POLITECNICO DI MILANO
V Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

ANALISI E CONFRONTO DI SCHEMI DI ROUTING CONTENT-BASED TRAMITE SIMULAZIONE

Relatore: Prof. Gianpaolo CUGOLA
Correlatore: Prof. Antonio CARZANIGA

Tesi di Laurea di:
Marco DELEVATI
Matr. n. 667972

Anno Accademico 2006-2007

Ringraziamenti

Giunto alla conclusione di questo lavoro, volevo ringraziare innanzitutto il mio relatore, Prof. Gianpaolo Cugola, e il mio correlatore, Prof. Antonio Carzaniga. Essi mi hanno dato la possibilità di lavorare su una tematica interessante e stimolante, facendomi conoscere e apprezzare il mondo della ricerca universitaria. Ciò per cui li ringrazio maggiormente, però, è di avermi permesso di lavorare in un clima amichevole ed informale e di essersi sempre dimostrati disponibili a fornirmi utili consigli e spiegazioni per superare i problemi che man mano ho incontrato nello sviluppo del presente lavoro. Tutto questo mi ha permesso di affrontare il periodo di tesi serenamente, molto spesso divertendomi, di imparare moltissimo e in generale di trarre il meglio da questa esperienza.

Non posso fare a meno di ringraziare anche i miei genitori che, nonostante le inevitabili incomprensioni, mi hanno sempre sostenuto, non mi hanno mai fatto mancare nulla, nemmeno le cose più superflue, e mi hanno permesso di arrivare dove sono. Così come non posso non ringraziare le uniche due persone al mondo che sanno sempre come prendermi (e non è facile!) e che spesso in questi anni hanno saputo smorzare i miei picchi di nervosismo: mia sorella e la nonna Clelia. Tra gli amici, un ringraziamento particolare lo dedico alla Fedi che col suo sorriso riesce sempre a mettermi di buon umore e che, nel bene e nel male, è la persona con cui ho condiviso quasi tutti i momenti più importanti della mia vita durante questo periodo universitario.

Infine voglio ringraziare tutti i parenti e gli amici che ho avuto vicino in questi anni, chi più chi meno hanno tutti contribuito a rendermi la persona che sono in questo momento.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Il sempre maggior numero di applicazioni distribuite presenti su Internet e sulle reti aziendali ha fatto emergere in maniera crescente il limite della comunicazione cosiddetta *address-based*, ovvero basata sull'indirizzo del destinatario di un messaggio. Tale paradigma di comunicazione risulta essere svantaggioso e poco efficiente in tutti quei contesti in cui vi siano più destinatari interessati ad un determinato messaggio. Si pensi ad esempio ad una newsletter, ai gruppi di comunicazione, alle chat, ai sistemi di aste online, ai sistemi di comunicazione aziendali, ai giochi online, ai sistemi di distribuzione delle informazioni relative ai titoli azionari, alle community online, ai sistemi peer-to-peer, ai sistemi di controllo del traffico, ai sistemi di allerta: sono solo alcuni esempi delle numerose applicazioni in cui un messaggio generato in un singolo punto della rete interessa numerosi destinatari, ai quali deve essere recapitato.

Per venire incontro alle crescenti esigenze di tali applicazioni si è dapprima pensato di intervenire a livello di rete, attraverso lo sviluppo di servizi di *multicast*. In generale il servizio di multicast (e in particolare multicast IP) funziona nel seguente modo: i destinatari interessati ad un certo tipo di informazioni si iscrivono ad un *gruppo di multicast*, il quale ha associato un indirizzo multicast. Una sorgente che voglia inviare un certo messaggio lo spedisce all'indirizzo del gruppo di multicast interessato a tale tipo di messaggio, saranno poi i router a distribuire il messaggio ai destinatari iscritti al gruppo. Tale soluzione risulta poco flessibile, presentando lo svantaggio di dover creare un gruppo di multicast per ogni possibile tipo di interesse e limitando così sia il numero dei possibili interessi sia la complessità degli stessi. Per questo motivo i servizi di multicast vengono usati principalmente in quegli ambiti in cui il numero degli interessi è limitato e gli interessi sono ben definiti, si pensi ad esempio ad un insieme di canali televisivi in cui ogni destinatario è interessato solo ad un insieme ben preciso di canali e il contenuto (film, docu-

mentari, etc...) è univocamente associato ad un singolo canale. Inoltre i servizi di multicast possono essere impiegati per fornire una base su cui costruire servizi più complessi a livello di middleware o applicativo.

