UNIVERSITÀ DI PISA
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Specialistica in Tecnologie Informatiche

Tesi di Laurea

ARCHITETTURA DISTRIBUITA PER IL MONITORAGGIO DI RETE NEAR-REALTIME

RELATORE

Dott. Luca Deri

AUTORE

Dott. Andrea Parrella

CONTRORELATORE

Prof. Marco Danelutto

Anno Accademico 2009/2010

Indice

1 Introduzione e Motivazioni **1**

1.1 Qualità del servizio 2

1.2 L'importanza del monitoraggio di rete 4

1.3 Analisi del livello applicativo 6

1.4 Monitoraggio attivo o passivo 6

1.5 Dislocazione delle sonde per il monitoraggio 7

1.6 Limiti dell'hardware esistente 8

1.7 Obiettivi del lavoro 8

1.8 Struttura della tesi 13

2 Stato dell'Arte **14**

2.1 Architetture di monitoraggio 14

2.1.1 IPMON 14

2.1.2 Nprobe 16

2.1.3 NG-MON 20

2.1.4 Gigascope 25

2.1.5 DOME 25

2.1.6 Interrogazione continua e storica su stream 28

2.1.7 Comparazione delle architetture 31

2.2 Componenti e tecnologie 32

2.2.1 Sincronizzazione temporale 32
2.2.2 Acquisizione del traffico 33
2.2.3 Aggregazione in flussi 36
2.2.4 Archiviazione ed interrogazione dei dati 38
2.2.5 Tecniche di data delivery su web 41

3 L'Architettura DRT-Mon **47**

3.1 Descrizione generale 48
3.2 Sonda di rete 52
3.2.1 Strutture dati 54
3.2.2 Descrizione delle attività 58
3.2.3 Selezione del traffico 61
3.2.4 Metriche utilizzate 63
3.2.5 Memorizzazione dei dati storici 65
3.3 Collettore 66
3.4 Console di monitoraggio 68
3.5 Messaggi 69
3.5.1 Sonda-Collettore 69
3.5.2 Collettore-Console 74

4 Validazione **78**

4.1 Dettagli sulla realizzazione 78
4.1.1 Sonda 78
4.1.2 Collettore 83
4.1.3 Console 85
4.2 Valutazione delle prestazioni 86
4.2.1 Sonda 88
4.2.2 Collettore 92

4.3 Confronto con lo stato dell'arte 92

5 Conclusioni **97**

5.1 Lavoro futuro 99

Elenco delle figure **99**

Elenco delle tabelle **102**

Elenco dei listati **103**

Acronimi **104**

Bibliografia **105**

Capitolo 1

Introduzione e Motivazioni

Le reti di computer stanno cambiando velocemente. Se fino a pochi anni fa la rete veniva usata essenzialmente per visitare pagine web e scambiare email o file, ora le reti vengono sfruttate in gran parte delle attività che svolgiamo tutti i giorni. La telefonia, le operazioni bancarie, le prenotazioni di biglietti aerei ecc. sfruttano tutte le potenzialità che le reti di computer offrono. Già nel 2001, analizzando i progressi nelle tecnologie di trasmissione e il comportamento degli utenti, è stato previsto un aumento del traffico di rete con un andamento simile a quello previsto dalla legge di Moore per i semiconduttori [1], ossia che il traffico internet sarebbe raddoppiato ogni anno. Stime più recenti [2, 3] prevedono che il traffico internet quintuplicherà tra il 2008 e il 2013. Questa crescita è dovuta a diversi fattori tra cui gli innumerevoli dispositivi che pervadono le abitazioni e che fanno uso della connessione di rete come ad esempio: televisori, pc, smartphone, e-book reader e media center.

Come mostrato nella figura 1.1, per il traffico consumer, la condivisione dei file tramite software peer-to-peer rappresenta la percentuale maggiore del traffico internet ma presto sarà sorpassato dal traffico multimediale e in particolare dai flussi video. Il cambiamento nel tipo di traffico che viaggia sulle reti impone un cambiamento anche sui requisiti che queste devono avere. Se in passato ci si poteva accontentare di un servizio di rete best-effort questo non è più vero.

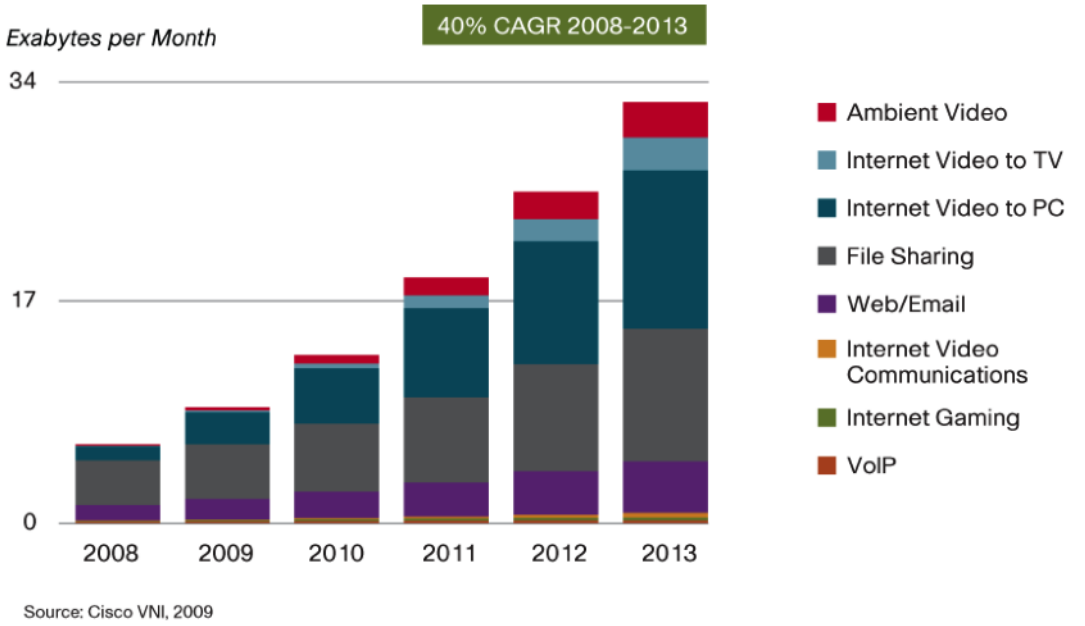


FIGURA 1.1: Cisco's Global Consumer Internet Traffic Forecast

1.1 Qualità del servizio

Per far si che la rete risponda alle esigenze delle applicazioni, è diventata molto importante la gestione della qualità del servizio (QoS) [4], cioè la possibilità di misurare e assegnare le risorse a un determinato tipo di traffico piuttosto che a un altro. Le metriche principali cui ci si riferisce con QoS sono:

Disponibilità: la percentuale di tempo in cui la rete è funzionante.

Throughtput: l'effettiva velocità di trasferimento dei dati.

Packet loss: la percentuale di pacchetti persi (di solito a causa di congestione).

Latenza: il tempo che impiega un dato per arrivare dalla sorgente alla destinazione.

Jitter: la varianza della latenza.

Come si vede nelle ultime due righe della tabella 1.1 il traffico video che sarà predominante nei prossimi anni è anche quello più esigente dal punto di vista della QoS.

1. Varied sensitivities of network data types

Traffic type	Sensitivities			
	Bandwidth	Loss	Delay	Jitter
Voice	Very low	Medium	High	High
E-commerce	Low	High	High	Low
Transactions	Low	High	High	Low
E-mail	Low	High	Low	Low
Telnet	Low	High	Medium	Low
Casual browsing	Low	Medium	Medium	Low
Serious browsing	Medium	High	High	Low
File transfers	High	Medium	Low	Low
Video conferencing	High	Medium	High	High
Multicasting	High	High	High	High

* Complex contents may include audio and video clips and fast animations.

Source: CQOS Inc.

Tabella 1.1: Requisiti di QoS per le principali applicazioni

Gli autori di un'analisi sul traffico di rete dal campus dell'università di Calgary verso YouTube [5] hanno verificato che il 24% delle transazioni video non viene completata e indicano tra le cause le scarse prestazioni di rete.

Con la diffusione delle console per videogiochi di settima generazione, sono sempre di più gli utenti che utilizzando videogiochi multiplayer real-time sfruttano la connessione di rete per sfidare altri giocatori provenienti da ogni parte del mondo. La banda di cui necessitano questi videogiochi è minima ma sono molto sensibili all'aumento di latenza [6]. Le misure effettuate su un gioco di simulazione automobilistica hanno dimostrato che una latenza superiore ai 100 ms rende irrealistica la simulazione.

La telefonia su IP (VoIP) [7], che sta velocemente sostituendo la telefonia tradizionale permettendo un'ottimizzazione nell'uso delle risorse, è un altro dei settori in cui la qualità del servizio ha un'importanza fondamentale. Il passaggio da connessioni a commutazione di circuito, dove tutte le risorse sono allocate per l'intera durata della comunicazione, a connessioni a commutazione di pacchetto richiede un'attenta gestione

delle priorità nell'instradamento del traffico. I fattori che degradano la qualità dell'audio sono: la perdita di pacchetti oltre una certa soglia, la latenza e il jitter. L'ITU-T Recommendation G.114 raccomanda una latenza non superiore ai 150 ms per far si che gli utenti non percepiscano una differenza nella qualità della comunicazione rispetto alla telefonia tradizionale. Anche la telefonia mobile con l'introduzione dell'IP Multimedia Subsystem (IMS) [8] si sta spostando su reti di comunicazione a commutazione di pacchetto proponendo complesse architetture per la convergenza di telefonia fissa e mobile su rete IP con esigenze analoghe a quelle descritte per il VoIP.

Un settore di nicchia dove però si fanno grandi investimenti nella qualità del servizio di rete è quello del trading in borsa automatizzato. Utilizzando dei server che seguono in tempo reale l'andamento della borsa, vengono applicati degli algoritmi per eseguire la compravendita ad alta frequenza dei titoli. Con questo tipo di attività l'azienda la cui piattaforma è in ritardo di 5 ms rispetto a un concorrente può perdere fino a 20 milioni di dollari di profitti. Con 100 ms di ritardo non ha più senso utilizzare questi sistemi automatici. Secondo TABB Group [9] il 25% degli investimenti in queste piattaforme, che per il 2011 è stimato in 305 milioni di dollari, è diretto al miglioramento della connettività di rete.

Per avere garanzie sulla qualità del servizio vengono stipulati dei contratti chiamati Service Level Agreement (SLA) [10] tra gli utilizzatori della rete e i fornitori del servizio. Gli SLA definiscono sotto quali condizioni il fornitore s'impegna a fornire un certo livello di qualità del servizio. Ad esempio: il provider si impegna a fornire un collegamento la cui latenza sia inferiore a 50 ms se l'utilizzatore invia al massimo 200 pacchetti al minuto. L'offerta di garanzia sulla qualità del servizio è una discriminante sulla scelta tra provider concorrenti e offre quindi la possibilità, per il provider, di avere maggiori introiti rispetto alla fornitura del servizio best-effort.

1.2 L'importanza del monitoraggio di rete

Internet è stata progettata applicando il principio end-to-end che consiglia di evitare l'inserimento all'interno del core network di funzionalità del livello applicativo [11].

L'applicazione di questo principio ha determinato il successo della rete internet poiché permette di avere una ridotta complessità del core network e dà la possibilità di progettare nuove applicazioni senza dover modificare i protocolli e gli apparati di rete già esistenti. D'altra parte questo principio limita la possibilità di differenziare il servizio di rete in base alle esigenze che hanno le diverse applicazioni. Infatti i nodi del core network non sono a conoscenza del tipo di traffico che li attraversa e quindi non possono applicare politiche diverse in base al contenuto applicativo dei pacchetti TCP/IP. Sono quindi necessari strumenti in grado di mostrare in ogni istante e con sufficiente dettaglio cosa sta succedendo sulla rete. Di seguito sono riportate le applicazioni più significative del monitoraggio di rete [12]:

Rispetto degli SLA

Per riuscire a verificare se la rete stia effettivamente fornendo le prestazioni specificate negli SLA, sono necessari degli strumenti di monitoraggio che catturino e analizzino il traffico in diversi punti della rete fornendo le misure per le metriche specificate nei contratti. Per risalire alle cause dei rallentamenti sono necessarie misure accurate e dettagliate del traffico e la generazione di matrici di traffico che forniscano informazioni su come il traffico fluisce all'interno della rete.

Pianificazione degli investimenti

La pianificazione degli investimenti nell'infrastruttura di rete richiede previsioni a lungo termine del traffico. Le previsioni si basano sull'esperienza degli amministratori di rete o su modelli matematici [13] ed entrambi hanno bisogno di strumenti di misura che forniscano i dati su cui fare le analisi. Purtroppo le previsioni a lungo termine non sono sempre sufficienti perché i comportamenti degli utenti possono variare rapidamente a causa di eventi eccezionali o per la diffusione di nuove applicazioni che sfruttano la rete in modo diverso. È quindi necessario un monitoraggio continuo per reagire tempestivamente a questi cambiamenti.

Traffic Engineering

Oltre alla pianificazione degli investimenti, la corretta distribuzione del traffico

sulla rete permette di migliorare la qualità del servizio diminuendo la congestione sui link più utilizzati o riservando l'uso di link qualitativamente migliori al solo traffico con alta priorità. L'attività di valutazione delle prestazioni e ottimizzazione prende il nome di Traffic Engineering [14].

Resoconti per gli utenti

I provider di solito hanno la necessità di generare resoconti sul traffico per gli utenti, specialmente nei casi in cui il pagamento del servizio è legato alla quantità di traffico scambiato o alla durata della connessione.

Rilevamento di attacchi

Tramite il monitoraggio di rete è possibile rilevare la diffusione di worm oppure attacchi di tipo denial-of-service (DoS) [15] con cui si tenta di bloccare il normale accesso degli utenti ai servizi di rete. Risalendo ai sistemi o alle reti origine degli attacchi è possibile bloccarli e intraprendere azioni legali.

1.3 Analisi del livello applicativo

Per la caratterizzazione del traffico IP i sistemi di monitoraggio devono, per quanto possibile, analizzare tutti i livelli della pila protocollare. Infatti, se non si ispeziona il livello applicativo non è possibile distinguere correttamente quale tipo di traffico sta passando sulla rete. Per esempio se ci si basasse solamente sulla porta TCP 80 per identificare il traffico web, s'includerebbero nell'analisi anche altri tipi di traffico come la condivisione di file peer-to-peer e la telefonia su IP che sfruttano tale porta per evitare il blocco da parte dei firewall.

1.4 Monitoraggio attivo o passivo

Per fare le misure sul traffico di rete si possono adottare due metodologie di monitoraggio: attivo o passivo. Il monitoraggio attivo consiste nell'inviare sulla rete pacchetti costruiti ad-hoc ed eseguire le misure su di essi. Questo metodo permette di fare misure

dal punto di vista del traffico (ad esempio latenza tra due punti, percentuale di pacchetti persi ecc.) che sono più complicate da fare con metodi passivi. Il monitoraggio passivo invece si limita ad analizzare il traffico che passa sulla rete senza l'invio di dati aggiuntivi per la misura. Il traffico che il metodo attivo invia sulla rete può modificare il comportamento della rete stessa alterando la misura che si sta eseguendo. Inoltre i pacchetti usati per il monitoraggio attivo (di solito ICMP) sono di solito considerati potenzialmente dannosi perché possono trasportare minacce per la rete e per questo motivo vengono rallentati o del tutto bloccati dai firewall. D'altra parte la possibilità di avere una misura dal punto di vista del traffico permette di calcolare metriche non facilmente misurabili con metodi passivi come il percorso di un pacchetto o la latenza di trasmissione end-to-end. Con i metodi passivi, anche disponendo di sonde distribuite sulla rete, c'è bisogno di identificare i pacchetti in transito per effettuare le misure.

1.5 Dislocazione delle sonde per il monitoraggio

L'analisi del traffico di rete può essere fatta dagli apparati di rete su cui passa il traffico (router o switch) oppure da sonde esterne. L'uso della sonda esterna permette di alleggerire il carico degli apparati evitando che questo vada a influenzare le misure. Infatti l'operazione di misura può rallentare l'instradamento dei pacchetti e quindi influenzare la misura stessa. Inoltre gli apparati di rete di solito usano software proprietario e non permettono l'implementazione di nuove soluzioni per il monitoraggio.

Per introdurre un nuovo sistema di monitoraggio come l'architettura DRT-Mon proposta nella tesi è quindi necessario usare sonde esterne verso cui viene replicato il traffico. La replicazione del traffico è un'operazione poco costosa perché viene fatta contemporaneamente all'instradamento. Esistono essenzialmente due modi per effettuare la replicazione:

1. *Port mirroring:* gli switch vengono configurati per replicare il traffico di un insieme di porte su una porta di monitoraggio. In questo caso la porta di monitoraggio deve avere una velocità sufficiente per trasmettere tutto il traffico da replicare.

2. *Network tap:* sono dei dispositivi con tre porte (ingresso, uscita e monitor) che si inseriscono sul collegamento tra due dispositivi di rete e che oltre a ritrasmettere il traffico che ricevono ne inviano una copia sulla porta di monitoraggio. Spesso questi dispositivi non si limitano alla sola replicazione ma possono eseguire una prima elaborazione o bilanciare il traffico su più dispositivi di monitoraggio.

1.6 Limiti dell'hardware esistente

Nella tesi vengono prese in considerazione reti ad alta velocità dove il tempo di inter-arrivo dei pacchetti è così basso che si hanno a disposizione pochi cicli di clock per l'acquisizione e l'elaborazione di ogni singolo pacchetto. In "Monitoring very high speed links" del 2001 [16] è stato calcolato che su un link a 10 Gbps, considerando pacchetti con una dimensione media di 300 byte, arriva un nuovo pacchetto ogni 240 ns. Sul processore più veloce che era disponibile, si potevano eseguire solamente 360 istruzioni per ogni pacchetto. Attualmente, con processori a 3.2 Ghz si hanno a disposizione 768 cicli di clock per l'elaborazione di ogni pacchetto. Per quanto riguarda la memorizzazione del traffico, se si volessero memorizzare tutti i pacchetti si dovrebbero scrivere su disco oltre 1 GB di dati al secondo. Questa mole di dati non può essere gestita senza effettuare un'aggregazione per diminuirne la quantità. I bus comunemente in uso non arrivano oltre i 6 Gb/s (SATA 3 e Serial Attached SCSI) e il throughput reale dei dischi è molto inferiore. Inoltre per memorizzare una sola ora di traffico sarebbero necessari oltre 4 TB di spazio.

1.7 Obiettivi del lavoro

L'evoluzione delle reti dettata dalle esigenze delle applicazioni multimediali che stanno prendendo sempre più piede ha fatto scaturire l'esigenza da parte degli amministratori di rete di avere strumenti per il monitoraggio in grado di analizzare con sufficiente dettaglio e tempestività lo stato della rete. Nel lavoro di tesi viene proposta l'architettura DRT-Mon che si prefigge di raggiungere gli obiettivi analizzati di seguito.

Architettura distribuita

Vista la complessità e l'estensione delle reti risulta impossibile avere un punto centrale di monitoraggio in cui far confluire tutto il traffico da monitorare. Se si volesse trasportare tutto il traffico da monitorare verso la console di amministrazione sarebbe necessaria una rete con prestazioni maggiori di quella che si sta monitorando rendendo impossibili le misure senza utilizzare una rete ad alte prestazioni dedicata al solo monitoraggio. Inoltre non si riuscirebbero ad assegnare timestamp sufficientemente precisi, necessari per effettuare misure come il jitter, visto che la rete di monitoraggio introdurrebbe degli errori troppo grandi. Per non incorrere in questi problemi è necessario eseguire le misure lì dove passa il traffico distribuendo le sonde nella rete. L'architettura distribuita permette inoltre di avere sonde con risorse computazionali e capacità di memorizzazione inferiori alla soluzione centralizzata facendo diminuire i costi. Le misure effettuate sui vari tronchi di rete devono comunque fornire una visione globale sullo stato della rete fornendo informazioni aggregate ma dando comunque la possibilità di avere i dettagli su quello che si sta monitorando.

Granularità delle analisi

La necessità di diminuire la quantità di dati da gestire porta ad aggregare i pacchetti in flussi e a presentare i risultati solamente al termine di un flusso. Questo approccio non permette di sapere esattamente cosa sia successo in un determinato intervallo di tempo perché non si riesce a sapere qual è stato l'andamento del traffico durante la durata del flusso; se ne conosce solo il valore medio. Per avere un maggior dettaglio, con protocolli come NetFlow, è possibile specificare un tempo massimo oltre il quale il flusso viene esportato verso il collettore ma comunque questo intervallo non scende sotto il minuto. In questo modo non si riescono ad identificare anomalie di breve durata che invece hanno grande impatto sulle prestazioni della rete.

Per gli stessi motivi sono da escludere anche i sistemi di monitoraggio che utilizzano il campionamento [17] per produrre le statistiche. Questi ultimi riescono

a misurare solo i flussi "elephant" che rappresentano l'80% del traffico mentre non danno nessuna informazione sui restanti flussi "mouse" che generano poco traffico. Come dimostrato in [18], i metodi utilizzati per il campionamento non permettono agli algoritmi di rilevamento delle anomalie di funzionare in modo corretto. Inoltre nel caso in cui il sistema di monitoraggio venga utilizzato per la rilevazione di attacchi, l'attaccante può creare traffico ad-hoc per sfuggire alle analisi. Nella tesi ci si pone come obiettivo la risoluzione temporale di un secondo che sebbene sia ancora un tempo troppo lungo per rilevare tutte le anomalie, dà maggiori informazioni sullo stato della rete rispetto alle altre soluzioni esistenti.

Presentazione dei risultati near-realtime

Come detto in precedenza, le reti vengono sempre più usate per trasportare i dati di applicazioni real-time. Un calo delle prestazioni della rete, anche di breve durata, può provocare ingenti perdite economiche o rendere inutilizzabili alcune applicazioni. I risultati delle misure effettuate su queste reti devono quindi essere disponibili appena si verifica un evento in modo da avere sistemi automatici che reagiscano alle nuove condizioni per garantire che la rete offra la qualità del servizio richiesta. I sistemi di monitoraggio comunemente usati come NetFlow [19] producono i risultati delle misure minuti dopo che l'evento si è verificato lasciando gli amministratori senza informazioni sullo stato della rete. Vista la grande quantità di dati da analizzare è molto difficile avere i risultati esattamente nell'istante in cui si verifica un evento. Il sistema di monitoraggio introduce una certa latenza e per questo si parla di monitoraggio near-realtime. È importante che la latenza introdotta dal sistema di monitoraggio sia contenuta in modo da minimizzare il ritardo con cui gli amministratori o gli strumenti automatici di gestione di rete ricevono le misure e reagiscono ai nuovi eventi. Quanto detto sulla granularità delle analisi e sull'aggregazione contrasta con la necessità di avere un sistema di monitoraggio real-time. L'obiettivo è di trovare un compromesso accettabile tra la volontà di effettuare misure esatte e il ritardo con cui l'amministratore di rete riceve le informazioni.

Impatto sul traffico

Applicando il principio di indeterminazione di Heisenberg al monitoraggio di rete si può dire che usare del traffico ad-hoc (come fanno i sistemi di monitoraggio attivo) per valutare le prestazioni della rete modifica il comportamento della rete stessa e quindi non è possibile sapere quale siano le reali prestazioni. Tra le soluzioni che fanno monitoraggio passivo vanno escluse anche quelle che utilizzano gli apparati di rete come router e switch per calcolare le statistiche perché di solito le risorse utilizzate per l'analisi vengono sottratte alle risorse utilizzate per l'instradamento del traffico provocando un peggioramento delle prestazioni della rete. Quindi si vuole un sistema che faccia monitoraggio passivo e che minimizzi l'impatto degli strumenti di monitoraggio sulla rete da monitorare.

Selezione dei dati

Il traffico interessante per l'amministratore o per i sistemi automatici di gestione di rete variano nel tempo. In alcuni momenti può essere necessario visualizzare solamente le statistiche sul traffico Web, in altri momenti solamente le statistiche sul traffico SMTP e così via. L'architettura dovrà avere un sistema per selezionare il traffico che deve contribuire alle statistiche. I criteri di selezione devono poter essere modificati dinamicamente ovvero senza interrompere il funzionamento del sistema.

Consultazione dei dati storici

I dati raccolti dal sistema di monitoraggio devono comunque essere conservati e indicizzati permettendo la consultazione dello storico e la generazione di statistiche differenti da quelle calcolate in precedenza. Questo perché gli amministratori di rete hanno la necessità di analizzare l'andamento del traffico nel passato per pianificare gli investimenti infrastrutturali o per indagare, cambiando il livello di dettaglio, su situazioni anomale che si sono verificate. Questa attività deve essere indipendente da quella di acquisizione e analisi real-time in modo che durante la consultazione dello storico non venga interrotto il monitoraggio.

Scalabilità

Le reti sono in continua evoluzione quindi un buon sistema di monitoraggio deve essere in grado di scalare con l'aumento della loro velocità e della loro complessità. Inoltre così come gli upgrade e le modifiche alla rete non devono comportare interruzioni del servizio, anche il sistema di monitoraggio deve poter essere aggiornato senza interrompere le operazioni di misura e presentazione dei risultati. L'aggiunta di nuove sonde deve avvenire quindi in modo dinamico mentre il sistema è in esecuzione.

Tolleranza ai guasti

Può avvenire che uno dei componenti del sistema di monitoraggio o una parte della rete con cui vengono trasportati i risultati smetta di funzionare. In questo caso il sistema deve segnalare il malfunzionamento continuando comunque a produrre le statistiche sul resto della rete. Vanno evitati i *single point of failure* (SPF), cioè che esistano dei componenti che in caso di guasto interrompano il funzionamento dell'intero sistema.

Privacy

Sulle reti transitano molti dati sensibili che possono essere sfruttati da terzi per tracciare profili degli utenti o per perpetrare truffe. Viste anche le leggi vigenti in materia di privacy è importante che i sistemi di monitoraggio non diffondano verso terzi i dati acquisiti dalle sonde. I componenti dell'architettura devono autenticare le entità con cui comunicano in modo da essere sicuri che i destinatari siano autorizzati ad avere quei dati. Inoltre andrebbero acquisiti e mantenuti solamente i dati necessari al monitoraggio di rete, eliminando o anonimizzando tutti gli altri.

Commodity hardware

La maggior parte dei sistemi di monitoraggio commerciali utilizza soluzioni con hardware dedicato che sebbene riescano a monitorare senza problemi reti ad alta velocità legano il sistema di monitoraggio all'uso di hardware specifico e di solito

molto costoso. L'approccio che si vuole utilizzare è quello di usare "commodity hardware", cioè hardware standard che vista la larga diffusione è di facile reperibilità, può essere sostituito con hardware equivalente di altri produttori e ha un costo ridotto.

Visualizzazione dei risultati

Così come le metriche devono essere misurate in tempo reale, anche la visualizzazione dei dati deve esserlo. Si vuole che l'interfaccia di visualizzazione sia accessibile da dispositivi eterogenei indipendentemente dal sistema operativo usato o dalle risorse computazionali a disposizione.

