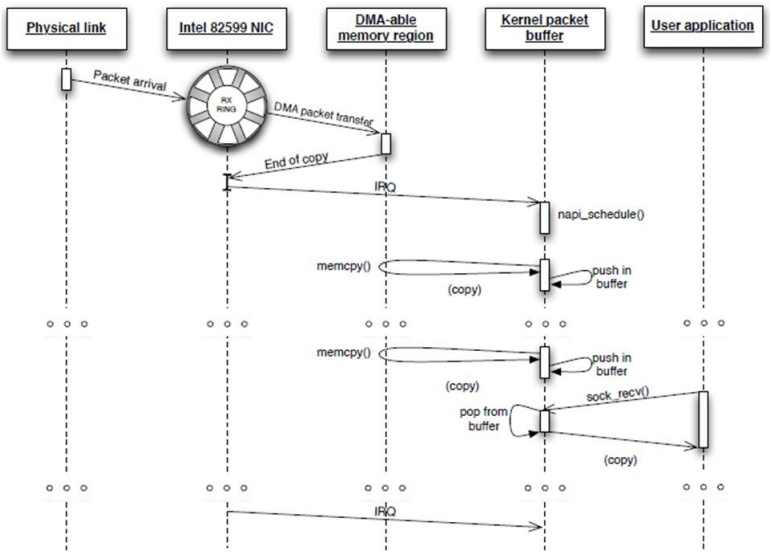


Figura 2.2: Stack di Rete Ottimizzato.

al secondo che possono portare il sistema in uno stato di sovraccarico peggiorandone le prestazioni. Per mitigare questo problema, quando viene ricevuta una richiesta di interrupt dovuta alla ricezione o alla trasmissione di un pacchetto, il driver *NAPI-aware* lancia una prima volta l'interrupt, ma differentemente dall'approccio tradizionale, anzichè copiare il pacchetto, lo inserisce in una coda. Quindi la routine di interrupt schedula l'esecuzione di una funzione *poll()* (può essere considerato come un nuovo thread), e disabilità richieste simili di interrupt future. Da questo momento è il nuovo *thread poll()* a controllare l'arrivo di nuovi frame e, quando arrivati, a copiarli ed accodarli nello stack di rete, senza più aspettare per una richiesta di interruzione. La modalità *polling* richiede un maggior consumo di CPU rispetto alla modalità *interrupt-driven* quando il carico di rete è basso, ma la sua efficienza cresce all'aumentare del carico, quando appunto il costo degli interrupt diventa estremamente significativo. La figura 2.2 schematizza quanto appena presentato;

- Packet throttling. Ogni volta che il traffico ad alta velocità supera la capacità elaborativa del sistema, i pacchetti devono essere scartati. I precedenti driver non *NAPI-aware* droppavano tali pacchetti a livello kernel, sprecando sforzi in comunicazioni e copie tra i driver di rete ed il kernel. I driver *NAPI-compliant* possono invece scartare il traffico direttamente nell'adattatore di rete attraverso meccanismi di controllo

del flusso, evitando cosi operazioni non necessarie.

2.1.4 Conclusioni

A fronte di quanto discusso in questo capitolo è possibile concludere che l'utilizzo di sistemi OTS basati su commodity HW e SW open-source sono ormai una valida alternativa all'hardware specializzato in quanto, grazie alle numerose innovazioni tecnologiche, permettono di ottenere elevate prestazioni di calcolo e di networking.

2.2 Cattura e processamento del traffico ad alte prestazioni

2.2.1 Introduzione

In questa sotto sezione vengono quindi descritte prima le principali limitazioni dei sistemi operativi correnti in merito alla cattura dei pacchetti, e poi alcune soluzioni per superare o ridurre tali limiti.

La figura 2.3 richiama per chiarezza lo schema logico di un generico sistema di monitoraggio. Il primo livello rappresenta la NIC, segue un driver che accede alla NIC ponendo i dati a disposizione del kernel (secondo livello). Il terzo livello è una architettura che consente l'accesso ai dati da parte delle applicazioni di livello utente. In generale la combinazione del driver e del framework viene definita come il motore di cattura. In fine il quarto livello rappresenta il livello applicativo, dove vengono implementate le differenti funzionalità in base a ciò che si vuole ottenere.

2.2.2 Limiti nella cattura del traffico di rete

Il processo di packet sniffing presenta un certo numero di potenziali criticità che, nonostante l'innovazione ottenuta dalle tecniche NAPI, sono ancora una possibile fonte di spreco di risorse.
Di seguito vengono riportate alcune problematiche che possono riscontrarsi durante l'attività di cattura dei pacchetti [PMSR2]:

- **Allocazione e de-allocazione delle risorse**. Ogni volta che la NIC riceve un nuovo pacchetto viene allocato un descrittore di pacchetto. Il descrittore viene rilasciato non appena il frame viene inoltrato al livello utente. Questo processo di allocazione e de-allocazione genera

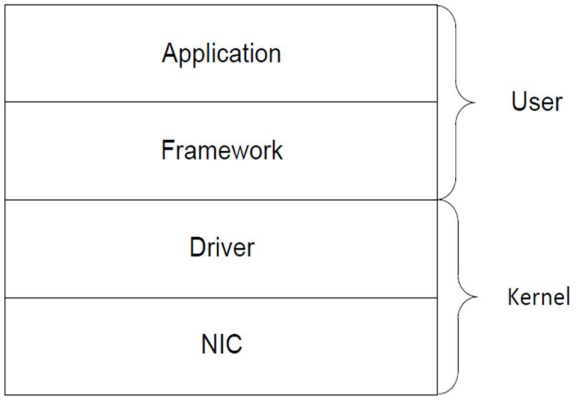


Figura 2.3: Schema concettuale di un generico stack di rete.

un overhead, in termini di tempo , che cresce all'aumentare del tasso di ricezione dei pacchetti;

- **Accesso serializzato al traffico**. Le moderne interfacce di rete includono tecnologie, quali RSS, che permettono di distribuire su più code il traffico ricevuto ottenendo una parallelizzazione del processo di cattura. Il problema nasce quando a livello utente i pacchetti vengono raggruppati in un unico buffer venendo così a formarsi un collo di bottiglia. Il processo di serializzazione a livello utente degrada le prestazioni del sistema perdendo il vantaggio ottenuto al livello dei driver.
  Inoltre, mischiando il traffico proveniente da code differenti, si può verificare una sovrapposizione nell'ordine con cui i pacchetti sono effettivamente stati ricevuti. La figura 2.4 illustra il concetto appena descritto;

- **Generazione di più copie dei frame dal driver fino al livello utente**. I pacchetti sono copiati almeno due volte: una prima volta dalla regione di memoria *DMA-able* del driver verso un buffer presente nel kernel, ed una seconda volta dal buffer nel kernel verso un buffer nello *user-level*. Queste operazioni richiedono un dispendio di risorse in termini di cicli di CPU;

- **Chiamate a sistema**.Per ogni pacchetto ricevuto è necessario realizzare una system-call che implica il passaggio dallo spazio utente allo spazio kernel e viceversa, con conseguente utilizzo delle risorse della CPU.

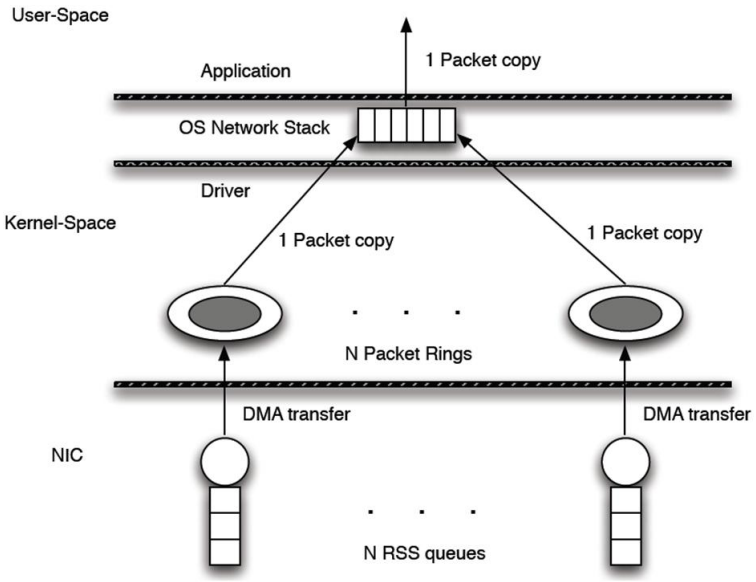


Figura 2.4: Accesso serializzato ai pacchetti.

2.2.3 Tecniche per ridurre i limiti nella cattura dei pacchetti

Vengono quindi presentate alcune delle possibili tecniche per limitare i problemi sopra citati:

- **Pre-allocazione e ri-uso delle risorse di memoria**. Questa tecnica consiste nell'allocare le risorse di memoria necessarie alla memorizzazione dei pacchetti, prima dell'inizio della ricezione del traffico. Quando un pacchetto viene trasferito verso lo user-space, il suo corrispondente descrittore non viene rilasciato, ma viene riutilizzato per memorizzare il riferimento ad un nuovo pacchetto in ingresso. Questa strategia permette di evitare l'hoverhead conseguente alla continua allocazione e de-allocazione dei descrittori per ogni nuovo pacchetto;

- **Percorsi paralleli diretti**. Al fine di risolvere il problema della serializzazione ed aumentare la scalabilità sono necessari percorsi paralleli tra le code in ingresso e le code a livello applicativo. Questa tecnica è mostrata nella figura 2.5;

- **Memory mapping**. Questa tecnica permette di mappare al livello applicativo la regione di memoria del kernel, potendo cosi leggerla e scrivervi senza dover utilizzare delle copie intermedie con conseguente risparmio in termini di cicli di CPU;

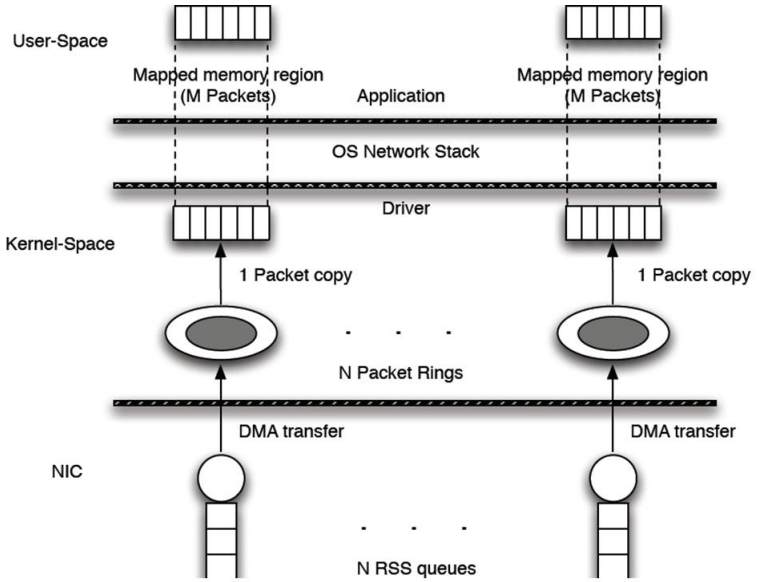


Figura 2.5: Esempio di due percorsi paralleli diretti.

- **Batch processing**. Questa soluzione consiste nell'accumulare un certo numero di pacchetti in un buffer per poi copiarli nella memoria mappata (kernel/user) in gruppi chiamati appunto batches. Questo approccio permette di ridurre il numero di system call ed il conseguente cambio di contesto, riducendo il numero di copie generate. Il principale problema delle tecniche di batching è l'incremento della latenza e del jitter, oltre alla non accuratezza con cui viene letto il timestamp dei pacchetti ricevuti, infatti prima di essere effettivamente processati, devono aspettare che il bacth sia pieno o, in alternativa, la scadenza di un timer;

- **Affinity e prefetching**. In generale sfruttando tecniche di memory, CPU ed interrupt affinity è possibile aumentare le prestazioni ottimizzando il processo di ricezione dei dati sfruttando al meglio le risorse disponibili di un sistema. In aggiunta il driver può fare un prefetch del pacchetto successivo mentre il pacchetto corrente è in fase di processamento. L'idea alla base del prefetch è quella di caricare nella cache del processore le locazioni di memoria che saranno utilizzate nell'immediato futuro con maggiore probabilità allo scopo di accedervi in tempi ridotti non appena necessario.

2.2.4 Conclusioni

A questo punto, dopo aver discusso prima dell'architettura generale dello stack di rete di un sistema operativo general-purpose e poi delle principali caratteristiche riguardanti le fasi di cattura e processamento di un pacchetto di rete appare evidente come, non potendo intervenire sulla struttura intrinseca di un sistema operativo sia necessario, al fine di ottimizzare le prestazioni, agire da prima direttamente sugli adattatori di rete e poi sui driver che, in sostanza, interfacciano la NIC ai livelli superiori.
La vera sfida è quella di ottimizzare il percorso seguito dal pacchetto dalla NIC verso l'applicazione in user-level e viceversa. Nelle pagine precedenti sono già state presentate diverse tecniche che seguono questa direzione.
Nel sotto capitolo successivo vengono presentati in modo più dettagliato i vantaggi che si possono trarre dall'utilizzo dei driver di I/O accelerato.

2.3 Accelerazione dell'I/O dei pacchetti di rete

2.3.1 Introduzione

La domanda a cui questo capitolo vuole dare una risposta è la seguente: dal momento che il principale obiettivo teorico di questa tesi non è quello di prendere in esame un ambiente in cui il fattore di studio sia l'elevato rate di traffico, quanto invece ricavare il timestamp nel modo più accurato possibile, qual'è l'obiettivo che porta a prendere in considerazione l'utilizzo di acceleratori dell'I/O dei pacchetti di rete?
Una possibile risposta è la seguente: *L'intervallo di tempo che intercorre tra il reale istante di ricezione di un pacchetto da parte dell'adattatore di rete e l'istante in cui viene invocata la funzione per ottenere una marcatura temporale può non essere trascurabile. Pertanto è bene riuscire a ridurre il più possibile tale intervallo. Ed è proprio questa la ragione per cui sarebbe utile adottare sistemi di accelerazione dell'I/O dei pacchetti. Va sempre ricordato che stiamo considerando sistemi OTS.*

Prendendo come riferimento un link a 10 Gbit/s è possibile ricevere fino a 14.88 milioni di pacchetti per secondo (pps), il che significa che il sistema deve essere in grado di processare un pacchetto ogni 67.2ns. Questa semplice osservazione, sommata a quanto fin'ora discusso rende evidente la necessità di adottare degli strumenti che permettano attraverso HW e SW commodity di ottimizzare le attività di cattura e trasporto dei pacchetti dalla NIC al Sistema Operativo (SO) e viceversa.
Nel presente capitolo vengono da prima presentate alcune soluzioni hardware.

Questo argomento tuttavia non viene approfondito non essendo oggetto della tesi. Dopo di che vengono invece affrontate differenti soluzioni software. Per ogni *Packet I/O engines*, viene descritta l'architettura del sistema, la API e quali sono le principali funzionalità da esso offerte.

2.3.2 Soluzioni hardware

Adattatori quali netFPGA [netFPGA] permettono di ridurre il carico elaborativo sulla CPU durante la cattura e la trasmissione dei pacchetti. Il risultato derivante dall'utilizzo di questi adattatori hardware è la capacità di catturare e trasmettere pacchetti di dimensione minima, ad un tasso di trasmissione e ricezione molto elevato, con un modesto carico sulla CPU.
Un'altro vantaggio ottenibile con queste schede è l'abilità di prendere via HW il timestamp dei pacchetti. Questa caratteristica è disponibile solo su pochi adattatori HW quali ad esempio l'Intel 82580 [Intel 82580]. Questi adattatori ricevono e trasmettono i pacchetti in DMA leggendo o scrivendo il pacchetto all'interno di banchi di memoria installati sulla scheda stessa. All'interno di essa viene registrato il valore relativo all'istante di ricezione dei frame. Il timestamp misurato è quindi approssimabile all'esatto istante in cui il pacchetto raggiunge la NIC. Il traffico in ingresso può essere inoltre taggato in accordo con le regole di filtraggio adottate in modo da bilanciare il traffico attraverso adattatori che facciano utilizzo di code per accumulare tali pacchetti. L'abilità di sincronizzare via Global Positioning System (GPS) il clock della scheda utilizzata per il timestamping dei pacchetti rende queste schede in grado di soddisfare esigenze di processamento estremamente avanzate [RDC] . Purtroppo il prezzo di un adattatore come un NetFPGA-10G è superiore in media ai 1700 $ [netFPGA+].

2.3.3 Soluzioni software

A questo punto vengono discusse alcune soluzioni software sviluppate per cercare di avvicinarsi ai risultati ottenibili con l'hardware. L'obiettivo non è quello di approfondire in modo estremamente accurato le seguenti tecnologie, ma piuttosto è quello di presentarne le principali caratteristiche, e soprattutto evidenziarne le similitudini e le differenze che li contraddistinguono.

Netmap

netmap è un framework progettato per consentire ai pacchetti di rete un accesso molto rapido alle applicazioni in user-space, o viceversa per il trasferimento di un pacchetto dal livello utente all'interfaccia di rete. L'efficienza

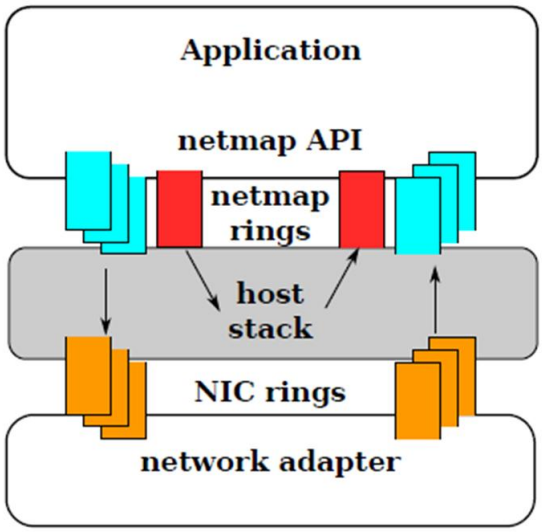


Figura 2.6: NIC in modalità netmap.

non è però ottenuta a spese della sicurezza delle operazioni, infatti in netmap tutte le azioni potenzialmente pericolose, come ad esempio programmare la NIC, sono validate dal SO il quale gestisce sempre direttamente la protezione della memoria [LRN12] .

Le principali tecniche implementate in netmap sono le seguenti:

- **Pre-allocazione e riutilizzo della memoria.** La dimensione del buffer dei pacchetti è fissa ed è allocata nella fase iniziale evitando un continuo processo di allocazione e de-allocazione per ogni nuovo pacchetto;

- **Batch processing.** Processamento di un elevato numero di pacchetti mediante un'unica system call, aspetto che, come è già stato discusso in 2.2.3, permette di ridurre i costi di CPU;

- **memory mapping.** Rimozione dei costi di *data-copy* garantendo alle applicazioni un accesso diretto e protetto ai buffer di pacchetto;

- **Percorsi paralleli diretti.** netmap permette di gestire il trasferimento diretto e parallelo di pacchetti dalla NIC verso lo user-level.

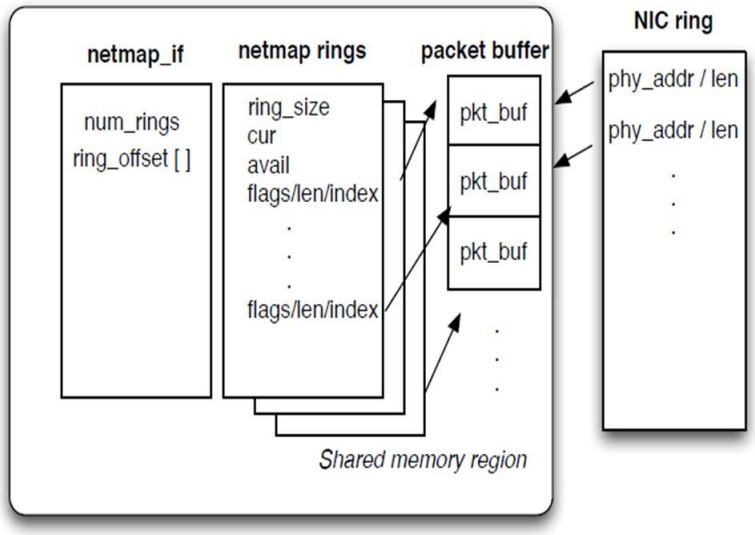


Figura 2.7: Struttura dati netmap.

Strutture dati in netmap. Un componente chiave in una architettura netmap è la struttura dati mostrata in figura 2.7. Tali strutture sono progettate per fornire le seguenti caratteristiche:

- ridurre l'overhead dovuto al processamento *per-packet*;
- ottenere un forwarding efficiente tra le interfacce;
- comunicazioni efficienti tra la NIC e lo stack host;
- supporto per adattatori con diverse code HW e sistemi multi-core.

Netmap supporta le caratteristiche sopra elencate associando ad ogni interfaccia tre tipi di oggetti *user-visible*, così come mostrato in figura 2.7.

Tutti gli oggetti per tutte le interfacce *netmap-enabled* risiedono nella stessa regione di memoria, allocata dal kernel in una area non paginabile, e condivisa da tutti i processi utenti. Al fine di ottenere un certo isolamento tra i vari processi e thread la memoria condivisa è mappata in differenti spazi di indirizzamento virtuale. I buffer di pacchetto hanno una dimensione fissa e sono condivisi dalle interfacce di rete e dai processi utente. Ogni buffer è identificato da un indice univoco, che può facilmente essere tradotto in un indirizzo virtuale dai processi utente o dal kernel, ed in un indirizzo fisico utilizzato dai motori DMA delle NIC. I buffer per tutti i ring netmap sono

pre-allocati quando l'interfaccia è settata in *netmap mode*. I meta-dati che descrivono il buffer (indice, lunghezza dei dati, alcuni flag) sono memorizzati dentro alcuni slot che sono parte dei ring di netmap. Infine, ogni buffer è referenziato da un ring netmap e dal corrispondente ring hardware.

Un ring netmap è sostanzialmente una replica *device-indipendent* di una coda circolare ed include:

- la dimensione del ring ed il numero di slot contenuti nel ring;
- la posizione di lettura o scrittura corrente nel ring (cur);
- il numero pacchetti ricevuti nei ring di ricezione, ed il numero di slot vuoti nei ring di trasmissione(avail);
- un array contenente la dimensione delle entries nel ring. Ogni slot dell'array contiene l'indice del corrispondente buffer di pacchetto, la lunghezza del pacchetto ed alcuni flag usati per richiedere operazioni speciali sul buffer.

Le strutture dati netmap sono condivise tra il kernel e lo user-space. Per evitare però che si verifichino dei conflitti, l'accesso a tali strutture può essere cosi rappresentato: il netmap ring è sempre posseduto dalle applicazioni in user-space ad eccezione del periodo di esecuzione di una system call. I thread del kernel non toccano mai un netmap ring. I buffer di pacchetto compresi tra cur e cur+avail-1 sono posseduti dalle applicazioni in user-space, mentre i buffer rimanenti sono posseduti dal kernel. Il limite tra le due regioni è aggiornato durante le system call.

API netmap. In questo paragrafo vengono riportate le linee guida utili a comprendere il funzionamento della API in netmap.

Un programma imposta un'interfaccia in netmap mode aprendo uno speciale device, */dev/netmap*, ed emettendo una *ioctl(..,NIOCREG,arg)* sul file descriptor. L'argomento, *arg*, contiene il nome dell'interfaccia, ed opzionalmente l'indicazione di quali ring vogliono essere controllati attraverso il file descriptor. Se la chiamata ha successo, la funzione ritorna la dimensione della regione di memoria condivisa dove vengono collocate tutte le strutture dati, e l'offset di *netmp_if* all'interno della regione. Una successiva funzione, *mmap()*, rende la memoria accessibile nello spazio di indirizzi del processo. Una volta che il file descriptor è vincolato ad una interfaccia ed al suo ring(s), due ulteriori chiamate ad *ioctl()* supportano la trasmissione e la ricezione dei

pacchetti. In particolare, la trasmissione richiede al programma di riempire i buffer *avail* nel ring di trasmissione, iniziando dallo slot *cur* (la lunghezza dei pacchetti è scritta nel campo *len* dello slot), e poi una istanza di *ioctl(..,NIOCTXSYNC)* per segnalare al SO la necessità di trasmettere il nuovo pacchetto. Questa chiamata a sistema passa le informazioni al kernel, ed al ritorno aggiorna il campo *avail* nel netmap ring, riportando gli slot che sono diventati disponibili a causa del completamento della trasmissione precedente.
Sul lato della ricezione, i programmi devono chiamare una *ioctl(.., NIOCRXSYNC)* per chiedere al sistema operativo quanti pacchetti sono disponibili per la lettura. La loro lunghezza ed il relativo payload sono immediatamente disponibili attraverso gli slot (partendo da *cur*) nel netmap ring.
Le chiamate *ioctl()* sono non bloccanti, non richiedono la copia dei dati (ad eccezione per la sincronizzazione degli slot nei ring), e possono gestire più pacchetti alla volta. Queste caratteristiche sono essenziali per ridurre l'overhead di gestione per pacchetto.

Infine, la parte in-kernel di queste system call opera nel modo seguente:

- valida i campi *cur/avail* ed il contenuto degli slot coinvolti (lunghezza ed indice dei buffer, sia nei ring netmap sia nei ring hardware);
- sincronizza il contenuto degli slot tra i ring hardware ed i ring netmap, ed invia i comandi alla NIC per segnalarle di inviare nuovi pacchetti;
- aggiorna il campo *avail* nel netmap ring.