Per garantire una maggiore flessibilità si è quindi pensato di adottare una soluzione simile ai gruppi di multicast, spostandosi però a livello di middleware: i sistemi *publish/subscribe*.

1.1 Publish/Subscribe

Il paradigma publish/subscribe è un paradigma di comunicazione asincrona basato su due primitive: i destinatari esprimono il loro interesse verso un certo tipo di contenuti tramite una primitiva di *subscribe*, mentre le sorgenti inviano i loro messaggi (o eventi) tramite una primitiva di *publish*. Il vantaggio di questo tipo di comunicazione è che le sorgenti dei messaggi possono semplicemente inviare i messaggi, senza preoccuparsi di chi sono i destinatari, mentre i destinatari possono semplicemente dichiarare il loro interesse per una certa classe di messaggi senza preoccuparsi di contattare le sorgenti per riceverli. Tipicamente per realizzare questo modello di comunicazione le sottoscrizioni sono raccolte da componenti detti *broker* (o *event dispatcher*), i quali si incaricano anche di consegnare i messaggi a tutti i destinatari interessati. Essendo implementato a livello di middleware questo meccanismo permette quindi di dichiarare un numero elevato di interessi, inoltre il grande grado di disaccoppiamento tra sorgenti e destinatari rende questo sistema particolarmente adatto a contesti dinamici, in cui sorgenti e destinatari cambiano in continuazione, così come i loro interessi.

I sistemi publish/subscribe si differenziano tra loro per due aspetti principali: la complessità delle sottoscrizioni e l'architettura dei broker. A seconda dell'espressività del linguaggio utilizzato per le sottoscrizioni si può infatti distinguere tra sistemi *subject-based* (o *topic-based*) e sistemi *content-based*. Nei sistemi subject-based esiste un insieme, definito a priori, di valori e solo uno di questi può essere associato ad ogni messaggio. Di conseguenza, un ricevente può solamente dichiarare il suo interesse per un sottoinsieme di questi valori, ricevendo i messaggi con associato uno dei valori desiderati. Nei sistemi content-based, invece, le sottoscrizioni sono costituite da espressioni che permettono di selezionare un messaggio sulla base del suo stesso contenuto. In tali sistemi l'insieme delle sottoscrizioni possibili non è più definito a priori, ma è determinato dalla complessità delle espressioni ammissibili. Inoltre, a differenza di ciò che avviene nei sistemi subject-based, un singolo messaggio può essere selezionato da più sottoscrizioni diverse tra loro. Uno

dei principali problemi di questi sistemi è quindi quello di riuscire a trovare un giusto equilibrio tra l'espressività delle sottoscrizioni e la complessità della implementazione. In tale contesto si parla poi di *filtering* per indicare il processo con cui si verifica se un messaggio viene selezionato da una espressione e, ovviamente, maggiore è l'espressività del linguaggio delle sottoscrizioni, più complessa risulterà la fase di filtering.

Per quanto riguarda l'architettura del broker, essa può essere centralizzata o distribuita. Nelle soluzioni centralizzate i destinatari contattano il broker per inviargli le proprie sottoscrizioni, il broker raccoglie tutte le sottoscrizioni, riceve i messaggi spediti dalle sorgenti e li inoltra a tutti i destinatari interessati. Nelle soluzioni distribuite ci sono invece più broker, tipicamente organizzati in una *overlay network*, che collezionano le sottoscrizioni dagli host locali e cooperano tra di loro per instradare i messaggi.

Il modello publish/subscribe viene di norma implementato dai middleware orientati ai messaggi (MOM, Message-Oriented Middleware), di cui i sistemi commerciali più conosciuti sono IBM WebSphere MQ, Java Messaging Service (JMS), Tibco Rendezvous. Tuttavia la grande maggioranza dei sistemi commerciali adotta l'approccio con un broker centralizzato e molti di questi sistemi sono sistemi subject-based [8]. Il motivo principale per cui tali sistemi non adottano ancora un approccio content-based veramente distribuito risiede nel fatto che uno dei limiti principali dei sistemi publish/subscribe, e soprattutto di quelli content-based, è la scalabilità. Per questo motivo, per quanto il paradigma publish/subscribe sarebbe quello più naturale per tantissime applicazioni presenti su Internet, non esistono, ad oggi, soluzioni, commerciali e non, in grado di realizzarlo su grosse reti né tanto meno a livello di Internet [15]. Pertanto gli sforzi della ricerca in questo settore si muovono su due fronti: l'ottimizzazione della fase di filtering, tramite l'ideazione di algoritmi di *matching* sempre più sofisticati, per far fronte alla diversità e al gran numero di sottoscrizioni tipiche di un ambiente eterogeneo e dinamico come Internet e la ricerca di soluzioni al problema della distribuzione delle sottoscrizioni su larga scala in maniera ottimale. Pertanto tali argomenti di ricerca fanno parte dell'area di ricerca relativa alla *comunicazione content-based*.