1.8 Struttura della tesi

Nella prima parte del capitolo 2 verranno presentate le architetture attualmente utilizzate per il monitoraggio di rete. Nella seconda parte invece verrà presentato lo stato dell'arte per quanto riguarda i componenti necessari alla realizzazione dell'architettura oggetto della tesi. Nel capitolo 3 verrà descritta l'architettura DRT-Mon definita per il lavoro di tesi. Infine nel capitolo 4 sarà presentato il prototipo che è stato realizzato per verificare il funzionamento dell'architettura e saranno valutate le sue prestazioni.

Capitolo 2

Stato dell'Arte

2.1 Architetture di monitoraggio

In questa sezione sono riportate le principali architetture esistenti per il monitoraggio di rete. Nel paragrafo conclusivo (2.1.7) vengono presentate delle tabelle comparative che evidenziano le differenze chiave tra le diverse architetture.

2.1.1 IPMON

Nell'ambito del progetto "Sprint IP Monitoring System" [20] cioè di un sistema di raccolta dati dalla dorsale Internet di Sprint è stato realizzato IPMON che si focalizza su quattro problemi principali:

1. raccolta delle informazioni sui pacchetti che passano sui link ad alta velocità
2. sincronizzazione dei dati catturati
3. manipolazione di grandi insiemi di dati
4. amministrazione del sistema.

IPMON adotta un monitoraggio di tipo passivo ed è stato il primo a permettere la raccolta dei dati in punti differenti della rete dando la possibilità di correlarli tramite l'uso di timestamp ad alta precisione. Per ogni pacchetto catturato, il sistema memorizza i primi 44 byte e il timestamp a 64 bit. Gli orologi dei sistemi sono sincronizzati con

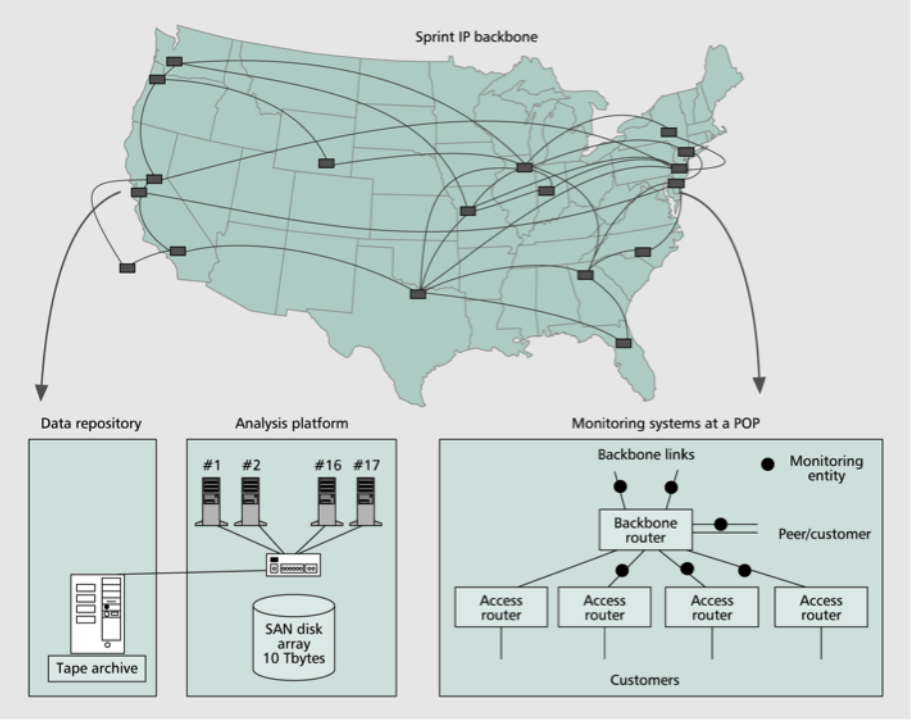


FIGURA 2.1: Architettura IPMON

una precisione di 5 μs usando come riferimento il sistema GPS. La rete su cui è stato installato è formata da link in fibra ottica OC-48 (2,5 Gb/sec) e OC-192 (10 Gb/sec).

La raccolta dei pacchetti dalla rete è fatta tramite PC Linux in cui è installato un array di dischi e un'interfaccia di rete SONET. La scheda di rete usata è la DAG di Endace[1] cui è inviata la copia del traffico tramite uno splitter ottico. L'uso della scheda DAG permette di generare i timestamp per ogni pacchetto, di estrarre i primi 48 byte del frame POS (i 4 byte dell'header POS e i 44 byte della parte iniziale del pacchetto), di preparare e trasferire i record (per un totale di 64 byte) in memoria principale tramite DMA. Ogni volta che 1 MB di dati è stato copiato in memoria principale, la scheda DAG invia un interrupt per comunicare all'applicazione di iniziare la copia dei dati dalla memoria al disco. Se si prende in esame il caso pessimo in cui i pacchetti occupano 40 byte (dimensione minima dei pacchetti TCP), la banda necessaria per trasferire i dati dalla scheda DAG alla memoria principale è di 3968 MB/s per link OC-48. È quindi necessario l'uso di un bus PCI 64 bit/66 Mhz. Considerando come dimensione media

[1] http://www.endace.com/

dei pacchetti 400 byte, è stato calcolato che i requisiti di banda per l'I/O su disco sono il 16% della velocità dei dati. Per un link OC-48 il disco deve scrivere in media 396.8 Mb/s e quindi in IPMON viene usato un sistema RAID con 5 dischi cha raggiunge la velocità di trasferimento di 400 Mb/sec.

Per correlare i dati acquisiti, ogni sistema IPMON deve essere sincronizzato con un segnale di clock globale. I timestamp sono assegnati utilizzando il clock interno delle schede DAG, che permettono di avere una granularità di 59,6 ns, e che vengono sincronizzate tramite un ricevitore GPS. Oltre alla sincronizzazione degli orologi delle schede DAG è necessaria la sincronizzazione degli orologi dei PC che le ospitano e per farlo viene utilizzato il protocollo NTP. I dati raccolti dal sistema IPMON sono compressi e inviati tramite un collegamento dedicato a una struttura centrale dove sono archiviati su nastri magnetici. Con i sistemi installati nel 2001 i dati raccolti in una giornata, occupavano circa 1,2 TB di spazio. L'analisi dei dati, inclusa la correlazione tra i record acquisiti dai sistemi IPMON, è fatta off-line copiandoli dai nastri magnetici a un cluster di 16 PC Linux. I dati sono stati utilizzati per la validazione di diversi progetti di ricerca portati avanti da SprintLabs.

2.1.2 Nprobe

Gli obiettivi nella progettazione dell'architettura Nprobe [21] erano quelli di eseguire la cattura e l'elaborazione del traffico a "wire-rate" senza perdita di pacchetti e di estrarre più informazioni possibili dai vari livelli dello stack di rete. E' stato scelto di eseguire l'elaborazione on-line, cioè senza salvare i pacchetti su disco, perché vista la banda limitata del disco è necessario applicare delle forme di compressione dei dati. Sono stati analizzati tre metodi di compressione:

1. Eliminazione delle informazioni ridondanti (loss-less) che si ripetono in tutti i pacchetti
2. Eliminazione di pacchetti a caso
3. Estrazione dai pacchetti delle sole informazioni interessanti

Il metodo scelto è il terzo e il principale contributo di questo lavoro è il sistema real-time che estrae le informazioni da tutti i livelli della pila protocollare. L'implementazione fa uso di commodity hardware e del sistema operativo GNU/Linux modificato per ottimizzare le scritture su disco. Il firmware della scheda di rete è stato modificato per avere timestamp ad alta risoluzione. Quando la scheda di rete riceve i pacchetti, aggiunge il timestamp e li copia in un buffer in spazio kernel. Ogni macchina per il monitoraggio ha uno o più buffer di ricezione che sono mappati nello spazio utente dei processi i quali si occupano di richiedere l'estrazione delle informazioni ai moduli che eseguono l'elaborazione dei singoli protocolli. Quando tutti i moduli hanno finito l'analisi, il buffer può essere riutilizzato. L'estrazione dei dati da un pacchetto di solito dipende anche dagli altri pacchetti appartenenti alla stessa sessione che hanno attraversato la sonda in precedenza. Viene quindi mantenuto uno stato associato ad ogni sessione elaborata. Quando tutti i pacchetti di una stessa sessione sono stati elaborati i dati estratti vengono copiati in un buffer (log buffer). Una volta accumulati dati sufficienti il buffer viene copiato in un array di dischi.

La scalabilità dell'implementazione si basa sullo striping dei pacchetti tenendo conto del fatto che pacchetti appartenenti a uno stesso flusso devono essere elaborati dallo stesso monitor. Come mostrato in Figura 2.3, tutti i monitor ricevono tutti i dati. Sono le schede di rete che selezionano quali pacchetti far analizzare al monitor calcolando una funzione hash sul valore del risultato dell'operazione XOR tra gli indirizzi IP di sorgente e destinazione. Questo metodo non permette di analizzare il traffico tra due host che supera la capacità di analisi di un singolo monitor.

Per ottimizzare il numero di copie in memoria, i buffer di ricezione in spazio kernel sono mappati nello spazio utente dei processi che effettuano l'analisi. I pacchetti catturati che hanno una continuità funzionale o semantica in uno qualsiasi dei livelli dello stack protocollare vengono raggruppati in una struttura dati chiamata "data association unit" (DAU). Ad ogni DAU è associata la struttura "state storage unit" (SSU) che contiene informazioni sullo stato dell'elaborazione. Poiché queste strutture

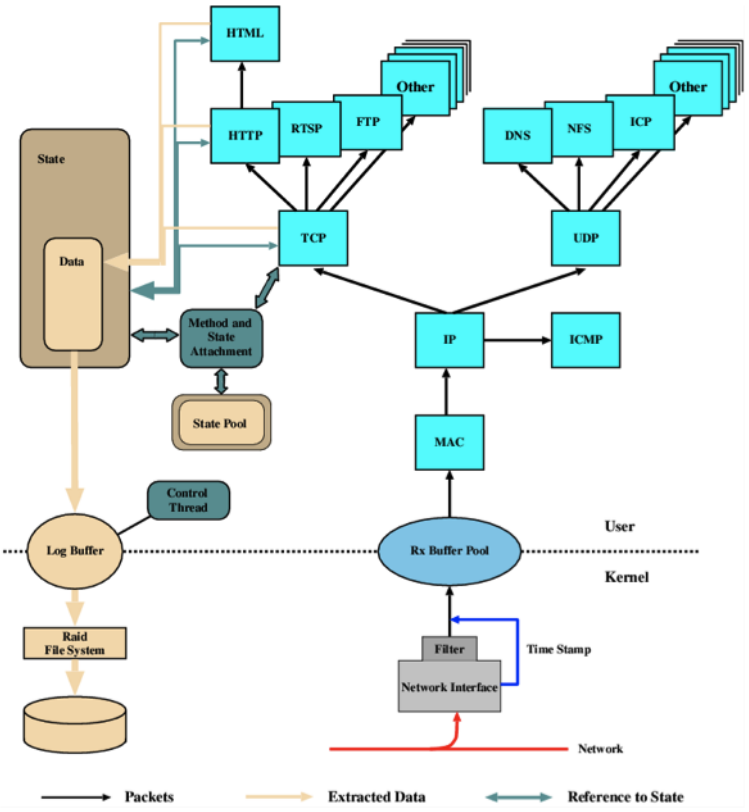


FIGURA 2.2: Architettura Nprobe

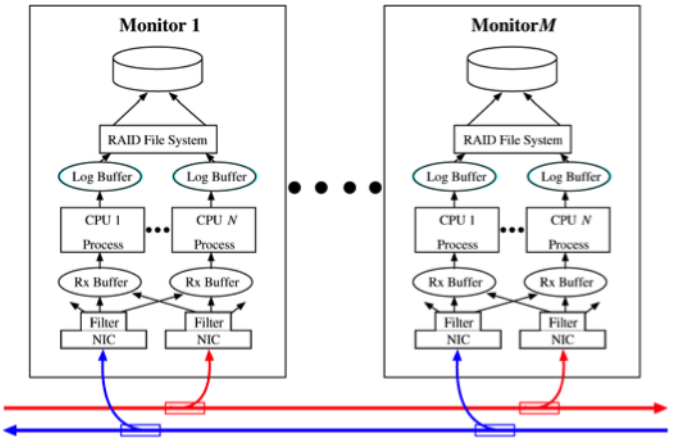


FIGURA 2.3: Striping dei pacchetti

devono essere continuamente allocate e deallocate vengono mantenute in un pool organizzato come stack per massimizzare la località dei riferimenti. I timestamp sono forniti dalla scheda di rete con la precisione di 1 μs e vengono memorizzati in un campo a 32 bit. Siccome il valore si azzera circa ogni 70 minuti, per sapere l'orario esatto il valore del campo timestamp va messo in relazione con l'orario di inizio della sessione che invece è memorizzato in un campo a 64 bit. Per ogni protocollo del livello di rete, di trasporto o applicativo esiste un modulo separato che esegue l'analisi. Questa scelta permette di espandere facilmente il sistema con moduli per l'analisi di altri protocolli. La validazione dell'architettura è stata eseguita in tre fasi per identificare i colli di bottiglia.

Componenti del sistema

È stata valutata la capacità dei sistemi di spostare i dati dalla rete al disco.

Prestazione dei moduli

L'analisi si è concentrata sui moduli TCP/IP, HTTP e HTML. I moduli sono stati valutati in modo indipendente dai componenti per l'acquisizione del traffico in modo da escludere altre possibili cause del degrado delle prestazioni.

Prestazioni globali del sistema

È stato osservato il comportamento dell'intero sistema all'aumentare del volume di traffico.

L'ambiente di test era costituito da un cluster di PC che inviava, tramite l'utility tcpreplay[2], del traffico di rete preparato in precedenza. Tutto il traffico generato è stato fatto passare su un cavo in fibra cui era collegato uno splitter ottico a sua volta collegato alla sonda di monitoraggio. Il PC utilizzato per il monitoraggio aveva le seguenti caratteristiche:

- Processore: Intel Xeon 2.4 Ghz con 512 KB di cache L2
- RAM: 2 GB DDR

[2] http://tcpreplay.synfin.net/

- BUS: 64 bit/66 Mhz PCI
- Controller disco: 3ware 7850-series ATA-RAID
- Dischi: 5 IBM Deskstar 120 GB 7200 RPM
- Scheda di rete: SysKonnect SK-9843-SX

Durante i test effettuati sui componenti del sistema è stato verificato che con pacchetti piccoli le prestazioni sono limitate dal grande numero di interrupt generati dalla scheda di rete. L'analisi delle prestazioni dei moduli ha invece mostrato che le operazioni per mantenere lo stato dei flussi HTML/HTTP limitano il numero dei flussi stessi. Il numero massimo di flussi analizzabili è 44000 con una velocità di 304 Mbps (117 kpps). Le prestazioni generali del sistema, misurate usando traffico reale, raggiungono la velocità di 280 Mbps con picchi massimi di 500 Mbps. Oltre questa soglia il sistema inizia a perdere pacchetti. La causa è stata identificata nella ridotta dimensione dei pacchetti.

2.1.3 NG-MON

NG-MON [22] tenta di proporre una soluzione flessibile, scalabile ed economica per il monitoraggio di rete passivo. Il processo di monitoraggio è stato suddiviso in più fasi allocando uno o più PC per ogni fase in base alle risorse necessarie. I requisiti di cui è stato tenuto conto durante la progettazione sono:

- Architettura distribuita: usando sistemi general purpose è difficile analizzare reti multi-gigabit quindi è necessario distribuire il carico su più PC.
- Cattura senza perdite di pacchetti: per fornire le informazioni richieste dalle varie applicazioni vanno catturati tutti i pacchetti.
- Analisi basata sui flussi: quando si analizza è meglio aggregare le informazioni in flussi per un'elaborazione più efficiente. In questo modo i pacchetti possono essere compressi senza perdita di informazioni.

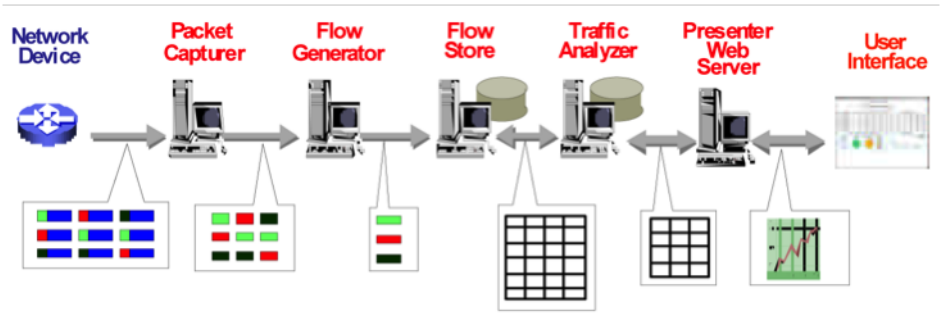


FIGURA 2.4: Architettura pipeline di NG-MON

- Spazio di memorizzazione limitato: anche aggregando i pacchetti in flussi, su reti ad alta velocità vanno memorizzati centinaia di megabyte al minuto. E' necessario un sistema di memorizzazione efficiente per memorizzare questi dati in uno spazio limitato.

- Supporto di diverse applicazioni: il sistema deve essere in grado di fornire i dati alle applicazioni in diversi formati.

Il sistema utilizza uno schema pipeline e tecniche di distribuzione del carico. Come si vede in Figura 2.4 il monitoraggio del traffico e l'analisi sono divisi in cinque fasi: cattura dei pacchetti, generazione dei flussi, memorizzazione dei flussi, analisi del traffico, e presentazione dei dati analizzati.

Il metodo di comunicazione tra le varie fasi è stato definito in modo da permettere la sostituzione di ogni fase con moduli più ottimizzati o con caratteristiche differenti. La cattura dei dati avviene distribuendo il traffico su più sonde. Ogni sonda elabora i pacchetti ricevuti mantenendo solo la parte dell'header cui si è interessati e sceglie a quale generatore di flusso inviare il dato. La scelta del generatore di flusso viene fatta calcolando una funzione hash sulla 5-tupla (IP sorgente, IP destinatario, porta sorgente, porta destinatario, protocollo). Usando la funzione hash si è sicuri che i dati appartenenti a uno stesso flusso siano elaborati dallo stesso generatore di flusso. Il formato dei dati inviati ai generatori di flusso sono mostrati in Figura 2.5 e occupano 28 byte per ogni pacchetto. Gli unici campi non estratti dagli header IP, UDP e TCP

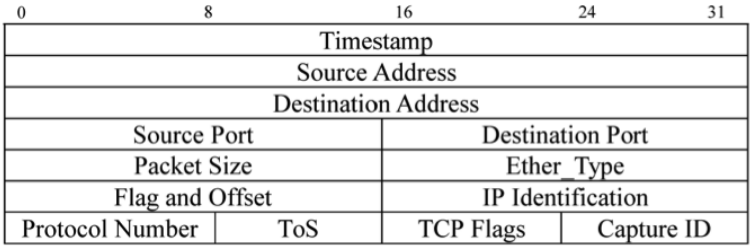


FIGURA 2.5: Formato dei dati inviati ai generatori di flusso

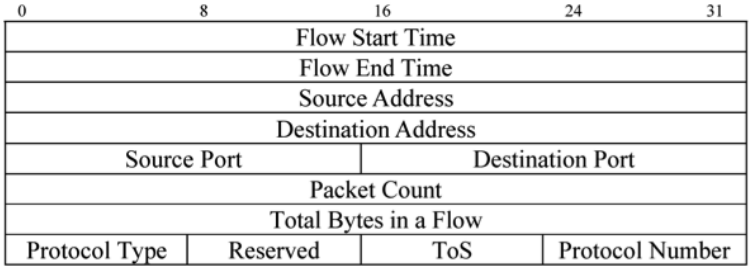


FIGURA 2.6: Formato dei dati di un flusso

sono Timestamp e Capture ID (che contiene l'identificatore della sonda che ha ricevuto il pacchetto).

Quando un generatore di flusso riceve un dato da una sonda cerca il flusso corrispondente in una tabella interna creando una nuova entry se non viene trovato. I pacchetti di uno stesso flusso sono aggregati nella tabella incrementando il contatore dei pacchetti e la lunghezza totale. Il generatore di flusso esporta i dati verso la fase successiva quando una delle seguenti condizioni è soddisfatta:

- il flusso è terminato (nel caso di flussi TCP è stato ricevuto un pacchetto con flag FIN)
- scade il timeout del flusso
- la tabella dei flussi è piena

In Figura 2.6 è mostrato il formato dei dati di un flusso. "Flow Start Time" indica il timestamp del primo pacchetto appartenente al flusso mentre "Flow End Time" indica il timestamp dell'ultimo pacchetto ricevuto. Lo spazio occupato da ogni flusso è di 32 byte. In ogni messaggio possono essere inviati i dati di circa 40 flussi.

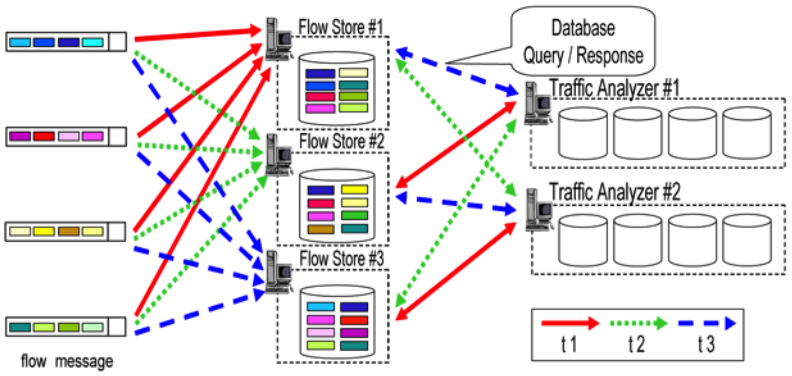


FIGURA 2.7: Distribuzione del carico nella fase di memorizzazione dei flussi

La soluzione adottata per evitare il collo di bottiglia nella memorizzazione dei flussi è quella di bilanciare il carico su più moduli con un algoritmo round-robin. Tutti i messaggi provenienti dai moduli di generazione dei flussi vengono inviati allo stesso modulo di memorizzazione per un periodo di tempo determinato dalla capacità e dal numero dei moduli di memorizzazione. Per l'utilizzo di questo algoritmo è necessario sincronizzare gli orologi dei generatori di flusso e la soluzione adottata è quella di usare NTP. Le operazioni di scrittura e d'interrogazione del database sono state separate in modo che non avvengano contemporaneamente. In Figura 2.7 si vede che all'istante t3 i messaggi sono ricevuti dal modulo 3 mentre gli analizzatori del traffico interrogano i moduli 1 e 2. Nei moduli di memorizzazione vengono mantenuti solamente i dati sui flussi più recenti quindi lo spazio necessario ad ogni modulo è limitato superiormente.

I moduli per l'analisi del traffico interrogano i moduli di memorizzazione creando matrici e tabelle. Per limitare lo spazio necessario a memorizzare i dati storici, viene usato un approccio simile a quello usato da Round Robin Database (RRD) [23]. RRD assume che l'interesse per i dettagli delle informazioni sul carico della rete diminuisca con il passare del tempo e quindi memorizza i dati con una risoluzione decrescente andando indietro nel tempo.

L'ultima fase, cioè quella di presentazione, è realizzata mediante l'uso di un server web che contiene le pagine con i risultati dell'elaborazione.

La convalida dell'architettura è state eseguita in modo analitico considerando i valori in Tabella 2.1. La dimensione totale dei pacchetti ricevuti dalla sonde su un link

Symbol	Description	Value
L	Link speed	10 Gbps
d	Full duplex factor	2
H_p	A single packet header data size	28 bytes
H_f	A single flow data size	32 bytes
C_{avg}	Average packet count per flow	16
P_{avg}	Average packet size	550 bytes

TABELLA 2.1: NG-MON: Simboli e valori

full duplex a 10 Gbps è:

$$T_p = L \times \frac{d}{8} = 10 \times \frac{2}{8} = 2.5 \text{ Gbyte/sec}$$

I generatori di flusso ricevono:

$$T_h = \frac{T_p}{P_{\text{avg}}} \times \left[H_p + \frac{\text{MAC header} + \text{IP header} + \text{UDP header}}{\text{the number of raw packet headers in an export UDP packet}} \right]$$

$$= \frac{2500}{550} \times \left[28 + \frac{14 + 20 + 8}{50} \right] = 131.1 \text{ Mbyte/sec}$$

La quantità di dati esportata dai generatori di flusso è:

$$T_f = \frac{T_p}{P_{\text{avg}}} \times \frac{H_f}{C_{\text{avg}}} = \frac{2500}{550} \times \frac{32}{16} = 9 \text{ MByte/sec}$$

Utilizzando PC con bus PCI 66Mhz/64 bit, che ha una banda di 533 Mbyte/sec, si possono utilizzare 4 schede ethernet a 1 Gbps (125 Mbyte/sec) su uno stesso PC. Una delle schede viene usata per inviare i dati ai generatori di flusso e le tre rimanenti per acquisire il traffico. Ogni PC è in grado di acquisire 375 MByte/sec quindi se la rete è a pieno carico sono necessari sette PC per analizzare tutto il traffico a 2.5 GByte/sec. Nella fase di generazione dei flussi vengono ricevuti 131.1 Mbyte/sec quindi è necessario un solo generatore di flusso con 2 interfacce di rete. La scrittura dei dati sui flussi prodotti in un minuto richiede 150 secondi, mentre l'analisi e quindi la lettura dei dati richiede 120 secondi per un totale di 4 minuti e 30 secondi. Sono quindi necessari 3 sistemi per effettuare l'inserimento e 2 per rispondere alle query. Su una rete a 10 Gbps utilizzata al massimo della sua capacità sono necessari 13 sistemi per fornire i dati ai

sistemi di analisi senza perdita di pacchetti.

2.1.4 Gigascope

La soluzione proprietaria realizzata da "AT&T" è basata su stream database ed utilizza un linguaggio SQL-like chiamato GSQL [24]. A differenza degli stream database comuni, che utilizzano sliding window, le query sono valutate su tumbling window. L'architettura è formata da uno *stream manager* che decide quali *query node* devono essere attivati e gestisce le operazioni di publish/subscribe. I query node sono processi che quando eseguiti si registrano presso lo stream manager. Quando un utente o un nodo necessita dell'output di un altro nodo esegue il subscribe presso lo stream manager. Da quel momento in poi il richiedente riceve l'output del nodo senza ulteriori interventi dello stream manager. Le query GSQL vengono trasformate in codice eseguibile C/C++ permettendo di avere le massime prestazioni a scapito della flessibilità.

Per ottimizzare le query GSQL modifica il piano d'accesso ai nodi e fa ottimizzazioni a basso livello sul codice generato. Una delle ottimizzazioni più significative è quella di spingere le query più in basso possibile dello stack di elaborazione, anche nella scheda di rete. Per questo motivo le query sono divise in query node ad alto livello (HFTA) e query node a basso livello (LFTA). Gli LFTA possono essere eseguiti dalle schede di rete in modo da avere prestazioni maggiori, i test hanno mostrato infatti che il sistema è in grado di avere una perdita di pacchetti sotto il 2% con traffico in ingresso a 610 Mbps.

2.1.5 DOME

L'architettura DOME (Distributed Online Measurement Environment) [25] definisce un sistema per il monitoraggio di rete distribuito. Lo schema generale dell'architettura (Figura 2.8) mostra l'utilizzo di sonde distribuite nella rete e un database centralizzato per la raccolta degli header dei pacchetti catturati. Le sonde possono essere sia dei router specializzati sia normali PC che utilizzano un network processor per svolgere gran parte dei compiti di elaborazione del traffico.