Di seguito è mostrato un esempio di utilizzo della API in netmap. A parte alcune macro utilizzate per navigare attraverso la struttura dati nella regione di memoria condivisa, i netmap clients non necessitano di nessuna libreria, ed il codice risulta compatto e leggibile.

```
fds.fd = open(/dev/netmap, O_RDWR);
strcpy(nmr.nm_name, ix0);
ioctl(fds.fd, NIOCREG, &nmr);
p = mmap(0, nmr.memsize, fds.fd);
nifp = NETMAP_IF(p, nmr.offset);
fds.events = POLLOUT;
for (;;){
poll(fds, 1, -1);
for (r = 0; r ¡ nmr.num_queues; r++){
ring = NETMAP_TXRING(nifp, r);
```

```
while (ring− >avail−− > 0){
i = ring-¿cur;
buf = NETMAP_BUF(ring, ring− >slot[i].buf_index);
... memorizza il payload in buf ...
ring− >slot[i].len = ... // imposta la lunghezza del pacchetto
ring− >cur = NETMAP_NEXT(ring, i);
}
}
}
```

A differenza di altri sistemi di I/O di pacchetto (come ad esempio PF_RING DNA) i processi che utilizzano netmap non possono causare un crash del kernel. Infatti, l'area di memoria condivisa non contiene regioni critiche di memoria del kernel, e gli indici dei buffer e le rispettive lunghezze sono sempre validate dal kernel prima di essere utilizzate.
Un processo mal configurato può tuttavia corrompere i ring di qualche altro netmap ring o buffer di pacchetti. La soluzione più semplice a questo problema è quella di implementare una regione di memoria separata per ogni ring, cosi che i clients non possano interferire tra di loro.

Una API è poco utile se non ci sono applicazioni che ne facciano uso, ed un ostacolo significativo allo sviluppo di nuove interfacce di programmazione è la necessità di adattare il codice già esistente alla API. Seguendo un approccio comune per i problemi di compatibilità, esiste una piccola libreria che si innesta sopra a netmap, la quale mappa le chiamate di libpcap all'interno delle chiamate di netmap.

PF_RING e PF_RING DNA

Anche se Direct NIC Access (DNA) e netmap sono state sviluppate in modi totalmente indipendenti, esse risolvono lo stesso problema: come spostare i pacchetti da/verso un adattatore di rete senza utilizzare troppi cicli di CPU. Le prestazioni di netmap e di DNA sono sostanzialmente le stesse [LD] .

PF_RING è una patch del kernel di Linux che permette di aumentare le prestazioni nella cattura dei pacchetti di rete grazie alla capacità di evitare alcuni layer del kernel e rimpiazzando alcune chiamate a sistema, quali ad esempio la *read()*, via DMA. PF_RING è *driver-indipendent* in quanto opera

all'interno del kernel tra la parte bassa dello stack di rete ed al di sopra della NAPI. Questo significa che qualsiasi driver può essere usato assieme a PF_RING.

PF_RING implementa un buffer circolare di memoria, realizzato secondo la tecnica di *memory-mapped*, allocato alla creazione del socket, sul quale i pacchetti in ingresso vengono copiati. Le applicazioni utente possono leggere i pacchetti semplicemente accedendo alla memoria ed incrementando l'offset relativo all'ultimo pacchetto letto. Questa soluzione evita chiamate a sistema per la lettura dei pacchetti. Inoltre, proprio come in netmap, la memoria è allocata staticamente senza quindi l'esigenza di allocazione e de-allocazione della memoria per ogni singolo pacchetto.
Allo scopo di aumentare ulteriormente queste caratteristiche, nel 2005 è stata introdotta una variante di PF_RING chimata PF_RING DNA(Direct NIC Access)[LD05].

Come per netmap, vengono anticipate per chiarezza da subito quali sono le principali funzionalità supportate e non supportate da PF_RING DNA per poi approfondirne gli aspetti più significativi:

- Pre-allocazione e riutilizzo della memoria;
- Percorsi diretti paralleli;
- Memory mapping;
- CPU, memory ed interrupt affinity;
- non supporta tecniche di aggressive prefetching;
- non supporta il batch processing.

Nella modalità DNA il buffer circolare sul quale i pacchetti in ingresso sono copiati è stato sostituito con un ring di memoria allocato dal driver per ospitare i puntatori ai pacchetti in ingresso. Siccome il memory ring ha bisogno di essere mappato nello user-space, un driver di rete modificato alloca il ring utilizzando uno spazio di memoria contiguo e non-swappable. Così come in netmap, i pacchetti in ingresso sono copiati nel ring dall'adattatore di rete, e l'applicazione in user-space che gestisce il buffer è responsabile della lettura dei pacchetti e dell'aggiornamento degli indici del prossimo slot che ospiterà i pacchetti entranti. Tutte le comunicazioni con l'adattatore di rete avvengono in DMA, così PF_RING mappa in user-space sia il packet

memory ring sia i card registres. Contrariamente a netmap questo non permette alle applicazioni nello spazio utente di manipolare i card-registres ma piuttosto di aggiornare lo stato della scheda attraverso system calls. L'approccio DNA massimizza le prestazioni evitando chiamate a sistema non necessarie. Tuttavia applicazioni mal configurate potrebbero potenzialmente settare i registri a valori non validi portando le applicazioni a non ricevere più i pacchetti. Questa possibilità è però considerata un problema minore, se confrontata ai benefici che si ottengono oltrepassando completamente il kernel durante il processamento dei pacchetti. Inoltre, siccome gli adattatori moderni utilizzano più code di ricezione, la versione 10 Gbit di DNA, crea un ring di memoria per ogni coda, così che l'applicazione possa leggere i pacchetti da singoli ring.
I dettagli implementativi di DNA sono nascosi all'applicazione utente che legge i pacchetti usando una API responsabile della manipolazione dei pacchetti e dell'aggiornamento dei registri.

Sono di seguito riportate alcune ulteriori caratteristiche di DNA:

- La maggior parte delle caratteristiche di PF_RING sono disponibili anche in DNA, il quale é più veloce ma con la limitazione che una coda in ricezione non può essere condivisa da più applicazioni. Invece più thread che risiedono nella stessa applicazione possono accedere in concorrenza alla medesima coda in quanto la libreria in user-space è thread-safe;

- Allo scopo sia di leggere i pacchetti sia di aggiornare i registri della scheda, per notificare la lettura dei pacchetti, DNA utilizza tecniche di memory mapping;

- I buffer di memoria sono allocati per coda, nel rispetto dei principi di località della memoria, aspetto che rende le tecniche Non Uniform Memory Access (NUMA) adatte allo scopo;

- In DNA, quando nessuna applicazione sta leggendo pacchetti dalle code, il kernel non svolge alcuna attività evitando alla CPU operazioni inutili, come ad esempio le notifiche di ricezione di pacchetti. Questo implica che in DNA il processo di interrupt sia abilitato solo durante le chiamate *poll()* che sono fatte esclusivamente quando non c'è nessun pacchetto disponibile per il processamento;

- Cosi come le code sono completamente indipendenti e gestite in user-space senza coinvolgimenti del kernel, le applicazioni che risiedono sopra

di esse sono completamente indipendenti e possono essere mappate su differenti core della CPU. Il risultato è che il sistema scala in modo lineare in funzione del numero di core, ed ogni coppia coda-applicazione, è indipendente dalle altre.

API in DNA. In DNA la API supporta un insieme di funzioni per l'apertura dei dispositivi per la cattura dei pacchetti in modo simile a quanto svolto in netmap. Lavora nel modo seguente: in primo luogo, l'applicazione deve essere registrata, *pfring set application name()*, e prima di ricevere i pacchetti, il socket di ricezione può essere configurato con diverse funzioni, quali ad esempio *pfring setdirection(), pfring set socket mode() o pfring set poll duration()*. Una volta che il socket è stato configurato, viene abilitato alla ricezione mediante il metodo *pfring enable ring()*. Successivamente al processo di inizializzazione, ogni volta che l'utente vuole ricevere dei dati deve usare la funzione *pfring recv()*. Infine, quando l'utente termina il processo di cattura vengono chiamate le funzioni di *pfring shutdown()* e *pfring close()*. Il processo viene quindi ripetuto per ogni coda di ricezione.

HPCAP

High-performance Packet CAPture (HPCAP) è una soluzione di packet sniffing basata su principi simili alle precedenti soluzioni, ma con alcune differenze significative prima fra tutte un timestamping del pacchetto più accurato. La figura 2.8 rappresenta la struttura generale di un sistema che utilizza come acceleratore di I/O un sistema HPCAP. In sintesi, HPCAP è implementato attraverso un thread a livello kernel, uno per ogni coda di ricezione. Per questo motivo si può dire che HPCAP adotta un approccio Kernel Level Polling Thread (KPT).

Esattamente come fatto per netmap e PF_RING DNA vengono subito riportate le principali caratteristiche di HPCAP.

- Tecniche supportate:
  - Pre-allocazione e riutilizzo della memoria;
  - Percorsi diretti paralleli;
  - Memory mapping;
  - CPU, memory ed interrupt affinity;
  - Aggressive Prefetching.

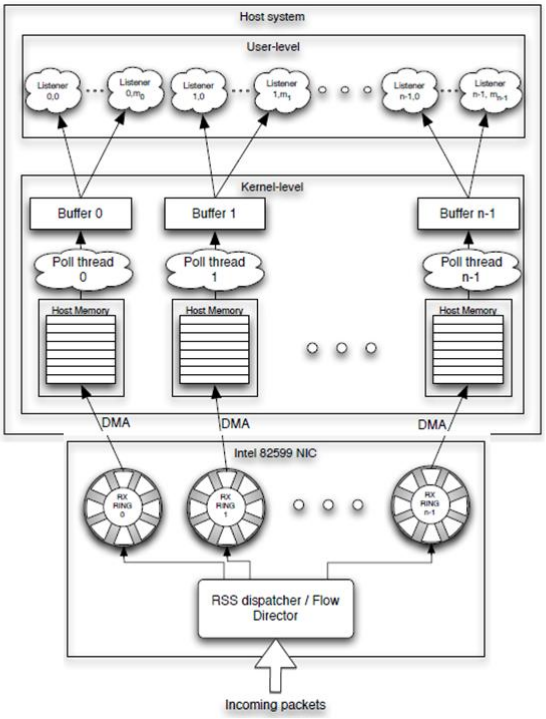


Figura 2.8: HPCAP schema

- Tecniche non supportate:
  - tecniche di zero-copy;
  - Batch-processing.

Con l'obiettivo di massimizzare le prestazioni, un sistema HPCAP crea un KPT per ogni coda di ricezione. Questi thread interrogano costantemente i corrispondenti descrittori del ring di ricezione nell'adattatore di rete, leggendo il primo descrittore nella coda per controllare se è già stato copiato nella memoria dell'host via DMA. Se il thread rileva che ci sono pacchetti disponibili nel ring, questi saranno copiati all'interno del buffer circolare del rispettivo thread. *Appena prima che questa copia sia eseguita, il thread chiamerà la funzione getnstimeofday() del kernel Linux al fine di ottenere il system time. Va fatto notare come a differenza degli acceleratori visti fin'ora la funzione di marcatura è realizzata prima della copia del pacchetto ai livelli superiori. I pacchetti sono marcati prima di essere copiati e di conseguenza il timestamp è più accurato. Questo miglioramento nell'accuratezza del timestamping è però ottenuto a spese della CPU.*

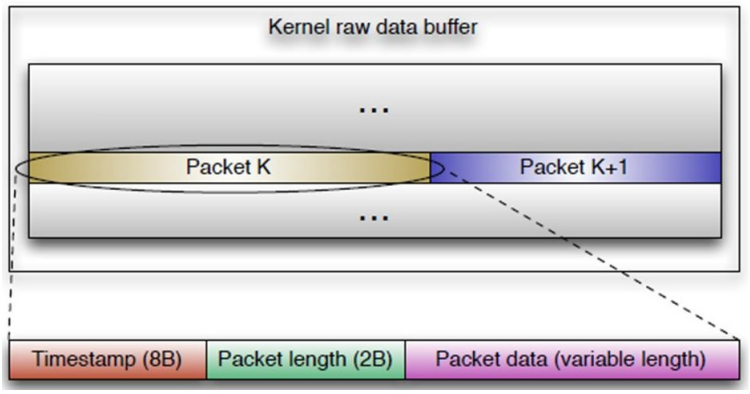


Figura 2.9: HPCAP kernel packet buffer

La memoria del buffer circolare è pre-allocata, prima di iniziare il processo di cattura, evitando proprio come in netmap ed in DNA l'allocazione dinamica per pacchetto. La dimensione può essere configurata ed il suo valore di default è 1GB e, a causa dei limiti della memoria del kernel, tutti i thread devono condividere tale valore di 1 GB di memoria. Per esempio, se si ha una interfaccia con 4 code, la lunghezza del buffer di ogni coda sarà pari a 256MB.
Tutti i pacchetti in ingresso sono copiati all'interno del buffer in un formato dati come quello mostrato nella figura 2.9: prima il timestamp del pacchetto è copiato (32-bit per il campo dei secondi ed altri 32-bit per il campo dedicato ai microsecondi), e poi, la lunghezza del pacchetto (un campo da 16-bit), e nell'ultimo campo i dati veri e propri del pacchetto. Questo formato dati fornisce un livello di astrazione maggiore rispetto alle tipiche strutture dati utilizzate ad esempio da *PCAP-lib*.

Ascoltatori multipli. Sempre come mostrato in figura 2.8, HPCAP supporta simultaneamente più applicazioni, definite come *listeners*, riceventi i pacchetti dalla stessa coda di ricezione in un sistema Single Producer, Multiple Consumer (SPMC).
Per questo motivo, il buffer di pacchetto ha un puntatore in scrittura (dove il *traffic sniffer* copia i pacchetti) e più puntatori in lettura (dove ogni listener legge i pacchetti). Allo scopo di mantenere la consistenza tra i dati il throughput di lettura dovrebbe essere settato al valore dell'ascoltatore più lento.

La struttura di un listener è costituita da quattro campi:

- **un identificatore del lister.** Il campo *listener identifier* è utilizzato per identificare i differenti listener attivi per ogni coda di ricezione. Questo campo è necessario per mantenere la consistenza tra differenti richieste di pacchetto provenienti da diversi ascoltatori. Il campo *listener identifier* è riempito quando un nuovo listener apre una sessione HPCAP;

- **un puntatore in lettura.** Questo campo è utilizzato per conoscere l'inizio del buffer di memoria dove deve essere eseguita la copia da trasferire in user-space quando un'applicazione utente solleva una richiesta di lettura;

- **un puntatore in scrittura.** Questo campo dice al KPT, dove copiare un nuovo pacchetto in ingresso. Quindi il campo è in sola lettura ed è aggiornato dal KPT attraverso operazioni atomiche, non sono necessari meccanismi di concurrency-proven;

- **un contatore di byte disponibili.** Questo contatore indica la quantità di bytes disponibili in ogni coda di ricezione. Il suo valore è incrementato dal KPT e decrementato dal thread in lettura più lento. In questo caso devono invece essere applicate tecniche di concurrency-proven per evitare che tale campo assuma valori non consistenti.

Queste caratteristiche permettono alle applicazioni di monitoraggio di concentrarsi sul processamento dei pacchetti, mentre un'applicazione differente li memorizza in un'area memoria in attesa di essere processati, permettendo quindi la sovrapposizione delle attività di data storing e data processing ottimizzandone le performance.

HPCAP packet reception scheme La fig. 2.10 mostra uno schema di ricezione dei pacchetti in HPCAP. Come in netmap, il processo di ricezione dei pacchetti non si basa più su un meccanismo *interrupt-driven*. Infatti, è un KPT che costantemente copia i pacchetti all'interno del buffer corrispondente, oltre ad eseguire, un attimo prima della copia l'operazione di timestamping. Va notato come le copie non sono fatte quando è un'applicazione in user-space a richiedere un pacchetto, ma sono fatte non appena sono disponibili nuovi pacchetti nell'adattatore di rete. Allo scopo di ottenere le migliori prestazioni sia il KPT sia il suo listener associato devono essere mappati per essere eseguiti nello stesso nodo NUMA.

API di livello utente La filosofia del kernel linux, dove *tutto è un file*, fornisce un modo semplice per comunicare tra applicazioni di user-level ed il

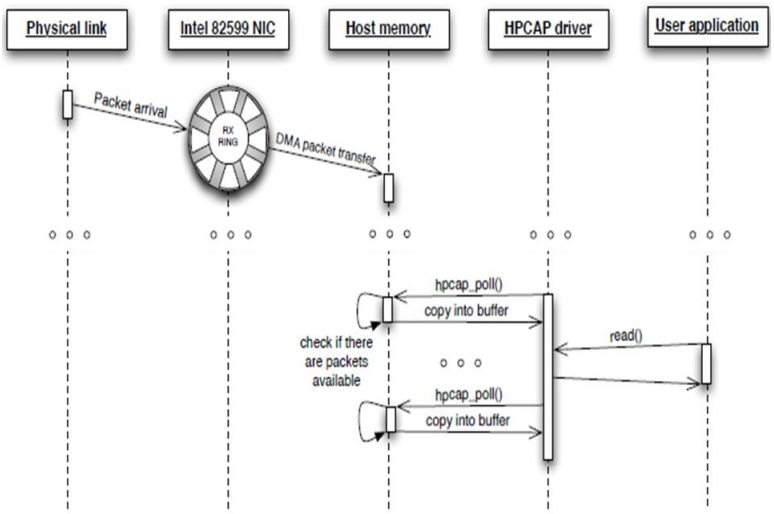


Figura 2.10: Schema di RX HPCAP.

driver HPCAP [VMM]. Viene istanziato un dispositivo, */dev/filesystem*, per ognuna delle code in ricezione appartenente ad ogni interfaccia disponibile. In questo modo, così come mostrato in figura 2.11, un sistema che gestisce N interfacce di rete con M code di ricezioni ciascuna vedrebbe i seguenti *device* nella sua */dev/directory*.

Una applicazione di livello utente che voglia catturare i pacchetti in arrivo sulla coda X dell'interfaccia $xgeY$ deve eseguire la funzione *open()* sul device textit/dev/hpcap xgeY X. Una volta che l'applicazione ha aperto il file relativo, essa può leggere i file in arrivo attraverso la chiamata al metodo *read()* eseguita sul file appena aperto. Infine, quando l'applicazione vuole interrompere la lettura dal buffer del kernel, esegue una chiamata *call()* sul device file.

2.4 Sincronizzazione

2.4.1 Introduzione

Disporre di un riferimento temporale accurato, affidabile e ad alta risoluzione è importante in diversi contesti. Ovviamente è particolarmente rilevante nelle attività di analisi e monitoraggio dello stato di una rete, piuttosto che all'interno di contesti multimediali e *real-time*, quali ad esempio nel Voice

```
...
/dev/hpcap_xge0_0
/dev/hpcap_xge0_1
        ...
/dev/hpcap_xge0_<M-1>
/dev/hpcap_xge1_0
/dev/hpcap_xge1_1
        ...
/dev/hpcap_xge<N-1>_0
...
/dev/hpcap_xge<N-1>_0
/dev/hpcap_xge<N-1>_1
        ...
/dev/hpcap_xge<N-1>_<M-1>
...
```

Figura 2.11: Contenuto in /dev/ directory in un sistema che esegue HPCAP.

over IP (VoIP), e così via. Le tecnologie utilizzate per sincronizzare i clock in ambienti distribuiti sono molteplici e dipendono dallo specifico contesto applicativo. Alcuni tra i protocolli più diffusi ed utilizzati per ottenere il sincronismo sono il protocollo [NTP] ed il protocollo [SNTP] da esso derivato. Un ulteriore protocollo utilizzato per ottenere il sincronismo è il PTP.

NTP vs PTP: punti chiave.

- NTP è utilizzato per la sincronizzazione a livello applicativo, ad esempio per la lettura/inserimento del timestamp nei pacchetti appartenenti ad una sessione audio/video. NTP è ampiamente diffuso in Internet e permette di ottenere una accuratezza dell'ordine di alcuni millisecondi;
- PTP è utilizzato per ottenere una sincronizzazione maggiore rispetto ad NTP, tipicamente attraverso il supporto di hardware dedicato. Permette di ottenere un sincronismo superiore ai 100 ns;
- Entrambi operano in banda su reti Ethernet;
- Entrambi incapsulano il riferimento temporale nei pacchetti NTP o PTP;

- Entrambi operano in modalità Master/Slave;

Un'ulteriore possibilità è sfruttare segnali radio mediante satelliti GPS. Tuttavia, questa soluzione richiede dei ricevitori GPS relativamente costosi installati nel clock degli apparati riceventi oltre alla necessità di disporre di antenne appropriate. Questa soluzione permette di ottenere elevatissime precisioni ma ad un costo spesso proibitivo, non solo in termini economici quanto sopratutto in termini di logistica.
In generale, soluzioni che sfruttino le tecniche SW di sincronizzazione sono più che sufficienti nella maggior parte dei contesti. Tuttavia per applicazioni più esigenti, che richiedono quindi precisioni inferiori alle decine di microsecondi o addirittura nanosecondi, non sono sufficienti.

2.4.2 NTP

La sotto rete dei server NTP include attualmente diverse migliaia di macchine sparse geograficamente nel mondo. NTP realizza una struttura gerarchica disposta su un certo numero di livelli o strati, così come schematicamente presentato in figura 2.12.
I server di primo livello (o di strato 1) sono sincronizzati direttamente con una fonte temporale esterna quale ad esempio un orologio atomico, GPS. I server di secondo livello ricevono i dati temporali da server dello strato precedente, e così via. Un server ottiene il sincronismo confrontando il suo orologio con quello di diversi altri server dello stesso strato o dello strato superiore. Questo meccanismo permette di aumentare la precisione del proprio riferimento temporale ed eliminare dalla rete NTP eventuali server con riferimenti errati.
NTP utilizza Coordinated Universal Time (UTC) ed è quindi indipendente dai fusi orari. Attualmente è in grado di sincronizzare gli orologi dei computer su internet entro un margine di 10 millisecondi e con una accuratezza di almeno 200 microsecondi all'interno di una Local Area Network (LAN) in condizioni ottimali.

In un apparato che supporti NTP, il server NTP è costituito da un processo operante a livello utente e da alcune altre funzioni accessorie. Al fine però di ottenere le migliori prestazioni è utile che la parte relativa alla sincronizzazione dell'orologio sia implementata nel kernel del sistema operativo, piuttosto che affidata ad un processo utente.

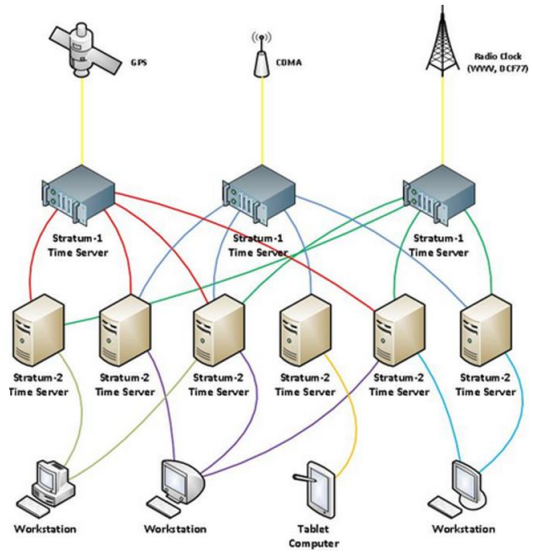


Figura 2.12: Schema concettuale dell'organizzazione dei server NTP

Sostanzialmente i pacchetti NTP contengono informazioni di timestamp da scambiare tra i server NTP e gli host attraverso la rete. Il dato, a 64 bit, scambiato dal protocollo è suddiviso in 32 bit che definiscono i secondi ed altri 32 bit per la parte decimale, potendo così rappresentare un intervallo ampio 2^{32} ed una risoluzione temporale teorica di 2^{-32} secondi. Ogni 2^{32} secondi l'intervallo temporale descritto dal protocollo si ripete, per cui è necessario specificare a quale periodo ci si riferisce. Questo non è un problema poiché 2^{32} secondi corrispondono a circa 136 anni.

Implementazione di NTP Il modello delle performance NTP include quattro statistiche che sono aggiornate ogni volta che un nodo client esegue una misura con un nodo server.

- Il *Delta (δ)* rappresenta la differenza tra il valore del clock di sistema (del nodo client) ed il clock del server;
- L'offset *theta (θ)* rappresenta la massima probabilità di sfasamento tra il clock del server rispetto al clock di sistema;
- La dispersione *epsilon (ϵ)* rappresenta l'errore massimo ereditato nella misura. Esso aumenta ad un tasso pari al valore di *maximum disciplined system clock frequency tolerance*;

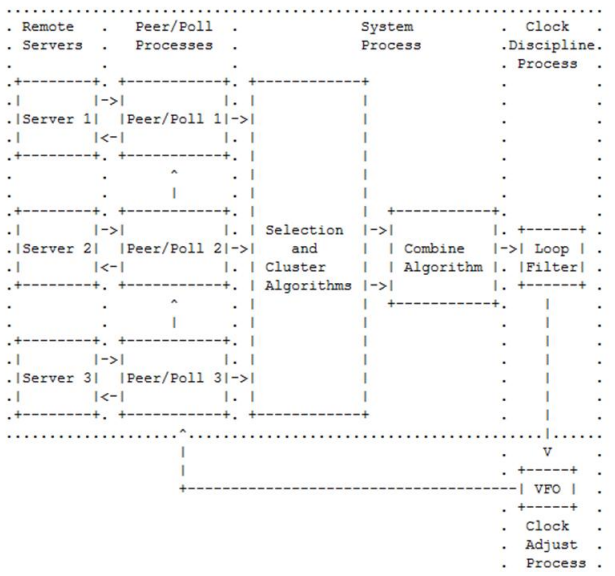


Figura 2.13: Modello di implementazione di NTP.