1.2 La Comunicazione Content-Based

Storicamente, per superare i limiti di scalabilità precedentemente illustrati, è stato necessario cambiare prospettiva, vedendo il modello publish/subscribe, in maniera più astratta, come un servizio di comunicazione. Per farlo è stato introdotto il

concetto di comunicazione content-based, ovvero un tipo di comunicazione in cui il flusso dei messaggi dal mittente al destinatario è guidato dal contenuto dei messaggi stessi piuttosto che dall'indirizzo del destinatario dei messaggi [4]. Si parla quindi di servizio di comunicazione content-based, per indicare il servizio che si occupa di far arrivare un determinato messaggio a tutti i destinatari interessati ad esso.

Ovviamente l'impiego più naturale di un servizio di questo tipo è nell'ambito dei sistemi publish/subscribe che per loro stessa natura presentano già un disaccoppiamento tra mittenti e destinatari interessati al contenuto di determinati messaggi. Ma tale paradigma si adatta bene anche a tutti quei contesti in cui la comunicazione è di tipo asincrono.

Da questo punto di vista quindi, il modello publish/subscribe viene visto come un vero e proprio servizio di rete. Si fa notare che non importa se tale servizio viene implementato a livello di rete (come ad esempio IP) o a livello applicativo (come overlay network), quello che conta è che esso viene considerato un vero e proprio servizio di rete, con tutto quello che ne consegue. Una conseguenza di questo approccio è che, per raggiungere il suo obiettivo, un servizio di comunicazione content-based si appoggia ad un servizio di routing in grado di distribuire in tutta la rete le informazioni necessarie ai "router" per instradare correttamente i messaggi dalle sorgenti ai riceventi interessati. Servono quindi dei protocolli di routing in grado di distribuire efficientemente tali informazioni e degli algoritmi distribuiti che siano in grado di instradare correttamente i messaggi coerentemente con le informazioni contenute in ogni singolo router, tali algoritmi vengono detti *schemi di routing*. In tale contesto si parla quindi di *content-based routing*.

1.3 Obiettivi del Presente Lavoro

Il presente lavoro di tesi si colloca nel contesto precedentemente descritto, cioè all'interno dell'area di ricerca della comunicazione content-based, e in particolare si occupa di problematiche relative al content-based routing. L'obiettivo proposto è quello di analizzare e mettere a confronto quattro schemi di routing content-based ed in particolare i quattro schemi descritti in [3]. Tali schemi di routing content-based sono quelli ad oggi più promettenti nell'ottica di portare la comunicazione content-based su reti geografiche e, pertanto, è sembrato opportuno sia analizzarli per scoprirne eventuali debolezze, punti critici e punti di forza sia confrontarli tra loro per capire se, nella pratica, uno sia migliore di un altro, globalmente oppure in determinati scenari. Si fa notare che l'ottica con la quale sono stati effettuati i confronti è per lo più quella delle prestazioni di rete, anche se in alcuni casi sono

stati analizzati anche altri aspetti, come ad esempio l'occupazione di memoria.

Per poter confrontare tali schemi si è quindi deciso di simularne il comportamento utilizzando un simulatore ad eventi. Dapprima è stato ideato un modello sufficientemente complesso dello scenario di rete da simulare e si sono scelti dei parametri di configurazione di tale modello su cui poter intervenire, per poterlo rendere il più possibile attinente alla realtà. Si è poi passati alla scelta del simulatore ad eventi più appropriato per rappresentare il modello ideato e la scelta è caduta su `OMNet++`. In seguito sono stati implementati gli schemi in questione all'interno del simulatore, sono stati definiti tutti i parametri della simulazione, sono state scelte le misure da raccogliere durante le simulazioni e infine sono stati analizzati i risultati delle simulazioni traendone le dovute conclusioni.