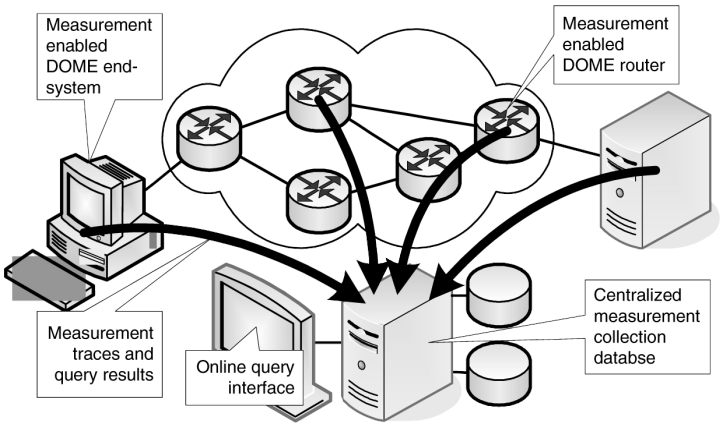


FIGURA 2.8: Schema generale architettura DOME

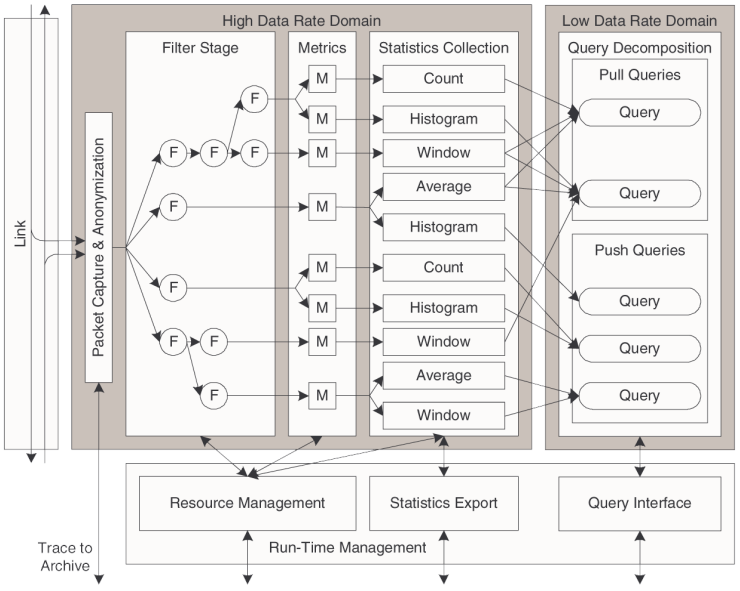


FIGURA 2.9: Architettura di un nodo DOME

L'architettura software delle sonde è divisa in due domini:

1. Alto data rate / requisiti computazionali bassi

2. Basso data rate / requisiti computazionali alti

Il dominio ad alto data rate (mostrato nella parte sinistra di Figura 2.9) si occupa dell'elaborazione di ogni singolo pacchetto che include: la cattura, il filtraggio, l'estrazione delle metriche e la generazione delle statistiche.

Il dominio a basso data rate (mostrato nella parte destra di Figura 2.9) gestisce le funzionalità del sistema di interrogazione. Queste funzionalità richiedono molte risorse

computazionali ma vengono eseguite a una frequenza molto inferiore a quella di arrivo dei pacchetti. Tra le funzioni svolte da questo dominio c'è la decomposizione delle query in filtri, metriche e statistiche mantenute dal dominio ad alto data rate.

Tutti gli header dei pacchetti acquisiti dopo essere stati anonimizzati vengono inviati a un database centrale. Per la trasmissione degli header e delle informazioni di controllo delle sonde viene utilizzato un link dedicato a 1 Gbps. Successivamente alla cattura, l'analisi viene eseguita in tre fasi:

Filtraggio

I pacchetti sono classificati secondo i filtri definiti tramite query al sistema.

Estrazione delle metriche

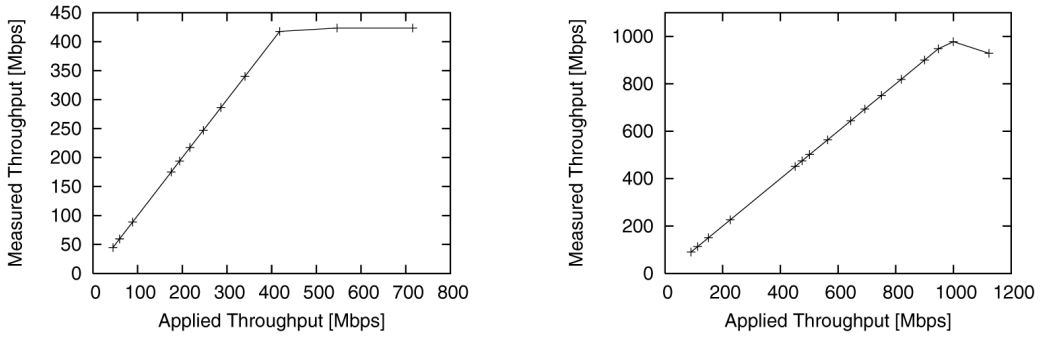
Vengono identificati i campi dei pacchetti che vengono utilizzati per calcolare le statistiche.

Calcolo delle statistiche

I dati dei campi estratti nella fase precedente vengono utilizzati per aggiornare contatori, medie e istogrammi.

Vista l'impossibilità di mantenere in memoria le statistiche alla massima risoluzione per lunghi periodi di tempo viene effettuata un'aggregazione sia temporale che spaziale. L'aggregazione temporale si ottiene aumentando l'intervallo su cui vengono calcolate le statistiche in modo da conservare un minor numero di valori. L'aggregazione spaziale invece riduce il dettaglio dei dati in memoria passando da una traccia completa a un solo flag che indica la presenza o meno di un particolare tipo di traffico.

L'architettura supporta query "continue" sui dati che arrivano e interrogazioni sui dati memorizzati. In base alle query "continue" inviate alla sonda le statistiche e il livello di aggregazione spaziale e temporale vengono modificati. In una query è possibile impostare un filtro, selezionare i valori di quali campi tenere in considerazione e quali statistiche generare. I tipi di statistiche selezionabili sono predefiniti e sono: flag di esistenza, contatore, media, istogramma, finestra e traccia completa. L'utente decide cosa vuole che sia esportato dalle sonde scegliendo tra "sommario" e "rapporto



(a) Small Packets (b) Large Packets

FIGURA 2.10: Throughput porta di monitoraggio

periodico". Le query "continue" vengono elaborate monitorando lo stream di pacchetti e inviando i risultati quando si verifica una particolare condizione o a intervalli di tempo regolari.

Un prototipo dell'architettura è stato realizzato utilizzando una network processor basato sulla piattaforma IXP2400 di Intel e installato sul link di accesso a internet dell'Università del Massachusetts. Il chip contiene 8 microengine ottimizzati per l'elaborazione dei pacchetti e ogni microengine esegue 8 thread in parallelo. Le statistiche generate includono contatori del numero di pacchetti, distribuzione dei protocolli di livello 3, dimensione dei pacchetti e numeri di porta TCP.

Test al simulatore hanno mostrato che inviando pacchetti da 60 byte il network processor IXP2400 può arrivare ad elaborare 1120 Mbps (pari a 900000 pacchetti al secondo). I test sul traffico sintetico effettuati in laboratorio mostrano (Figura 2.10) come stream di pacchetti piccoli causano congestione sulla porta di monitoraggio che deve trasmettere i dati al database centrale.

2.1.6 Interrogazione continua e storica su stream

Il lavoro presentato in [26] parte dall'osservazione che nei Data Stream Management Systems (DSMS) manca la possibilità di fare contemporaneamente interrogazioni in tempo reale e sui dati archiviati. Visto che l'archivio con i dati deve essere continuamente aggiornato e indicizzato sono stati utilizzati indici bitmap che permettono di rispondere in modo efficiente a molte interrogazioni su grossi insiemi di dati in sola

lettura. Uno dei motivi per cui gli indici bitmap possono essere aggiornati in modo efficiente è che i dati storici non sono mai modificati quindi l'indice non va mai modificato. Un secondo motivo è che è possibile aggiornare alcuni indici bitmap senza ordinare i record. Questo significa che è possibile aggiungere nuovi record a un indice bitmap in tempo proporzionale al numero di nuovi record da aggiungere.

I tre requisiti chiave del sistema di monitoraggio sono:

Flessibilità

L'amministratore di rete ha bisogno di selezionare le macchine e i protocolli che sono rilevanti per il problema che sta analizzando. Le nuove analisi devono poter essere eseguite tempestivamente in risposta a nuove problematiche.

Storia

Un sistema di monitoraggio efficace ha bisogno di accedere ai dati storici della rete. La storia passata permette all'amministratore di escludere falsi positivi comparando gli attuali comportamenti della rete con quelli passati.

Prestazioni

Durante i periodi di picco vengono generati decine di migliaia di flussi al secondo ed è proprio durante i periodi di picco che strumenti di monitoraggio efficaci sono essenziali.

Per realizzare il sistema di monitoraggio (Figura 2.11) sono stati uniti TelegraphCQ e FastBit. TelegraphCQ è un data stream management system basato su PostgreSQL mentre FastBit è un sistema per l'indicizzazione dei dati che sfrutta bitmap index compressi. Il sistema riceve in ingresso uno stream di flussi, l'*Ingress Manager* li converte in un formato compatibile con TelegraphCQ e FastBit e memorizza i dati su disco in attesa che vengano indicizzati con FastBit. Il *Controller* è un componente che riceve i risultati da TelegraphCQ, richiede i dati storici tramite FastBit e genera i report per l'amministratore.

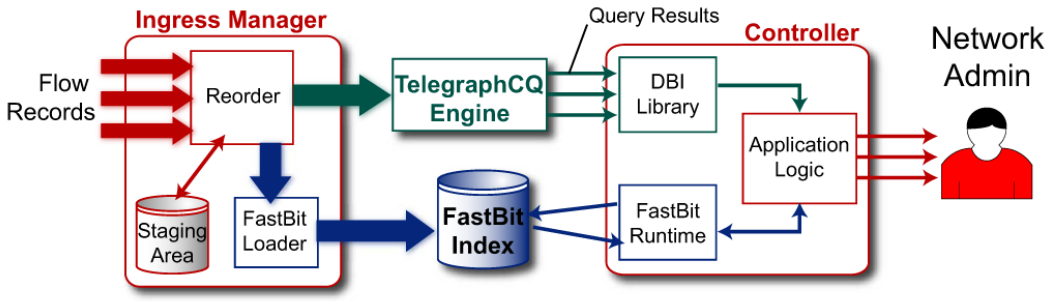


FIGURA 2.11: Schema del prototipo facente uso di TelegraphCQ e FastBit

I test del sistema sono stati eseguiti utilizzando traffico reale catturato tra il 2004 e il 2005 presso il Berkeley Lab. La velocità media di arrivo dei flussi è 500 record/sec ma ci sono picchi di 55000 record/sec.

Le prestazioni dei diversi moduli del sistema sono state valutate separatamente in modo da determinare il throughput massimo di ognuno di essi. Le macchine utilizzate per le analisi sono dual Pentium 4 a 2.8 GHz con 2 GB di RAM e un sistema RAID per l'archiviazione dei dati in grado di arrivare a 60 MB/sec in lettura/scrittura.

L'Ingress Manager e il Controller possono elaborare fino a 100.000 flussi o risultati al secondo e nessuno dei due rappresenta un collo di bottiglia per il sistema. Per quanto riguarda TelegraphCQ invece non si superano i 25.000 flussi al secondo e questo valore è influenzato dalla dimensione della finestra utilizzata per rispondere alle query. Con finestre troppo piccole i flussi analizzati scendono a 6.000 al secondo. Il tempo per la creazione degli indici in FastBit è stato valutato facendo variare la dimensione dei blocchi di record inseriti da 1.000 a 100 milioni di tuple (Figura 2.12). Ogni tupla contiene 11 attributi e occupa 48 byte. La massima velocità di 213.092 tuple/sec è ottenuta quando si utilizzano blocchi da 10 milioni di tuple. Il collo di bottiglia è quindi rappresentato da TelegraphCQ.

La valutazione delle prestazioni dell'intero sistema ha mostrato che il numero di record al secondo, usando una finestra larga un secondo, varia tra 9.000 e 20.000. Questo non è sufficiente a monitorare la rete del laboratorio nei periodi di picco. Per ovviare a questo problema è stato proposto di bufferizzare su disco i flussi in attesa che possano essere elaborati.

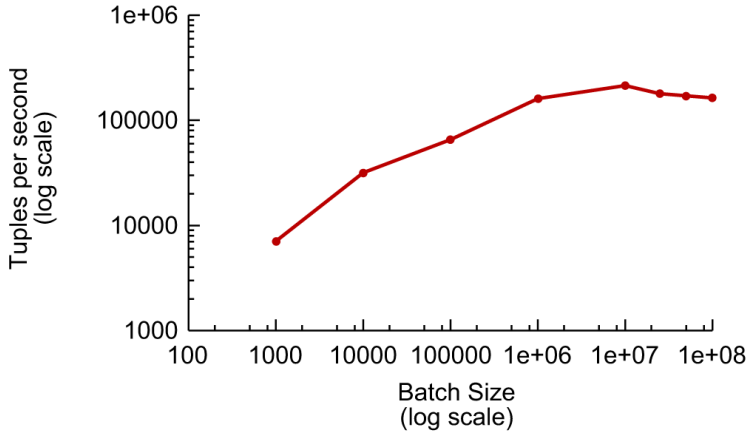


FIGURA 2.12: Velocità di creazione dell'indice bitmap in funziona della dimensione dei blocchi

	IPMON	**Nprobe**	**NG-MON**
Distribuita	✓	✗	✓
Dati trasferiti	Tutti gli header dei pacchetti	✗	Flussi
Monitoraggio online	✗	✗	✓
Real-time	✗	✗	✗
Archivio storico	✓	✓	✓ (con perdita di dettaglio)
Granularità	59,6 ns	Inizio sessione 1 μs	Inizio sessione 1 s
Selezione dei dati	N.D.	N.D.	N.D.
Privacy	Solo header	N.D.	Solo header
Requisiti Hardware	Endace DAG	Commodity + modifica FW schede di rete	Commodity
Console	N.D.	N.D.	N.D.
Unità di monitoraggio	Header pacchetto	Flusso	Flusso

TABELLA 2.2: Caratteristiche delle architetture analizzate (parte 1)

2.1.7 Comparazione delle architetture

Dalla descrizione delle architetture presentate sono state estratte le caratteristiche chiave e riassunte nelle tabelle 2.2 e 2.3. Le caratteristiche non specificate dagli articoli o che non possono essere dedotte vengono indicate con "N.D.".

	Gigascope	**DOME**	**TelegraphCQ + Fastbit**
Distribuita	✓	✓	✗
Dati trasferiti	Risultati query	Header pacchetti	Risultati query
Monitoraggio online	✓	✓	✓
Real-time	N.D.	✓	✗
Archivio storico	N.D.	✓	✓ (con perdita di dettaglio)
Granularità	N.D.	Variabile	Inizio sessione 1 s
Selezione dei dati	GSQL	Query push/pull	CSQL
Privacy	N.D.	Header anonimizzati	Solo header
Requisiti Hardware	N.D.	Commodity + Network Processor	Commodity
Console	N.D.	N.D.	N.D.
Unità di monitoraggio	Pacchetto	Header pacchetto	Flusso

Tabella 2.3: Caratteristiche delle architetture analizzate (parte 2)

2.2 Componenti e tecnologie

In questo paragrafo sono riportati i componenti e le tecnologie utilizzate per il monitoraggio di rete passivo, per l'interrogazione e per la presentazione dei dati. Questi sistemi sono stati utilizzati nella realizzazione del prototipo per la validazione dell'architettura.

2.2.1 Sincronizzazione temporale

Tutti i sistemi real-time devono avere la nozione di tempo. Il riferimento temporale può essere assoluto, cioè corrispondente all'orario scandito da un orologio, oppure relativo a specifici eventi. I sistemi distribuiti devono avere un sistema di riferimento temporale consistente per sincronizzare le loro attività e per identificare l'istante in cui si verifica un evento sui nodi dell'architettura [27]. Per sincronizzare i nodi sono

necessari tre elementi: la fonte dell'informazione temporale, i canali di comunicazione per distribuire l'informazione e il protocollo di sincronizzazione. Le fonti con cui sincronizzare gli orologi dei nodi possono essere diverse, si possono usare segnali radio, il GPS oppure orologi atomici. Il GPS è un ottimo metodo ma senza utilizzare antenne esterne non può essere utilizzato all'interno dei data center. La distribuzione dell'informazione tra computer avviene di solito attraverso la rete che può introdurre disturbi sulla propagazione dell'ora esatta e rende quindi necessario un protocollo che tenga conto della latenza e del jitter sincronizzando comunque l'orologio dei nodi con un'alta precisione. In letteratura sono stati proposti diversi modelli e tecniche di sincronizzazione, il protocollo maggiormente diffuso è il Network Time Protocol [28]. Una volta sincronizzati gli orologi interni dei nodi la sincronizzazione va comunque ripetuta periodicamente perché i computer per scandire il passare del tempo utilizzano solitamente un oscillatore al quarzo che può discostarsi dall'orario esatto anche di 1 secondo ogni 24 ore.

2.2.2 Acquisizione del traffico

Per l'acquisizione del traffico esistono essenzialmente due approcci: l'utilizzo di hardware dedicato oppure l'utilizzo di soluzioni software che usano commodity hardware. Il vantaggio offerto dalle soluzioni con hardware dedicato come ad esempio le schede Endace DAG è quello di garantire la cattura di tutto il traffico a qualsiasi velocità offrendo funzionalità di bilanciamento del traffico tra più processi [29], sincronizzazione temporale mediante l'uso di dispositivi esterni quali il GPS e timestamping dei pacchetti. Inoltre queste schede sono programmabili e specializzate per l'elaborazione del traffico permettendo di scaricare su di loro gran parte del carico computazionale. Lo svantaggio è l'elevato costo di queste schede e il fatto di essere legati a un produttore di hardware.

Da alcuni anni anche le soluzioni software permettono ottime prestazioni grazie all'evoluzione dell'hardware e all'eliminazione di alcune delle limitazioni imposte dai sistemi operativi. Prima dell'introduzione di queste soluzioni, la percentuale di pacchetti

che si riuscivano a catturare era minima.

Un primo passo verso la cattura efficiente del traffico è stato fatto passando da uno strato di rete basato su *interrupt* a uno facente uso del *device polling* [30]. Con lo strato di rete basato su interrupt ogni volta che la scheda di rete deve comunicare con la cpu (ad esempio quando arriva un nuovo pacchetto) invia un interruzione. Il trattamento delle interruzioni è un'operazione molto costosa perché implica la sospensione del processo in esecuzione e la commutazione di contesto con conseguente svuotamento della cache ed esecuzione dell'handler dell'interruzione. Con alti carichi di rete la maggior parte delle risorse della CPU sono spese per il trattamento delle interruzioni lasciando pochissimi cicli di clock per l'elaborazione del traffico. Con il polling il sistema interroga periodicamente il dispositivo per controllare se richiede attenzione e in caso affermativo invoca l'handler appropriato. Eseguire il polling quando si è già nel giusto contesto permette di evitare l'overhead del context switch. Questo porta a un aumento della latenza con cui vengono elaborati gli eventi ma permette di utilizzare la CPU per l'elaborazione del traffico e non solamente per il trattamento delle interruzioni.

Oltre al device polling molte delle risorse utilizzate per la cattura del traffico erano sprecate copiando i dati dallo spazio kernel allo spazio utente. Quindi la libreria *libpcap*[3], comunemente usata per la cattura del traffico, è stata modificata per utilizzare il memory map in modo da evitare queste copie.

Lo standard attuale per la cattura del traffico di rete su sistemi linux è PF_RING [31] che permette di migliorare le prestazioni ottenute con il device polling che comunque non permetteva di catturare tutto il traffico con pacchetti di piccole dimensioni. Dalle misure effettuate era emerso che anche se l'utilizzo di memory map aveva ridotto il tempo per il passaggio da spazio kernel a spazio utente gran parte del tempo veniva ancora perso spostando i pacchetti dalla scheda di rete allo spazio utente attraverso le strutture dati e le code in spazio kernel. Per risolvere questo problema è stato introdotto un nuovo tipo di socket (PF_RING) ottimizzato per la cattura dei pacchetti e basato su un buffer circolare dove vengono copiati i pacchetti in ingresso. Il socket

[3] http://www.tcpdump.org/

Packet Size (bytes)	**Linux 2.4.23/RTIRQ NAPI+PF_RING (Pkt Capture)**		**Linux 2.4.23/RTIRQ NAPI+PF_RING (nProbe)**	
64	550 789 pps	$\sim$ 202 Mbit	376 453 pps	$\sim$ 144 Mbit
512	213 548 pps	$\sim$ 850 Mbit	213 548 pps	$\sim$ 850 Mbit
1 500	81 616 pps	$\sim$ 970 Mbit	81 616 pps	$\sim$ 970 Mbit

Tabella 2.4: Prestazioni di PF_RING

viene associato in sola lettura a una scheda di rete e il buffer è allocato staticamente quando il socket viene creato. I nuovi pacchetti ricevuti dalla scheda di rete vengono memorizzati nel buffer circolare o scartati se il buffer è pieno. I pacchetti ricevuti non vengono inoltrati ai layer superiori facendo aumentare le prestazioni generali. Il buffer circolare viene esportato in spazio utente tramite memory map. I vantaggi apportati da questa soluzione sono:

- I pacchetti non sono accodati nelle strutture dati del kernel.
- L'uso di memory map permette alle applicazioni in spazio utente di accedere al buffer circolare senza l'overhead dovuto alle system call.
- Anche se il kernel non supporta il device polling il sistema resta utilizzabile anche con molto traffico visto che il tempo necessario a gestire le interruzioni è molto inferiore a quello necessario nel caso della normale gestione dei pacchetti.
- Più applicazioni possono aprire socket PF_RING contemporaneamente senza interferire tra loro ovvero senza che l'applicazione più lenta rallenti quelle più veloci.

I test effettuati utilizzando un PC equipaggiato con CPU Pentium 4 a 1.7 GHz e scheda di rete Intel a 1 Gbps hanno dato i risultati mostrati in Tabella 2.4. Le ultime due colonne mostrano le prestazioni di nProbe (una sonda NetFlow) ed evidenziano come l'analisi NetFlow sia limitata solamente dalle risorse CPU disponibili perdendo pacchetti solo nel caso in cui questi siano molto piccoli. Considerando che spesso le reti Gbit utilizzano jumbo frame (>= 9.000 byte) PF_RING può essere utilizzato per analizzare il traffico a wire speed.

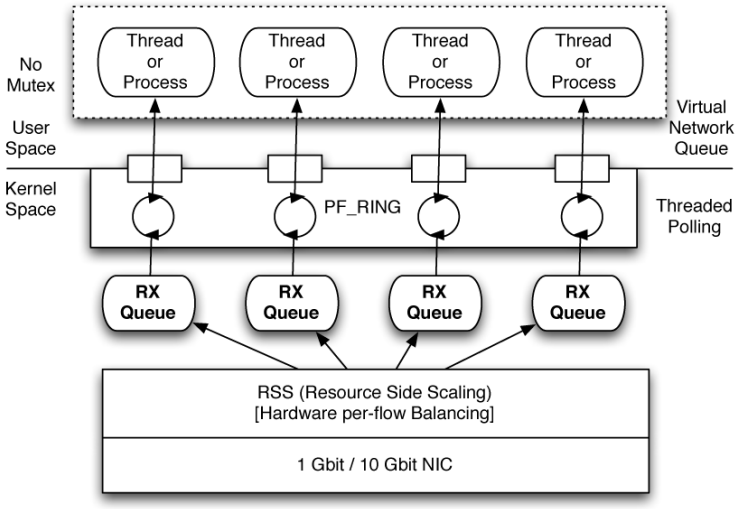


FIGURA 2.13: Threaded NAPI

Un altro passo in avanti nella cattura del traffico utilizzando commodity hardware è stato fatto con la diffusione dei sistemi multi-core. In [32] vengono descritte le problematiche che si hanno nello sfruttare il parallelismo offerto da questi sistemi e viene proposto un nuovo sistema per la cattura del traffico chiamato Threaded NAPI (TNAPI). TNAPI sostituisce il meccanismo di polling dei pacchetti di Linux (chiamato NAPI) con una nuova versione che sfrutta la tecnologia Resource Side Scaling (RSS) la quale permette di separare i pacchetti in base al flusso di appartenenza e di inserirli in code di ricezione multiple. Associando ogni coda a un kernel thread separato è quindi possibile eseguire il polling in parallelo (Figura 2.13) e sfruttare le risorse messe a disposizione dalle architetture multi-core. Dai test effettuati è stato verificato che utilizzando correttamente TNAPI la velocità di cattura dei pacchetti scala linearmente con il numero di interfacce di rete. La velocità massima a cui è stato testato il sistema è stata 4 Gbps e non si sono verificate perdite di pacchetti durante la cattura.

2.2.3 Aggregazione in flussi

Per riuscire ad aggregare i pacchetti in flussi su reti ad alta velocità è necessaria una struttura dati che permetta di effettuare inserimenti ed aggiornamenti in tempo costante. In [33] viene fatta un'analisi delle tecniche basate su hash per raggiungere questi

risultati. Molte di queste tecniche sfruttano hardware dedicato che utilizza memorie associative o memorie SDRAM per effettuare la ricerca nella tabella hash [34, 35].

Visto il requisito di utilizzare commodity hardware, nella tesi si sono prese in considerazione solamente soluzioni senza hardware dedicato. In [36] viene consigliato l'uso dello schema *Cuckoo Hashing* [37] che viene descritto di seguito.

Il dizionario Cuckoo Hashing utilizza due tabelle hash, T_1 e T_2, di lunghezza r e due funzioni hash $h_1, h_2 : U \to \{0, ..., r-1\}$. Ogni chiave $x \in S$ è memorizzata nella cella $h_1(x)$ di T_1 o $h_2(x)$ di T_2 ma mai in entrambe. Per quanto riguarda la fase di inserzione si utilizza "l'approccio del cuculo"[4] "cacciando via" dalla struttura dati le chiavi che collidono inserendole in una nuova posizione finché tutte le chiavi hanno il proprio "nido". Per l'inserzione dell'elemento x l'algoritmo verifica se la cella $h_1(x)$ di T_1 è occupata e se non lo è l'inserimento si conclude ponendo $T_1[h_1(x)] \leftarrow x$. Altrimenti si rimuove l'elemento corrente e al suo posto si inserisce x rendendo l'elemento corrente "senza nido". L'elemento rimosso viene inserito in T_2 con lo stesso procedimento usato per x (eventualmente rimuovendo l'elemento presente). Questo algoritmo potrebbe portare a un numero di iterazioni infinito quindi viene posto un limite al numero di iterazioni. Se si raggiunge il numero massimo di iterazioni l'articolo originale consiglia un rehashing dell'intera tabella usando due funzioni hash differenti. Un miglioramento del Cuckoo Hash viene proposto in [38] dove si dimostra che usando una memoria aggiuntiva (stash) per memorizzare gli elementi che non possono essere inseriti nelle tabelle hash si riesce ad abbassare la probabilità che ci sia bisogno del rehashing. Ricreare l'hash sarebbe troppo costoso nel caso dell'aggregazione dei pacchetti quindi è molto importante avere una funzione hash con una bassa percentuale di collisioni.