- Il jitter *PSI* ψ è definito come la radice quadratica media (*root-mean-square (RMS) average*) della più recente differenza di offset, e rappresenta l'errore nominale nella stima dell'offset.

Il protocollo NTP include meccanismi per combinare le statistiche di diversi server per disciplinare in modo più accurato e calibrato il clock di sistema. Considerando quindi una popolazione di server si ha che l'offset di sistema (θ) rappresenta la massima probabilità di offset stimata per l'intera popolazione dei server. Il jitter di sistema (ψ) rappresenta l'errore nominale nella stima dell'offset di sistema. Le statistiche δ ed ϵ sono accumulate ad ogni livello (stratum) dal clock di riferimento per produrre le statistiche di root delay (δ) e root dispersion (ϵ).
La distanza di sincronizzazione *lambda* (λ), uguale a $\epsilon + \delta / 2$ rappresenta l'errore massimo dovuto a tutte le altre cause.

La figura 2.13 rappresenta l'architettura di una tipica implementazione multi-thread. Include due processi dedicati ad ogni server, un process peer per ricevere i messaggi dal serve clock di riferimento, ed un processo poll per trasmettere i messaggi al server.

Questi processi operano su una struttura dati comune, chiamata *associazione*, la quale contiene le statistiche sopra descritta oltre ad alcune ulteriori informazioni. Un client invia un pacchetto ad uno o più server, il quale in

pratica scambia gli indirizzi di sorgente e destinazione e le porte, sovrascrive alcuni campi nel pacchetto e li restituisce al client immediatamente oppure, a seconda che operi in modalità client/server o in modalità simmetrica, un certo tempo dopo. Non appena il client riceve il messaggio NTP in risposta dal server, viene calcolato il valore dell'offset theta tra il clock del peer ed il clock di sistema, insieme alle statistiche associate δ, ϵ e ψ.

A questo punto il system process sfrutta degli algoritmi per determinare il candidato più affidabile, tra i vari nodi server contattati, per sincronizzare il clock di sistema. L'algoritmo di selezione sfrutta i principi di rilevazione di guasti bizantini per scartare dalla popolazione i candidati presumibilmente non corretti chiamati *falsetickets*, lasciando solo i candidati buoni chiamati *truechimers*. Esso utilizza principi statistici per trovare l'insieme di truechimers più accurato possibile. L'algoritmo calcola quindi l'offset finale del clock attraverso una media statistica dei clock dei truechimers sopravvissuti.
Associato al processo appena descritto, chiamato *clock discipline*, si ha poi il processo di *clock-adjust*, ripetuto ciclicamente per iniettare il valore di offset e mantenere così la frequenza tra i nodi costante. Un client invia messaggi ad ogni server con un intervallo pari a 2^τ secondi. In NTPv4, τ ha un valore che può variare da 4(16 s) fino a 17(36 h).

2.4.3 PTP

Una delle alternative ad NTP è PTP. PTP è definita dallo standard [IEEE 1588] , pubblicato nel Novembre del 2002. Una revisione di PTP con caratteristiche aggiuntive e prestazioni migliorate è stata pubblicata nel 2008 conosciuta come PTP versione 2 o IEEE 1588-2008. Quanto di seguito scritto si riferisce alla seconda versione.
PTP ed NTP sono simili sotto diversi aspetti. Entrambi sono protocolli packet-based ed entrambi propagano il timestamp partendo da un dispositivo di riferimento verso altri dispositivi appartenenti alla rete. Inoltre anche il modo di sincronizzare i dispositivi è simile ed entrambi supportano apparati con diversi caratteristiche in merito a precisione, risoluzione e stabilità.
Quando viene utilizzato un apparato hardware *PTP-capabale*, i dispositivi che utilizzano PTP su una LAN possono sincronizzare i loro clock con una

precisione di decine di nanosecondi tra di loro. Invece senza il supporto di timestamping hardware, le implementazioni PTP possono ottenere precisioni dell'ordine del millisoecondo [RG] .
Va comunque detto che NTP rimane a tutt'oggi la tecnologia di sincronizzazione più diffusa.

PTP considera due tipi di clock, chiamati master e slave. Un clock in un dispositivo foglia è noto come clock ordinario, un clock in un dispositivo trasmittente, come ad esempio uno switch ethernet, è noto come boundary clock. Un master, idealmente controllato da un clock radio o da un ricevitore GPS, sincronizza gli slave ad esso collegati.

L'algoritmo di sincronizzazione. Una fase di inizializzazione elegge il clock più accurato nella rete al ruolo di clock di sistema attraverso l'algoritmo Best Master Clock (BMC).
La determinazione dello stato (master/slave) di un nodo è il compito svolto dell'algoritmo Best Master Clock. BMC compara le proprietà di accuratezza, stratum, varianza, deriva, etc. dei clock presenti nella rete PTP e determina gli stati per tutte le porte locali degli elementi PTP. Ogni elemento PTP fa girare l'algoritmo BMC localmente al fine di impostare lo stato delle sue porte. BMC trasferisce le proprietà del master clock agli slave ciclicamente nei messaggi di sincronizzazione. Il vantaggio di questo algoritmo è che i nodi non devono negoziare questi stati ma possono calcolarli individualmente. Questo aspetto garantisce che la rete PTP possa essere configurata in modo automatico all'interno di una struttura ad albero partendo dal miglior clock disponibile.

La successiva fase di sincronizzazione è quindi divisa in due parti:

- Nella prima fase il master e lo slave correggono le rispettive differenze di tempo con la misura dell'offset. Per ottenere tale correzione, il master invia un messaggio di sincronizzazione, il *SYNC message*, allo slave ad intervalli ciclici (ogni due secondi nelle impostazioni di base). Questo messaggio contiene l'ora corrente del master clock. Questa informazione, intrinsecamente riporta già un grado di imprecisione che si tenta di ridurre andando a prendere il timestamp il più vicino possibile all'interfaccia di uscita del pacchetto. La stima di tale imprecisione è inviata allo slave attraverso un secondo messaggio, il messaggio *follow-up*. Una volta ricevuto il messaggio follow-up, lo slave calcola il valore di correzione. Questa operazione è ripetuta al fine di ottenere un livello

di sincronismo adatto alle specifiche esigenze. Tuttavia, sarà sempre presente un certo grado di sfasamento, che dipenderà da molti fattori quali, ad esempio, l'architettura di rete, il tipo di apparati utilizzati, l'utilizzo di tecniche di timestamping hardware piuttosto che software, etc.

- La seconda fase del processo di sincronizzazione determina il delay time (il run time sulla rete), tra lo slave ed il master. Questo è determinato attraverso due messaggi, un primo messaggio di *Delay Request* seguito dal relativo *Delay Response*. Il *delay request packet* determina l'istante tempo di trasmissione del messaggio. Il master genera un timestamp alla ricezione del pacchetto e restituisce il tempo di ricezione allo slave nel pacchetto di Delay Response. Dal valore del timestamp di trasmissione locale ed il valore del timestamp di ricezione del master, lo slave determina il valore del delay time. La misura del ritardo è iterata ad intervalli irregolari (4 - 60 secondi) maggiori degli intervalli delle misure di offset. Questo approccio alleggerisce il carico sia sulla rete che sui dispositivi terminali. é utile notare come tale ritardo sia un ritardo simmetrico ($T\_Delay\ Send = T\_Delay\ Return = TDelay$) tra il master e lo slave, caratteristica molto importante per le misure di accuratezza e precisione.

2.5 Conclusioni

Il percorso seguito nel capitolo di descrizione dello stato dell'arte ha preso in considerazione alcuni degli aspetti più rilevanti nel dominio del monitoraggio, cattura e processamento del traffico all'interno di reti a commutazione di pacchetto. Quanto discusso permette di comprendere da un lato l'elevato livello di complessità che caratterizza l'ambiente in questione, dall'altro mostra una varietà di possibili tecnologie e strategie volte ad ottimizzare le prestazioni del sistema in esame.
Infine sono stati presi in considerazione due differenti protocolli, NTP e PTP, utilizzati per ottenere un certo grado di sincronizzazione. L'obiettivo non è quello di entrare nel dettaglio di tali protocolli quanto piuttosto comprendere che una certa imprecisione è naturalmente intrinseca a tali tecnologie. Solo mediante soluzioni basate su GPS e attraverso il supporto dell'hardware è possibile ottenere prestazioni più elevate.

Capitolo 3

IP_SLA

3.1 Introduzione

Nel presente capitolo vengono da prima presentati alcuni aspetti di carattere generale legati al protocollo Cisco Internet Protocol Service Level Agreement (IP SLA) utili a comprenderne il funzionamento ed il contesto operativo, successivamente ne vengono quindi presentatati i punti chiave definiti nella rfc 6812 [IPSLA] in merito sia all'algoritmo che sta alla base del protocollo sia sul formato dei messaggi generati durante le sessioni IP SLA. In particolare la seconda raccoglie le informazioni utili a comprendere alcune delle scelte implementative adottate o meno nel protocollo PNPM SLA che verrà successivamente presentato come strumento alternativo nelle operazioni di monitoraggio delle prestazioni in reti a commutazione di pacchetto.

Prima di procedere è necessario rispondere alla seguente domanda: perchè sono necessari strumenti per il monitoraggio delle prestazioni di rete? Naturalmente esiste una varietà di ragioni ma la risposta più semplice può essere la seguente: nel mercato dei servizi di rete vengono stipulati accordi di fornitura di un certo servizio tipicamente in cambio di un corrispettivo economico. Tanto il fornitore di servizi quanto il cliente sono interessati a conoscere con precisione il reale livello del servizio offerto. Idealmente nessun cliente vuole pagare per un servizio peggiore di quello concordato e nessun fornitore vorrebbe/dovrebbe offrire un servizio differente da quello previsto, teoricamente ne peggiore ne migliore.
L'aspetto maggiormente sensibile in questo scenario è dovuto alla complessità nell'ottenere strumenti in grado di fornire analisi e risultati con il livello di precisione desiderata. All'interno di questo scenario tecnologie quali IP SLA di Cisco diventano fondamentali. Il protocollo PNPM SLA rappresenta una

proposta sperimentale alternativa, ma gli obiettivi dei due protocolli sono i medesimi.

Alcuni esempi di servizio in cui è fondamentale disporre di strumenti di analisi degli Service Level Agreement (SLA) possono essere:

- Applicazioni *Business-Critical*(Entreprise Relationship Management (ERP), Customer Relationship Management (CRM), Material Requirement Planning (MRP), etc.);
- Applicazioni per traffico dati (Server Web);
- Applicazioni per audio e video conferenza (Skype).

Vengono ora richiamati i concetti delle principali metriche utilizzate per monitorare le prestazioni di una rete a commutazione di pacchetto. Anche se questi aspetti potrebbero essere trattati tanto nei successivi capitoli 4 e 5, quanto nel capitolo precedente dedicato allo stato dell'arte, si è scelto di parlarne in questa sezione in quanto è il primo punto in qui la rete stessa assume un ruolo centrale. Nel capitolo precedente l'attenzione era infatti focalizzata prevalentemente su aspetti interni ai sistemi OTS, mentre nei capitoli successivi si entrerà nello specifico di alcuni concetti, quali appunto le principali metriche utilizzate per valutare le prestazioni di una rete, che è bene siano chiari.

- Il **Network Delay**, o ritardo di rete, descrive il tempo speso da un pacchetto nell'attraversamento della rete dalla sorgente fino alla destinazione. Il valore di tale ritardo è il risultato della somma di un certo numero di fattori, che possono però essere riassunti nella somma di due valori; il ritardo di processamento, inteso in generale come il tempo speso dal pacchetto all'interno di un apparato (ricezione, elaborazione e trasmissione), ed il ritardo di propagazione all'interno della rete. In un percorso di routing simmetrico, dove tutti i pacchetti attraversano gli stessi *hop* in entrambe le direzioni, il ritardo di rete in ciascuna direzione può assumere un valore molto simile. Tuttavia, anche in presenza di percorsi perfettamente simmetrici si possono avere ritardi variabili nelle due direzioni dovuti principalmente al differente tempo speso all'interno dei buffer. In generale comunque la misura di *round-trip delay* è appropriata nei percorsi simmetrici.
  Viceversa la misura del ritardo di *round-trip* presenta alcune limitazioni nelle reti dove viene applicato un routing asimmetrico. In questo caso

infatti il ritardo accumulato dalla sorgente verso la destinazione non è detto assuma un valore simile a quello che assumerebbe nel percorso inverso. Per questa ragione all'interno di percorsi asimmetrici prevalgono misure di tipo *one-way*;

- Il **Jitter** misura la variazione del ritardo di ricezione tra pacchetti che sono stati trasmessi in sequenza ad una certa frequenza. Assume particolare importanza soprattutto in applicazioni multimediali ed interattive (esempio: video conferenze). In un sistema ideale se l'intervallo di trasmissione dei pacchetti da parte di una sorgente è costante, ad esempio 1 pacchetto ogni 20 ms, anche l'intervallo di ricezione dovrebbe essere costante ed assumere lo stesso valore. In circostanze realistiche, a causa dei ritardi accumulati nella rete, e spesso a causa dell'intervallo di trasmissione non perfetto, i pacchetti possono raggiungere la destinazione ad intervalli sia maggiori che inferiori. Spesso, soprattutto nel contesto di applicazioni multimediali, si utilizzano dei buffer dedicati in ricezione al fine di rendere il valore del Jitter il più costante possibile.

- Il fenomeno della **Packet Loss**, o perdita di pacchetto, si verifica principalmente quando un pacchetto anzichè venire processato e, nel caso di un apparato di commutazione (router o switch) instradato (a meno che non sia esso stesso il destinatario finale del pacchetto), viene scartato. Nel capitolo dedicato allo stato dell'arte è già stato discusso il problema della perdita di pacchetto con riferimento agli apparati terminali. In particolare sono state considerate le principali cause che, all'interno di un sistema OTS, possono portare la perdita di pacchetto, e quasi tutte sono dovute all'impossibilitò di memorizzare il pacchetto (o un relativo puntatore) all'interno di un buffer. Questa incapacità è dovuta ad un tasso di ricezione troppo elevato, ad una capacità elaborativa non sufficiente o ad una somma delle due circostanze. Nel caso di apparati quali router o switch il problema si può riassumere nell'impossibilità di instradare il pacchetto verso il così detto *next-hop*. Un'ulteriore causa che può portare alla perdita di pacchetti è dovuta a politiche di Quality of Service (QoS) e di sicurezza, infatti apparati di rete potrebbero intenzionalmente scartare un pacchetto con specifiche caratteristiche.

- L'accuratezza delle misure è un altro aspetto estremamente importante che è già stato discusso nella sezione dedicata alla sincronizzazione 2.4. Vengono brevemente ricordati i seguenti concetti; L'accuratezza delle misure è affetta dal ritardo di processamento, che comprende la latenza nei buffer, dall'accuratezza del clock di sistema dei dispositivi di misura

e dal metodo utilizzato per ottenere la sincronizzazione tra i vari apparati. I protocolli NTP, PTP e GPS sono comunemente utilizzati per la sincronizzazione dei clock. Abbiamo visto come in generale all'interno di reti LAN le prestazioni offerte da NTP siano il più delle volte sufficienti, viceversa l'utilizzo del GPS diviene necessario per la sincronizzazione nelle Wide Area Network (WAN) o comunque in specifici contesti che richiedono sincronizzazioni dell'ordine dei nanosecondi.
Inoltre va sottolineato come errori di alcuni millisecondi, quali tipicamente quelli generati da sincronizzazione *NTP-based*, affliggono principalmente la precisione delle misure *one-way* dove l'accuratezza della sincronizzazione dei clock assume un ruolo più importante. Invece, metriche quali il *round-trip time* ed il *Jitter* sono meno sensibili. Questa è la principale ragione per cui, se si hanno rispettivamente: elevati requisiti di precisione, la possibilità di utilizzare percorsi simmetrici e l'impossibilità di utilizzare un sistema basato su GPS, spesso si preferisce adottare un approccio di misura *two-way* piuttosto che *one-way*.

3.2 Concetti chiave di IP SLA

Il protocollo IP SLA è configurato nella maggior parte dei router e degli switch Cisco e viene utilizzato per misurare parametri del livello di servizio quali la latenza di rete, la variazione del ritardo inter-pacchetto e la perdita di pacchetto. I differenti tipi di misura realizzabili possono essere sia semplici test di connettività (ping like), fornendo quindi misure di latenza *one-way* e *round-trip*, sia test per fornire un più ricco insieme di statistiche, che includono il *Jitter* e la *packet loss*.

Architettura e metriche E' presa in considerazione una rete che utilizzi apparati proprietari Cisco. La figura 3.1 mostra come operazioni Cisco IP SLA siano utilizzabili per il monitoraggio dei server e per il monitoraggio di singoli router [IPSLAarchitecture] . IP SLA non adotta un'architettura di rete standard quale può essere ad esempio la struttura gerarchica utilizzata dal protocollo NTP. Il concetto alla base dell'architettura di rete IP SLA è che questa sia costituita dalla somma degli apparati sui quali IP SLA è configurato. Su tale rete è poi possibile realizzare le operazioni necessarie al monitoraggio delle prestazioni.

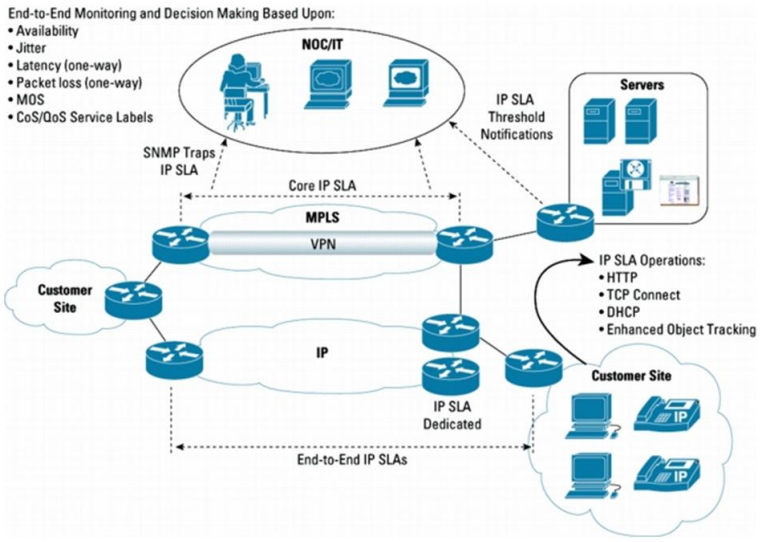


Figura 3.1: Architettura IP SLA.

Il protocollo IP SLA basa le proprie misure sulla generazione di traffico artificiale, quindi mediante test attivi. Prevede l'utilizzo di due componenti o attori, *entrambi* implementati nei dispositivi Cisco:

- una sorgente, *Sender*, la quale sia in grado di generare ed analizzare il traffico IP SLA;
- un ricevitore, *Responder*, il quale è utilizzato opzionalmente per aumentare l'accuratezza ed il dettaglio delle misure. Il Responder può di eliminare il proprio tempo di processamento del pacchetto attraverso l'inserimento del timestamp relativo all'istante di trasmissione del pacchetto. Questo aspetto può essere utile la dove si sia interessati a conoscere l'effettivo tempo di round-trip[1] del pacchetto indipendentemente dal carico elaborativo a cui è sottoposto l'apparto ricevente. Naturalmente tale proprietà è ottenibile grazie alla capacità del protocollo di memorizzare in opportuni campi dei pacchetti di misura i valori dei timestamps sia in ricezione che in trasmissione.

IP SLA è quindi sostanzialmente una soluzione complementare a tecnologie di accounting di rete quali ad esempio NetFlow [2]. Opera al di spora di un'infrastruttura esistente collezionando dati in tempo reale ed analizzando

[1] A valle degli aspetti presentati nel capitolo dedicato allo stato dell'arte è immediato intuire che il concetto di *effettivo tempo di round-trip* è comunque relativo.

[2] NetFlow è oggi integrato nella maggior parte degli switch e dei router di classe en-

il comportamento della rete e degli apparati che operano su di essa.

Di seguito vengono riportate le metriche utilizzabili mediante IP SLA. Durante una sessione di analisi è possibile valutare più metriche contemporaneamente;

- *Delay*, sia *round-trip* sia *one-way*;
- *Jitter*;
- *Packet Loss*;
- *Packet Sequencing*. Il formato dei pacchetti IP SLA prevede l'utilizzo di un campo *Sequence Number*, mediante il quale è possibile identificare i singoli pacchetti e rilevare eventuali ricezioni fuori ordine oltre ad eventuali perdite;
- Rilevamento di eventuali pacchetti corrotti;
- *Voice quality scores*, o Mean Opinion Score (MOS);
- *Path*, ovvero permette di conoscere il percorso seguito da un pacchetto dalla sorgente fino alla destinazione;
- Il tempo di download da server File Transfer Protocol (FTP) o da siti web mediante il protocollo Hypertext Tranfer Protocol (HTTP).

Per quanto riguarda invece le differenti operazioni realizzabili mediante IP SLA è possibile definire una classificazione basata sul tipo di servizio che si vuole analizzare:

- Operazioni *ICMP-based* (ping-like);
- Operazioni *UDP-based operations* (Jitter, DNS, DHCP,etc.);
- Operazioni *TCP-based* (Connsessioni TCP, FTP, HTTP,etc.);
- Operazioni *Layer 2* (Frame Relay, ATM, MPLS);
- Operazioni *VoIP-related* (Jitter, Monitoraggio del ritardo di registrazione presso il Gatekeeper, Setup delle chiamata).

terprise e rappresenta una tecnologia primaria per l'accounting del *Network* per la rilevazione delle anomalie della rete. NetFlow risponde essenzialmente alle seguenti domande sul traffico della rete: chi, cosa, quando, dove e come.

Esempio di configurazione di IP SLA su un router Cisco. A titolo di esempio viene riportato lo schema del pattern utilizzato quando si vuole configurare un test IP SLA su un router Cisco.

```
router#config t
router(config)#ip sla <IP SLA ID>
router(config-ip-sla)#<IP SLA command>
router(config-ip-sla-echo)#exit
router(config)#ip sla schedule <IP SLA ID> start <start-time>
life <duration>
router(config)#exit
router#
```

Prima di tutto occorre impostare un identificativo del test, *IP SLA ID*, il quale deve essere un numero unico e può assumere un valore compreso tra 1 e 2147483647. Dopo di che va specificato un comando, *IP SLA command*, attraverso il quale viene specificato il tipo di test che si desidera effettuare. Infine è necessario schedulare il test, ovvero definire l'istante di inizio, *start-time*, e la durata del test,*duration*.

Si fa notare come in IP SLA siano presenti i seguenti due comandi utilizzati per ottenere un certo grado di precisione nella sincronizzazione tra gli apparti. Sostanzialmente permettono di impostare soglie di tolleranza oltre le quali vengono generati degli errori ed il test non prosegue. Tipicamente IP SLA sfrutta il protocollo NTP come strumento di acquisizione del tempo e del sincronismo da parte degli apparati coinvolti.

- **Clock-tolerance.** Il comando *clock-tolerance* imposta l'errore di sincronizzazione massimo tra i clock di sistema. Questo comando imposta una soglia oltre la quale verrà generato un errore. Tale soglia può assumere un valore assoluto o un valore percentuale;

- **Precision** Per impostare il livello di precisione al quale misurare le statistiche per le operazioni di SLA, è possibile utilizzare il comando *precision*.

L'esempio seguente mostra come impostare una precisione del microsecondo, come configurare una tolleranza massima nell'offset di sincronizzazione attraverso il protocollo NTP pari al 10% e come impostare la priorità di pacchetto al valore *high* per operazioni *IP SLA UDP Jitter* [IPSLAprecision] .

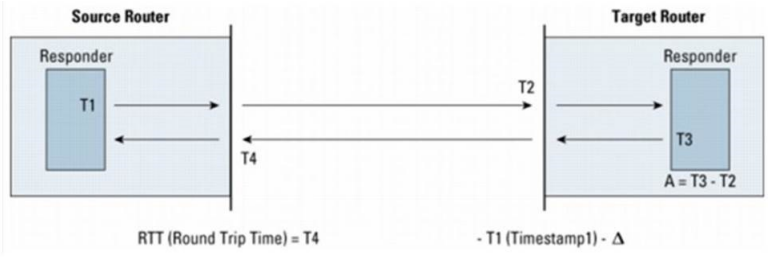


Figura 3.2: Cisco IOS IP SLAs Responder.

```
IP SLA Configuration mode
ip sla 1
udp-jitter 192.168.202.169 9006
precision microseconds
clock-tolerance ntp oneway percent 10
probe-packet priority high
frequency 300
```

Accuratezza Lato responder è possibile calcolare due timestamp, il primo quando il pacchetto raggiunge la destinazione, idealmente quando viene intercettato dall'interfaccia di rete ed il secondo quando il pacchetto viene restituito dalla destinazione verso la sorgente. Questa operazione di timestamping è molto importante in quanto i router e gli switch assegnano maggiore priorità al traffico destinato ad altre locazioni prima che ai pacchetti destinati al proprio indirizzo IP. Quindi se un apparato è sottoposto ad un elevato carico elaborativo i ping-test possono rilevare un tempo di risposta particolarmente inaccurato. Pertanto l'attività di timestamping svolta sul responder con IP SLA permette di eliminare il tempo di attesa e processamento del pacchetto, ottenendo una rappresentazione del tempo di risposta più accurato.

Come mostrato in figura 3.2 Al fine di ottenere il tempo di round-trip vengono utilizzati quattro valori di timestamp. Al target router, con la funzione di responder abilitata, il Timestamp 2 è sottratto al Timestamp 4 per produrre il tempo speso nel processamento del pacchetto. Questo è rappresntato dal delta (δ). Il delta è poi sottartto dal tempo di round-trip complessivo. Notare anche che lo stesso principio è applicato sul router di sorgente dove il timestamp in ingresso 4 è anche preso al livello interrupt per permettere una migliore accuratezza.