1.4 Organizzazione dell'esposizione

L'organizzazione dell'esposizione rispecchia le tappe affrontate per portare a termine il presente lavoro, come descritte nel paragrafo precedente. Pertanto nel Capitolo 2 verrà spiegato nel dettaglio cosa si intende per content-based routing e verranno descritti i quattro schemi di routing oggetto dello studio, nel Capitolo 3 verrà esposto il modello delle simulazioni effettuate, le assunzioni fatte, le misure scelte, e gli strumenti statistici utilizzati, mentre nel Capitolo 4 verrà descritto come tale modello è stato implementato e verranno spiegate le scelte implementative più importanti, dando anche una breve panoramica sul simulatore `OMNet++`. Il Capitolo 5 è invece riservato all'esposizione dei risultati ottenuti e ad una loro analisi. Infine nel Capitolo 6 verranno espresse le conclusioni finali suggerite dai risultati ottenuti, con particolare attenzione ad indirizzare gli sforzi futuri di ricerca in questo ambito.

A questo punto risulta opportuno aprire una piccola parentesi per quanto riguarda il linguaggio utilizzato. La letteratura riguardante il content-based routing è interamente in lingua inglese e ai fini dell'esposizione si è deciso di non operare, come troppo spesso si vede fare, improbabili traduzioni dall'inglese, ma di lasciare la terminologia originale, non solo per quei termini che costituiscono de facto la nomenclatura di riferimento, ma anche per alcune parole che, pur non essendo termini tecnici, risultano più chiare e comprensibili in lingua inglese, o perché non hanno una traduzione soddisfacente in italiano oppure perché la loro traduzione non consente di coglierne le sfumature di significato.

Capitolo 2

Il Content-Based Routing

Per comunicazione content-based si intende un tipo di comunicazione in cui il flusso dei messaggi dal mittente al destinatario è guidato dal contenuto dei messaggi stessi piuttosto che dall'indirizzo del destinatario dei messaggi [4]. Più nel dettaglio in un servizio di comunicazione content-based i riceventi dichiarano i loro interessi per mezzo di opportuni predicati di selezione, mentre le sorgenti semplicemente spediscono i messaggi [2]. Il servizio di comunicazione si occupa di far arrivare un determinato messaggio a tutti i destinatari interessati ad esso. All'interno di questo contesto si parla quindi di *content-based networking*, cioè un servizio di comunicazione orientato ai messaggi in cui un messaggio viene consegnato a tutte le destinazioni che hanno dichiarato un predicato di selezione che comprende il contenuto di quel messaggio e che sono perciò interessate a tale messaggio.

Col termine *content-based routing*, d'ora in avanti *CBR*, si intende l'algoritmo distribuito che si incarica di consegnare i messaggi dai mittenti ai destinatari interessati. Il tipo di algoritmo utilizzato determina il cosidetto schema di routing o *routing scheme*. Analogamente a quello di una tradizionale rete address-based, uno schema di routing in una rete content-based definisce delle funzioni di *matching*, *forwarding* e *header* che tutte insieme realizzano la funzione di consegna dei messaggi.

Nel presente capitolo verrà dapprima proposto un modello standard di una rete content-based e delle sue funzioni di routing e quindi, con riferimento al modello appena presentato, verranno descritti e spiegati i quattro schemi di routing oggetto del presente lavoro di tesi. Il modello di una rete content-based, così come le descrizioni degli schemi di routing sono presi da [3]. In [3], tuttavia, il taglio della presentazione è incentrato sulla analisi della occupazione di memoria degli schemi di routing presentati, mentre ai fini del nostro lavoro tali particolari vengono tralasciati, offrendo un taglio più orientato alla analisi delle prestazioni di rete. Allo

stesso modo, in quanto lo scopo di questo capitolo è di fornire al lettore tutti gli strumenti necessari per comprendere i risultati ottenuti, senza nessuna pretesa di coprire per intero la tematica del CBR, verranno tralasciati tutti quei dettagli che non sono ritenuti strettamente necessari a tale scopo, come ad esempio gli aspetti relativi alle dimostrazioni di correttezza ed efficienza degli schemi presentati che possono essere trovate altrove (vedi sempre [3]). Sempre per agevolare la comprensione dei risultati ottenuti, la fine del capitolo è dedicata a fare chiarezza sull'uso di alcuni termini tipicamente utilizzati nel contesto del CBR e ad alcune osservazioni conclusive.