[4] La femmina del cuculo non costruisce il nido ma depone le uova in quello di altri uccelli; questi portano avanti la cova dell'uovo estraneo, che schiude prima degli altri. Il piccolo cuculo, allora, butta fuori dal nido le altre uova, e rimane unico destinatario delle attenzioni dei genitori forzatamente adottivi.

2.2.4 Archiviazione ed interrogazione dei dati

L'attuale generazione di strumenti scientifici e di simulazione al computer sono in grado di produrre massicce quantità di dati di cui però solo una minima parte contengono informazioni utili. La ricerca di un relativamente piccolo numero di record sparso in vasti archivi è una sfida comune in molti campi scientifici. I Database Management Systems (DBMS) sono progettati per affrontare questi problemi ma non sono ottimizzati per l'esplorazione di dati scientifici. Questi sono infatti progettati per applicazioni transazionali dove i record sono modificati di frequente come nel caso dei conti in banca. Al contrario le applicazioni scientifiche non hanno necessità di effettuare questi tipi di transazioni. La maggior parte delle interazioni con i dati scientifici avvengono per la selezione di record su cui effettuare ulteriori analisi.

Nel progetto FastBit [39] sono state sviluppate delle tecniche ottimizzate per la gestione di dati scientifici rappresentati da valori numerici basate su *bitmap index*. I bitmap index comuni sono efficaci solamente su variabili a bassa cardinalità mentre i dati scientifici di solito contengono variabili con cardinalità molto alta come numeri in virgola mobile che si ripetono raramente. Di seguito vengono riportate le quattro tecniche principali che rendono FastBit uno strumento molto efficiente per l'indicizzazione di dati scientifici.

Bitmap index

Nei classici DBMS le strutture dati utilizzate per memorizzare gli indici sono varianti dei B-tree. In FastBit i valori indicizzati sono memorizzati in bitmap che rappresentano le stesse informazioni contenute nei B-tree ma tramite una sequenza di bit. I bitmap index sono particolarmente utili per applicazioni che richiedono un grande numero di query. Uno dei motivi principali è che i risultati delle query possono essere calcolati eseguendo operazioni logiche bit a bit sulle bitmap. Visto che l'hardware dei computer supporta queste operazioni logiche in modo efficiente il calcolo dei risultati delle query ne beneficia. Un altro motivo per usare bitmap index è che i risultati di diversi bitmap index possono essere facilmente combinati. Infatti il risultato dell'interrogazione di un bitmap index è

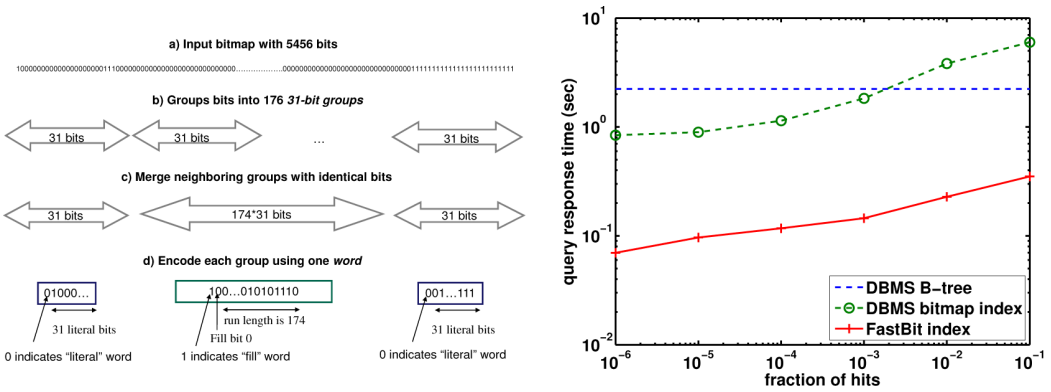
a sua volta un bitmap index e quindi combinare i risultati di query diverse consiste semplicemente nel fare altre operazioni logiche bit a bit. Il più grande problema è che la loro dimensione aumenta linearmente con il numero di valori distinti che devono indicizzare. Per questo motivo sono necessarie tecniche di compressione, codifica e binning che permettano di far diminuire lo spazio occupato e aumentare le prestazioni.

Bitmap compression

Ogni bitmap index di solito contiene un grande numero di bit con valore 0 rendendo facile l'operazione di compressione. Il metodo di compressione più usato è il Byte-aligned Bitmap Code (BBC) [40]. Con FastBit è stato sviluppato un nuovo metodo chiamato Word-Aligned Hybrid (WAH) [41] che ha dimostrato di velocizzare le operazioni logiche bit a bit fino a 10 volte rispetto al BBC. WAH adotta un approccio ibrido che per rappresentare sequenze lunghe di 0 o 1 usa un run-legth encoding mentre per le sequenze corte utilizza bit letterali. Inoltre i dati compressi sono word-aligned e quindi più facilmente elaborabili dal processore. Le parole risultanti dalla compressione WAH possono essere di due tipi: *literal* e *fill*. In 2.14 (a) viene mostrata una parola il cui primo bit, indicante il tipo, ha valore 0 (literal) e i restati bit la sequenza non compressa. Le parole fill invece hanno il primo bit che vale 1, il secondo bit che rappresenta il valore della sequenza di bit uguali e i restanti 30 bit la lunghezza della sequenza. In 2.14 (b) viene mostrato l'effetto della compressione sul tempo di risposta alle interrogazioni e come con WAH il tempo sia di circa 14 volte inferiore rispetto ai DBMS che utilizzano bitmap index BBC.

Bitmap encoding

Le tecniche di bitmap encoding manipolano le bitmap prodotte dai bitmap index per ridurre il numero di bitmap in un indice o il numero di bitmap necessari a rispondere alle interrogazioni. La nuove classe di metodi di codifica proposto con FastBit è stato chiamato *multi-level index*. L'idea è quella di creare una gerarchia di contenitori indicizzati tramite bitmap index e in ogni contenitore inserire un



(a) an illustration of WAH (b) time (s) to answer queries

FIGURA 2.14: Effetti della compressione sul tempo di risposta alle query

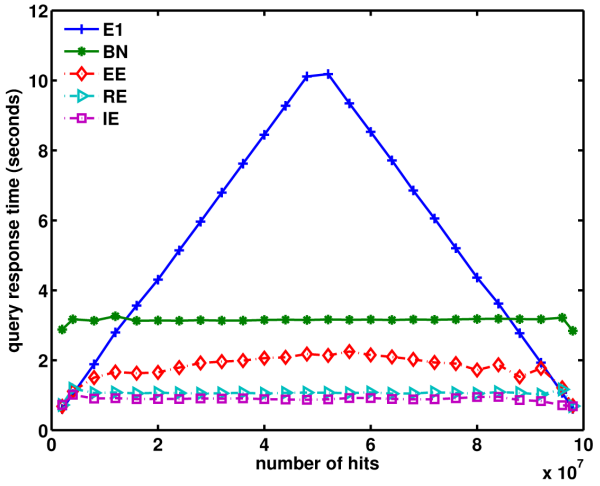


FIGURA 2.15: Tempo di risposta medio al variare del bitmap encoding

range di valori. Man mano che si scende verso le foglie dell'albero il range di valori è sempre più piccolo. Per rispondere alle query si percorre l'albero dalla radice verso le foglie selezionando i nodi che includono i valori ricercati. In 2.15 si vede come il tempo di risposta alle query con multi-level index (EE, RE, IE) è in media da 3 a 5 volte inferiore rispetto ai più classici basic bitmap index (E1) e binary encoded index (BN).

Binning

I dati scientifici sono spesso calcolati o raccolti con alta precisione quindi gli indici generati possono essere molto grandi e lenti da usare. È stato osservato che di solito le interrogazioni non sono eseguite con una precisione esatta ma ad

esempio in caso di numeri in virgola mobile le interrogazioni richiedono valori con una precisione più bassa. Raggruppare i valori in intervalli (binning) può ridurre il numero di bitmap in un indice e migliorare il tempo di risposta delle query. Usando il binning si possono verificare situazioni in cui è necessario accedere ai dati per rispondere alle query (ad esempio se si richiedono dati presenti solo parzialmente in un bin). In questo caso i valori all'interno del bin sono considerati dei candidati per la risposta e va eseguito un *candidate check*. Quando si rende necessario il candidate check questo domina il tempo di risposta delle query. L'approccio proposto per ottimizzare questo controllo, chiamato Order-preserving Bin-based Clustering (OrBiC) [42], organizza i valori di ogni bin in modo contiguo sul disco in modo da ridurre il tempo richiesto per le operazioni di I/O.

In conclusione FastBit condivide diversi dei difetti degli altri bitmap index, il più importante dei quali è che sono lenti quando vengono aggiornati record esistenti. Questo impedisce che vengano usati efficacemente nelle applicazioni dove i dati cambiano frequentemente. FastBit è invece efficace nei casi in cui i cambiamenti avvengano solo nei nuovi record e per questo motivo si rende adatto per l'interrogazione di archivi contenenti dati storici sul traffico di rete come dimostrato in [43].

2.2.5 Tecniche di data delivery su web

Con l'evoluzione del web che ha portato al Web 2.0 è stato introdotto l'uso di AJAX (Asynchronous JavaScript and XML) [44] per ovviare al basso grado di interattività delle applicazioni web. L'obiettivo era quello di migliorare la reattività delle pagine web scambiando quantità di dati minime con il server e aggiornando solo le parti necessarie dell'interfaccia utente.

Il modello di interazione comunemente usato su web chiamato REST (Representational State Transfer) richiede che tutte le comunicazioni tra browser e server siano iniziate dal client. Con questo schema ogni interazione tra client e server è indipendente dalle altre favorendo la scalabilità ma precludendo la possibilità che i server inviino notifiche asincrone.

Ci sono diversi casi in cui è importante che il server notifichi gli eventi in tempo reale e la visualizzazione dei risultati nel monitoraggio di rete è uno di questi. I due approcci utilizzabili per aggiornare le pagine web in seguito a un cambio di stato sul server sono il *pull* o il *push*. Nel metodo pull i client richiedono gli aggiornamenti al server ad intervalli prefissati. Nel metodo push, invece, i client sottoscrivono gli argomenti di interesse e il server invia loro i cambiamenti in maniera asincrona quando lo stato cambia. Di seguito vengono presentate in modo più dettagliato le possibili soluzioni al problema delle notifiche asincrone.

HTTP Pull

La maggior parte delle applicazioni AJAX controlla ad intervalli regolari di lunghezza prefissata, chiamata *Time To Refresh* (TTR), se ci sono aggiornamenti da parte del server. Questo controllo avviene indipendentemente dal fatto che lo stato sul server sia cambiato o meno. Per avere i dati sempre aggiornati questo controllo deve avvenire frequentemente producendo molto traffico sulla rete per l'invio di messaggi non necessari. Inoltre l'applicazione perde tempo nel richiedere gli aggiornamenti facendo diminuire la reattività delle applicazioni stesse. L'ideale sarebbe che la frequenza dei controlli sia uguale al *Publish Rate* (PR). Se la frequenza è più bassa del PR il client può perdere degli aggiornamenti. Anche se sono state sviluppate tecniche per adattare il TTR alla frequenza di aggiornamento del server questo metodo non permette di avere una completa accuratezza dei dati e genera traffico non necessario.

HTTP Streaming

Questo metodo si basa su un'idea introdotta nel 1992 da Netscape e ha due varianti: Page Streaming e Service Streaming. Il **Page Streaming** consiste nell'inviare i dati in risposta a una connessione HTTP precedentemente stabilita e tenuta aperta dal server (long-lived HTTP connection). Il client non chiude la connessione dopo l'arrivo dei dati ma attende l'arrivo di aggiornamenti. Il **Service Streaming** si basa sull'oggetto XMLHttpRequest tenendo anche in questo caso

la connessione aperta e attendendo gli aggiornamenti da parte del server ma a differenza del Page Streaming la connessione avviene in background.

Comet

L'applicazione dello schema Service Streaming in AJAX viene chiamata Reverse AJAX o Comet [45]. Comet permette al server di inviare un messaggio al client quando si verifica un evento senza che il client ne faccia esplicita richiesta. Per l'invio delle notifiche in Comet è possibile scegliere tra due tecniche: **Streaming** o **Long polling** (chiamato anche Asynchronous-Polling). Nel caso di streaming il server invia risposte parziali senza chiudere la connessione usando l'HTTP chunked mode. I dati sono inviati in modo incrementale sulla connessione aperta e quindi mostrati dal browser. Con il long polling è sempre il client ad aprire una connessione verso il server e la mantiene aperta finché il server non ha notifiche da inviare. Quando viene ricevuta una notifica la connessione viene chiusa e il client provvede a riaprirne un'altra.

In [46] vengono analizzate le prestazioni dei due approcci push e pull. Le analisi sono state fatte facendo variare le seguenti variabili indipendenti:

- Numero di utenti 100, 500, 1000, 2000, 5000 e 10000; la variazione permette di trovare il numero massimo di utenti che un server può gestire contemporaneamente.
- Intervallo di pubblicazione 1, 5, 15, 30 e 50 secondi; la frequenza con cui vengono pubblicati gli aggiornamenti è importante per determinare le prestazioni di Comet che si comporta più come un sistema pull che come un sistema push quando l'intervallo di pubblicazione è molto breve.
- Intervallo di pull 1, 5, 15, 15, 30 e 50 secondi; quando si usa l'approccio pull la frequenza con cui si effettua il polling ha effetto sulle misure.
- Modalità Comet o pull.
- Numero totale di messaggi fissato a 10.

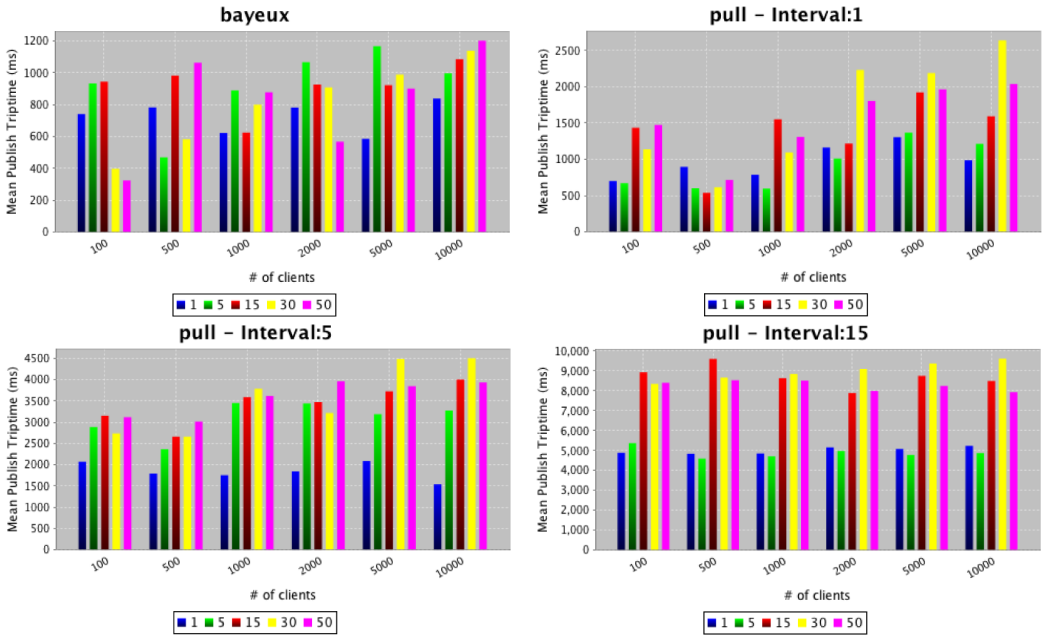


FIGURA 2.16: Mean Publish Trip-time

Le misure più significative effettuate sono le seguenti:

Mean Publish Trip-time (MPT)

Il trip-time è stato definito come:

Trip-time = |Data Creation Date − Data Receipt Date|

Data Creation Date è la data in cui il server crea il messaggio.

Data Receipt Date è la data in cui il client riceve il messaggio.

Questa metrica mostra quanto tempo ci mette un messaggio pubblicato a raggiungere il client e viene usato per verificare con che velocità il client riceve la notifica di un evento. Un dato viene definito *coerente* se il suo valore sul server e sul client è sincronizzato. Un dato con un basso trip-time porta a un alto grado di coerenza. In Figura 2.16 si vede che l'MPT per la soluzione Comet (indicato come Bayeux) è al massimo 1200 ms (le barre colorate indicano l'intervallo di pubblicazione). Va notato che l'MPT è calcolato sui messaggi realmente ricevuti non tenendo conto di eventuali perdite che si verificano nell'approccio pull. Da questi risultati si può trarre la conclusione che l'approccio pull ha un minore grado di coerenza rispetto all'approccio push.

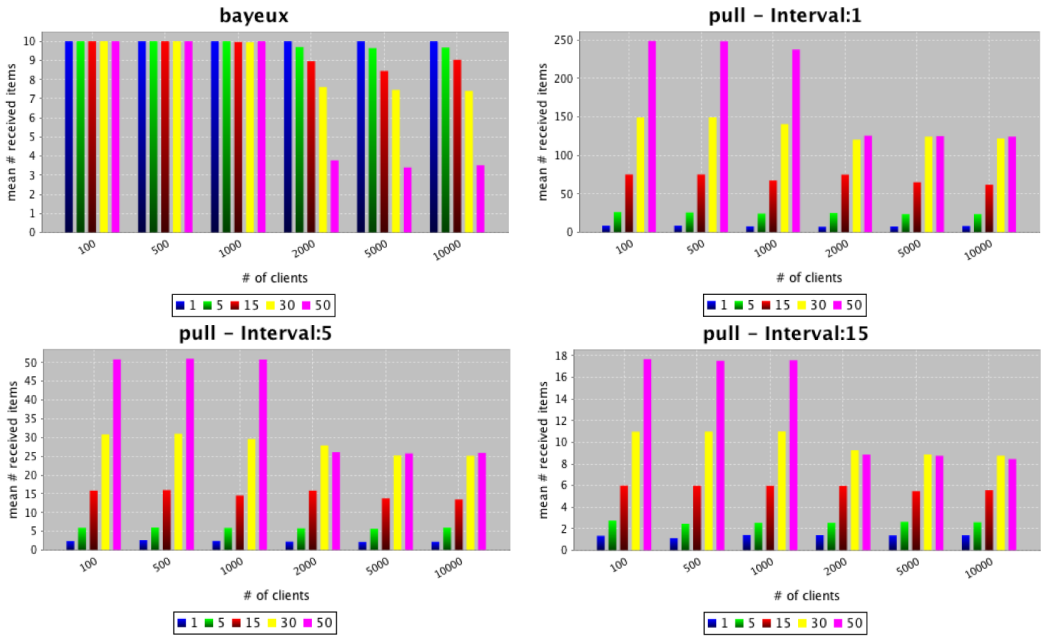


FIGURA 2.17: Mean Received Publish Messages

Received Publish Messages (RPM)

La misura viene fatta pubblicando 10 messaggi e contando quanti messaggi (non unici) raggiungono il client. Se il client riceve più di 10 messaggi significa che si stanno scambiando messaggi ridondanti. Per ottenere un alto grado di coerenza la soluzione pull deve avere un'alta frequenza di polling ma come si vede dalla Figura 2.17, l'alta frequenza di polling porta ad avere un grande overhead sulla rete fino ad arrivare al 96% di richieste non necessarie.

Received Unique Publish Messages (RUPM)

In Figura 2.18 viene mostrato il conteggio dei messaggi unici ricevuti dal client utile a determinare se ci sono aggiornamenti che non sono stati ricevuti. Se l'intervallo di pubblicazione è maggiore o uguale all'intervallo di pull il client riceve la maggior parte dei messaggi però, come detto in precedenza, utilizzare intervalli di pull brevi porta ad avere un grande overhead sulla rete. Con comet invece tutti i messaggi sono ricevuti dal client in modo corretto.

Negli esperimenti è stato dimostrato che per avere alta coerenza e alte prestazioni della rete va scelto un approccio push. L'approccio pull infatti non raggiunge lo stesso

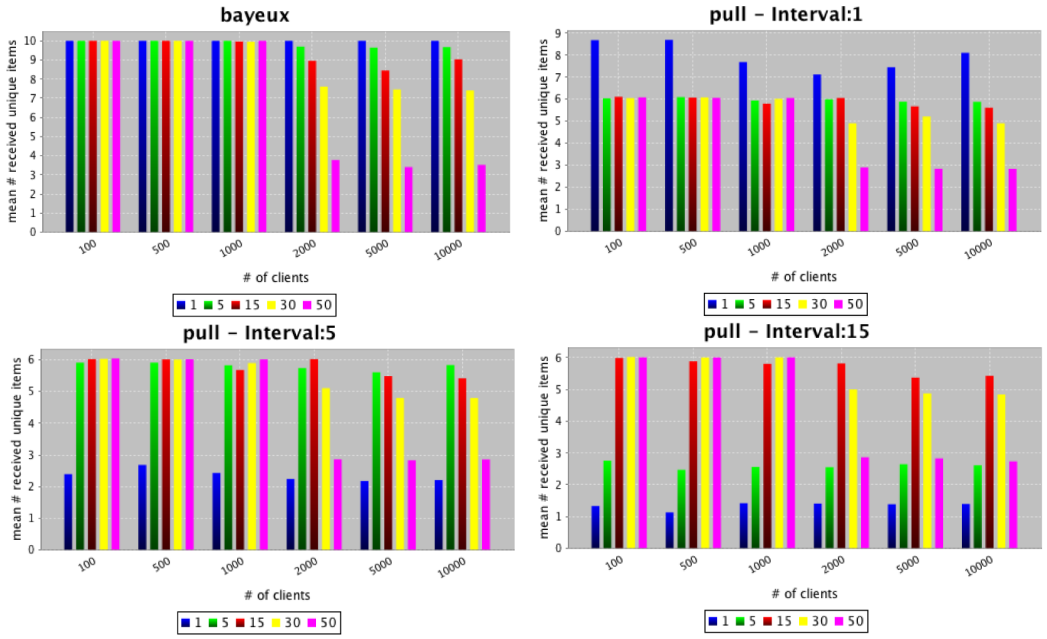


FIGURA 2.18: Mean Received Unique Publish Messages

grado di coerenza del push anche con un'alta frequenza di polling.

Una possibile alternativa a Comet sono i WebSockets che danno la possibilità di avere connessioni persistenti tra client e server e saranno inclusi nello standard HTML5. Purtroppo però al momento sono pochissimi i browser che supportano tale tecnologia.

In questo capitolo sono state analizzate le architetture esistenti e i componenti necessari per la realizzazione di un sistema per il monitoraggio di rete. Nel prossimo capitolo sarà presentata l'architettura DRT-Mon oggetto della tesi.

Capitolo 3

L'Architettura DRT-Mon

L'architettura proposta permette di avere un sistema di monitoraggio di rete distribuito near-realtime in grado cioè di presentare i risultati del monitoraggio dopo pochi istanti rispetto al verificarsi di un evento. Il sistema effettua monitoraggio passivo e utilizza un modello distribuito in modo da non dover replicare e trasportare i pacchetti dalle sottoreti dove transitano ad un unico punto centrale. Distribuendo delle sonde nelle vicinanze dei punti da monitorare si riduce la quantità di traffico che ognuna di esse deve elaborare facendo diminuire i loro requisiti computazionali e le prestazioni richieste dalla rete di monitoraggio per il trasporto dei dati da analizzare. Il sistema oltre a fornire un monitoraggio in tempo reale memorizza tutto il traffico calcolato in modo da permettere l'analisi a posteriori degli eventi che si sono verificati sulla rete.

La definizione di questa architettura deriva dalla necessità per gli amministratori di rete di avere uno strumento configurabile che permetta di conoscere istante per istante lo stato di tutta la rete dando loro la possibilità di variare il livello di dettaglio con cui visualizzare le statistiche passando da una visione globale della rete ai dettagli su una singola connessione. Per poter variare arbitrariamente il livello di dettaglio, sia sulle statistiche in tempo reale che quelle sui dati storici, le misure sono effettuate in modo esatto escludendo approcci come il campionamento o l'aggregazione con perdita di dettaglio come ad esempio fa MRTG.

Nel paragrafo 3.1 viene presentata l'architettura generale. Nei paragrafi successivi

vengono invece presentati i nodi che compongono l'architettura: Sonda di rete (par. 3.2), Collettore (par. 3.3) e Console di monitoraggio (par. 3.4).

3.1 Descrizione generale

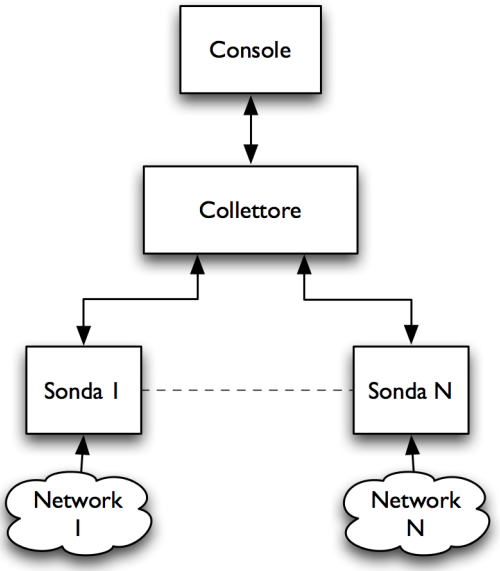


FIGURA 3.1: Architettura del sistema di monitoraggio

L'architettura, il cui schema è presentato in figura 3.1, adotta il modello di elaborazione data parallel Map & Reduce. Questo modello prende in ingresso un problema, lo divide in sottoproblemi e li distribuisce a dei worker che li risolvono in parallelo. Un worker a sua volta può essere parallelizzato seguendo lo stesso o altri schemi. Quando i worker finiscono l'analisi dei sottoproblemi inizia la fase di reduce in cui i risultati vengono uniti applicando un operatore associativo. Il modello è applicabile se ogni sottoproblema è indipendente dagli altri ovvero se i worker non devono comunicare tra loro per completare l'elaborazione e se è possibile unire i sottoproblemi mediante l'uso di operatori associativi.

Nel caso del monitoraggio di rete il traffico che passa sulle sottoreti da analizzare viene replicato e distribuito alle sonde le quali si occuperanno dell'elaborazione. La distribuzione del traffico alle sonde, che rappresentano i worker del modello Map & Reduce, non può essere bilanciato. Il primo motivo che impedisce il bilanciamento

del carico è legato al fatto che una sonda riceve solo il traffico della sottorete vicino cui è posizionata. Ci possono essere momenti in cui un tronco di rete è scarico e di conseguenza lo è la sonda che analizza quel traffico mentre altri tronchi di rete possono essere saturi con conseguente saturazione delle sonde. Il secondo motivo è la necessità di far elaborare tutti i pacchetti appartenenti alla stessa sessione a una sola sonda visto che questa deve mantenere lo stato della sessione la cui replicazione limiterebbe la scalabilità del sistema.

Il carico della rete varia durante la giornata e quindi può essere necessario e conveniente allocare dinamicamente le risorse hardware delle sonde. Variare durante l'esecuzione il numero di macchine dedicate all'elaborazione del traffico permette di ottimizzare l'uso delle risorse e di risparmiare energia spegnendo le macchine non utilizzate nei periodi in cui la rete registra un minore carico. In questa direzione vanno progetti di ricerca come Flowstream [47] che sfruttando macchine virtuali e switch OpenFlow [48] permettono l'allocazione dinamica delle risorse e la conseguente distribuire del traffico.