Un vantaggio aggiuntivo di usare i due timestamp al target router è la capacità di ottenere il ritardo one-way, il jitter, e la packet loss direzionale. Queste statistiche sono ritiche perchè un grosso problema dell'ambiete di rete è l'asincronismo. Tuttavia, per catturare misure di ritardo one-way, è necessaria la configurazione del protocollo NTP sia sul router sorgente che sul router destinazione. Entrambi hanno bisogno di sincronizzarsi con uno stesso clock sorgente.

Impatto sulle prestazioni I routers e gli switch non sono progettati primariamente per operazioni di monitoraggio della rete quanto invece sono sviluppati per operazioni di commutazione dei pacchetti/frame. Il formato dei pacchetti generati da IP SLA, aspetto che verrà discusso nella sezione seguente, ha una dimensione non trascurabile a causa dell'elevato numero di campi utilizzati per ottenere i risultati fin'ora presentati. IP SLA memorizza ogni risultato localmente, quindi in pratica il carico elaborativo dovuto a questi fattoti aumenta con l'aumentare del numero di operazioni e test che si desidera realizzare.

La figura 3.3 mostra la relazione tra il carico di CPU (in %) ed il numero totale di operazioni di calcolo del jitter richieste per secondo. Le tre linee rappresentano rispettivamente 500, 1000, e 2000 operazioni. I risultati del test si riferiscono a prove realizzate su un apparato Cisco 7200VXR/NPE-225 con IOS 12.2(8)T5. Nel test ogni operazione, *Jitter UDP*, consiste nella trasmissione di 10 pacchetti da 64-bytes ciascuno, con un intervallo di trasmissione pari a 20 ms [BCRW] . Come si può osservare dalla figura il carico sulla CPU cresce notevolmente all'aumentare del numero di operazioni richieste fino ad arrivare ad oltre il 70% nel caso di test eseguiti su 2000 probes a 60 operazioni al secondo. Una delle sfide affrontate dal protocollo PNPM SLA è appunto quella di ridurre il carico elaborativo sugli apparati che svolgono attività di monitoraggio riducendo la complessità legata all'operazione di processamento dei pacchetti di test attraverso un formato più semplice del pacchetto stesso.

Considerazioni sulla sicurezza in IP SLA. Vengono infine presentati i due principali concetti legati alla sicurezza in IP SLA. Va tuttavia precisato che i seguenti aspetti non valgono solo in riferimento al protocollo IP SLA, ma hanno una valenza ben più generale.

- *L'esposizione dei parametri e degli apparati di misura dovrebbe rimanere segreta.* L'osservazione dei dati risultanti dai test IP SLA potrebbe

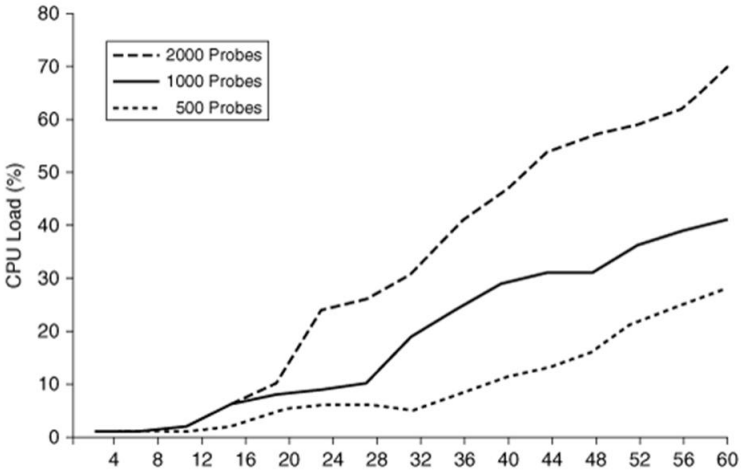


Figura 3.3: Numero di operazioni IP SLA per secondo.

fornire ad un attaccante informazioni circa il percorso seguito dai pacchetti nella rete o informazioni sugli end-point coinvolti nelle comunicazioni. Se un hacker conoscesse dove vengono effettuate le misure nella rete e quali metriche vengono monitorate, potrebbe attaccare contro tali apparati o più semplicemente diffondere informazioni e risultati che l'operatore può non volere che diventino pubbliche. In alternativa, un cliente dell'operatore potrebbero tentare di contraffare il traffico in modo da falsificare gli SLA misurati;

- *Protezione da specifici attacchi contro il Responder.* Uno scenario di attacco differente potrebbe riguardare la trasmissione di richieste di misura ad un *Responder* utilizzando un indirizzo IP di sorgente falso. Questo attacco, noto in generale come Spoofing IP, ha come obiettivo quello di fare attaccare la vittima, ovvero la macchina avente come indirizzo IP quello utilizzato dall'attaccante per avviare la sessione IP SLA, proprio dal Responder.

Per gestire il primo tipo di attacco è necessario, dove possibile, mantenere l'infrastruttura di rete segreta. Dove invece questo non sia fattibile, ad esempio il Responder IP SLA deve rendere nota la porta per i messaggi di controllo (porta UDP 1967), vanno utilizzate altre tecniche di protezione quali ad esempio sistemi di Intrusion Detection System (IDS).
Invece comunicazioni non autorizzate con il Responder possono essere risolte

utilizzando la tecnica di autenticazione MD5 [3], con la quale la comunicazione tra la sorgente IP SLA ed il Responder viene appunto autenticata. Questo significa che le richieste sono accettate solo da parte di Sender autenticati. Notare che MD5 non si occupa della cifratura dei pacchetti IP SLA, ma solo dell'autenticazione del mittente. In IP SLA la cifratura è considerata una *best practice* ma non è obbligatoria.

3.3 Il protocollo

Dopo aver presentato alcune delle principali caratteristiche del protocollo IP SLA, vengono discusse con maggiore dettaglio sia la logica di funzionamento dell'algoritmo che sta alla base del protocollo sia il formato dei pacchetti generati durante la sua attività.

IP SLA consiste di due fase separate e sequenziali: la fase di controllo e la successiva fase di misura. Ciascuna di esse comprende informazioni scambiate tra un elemento di rete che agisce da *Sender* ed un elemento designato al ruolo di *Responder*.

- La fase di controllo costituisce la base del protocollo ed è caratterizzata dallo scambio di alcuni messaggi iniziali. Tali messaggi permettono di stabilire l'identità del sender e fornire le informazioni utilizzate nella fase di misura, ovvero durante i test veri e propri. Il sender trasmette un primo messaggio di *Control-Request* che viene confermato attraverso un messaggio di *Control-Response* da parte del destinatario. Il Control-Request può essere trasmesso più volte se non viene ricevuto il relativo acknowledge. Il numero di volte che il messaggio è ritrasmesso è configurabile dal sender.

- La fase di misura segue la fase di controllo, ed è composta da una sequenza di coppie di messaggi, il *Measurement-Request* e il *Measurement-Response* che rappresentano i veri e propri *pacchetti di test*. Ogni messaggio di Measurement-Request è confermato dal responder con il relativo messaggio di Measurement-Response. Il numero e la frequenza con la quale i messaggi sono inviati viene configurata sull'elemento sender.

[3] L'acronimo MD5 (Message Digest algorithm 5) indica un algoritmo crittografico di hashing realizzato da Ronald Rivest nel 1991 e standardizzato con la RFC 1321. Una delle sue principali applicazioni è appunto legata all'autenticazione della firma digitale.

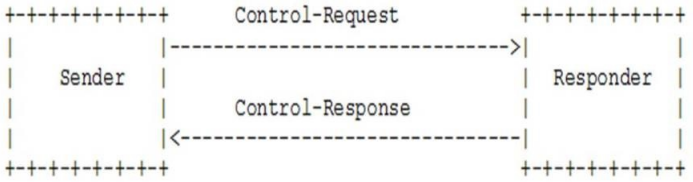


Figura 3.4: Esempio di scambio Control-Request, Control-Response.

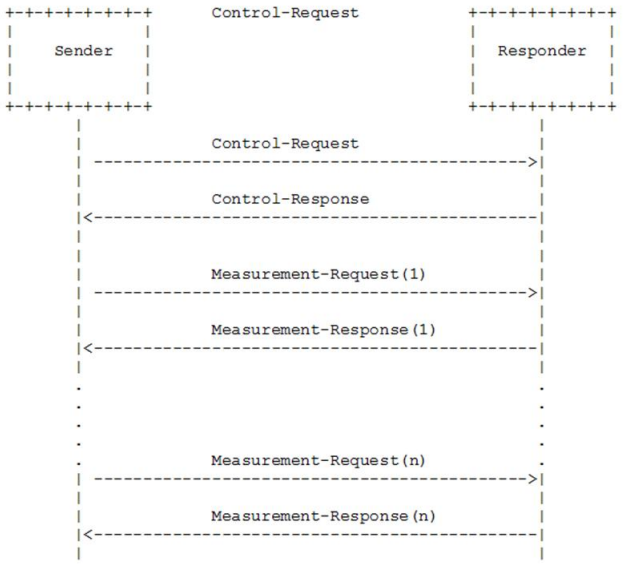


Figura 3.5: Esempio di una sessione IP SLA.

La figura 3.5 riporta schematicamente quanto sopra descritto. Si possono osservare le due distinte fasi del protocollo. La fase di controllo viene svolta un'unica volta all'inizio di ogni nuova sessione IP SLA mentre la fase di monitoraggio ha una durata che dipende dal tipo dei test che si desidera realizzare.

3.3.1 Fase di controllo iniziale

La fase di controllo inizia con un messaggio di Control-Request inviato dal sender e termina con un messaggio di Control-Response trasmesso dal responder. Il messaggio di Control-Request, che viene inviato alla porta UDP 1167 del responder, serve per richiedere l'apertura di una nuova porta UDP da utilizzare nella successiva fase di misura. Inoltre permette di indicare per quanto tempo tale porta deve rimanere aperta e quindi in pratica permette di

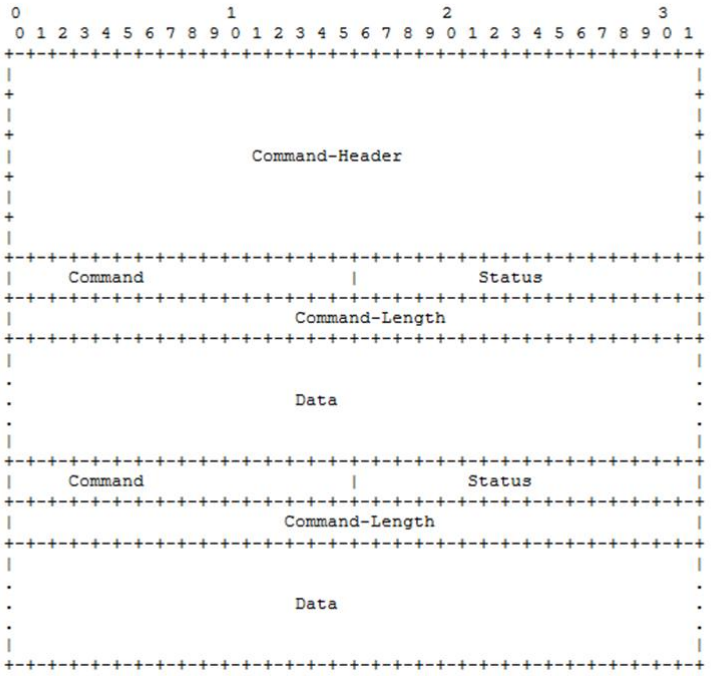


Figura 3.6: Esempio di due sezioni CSLD.

specificare la durata del test. Il responder restituisce un messaggio di Control-Response con il campo *Status*, presente nella sezione *Command-Header* del messaggio, propriamente settato. Il campo Status indica "Success" se l'identità del sender è verificata e se è stato possibile aprire una porta UDP da dedicare alla fase dei test. In tutti gli altri casi assume un valore diverso da zero, vale a dire che la richiesta non è stata approvata.

Il messaggio *Control-Request*

Il messaggio di Control-Request consiste in un Command-Header seguito da una o più sezioni Command, Status, Length, Data (CSLD). In IP SLA ci dovrebbero essere sempre almeno due sezioni CSLD, una che rappresenta la sezione di autenticazione e l'altra che riporta le informazioni per la successiva fase di misura.

Il Command-Header è la prima sezione di un messaggio Control-Request e riporta i campi mostrati in figura 3.7.

- Il campo *Sequence Number* viene utilizzato per mappare una richiesta alla relativa risposta;

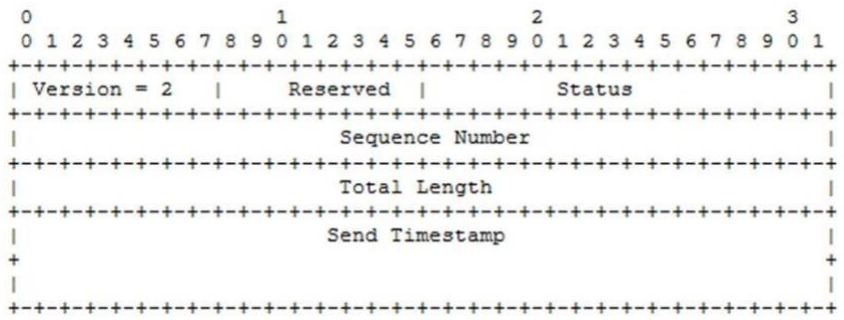


Figura 3.7: Command-Header IP SLA.

- Il campo *Status* indica l'esito del messaggio;
- Il campo *Total Length* riporta il numero di bytes del messaggio;
- Il campo *Send Timestamp* assume il valore dell'istante di tempo in cui il messaggio è stato sottomesso per la trasmissione. Questo campo può essere utilizzato allo scopo di prevenire attacchi di tipo replay. Dovrebbe quindi essere aggiornato quando viene trasmessa la relativa risposta.
- **CSLD di autenticazione.** Il primo CSLD è quello relativo all'autenticazione del sender, il quale inserisce nel messaggio la prova della conoscenza di un certo segreto condiviso attraverso l'utilizzo dell'algoritmo crittografico MD5.
  In particolare i seguenti campi sono quelli utilizzati a tale scopo:
  - *mode*. Indica il tipo di autenticazione utilizzata. Può assumere i valori 0,1,2 che rispettivamente indicano, No Autenticazione, SHA246, HMAC-SHA256.
  - *Key ID*. Indica l'indice del segreto condiviso che deve essere usato per l'autenticazione del messaggio di Control-Request.
  - *Random Number*. Questo numero serve per rendere più difficile scoprire la chiave privata, proprio perché l'aggiunta di tale numero annulla l'effetto di un pre-calcolo della chiave. Permette quindi di prevenire attacchi dizionario.
  - *Message Authentication Digest*. Contiene il digest calcolato sull'intero pacchetto.

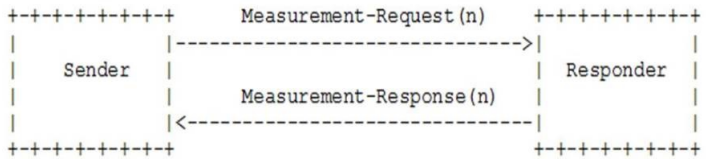


Figura 3.8: Schema di un generico scambio di frame di misura.

- **CSLD di misura.** Questa seconda sezione CSLD indica il tipo di misura che si vuole realizzare. Definisce quindi gli indirizzi Internet Protocol (IP) e le porte UDP che verranno utilizzati, la durata per la fase di misura, ed alcuni campi aggiuntivi quali *Role*, per indicare il ruolo dell'end-point che riceve il messaggio di controllo, sender o responder, ed il campo *Session Identifier* che riporta un identificativo univoco della sessione.

Il messaggio *Control-Response*

Il messaggio Control-Response, inviato in risposta al messaggio Control-Request, riflette il Command-Header con il campo *Status* aggiornato. Tale valore viene anche aggiornato nelle due sezioni CSLD. Per il resto, il formato del pacchetto è identico al messaggio di Control-Request precedentemente descritto. Naturalmente oltre all'aggiornamento del campo *Status*, vengono anche aggiornati il valore del *Send Timestamp* presente nel Command-Header e nel Message Authentication Digest.

3.3.2 Fase di misura

Nella fase di misura il sender trasmette uno stream di messaggi di misura. I messaggi sono tra loro trasmessi con un certo intervallo temporale configurabile sul sender. Tale intervallo dipende dalle specifiche condizioni in cui vengono svolti i test.

Il formato dei messaggi di misura, mostrato in figura 3.9, è lo stesso in entrambe le direzioni. L'unica differenza è dovuta all'aggiornamento dei campi designati con il prefisso del responder piuttosto che del sender.

I campi indicati in figura 3.9 hanno il seguente significato:

- *Measurement-Type*. Può assumere tre valori: 1-Reserved, 2-Reserved, 3-Misure UDP;

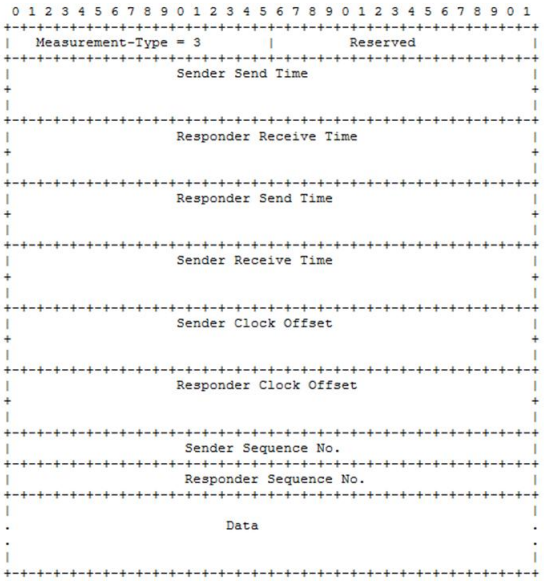


Figura 3.9: Formato di un messaggio di misura.

- *Sender/Responder Send Time*. Indica il timestamp relativo all'istante in cui il messaggio è stato sottomesso per la trasmissione;
- *Sender/Responder Receive Time*. Indica il timestamp relativo all'istante in cui il messaggio è stato ricevuto;
- *Sender/Receiver Clock Offset*. Rappresenta una stima della differenza tra il valore del Sender/Responder Send/Receive Time misurato e l'effettivo istante di trasmissione/Ricezione espresso in secondi e millisecondi. Tuttavia nella rfc 6812, in cui è appunto descritto il protocollo IP SLA, non viene specificato il modo in cui tale valore viene calcolato.
- *Sender/Responder Sequence Number*. Il valore del Sequence Number viene incrementato iterativamente dai due nodi.
- *Data*. Viene utilizzato per raggiungere la dimensione minima del pacchetto pari a 64 bytes.

3.4 Conclusioni

Il protocollo IP SLA di Cisco è implementato all'interno di apparati di rete, quali ad esempio i router e gli switch, e quindi compete come qualunque processo per usufruire delle risorse di sistema; CPU, memoria, Access Control

List (ACL), buffer, schede di rete, etc.
Sono almeno due i fattori che hanno spinto Telecom Italia verso la progettazione di un nuovo protocollo dedicato alle attività di monitoraggio delle prestazioni di rete:

- In IP SLA ogni messaggio di test, *Measurement-Request* e *Measuremnt-Response*, contiene un elevato numero di informazioni, con particolare riferimento all'utilizzo di quattro campi destinati alla gestione del timestamp, due campi dedicati alla stima dell'offset del clock ed un campo dedicato alla numerazione dei pacchetti. L'intenzione del protocollo PNPM SLA è quella di ridurre fortemente non tanto la dimensione del pacchetto, che comunque non può essere inferiore a 64-byte, quanto il numero di campi da processare. Nel capitolo dedicato ai sistemi OTS è stato osservato come la gestione del timestamp non è un aspetto banale, sia dal punto di vista della precisione con cui tale valore può essere ricavato sia per il carico elaborativo introdotto per ottenerlo (system call, context-switch, etc.). In PNPM SLA non vengono utilizzati ne i campi dedicati alla gestione del timestamps ne il campo destinato alla numerazione dei pacchetti;

- Un'ulteriore ragione che ha guidato verso lo sviluppo di PNPM SLA è lagata al fatto che IP SLA è un protocollo proprietario e molti aspetti interni al protocollo non sono chiari. Ad esempio non è chiaro il modo in cui viene calcolato il valore del Sender/Responder Clock Offset, da cui la capacità di dimostrare il reale livello di servizio offerto da parte di un service provider, quale è Telecom, può non essere cosi banale.

- Inoltre, non vanno trascurate ragioni strettamente economiche. Utilizzare un protocollo di proprietà di una terza parte ha un costo non indifferente.

Queste motivazioni, sommate al desiderio di disporre di strumenti sviluppati internamente all'azienda su cui averne il pieno controllo e la massima conoscenza, hanno portato prima allo studio di una metodologia di carattere generale, PNPM per l'analisi delle prestazioni in reti a pacchetto, poi allo sviluppo di un nuovo protocollo, PNPM SLA, a supporto di tale metodologia.

Capitolo 4

PNPM

4.1 Introduzione

PNPM [PNPM] è una metodologia di monitoraggio delle prestazioni in reti a commutazione di pacchetto sviluppata in Telecom Italia.

Vengono di seguito presentati i concetti chiave di PNPM:

- Il metodo è potenzialmente applicabile ad ogni tipo di traffico a pacchetto, sia in *unicast* che in *multicast*. Il suo utilizzo non richiede alcuna estensione ai protocolli già esistenti, aspetto che evita problemi di interoperabilità. È quindi *Technology* e *Vendor Independent*;

- PNPM è stato originariamente progettato per misure passive[1] ma può anche essere utilizzato per misure attive;

- Al fine di rilevare la packet loss, PNPM utilizza un approccio alternativo alla numerazione dei singoli pacchetti. Attraverso il conteggio ed il confronto del numero dei pacchetti trasmessi da un lato ed il numero di quelli ricevuti dall'altro all'interno di un dato intervallo temporale è possibile rilevare eventuali perdite. Questa operazione presenta il vantaggio di non dover leggere/scrivere dal/sul pacchetto un identificativo univoco, con conseguente riduzione del tempo di processamento speso per ogni pacchetto. E' invece necessario che i dispositivi che effettuano le misure siano tra loro sincronizzati, infatti i due nodi devono fare rifer-

[1] Il concetto di misura passiva, che si contrappone a quello di misura attiva, fa riferimento all'attività di misura svolta su traffico reale, quale ad esempio il traffico generato da un'applicazione multimediale, l'up-load di un file, il down-load di una pagina web, etc. Viceversa le misure attive sono svolte su traffico generato artificialmente.

imento esattamente allo stesso set di pacchetti durante le operazioni di aggiornamento dei contatori di ricezione e di trasmissione;

- PNPM prevede la possibilità di effettuare sia misure *end-2-end*, quindi tra i nodi terminali della sessione di monitoraggio, sia misure *hop-by-hop*, quindi tra nodi intermedi, appartenenti al percorso seguito dal flusso dati tra i due nodi terminali;

- Un aspetto primario è la caratterizzazione dello sfasamento tra i clock degli apparati che partecipano alle attività di monitoraggio. Tale caratterizzazione dipende fortemente dalle condizioni operative, in termini di quali apparati sono coinvolti, a quale carico elaborativo sono sottoposti e alla topologia di rete in esame. Gli argomenti legati alla sincronizzazione sono già stati considerati nella sezione dedicata, nella quale è stato sottolineato che attraverso l'utilizzo dei normali protocolli dedicati, quali NTP e PTP, un certo sfasamento sarà sempre presente. Naturalmente l'utilizzo di misure medie ed analisi *round-trip* hanno l'intenzione di mitigarne gli effetti.

- In merito alla tipologia di traffico che si desidera monitorare PNPM applica una strategia di tipo *flow-based*. Questo approccio è conveniente quando si è interessati ad una sola porzione limitata del flusso di traffico. Questo potrebbe essere il caso, per esempio, di applicazioni quali l'Internet Protocol Television (IPTV) o in generale di specifiche applicazioni con elevati requisiti di QoS. Tale approccio richiede la necessità di conoscere nel dettaglio il percorso seguito dai flussi dati.
  La strategia *flow-based* si contrappone a quella *link-based*, la quale prevede di realizzare le misure su tutto il traffico link per link. Il link può essere sia un link fisico che un link logico (per esempio un link Ethernet, VLAN, Multi Protocol Label Switching (MPLS)).

4.2 Il metodo

Questa sezione descrive la tecnica con cui vengono svolte le misure della *packet-loss*, del *delay* e del *jitter*. Tali misure saranno concettualmente ottenute nello stesso modo attraverso il protocollo PNPM_SLA 5 che verrà presentato nel capitolo successivo.

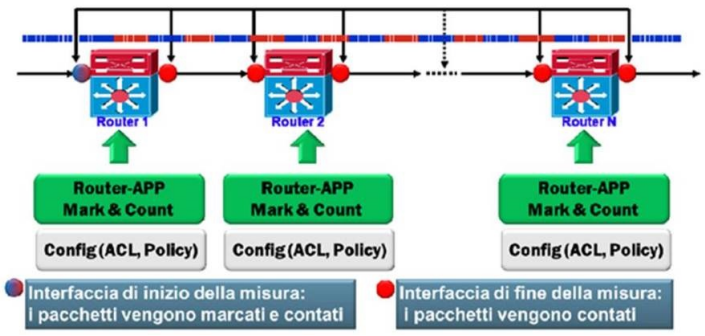


Figura 4.1: Implementazione di PNPM sui router.