2.1 Modello di una Rete Content-Based

2.1.1 Alcune Definizioni

Una rete content-based si può modellare come un grafo connesso, non orientato e finito $G = (V, E)$, dove un vertice $v \in V$ rappresenta un *processor* e un arco $e \in E$ rappresenta un canale di comunicazione bidirezionale tra due processor. Senza perdita di generalità, si può assumere che un processor agisca sia come un *host*, che produce e consuma messaggi, sia come un *router*, che inoltra i messaggi per conto di altri processor. Ogni processor ha un identificatore numerico, per cui se $n = |V|$, allora l'insieme dei processor è dato da $V = \{1, \ldots, n\}$.

Una rete content-based è anche definita dall'insieme $\mathcal{M}$ dei messaggi ammissibili e dall'insieme $\mathcal{P}$ dei possibili predicati di selezione su $\mathcal{M}$. Un predicato $p \in \mathcal{P}$ è un funzione booleana totale $p : \mathcal{M} \to \{0, 1\}$ che definisce implicitamente un insieme di messaggi $\mathcal{M}(p) = \{m | m \in \mathcal{M} \wedge p(m) = 1\}$. Quando non vi sia rischio di ambiguità, tale notazione viene semplificata usando direttamente i predicati per denotare l'insieme di messaggi che definiscono, scrivendo quindi semplicemente p per indicare $\mathcal{M}(p)$. Conseguentemente si scrive $m \in p$ per dire che un predicato p individua un messaggio m quando $p(m) = 1$.

Ai fini della nostra definizione generale del modello di rete content-based, possiamo ignorare la concreta struttura e rappresentazione dei messaggi e dei predicati. Tuttavia, per rendere più agevole la comprensione dei concetti qui esposti, si può pensare ad un messaggio in una implementazione concreta di una rete content-based, come ad una tupla di attributi. Per esempio il messaggio $[alert = congestion, severity = 3, location = highway1]$ potrebbe descrivere un evento relativo ad una applicazione di controllo del traffico stradale. Un predicato di selezione consisterà tipicamente in una disgiunzione di congiunzioni di vincoli sugli attributi.

Per esempio, il predicato $(alert = congestion \wedge severity > 2) \vee (alert = accident)$ selezionerebbe il messaggio precedente.

Per la definizione del nostro modello avremo bisogno delle seguenti definizioni di funzioni: una funzione *vicino*, $N : V \to \mathbb{P}(V)$, tale che $N(v)$ restituisce i vicini del processor v. Una funzione *predicato*, $pred : V \to \mathcal{P}$ che associa ogni processor v con un predicato $pred(v)$. Il predicato $pred(v)$ rappresenta gli interessi di tutte le applicazioni presenti su v, per cui si dice che un processor v è interessato ad un messaggio m quando $m \in pred(v)$.

2.1.2 Lo Schema di Routing Content-Based

Passiamo ora a definire cosa si intende per schema di routing content-based con riferimento alle definizioni introdotte precedentemente. Innanzitutto, affinché la definizione funzioni è necessaria la seguente assunzione: un processor v può spedire un *pacchetto* a uno dei suoi vicini tramite l'invocazione di una primitiva di comunicazione a livello di link e, per invocare tale primitiva, v necessita solamente di specificare la destinazione u alla quale intende inviare il pacchetto. Nel nostro contesto un pacchetto c è costituito da un messaggio $msg(c) \in \mathcal{M}$ e da un header $hdr(c) \in \mathcal{H}$, dove $\mathcal{H}$ è l'insieme dei possibili header dei messaggi. Ogni schema di routing definisce il suo specifico insieme degli header $\mathcal{H}$ e ovviamente l'header di un pacchetto contiene informazioni utili per la corretta consegna del pacchetto stesso in accordo con lo specifico schema di routing.

Oltre ad $\mathcal{H}$, uno schema di routing content-based definisce un algoritmo distribuito in cui ogni processor v implementa le seguenti funzioni:

- una *initial header function*, $\mathsf{Init}_v : \mathcal{M} \to \mathcal{H}$ che, dato un nuovo messaggio m generato da v, genera un header iniziale $\mathsf{Init}_v(m)$;
- una *header function*, $\mathsf{Hdr}_v : \mathcal{H} \to \mathcal{H}$ che, dato un header h restituisce un nuovo header $\mathsf{Hdr}_v(h)$;
- una *forwarding function*, $\mathsf{Fwd}_v : \mathcal{H} \times \mathcal{M} \to \mathbb{P}(N(v))$ che, dato un header h e un messaggio m restituisce un sottoinsieme $\mathsf{Fwd}_v(h, m)$ dell'insieme dei vicini $N(v)$ di v.