L'amministratore di rete interagendo con la console di monitoraggio visualizza le statistiche sul traffico aggiornate in tempo reale e ha la possibilità di selezionare la tipologia di traffico o i risultati di quali sonde considerare. Inoltre il sistema dà la possibilità di analizzare i dati storici selezionando l'intervallo temporale di cui si vogliono avere le statistiche. La console comunica con un collettore che si occupa di inoltrare le interrogazioni alle sonde e di ricevere da queste ultime le statistiche sul traffico analizzato.

Le sonde una volta avviate contattano il collettore cui fanno capo in modo da segnalare la loro presenza e ricevere la configurazione impostata dall'amministratore. Una volta effettuata la registrazione la sonda inizia l'elaborazione dei pacchetti che riceve in ingresso calcolando le statistiche sul traffico e salvando sul disco i dati sulle sessioni analizzate. Le statistiche, sia quelle real-time sia quelle storiche, vengono calcolate solo sulle sessioni selezionate dall'amministratore mediante l'uso di ACL. Allo stesso tempo tutte le sessioni catturate vengono memorizzate per poter essere filtrate e consultate in un secondo momento con criteri di selezione differenti. Una volta al secondo

la sonda esporta le statistiche verso il collettore attraverso il canale di comunicazione precedentemente aperto. Il collettore aggrega i risultati ricevuti dalle sonde e li invia alla console di monitoraggio. L'utilizzo di questo modello data parallel permette la distribuzione del carico su più macchine e la diminuzione del tempo di servizio con cui le sonde producono i risultati rispetto alla soluzione centralizzata con una sola sonda. Questo permette di avere sonde con requisiti computazionali inferiori e quindi un costo minore.

Visto il requisito del modello data parallel usato, che impone nell'operazione di reduce delle misure l'uso di operatori che godono della proprietà associativa, le metriche ossia le grandezze da misurare devono essere scelte di conseguenza. Nel paragrafo 3.2.4 vengono descritte e discusse le metriche scelte.

I nodi dell'architettura potrebbero diffondere informazioni che violano la privacy degli utenti. Per questo motivo può essere necessario rendere anonimi i dati o cifrare le comunicazioni e autenticare i nodi. Rendere anonimi i dati come ad esempio gli indirizzi IP [49] ha lo svantaggio di impedire agli amministratori di rete analisi legate alla posizione geografica delle sorgenti di traffico. Questo tipo di analisi è utile per combattere lo spam o per decidere come dislocare i server per offrire servizi più performanti per gli utenti e meno costosi per i fornitori. Se si sceglie di non rendere anonimi i dati catturati può essere necessario proteggerli durante le comunicazioni tra i nodi dell'architettura. Una buona soluzione è sfruttare il protocollo TLS (Transport Layer Security) [50] che permette di autenticare le connessioni e cifrare il traffico. In questo modo si evita che un malintenzionato si finga un nodo della rete per ricevere il risultato delle analisi o le intercetti durante la comunicazione.

Avendo come obiettivo il monitoraggio di reti ad alta velocità esiste la problematica di come gestire la quantità di dati ricevuti in un secondo. Nel caso di reti a 10 Gbps la velocità del bus PCIe degli attuali PC è sufficiente a trasferire gli 1,25 GB/s (half-duplex) dalla scheda di rete alla memoria permettendo quindi un'elaborazione real-time. Per quanto riguarda la memorizzazione il bus SATA 3 raggiunge la velocità massima di 6 Gbps (analoga a quella offerta da sistemi SCSI) che non è sufficiente

per la memorizzazione di tutti i pacchetti su una rete a pieno carico. La possibilità di memorizzazione del traffico viene inoltre ridotta dalla velocità effettiva dei dischi e dalla loro capienza. Nel caso di un link utilizzato all'80%, un'ora di traffico occuperebbe 3,6 TB.

Un'altra problematica è la quantità di dati inviata alla console di controllo. Nel caso di monitoraggio real-time le sonde che ricevono i pacchetti dovrebbero inviare gli aggiornamenti delle statistiche per ogni pacchetto generando una quantità di traffico pari a quella della rete che si sta monitorando. Se poi si hanno più sonde la rete che collega i dispositivi di monitoraggio dovrebbe avere prestazioni migliori di quella che si vuole monitorare.

La soluzione comunemente adottata è quella di aggregare i pacchetti in flussi cioè di considerare come unità di monitoraggio l'insieme dei pacchetti appartenenti a una stessa sessione. L'identificazione della sessione avviene mediante la 5-tupla riportata in figura 3.2 i cui campi sono estratti dagli header IP e TCP/UDP.

IP sorg.	IP dest.	Porta sorg.	Porta dest.	Protocollo

FIGURA 3.2: Campi che identificano un flusso

Questa scelta permette di ridurre notevolmente la quantità di dati da elaborare ma causa la perdita della possibilità di avere le informazioni in tempo reale visto che l'elaborazione dei dati non può essere effettuata prima del termine del flusso. Inoltre si perdono tutte le informazioni su eventuali picchi di traffico o aumenti del jitter in quanto si hanno solo i valori totali come: durata del flusso, numero di pacchetti e byte trasferiti.

Per ovviare a questi problemi nella tesi viene introdotto il concetto di microflusso. Un microflusso contiene le stesse informazioni di un flusso ma ha una durata temporale prefissata molto breve. Il loro vantaggio è che sono completi e quindi elaborabili al termine di un intervallo temporale definito a priori chiamato epoca. L'identificazione univoca dei microflussi avviene ancora una volta mediante i campi di figura 3.2 cui si aggiunge l'identificatore dell'epoca. La durata dell'epoca è stata scelta di un secondo in modo da avere un compromesso tra la riduzione della quantità di dati da memorizzare

e un sufficiente dettaglio sullo stato della rete. I risultati del monitoraggio saranno disponibili solamente dopo la fine dell'epoca e per questo motivo non si può parlare di monitoraggio real-time in senso stretto visto che passa del tempo tra l'istante in cui si verifica un evento e la sua visualizzazione sulla console di monitoraggio. Si parlerà quindi di monitoraggio near-realtime.

Con l'adozione dei microflussi la quantità di dati generati rispetto alle soluzioni che fanno uso di flussi aumenta. Per questo motivo non è possibile esportare tutti i microflussi verso il collettore. La soluzione scelta è quella di elaborare e memorizzare i microflussi sulle singole sonde ed esportare solamente le statistiche al termine dell'epoca.

I microflussi oltre a contribuire al calcolo delle statistiche vengono indicizzati e memorizzati nei dispositivi di storage locali ad ogni sonda. La scelta di mantenere i moduli di memorizzazione vicini alle sonde è dovuto al fatto che si vuole evitare di sovraccaricare la rete di monitoraggio. Inviando delle statistiche pre-elaborate al collettore si avranno messaggi di pochi KB inviati una sola volta al secondo ovvero al termine di ogni epoca di monitoraggio. Comunque se con un alto numero di sonde il fan-in o la capacità del link in ingresso al collettore non dovesse essere sufficiente questo può essere a sua volta distribuito nella rete mediante una struttura ad albero i cui nodi eseguono una prima fase della reduce delle statistiche.

3.2 Sonda di rete

La sonda è collegata a un dispositivo per il mirroring del traffico (splitter ottico o switch con port mirroring) e riceve una copia dei pacchetti che passano sul tronco di rete. Per permettere una corretta correlazione dei dati acquisiti dalle diverse sonde il riferimento temporale utilizzato deve essere uguale per tutte. Il problema è risolto utilizzando il protocollo NTP (vedi 2.2.1) per sincronizzare l'orologio dei nodi più volte durante il giorno.

Visto il grande numero di pacchetti ricevuti al secondo la sonda è la parte più critica del sistema che ne può limitare le prestazioni globali. Nella sua progettazione va tenuto conto delle seguenti problematiche:

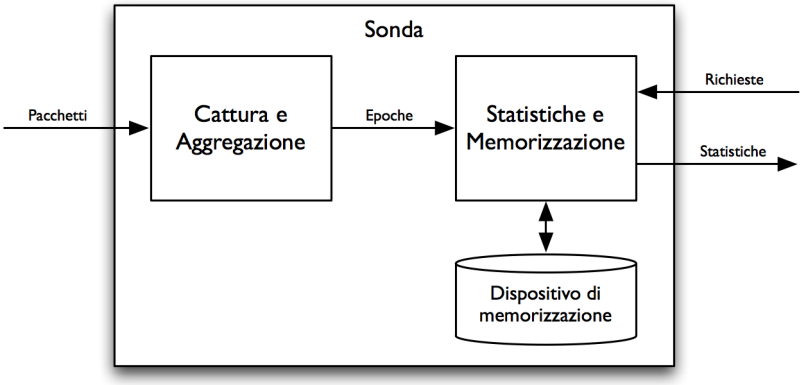


FIGURA 3.3: Schema della sonda

1. *Interrupt mitigation*

   Se la CPU è impegnata a gestire le interruzioni ogni volta che arriva un pacchetto non rimangono abbastanza risorse per le analisi.

2. *Sfruttamento del parallelismo*

   Ormai tutti i comuni PC hanno processori multi-core e multi-thread i cui vantaggi vanno sfruttati per aumentare le prestazioni della sonda.

3. *Uso efficiente della memoria*

   Visto il grande numero di microflussi creati e aggiornati ogni secondo è importante avere una loro rappresentazione in memoria compatta. Le motivazioni sono essenzialmente due:

   (a) Se non si ha una rappresentazione compatta si occupa velocemente tutta la memoria RAM a disposizione e il sistema operativo comincia ad effettuare lo swap su disco compromettendo drasticamente le prestazioni.

   (b) La diminuzione della dimensione del working set permette di ridurre i fault di cache, altra causa di degrado delle prestazioni.

4. Utilizzo di strutture dati efficienti

   L'utilizzo di strutture dati che offrono tempo di accesso costante anche nel caso pessimo permette di aumentare le prestazioni della sonda.

L'acquisizione del traffico avviene senza l'ausilio di hardware dedicato utilizzando la libreria PF_RING (vedi 2.2.2) che, oltre a risolvere i problemi legati alla gestione degli interrupt e alla copia in spazio utente dei pacchetti, permette di sfruttare il parallelismo dei sistemi multi-core. Grazie ai vantaggi offerti dalle architettura NUMA e da PF_RING è possibile eseguire un'istanza della sonda per ogni core disponibile e aspettarsi una buona scalabilità. Infatti sulle architetture NUMA è possibile eseguire le istanze della sonda su core differenti effettuando accessi in memoria in parallelo senza avere conflitti rendendo le istanze completamente indipendenti tra loro.

Per ridurre il tempo di servizio della sonda il modulo viene parallelizzato usando la forma di parallelismo pipeline (figura 3.3). Il primo stadio del pipeline si occupa di catturare e aggregare i pacchetti in microflussi e selezionare quali contribuiranno alla generazione delle statistiche calcolandone una prima parte (nel paragrafo 3.2.4 viene spiegato come è suddivisa la generazione delle statistiche). Al termine di un epoca tutti microflussi generati vengono inviati al secondo stadio che provvede ad elaborare la seconda parte delle statistiche, a memorizzare i microflussi sul disco e a comunicare con il collettore per inviare i risultati o ricevere una nuova configurazione. Il primo stadio è CPU-bound ma non può essere parallelizzato a causa della grana fine dei calcoli che effettua e della necessità di mantenere lo stato di tutti i flussi. Per questo motivo è importante avere una sua implementazione efficiente.

3.2.1 Strutture dati

La sonda per mantenere in memoria le informazioni sulle sessioni utilizza le seguenti strutture dati:

1. *Dizionario dei flussi attivi*
2. *Epoche* (figura 3.4)
3. *Microflussi*

Il dizionario contiene l'elenco dei flussi attivi e utilizza come chiave l'etichetta del flusso (5-tupla in figura 3.2) permettendo l'aggregazione dei pacchetti in flussi in tempo

costante. Un flusso è composto da un header contenente informazioni generali sulla sessione e dalla lista di microflussi che la compongono. Gli header dei flussi delle sessioni terminate possono essere utilizzati per l'esportazione di flussi NetFlow rendendo la sonda compatibile anche con questo standard. Il dizionario è implementato tramite una tabella hash chiamata Cuckoo Hashing (vedi paragrafo 2.2.3). Il vantaggio nell'usare questa soluzione scaturisce dal fatto che la maggior parte degli accessi al dizionario avvengono per operazioni di aggiornamento. Le tabelle Cuckoo Hashing al contrario dei dizionari implementati tramite liste concatenate o alberi hanno un tempo di ricerca costante. Con Cuckoo Hashing infatti le collisioni tra due chiavi diverse vengono risolte in fase di inserzione. Con liste concatenate le collisioni degradano le prestazioni di ogni accesso al dizionario mentre nel caso di alberi la complessità per l'estrazione degli elementi non scende sotto $O(\log n)$.

I campi che fanno parte dell'header del flusso sono:

Identificatore del flusso

La 5-tupla (SrcIP, DstIP, SrcPort, DstPort, Protocol).

Stato del flusso

Il flag che identifica lo stato del flusso. Può assumere i seguenti valori:

Invalid: il flusso non è valido.

New: è stato ricevuto il primo pacchetto appartenente al flusso.

Active: è stato ricevuto più di un pacchetto e la sessione non è ancora terminata.

Timedout: non sono stati ricevuti pacchetti appartenenti al flusso per più di 30 secondi, il flusso viene considerato scaduto e quindi terminato.

Fin: è stato ricevuto un pacchetto TCP con il flag FIN settato, il sistema deve attendere un pacchetto con flag ACK prima che la sessione possa essere considerata chiusa.

Closed: la sessione è terminata o a causa della ricezione di un pacchetto con flag FIN seguito da un pacchetto con flag ACK oppure a causa della ricezione di un pacchetto con flag RST.

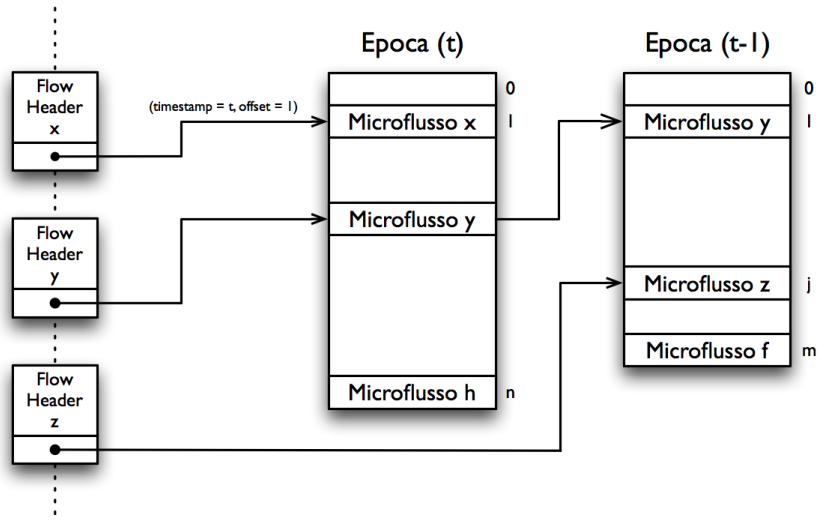


FIGURA 3.4: Lista delle epoche

Timestamp del primo pacchetto ricevuto (in secondi)

L'orario di inizio della costruzione del microflusso.

Timestamp dell'ultimo pacchetto ricevuto (in secondi)

L'orario dell'ultimo pacchetto ricevuto.

Totale pacchetti ricevuti

La somma dei pacchetti ricevuti in tutti i microflussi che compongono la sessione.

Totale byte trasferiti

La somma dei byte trasferiti in tutti i microflussi che compongono la sessione.

Offset nell'epoca

La posizione del microflusso all'interno dell'ultima epoca in cui è presente.

Nell'header hanno particolare importanza due campi: *Timestamp dell'ultimo pacchetto ricevuto* e *Offset nell'epoca*. Il primo, essendo espresso in secondi e avendo epoche della durata di un secondo, permette di identificare l'epoca in cui è presente l'ultimo microflusso facente parte della sessione. Il secondo campo invece permette di conoscere la posizione del microflusso all'interno dell'epoca. In figura 3.4 si vede che la sessione x ha ricevuto almeno un pacchetto durante l'epoca t e il microflusso si trova nella posizione 1 all'interno dell'epoca. La sessione z invece non ha ricevuto pacchetti

nell'ultimo secondo ma li ha ricevuti nell'epoca t-1. La scelta di utilizzare questo tipo di riferimenti è stata fatta per ottimizzare l'utilizzo della memoria e per svincolarli dal modo in cui sono rappresentati in memoria in modo da poter essere memorizzati su disco senza ulteriori modifiche.

Un certo numero di epoche devono essere mantenute in memoria per la generazione delle statistiche. Ogni epoca contiene l'elenco dei microflussi attivi nell'intervallo temporale cui si riferisce e un intestazione contenente i seguenti campi:

Timestamp

L'orario di inizio dell'epoca.

Filtro

Il filtro applicato durante l'epoca per la selezione dei dati su cui calcolare le statistiche.

Statistiche

Le statistiche generate durante l'epoca.

Numero di microflussi nell'epoca

Utilizzato come indice di inserzione per i nuovi microflussi.

Ogni microflusso contiene i valori aggregati durante un'epoca dei pacchetti appartenenti allo stesso flusso. Di seguito vengono descritti i campi che lo compongono:

Pacchetti ricevuti

La somma dei pacchetti ricevuti durante l'epoca corrente.

Byte trasferiti

La somma dei byte trasferiti durante l'epoca corrente.

Offset epoca precedente

Il numero di epoche passato dalla ricezione dell'ultimo pacchetto appartenente alla sessione. È un numero compreso tra 1 e il timeout fissato per le sessioni.

Offset del microflusso precedente

La posizione del microflusso all'interno dell'ultima epoca in cui è presente.

Flag TCP

Nel caso di una sessione TCP contiene l'OR dei flag dei pacchetti trasferiti durante l'epoca.

Stato del filtro

Indica se il microflusso partecipa o meno alla generazione delle statistiche durante l'epoca.

Riferimento al flusso

Necessario nei casi in cui si debba accedere all'intestazione del flusso durante l'analisi di un microflusso.

3.2.2 Descrizione delle attività

Il diagramma della attività UML del primo stadio della sonda è mostrato in figura 3.5. Alla ricezione di un pacchetto si controlla se questo fa parte dell'epoca attualmente attiva o se fa parte di una nuova epoca. In quest'ultimo caso l'epoca attiva viene inviata allo stadio successivo e ne viene creata una nuova. Al momento della creazione dell'epoca il suo header viene inizializzato e viene copiato l'identificatore del filtro ricevuto dal collettore. Il secondo stadio lavorerà sulle epoche ricevute dal primo stadio il quale non ha più bisogno di accedere a quelle epoche. Questo permette di evitare di sincronizzare gli accessi alle epoche e ai microflussi che le compongono. L'unico momento in cui i due stadi hanno bisogno di accedere alla stessa struttura dati è la fase di rimozione dei flussi scaduti dal dizionario. Il numero di accessi concorrenti al dizionario è comunque trascurabile visto che mantenendo una lista LRU dei flussi il secondo stadio accederà solamente ai flussi scaduti.

Dal pacchetto ricevuto viene estratto l'identificatore del flusso che viene usato per calcolare la funzione hash con cui accedere al dizionario. Se il flusso non è presente nel

dizionario viene fatto un nuovo inserimento marcando il flusso come *New* in modo da sapere che è un nuovo flusso e deve ancora essere aggiornato.

Se il pacchetto fa parte di un nuovo flusso o se il campo *Timestamp dell'ultimo pacchetto ricevuto* è diverso dal timestamp dell'epoca corrente deve essere creato un nuovo microflusso. Per la creazione del nuovo microflusso si aggiunge un entry nell'epoca corrente e si aggiornano i campi *Timestamp dell'ultimo pacchetto ricevuto* e *Offset nell'epoca* nell'header del flusso. In caso contrario si accede al microflusso e si aggiornano i suoi campi in base al pacchetto ricevuto. Nel caso di sessioni TCP viene aggiornato il campo contenente l'OR dei flag dell'header TCP e viene rilevata l'eventuale terminazione del flusso. Se il flusso è terminato il suo header può essere rimosso dal dizionario. In base alla valutazione dello stato del filtro (vedi 3.2.3) viene deciso se aggiornare o meno le statistiche dell'epoca. Al termine dell'aggiornamento del microflusso vengono infine aggiornati i campi del flusso: *Stato del flusso*, *Totale pacchetti ricevuti* e *Totale byte trasferiti*.

Il secondo stadio della sonda è descritto dal diagramma in figura 3.6. Al termine di ogni epoca vengono generate le statistiche che non possono essere generate nel primo stadio poiché richiedono l'analisi di epoche precedenti. I flussi scaduti vengono rimossi dal dizionario e inseriti nel database Fastbit (vedi paragrafo 2.2.4). Le epoche invece vengono memorizzate su disco senza modifiche visto che tutti i campi necessari alle interrogazioni storiche non contengono puntatori a zone di memoria che non sarebbero più validi una volta memorizzati su disco.

Uno dei compiti del secondo stadio è quello di comunicare con il collettore sia per ricevere da questo richieste di configurazione o interrogazioni sia per inviare le statistiche. Nel caso della ricezione di un nuovo filtro il primo stadio viene notificato in modo da applicare la nuova configurazione nell'epoca successiva a quella di ricezione. Nel caso di interrogazioni storiche vengono eseguite le query sul database Fastbit in base al filtro e all'intervallo temporale richiesto. Una volta estratti gli header dei flussi vengono lette dal disco le epoche contenenti i microflussi necessari alla generazione delle statistiche e i risultati inviati alla console.

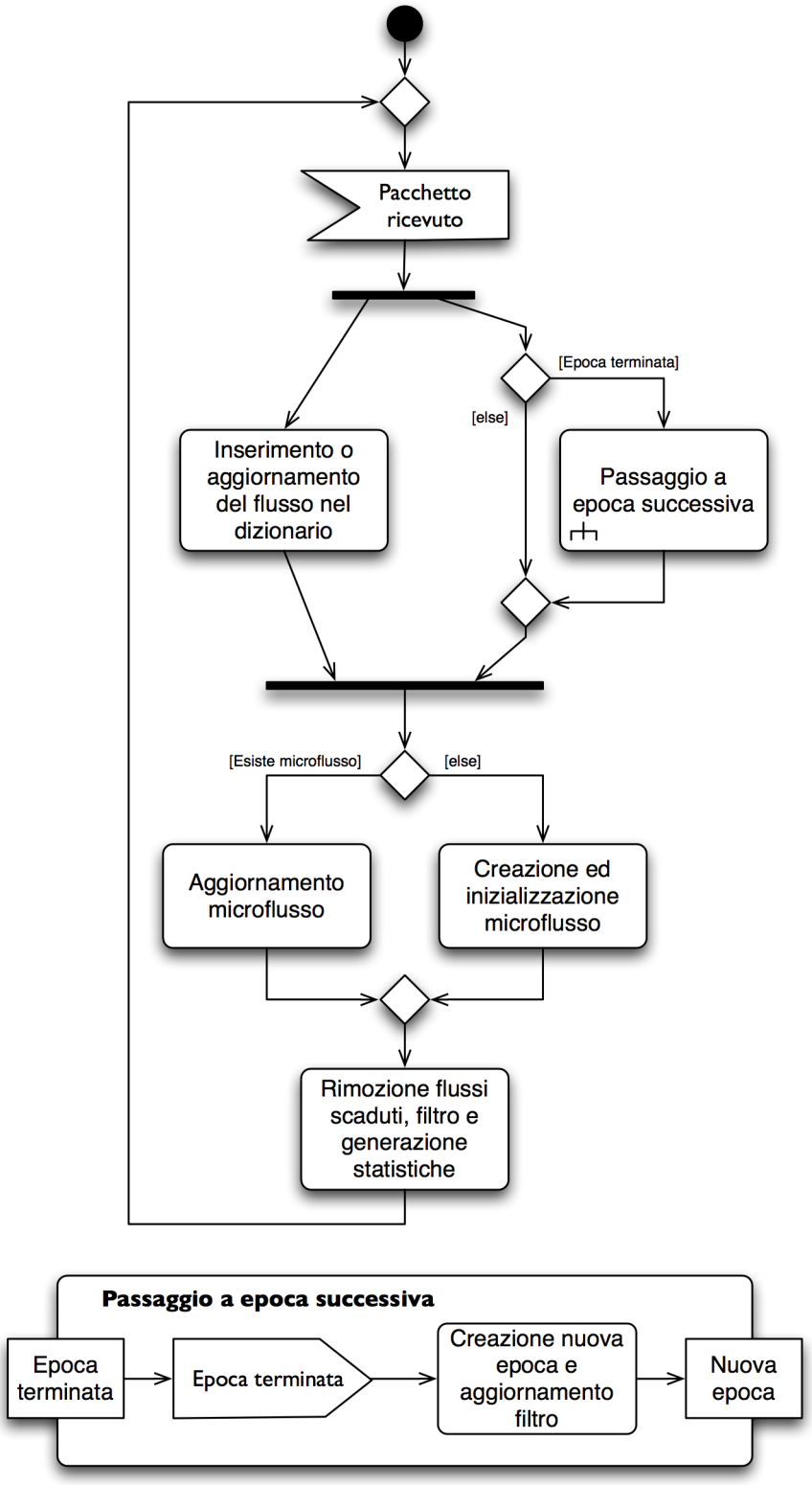


FIGURA 3.5: Diagramma delle attività del primo stadio della sonda

3.2.3 Selezione del traffico

L'amministratore deve poter selezionare su quali flussi devono essere calcolate le statistiche e quali sonde devono comunicare le statistiche al collettore. Per questo motivo tramite la console viene data la possibilità di scrivere un Access Control List (ACL) che permette di selezionare i flussi interessanti in base al loro header e di indicare l'elenco delle sonde da cui si vogliono ricevere i risultati. Il collettore riceve l'ACL dalla console e la invia a tutte le sonde selezionate. I campi che compongono l'ACL sono:

- IP/Subnet Mask sorgente
- IP/Subnet Mask destinatario
- Intervallo porte sorgente

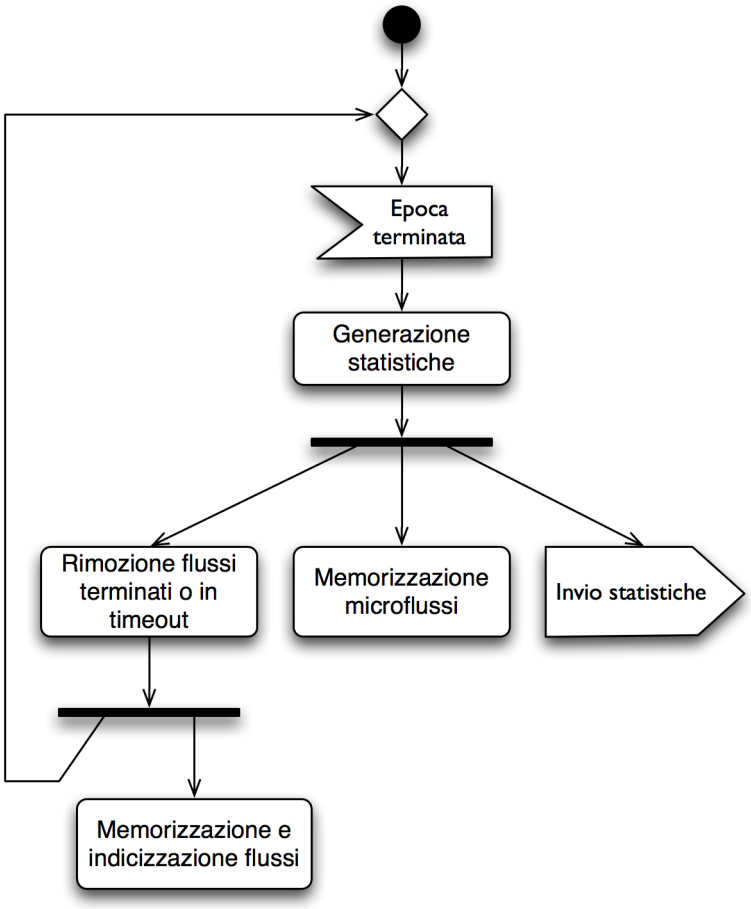


FIGURA 3.6: Diagramma delle attività del secondo stadio della sonda

- Intervallo porte destinatario
- Protocollo
- Stato del flusso
- Azione

Ogni volta che la sonda deve controllare se un microflusso partecipa alla generazione delle statistiche la lista delle ACL viene scandita confrontando i campi di ogni riga con l'header del flusso. Per ogni riga dell'ACL se i campi dell'header del flusso corrispondono a quelli della riga l'azione specificata viene applicata altrimenti si passa alla riga successiva. Il campo azione permette di definire se il flusso deve essere incluso (*Permit*) o meno (*Deny*) nelle statistiche. Nel caso in cui tutta l'ACL sia analizzata senza trovare una regola applicabile viene applicata l'azione predefinita *Deny*.