4.2.1 Misura della packet loss

L'idea che sta alla base della misura della *packet loss* è quella di dividere virtualmente un flusso di pacchetti appartenenti ad una stessa sessione di traffico dati all'interno di blocchi consecutivi. Ciascun blocco contiene quindi un certo numero di pacchetti marcati in modo identico. In altre parole, si può immaginare che un flusso di dati venga frammentato ed incapsulato in tanti blocchi successivi. In sostanza la *packet loss* è rilevata mediante il confronto tra il numero dei pacchetti trasmessi ed il numero dei pacchetti ricevuti ed immagazzinati all'interno di un certo blocco. Di seguito viene dettagliato il concetto appena esposto. Va inoltre ribadito che PNPM non utilizza alcun numero di sequenza per identificare i pacchetti.

Dal momento che un flusso dati è un'entità continua e non può essere interrotto quando si vuole leggere la variabile contatore nella quale è memorizzato il numero di pacchetti trasmessi/ricevuti, potrebbe essere difficile definire con precisione l'istante in cui leggere il contatore stesso senza che vi sia un errore di sfasamento tra il valore corrente della variabile ed il reale numero di frame trasmessi/ricevuti. Una soluzione per essere certi di leggere il contatore in un istante stabile è appunto quello di dividere virtualmente il flusso dati in blocchi sequenziali. Tutti i pacchetti appartenenti ad uno stesso blocco vengono colorati[2] in uno stesso modo e pacchetti appartenenti a blocchi adiacenti saranno diversamente marcati. La variabile contatore relativa ad un certo blocco, e quindi relativa ad un certo intervallo temporale, verrà letta solo nell'intervallo successivo in modo tale da garantire che il valore in esso contenuto sia ormai stabile.
L'alternanza nella marcatura tra un blocco ed il successivo rappresenta una

[2]Nella metodologia PNPM il termine colorato è da intendersi come sinonimo di marcato o evidenziato.

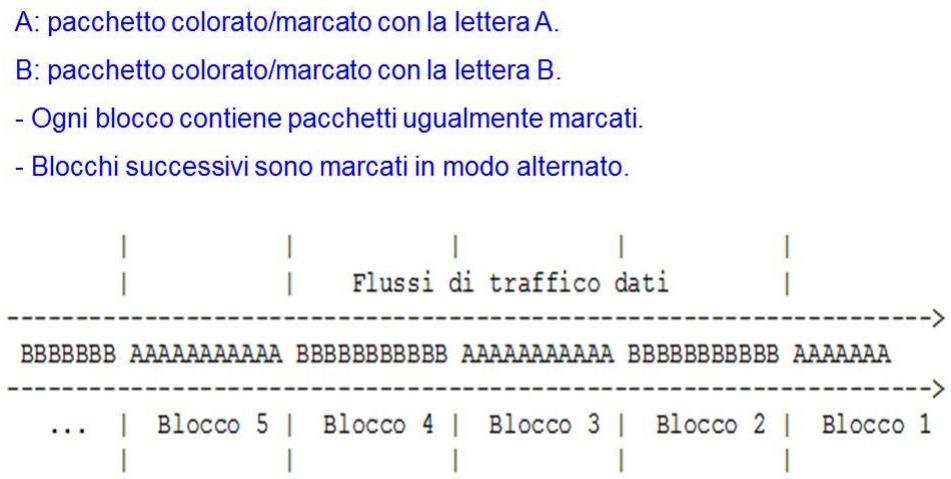


Figura 4.2: Flusso di traffico colorato.

sorta di segnale di auto-sincronizzazione che garantisce la consistenza nelle misure prese dai differenti apparati lungo il percorso di misura. Ogni volta che il colore cambia, il blocco precedente termina e ne inizia uno nuovo. Il numero di pacchetti inseriti in ogni blocco dipende dal criterio scelto, si potrebbe ad esempio considerare di inserire in ciascun blocco un numero fisso di pacchetti oppure assegnare a ciascun blocco una stessa durata temporale. Il vantaggio che si ottiene attraverso questo approccio è dovuto alla garanzia che il valore letto in un certo contatore fa riferimento al numero dei pacchetti ricevuti nel periodo precedente, e quindi ormai concluso. A questo punto confrontando i valori assunti dai contatori di due apparati distinti, relativamente ad uno stesso blocco di frame, è immediato rilevare se c'è stata o meno *packet loss*.

Facendo riferimento alla figura 4.2 supponiamo di voler monitorare la perdita di pacchetti sul collegamento tra due router chiamati *R1* ed *R2*. Secondo il metodo il traffico è colorato alternativamente con due colori/lettere diversi, A e B. Ogni volta che cambia la lettera, la transizione genera una sorta di segnale ad onda quadra, come illustrato in figura 4.3.

La marcatura del traffico può essere fatta da *R1* stesso o da un router che lo precede nel percorso. *R1* avrà bisogno di due contatori per ogni interfaccia di uscita, *C(A)R1* e *C(B)R1*, al fine di contare rispettivamente i pacchetti marcati con la lettera A ed i pacchetti marcati con la lettera B. Durante la trasmissione di frame di colore A, verrà incrementato solo il contatore *C(A)R1*, viceversa quando il traffico è colorato con la lettera B, solo *C(B)R1*

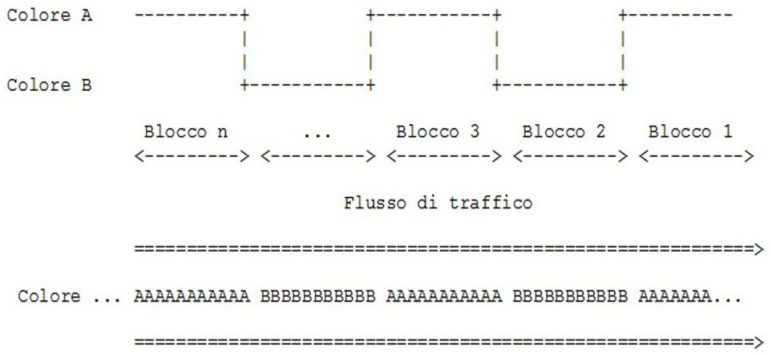


Figura 4.3: Applicazione del metodo PNPM per calcolare la perdita di pacchetti sul link.

viene incrementato. *C(A)R1* e *C(B)R1* possono essere utilizzati come valori di riferimento per determinare la perdita di pacchetti dall'apparato *R1* a qualsiasi altro punto di misurazione lungo il percorso. L'apparato *R2*, allo stesso modo, avrà bisogno di due contatori sulla sua interfaccia di ingresso, *C(A)R2* e *C(B)R2*, per contare rispettivamente i pacchetti ricevuti su tale interfaccia di colore *A* e *B*. Al termine dell'intervallo temporale associato al blocco *A*, sarà possibile confrontare i valori assunti dai contatori *C(A)R1* e *C(A)R2* e poter così verificare l'eventuale perdita di pacchetti all'interno del blocco. L'operazione è ripetibile in modo del tutto analogo per il blocco *B*.

La figura 4.4 riporta i contatori che possono essere utilizzati per calcolare la perdita di pacchetti tra R1 ed R2. La prima colonna elenca la sequenza dei blocchi di traffico, mentre le altre colonne contengono i contatori dei pacchetti colorati con la lettera A ed i pacchetti colorati con la lettera B, per R1 ed R2. In questo esempio si assume che i valori dei contatori vengono resettati ogni volta che un blocco termina e il suo contatore è stato letto: con questa ipotesi, la tabella mostra solo i valori relativi, cioè il numero esatto di pacchetti di ogni colore all'interno di ciascun blocco. Se i valori dei contatori non fossero resettati, la tabella conterrebbe allora i valori cumulativi. I valori relativi potrebbero comunque essere determinati dalla differenza dal valore del blocco precedente dello stesso colore.

Durante un blocco A (blocchi 1, 3 e n +1), tutti i pacchetti sono colorati con la lettera A, pertanto i contatori *C(A)* vengono incrementati del numero di pacchetti ricevuti sulla relativa 'interfaccia, mentre i contatori *C(B)* sono a zero. Viceversa, durante un blocco B (blocchi 2, 4 e n), tutti i pacchetti ricevuti sono di colore B, quindi i contatori *C(A)* sono pari a zero, mentre

Blocco	C(A)R1	C(B)R1	C(A)R2	C(B)R2	Loss
1	375	0	375	0	0
2	0	388	0	388	0
3	382	0	381	0	1
4	0	377	0	374	3
...	...	...	...	...	...
n	0	387	0	387	0
n+1	379	0	377	0	2

Figura 4.4: Valutazione dei contatori per le misure di packet loss.

i contatori *C(B)* vengono incrementati. Quando un blocco termina (a causa del cambio di colore) i contatori relativi arrestano l'incremento ed è quindi possibile leggerli per confrontare i valori misurati sui router R1 e R2 e calcolare così la perdita di pacchetti all'interno di tale blocco.

Ad esempio osservando la tabella in figura 4.4, si può notare che durante il primo blocco (lettera A), i contatori *C(A)R1* e *C(A)R2* hanno lo stesso valore (375), che corrisponde al numero esatto dei pacchetti del primo blocco (quindi non c'è stata perdita). Anche durante il secondo blocco (lettera B) i contatori di R1 e R2 hanno lo stesso valore (388), che corrisponde al numero di pacchetti del secondo blocco (senza perdita). Durante i blocchi tre e quattro, i contatori di R1 e R2 sono diversi, il che significa che alcuni pacchetti sono andati perduti: nell'esempio, un singolo pacchetto (382-381) è stato perso durante il blocco tre e tre pacchetti (377-374) sono stati persi durante il quarto blocco.

Il metodo presentato per la rilevazione della perdita di pacchetti può essere esteso a reti più complesse e potenzialmente a qualsiasi apparato che implementi PNPM.

4.2.2 Analisi della latenza di rete one-way e two-way

Lo stesso principio utilizzato per la misura della *packet loss* può essere applicato alla misura del ritardo *one-way*, l'alternanza dei colori può essere usata come riferimento temporale per calcolare il ritardo. Ogni volta che il colore cambia un apparato di rete può memorizzare il timestamp del primo pacchet-

to contenuto nel nuovo blocco; Tale timestamp può essere confrontato con il timestamp dello stesso pacchetto su un secondo router per calcolare cosi il ritardo di pacchetto. Allo scopo di avere più misure, è possibile memorizzare più timestamps relativi a pacchetti all'interno del blocco corrente.
Nel contesto delle misure unidirezionali, allo scopo di confrontare coerentemente i timestamps collezionati sui differenti apparati, i nodi di rete devono essere sincronizzati, ovvero i clock di sistema utilizzati per le attività di marcatura in trasmissione ed in ricezione devono essere tra di loro sincronizzati. Inoltre, una misura è valida solo se non ci sono state perdite di pacchetto e non ci sono stati pacchetti ricevuti fuori sequenza. Infatti ad esempio il primo pacchetto trasmesso da R1 potrebbe essere differente dal primo pacchetto effettivamente ricevuto da R2 e di conseguenza il calcolo del ritardo di pacchetto non sarebbe corretto.

Ritardo medio

Dal momento che PNPM non prevede la possibilità di utilizzare i *sequence number* per l'identificazione dei pacchetti, il metodo per la misura del ritardo così come è stato esposto fin'ora è sensibile ai pacchetti ricevuti fuori ordine. Inoltre va sottolineato che PNPM prevede che il valore del timestamp relativo all'istante di trasmissione non venga aggiunto al pacchetto in esame. Questo aspetto impedisce al ricevente di confrontare tale valore con quello del timestamp relativo all'istante di ricezione dello stesso pacchetto e quindi non permette di considerare valori istantanei. Inoltre i router, a differenza delle sonde, quali ad esempio dei normali PC, non hanno la possibilità di leggere ed aggiornare i valori dei timestamp, potendo cosi gestire solo la *packet loss*.

Allo scopo di superare questi limiti è stato considerato un differente approccio, in primo luogo mediante il concetto di *valore medio* delle misure del *delay* e del *jitter*, e poi attraverso l'utilizzo di sonde dedicate alle attività di monitoraggio ed analisi delle prestazioni di rete. Questo approccio rende il metodo robusto sia in merito alla ricezione di pacchetti *out-of-order*, ovvero pacchetti il cui ordine di ricezione e differente dall'ordine di trasmissione all'interno di un certo stream , sia alla *packey-loss*.

Ritardo medio one-way e two-way. Viene quindi proposto il concetto di timestamp medio (Average Timestamp (ATS)) relativo ad un certo intervallo di misura come strumento per il calcolo del valore del delay medio (Average Delay (AD)).

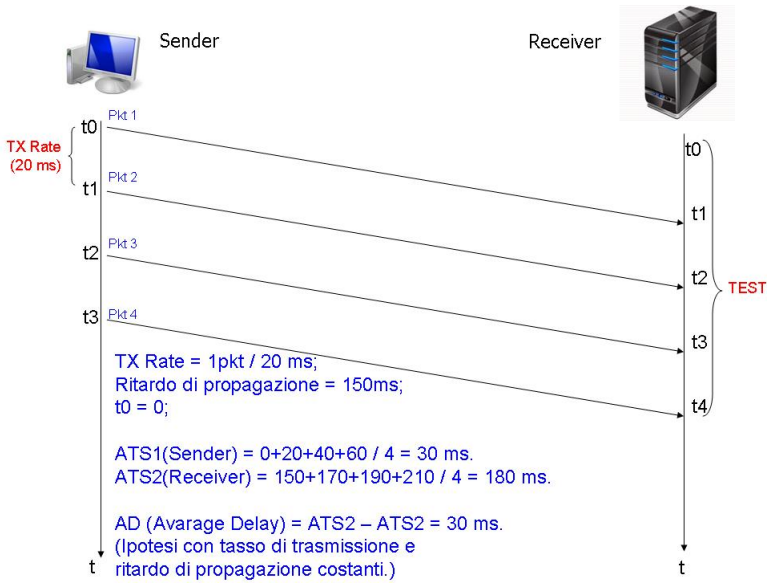


Figura 4.5: Calcolo del ritardo medio tra due sonde.

La figura 4.5 riassume quanto di seguito esposto.

$$\texttt{ATS} = \sum_{k=1}^{N} \frac{TSi}{N};$$

- *TSi* rappresenta il timestamp del pacchetto *i-mo*;
- N è il numero di pacchetti ricevuti in un certo intervallo.

Al termine di ogni intervallo di marcatura è teoricamente possibile calcolare il ritardo medio (AD):

```
AD = ATS2-ATS1;
```

- ATS1 rappresenta il valore del timestmp medio calcolato dalla sonda in trasmissione. Quindi rappresenta il timestamp medio relativo agli istanti di trasmissione dei pacchetti;
- ATS2 rappresenta invece il timestamp medio relativo agli istanti di ricezione dei pacchetti da parte dell'*end-point*.

E' inoltre possibile calcolare il valore del Round Trip Delay medio (RTDM) o Round Trip Time (RTT) "virtuale" per ogni periodo di misura sommando i valori di ritardo medio one-way calcolati localmente:

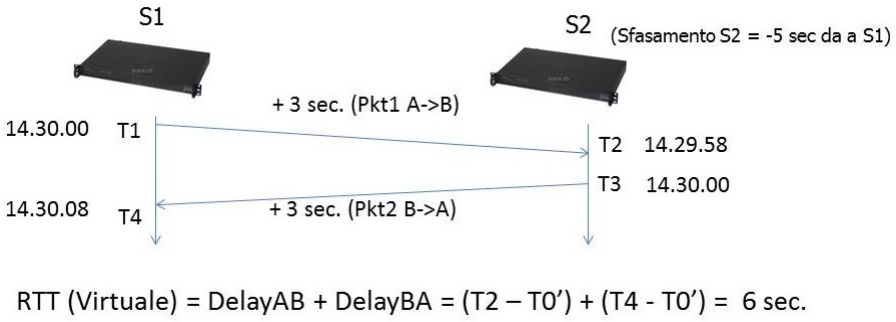


Figura 4.6: Round Trip Time Virtuale.

```
RTDM = DMup + DMdown;
```

Dove *DMup* è il ritardo medio calcolato in una direzione e *DMdown* è il ritardo medio calcolato nell'altra direzione del collegamento tra i 2 punti di misura.
Il RTDM ed il Jitter godono, diversamente del Delay Medio one-way, della proprietà di non essere sensibili allo sfasamento dei *clock* dei due punti di misura.

Al fine di precisare meglio il concetto appena esposto può essere utile osservare la figura 4.6. I dati in essa riportati hanno solo titolo di esempio.

Le due sonde, S1 ed S2, non sono tra loro sincronizzate. Tuttavia, indipendentemente dall'entità di tale sfasamento, attraverso misure di tipo two-way, siano esse reali come nel caso di IP SLA, o virtuali come nel caso di PNPM SLA, è possibile ottenere il valore del Delay Medio two-way (DMTW).

4.2.3 Misure delle variazioni del ritardo

- In modo simile alle misure del ritardo *one-way*, il metodo può anche essere utilizzato per le misure del *jitter one-way* (One Point Jitter medio (OPJM)). L'alternanza della marcatura può essere utilizzata infatti come riferimento temporale per per misurare le variazioni del ritardo;

- Inoltre lo stesso concetto applicato nella misura del ritardo medio può essere applicato per il calcolo della variazione media del ritardo.

$$\texttt{OPJM} = \sum_{k=1}^{N} -1 \frac{(TS(i+1) - TSi)}{N-1} = \frac{(TSn - TS1)}{N-1};$$

Al termine di ogni periodo di test è possibile calcolare il Two Point Jitter medio (TPJM) tra due *end-point* come differenza tra i due OPJM precedentemente calcolati.

```
TPJM = OPJM2-OPJM1;
```

4.2.4 Marcatura dei pacchetti

L'operazione di marcatura è fondamentale allo scopo di creare i blocchi dei pacchetti. A seconda del tipo di analisi che si vuole realizzare è possibile impostare misure *end-to-end* piuttosto che misure *linked-based*.

- Al fine di ottenere misure *end-to-end* è necessario abilitare i punti di misura il più vicino possibile alla sorgente ed alla destinazione. È in generale desiderabile avere una singolo nodo che effettua la colorazione in quanto è più semplice da gestire e non aumenta il rischio di conflitti causabili dalla presenza di più nodi sorgente che marcano i pacchetti. Nel caso di misure *end-to-end* i pacchetti vengono colorati solo alla sorgente. Dal momento che i parametri di *delay* e *jitter* si riferiscono a valori medi, se anche i pacchetti appartenenti ad uno stesso flusso di traffico seguissero percorsi differenti verso una certa destinazione i risultati ottenuti rimarrebbero validi. Naturalmente in caso di perdita di pacchetti tali misure ne risentono, aspetto che tuttavia vale indipendentemente dal fatto che i pacchetti seguano o meno lo stesso percorso;

- Per misure *link-based* invece, tutto il traffico deve essere colorato quando viene trasmesso sul link, quindi non si hanno più come punti di colorazione solo le due estremità del percorso ma a queste si aggiungono tutte le sonde poste su di esso. Se il traffico è già stato colorato, allora deve essere colorato nuovamente perchè il colore deve essere consistente su tutto il link. Ciò significa che ogni hop appartenente al percorso deve ricolorare il traffico; non è invece richiesto che il colore sia consistente tra link differenti.

L'operazione di marcatura viene scandita da un intervallo temporale. Tale periodo di tempo viene definito come Marking Period (MP). Invece il Marking Cycle Period (MCP) contiene un certo numero di MP. Il valore di MCP

deve essere un multiplo di MP.
Una delle tecniche possibili per marcare i pacchetti è quella di settare uno specifico bit nell' header IP del pacchetto. PNPM propone di utilizzare uno dei due bit meno significativi del campo Diff Service (DS) dell'IP Header. Il bit settato verrà definito come Coloured Bit (CB). Questa non è tuttavia l'unica tecnica possibile. Infatti nell'implementazione del protocollo PNPM SLA è prevista la possibilità di utilizzare un campo dedicato alla marcatura presente in ciascun pacchetto di test. Questa seconda strategia permette di non fare affidamento sulla inalterabilità dell'header IP che di fatto può essere manipolato da uno degli apparati di commutazione non appartenenti alla rete di Telecom Italia.

Ogni MCP inizia sempre con lo stesso *marking* e termina sempre con la marcatura opposta. Ad esempio se il valore del campo CB è inizializzato a 0 dovrà terminare con valore pari a 1. Seguendo questa semplice regola ogni MCP segue sempre la stessa sequenza di marcatura ed in questo modo ogni sistema di conteggio sa quando e quale contatore leggere.
Vanno quindi considerate due fasi che si sovrappongono. La fase di marcatura del pacchetto e aggiornamento dei contatori e la fase di analisi dei risultati ottenuti. Infatti mentre all'interno di un periodo avviene la marcatura di un flusso contemporaneamente si svolge l'analisi di quanto monitorato nel flusso precedente.
Nell'implementazione di TI l'alternanza tra le due fasi assume un valore pari di default pari 5 minuti secondo la logica che tale valore permette di ottenere un buon compromesso tra la frequenza in cui le misure devono essere prese e la stabilità delle misure stesse.

L'algoritmo di marcatura sviluppato da TI prevede le seguenti assunzioni:

- Se il parametro *Marking*, che è uno dei parametri che vengono settati sulla sonda è al valore "ON", il processo di marcatura inizia ogni MP;
- Se la lunghezza del MP è dispari, (es. 5 minuti): se siamo in un minuto dispari è un periodo pari, e viceversa;
- Se la lunghezza del MP è pari (es. 10 minuti): se siamo in un minuto divisibile per il doppio della lunghezza del periodo è un periodo dispari, altrimenti è un periodo pari;
- Se siamo in un periodo pari il *matching packets* (Sorgente, destinazione, DSCP)-> DS = DS0. Se invece siamo in un periodo dispari il *matching packets* (Sorgente, destinazione, DSCP)-> DS = DS1. Ogni pacchetto, eseguita la fase di marcatura, prosegue nella fase di conteggio.

4.2.5 Conteggio dei pacchetti

Poiché il valore di marcatura cambia periodicamente tra due valori, sono necessari due contatori (uno per ogni valore): un contatore per i pacchetti con colore "A" ed un contatore per i pacchetti di colore "B". Per ogni flusso da monitorare e per ogni interfaccia dove il monitoraggio è attivo, sono necessari quindi due contatori. Per esempio, per controllare separatamente tre flussi su un router con 4 interfacce coinvolte, saranno necessari 24 contatori. In caso di misure link-based, il comportamento è simile, tranne che le operazioni di colorazione e di conteggio vengono eseguite link-by-link ad ogni estremo del collegamento.

Per contare il numero esatto di pacchetti trasmessi in un periodo di marcatura, i router devono attendere che il periodo corrente sia terminato ed aspettare un certo intervallo di tempo in modo da essere certi che il valore del contatore sia ormai stabile. Utilizzando un intervallo di marcatura costante, soluzione di fatto implementata in PNPM, è possibile configurare la lettura dei contatori secondo un timer che per esempio, se ogni blocco ha una durata di 5 minuti, legga il contatore per il colore A ogni 5 minuti nel centro del blocco successivo (con colore B). La ragione che porta ad eseguire la lettura dei contatori di un periodo, all'interno della metà del periodo successivo, in termini di tempo, è semplicemente dovuta all'idea di mantenere un certo margine per far si che i contatori abbiano ormai valori stabili. Di fatto tale intervallo potrebbe essere fortemente ridotto in quanto i contatori non avranno certo bisogno di un tale periodo per stabilizzarsi. Vale tuttavia il principio di generalità adottato da PNPM, il quale non definisce caratteristiche tecniche o prestazionali che devono avere gli apparati in cui esso viene implementato.

In PNPM per ogni pacchetto marcato sono necessarie quattro variabili per mantenere le informazioni necessarie per la successiva fase di analisi:

- FTS: (first timestamp)memorizza il timestamp del primo pacchetto ricevuto;
- LTS: (last timestamp)memorizza il timestamp dell'ultimo pacchetto, tale valore è sovrascritto ad ogni nuovo pacchetto;
- Npkt: memorizza il numero di pacchetti ricevuti;
- Nbyt: memorizza il numero di byte ricevuti;

Il DTSS: (delta timestamp sum) può essere calcolato alla ricezione di ogni nuovo pacchetto ed il suo valore sarà pari a `DTSS + (LTS -FTS)`. Ciò evita la

possibilità che si verifichi un *overflow* in quanto il valore maggiore assumibile da DTSS è tale da non eccedere lo spazio di memoria allocato alla variabile stessa.

Le variabili vengono aggiornate alla ricezione di ogni nuovo pacchetto.

Esempio: pseudo-codice, (colore = 1)

```
if(Npkt = 0)then
{
FTS1 = Timestamp del pacchetto;
Npkt = 1;
Nbyt = sizeof(packet);
DTSS1 = 0;
} else {
LTS = Timestamp del pacchetto;
Npkt++;
Nbyt += sizeof(packet);
DTSS += (LTS-FTS);
}
```

4.2.6 Collezione e scambio dei dati

I nodi abilitati ad effettuare l'attività di monitoraggio collezionano i valori dei contatori, ma non sono in grado di utilizzare direttamente queste informazioni per misurare la perdita di pacchetti o le misure *two-way* avendo esclusivamente i rispettivi campioni. Per questa ragione è necessario che le sonde o al termine del test complessivo o al termine di ciascun periodo di marcatura si scambiano i risultati ottenuti localmente cosi da poter calcolare tutte le statistiche fin qui discusse.

4.3 Conclusioni

Vantaggi del metodo PNPM:

- Semplice da implementare: può essere implementato utilizzando funzioni già disponibili sulle principali piattaforme di routing;
- Ai fini della marcatura dei pacchetti non richiede alcuna operazione particolare;

- Accuratezza delle misure di *packet loss*: la capacità di rilevare la perdita del singolo pacchetto è ottenuta mediante misure passive;
- Applicabilità a qualsiasi tipo di traffico: Ethernet, IP, MPLS, ecc, sia unicast che multicast;
- Il metodo può tollerare *out-of-order* grazie alla capacità di ottenere il valore medio delle misure di ritardo;
- Nessun problema di interoperabilità: le caratteristiche necessarie per l'attuazione del metodo sono disponibili su tutte le piattaforme di routing correnti.