Uno specifico schema di routing è quindi definito completamente dal tipo di header e dalle funzioni Init, Hdr e Fwd. Un generico schema di routing si comporta quindi nel seguente modo: per un messaggio m con origine in v, il processor v crea l'header del rispettivo pacchetto come $h = \mathsf{Init}_v(m)$ e spedisce un pacchetto $c = \langle h, m \rangle$ a tutti i processor in $\mathsf{Fwd}_v(h, m)$. Quando, invece, un processor u deve

inoltrare un pacchetto in arrivo c, esso ne estrae l'header $h = hdr(c)$ e il messaggio $m = msg(c)$, calcola l'insieme dei next-hop processor $\mathsf{Fwd}_u(h, m)$ e inoltra a tutti i processor contenuti in questo insieme un nuovo pacchetto $c' = \langle \mathsf{Hdr}_u(h), m \rangle$.

2.1.3 Correttezza ed Efficienza di uno Schema di Routing Content-Based

Molto informalmente uno schema di routing content-based S si dice *corretto* quando, dato un messaggio m immesso nella rete da un processor v, m viene consegnato a tutti i processor interessati ad esso, ovvero a tutti quei processor che dichiarano un predicato di selezione p, tale che $p(m) = 1$.

Uno schema di routing dovrebbe essere anche efficiente. Tuttavia, data la natura "multicast" del content-based routing, l'efficienza di uno schema di routing content-based può essere intesa in diversi modi. Nel presente lavoro si misurerà l'efficienza di uno schema di routing secondo due misure:

- il costo totale di comunicazione di un messaggio, inteso come il numero di pacchetti trasmessi per ogni messaggio immesso nella rete;

- la latenza di consegna di un messaggio per ogni processor.

Pertanto S minimizza la latenza se, dato un messaggio m immesso nella rete dal processor v, il messaggio m segue il percorso minimo da v ad ogni processor interessato u, dove il percorso minimo è quello calcolato ponendo come pesi dei link le loro le latenze. S minimizza invece il costo totale di comunicazione se il cammino da v ad ogni processor u è quello col minor numero possibile di hop. Uno schema di routing si dice quindi *ideale* se minimizza nell'ordine la latenza e il costo totale di comunicazione.

2.2 Gli Schemi di Routing

In questa sezione vengono spiegati i quattro schemi di routing content-based oggetto del presente lavoro. D'ora in avanti tali schemi di routing verranno nominati solamente tramite la loro sigla presente nel titolo del relativo paragrafo.

Nel prosieguo inoltre si userà la convenzione per cui i processor sorgenti saranno sempre indicati con la lettera v, mentre tutti gli altri, siano essi intermedi o riceventi, con la lettera u.

2.2.1 PSF: Per-Source Forwarding

L'idea del PSF è quella di rappresentare, per ogni processor sorgente v, l'albero dei cammini minimi con radice in v, T_v (cioè T_v è l'albero calcolato con l'algoritmo di Dijkstra). Per spiegare questo schema di routing è utile definire alcune notazioni:

- $child_u(v)$ restituisce l'insieme dei figli del processor u nell'albero T_v
- T_v/w indica l'insieme dei processor presenti nel sotto-albero di T_v avente radice in w
- $T_v/(u,w)$ indica l'insieme dei processor discendenti dell'arco (u,w) nell'albero T_v o più formalmente

$$T_v/(u,w) = \begin{cases} T_v/w, & \text{se } w \in child_u(v) \\ \emptyset, & \text{altrimenti} \end{cases}$$

Per esempio, nella rete di Figura 2.1, $T_1/(6,7)$ è l'insieme $\{4,7,8,11,12\}$, mentre $T_1/(6,9)$ è l'insieme vuoto.