Le sonde possono avere una sola ACL attiva composta da un numero arbitrario di righe. Quando viene inviata la richiesta di filtraggio il collettore assegna un identificatore univoco associato all'ACL. Questo identificatore viene ripetuto nel messaggio contenente le statistiche generate dalla sonda in modo da sapere quale filtro è stato applicato per ottenere le statistiche. Quando vengono cambiati i criteri di selezione le sonde ricevono l'ACL dal collettore ma la applicano solamente al termine dell'epoca corrente. In questo modo tutti i microflussi dell'epoca sono filtrati secondo lo stesso criterio e hanno lo stesso identificatore del filtro.

Il numero di filtri impostati influenza il tempo richiesto per controllare se un microflusso deve essere incluso nel calcolo delle statistiche. Per ridurre il numero di controlli dell'ACL si marcano i flussi e i microflussi che hanno passato il filtro in modo che per i flussi attivi sia necessario un solo confronto per sapere se il pacchetto deve contribuire alle statistiche o meno. La marcatura consiste nello specificare l'identificatore univoco associato all'ACL analizzata e l'azione applicata. L'ACL viene scandita nei seguenti casi:

1. *Il pacchetto ricevuto appartiene a una nuova sessione*

   È stato ricevuto il primo pacchetto appartenente a un nuovo flusso quindi va filtrato e marcato il flusso.

2. *L'identificatore univoco di filtraggio è cambiato*

   È stato ricevuto il primo pacchetto dell'epoca appartenente a un flusso il cui identificatore del filtro è diverso da quello richiesto per l'epoca. Questo caso si verifica per tutti i flussi attivi nell'epoca successiva al cambiamento del filtro.

Il controllo dell'ACL viene fatto nel primo stadio del pipeline perché è necessario accedere al dizionario per marcare i flussi che passano il filtro e per conoscere lo stato del filtro dei flussi attivi. In questo modo si limita il numero di accessi concorrenti al dizionario che potrebbero serializzare l'esecuzione dei due stadi della sonda.

3.2.4 Metriche utilizzate

Di seguito vengono elencate e descritte le metriche misurate dalle sonde per ogni epoca. Come detto nel paragrafo 3.2.3, partecipano al calcolo delle statistiche solamente i flussi che passano la fase di filtraggio. Nel primo stadio della sonda vengono fatte tutte le misure che non richiedono l'analisi delle epoche precedenti o l'iterazione sul dizionario ovvero tutte quelle statistiche che possono essere calcolate in tempo costante. Le statistiche generate nel primo stadio e inviate con l'epoca al secondo stadio sono:

Stato dei flussi

Per ogni possibile stato viene calcolato il numero di flussi che si trova in quello stato. I possibili stati sono:

Nuovo flusso: quanti microflussi non appartenenti a un flusso presente nel dizionario sono stati creati durante l'epoca.

Flusso attivo: il numero di flussi attivi ovvero il numero di microflussi creati durante l'epoca.

Flusso terminato: il numero di flussi terminati in seguito alla ricezione di un pacchetto TCP con flag RST o della sequenza FIN/ACK.

Dati trasferiti

La somma dei dati trasferiti durante l'epoca.

Pacchetti trasferiti

La somma dei pacchetti trasferiti durante l'epoca.

Nel secondo stadio vengono calcolate le seguenti statistiche:

Flussi scaduti

Per calcolare questa metrica gli header dei flussi vengono mantenuti in una lista LRU. Al termine di ogni epoca la lista viene scandita controllando il timestamp dell'ultimo pacchetto ricevuto e rimuovendo i flussi scaduti dal dizionario. Usando la lista ordinata secondo il criterio LRU vengono scanditi solamente i flussi scaduti visto che appena si trova un flusso non scaduto la scansione può essere interrotta.

Flussi che hanno generato più traffico

L'elenco dei *Top-k* flussi che hanno generato più traffico nelle ultime n epoche. Questa metrica viene calcolata su una finestra scorrevole la cui dimensione è limitata dalla memoria a disposizione. Infatti per effettuare la ricerca dei *Top-k* flussi vanno mantenute in memoria tutte le epoche appartenenti alla finestra.

Flussi con più pacchetti trasferiti

L'elenco dei *Top-k* flussi che hanno trasferito più pacchetti nelle ultime n epoche. Valgono le considerazioni fatte in precedenza.

Tutte le metriche vengono inviate al collettore allo scadere di ogni epoca in modo da avere un continuo aggiornamento sullo stato della rete. Il calcolo dei Top-k flussi avviene usando *sliding window* che, a differenza di altre soluzioni che utilizzano *tumbling window*, ha il vantaggio di avere aggiornamenti allo scadere di ogni epoca anche se questo comporta un maggiore utilizzo di memoria dovendo mantenere in RAM tutte le epoche facenti parte della finestra.

Nella definizione delle metriche si è tenuto conto del fatto che la reduce si può applicare solamente nel caso di operatori associativi. Quindi ad esempio non è possibile

sapere la dimensione media dei pacchetti che passano sulla rete inviando al collettore la dimensione media dei pacchetti rilevati da ogni sonda. È necessario inviare il totale dei dati e del numero di pacchetti trasferiti facendo calcolare la media al collettore o alla console.

La reduce delle metriche le cui misure sono scalari o liste di scalari viene eseguita semplicemente facendo la loro somma. La reduce dei Top-k flussi viene invece fatta applicando l'operatore $max(a, b)$ sui campi dei flussi da confrontare. Sia l'operatore somma che l'operatore max godono della proprietà associativa a patto che i flussi considerati sulle sonde siano indipendenti. Vale a dire che i pacchetti di una sessione devono essere analizzati tutti e solo dalla stessa sonda o trattati come due sessioni differenti.

Terminata la generazione delle statistiche queste vengono inviate al collettore insieme all'identificatore della sonda (Probe ID) e all'identificatore del filtro applicato (Filter ID).

3.2.5 Memorizzazione dei dati storici

Al termine di ogni epoca si ha l'elenco dei microflussi pronto per essere memorizzato su disco. Per ridurre la dimensione occupata, ogni epoca viene compressa e scritta sequenzialmente in un file. Ogni 10 minuti viene creato un nuovo file contenente 600 epoche. Nell'intestazione del file vengono memorizzati gli offset dove iniziano le epoche in modo che in fase di lettura possano essere caricate solo quelle necessarie. Il nome del file identifica l'intervallo temporale delle epoche che contiene. Per la compressione è stata scelta la libreria QuickLZ[1] per via delle sue prestazioni in fase di compressione e decompressione. Questa libreria raggiunge i 308 MByte/s utilizzando un solo core di un processore Intel Core i7 920.

Il secondo stadio della sonda si occupa di raggruppare i flow header e di inserirli nel database. I flow header possono essere memorizzati su disco solamente quando il flusso è terminato ovvero quando si trova nello stato *Closed* o *Timedout* e non devono più essere modificati. Questo perché per la loro indicizzazione è stato scelto di utilizzare il

[1] http://www.quicklz.com/

database FastBit che ha ottime prestazioni in fase di inserimento e ricerca ma solamente se i dati inseriti non variano nel tempo.

L'interrogazione dei dati storici può essere effettuata solamente sui flussi terminati e già inseriti nel database. Quando viene inviata alla sonda una richiesta che si riferisce all'archivio storico viene prima di tutto interrogato il database FastBit il quale restituirà l'elenco dei flussi cui si è interessati. Estraendo il minimo timestamp iniziale e il massimo timestamp finale tra tutti i flussi si possono leggere e decomprimere le epoche dai file. Analizzando le epoche vengono generate le statistiche nell'intervallo temporale selezionato e il risultato viene inviato al collettore.

3.3 Collettore

Il collettore si occupa di comunicare con la console di monitoraggio e con le sonde. Dalla console riceve i messaggi di configurazione mentre dalle sonde riceve i risultati real-time o i risultati delle interrogazioni sullo storico. Il collettore riceve richieste di connessione e registrazione dalle sonde e dalle console di monitoraggio. Quando una sonda si registra sul collettore questo invia la configurazione attiva e in particolare l'ACL con cui deve essere selezionato il traffico. La stessa configurazione viene inviata anche quando si registra una console in modo che sappia come sono filtrati i dati che riceverà. Nel caso di più console di monitoraggio connesse è possibile avere un solo filtro attivo quindi quando da una console viene impostato un filtro le altre ricevono e mostrano l'avvenuto aggiornamento.

Il collettore riceve le statistiche e l'identificatore della sonda che le ha inviate. Per ogni epoca raccoglie i messaggi con le statistiche dalle sonde e applicando la reduce genera le statistiche aggregate da inviare alla console di monitoraggio.

Allo scadere di ogni epoca, tutte le sonde comunicano al collettore le statistiche. La banda richiesta per le comunicazioni e le risorse necessarie a gestire le comunicazioni con centinaia di sonde possono limitare la scalabilità dell'architettura. Per ovviare a questo problema il collettore può essere parallelizzato realizzando la struttura ad albero di figura 3.7 dove le sonde vengono divise in gruppi facenti capo a collettori

diversi. I collettori intermedi invieranno le statistiche aggregate al collettore centrale che a sua volta le invierà alla console di monitoraggio. Questa struttura permette una buona scalabilità dell'architettura anche se fa aumentare in modo logaritmico, dipendente dall'arietà dell'albero dei collettori, la latenza tra il verificarsi dell'evento e la visualizzazione della misura.

Nel caso dell'utilizzo di sistemi multi-core, se si allocano più istanze della sonda su ogni macchina, può essere conveniente eseguire anche un istanza del collettore in locale in modo da avere i dati già aggregati in uscita dal sistema.

Al termine di ogni epoca le sonde inviano un messaggio al collettore che li raccoglie e invia il risultato entro un certo tempo limite alla console. I messaggi che arrivano oltre il tempo limite vengono scartati segnalando quali sono le sonde che non stanno trasmettendo. Questa segnalazione avviene finché il canale di comunicazione tra sonda e collettore è attivo in modo da sapere se è la sonda che non riesce a comunicare il risultato in tempo utile oppure se la connessione ha smesso di funzionare. Nel primo caso la soluzione è aggiungere altre sonde distribuendo il traffico in modo da fare diminuire il tempo di servizio della sonda. Nel secondo caso significa che si è verificato un guasto sulla sonda o sulla rete utilizzata per collegare la sonda al collettore ed è necessario un intervento tecnico.

La radice del collettore rappresenta un Single Point of Failure (SPF). Nel caso si verifichi un guasto, con la struttura ad albero proposta, questo è tanto più dannoso quanto più si verifica vicino alla radice. Infatti un guasto su una sonda blocca i risultati ricevuti solamente da quella sonda, un guasto su un collettore intermedio blocca i

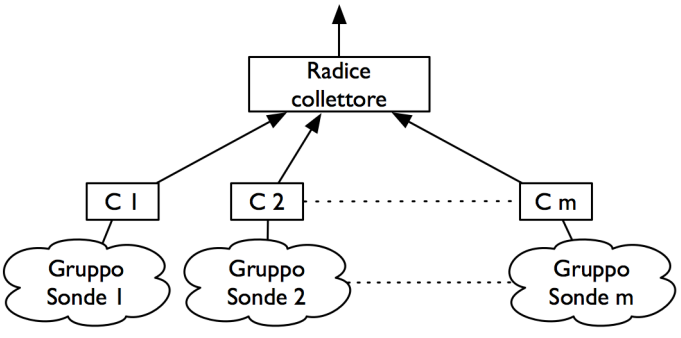


FIGURA 3.7: Schema collettore ad albero

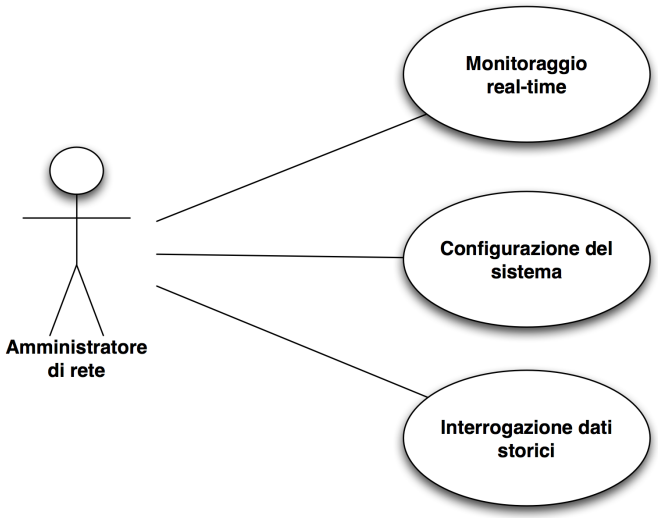


FIGURA 3.8: Caso d'uso del sistema

risultati su una parte della rete mentre un guasto sulla radice del collettore blocca i risultati di tutta la rete. D'altra parte l'unico guasto che provoca perdita di dati è quello delle sonde. Infatti grazie alla conservazione dell'archivio storico quando un guasto viene ripristinato si hanno di nuovo a disposizione i dati raccolti.

3.4 Console di monitoraggio

La console ha il compito di interfacciare l'amministratore di rete al sistema di monitoraggio. Tramite la console l'amministratore può visualizzare le statistiche in tempo reale, impostare filtri sul traffico da analizzare e interrogare l'archivio storico per effettuare analisi sugli eventi passati (figura 3.8).

Il sistema non pone limiti al numero di console. L'unico limite è sulla selezione del traffico, infatti com'è stato spiegato ci può essere un solo criterio di selezione attivo. Quando una console imposta un criterio di selezione a tutte le altre viene notificato il cambiamento e ricevono i dati filtrati di conseguenza.

È stato scelto di utilizzare una console basata su tecnologie web per i vantaggi che queste offrono:

1. *Cross-platform:* il sistema può essere consultato e configurato indipendentemente dall'hardware o dal sistema operativo utilizzato per eseguire la console di

amministrazione.

2. *Requisiti hardware:* la console di amministrazione può avere minori requisiti computazionali permettendo la consultazione del sistema anche da dispositivi embedded come gli smartphone.

3. *Aggiornamenti del sistema:* la modifica del sistema non comporta la modifica delle console di monitoraggio semplificando quindi la distribuzione del software.

La presentazione dei dati in tempo reale avviene grazie alle tecniche di data delivery su web che permettono di effettuare il push dei dati dal collettore al browser (vedi 2.2.5). Lo stream di statistiche aggregate dal collettore viene inviato al browser che genera i grafici e le tabelle utili all'amministratore per le analisi sullo stato della rete.

3.5 Messaggi

In questo paragrafo viene descritto il formato dei messaggi che i nodi del sistema si scambiano. Tutte le comunicazioni avvengono mediante connessioni TCP che garantiscono la consegna dei messaggi gestendo le ritrasmissioni, il controllo del flusso e il controllo della congestione.

3.5.1 Sonda-Collettore

I messaggi inviati tra sonde e collettore sono composti da un header lungo 4 byte (figura 3.9), contenente due campi che specificano il tipo di messaggio e la sua lunghezza, e dal payload che varia da messaggio a messaggio. Il campo contenente la lunghezza occupa 3 byte permettendo messaggi lunghi fino a 16 MB. In questo paragrafo vengono definiti i messaggi che le sonde scambiano con il collettore e i messaggi che i collettori scambiano tra loro.

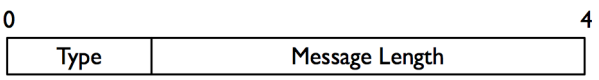


FIGURA 3.9: Header dei messaggi

Registrazione sonda

Inviato dalla sonda quando si collega al collettore, contiene l'ID della sonda (che occupa 2 byte). Il collettore che lo riceve aggiunge l'ID della sonda alla lista delle sonde connesse. Se il collettore non è quello radice il messaggio viene inoltrato al collettore padre e propagato verso la radice. In questo modo il collettore alla radice dell'albero conosce l'elenco di tutte le sonde registrate.

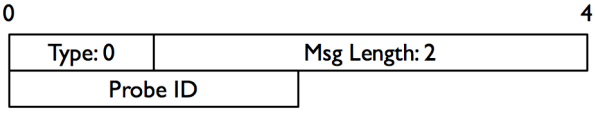


FIGURA 3.10: Registrazione Sonda

Disconnessione sonda

Inviato da una sonda prima di disconnettersi o, nel caso ci siano più collettori, da un collettore che rileva la disconnessione di una sonda. Anche in questo caso il messaggio che contiene l'ID della sonda viene propagato verso la radice dell'albero.

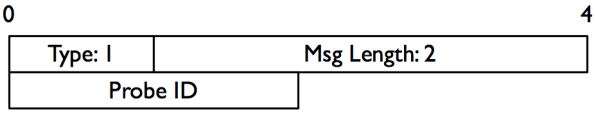


FIGURA 3.11: Disconnessione Sonda

Interruzione invio statistiche

Interrompe sia l'invio delle statistiche in tempo reale che l'invio dei risultati delle interrogazioni dell'archivio contenente i dati storici. Il messaggio contiene l'elenco degli ID delle sonde la cui lunghezza è ottenuta leggendo la dimensione del payload.

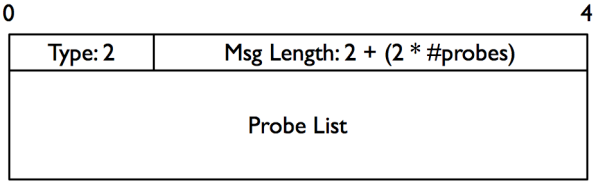


FIGURA 3.12: Interruzione invio statistiche

Invio statistiche

I messaggi di questo tipo (figura 3.13(a)) contengono le statistiche calcolate dalle sonde o quelle aggregate dai collettori intermedi. Sono inviati al termine di ogni epoca nel caso di monitoraggio in tempo reale o appena disponibili nel caso di interrogazioni dell'archivio storico. I campi contenuti nel messaggio sono i seguenti:

Timestamp: intero di 4 byte che identifica l'epoca.

Filter ID: intero di 1 byte che identifica il filtro applicato per selezionare i flussi che hanno contribuito alla generazione delle statistiche.

Statistiche dell'epoca: sei interi contenenti il numero di nuovi flussi, il numero di microflussi attivi, il numero di flussi terminati, il numero di flussi scaduti, il numero di byte trasferiti e il numero di pacchetti trasferiti (figura 3.13(b)).

Top Flows: elenco dei 10 flussi che hanno generato più traffico e dei 10 flussi con il maggior numero di pacchetti trasferiti negli ultimi 30 secondi.

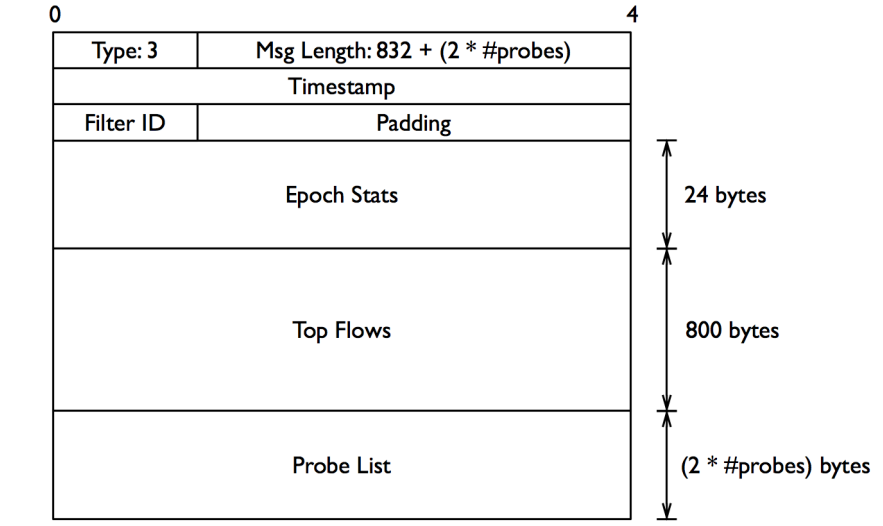
Elenco delle sonde: la lista degli ID delle sonde che hanno contribuito alle statistiche.

Ogni flusso (figura 3.13(c)) occupa 40 byte quindi i 20 flussi esportati occupano un totale di 800 byte. Tra i campi esportati con il flusso c'è *Top Value* che rappresenta il valore misurato nell'intervallo di tempo su cui sono stati estratti i Top-k flussi. Nel caso dei flussi che hanno generato più traffico è la quantità di byte trasferiti negli ultimi 30 secondi da quel flusso.

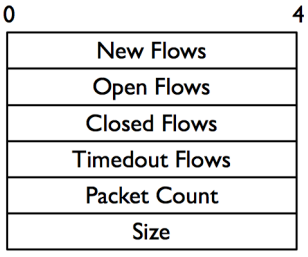
Invio ACL

Questo messaggio (figura 3.14(a)) viene inviato quando una nuova sonda si collega al collettore o quando l'amministratore imposta un nuovo criterio di selezione del traffico. Il messaggio contiene i seguenti campi:

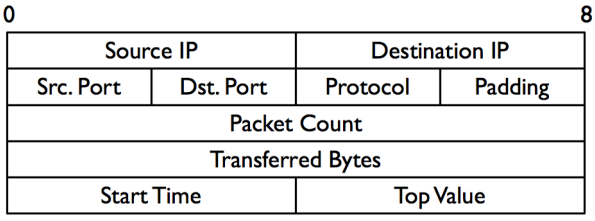
Start timestamp: intero di 4 byte che specifica l'epoca iniziale da cui estrarre le statistiche storiche. Impostando il valore a zero si imposta il filtro per le interrogazioni in tempo reale.



(a) Formato del messaggio



(b) Epoch Stats



(c) Top Flow

FIGURA 3.13: Statistiche Sonda-Collettore

End timestamp: intero di 4 byte che specifica l'epoca finale da cui estrarre le statistiche storiche. Nel caso di interrogazioni in tempo reale anche questo campo viene posto a zero.

Filter ID: intero di 1 byte che identifica la richiesta inviata. Questo ID è scelto dal collettore e inserito nella richiesta. La console riceverà l'ID tramite la notifica inviata dal collettore per comunicare l'accettazione dell'ACL.

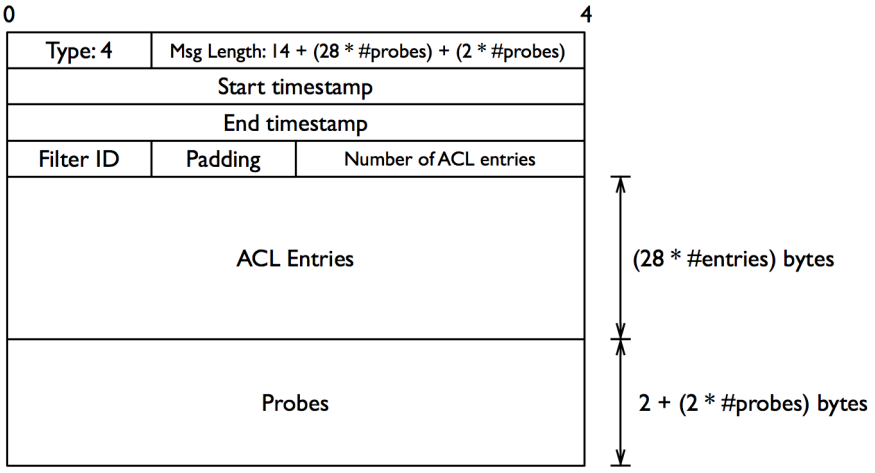
Numero di regole: un intero di 2 byte che specifica il numero di regole che compongono l'ACL da applicare.

Elenco delle regole: l'elenco delle regole che fanno parte della richiesta. Ogni regola occupa 28 byte e i suoi campi sono mostrati in figura 3.14(b).

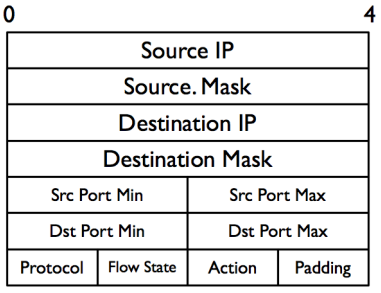
Numero di sonde: un intero di 2 byte che specifica il numero di sonde che devono

inviare i risultati. Se si imposta il valore a zero i risultati saranno inviati da tutte le sonde. Se il valore è invece maggiore di zero tutte le sonde non incluse nel filtro interrompono l'invio delle statistiche.

Elenco delle sonde: l'elenco degli ID delle sonde che devono inviare le statistiche.



(a) Formato del messaggio



(b) ACL Entry

FIGURA 3.14: ACL

3.5.2 Collettore-Console

Le comunicazioni tra il collettore radice e la console avvengono utilizzando messaggi in formato JSON (JavaScript Object Notation). Questi oltre ad avere il vantaggio di essere molto semplici sono direttamente interpretabili dal browser visto che sono una rappresentazione di oggetti JavaScript.

Ogni messaggio scambiato tra collettore e console consiste in un oggetto JSON contenente due proprietà:

1. `type:` una stringa utilizzata per determinare il tipo di messaggio.
2. `message:` un oggetto che rappresenta il contenuto del messaggio.

I messaggi definiti sono i seguenti:

Interruzione invio statistiche

Tipo del messaggio: `stop_probes`

Proprietà:

Nome	**Tipo**	**Descrizione**
`probes`	array di interi	elenco degli ID delle sonde da interrompere.

Invio statistiche

Tipo del messaggio: `send_stats`

Proprietà:

Nome	**Tipo**	**Descrizione**
`registered_probes`	array di interi	elenco degli ID delle sonde registrate.
`probes`	array di interi	elenco degli ID delle sonde che hanno contribuito alle statistiche.
`timestamp`	intero	orario in cui è iniziata l'epoca cui si riferiscono le statistiche.
`filter_id`	intero	ID dell'ACL applicata durante il calcolo delle statistiche.
`epoch_stats`	oggetto	le statistiche dell'epoca.