I possibili svantaggi derivanti dall'utilizzo della metodologia PNPM come strumento di monitoraggio delle prestazioni di rete vengono definiti nel capitolo dedicato al protocollo PNPM SLA. Questa motivazione si spiega con il fatto che lo sviluppo del protocollo segue le specifiche di PNPM e quindi molti dei possibili vantaggi cosi come degli svantaggi sono tra loro correlati. Viene fatto però notare fin da subito che le maggiori criticità riscontrate nello studio e nella successiva implementazione di PNPM mediante il protocollo PNPM SLA sono proprio dovute alla mancanza dei valori di *Sequence Number* e di *Timestamp*. Infatti se da un lato tali valori appesantiscono il carico elaborativo dovuto al processamento del pacchetto, dall'altro sono informazioni estremamente funzionali, ed infatti ampiamente utilizzate da altri protocolli al fine di svolgere attività di monitoraggio, trasmissione del traffico ed analisi delle prestazioni di rete.

Capitolo 5

PNPM_SLA

5.1 Introduzione

Il protocollo PNPM SLA è stato sviluppato con l'obiettivo di fornire alla metodologia PNPM una struttura formale sulla quale operare. E' un protocollo dedicato all'attività di monitoraggio delle prestazioni in reti a commutazione di pacchetto.
Il protocollo PNPM_SLA è in grado di ottenere informazioni relative a packet loss, delay one-way, delay two-way, jitter e valori minimi e massimi di tali metriche, senza la necessità di utilizzare alcun campo nei pacchetti scambiati fra le sonde al fine di memorizzare informazioni quali i timestamp, i numeri di sequenza o eventuali sfasamenti del clock.
Appare quindi evidente fin da subito una prima differenza tra tale protocollo ed il protocollo IP_SLA. Infatti l'obiettivo chiave sul quale punta PNPM_SLA è proprio quello di ridurre il più possibile il tempo di elaborazione dei pacchetti generati. Attualmente viene preso in considerazione il monitoraggio di traffico artificiale generato da sonde attive dedicate.

5.2 Aspetti chiave di PNPM_SLA

Successivamente vengono discussi nell'ordine i seguenti aspetti:

- Il contesto in cui viene sviluppato PNPM_SLA. In particolare quali vincoli sono stati presi in considerazione ai fini del suo sviluppo;
- L'architettura logica di una rete in cui venga utilizzato PNPM_SLA come strumento di monitoraggio e l'algoritmo che regola una sessione PNPM_SLA;

- Quali sono le metriche da esso gestite ed in che modo si svolgono le analisi;
- Come PNPM_SLA cerca di ottenere accuratezza nelle misure e sincronizzazione tra gli apparati;
- Il protocollo:
  - Temporizzazione di una sessione PNPM_SLA
    - Fase di controllo e configurazione;
    - Fase di testing;
    - Fase di scambio delle informazioni;
  - Il formato dei pacchetti generati e scambiati durante tali sessioni.

5.2.1 Contesto e vincoli

I punti fondamentali che meglio descrivono il contesto applicativo in cui è stato sviluppato il protocollo sono:

- *Traffico artificiale.* Costruzione di pacchetti progettati in modo da minimizzare il tempo di elaborazione sulle sonde eseguenti i test. Inoltre non è affatto necessario, anzi, generare una elevata quantità di traffico sia per evitare di appesantire inutilmente le sonde PNPM_SLA, il cui compito non è quello di operare a bit rate elevati, sia per evitare di peggiorare le prestazioni di rete. Entrambi questi due aspetti sarebbero infatti un controsenso viste le finalità del protocollo;
- *Sonde dedicate.* Apparati il cui unico obiettivo è eseguire il protocollo PNPM_SLA. In particolare tale esigenza è dettata dalla necessità di adottare sonde client COTS il più possibile economiche;
- *Analisi end-to-end.* L'idea sottostante a questo punto è quella di posizionare in posizioni strategiche le sonde PNPM_SLA ai bordi della rete che si intende monitorare;

Nessuno di tali elementi è esplicitamente richiesto ai fini di interoperabilità con la metodologia PNPM. La scelta di operare in tale contesto è motivata dall'obiettivo di raggiungere le migliori prestazioni possibili.

Volendo riassumente in una definizione il principale vincolo considerato nella progettazione del protocollo è possibile affermare che il vero collo di

bottiglia è quello economico. Se si considera infatti che il numero di apparati che dovranno eseguire il protocollo è potenzialmente molto elevato (alcune centina di sonde per esempio), e che tali apparati saranno dedicati esclusivamente ad attività di monitoraggio, è evidente che il costo per la singola macchina deve essere il più possibile contenuto.
Questo concetto si traduce molto semplicemente in sonde aventi limitate capacità hardware:

- Processori poco performanti. CPU Clock ¡ 1GHz. 700GHz per il Raspeberry pi;
- Poca RAM. Nel caso del Raspberry pi si dispone di appena 512 MByte;
- Una sola NIC. Vincola alcuni aspetti nello sviluppo del software;
- Problemi nel mantenere gli apparati sincronizzati. E' necessario prevedere un recupero periodico del sincronismo.
- Costanza nel tasso di trasmissione non facile da ottenere;
- In generale, maggiore variabilità temporale nell'esecuzione delle operazione di cattura e processamento o trasmissione del traffico. Quindi è possibile osservare un peggioramento nella precisione con cui vengono prese le misure;
- etc.

In conclusione, l'idea è: preso atto dei vincoli intrinsechi al contesto di sviluppo di PNPM_SLA sono state prese in considerazione alcune strategie (traffico artificiale, sonde dedicate, etc.) aventi l'intenzione di ridurne il più possibile gli effetti negativi.

5.2.2 Topologia di una rete PNPM_SLA

Di seguito vengono presentati i concetti relativi alla topologia di un sistema che utilizzi PNPM_SLA come strumento di monitoraggio così da comprenderne lo logica operativa. La figure 6.1 riassume l'idea di topologia di rete PNPM_SLA e del ruolo assunto dalle sonde in campo.

Le sonde PNPM_SLA sono distribuite all'interno di una normale rete TCP/IP o UDP/IP. Idealmente sono posizionate in punti strategici, dove cioè si abbia maggior interesse a valutare le prestazioni. L'idea che riassume il tutto può essere la seguente: *attraverso la generazione di traffico artificiale*

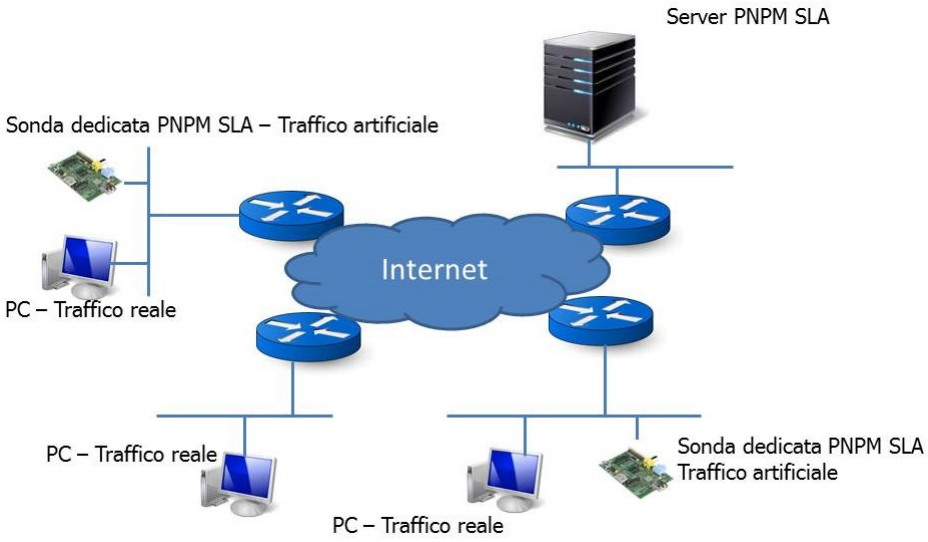


Figura 5.1: Modello di rete PNPM SLA.

PNPM_SLA, immerso all'interno di traffico reale, le sonde dedicate alla generazione ed alla cattura del traffico artificiale sono in grado di determinare l'andamento generale della porzione di rete sotto osservazione.

Le sonde PNPM_SLA possono ricoprire o il ruolo di client o il ruolo di server:

- Sonda Client. Apparato dedicato COTS con forti vincoli economici. Su di esso è in esecuzione un processo PNPM_SLA client;

- Sonda Server. Apparato idealmente dedicato ma sul quale non si hanno in realtà particolari vincoli economici. Esegue un processo PNPM_SLA server.

Una tipica istanza dell'architettura può essere rappresentata secondo il classico paradigma *client-server*.

- Il **server** deve essere un apparato con buone capacità elaborative dovendo potenzialmente gestire contemporaneamente un elevato numero di *sessioni PNPM_SLA*. L'indirizzo IP ed il numero di porta associati all'attività di segnalazione devono essere noti alle sonde *client*. E' attraverso tale indirizzo che il server riceve le richieste per avviare nuove sessioni di test da parte dei client. Il server, nel caso accetti di avviare un test con la sonda richiedente comunica un insieme di informazioni

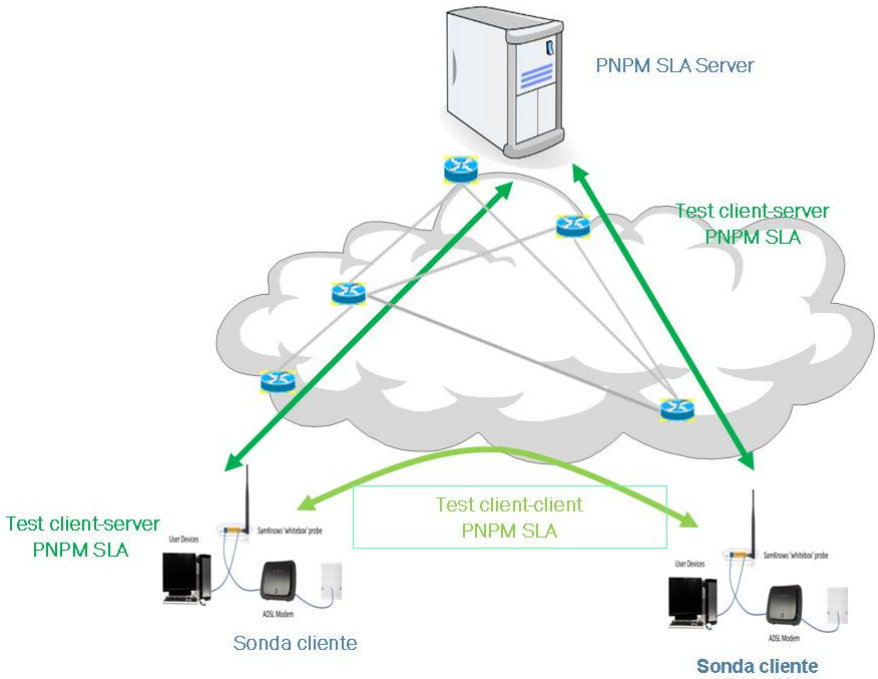


Figura 5.2: Architettura PNPM SLA.

tra le quali la porta UDP a cui indirizzare i successivi pacchetti di testing.

- Il **client** deve conoscere l'indirizzo a cui contattare il server per avviare una sessione PNPM_SLA. Nel messaggio di richiesta definisce le caratteristiche che desidera vengano utilizzate nella successiva fase di test. In risposta può ricevere una accettazione, una negazione o eventualmente una contro richiesta.

5.2.3 Metriche considerate

Come è già stato discusso per il protocollo IP SLA le metriche più utilizzate al fine di valutare il livello di servizio all'interno di una certa rete sono il *delay (one-way e round-trip), il jitter e la packet loss*. E' poi possibile definire ulteriori metriche, ad esempio per valutare il comportamento di specifici servizi, quali il Domain Name System (DNS), il Dynamic Host Configuration Protocol (DHCP), connessioni HTTP, etc.

In PNPM_SLA le metriche prese in considerazioni sono appunto la *packet-loss* il *delay* ed il *jitter*. Inoltre, in assenza di perdita di pacchetti, è possibile ottenere anche misure puntuali quali il ritardo ed il jitter minimo e massimo.

In PNPM_SLA non esiste il concetto di Round-Trip Reale. Un pacchetto viene generato dalla sorgente ed una volta raggiunta la destinazione, dopo essere stato processatto, cessa il suo scopo. Questo aspetto permette di ottenere quindi normali analisi one-way. Per ottenere invece l'equivalente delle misure two-way è necessario che i due apparati coinvolti si scambino le rispettive misure one-way in modo da ricavare in sostanza la misura two-way virtuale.

Di seguito viene descritto come, da un punto di vista concettuale, in PNPM_SLA si ottengono le metriche sopra citate. L'implementazione nel software è più articolata in quanto vengono considerati differenti aspetti legati alla sincronizzazione ed alla necessità di massimizzare la precisione con cui vengono rilevate le marcature temporali. Questi due aspetti vengono approfonditi nella sezione 5.4. Va infine ricordato che in nessuno dei campi di un pacchetto di test PNPM_SLA è riportata alcuna informazione relativa ne al timestamp ne al numero di sequenza.

- Perdita di pacchetto. Ciascuna sonda conosce il numero di pacchetti che verranno trasmessi all'interno di una sessione PNPM_SLA. Quindi è immediato attraverso l'uso di semplici variabili contatore scoprire se il numero di pacchetti ricevuti corrisponde al numero di pacchetti che sarebbero dovuti essere effettivamente stati ricevuti. Tuttavia ,in prima istanza, non è possibile per la destinazione conoscere le motivazioni che hanno portato a perdere uno o più pacchetti. In altre parole non è possibile sapere se il pacchetto non stato ricevuto in quanto mai trasmesso, piuttosto perchè è stato scartato da un router lungo il suo percorso e cosi via. Inoltre, non essendo i vari pacchetti numerati, non è cosi semplice ne scoprire quale pacchetto è andato perso ne accorgersi di una eventuale duplicazione.

- Latenza di rete(media) one-way. La sonda destinazione non solo conosce il numero di pacchetti coinvolti ma conosce anche l'istante teorico di trasmissione di ciascuno dei singoli pacchetti da parte della sorgente. Questa informazione è ricavabile in fase di handshake. Per tanto sottraendo all'istante di ricezione del paccheto i-mo l'istante di trasmissione presunto dello stesso pacchetto e possibile ottenere il delay one-way. Sommando tutti i delay one-way per pacchetto e dividendone il risultato per il numero di pacchetti ricevuti si ottiene il delay medio one-way.

- Latenza di rete(media) two-way. Ciascuna sonda durante la fase di test svolge le operazioni scritte nei due punti precedenti. Al fine di ottenere l'informazione relativa al DMTW è quindi sufficiente che gli apparati si scambino i rispettivi valori di Delay Medio one-way (DMOW). Questa operazione viene eseguita nella terza ed ultima fase del protocollo, appunto definita come la fase di scambio delle informazioni raccolte durante la fase di test.

- Jitter(medio) one-way. Il jitter rappresenta il ritardo di inter-arrivo tra i vari pacchetti. Sottraendo l'istante di ricezione del paccheto i-mo-1 all'istante di ricezione del pacchetto i-mo si ottiene il jitter relativo all'intervallo i-mo. Dati N pacchetti ricevuti da una certa sonda sono presenti N-1 intervalli. Sommando il valore di tali periodi e dividendolo sempre per N-1 si ottiene l'intervallo medio di ricezione tra i vari pacchetti appartenenti ad una sessione PNPM_SLA. E' importante osservare come tale valore sia non solo influenzato dai ritardi che i pacchetti possono accumulare in rete ma anche dalla precisione con cui gli stessi sono trasmessi dalla sorgente. Infatti una delle principali problematiche che si riscontrano su apparati di fascia molto bassa, quale è appunto un Raspberry, è legata alla difficoltà di ottenere un tasso di trasmissione costante. Tale imprecisione aumenta considerevolmente all'aumentare del tasso di trasmissione.

- Misure di Delay e Jitter minimo e massimo. Queste informazioni sono facilmente ricavabile mediante l'utilizzo di opportune variabili in cui vengano aggiornate, pacchetto dopo pacchetto, tali informazioni. In presenza di packet loss perdono rilevanza in quanto naturalmente assumeranno dei valori errati. Ad esempio nel caso del jitter, se venisse perso il pacchetto i-mo, il jitter max sarebbe il risultato della differenza tra il timestamp del pacchet i-mo-1 e il timestamp del pacchett i-mo+1.

Tali metriche vengono considerate all'interno di sessioni PNPM_SLA aventi una certa durata. Se per esempio si considera una sessione della durata

di 30 minuti con intervalli di marcatura di 5 minuti verranno accumulati 6 distinti valori relativi alle differenti metriche sopra descritte.

5.2.4 Accuratezza delle analisi e sincronizzazione tra le sonde

- Dal momento che PNPM_SLA è in sostanza utilizzato per la validazione delle SLA l'accuratezza con cui vengono realizzate le misure è un aspetto centrale con immediate ricadute economiche. Infatti la maggior parte degli SLA prevedono sanzioni nel caso di non conformità con quanto definito nella SLA stessa;

- PNPM_SLA utilizza il protocollo di sincronizzazione NTP. Tutte le sonde che partecipano ad una sessione PNPM_SLA devono quindi sincronizzare i rispettivi clock attraverso una o più richieste ad un server NTP. Sono già stati studiati i vantaggi e soprattutto i limiti di tale protocollo. Non c'è modo do ottenere una sincronizzazione tra i clock di sistema degli apparati ad alcune centinaia di microsecondi, neppure se tali apparati sono all'interno della stessa LAN. Questa è una delle ragioni per cui è utile considerare misure *round-trip*;

- PNPM_SLA concentra le sua analisi all'interno di intervalli temporali. In altre parole, non fornisce analisi ping-like, ma osserva intervalli aventi une certa durata, tipicamente periodi multipli di 5 minuti, riportandone quindi i valori medi. D'altra parte considerando l'impiego di apparati simili al Raspberry è prevedibile una maggiore variabilità nel singolo scambio di pacchetto rispetto all'impiego di apparati di rete con costi decisamente maggiori. Ciò che è rilevante è mantenere una costanza nei risultati ottenuti nel breve periodo ed una precisione nelle misure prossima al millisecondo.

- Lato server, ovvero l'apparato che potenzialmente necessita di maggiori capacità prestazionali, è previsto in una successiva fase di sviluppo, l'impiego di driver di I/O accelerato al fine di ottimizzare il tempo di processamento del pacchetto e minimizzare il ritardo di trasmissione.

5.3 Il protocollo

5.3.1 Temporizzazione del protocollo

Di seguito vengono presentate le fasi che scandiscono l'attività del protocollo ed il formato dei messaggi da esso generati.

Una sessione PNPM_SLA è sempre caratterizzata da tre intervalli temporali che si succedono in sequenza:

- **Fase di Handshake.** Permette alle parti di concordare i valori dei parametri utilizzati nella successiva fase dei test, ad esempio il tasso di trasmissione dei pacchetti, l'istante di inizio delle misure, etc. E' sempre avviata da una sonda client attraverso un messaggio di *Control-Request*. Il destinatario del messaggio può essere sia il server, sia un'altra sonda client. Tuttavia, questo secondo scenario, che comunque è già previsto nella logica algoritmica e nel formato dei pacchetti PNPM_SLA, non è attualmente implementato nel software fin qui sviluppato.

- **Fase di test.** Durante questa fase vengono scambiati tra la sonde in esame due flussi di pacchetti di test identici. All'interno di ciascun periodo di marcatura le sonde accumulano le informazioni utili a calcolare le metriche relative a quel dato periodo.

- **Fase di scambio delle informazioni raccolte.** Al termine del periodo di test gli apparati coinvolti si scambiano i rispettivi risultati. A questo punto ciascuna sonda è in grado di calcolare le metriche presentate in precedenza.

La somma di tali intervalli scandisce la durata di ciascun periodo di marcatura. Naturalmente ciascun test può essere composto da un numero variabile di sotto intervalli. Di default la durata di un MP è pari a cinque minuti. Nell'implementazione software è inoltre prevista la possibilità di generare MP di durata pari a due minuti. La figura 5.3 evidenzia il concetto di temporizzazione e presenta la durata in secondi assunta da ciascuno dei sotto periodi. Tali valori sono parametrizzabili e possono quindi essere modificati in base ad esigenze più specifiche.

Una coppia di messaggi, *Control-Request* e *Control-Response*, caratterizza la fase utile a definire l'apertura di una nuova sessione di test PNPM_SLA. Il Client trasmette un messaggio di Control-Request, al quale segue un messaggio di *Control-Response* contenente l'esito della richiesta. Il messaggio

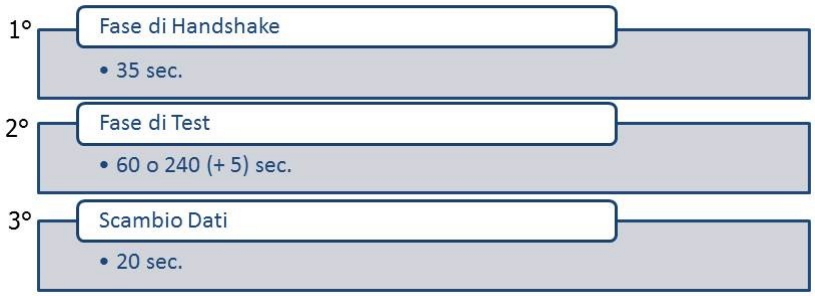


Figura 5.3: Temporizzazione PNPM SLA.

di *Control-Request* può essere trasmesso più volte se non viene ricevuto il relativo messaggio di *Control-Response*. Il numero di volte che può essere ritrasmesso è configurabile dal Sender in modo da evitare di interrogare all'infinito una sonda che per diverse ragioni potrebbe non essere raggiungibile. Conclusa la fase di controllo ha inizio l'effettiva fase di test durante la quale vengono generati due flussi di traffico, dal client al server e viceversa, ed accumulate le informazioni utili al calcolo delle metriche considerate. Al termine del periodo di test, le sonde coinvolte si scambiano i valori accumulati durate il periodo di marcatura appena concluso potendo così calcolare anche i valori medi.

5.3.2 Fase di controllo

La fase di controllo rappresenta la fase iniziale del protocollo e permette alle sonde di negoziare tutti i parametri da utilizzare durante i test. I valori negoziati sono:

- Durata del test (es: 30 minuti);
- Intervallo del periodo di marcatura (es: 5 o 2 minuti);
- Tasso di trasmissione dei pacchetti (es: 1 pacchetto ogni 20 millisecondi);
- Coloured Bit, ovvero il valore della marcatura iniziale dei pacchetti (es: 0/1);
- Class of Service, ovvero il tipo di traffico che si desidera monitorare (es: data, voce, etc.) nell'ipotesi di traffico reale.

Campo	Byte	Descrizione
Version	8	Versione supportata (Attualmente esiste solo una prima versione di PNPM SLA)
Riserved	8	Campo riservato, Deve essere settato a 0.
Status	16	Indica il successo o il fallimento del messaggio.
Sequence Number	32	Usato per mappare la richiesta alla risposta.
Total Length	32	Lunghezza totale del messaggio in byte.

Tabella 5.1: Command-Header

La sonda destinazione, ricevuto il messaggio di *Control-Request*, può agire nei modi seguenti:

- Accettare la nuova richiesta e rispondere con un messaggio di *Control-Response* (Campo *Status* settato al valore 1);
- Rifiutare la richiesta. Questa situazione può essere dovuta a differenti fattori, ad esempio nel caso in cui il server abbia raggiunto il numero massimo di sonde ammissibili (Campo *Status* settato al valore 2);
- Infine, se la richiesta non può essere accettata così come proposta, il *responder* trasmette un nuovo messaggio di *Control-Request* in cui inserirà i parametri che desidera supportare (ad esempio un tasso di trasmissione inferiore, una differente durata del test, etc.). A questo punto il *sender* risponderà con il relativo messaggio di *Control-Response* accettando (Status = 1) o rifiutando (Status = 2) i nuovi parametri.

Messaggio di Control-Request

Un messaggio di *Control-Request*, rappresentato in figura 5.4, contiene due sezioni:

- Command-Header;
- Sezione CSLD.

Il contenuto di *Command-Header* è mostrato in figura 5.5 ed i campi presenti assumono il seguente significato:

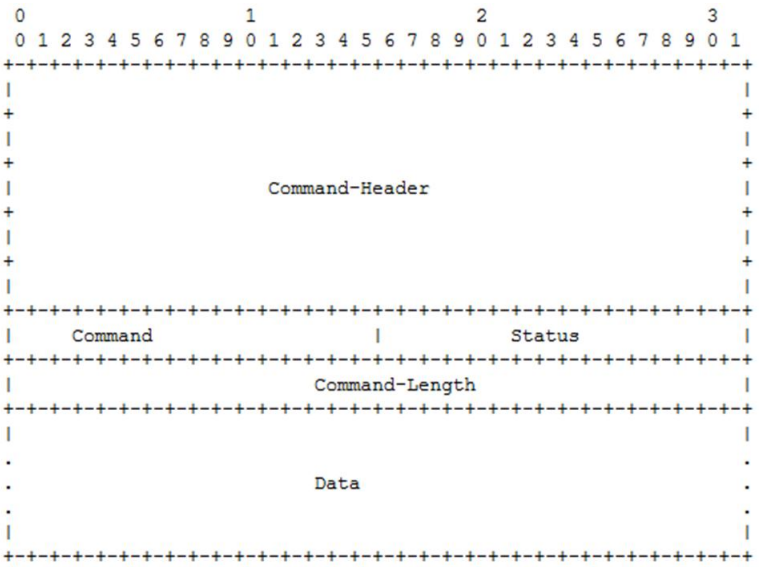


Figura 5.4: Messaggio di Control-Request PNPM SLA.