L'oggetto `epoch_stats` contiene le seguenti proprietà:

Nome	**Tipo**	**Descrizione**
`new_flows`	intero	il numero di nuovi flussi ricevuti.
`open_flows`	intero	il numero di microflussi attivi.
`closed_flows`	intero	il numero di flussi terminati.
`timedout_flows`	intero	il numero di flussi scaduti.
`pkt_count`	intero	il numero di pacchetti transitati.
`size`	intero	il numero di byte trasferiti.
`top_pkts`	array di oggetti	i 10 flussi che hanno trasferito più pacchetti.
`top_size`	array di oggetti	i 10 flussi che hanno trasferito più byte.

Ogni flusso è rappresentato da un oggetto con le seguenti proprietà:

Nome	**Tipo**	**Descrizione**
`src_ip`	stringa	l'indirizzo IP della sorgente.
`src_port`	intero	la porta della sorgente.
`dst_ip`	stringa	l'indirizzo IP del destinatario.
`dst_port`	intero	la porta del destinatario.
`protocol`	stringa	il nome del protocollo utilizzato.
`start_time`	intero	orario di inizio del flusso.
`pkt_count`	intero	numero di pacchetti del flusso.
`flow_size`	intero	numero di byte trasferiti dal flusso.
`top_value`	intero	il valore misurato sul flusso per la sua selezione.

Invio ACL

Tipo del messaggio: `set_filter`

Proprietà:

Nome	**Tipo**	**Descrizione**
`start_timestamp`	intero	orario iniziale dell'intervallo di interrogazione storica o zero per interrogazioni in tempo reale.
`end_timestamp`	intero	orario finale dell'intervallo di interrogazione storica o zero per interrogazioni in tempo reale.
`acl_entries`	array di oggetti	elenco delle regole da applicare.
`probes`	array di interi	elenco degli ID delle sonde da cui si vogliono ricevere le statistiche o `null` se si vuole applicare il filtro a tutte le sonde.

Le regole dell'ACL sono oggetti con le seguenti proprietà:

Nome	**Tipo**	**Descrizione**
`src_ip`	stringa	l'indirizzo IP della sorgente.
`src_mask`	stringa	la subnet mask della sorgente.
`dst_ip`	stringa	l'indirizzo IP del destinatario.
`dst_mask`	stringa	la subnet mask del destinatario.
`src_port_min`	intero	l'estremo inferiore dell'intervallo di porte della sorgente.
`src_port_max`	intero	l'estremo superiore dell'intervallo di porte della sorgente.
`dst_port_min`	intero	l'estremo inferiore dell'intervallo di porte del destinatario.
`dst_port_max`	intero	l'estremo superiore dell'intervallo di porte del destinatario.
`protocol`	intero	il protocollo usato ("TCP" = 6, "UDP" = 17 o "ANY" = 0).
`flow_state`	stringa	lo stato del flusso.
`action`	stringa	l'azione da applicare.

In questo capitolo è stata presentata l'architettura DRT-Mon, il funzionamento dei suoi componenti e il formato dei messaggi scambiati. Nel prossimo capitolo verrà presentato il prototipo realizzato per validare l'architettura, saranno valutare le sue prestazioni e sarà verificato il raggiungimento degli obiettivi definiti nel capitolo 1.

Capitolo 4

Validazione

Nella prima parte di questo capitolo saranno descritti i dettagli sulla realizzazione dell'architettura DRT-Mon. In seguito verranno valutate le prestazioni e verificato il raggiungimento degli obiettivi prefissati nel paragrafo 1.7 comparando la soluzione proposta a quelle esistenti.

4.1 Dettagli sulla realizzazione

Lo scopo della tesi è di riuscire a monitorare reti ad alta velocità in tempo reale facendo uso di commodity hardware. Il software sviluppato è stato realizzato utilizzando come architettura hardware di riferimento l'AMD64. Ogni nodo del sistema (figura 4.1) è rappresentato da un comune PC con schede di rete a 1 o 10 Gbps.

4.1.1 Sonda

Dal punto di vista delle prestazioni la sonda è la parte più critica di tutto il sistema. Per la sua realizzazione è stato scelto il linguaggio C visto il controllo che offre sulle risorse hardware e in particolare sulla gestione della memoria. La cattura dei pacchetti dalla rete avviene utilizzando la libreria *libpcap*[1] modificata per sfruttare PF_RING. La scelta di utilizzare questa libreria è dettata dalla volontà di utilizzare un API standard in modo da garantire la portabilità del software su qualsiasi sistema operativo. La libreria

[1] http://www.tcpdump.org/

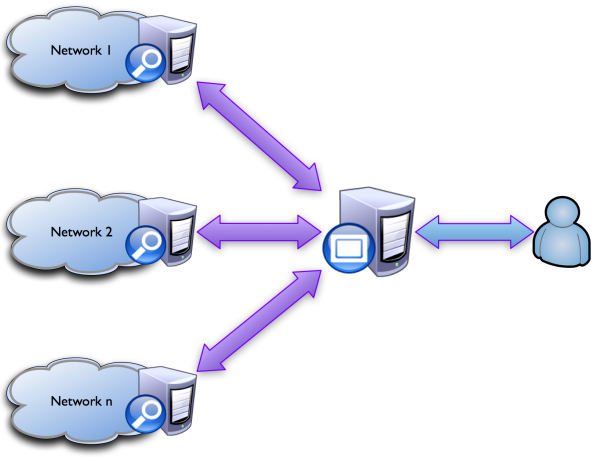


FIGURA 4.1: Nodi dell'architettura

libpcap permette inoltre la lettura di pacchetti precedentemente catturati dal disco rendendo possibile la verifica della correttezza dell'implementazione e delle prestazioni utilizzando dati proveniente da scenari reali.

Quando la sonda viene avviata mette la scheda di rete in modalità promiscua ed inizia la cattura dei pacchetti. La modalità promiscua fa in modo che la scheda di rete inoltri ai livelli superiori dello stack TCP/IP anche i frame che hanno come destinatario un indirizzo del layer 2 differente da quello della sonda. Dai pacchetti catturati viene estratta l'etichetta del flusso e calcolata la funzione hash per l'accesso al dizionario dei flussi. Nel listato 4.1 viene mostrata la definizione dell'etichetta.

```
typedef struct {
        struct in_addr src_ip;
        struct in_addr dst_ip;
        unsigned short src_port;
        unsigned short dst_port;
        u_char protocol;
        u_char padding[3];
} flowlabel_t;
```

LISTATO 4.1: Etichetta del flusso

Il dizionario dei flussi è implementato tramite lo schema Cuckoo Hashing (par. 2.2.3). La chiave di accesso al dizionario è l'etichetta del flusso mentre il valore associato è il riferimento all'header del flusso mostrato nel listato 4.2.

Per riuscire ad inserire i flussi in tempo costante usando Cuckoo Hashing il fattore di carico della tabella (implementata tramite due array) deve essere inferiore al 49% quindi più della metà della memoria occupata viene sprecata. Nella tabella hash vengono inseriti solamente i puntatori agli header del flusso in modo da limitare lo spreco di memoria.

La funzione hash è stata scelta in modo che fosse veloce da calcolare e che avesse una bassa percentuale di collisioni. La maggior parte dei software open source che fanno uso di dizionari utilizzano la funzione hash chiamata lookup3 [51]. Durante la selezione della funzione hash è stata testata la funzione MurmurHash2 [52] che permette di elaborare più del doppio dei dati rispetto a lookup3 e che ha una percentuale di collisioni paragonabile.

```
typedef struct fh {
    flowlabel_t label;
    unsigned int first_pkt_timestamp;
    unsigned int last_pkt_timestamp;
    unsigned long pkt_count;
    unsigned long flow_size;
    unsigned int tu_offset;
    u_char flow_state;
    filter_check_t filter_check;
    struct fh *prev;
    struct fh *next;
} flowheader_t;
```

Listato 4.2: Intestazione del flusso

Gli ultimi due campi della struttura flowheader_t vengono usati per mantenere la lista LRU utilizzata per la ricerca dei flussi scaduti. Il campo filter_check contiene

l'identificatore e lo stato dell'ultimo filtro applicato al flusso. Tramite il controllo di questo campo è possibile sapere se il flusso partecipa o meno alle statistiche. La struttura occupa un totale di 64 byte incluso il padding che permette di avere le strutture allineate in memoria.

All'avvio della sonda per ogni epoca viene allocata la memoria in grado di contenere un certo numero di microflussi fissato a priori. Nella struttura che rappresenta l'epoca (listato 4.3) viene memorizzato il timestamp associato, il filtro utilizzato per selezionare i microflussi, le statistiche e l'elenco dei microflussi.

Nel listato 4.4 viene mostrata la definizione di un microflusso. Così come per l'intestazione dei flussi anche qui c'è il campo filter_check che permette di sapere se il microflusso contribuisce alla statistiche. Questa ripetizione del valore è necessaria perché il filtro può cambiare da un'epoca all'altra e il valore memorizzato nell'intestazione del flusso si riferisce solamente all'ultimo filtro applicato.

```
typedef struct {
    unsigned int new_flows;
    unsigned int open_flows;
    unsigned int closed_flows;
    unsigned int timedout_flows;
    unsigned int pkt_count;
    unsigned int size;
} epoch_stats_t;

typedef struct {
    unsigned int timestamp;
    unsigned char filter_id;
    unsigned char padding[3];
    epoch_stats_t epoch_stats;
    topflow_msg_t top_pkts[TOP_SIZE];
    topflow_msg_t top_size[TOP_SIZE];
```

```
} stats_msg_t;


typedef struct {
    stats_msg_t stats_msg;
    filter_t filter;
    int insert_offset;
    microflow_t mf[MAX_FLOWS];
} epoch_t;
```

Listato 4.3: Definizione epoca

```
typedef struct {
    unsigned int pkt_count;
    unsigned int size;
    unsigned int prev_mf_offset;
    unsigned char prev_ts_offset;
    unsigned char flag_tcp;
    filter_check_t filter_check;
    flowheader_t *flowheader;
} microflow_t;
```

Listato 4.4: Definizione microflusso

Ogni microflusso occupa 24 byte incluso il padding. Fissando il numero di epoche a 31 in modo da avere le 30 epoche per effettuare le analisi più un epoca in cui inserire i nuovi microflussi e fissando il numero di microflussi massimo per epoca a 200 000 si ha che ogni epoca occupa 4,8 MB e il totale delle epoche occupa 148,8 MB di memoria RAM.

Per quanto riguarda la selezione del traffico il filtro è composto da un numero arbitrario di regole la cui definizione è mostrata nel listato 4.5.

```
typedef struct {
```

```
    struct in_addr src_ip;
    struct in_addr src_mask;
    struct in_addr dst_ip;
    struct in_addr dst_mask;
    u_int16_t src_port_min;
    u_int16_t src_port_max;
    u_int16_t dst_port_min;
    u_int16_t dst_port_max;
    u_int8_t protocol;
#define ACL_PROTOCOL_ANY   0
#define ACL_PROTOCOL_TCP   IPPROTO_TCP
#define ACL_PROTOCOL_UDP   IPPROTO_UDP
    u_char flow_state;
    u_char action;
#define ACL_ACTION_DENY    0
#define ACL_ACTION_PERMIT  1
    u_char padding;
} acl_entry_t;
```

LISTATO 4.5: Definizione regola ACL

4.1.2 Collettore

Il collettore è formato da due parti. La prima parte comunica con le sonde scambiando messaggi in formato binario. La seconda parte comunica con un server Comet che permette l'interfacciamento con le console. I messaggi inviati e ricevuti dal server Comet contengono stringhe in formato JSON.

Per le connessioni in ingresso al collettore da parte delle sonde sono stati utilizzati socket TCP asincroni in modo da permettere la comunicazione con centinaia di sonde con un singolo processo. Per gestire il grande numero di socket è stata utilizzata la

libreria *libevent*[2] la quale fornisce la possibilità di impostare delle callback che vengono richiamate ogni volta che si verifica un evento sul file descriptor del socket.

Man mano che le statistiche vengono ricevute dalle sonde si aggiorna la struttura dati che contiene le statistiche per l'epoca (listato 4.6). I campi interi vengono riempiti sommando quelli analoghi nel messaggio ricevuto dalle sonde. Nei campi di tipo topflow_msg_t invece vengono inseriti i 10 flussi con "Top Value" maggiore e mantenuti ordinati secondo questo valore. All'arrivo di ogni risultato dalle sonde viene fatto un *insertion sort* eliminando i flussi in eccesso.

```
typedef struct {
    unsigned int current_timestamp;
    unsigned short active_probes_id[65536];
    unsigned short active_probes_count;
    unsigned int new_flows;
    unsigned int open_flows;
    unsigned int closed_flows;
    unsigned int timedout_flows;
    unsigned int pkt_count;
    unsigned int size;
    topflow_msg_t top_pkts[TOP_SIZE];
    topflow_msg_t top_size[TOP_SIZE];
} collector_stats_t;
```

LISTATO 4.6: Statistiche sul collettore

Quando il server Comet riceve un nuovo filtro da parte di una console assegna un ID alla richiesta e replica il messaggio JSON a tutte le console connesse in modo che queste sappiano qual'è il filtro applicato. Il filtro ricevuto viene inoltre convertito dal formato JSON al formato binario utilizzato dalle sonde e inviato a tutte le sonde o i collettori connessi.

[2] http://www.monkey.org/~provos/libevent/

FIGURA 4.2: Schema del sistema web

Come server Comet è stato utilizzato APE[3] (Ajax Push Engine), un sistema open source per effettuare il push dei dati verso il browser. Utilizzando solo sistemi standard non richiede l'utilizzo di plugin sul browser permettendone l'utilizzo su qualsiasi sistema operativo. Il progetto APE consiste in un server HTTP ottimizzato per lo streaming dei dati e in un framework chiamato APE JavaScript Framework (AJF) che permette l'invio e la ricezione dei messaggi sul client. Il server APE può essere utilizzato solamente per il push dei dati. Per fornire gli altri contenuti statici è necessario l'utilizzo di un server web separato secondo lo schema di figura 4.2. Il server web scelto è Mongoose[4] in quanto cross-platform e di dimensioni minime.

Il server APE può essere esteso tramite Server-Side Javascript (SSJS) in modo da implementare logiche personalizzate. Per permettere il push dei dati verso il browser è stato realizzato un modulo che si mette in ascolto su una porta TCP e riceve dal collettore i messaggi JSON che verranno inoltrati senza modifiche ai browser connessi. Quando il modulo riceve una richiesta da parte della console questa viene inviata al collettore ma anche mantenuta in memoria in modo da essere inviata alle eventuali nuove console che si collegano.

4.1.3 Console

La console fa uso esclusivo di un browser web senza l'utilizzo di plug-in per la visualizzazione dei dati o la comunicazione con il collettore. Tutto il codice eseguito sul client è scritto in JavaScript utilizzando la libreria jQuery[5] e il framework AJF.

[3] http://www.ape-project.org/
[4] http://code.google.com/p/mongoose/
[5] http://jquery.com/

FIGURA 4.3: APE Javascript Framework

Le pagine web che mostrano le statistiche caricano il framework AJF utilizzato per ricevere i risultati e inviare i comandi al server APE (figura 4.3). Ogni volta che il server APE invia un messaggio al browser su quest'ultimo viene eseguito l'handler onRaw() che provvede ad aggiornare le statistiche o il filtro a seconda del messaggio ricevuto. Per l'invio dei comandi dal browser al server APE si utilizza invece il metodo sendCmd(). Per disegnare i grafici è stata usata la libreria Flot[6] per jQuery. Questa libreria permette di tracciare grafici in tempo reale sul client e di interagirci con operazioni di zoom o traslazione.

In figura 4.4 viene mostrata la finestra del browser durante la visualizzazione dei grafici che mostrano l'andamento nel tempo dei byte trasferiti e del numero di pacchetti trasferiti. Il grafico, che nel momento dell'acquisizione visualizzava un intervallo temporale di 2 minuti, permette di apprezzare brevi picchi del traffico che non verrebbero rilevati da altri strumenti di monitoraggio. Il sistema oltre a permettere la visione generale dell'andamento del traffico permette di scendere nel dettaglio delle connessioni. Nella figura 4.5 vengono mostrati i flussi che hanno generato più traffico verso la porta TCP 80. Questi criteri possono essere modificati dinamicamente clickando sul tasto "Filter" e selezionando nel popup che appare la tipologia di traffico cui si è interessati.

4.2 Valutazione delle prestazioni

Nella valutazione delle prestazioni ci si è concentrati sulla generazione delle statistiche in tempo reale. Per effettuare le misure replicando traffico reale sono stati usati dei

[6] http://code.google.com/p/flot/

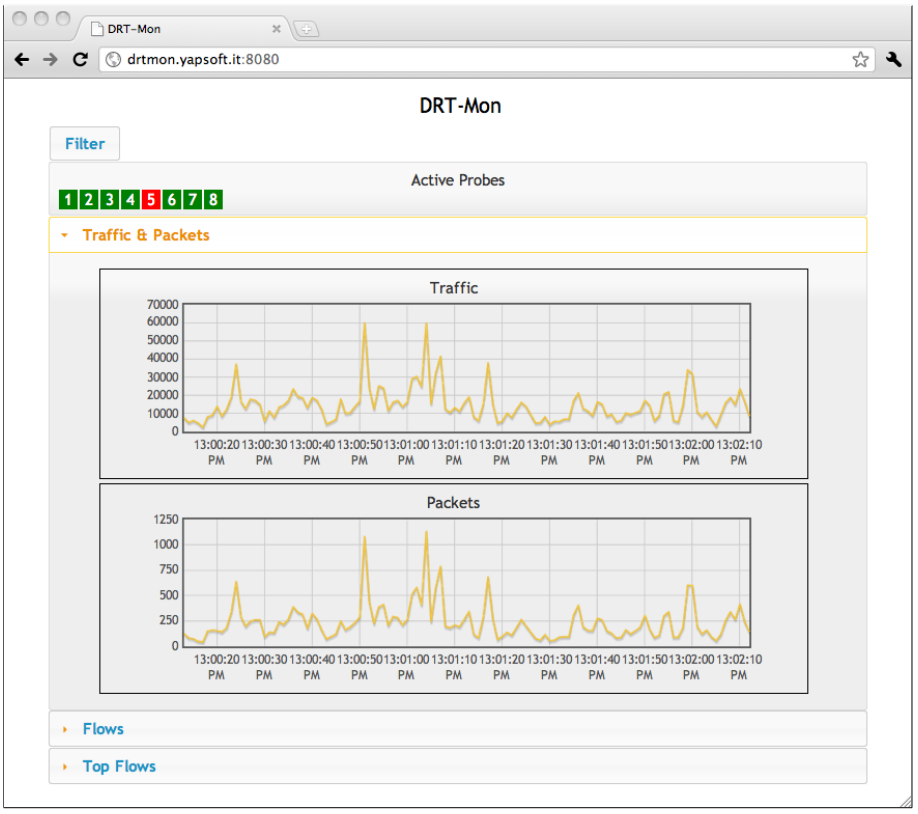


FIGURA 4.4: Screenshot console di amministrazione: Traffico

	Numero di pacchetti	**Dati trasferiti**	**Flussi attivi**
Media (1 s)	1,21 M	950 MB	234 K
Totale (60 s)	72,4 M	57,1 GB	5,77 M

TABELLA 4.1: Statistiche dataset

dataset forniti dall'associazione CAIDA[7]. Essi sono stati creati nel 2009 installando dei sistemi di monitoraggio presso il datacenter di Equinix[8] a Chicago e acquisendo i pacchetti su un collegamento OC192 tra Chicago e Seattle. Questi sono stati anonimizzati modificando gli indirizzi IP tramite il software Crypto-PAn e rimuovendo il payload.

Vista la grande dimensione dei dataset si è selezionato un intervallo temporale, della durata di un minuto, in cui c'è un picco di traffico. Le statistiche interessanti della traccia sono riportate in tabella 4.1.

[7]The CAIDA Anonymized 2009 Internet Traces - 21/5/2009 - Colby Walsworth, Emile Aben, kc claffy, Dan Andersen - http://www.caida.org/data/passive/passive_2009_dataset.xml

[8]http://www.equinix.com/

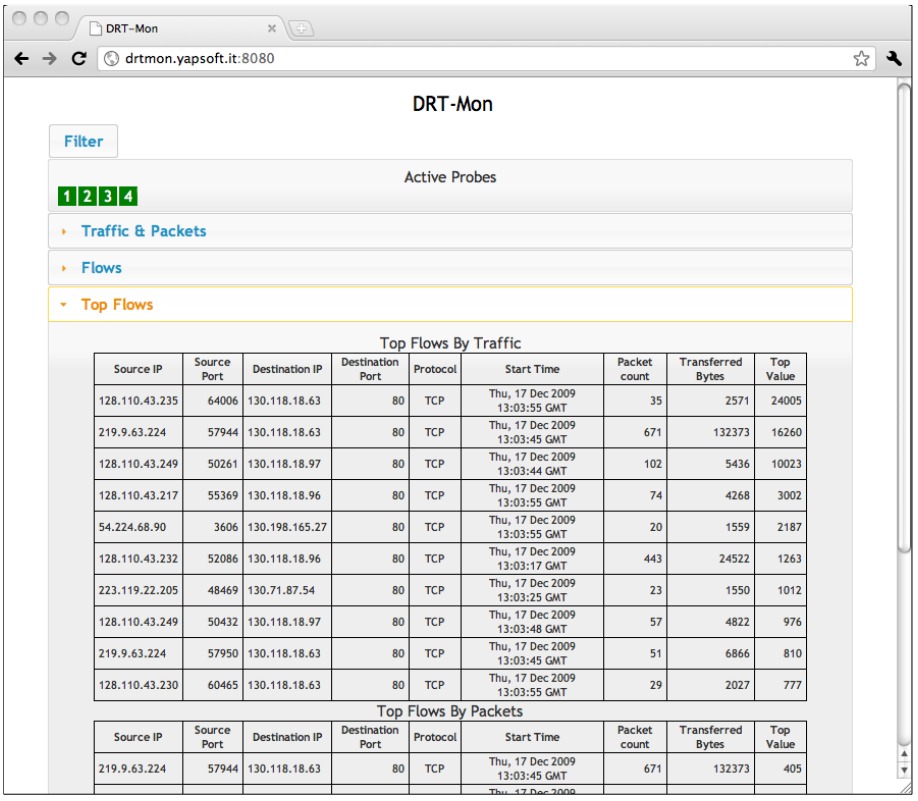


FIGURA 4.5: Screenshot console di amministrazione: Top-k flussi

4.2.1 Sonda

In una prima fase è stato valutato il tempo di servizio di una singola sonda leggendo la traccia precedentemente caricata in un ramdisk. Il ramdisk è stato utilizzato perché nel caso di lettura della traccia dal disco questo diventa un collo di bottiglia. In uno scenario reale con l'uso di PF_RING i pacchetti vengono copiati dalla scheda di rete direttamente in memoria rendendo le prestazioni più vicine a quelle di una lettura da ramdisk rispetto a quelle di una lettura da disco. Il PC utilizzato per effettuare questo test aveva le seguenti caratteristiche:

- Processore: Intel Core i7 930 (2,8 GHz)
- RAM: 8 GB DDR3
- Sistema operativo: Debian GNU/Linux 6.0 - Kernel 2.6.32-5-amd64
- Hard disk: 2 x 750 GB 7500 RPM SATA II (software RAID 1)

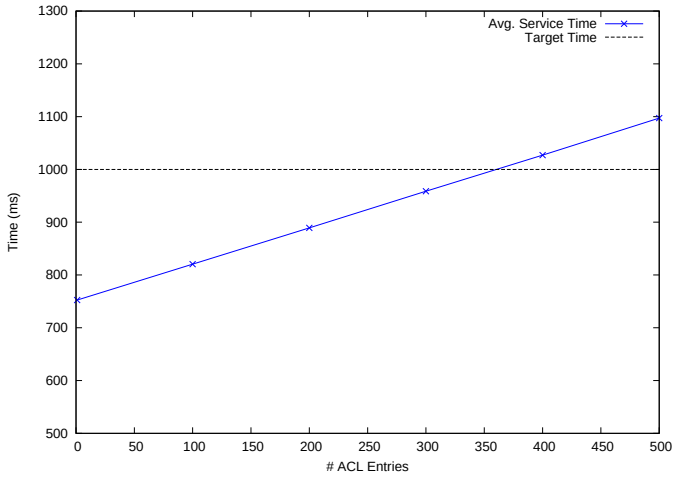


FIGURA 4.6: Tempo di servizio della sonda

Byte trasferiti	**Traccia pcap (solo header)**	**Dimensione archivio DRT-Mon**	**Riduzione spazio**
56,1 GB	4,02 GB	441 MB	89 %

TABELLA 4.2: Occupazione su disco dell'archivio storico

Per l'elaborazione dei pacchetti è stato utilizzato un unico core dei quattro disponibili. Nella figura 4.6 viene riportato il grafico del tempo di servizio della sonda al variare del numero di regole dell'ACL analizzate prima di selezionare i pacchetti. La linea tratteggiata rappresenta il tempo di servizio massimo oltre il quale la sonda diventa un collo di bottiglia. Come si vede, con 1,21 milioni di pacchetti al secondo in ingresso, la sonda può applicare fino a 350 regole rispettando i requisiti sul tempo di servizio.

L'occupazione su disco dell'archivio storico è calcolata sommando lo spazio occupato dalle epoche compresse e dagli header dei flussi. Nella tabella 4.2 viene riportato il numero totale di byte trasferiti, la dimensione della traccia contenente i soli header, la dimensione dei dati memorizzati nell'archivio di DRT-Mon e la percentuale di riduzione dello spazio occupato dalla soluzione DRT-Mon rispetto alla traccia pcap.

Lo spazio occupato su disco dalla traccia pcap dipende esclusivamente dal numero di pacchetti catturati mentre nel caso di DRT-Mon lo spazio occupato dipende dal numero di flussi attivi in un secondo. Nel caso di tracce pcap l'occupazione di memoria, fissato il numero di pacchetti monitorati, è quindi costante mentre nel caso di DRT-Mon si può ottenere una maggiore o minore occupazione in base al numero dei pacchetti monitorati che appartengono alla stessa sessione.

La scrittura di 441 MB al minuto (in media 7,35 MB/s) è sostenibile da qualsiasi hard disk senza dover ricorrere a costosi sistemi RAID. Inoltre visto il basso costo per MB degli hard disk si possono mantenere archivi settimanali o mensili con costi molto bassi.

La valutazione delle prestazioni catturando il traffico direttamente dalla rete è stata limitata da due fattori:

- Rete ad 1 Gbps. Non avendo a disposizione una rete a 10 Gbps non è stato possibile valutare le effettive prestazioni che avrebbe il sistema su questo tipo di reti.
- Impossibilità di sfruttare la CPU multi-core. L'hardware a disposizione non è in grado di sfruttare TNAPI per bilanciare il traffico su più core.