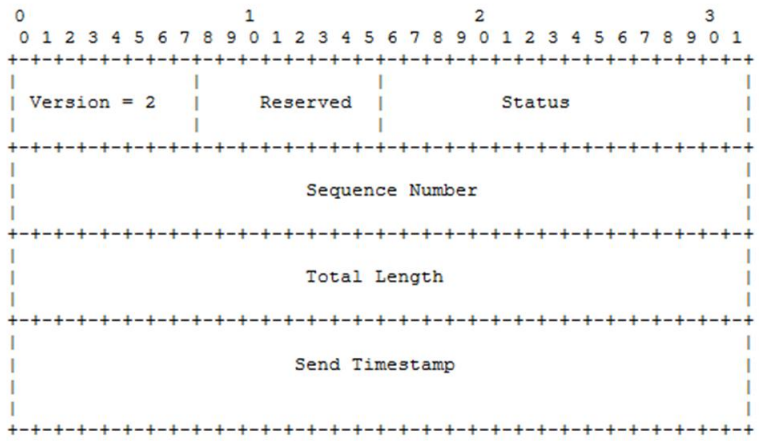


Figura 5.5: Command-Header PNPM SLA.

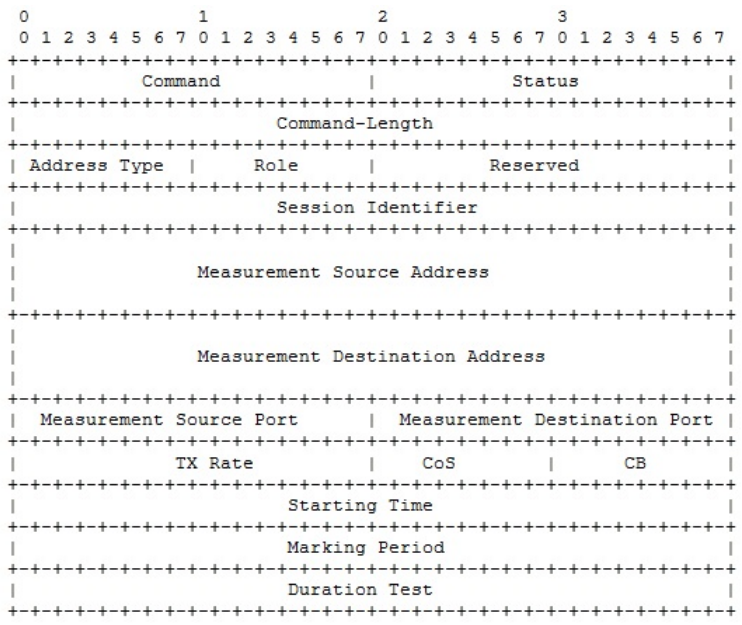


Figura 5.6: CSLD Request-Message PNPM SLA.

La parte restante del *Request-Message*, CSLD, mostrata in figura 5.6, riporta il tipo di misure che dovranno essere realizzate nella fase di monitoraggio e contiene le seguenti informazioni:

Messaggio di Control-Response

Il responder, sia esso il server o una sonda client, invia un messaggio di risposta del tutto analogo al messaggio di *Control-Request* con i campi *Status* e *Role* aggiornati. In questo modo è possibile capire l'esito della richiesta e se la risposta rappresenta un messaggio di *Control-Response* o un nuovo messaggio di *Control-Request*.

I possibili valori che può quindi assumere il campo *Status* sono:

- $0 -> $ Richiesta di una nuova sessione;
- $1 -> $ Richiesta Accolta;
- $2 -> $ Richiesta Respinta.

Campo	**Byte**	**Descrizione**
Command	16	Indica se il messaggio è di tipo Control-Request o Control-Response.
Status	16	Indica l'esito di una richiesta.
Command-Length	32	Indica la dimensione del CSLD in ottetti.
Address Type	8	Permette di differenziare tra indirizzi IPv4 e IPv6.
Role	8	Indica se l'*end-point* è il server Telecom o una sonda client.
Delta Starting Time	16	Indica l'istante di trasmissione del primo pacchetto.
Session Identifier	32	Identificatore della sessione.
Measurement Source Address	32	Indirizzo IP della sonda Sender.
Measurement Destination Address	32	Indirizzo IP dell'apparato di destinazione.
Measurement Source Port	16	Porta della sonda Sender.
Measurement Destination Port	16	Porta dell'apparato di destinazione.
TX Rate	16	Indica il tasso di trasmissione dei pacchetti. Il valore è da considerarsi in millisecondi (esempio: TX Rate = 0000000000010100 = 20 = 1 pacchetto ogni 20 millisecondi).
CoS	8	Indica il tipo di traffico contenuto nel payload.
CB	8	Indica il bit di marcatura iniziale dei successivi pacchetti di test (0x8 / 1x8)
Starting Time	32	Indica l'istante di inizio del test.
Marking Period	32	Indica la durata di ciascun periodo di marcatura. Il suo valore è da intendersi espresso in secondi.
Duration Test	32	Indica la durata complessiva del test. Il suo valore è da intendersi espresso in secondi.

Tabella 5.2: Sezione CSLD

- I valori del campo Status sono applicabili sia al *Command-Header* sia alla sezione *CSLD*. Nel Command-Header, il campo Status indica successo solo se la sezione CSLD ha il campo Status impostato al valore 0.

- Il campo *Role* definisce se il messaggio è di tipo *Control-Request* o *Control-Response*. Se il receiver riceve un pacchetto con Role = 0 significa che ha ricevuto un Control-Request (ed il campo status assumerà valore 0), se invece il campo Role = 1 il messaggio è di tipo Control-Response (ed il campo Status potrà assumere valore 1, in caso di richiesta accolta, o 2 in caso di richiesta respinta).

In linea di principio non c'è motivo per cui la sonda client, che è un apparato dedicato alle attività di monitoraggio, non debba accettare uno o più parametri proposti dal Sender. In questo progetto di tesi, e secondo le attuali direttive di Telecom Italia, l'apparato ricoprente il ruolo di sonda client è un dispositivo noto, ovvero di cui se ne conoscono le caratteristiche operative ed i limiti prestazionali.

5.3.3 Fase di Test

La fase di test rappresenta la seconda parte del protocollo e consiste nella trasmissione e ricezione di pacchetti definiti come *Measurement-Request*. La sonde PNPM_SLA, in accordo con quanto impostato nella fase di Controllo, avviano lo scambio dei pacchetti di misura in un certo istante in modo sincrono. I frame verranno quindi trasmessi con la frequenza e per la durata predefinita.Questa seconda fase ha una durata di default pari a quattro minuti ed opzionalmente pari a due minuti. In questo intervallo le sonde collezionano tutte le informazioni utili a calcolare le metriche di analisi. Il numero di pacchetti trasmessi e ricevuti, il delay medio one-way, il jitter medio, il delay ed il jitter minimo e massimo.

Nota la frequenza di trasmissione e nota la durata di un MP è immediato ricavare il numero di pacchetti che verranno trasmessi ed è quindi semplice in ricezione verificare l'eventuale perdita di uno o più pacchetti.
Il campo *Sequence Number* che è normalmente presente nei pacchetti IP SLA di *Request-Measurement* e *Request-Response*, non è invece utilizzato in PNPM_SLA. Ovviamente il sequence number è utile in particolare per rilevare non tanto la perdita di uno o più pacchetti, in quanto questa situazione è facilmente osservabile con l'utilizzo di semplici variabili contatore, ma per rilevare la ricezione di pacchetti ricevuti fuori ordine o eventuali pacchetti

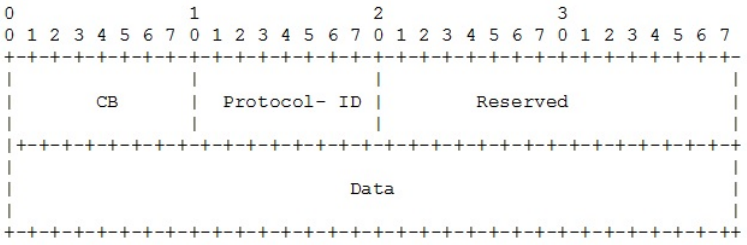


Figura 5.7: Pacchetto di misura PNPM SLA.

Campo	**Byte**	**Descrizione**
Coloured Bit	8	Indica il valore della marcatura corrente del pacchetto. Tale valore cambia ad ogni intervallo di marcatura.
Protocol Identifier	8	Indica un valore che identifichi in modo univoco il protocollo PNPM_SLA.
Reserved	16	Impostato a 0. Eventualmente utilizzabile per estensioni future.
Data	32	Questo campo è utilizzato per raggiungere la dimensione minima del pacchetto.

Tabella 5.3: Pacchetto di test PNPM SLA

duplicati. L'impatto sui risultati delle statistiche, in presenza di uno o più *out-of-order*, è potenzialmente critico. Per questo motivo è molto importante il valore assunto dal tasso di trasmissione dei pacchetti. Tale valore deve essere impostato in modo che l'intervallo di trasmissione tra un pacchetto e quello successivo renda estremamente improbabile la sovrapposizione di due pacchetti secondo la logica che, data l'entità del ritardo tra un pacchetto e quello successivo, e considerato il delay end-to-end, non è possibile che i due pacchetti vengano ricevuti in ordine inverso rispetto all'ordine con cui sono stati trasmessi.

In figura 5.7 viene quindi rappresentato il formato del pacchetto di misura ed il significato dei campi utilizzati.

5.3.4 Fase di scambio dati

La terza ed ultima fase è appunto definita come fase di scambio dei dati collezionati. Ciascuna sonda client che ha partecipato ad una fase di test

trasmette al server, e potenzialmente riceve dal server, le informazioni raccolte localmente. In assenza di tale scambio, per le ragioni descritte nelle sezioni precedenti, non sarebbe possibile calcolare i valori di round-trip virtuali che, considerato fattori quali il livello di sincronizzazione ottenibile mediante NTP, la tipologia di traffico utilizzato per ricavare le analisi, etc., è particolarmente rilevante. Infatti è differente considerare traffico real,e ad esempio di tipo streaming, dal traffico PNPM_SLA. Nel primo caso infatti l'importanza legata al jitter è nettamente maggiore. Basti considerare ad esempio lo studio sul dimensionamento di buffer e tecniche di mitigazioni del jitter che lo interessano.
Nella attuale implementazione sono previsti circa venti secondi in cui gli apparati si scambiano le rispettive informazioni, svolgono le analisi e generano un file di testo memorizzato localmente contente i risultati.
Un successivo sviluppo può portare ad un raffinamento di questa fase, ad esempio attraverso la trasmissione via FTP delle analisi ad un terzo nodo esclusivamente dedicato alla collezione dei dati raccolti nell'intera rete PNPM_SLA, oltre che alla realizzazione di una interfaccia grafica atta a mostrare tali risultati in modo semplice ed intuitivo così da permettere ad ogni tipo di utente di comprendere facilmente il livello di funzionamento della rete sotto osservazione.

5.4 Implementazione del timestamping

L'operazione di marcatura relativa ad un certo evento è importante in diverse situazioni. La precisione con cui è possibile ottenerla dipendente poi dallo specifico contesto applicativo. Come è già stato discusso nel capitolo 2, via HW è possibile ottenere misure con precisioni vicine al nanosecondo, ma il costo in termini economici di tali apparati è ancora elevato. Inoltre grazie ai nuovi processori multi-core, ai nuovi adattatori di rete, ed ai nuovi driver dedicati alla cattura ed al processamento del traffico ad alte prestazioni, anche in ambienti meno avanzati è possibile ottenere buone prestazioni.

I nuovi acceleratori di I/O cercano di superare i limiti degli stack di rete general-purpose applicando tecniche quali:

- Pre-allocazione della momoria.
- Accessi zero-copy tra il kernel e lo user-space (memory mapped).
- Batch processing.
- ...

Tuttavia tecniche quali ad esempio il processamento dei pacchetti a blocchi (batch processing)) rischiano addirittura di peggiorare l'accuratezza con cui il timestamp viene ottenuto.

Il modo in cui, nella attuale implementazione del software, è effettivamente realizzata l'attività di timestamping è relativamente articolato in quanto vengono considerati differenti aspetti legati alla sincronizzazione ed alla necessità di massimizzare la precisione con cui sono rilevate le marcature temporali.

Vengono quindi di seguito richiamati due aspetti particolarmente importanti al fine di massimizzare da un lato la precisione e minimizzare dall'altro il carico elaborativo in particolare sulla sonda server.

1. In precedenza è stato scritto che durante la fase di handshake le sonde prendono conoscenza dell'istante di inizio del test. Quindi, per ipotesi, le sonde potrebbero concordare un dato minuto, ed in quel minuto iniziare la trasmissione del primo pacchetto di test per continuare a trasmettere ad un tasso costante. Tuttavia in fase di test si è osservato un problema se quanto appena scritto viene applicato alla lettera. In

pratica se la latenza di rete è minore dell'errore di sincronizzazione tra gli apparati, uno o più pacchetti vengono perduti dall'una o dall'altra sonda. In altre parole, se si considerano due sonde A e B, e per ipotesi la sonda B ha un riferimento temporale maggiore rispetto alla sonda$A$, quest'ultima inizierà a trasmette i pacchetti prima ancora che B sia effettivamente pronta a riceverli. In conclusione ai fini di evitare tale fenomeno è stato introdotto un tempo di attesa tale da garantire che, nonostante un certo sfasamento sempre presente tra gli apparati, non possa mai verificarsi packet-loss per ragioni di sincronizzazione. Notare inoltre che un ulteriore periodo di attesa è mantenuto anche nella fase conclusiva del test cosi da dare il tempo a pacchetti aventi accumulato un certo ritardo di essere ugualmente ricevuti.

2. Una seconda osservazione è la seguente: la sonda server può avere potenzialmente un numero di sessioni attive molto elevato. Se consideriamo un dato istante di inizio del test ed un dato tasso di trasmissione è ragionevole considerare che il server si troverà nell'arco del test a rilevare dei burst relativi sia alla ricezione sia alla trasmissione simultanea dei pacchetti. Viceversa tra un picco e quello successivo si ritroverà in una sostanziale situazione di inattività. Questo scenario ha un evidente impatto negativo sulla precisione delle misure. Un primo esempio si può osservare nel caso in cui il server si ritrovi a dover trasmettere nello stesso istante un elevato numero di pacchetti, ciascuno destinato ad una differente sonda client. Per forza la precisione con cui viene trasmesso il primo frame è maggiore di quella con cui viene trasmesso l'ultimo pacchetto. Un secondo esempio di degrado delle prestazioni è relativo alla fase di ricezione dei pacchetti. Ciascuno di essi infatti, una volta catturato, porta ad una operazione di timestamping ed alla esecuzione di un certo numero di iterazioni utili per la collezione delle informazioni sul test. Di nuovo, considerando un ipotetico buffer contenente i pacchetti ricevuti nell'istante i-mo, la precisione ottenuta nell'analisi del primo pacchetto estratto dal buffer sarà migliore rispetto a quella che si avrà per gli ultimi pacchetti da esso estratti. Dall'altra parte un fenomeno simile è già stato trattato quando venivano presentati i vantaggi e gli svantaggi della tecniche nota come *Batch processing* utilizzata anche da alcuni driver di I/O accelerato.
   Al fine di ridurre tali problematiche l'implementazione attuale del software PNPM_SLA utilizza la seguente strategia: oltre agli sfasamenti inseriti nel punto precedente, ciascuna sonda adotta un ulteriore ritardo nell'istante di trasmissione del primo pacchetto. Tale ritardo è specifico per ciascuna sonda e garantisce che le sonde client non inizino

a trasmettere i pacchetti nello stesso istante, e che mantengano inoltre tale sfasamento durante l'intero periodo del test. Va precisato che, al fine di ottenere tale trucchetto, il tasso di trasmissione di ciascuna sonda è pari al valore del tasso di trasmissione del server diviso per il numero massimo di sonde con cui il server è disponibile ad avviare un test. Quindi se per esempio il server trasmette ad un rate di 1 pacchetto ogni 5 ms e supporta fino a 100 sonde contemporaneamente, ciascuna sonda client avrà un tasso di trasmissione pari ad 1 pacchetto ogni 500 ms (480 pacchetti se il test ha una durata di 5 minuti).

5.5 Conclusioni

Il protocollo PNPM_SLA, durante la fase di misura, cura due aspetti:

- il formato del pacchetto;
- la relativa trasmissione e ricezione da parte delle sonde.

La struttura del pacchetto di misura permette di ottimizzare il tempo di elaborazione ad esso associato. Dal momento che vengono presi in considerazioni apparati COTS l'idea alla base di tutto è quella di ridurre al minimo il numero di operazioni che la sonda deve svolgere per ciascun pacchetto, sia in fase di trasmissione che in fase di ricezione.
Preso atto dell'impossibilità di adottare tecniche di sincronizzazione migliori di NTP, in PNPM_SLA si risolve almeno in parte il problema dello sfasamento tra i clock delle sonde, oltre che il problema della deriva degli stessi, mediante il concetto di DMTW, ovvero di misure di round-trip virtuali.

Alcune osservazioni:

- L'utilizzo del bit DiffServ, nell'header IP, come campo dedicato alla marcatura dei pacchetti, può non essere una soluzione praticabile in alcuni casi, ad esempio al di fuori della rete direttamente controllata da Telecom Italia. Mediante l'utilizzo di PNPM_SLA è possibile evitare tali problematiche grazie alla presenza di un campo espressamente dedicato alla marcatura PNPM.
- L'impossibilità di utilizzare un campo *sequence number* certamente permette di ottenere il vantaggio legato alla capacità di non processare tale informazione, tuttavia si perdono i benefici da esso derivanti, i quali permetto una gestione più semplice sia della perdita di pacchetto sia della eventuale ricezione di frame fuori ordine.

- Un approccio assolutamente simile a quanto appena discusso per il *sequence number* può essere fatto in merito all'assenza dei campi *Send_Timestamp* e *Receive_timestamp*. Tali campi, così come il numero di sequenza del pacchetto, sono utilizzati da diversi protocolli che si occupano del trasferimento/monitoraggio dei pacchetti all'interno di reti a commutazione di pacchetto (Real-Time Transport Protocol (RTP), IP SLA, Transmission Control Protocol (TCP), etc.). Naturalmente la semplicità con cui si possono ottenere informazioni di prestazioni grazie alla presenza dei campi sopra citati è elevata. In altre parole il prezzo da pagare per un loro non utilizzo è pagato da un algoritmo potenzialmente più complesso.

Capitolo 6

Implementazione e test del protocollo

6.1 Introduzione

A valle di quanto presentato in merito a PNPM_SLA è quindi possibile descrivere una prima implementazione del protocollo avente come principali obiettivi, da un lato verificare la reale funzionalità del software in una semplice rete reale e dall'altro osservare i risultati che si ottengono in termini di precisione delle misure.
Nello specifico la metrica sulla quale si è maggiormente concentrata l'attenzione è il *Delay medio two-way*. Questa informazione è infatti particolarmente utile se si considera il modo in cui è strutturato il protocollo PNPM_SLA. Si ricordi infatti quanto già descritto sia in merito ai concetti di Round Trip Time virtuale sia soprattutto al coinvolgimento di sonde aventi prestazioni potenzialmente molte differenti tra di loro. Questo secondo aspetto è infatti particolarmente rilevante nel modo in cui vengono considerate le metriche del Jitter e dei valori minimi e massimi. In altre parole non è possibile trarre delle considerazioni di qualità in merito a tali metriche senza una profonda caratterizzazione di ogni singolo apparato coinvolto nei test con particolare attenzione alla precisione in fase di trasmissione dei pacchetti. E' normale che il Jitter osservato da una sonda destinazione avente prestazioni superiori rispetto alla sonda sorgente, sia peggiore di quello osservabile nello scenario opposto essendo potenzialmente meno preciso il tasso di trasmissione dei pacchetti. In conclusione, in questa prima fase di implementazione del protocollo e nel considerare i reali obiettivi del protocollo, è il DMTW la metrica di maggior rilevanza e sulla quale si è lavorato con maggior attenzione durante la fase di test.

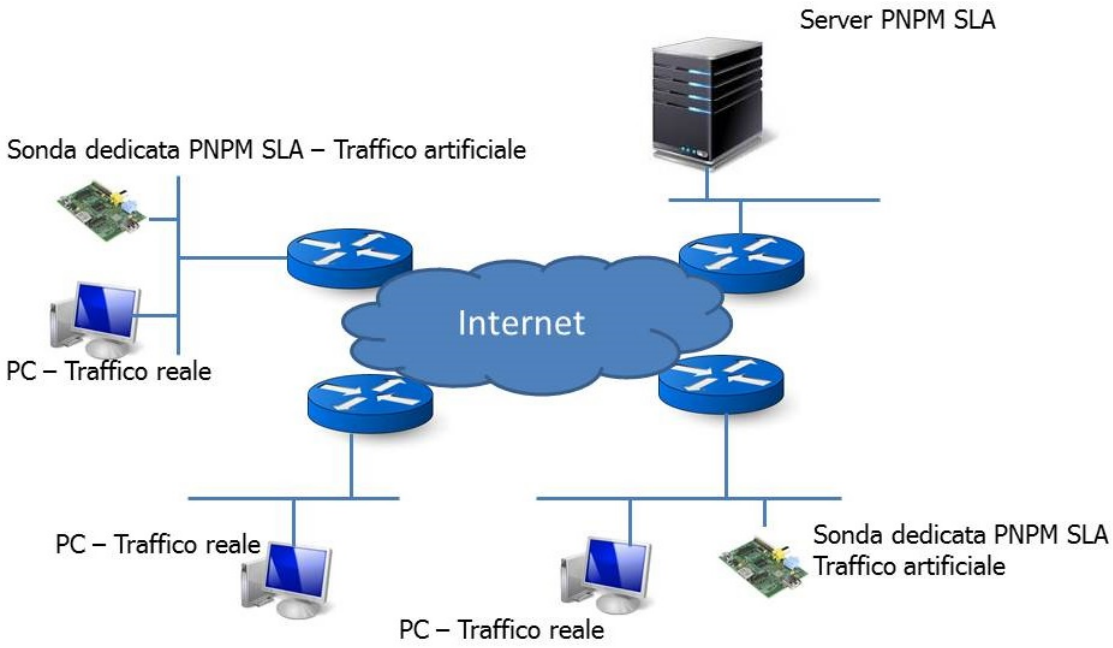
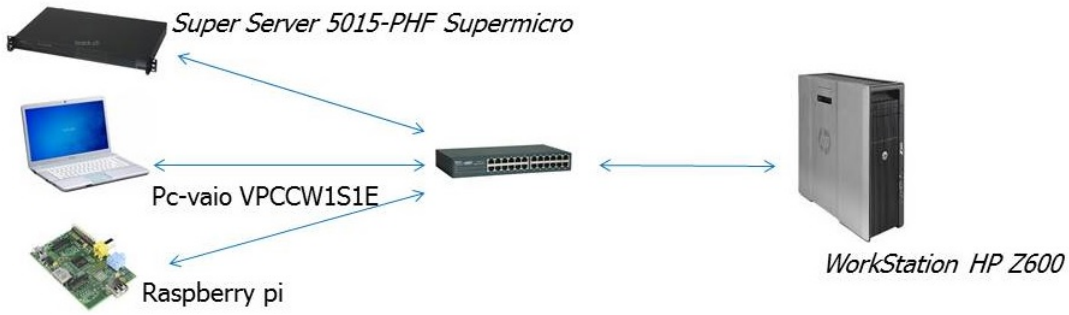


Figura 6.1: Modello di rete PNPM SLA.

6.2 Modello di rete PNPM SLA

Viene richiamata la figura 6.1 per mostrare gli attori PNPM SLA ed i ruoli da essi ricoperti. Osservando tale figura si può apprezzare meglio il concetto di *sonde dedicate*, ovvero apparati in grado di generare traffico PNPM iniettato all'interno di traffico reale.

Gli apparati di rete messi a disposizione da TI e sui quali si sono svolti i test di laboratorio sono l'HP Z600, il Supermicro ed il Raspberry pi. A questi va poi aggiunto un PC Vaio VPCCW1S1E. Va inoltre precisato che tali dispositivi non rappresentano necessariamente quelli che verranno utilizzati effettivamente sul campo. Ciò nonostante la scelta non è stata casuale, infatti sulla base delle principali caratteristiche tecniche è possibile affermare che costituiscono un modello ragionevolmente vicino agli apparati finali e che permettono di ricreare una architettura di rete prossima a quella reale.
In figura 6.2 vengono riportate alcune informazioni su tali apparati rimandando una analisi più approfondita una volta che verranno selezionati in modo stabile quelli definitivi.



	HP Z600	Supermicro	PC Vaio	Raspberri
CPU	Xeon (2GHz)	Intel ATOM D510 (1,66 GHz)	Intel Core 2 Duo P7450 (2,13 GHz)	ARM1176JZF-S (700 MHz)
RAM	4 GB	4 GB	4 GB	512 MB
#NIC	2	2	1	1
Costo	~1000 €	~300 €	~400 €	~40 €

Figura 6.2: Apparati di rete coinvolti.

Di seguito sono riportati i ruoli adottati dalle sonde ed alcune ulteriori informazioni di carattere generale:

- La work-station HP Z600 è l'apparato maggiormente performante messo a disposizione e ricopre all'interno dell'architettura dei test il ruolo di server. Il sistema operativo preso in considerazione è *Ubuntu 12.04*
- Nello svolgimento dei test il Supermicro ricopre il ruolo di client. Tuttavia costituisce un apparato eccessivamente costoso e per questa ragione non rappresenta una reale soluzione da impiegarsi come apparato client in una implementazione su scala. In particolare disponendo di doppia connessione Gigabit Ethernet LAN permetterebbe di superare alcune delle limitazioni riscontrate nello sviluppo del software che fa invece riferimento alla disponibilità di un'unica interfaccia ethernet prendendo in considerazione sonde client meno performanti e soprattutto più economiche. Il Supermicro modello H8SGL è infatti una scheda madre progettata per soluzioni Server. Il acSO preso in considerazione è *Ubuntu 12.04*
- Il Raspberry Pi rappresenta il secondo apparato client. E' un single-board computer sviluppato nel Regno Unito dalla Raspberry Pi Foun-

dation. L'idea di base è la realizzazione di un dispositivo economico, concepito per stimolare l'insegnamento di base dell'informatica e della programmazione nelle scuole. Non è certamente un apparato studiato con l'intenzione di ottenere misure di precisione ne con l'obiettivo di sottoporlo ad importanti carichi elaborativi.
Il progetto ruota attorno a un System-on-a-chip (SoC) Broadcom BCM2835, che incorpora un processore ARM1176JZF-S a 700 MHz, una GPU VideoCore IV, e 256 o 512 Megabyte di memoria. Non prevede né hard disk né una unità a stato solido, affidandosi invece a una scheda SD per il boot e per la memoria non volatile. La scheda è stata progettata per ospitare sistemi operativi basati su un kernel Linux o RISC OS.
Il modello utilizzato nei test è equipaggiato con due porte USB, un controller Ethernet 10/100, 512 Megabyte di RAM e costa circa 35 Dollari.
Raspberry PI non è fornito di un real-time clock, così il sistema operativo deve usare un network time server, o chiedere l'ora all'utente al bootstrap per avere accesso a data e ora.
Raspberry Pi utilizza Linux come sistema operativo. La versione di SO scelta è Debian.

Breve riepilogo sul modello utilizzato:

- Model B
- Prezzo di offerta: USD 35 (GBP 22)
- SoC: Broadcom BCM2835 (CPU + GPU + DSP + SDRAM)
- CPU: 700 MHz ARM1176JZF-S core (famiglia ARM11)
- GPU: Broadcom VideoCore IV, OpenGL ES 2.0, 1080p30 H.264 high-profile decode
- Memory (SDRAM): 512 Megabytes (condivisa con la GPU)
- USB 2.0 ports: 2 (attraverso un hub USB integrato)
- Memoria: SD / MMC / SDIO card slot
- Collegamenti di rete: Ethernet 10/100 (RJ-45)
- Periferiche di basso livello: 2x13 header pins for GPIO, SPI, I2C, UART, +3,3 Volt, +5 Volt
- Real-time clock: No clock or battery
- Corrente (potenza) assorbita: 300 mA, (1,5 W) 700 mA, (3,5 W)
- Alimentazione: 5 V via MicroUSB o GPIO header
- Dimensioni: 85,60 mm x 53,98 mm (3.370 inch x 2.125 inch)

- Sistemi operativi supportati: Debian GNU/Linux, Fedora, Arch Linux e Gentoo
- Sistemi operativi non supportati: RISC OS (shared source)

6.3 Test

Di seguito sono presentati un certo numero di test aventi l'obiettivo di verificare da un lato la funzionalità del protocollo PNPM SLA ed implicitamente del software sviluppato, dall'altro osservare le prestazioni al variare di alcuni parametri quali ad esempio il tasso di trasmissione e la latenza di rete introdotta artificialmente.

Va sottolineato come tali prove permettano di fornire informazioni utili sul protocollo e sulla sua efficacia ma al tempo stesso è bene ricordare che tali informazioni sono ottenute mediante test su determinati apparati aventi specifiche prestazioni. Cambiare apparati implica la necessità di eseguire nuovi test al fine di caratterizzarne il funzionamento con maggiore precisione. Nei precedenti capitoli è stato approfondito come ciascun apparato e ciascun sistema operativo possa ottenere a parità di capacità e costi prestazioni differenti sia per ragioni legate all'hardware sia per ragioni legate al software.

Naturalmente il funzionamento in se dell'algoritmo è indipendente dal sistema su cui viene ad essere eseguito. In linea di principio anche il software dovrebbe essere progettato in modo da poter essere indipendente dalla piattaforma e dal sistema operativo tuttavia l'attuale implementazione è stata testata sugli apparati presentati in precedenza. In futuro può essere interessante testarne il funzionamento su un maggior numero di piattaforme.

Durante la fase di test gli apparati coinvolti non eseguono altri task al di fuori del sistema operativo cosi come la porzione di rete interessata è tendenzialmente scarica. La topologia dei test è quella riportata in figura 6.2. Tutti i terminali collegati allo switch, ad eccezione di quelli che eseguono il protocollo, sono spenti o in stand by.

Per tutte le motivazioni espresse nel capitolo dedicato alla sincronizzazione, il protocollo NTP è lo strumento utilizzato per ottenere una certa sincronizzazione durante i test. In particolare il Server NTP interrogato è posto nella stessa LAN in cui vengono svolti i test.

I test presi in esame sono tre ed hanno naturalmente obiettivi differenti:

- Simulazione di una prova di carico;
- Analisi del ritardo software attraverso l'introduzione di ritardo artificiale inserito mediante l'utilizzo dell'ANUE;
- Verifica del corretto funzionamento del Raspberry pi.

L'obiettivo comune è osservare se i problemi legati al timestamp ed alla sincronizzazione portano ad avere ritardi costanti e quindi prevedibili, oppure introducono dei ritardi variabili e quindi ad una situazione più complessa. La stessa frase può essere riformulata come segue: L'obiettivo comune è osservare se il *ritardo software* è costante.

Ritardo Software Il concetto di tale ritardo può essere esposto nel modo seguente: È il ritardo che introduce l'algoritmo (o il SW) durante la sua esecuzione al fine di svolgere un certo task. Maggiore è la complessità dell'algoritmo maggiore è tale ritardo che naturalmente dipende anche dall'apparato su cui è in esecuzione, sia per aspetti strettamente legati all'hardware ed alla piattaforma, sia per aspetti legati allo specifico sistema operativo.
La capacità di osservare dei ritardi costanti permette di sottrarre, con 'prudenza', tale ritardo dal valore dell'imprecisione misurata.

ANUE L' ANUE è uno strumento di test sofisticato che permette di simulare differenti scenari che si possono verificare in reti reali. Per i nostri scopi è stato utilizzato come fonte di introduzione di ritardo artificiale.

E' necessario anticipare alcune informazioni che interessano i test realizzati:

- I valori riportati sul piano cartesiano sono il risultato del valore medio di un numero di test, compreso tra 6 ed 12, aventi durata pari a due minuti ciascuno;
- Al di sopra di un certo tasso di trasmissione gli apparati, che utilizzano NTP come protocollo di sincronizzazione, non sono più sufficientemente allineati. Questa è la ragione che porta ad ottenere packet loss al di sotto di una certa soglia;
- E' possibile by passare il problema sopra esposto, e di conseguenza la mancanza di sincronizzazione che è sempre presente tra apparati che utilizzino NTP, mediante opportuni accorgimenti nella scrittura del software.

6.3.1 Test 1

Nel primo test vengono analizzate le misure di delay medio two-way e di packet loss all'aumentare del tasso di trasmissione tra l'apparato HP Z600 ed il Supermicro. In particolare i due end-point svolgono rispettivamente il ruolo di master (server PNPM_SLA) e slave (client PNPM_SLA) e sono interconnessi mediante uno switch. Non disponendo di un elevato numero di sonde client non è possibile svolgere un vero e proprio test di carico. Tuttavia considerando che l'apparato Supermicro ha prestazioni strettamente superiori a quelle ottenibili da un client come il Raspberry è possibile utilizzare questo tipo di prova come una simulazione di test di carico.
Va osservato come in questo scenario il collo di bottiglia sia rappresentato dalla macchina meno performante, ovvero il Supermicro. Avendo infatti a disposizione almeno due HP Z600 le informazioni ottenute sarebbero state ancor più significative.

Analisi dei risultati e conclusioni sul test

La figura 6.3 evidenzia l'incremento del DMTW all'aumentare del tasso di trasmissione. Nelle condizioni in cui si svolge la prova è possibile osservare un andamento relativamente costante fino a circa $\frac{1pkt}{50ms}$. Con un tasso di trasmissione maggiore il delay cresce in modo esponenziale. Osservano la tabella in figura 6.4 si osserva che non si verifica packet loss nei casi riportati, tuttavia è presente con tassi di trasmissione che si avvicinano a $\frac{1pkt}{2ms}$. Tale fenomeno è dovuto ad una sincronizzazione non più sufficiente.

6.3.2 Test 2

Nel secondo test si analizza il ritardo software attraverso l'introduzione di ritardo artificiale inserito mediante l'utilizzo dell'ANUE. E' immediato notare che ritardi dell'ordine dei millisecondi, all'interno della topologia in cui si svolgono le prove, non costituiscono di certo la reale latenza di rete. I due apparati coinvolti sono infatti interconnessi da un unico switch ed il diametro della rete è inferiore ai cinque metri. Un ritardo di rete ragionevole è al più dell'ordine di alcune decine di microsecondi. I valori osservati rappresentano quindi ciò che in precedenza è stato definito *ritardo software*. L'obiettivo di questo test è appunto quello di misurare tale ritardo ed osservare se al variare del ritardo artificiale introdotto mantiene una certa costanza. Il ritardo inserito è perfettamente noto e in uno scenario di test come quello in esame è assolutamente considerabile come l'esatta latenza di rete (con al più un

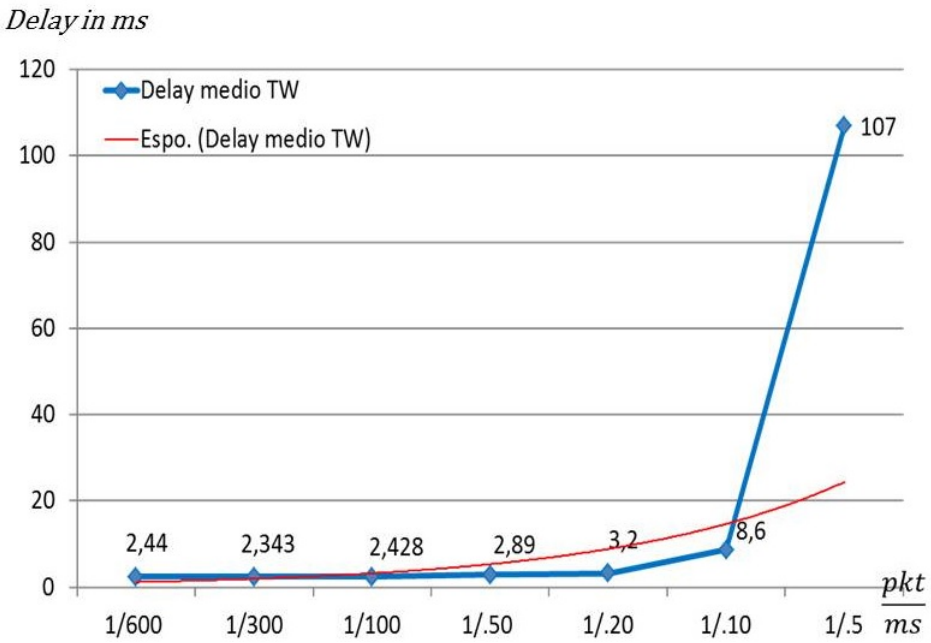


Figura 6.3: Test 1.

$Tx\_rate = \frac{1\ pkt}{X\ ms}$	# Pkt. / min.	Throughput	Packet Loss	DMTW
1/600	100	6,4 KB/min.	NO	2,44
1/300	200	12,8 KB/min.	NO	2,343
1/100	600	38,4 KB/min.	NO	2,428
1/50	1200	76,8 KB/min.	NO	2,89
1/20	3000	192 KB/min.	NO	3,2
1/10	6000	384 KB/min.	NO	8,6
1/5	12000	768 KB/min.	NO	>100
<1/5	--	--	SI	--

Figura 6.4: Tabella Test 1.

imprecisione di alcuni microsecondi).

Questa prova è la più significativa. Mediante l'utilizzo dell'ANUE è possibile apprezzare in modo chiaro l'effettivo ritardo introdotto dal software in merito all'analisi del DMTW. Questo test permette di trarre alcune importanti conclusioni sull'effettiva bontà dei dati raccolti e , più in generale, sulle reali prestazioni del protocollo PNPM_SLA.

Analisi dei risultati e conclusioni sul test

I valori riportati sull'asse delle ascisse rappresentano il ritardo artificiale introdotto mediane l'ANUE. I valori invece riportati sul piano cartesiano, nella porzione azzurra della barra verticale rappresentano il DMTW misurato al termine dei test sottraendovi il valore del ritardo aggiunto. Tale ritardo viene definito come ritardo software per le motivazioni precedentemente esposte. Di conseguenza la somma dei due valori rappresenta, per ciascuna delle colonne il DMTW.

I risultati ottenuti sono particolarmente interessanti in quanto è possibile osservare che il ritardo software è *relativamente* costante. Si osservano infatti valori che oscillano tra i 2,26 ms ed i 2,44 ms. Naturalmente all'aumentare del numero di prove tali valori avrebbero potuto anche essere maggiori o minori di quelli presentati ma sempre rimanendo all'interno di un intervallo molto vicino.

Se ammettiamo che tale ritardo non è imputabile alla *latenza di rete*, ma piuttosto ai fattori hardware, software ed algoritmici è possibile sostenere che tale ritardo, essendo costante, è sottraibile dal calcolo del DMTW.
Naturalmente sarebbe un errore valutare di sottrarre tanto il minimo quanto il massimo dei valori ottenuti dai test. Infatti è comunque necessaria cautela vista una certa variabilità nei risultati fin qui ottenuti. Per esempio una soluzione ragionevole può essere quella di considerare il maggiore tra i valori registrati e, per sicurezza, aggiungervi un ulteriore margine percentuale.
Ad ogni modo, a valle dei dati attualmente disponibili, un *lower bound* accettabile, ovvero un valore che si potrebbe sottrarre al delay misurato potrebbe essere pari a 2 ms.

A questo punto, sottraendo tale valore ai dati ottenuti, è possibile affermare che, nelle condizioni sperimentali considerate, si apprezza una precisione nelle misure del DMTW pari o inferiore al millisecondo.
Tenuto conto dei margini di miglioramento sia nel software che in alcuni as-

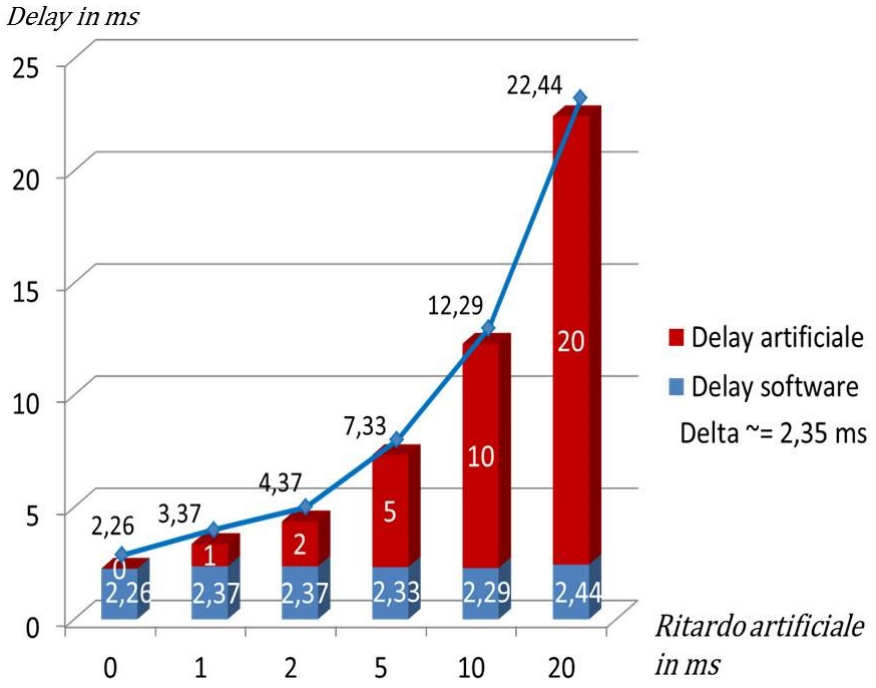


Figura 6.5: Test 2.

petti dell'algoritmo e avendo la possibilità di condurre test più approfonditi è auspicabile che tale imprecisione possa diminuire ulteriormente.

6.3.3 Test 3

Sul Raspberry pi non sono stati ancora svolti veri e propri test. Ad ogni modo la prova presa in esame è simile al primo test. La principale differenza è rappresentata dalla sonda client che in questo secondo scenario è il Raspberry Pi e dal numero limitato di informazioni raccolte.

Analisi dei risultati e conclusioni sul test

Vengono nuovamente analizzate le metriche del round trip time virtuale e della packet loss all'aumentare del rate tra i due apparati. Viste le prime informazioni in merito a tali metriche sarebbe utile approfondire le ricerche su tale apparato. Non è infatti da escludere la possibilità di riuscire ad ottenere un delay medio relativamente costante anche su tali apparati. A tal fine sarebbe però necessario sia ottimizzare il codice in modo da renderlo computazionalmente più leggere sia operare alcune migliorie sull'algoritmo stesso tali da ridurre il numero di iterazioni eseguite durante il suo funzionamento.

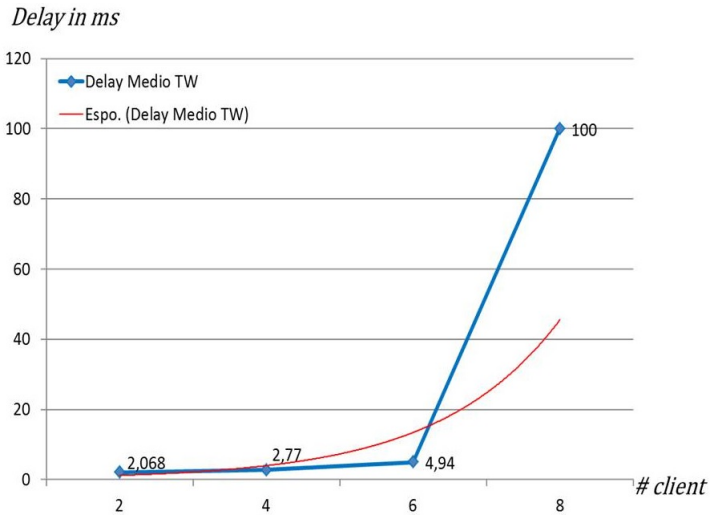


Figura 6.6: Test 3.

Sonda Server	TX Rate	# Pkt. / min.	Throughput
HP Z600	1/50	1200	76,8 KB/min.

# Sonde Client	TX Rate Client (1 pkt / X ms)	# Pkt. / min.(Client)	Throughput (client)	Packet Loss	Delay Medio TW
2	1/100	600	38,4KB/min.	NO	2,068
4	1/200	300	19,2KB/min.	NO	2,77
6	1/300	200	12,8KB/min.	NO	4,94
>6	--	--	--	SI	>100

Figura 6.7: Tabella Test 3.

Ciò nonostante i primi risultati schematizzati, nella tabella in figura 6.7, sono incoraggianti e suggeriscono che anche un apparato simile al Raspberry pi, se non il Rapsberry stesso, possa essere seriamente preso in considerazione come sonda PNPM_SLA definitiva.

Capitolo 7

Conclusioni e possibili sviluppi futuri

Nello sviluppo del protocollo PNPM_SLA, a valle delle ricerche svolte nei primi capitoli della tesi, si è deciso di puntare sulla riduzione del tempo di elaborazione dei pacchetti al fine di ottenere misure di prestazioni simili a quelle ottenute da altri protocolli con finalità analoghe. Inoltre in PNPM_SLA sono presi in esame apparati COTS in quanto uno dei vincoli primari è quello economico. D'altra parte la capacità di utilizzare un protocollo di monitoraggio che non sia di proprietà di terze parti costituirebbe già di per se un importante risparmio.

Rispetto alla situazione di partenza sono stati compiuti diversi passi avanti. Il nuovo protocollo è stato progettato, implementato e testato con risultati incoraggianti in previsione di ulteriori sviluppi. Ad esempio sarà utile renderlo più flessibile permettendo a sonde diverse di svolgere dei test aventi un differente tasso di trasmissione.
Inoltre tutte le problematiche legate alla sicurezza non sono state prese in considerazione in questa fase.

I successivi passi da compiere con l'obiettivo di migliorare i risultati fin quì raggiunti sono:

- Definire in modo stabile quali apparati si vorranno utilizzare effettivamente sul campo;
- Ottimizzare il software ad oggi sviluppato risolvendo, ad esempio, alcuni possibili problemi di interoperabilità riscontrabili tra piattaforme differenti;

- Anche se i risultati raggiunti nei test sono incoraggianti è necessario rendere dove possibile l'algoritmo stesso più flessibile ed al tempo stesso valutare strategie per renderlo più leggero da un punto di vista computazionale;

- Avendo considerato i numerosi punti di forza presentati in merito ai driver di I/O accelerato andrebbe valutata la possibilità di un loro impiego effettivo sulla sonda server;

- Quindi ripetere ed approfondire i test nelle nuove condizioni.

In conclusione, i risultati ad oggi raggiunti attraverso l'impiego di PNPM_SLA permettono di affermare che l'idea di adottare apparati COTS al fine di ottenere misure di prestazioni di rete è valida. Tuttavia, visto il numero di aspetti critici da considerare quali ad esempio la sincronizzazione, la precisione, la capacità degli apparati coinvolti, etc. è necessario approfondire, anche attraverso i punti sopra elencati, il lavoro svolto al fine di ottenere i risultati realmente desiderati.

Bibliografia

[TI] Telecom Italia, *http://www.telecomitalia.com/tit/it.html*. 1.1

[PNPM] A. Capello, M. Cociglio, L. Castaldelli. *A packet based method for passive performance monitoring draft-tempia-opsawg-p3m-03, February 2013*. 4.1

[LRN1] Luigi Rizzo Netmap, *http://info.iet.unipi.it/ luigi/netmap/*. 3

[IPSLA] M. Chiba, A. Clemm, S. Medley, J. Salowey, S. Thombare, E. Yedavalli, *http://www.rfc-editor.org/rfc/rfc6812.txt, Cisco Systems January 2013*. 3.1

[TCP DUMP] TCP DUMP, *http://www.tcpdump.org/*. 2.1.3

[SNORT] Snort, *http://www.snort.org/*. 2.1.3

[libpcap] netFPGA+, *http://www.linuxfromscratch.org/blfs/view/svn/basicnet/libpcap.html*. 2.1.3

[OSPU] Sistemi Operativi più utilizzati, *http://en.wikipedia.org/wiki/Usage_share_of_operating_syste* 2.1.3

[PMSR1] Pedro Marìa Santiago del Rìo. *Internet Trafic Classification for High-Performance and Off-The-Shelf Systems, PH.d. Thesis, Madrid 2013*. 2.1.3

[NAPI] New API, *http://www.linuxfoundation.org/collaborate/workgroups/networking/napi*. 2.1.3

[PMSR2] Pedro Marìa Santiago del Rìo, *Internet Trafic Classification for High-Performance and Off-The-Shelf Systems, PH.d. Thesis, Madrid 2013, pp 31-35*. 2.2.2

[netFPGA] netFPGA, *http://netfpga.org/*. 2.3.2

[Intel 82580] Intel 82580, *http://www.intel.com/content/dam/doc/datasheet/82580-eb-db-gbe-controller-datasheet.pdf*. 2.3.2

[RDC] Luigi Rizzo, Luca Deri, Alfredo Cardigliano. *10 Gbit/s Line Rate Packet Processing Using Commodity Hardware: Survey and new Proposals*. 2.3.2

[LRN12] Luigi Rizzo. *netmap: a novel framework for fast packet I/O, Università di Pisa, Italia, Proceedings of the 2012 USENIX Annual Technical Conference, June 2012*. 2.3.3

[LD05] Luca Deri. *nCap: Wire-speed Packet Capture and Transmission, ntop.org, Univerisità di Pisa, Italia, 2005*. 2.3.3

[netFPGA+] netFPGA+, *http://netfpga.org/10G_specs.html*.

[LD] Luca Deri, *http://www.ntop.org/pf_ring/dna-vs-netmap/*. 2.3.3

[VMM] Víctor Moreno Martínez, *Development and evaluation of a low-cost scalable architecture for network traffic capture and storage for 10Gbps networks, Master's Thesis, Universidad Autónoma de Madrid Escuela Politécnica Superior, 2012*. 2.3.3

[NTP] D. Mills, J. Martin,J. Burbank,W. Kasch. *http://www.rfc-editor.org/rfc/rfc5905.txt June 2010*. 2.4.1

[SNTP] David Mills. *http://tools.ietf.org/html/rfc2030*. 2.4.1

[IEEE 1588] Standard IEEE 1588, *http://www.nist.gov/el/isd/ieee/ieee1588.cfm*. 2.4.3

[RG] Rick Ratzel, Rodney Greenstreet. *An introduction to PTP and its significance to NTP practitioners*. 2.4.3

[DSA] Darryl Veitch, Satish Babu, Attila Pàasztor. *Robust Synchronization of Software Clocks Across the Internet*. **??**

[BCRW] Benoit Claise, Ralf Walter. *http://etutorials.org/Networking/network+management*
3.2

[IPSLAprecision] *http://www.cisco.com/en/US/docs/ios/ipsla/command/reference/sla_02.h*
3.2

[IPSLAarchitecture] *http://www.cisco.com/en/US/technologies/tk648/tk362/tk920/technolo*
3.2