L'ambiente di test era composto da due PC. Il primo è stato usato per l'invio della traccia pcap utilizzando il software tcpreplay[9] e il secondo è stato usato come sonda. I due PC sono stati collegati tramite un cavo ethernet cat. 5e incrociato in modo da evitare degradi delle prestazioni che potevano essere introdotti da uno switch o da altri dispositivi connessi alla rete. Il PC utilizzato per l'invio dei pacchetti aveva le seguenti caratteristiche:

- Processore: Intel Core 2 Duo (2 GHz)
- RAM: 4 GB DDR3
- Sistema operativo: Mac OS X (Versione 10.6.6)

[9] http://tcpreplay.synfin.net/

Pkt/s	**Bit/s**	**Flussi attivi**
314,9 K	2 G	97,6 M

Tabella 4.3: Prestazioni sonda con pacchetti senza payload

- Hard disk: 500 GB 7200 RPM SATA II
- Scheda di rete: Gigabit Ethernet

Il PC utilizzato per la cattura dei pacchetti aveva le seguenti caratteristiche:

- Processore: Intel Core 2 6600 (2,4 GHz)
- RAM: 4 GB DDR2
- Sistema operativo: Gentoo Linux - 2.6.31-gentoo-r10 (amd64)
- PF_RING: 4.6.0 Transparent Mode 2
- Hard disk: 250 GB 7200 RPM SATA II
- Scheda di rete: Intel 82540EM Gigabit Ethernet (PCI 66 MHz 32-bit)

Le prestazioni sono state misurate fissando il numero di regole per la selezione del traffico a 25 e il numero totale dei pacchetti inviati a 5 M. La velocità massima raggiunta durante l'invio dei pacchetti senza l'aggiunta del payload (non contenuto nelle tracce pcap) è di 315 Kpps. In tabella 4.3 sono riportate le prestazioni ottenute.

Come si vede in questo caso la sonda è stata in grado di analizzare tutto il traffico perdendo una quantità trascurabile di pacchetti. La seconda colonna della tabella riporta la velocità teorica ottenuta estraendo la lunghezza dei pacchetti dall'header IP che include il payload.

Nella tabella 4.4 sono riportati i risultati ottenuti utilizzando un'opzione del software tcpreplay che permette di aggiungere un payload fittizio ai pacchetti da inviare. La sonda è riuscita a catturare i pacchetti in transito su una rete carica all'83 %.

Pkt/s	**Bit/s**	**Flussi attivi**
130,9 K	831,4 M	54,5 M

TABELLA 4.4: Prestazioni sonda con pacchetti contenenti un payload fittizio

4.2.2 Collettore

Le prestazioni del collettore sono state valutate creando un piccolo software che prende come parametro il numero di connessioni da aprire verso il collettore e invia dei messaggi contenenti statistiche precedentemente preparate. Dai test effettuati simulando l'invio delle statistiche da parte di 1000 sonde, eseguendo il collettore sullo stesso PC con cui sono state valutate le prestazioni della sonda, il tempo di servizio medio è di 50 ms. Se dal punto di vista delle prestazioni il collettore non crea molti problemi, il dato da analizzare è la quantità di banda necessaria per ricevere i dati dalle sonde. Nel caso di 1000 sonde, ogni secondo vengono trasferiti 840 KB. Per ridurre la capacità del link in ingresso al collettore si può adottare la soluzione proposta di avere più collettori che effettuano aggregazioni parziali dei dati.

4.3 Confronto con lo stato dell'arte

In questo paragrafo vengono analizzati gli obiettivi definiti nel paragrafo 1.7 verificando se e come l'architettura DRT-Mon li rispetta. Inoltre vengono messe in risalto le novità introdotte nella tesi rispetto alle soluzioni esistenti.

Architettura distribuita

L'architettura DRT-Mon ha il vantaggio, rispetto alle altre soluzioni distribuite analizzate nel capitolo 2, di non dover spostare tutti i dati verso un punto centrale. Le architetture attuali tendono infatti ad esportare i dati dalle sonde alla console di amministrazione creando un sovraccarico della rete e del nodo centrale di elaborazione. In DRT-Mon l'elaborazione avviene invece su sonde indipendenti che inviano alla console di amministrazione solamente i dati necessari a rispondere

alle interrogazioni. Questo si traduce in bassi requisiti per la rete di monitoraggio e alta tolleranza ai guasti.

Granularità delle analisi

Avere alta granularità significa dover gestire ed eventualmente trasferire una maggiore quantità di dati. Le soluzioni che aggregano i pacchetti in flussi permettono di limitare la quantità di dati esportati rispetto alle soluzioni che effettuano le analisi sui pacchetti ma hanno una granularità troppo bassa. Una novità introdotta in DRT-Mon è il concetto di microflusso. Grazie ai microflussi è possibile avere un compromesso tra l'alta granularità e l'occupazione di memoria rendendo possibili, che come visto nel paragrafo 4.2, le analisi in near-realtime di reti ad alta velocità con la risoluzione temporale di 1s.

Presentazione dei risultati near-realtime

L'alta granularità delle analisi è utile soprattutto se queste possono essere effettuate in tempo reale ossia se la visualizzazione dei risultati avviene pochi istanti dopo il verificarsi degli eventi sulla rete. Nella letteratura legata al campo del monitoraggio di rete spesso il termine realtime viene utilizzato con il significato di monitoraggio online ossia continuo. In questa classe rientrerebbero quasi tutte le soluzioni analizzate nel capitolo 2 che presentano i risultati solamente al termine delle sessioni monitorate. Nella tesi si è voluti andare oltre il monitoraggio online e arrivare a presentare i risultati pochi decimi di secondo dopo il verificarsi degli eventi. I test hanno dimostrato che è stato possibile monitorare un link a 1 Gbps con una singola sonda rispettando questi requisiti.

Impatto sul traffico

Per evitare di modificare il comportamento del traffico di rete le sonde ricevono una replica del traffico effettuando monitoraggio passivo. Nel caso in cui non si disponesse di una rete dedicata per il monitoraggio ma i risultati delle sonde dovessero essere esportati tramite la rete che si sta monitorando, la dimensione contenuta dei messaggi scambiati permette di avere comunque un impatto minimo

sul traffico misurato. Tutte le soluzioni prese in considerazione, che permettono di avere più sonde di monitoraggio, fanno uso di collegamenti dedicati per il trasporto dei risultati. Queste soluzioni, nel caso di reti geografiche, limitano la scalabilità del sistema di monitoraggio richiedendo ingenti investimenti per l'installazione di una rete di monitoraggio ad alte prestazioni o andando a limitare le prestazioni reali della rete nel caso di utilizzo della rete monitorata per il trasporto dei dati.

Selezione dei dati

Tra le architetture analizzate, le uniche in grado di effettuare monitoraggio online e di variare dinamicamente i criteri di selezione e aggregazione del traffico a tempo di esecuzione sono Gigascope e DOME. Nell'architettura DRT-Mon è stato ottenuto un risultato simile utilizzando un sistema ad ACL mediante il quale l'amministratore può scegliere se è interessato al monitoraggio continuo o a un'interrogazione dell'archivio storico mediante la selezione dei flussi in base all'IP, la porta, il protocollo di trasporto o lo stato del flusso. La variazione dinamica dell'ACL impostata permette il drill-down delle analisi da una visione globale fino alla focalizzazione su una singola sessione.

Consultazione dei dati storici

A differenza della altre soluzioni che memorizzano su disco gli header dei pacchetti o i flussi, DRT-Mon memorizza su disco i microflussi. Questo permette di effettuare analisi sullo storico con una granularità pari a quella del monitoraggio near-realtime variando l'ACL utilizzata per la selezione dei dati. Per l'indicizzazione dei microflussi viene utilizzato Fastbit che grazie all'uso di bitmap index compressi permette di avere operazioni di indicizzazione e ricerca molto veloci.

Scalabilità

L'utilizzo di uno schema distribuito permette la scalabilità del sistema. Il traffico non deve essere analizzato su una singola sonda ma può essere distribuito tra più sonde rendendo possibile il monitoraggio di reti ad alta velocità. Inoltre l'aggiunta di una nuova sonda non richiede una riconfigurazione del sistema. La sonda

una volta attivata si registra presso il collettore e contribuisce alla generazione delle statistiche. Come visto nel paragrafo 4.2.1 una singola sonda è in grado di analizzare il traffico di una rete gigabit rendendo il sistema adatto a monitorare reti complesse e ad alta velocità. L'uso di più collettori collegati mediante una topologia ad albero permette di effettuare aggregazioni parziali dei risultati riducendo il traffico sulla rete. Sfruttando i processori multi-core è possibile eseguire più istanze della sonda su uno stesso PC ed esportare i risultati già aggregati verso il collettore radice.

Tolleranza ai guasti

Il guasto di un nodo del sistema provoca ripercussioni differenti a seconda del livello dell'albero in cui avviene. Se il collettore centrale smette di funzionare non è più possibile visualizzare i risultati del monitoraggio ma questi vengono comunque tenuti in memoria dalle sonde e possono essere consultati quando il guasto viene ripristinato. Se invece il guasto si verifica su un collettore intermedio l'interruzione della ricezione delle statistiche avviene solo per le sonde collegate a quel collettore. L'unico guasto che provoca una perdita delle informazioni è quello della sonda ma comunque si perdono solo i dati relativi a una parte della rete. Per permettere un tempestivo intervento per la risoluzione dei guasti questi sono segnalati agli amministratori di rete tramite la console di monitoraggio.

Privacy

Estraendo solamente l'header dei pacchetti la privacy degli utenti è rispettata. Nel caso in cui si abbiano requisiti stringenti che impongano l'oscuramento anche delle informazioni contenute nell'header si possono usare tecniche per l'anonimizzazione che preservino il prefisso degli indirizzi in modo da rendere anonimi i dati ma avere comunque informazioni sulla provenienza del traffico. Utilizzando TLS si evita di far viaggiare in chiaro le informazioni e si autenticano tra loro i nodi dell'architettura.

Commodity hardware

Uno dei vantaggi nell'uso del commodity hardware è il fatto di non essere legati a un produttore hardware che impone le sue tecnologie e i suoi prezzi. Ad esempio, l'architettura DOME è stata realizzata utilizzando Network Processor Intel la cui produzione è stata interrotta. Questo rende impossibile la riparazione delle sonde di rete o nuove installazioni del sistema senza effettuare lavori di adattamento del software al nuovo hardware. DRT-Mon ha dimostrato di poter raggiungere ottime prestazioni utilizzando commodity hardware di fascia bassa che oltre a non porre vincoli sul tipo di hardware utilizzato ha anche un costo molto basso.

Visualizzazione dei risultati

Nessuna delle soluzioni analizzate nel capitolo 2 affronta il problema della presentazione dei risultati. In DRT-Mon si è scelto di usare tecnologie web, e in particolare il modello Comet, in modo da permettere la visualizzazione dei risultati in modo indipendente dal dispositivo o dal sistema operativo utilizzato. Grazie a queste tecnologie gli amministratori di rete hanno infatti la possibilità di consultare i risultati anche tramite uno smartphone quando non si trovano nel loro ufficio o non hanno a disposizione un PC.

Capitolo 5

Conclusioni

Vista l'evoluzione delle attuali reti di computer i sistemi per il loro monitoraggio devono evolversi di conseguenza. La complessità delle reti, la loro velocità e la tipologia di traffico che trasportano rende necessari sistemi in grado di monitorare la rete in diversi punti e con un'alta granularità. Dall'analisi delle soluzioni esistenti è emerso che senza l'ausilio di hardware dedicato non si riesce ad avere un monitoraggio in tempo reale e con un dettaglio sufficiente. Le soluzioni che usano commodity hardware hanno la necessità di aggregare i pacchetti in flussi impedendo di avere un'alta granularità delle analisi. Le stesse considerazioni valgono per il mantenimento dei dati in un archivio che possa essere consultato in un secondo momento. La velocità con cui arrivano i pacchetti o gli header rende necessario l'uso di costosi dispositivi di memorizzazione. Dalla volontà di migliorare i sistemi attuali offrendo la possibilità di monitorare un gran numero di sottoreti con alta granularità, di presentare i risultati in tempo reale e con una buona scalabilità utilizzando commodity hardware è nata l'architettura DRT-Mon.

In DRT-Mon è stato introdotto il concetto di microflusso che permette di avere un compromesso tra la necessità di aggregare i pacchetti per renderne possibile l'elaborazione e la granularità necessaria a rilevare anomalie di breve durata. La riduzione della quantità di dati ottenuta con i microflussi permette di avere archivi storici di dimensioni ridotte permettendo la memorizzazione di lunghi periodi di tempo all'interno delle sonde. Rispetto ai flussi, i microflussi fanno aumentare la dimensione dei dati acquisiti

Distribuita	✓
Dati trasferiti	Risultati aggregati
Monitoraggio online	✓
Real-time	Near-realtime
Archivio storico	✓ (distribuito)
Granularità	1 s

Selezione dei dati	ACL
Privacy	Solo header
Requisiti Hardware	Commodity hardware
Console	Web 2.0 + Comet
Unità di monitoraggio	Microflusso

Tabella 5.1: Caratteristiche dell'architettura DRT-Mon

e per questo motivo non è possibile esportarli come si fa per i flussi. L'elaborazione avviene quindi sulle sonde in modo da trasferire solamente i dati necessari alla visualizzazione dei risultati. L'uso dell'architettura distribuita permette la scalabilità della soluzione e il dimensionamento dell'hardware delle sonde di rete in base alla sottorete in cui devono essere posizionate.

L'architettura è composta da tre tipi di nodi: sonda di rete, collettore e console di monitoraggio. Questi sono stati realizzati utilizzando differenti tecnologie allo stato dell'arte. Le sonde fanno uso di PF_RING per ridurre l'overhead introdotto dal kernel nell'acquisizione dei pacchetti, di strutture dati che hanno tempo di accesso costante per riuscire ad elaborare il traffico in tempo reale e di sistemi di indicizzazione basati su bitmap index compressi per l'accesso all'archivio storico. Il collettore sfrutta socket asincroni ed è stato realizzato in modo da poter gestire centinaia di sonde senza introdurre ritardi considerevoli. La console di monitoraggio è stata realizzata utilizzando esclusivamente tecnologie Web. I dati vengono inviati al browser sfruttando il modello Comet e presentati all'utente mediante grafici e tabelle generati dal browser stesso tramite JavaScript. Nella tabella 5.1 vengono riportate le principali caratteristiche dell'architettura DRT-Mon.

Il processo di valutazione delle prestazioni è stato limitato dalla non disponibilità di hardware adatto a sfruttare a pieno PF_RING ma ha comunque dimostrato che è possibile monitorare con una singola sonda reti a 1 Gbps. Inoltre è stato verificato che il sistema è in grado di gestire centinaia di sonde senza problemi. Grazie a questa

caratteristica, nel caso in cui una sonda non sia sufficiente a monitorare un singolo link, si può distribuire il traffico su più sonde e far aggregare i risultati al collettore.

5.1 Lavoro futuro

La fase di validazione ha dimostrato che l'architettura è in grado di scalare con l'aumentare del numero di sottoreti e della velocità dei collegamenti e di fornire informazioni sullo stato della rete in near-realtime. Sono comunque stati identificati alcuni aspetti che possono essere migliorati ed estesi.

Aggiunta di nuove metriche

Nella tesi sono state prese in considerazione alcune metriche come il numero di pacchetti, il traffico generato e i microflussi attivi. Queste possono essere integrate con nuove metriche in base alle esigenze delle diverse tipologie di rete.

Soglie di allarme

L'aggiunta di sistemi per la definizione di soglie di allarme che una volta superate inviano delle notifiche alla console di amministrazione o cambiano automaticamente i criteri di selezione del traffico per puntare l'attenzione su situazioni anomale è un'altra area importante su cui lavorare.

Estensione della definizione di flusso

Diversi protocolli di rete utilizzano più connessioni per svolgere i loro compiti. Può essere utile estendere la definizione di flusso in modo da tenere conto di questa correlazione ed effettuare analisi più precise.

Interrogazioni multiple

L'architettura così com'è stata definita permette l'impostazione di un solo criterio di selezione dei dati. In alcuni casi può essere necessario visualizzare contemporaneamente i risultati ottenuti con criteri di selezione differente. L'estensione dell'architettura in modo da permettere questo tipo di interrogazioni è un'altra delle migliorie che si possono apportare all'architettura proposta.

Elenco delle figure

1.1 Cisco's Global Consumer Internet Traffic Forecast 2

2.1 Architettura IPMON 15

2.2 Architettura Nprobe 18

2.3 Striping dei pacchetti 18

2.4 Architettura pipeline di NG-MON 21

2.5 Formato dei dati inviati ai generatori di flusso 22

2.6 Formato dei dati di un flusso 22

2.7 Distribuzione del carico nella fase di memorizzazione dei flussi 23

2.8 Schema generale architettura DOME 26

2.9 Architettura di un nodo DOME 26

2.10 Throughput porta di monitoraggio 28

2.11 Schema del prototipo facente uso di TelegraphCQ e FastBit 30

2.12 Velocità di creazione dell'indice bitmap in funziona della dimensione dei blocchi 31

2.13 Threaded NAPI 36

2.14 Effetti della compressione sul tempo di risposta alle query 40

2.15 Tempo di risposta medio al variare del bitmap encoding 40

2.16 Mean Publish Trip-time 44

2.17 Mean Received Publish Messages 45

2.18 Mean Received Unique Publish Messages 46

3.1 Architettura del sistema di monitoraggio 48
3.2 Campi che identificano un flusso 51
3.3 Schema della sonda 53
3.4 Lista delle epoche 56
3.5 Diagramma delle attività del primo stadio della sonda 60
3.6 Diagramma delle attività del secondo stadio della sonda 61
3.7 Schema collettore ad albero 67
3.8 Caso d'uso del sistema 68
3.9 Header dei messaggi 69
3.10 Registrazione Sonda 70
3.11 Disconnessione Sonda 70
3.12 Interruzione invio statistiche 70
3.13 Statistiche Sonda-Collettore 72
3.14 ACL 73

4.1 Nodi dell'architettura 79
4.2 Schema del sistema web 85
4.3 APE Javascript Framework 86
4.4 Screenshot console di amministrazione: Traffico 87
4.5 Screenshot console di amministrazione: Top-k flussi 88
4.6 Tempo di servizio della sonda 89

Elenco delle tabelle

1.1 Requisiti di QoS per le principali applicazioni 3

2.1 NG-MON: Simboli e valori 24
2.2 Caratteristiche delle architetture analizzate (parte 1) 31
2.3 Caratteristiche delle architetture analizzate (parte 2) 32
2.4 Prestazioni di PF_RING 35

4.1 Statistiche dataset 87
4.2 Occupazione su disco dell'archivio storico 89
4.3 Prestazioni sonda con pacchetti senza payload 91
4.4 Prestazioni sonda con pacchetti contenenti un payload fittizio 92

5.1 Caratteristiche dell'architettura DRT-Mon 98

Elenco dei listati

4.1 Etichetta del flusso 79
4.2 Intestazione del flusso 80
4.3 Definizione epoca 81
4.4 Definizione microflusso 82
4.5 Definizione regola ACL 82
4.6 Statistiche sul collettore 84

Acronimi

ACL	**A**ccess **C**ontrol **L**ist
AJAX	**A**synchronous **J**ava**S**cript and **X**ML
DDoS	**D**istributed **D**enial of **S**ervice
DoS	**D**enial **o**f **S**ervice
DSMS	**D**ata **S**tream **M**anagement **S**ystem
HTML	**H**yper**T**ext **M**arkup **L**anguage
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
IP	**I**nternet **P**rotocol
IPTV	**I**nternet **P**rotocol **T**elevision
ISP	**I**nternet **S**ervice **P**rovider
JSON	**J**ava**S**cript **O**bject **N**otation
LRU	**L**east **R**ecently **U**sed
QoS	**Q**uality **o**f **S**ervice
REST	**R**epresentational **S**tate **T**ransfer
RRD	**R**ound **R**obin **D**atabase
SLA	**S**ervice **L**evel **A**greement
SPF	**S**ingle **P**oint of **F**ailure
SQL	**S**tructured **Q**uery **L**anguage
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
VoIP	**V**oice **o**ver **I**nternet **P**rotocol

Bibliografia

[1] K. Coffman and A. Odlyzko, "Internet growth: Is there a "moore's law" for data traffic," *Handbook of massive data sets*, Jan 2001.

[2] I. Cisco Systems, "Cisco visual networking index: Forecast and methodology, 2008-2013," pp. 1–13, Dec 2009.

[3] I. Cisco Systems, "Hyperconnectivity and the approaching zettabyte era," pp. 1–16, Dec 2009.

[4] A. Dutta-Roy, "The cost of quality in internet-style networks," *Spectrum, IEEE*, vol. 37, pp. 57 – 62, Sep 2000.

[5] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, Oct 2007.

[6] L. Pantel and L. Wolf, "On the impact of delay on real-time multiplayer games," *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*, Jan 2002.

[7] B. Goode, "Voice over internet protocol (voip)," *Proceedings of the IEEE*, vol. 90, pp. 1495 – 1517, Sep 2002.

[8] M. Tadault, S. Soormally, and L. Thiebaut, "Network evolution towards ip multimedia subsystem," *Alcatel Telecommunications Review*, Jan 2003.

[9] "TABB Group [Online]." http://www.tabbgroup.com/.

[10] C. Molina-Jimenez, S. Shrivastava, and J. Crowcroft, "On the monitoring of contractual service level agreements," *Proceedings First International Workshop on Electronic Contracting*, Jan 2004.

[11] M. Blumenthal and D. Clark, "Rethinking the design of the internet: the end-to-end arguments vs. the brave new world," *Transactions on Internet Technology*, vol. 1, Aug 2001.

[12] M. Grossglauser and J. Rexford, "Passive traffic measurement for ip operations," in *The Internet as a Large-Scale Complex System*, pp. 91–120, Oxford University Press, 2005.

[13] K. Papagiannaki, N. Taft, Z.-L. Zhang, and C. Diot, "Long-term forecasting of internet backbone traffic," *Neural Networks, IEEE Transactions on*, vol. 16, pp. 1110 – 1124, Sep 2005.

[14] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, "Rfc3272: Overview and principles of internet traffic engineering," *RFC Editor United States*, 2002.

[15] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Aug 2003.

[16] G. Iannaccone, C. Diot, I. Graham, and N. McKeown, "Monitoring very high speed links," *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, Nov 2001.

[17] "sFlow.org [Online]." http://www.sflow.org/sFlowOverview.pdf.

[18] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?," *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, Oct 2006.

[19] "Introduction to cisco ios® netflow," *Cisco Systems, Inc.*, 2007.

[20] C. Fraleigh, F. Tobagi, C. Diot, B. Lyles, and S. Moon, "Design and deployment of a passive monitoring infrastructure," *Proceedings of the Thyrrhenian International Workshop on Digital Communications: Evolutionary Trends of the Internet*, pp. 556–575, Jan 2001.

[21] A. Moore, J. Hall, C. Kreibich, E. Harris, and I. Pratt, "Architecture of a network monitor," *Proceedings of the Passive & Active Measurement Workshop 2003*, Jan 2003.

[22] S. Han, M. Kim, H. Ju, and J. Hong, "The architecture of ng-mon: a passive network monitoring system for high-speed ip networks," *Lecture Notes in Computer Science*, Jan 2002.

[23] T. Oetiker and D. Rand, "Mrtg–the multi router traffic grapher," *USENIX LISA*, Jan 1998.

[24] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope: a stream database for network applications," *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, Jun 2003.

[25] T. Wolf, R. Ramaswamy, S. Bunga, and N. Yang, "An architecture for distributed real-time passive network measurement," *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006*, Jan 2006.

[26] F. Reiss, K. Stockinger, K. Wu, A. Shoshani, and J. M. Hellerstein, "Enabling real-time querying of live and historical stream data," *Proceedings of the 19th International Conference on Scientific and Statistical Database Management*, Jan 2007.

[27] N. Suri, M. Hugue, and C. Walter, "Synchronization issues in real-time systems," *Proceedings of the IEEE*, Jan 1994.

[28] D. Mills, "Internet time synchronization: The network time protocol," *Communications*, Jan 1991.

[29] L. Degioanni and G. Varenni, "Introducing scalability in network measurement: toward 10 gbps with commodity hardware," *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, Oct 2004.

[30] L. Rizzo, "Device Polling support for FreeBSD [Online]," 2001. http://info.iet.unipi.it/ luigi/polling/.

[31] L. Deri, "Improving passive packet capture: Beyond device polling," *Proceedings of SANE*, Jan 2004.

[32] L. Deri and F. Fusco, "Exploiting commodity multi-core systems for network traffic analysis," pp. 1–11, Jan 2009.

[33] A. Kirsch, M. Mitzenmacher, and G. Varghese, "Hash-based techniques for high-speed packet processing," *Algorithms for Next Generation Networks*, pp. 181–218.

[34] Y. Qi, B. Xu, F. He, B. Yang, J. Yu, and J. Li, "Towards high-performance flow-level packet processing on multi-core network processors," *ANCS '07: Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, Dec 2007.

[35] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: an aid to network processing," *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, Aug 2005.

[36] M. Zukowski, S. Héman, and P. Boncz, "Architecture-conscious hashing," *DaMoN '06: Proceedings of the 2nd international workshop on Data management on new hardware*, Jun 2006.

[37] R. Pagh and F. F. Rodler, "Cuckoo hashing," Jan 2002.

[38] A. Kirsch, M. Mitzenmacher, and U. Wieder, "More robust hashing: Cuckoo hashing with a stash," *Algorithms-ESA 2008*.

[39] K. Wu, S. Ahern, E. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. Geddes, J. Gu, H. Hagen, B. Hamann, W. Koegler, J. Lauret, J. Meredith, P. Messmer,

E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. R. bel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M. Zhang, "Fastbit: interactively searching massive data," *Journal of Physics: Conference Series 180*, Jan 2009.

[40] G. Antoshenkov, "Byte-aligned bitmap compression," *Data Compression Conference*, Jan 2002.

[41] E. Otoo and A. Shoshani, "An efficient compression scheme for bitmap indices," *Citeseer*, Jan 2004.

[42] K. Wu, K. Stockinger, and A. Shoshani, "Breaking the curse of cardinality on bitmap indexes," *Scientific and Statistical Database Management*, Jan 2008.

[43] L. Deri, V. Lorenzetti, and S. Mortimer, "Collection and exploration of large data monitoring sets using bitmap databases," *Traffic Monitoring and Analysis*, Jan 2010.

[44] J. J. Garrett, "Ajax: A New Approach to Web Applications [Online]." http://www.adaptivepath.com/ideas/essays/archives/000385.php.

[45] A. Russell, "Comet: Low latency data for browsers," *alex.dojotoolkit.org*, Jan 2006.

[46] E. Bozdag, A. Mesbah, and A. V. Deursen, "Performance testing of data delivery techniques for ajax applications," *Journal of Web Engineering*, Jan 2008.

[47] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy, "Flow processing and the rise of commodity network hardware," *SIGCOMM Computer Communication Review*, vol. 39, Mar 2009.

[48] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Computer Communication Review*, vol. 38, Mar 2008.

[49] R. Ramaswamy and T. Wolf, "High-speed prefix-preserving ip address anonymization for passive measurement systems," *IEEE/ACM TRANSACTIONS ON NETWORKING*, vol. 15, Feb 2007.

[50] T. Dierks and E. Rescorla, "Rfc 5246—the transport layer security (tls) protocol version 1.2," *Internet Engineering Task Force*, Jan 2008.

[51] B. Jenkins, "A Hash Function for Hash Table Lookup [Online]." http://www.burtleburtle.net/bob/hash/doobs.html.

[52] A. Appleby, "MurmurHash [Online]." http://sites.google.com/site/murmurhash/.