Per ogni processor sorgente v, PSF annota per ogni arco (u,w) la disgiunzione dei predicati dei processor appartenenti all'insieme $T_v/(u,w)$. Le rappresentazioni di questi alberi sono distribuite attraverso tutta la rete, in modo che ogni processor possa memorizzare la sua visione locale dell'albero T_v per ogni processor v. Più nel dettaglio, il processor u associa un predicato ad ogni ramo (u,w) in T_v, per ogni processor v (incluso $v = u$) e per ogni processor vicino $w \in N(u)$. Inoltre lo schema inserisce nell'header del pacchetto l'identificativo del processor sorgente v di un messaggio m, in modo che ogni processor u, ricevendo un pacchetto contenente m, possa inoltrarlo lungo i rami di T_v.

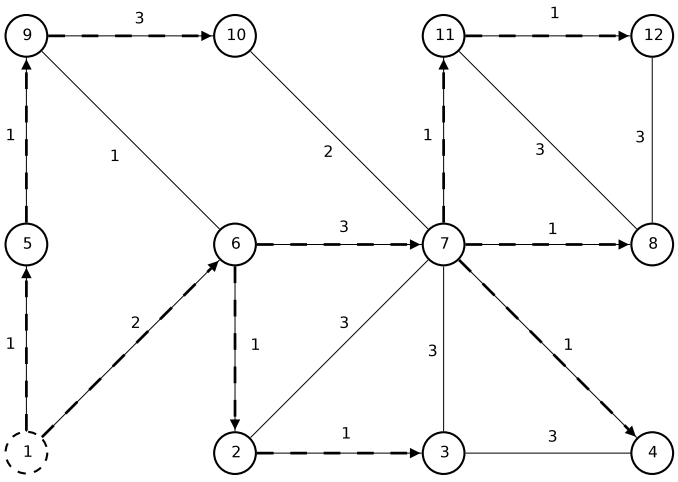


Figura 2.1: Rete di esempio con indicato l'albero T_1.

sorgente	*prossimo-hop*	→	*predicato*
...			
1	2	→	$p_2 \vee p_3$
1	7	→	$p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12}$
1	9	→	$\emptyset$
...			

Figura 2.2: Funzione F_6 per la rete d'esempio.

Formalmente, PSF consiste in una funzione $F_u : V \times N(u) \rightarrow \mathcal{P}$ che mette in relazione un processor sorgente v e un processor vicino w con un predicato $F_u(v, w) = \bigvee_{x \in T_v/(u,w)} pred(x)$.

La Figura 2.1 mostra una rete d'esempio: le linee sottili rappresentano i link della rete, ognuno dei quali con associato un peso (che potrebbe essere inteso come la latenza), mentre le linee spesse tratteggiate rappresentano gli archi dell'albero dei cammini minimi T_1 avente radice nel processor 1. La Figura 2.2 mostra invece un frammento rilevante della funzione F_6 del processor 6 (nella tabella la notazione p_x significa $pred(x)$).

Le funzioni Init, Hdr e Fwd per il PSF sono quindi le seguenti:

- $\mathsf{Init}_v(m) = v$, per ogni processor v e per ogni messaggio m
- $\mathsf{Hdr}_u(h) = h$, per ogni processor u e per ogni messaggio m
- $\mathsf{Fwd}_u(v, m) = \{w | m \in F_u(v, w)\}$, dove v è il processor sorgente estratto dall'header h

Quindi ogni processor crea un header iniziale con il suo identificativo mentre per i pacchetti che deve inoltrare lascia lo stesso header del pacchetto in entrata anche in quello di uscita. La funzione di forwarding Fwd_u chiamata su un messaggio m, legge il processor sorgente v dall'header h e restituisce l'insieme dei processor del prossimo hop $\mathsf{Fwd}_u(v, m)$, cioè u inoltra il messaggio m avente origine in v a tutti i processor successori di u nell'albero T_v, tali che l'arco (u, w) in T_v è associato ad un predicato p che seleziona m. Il processor u compie questa decisione di inoltro basandosi esclusivamente sulla sua vista locale di T_v.

2.2.2 iPS: Improved Per-Source Forwarding

Per quanto PSF sia uno schema ideale[1] esso presenta un limite nella quantità di informazioni che ogni nodo deve memorizzare localmente. Senza entrare nei dettagli[2], la quantità di memoria occupata da PSF è fortemente condizionata sia

[1] in [3] se ne può trovare una dimostrazione informale.

[2] vedi sempre [3] per una analisi approfondita della quantità di memoria occupata da PSF.

dal numero r dei processor riceventi sia da quello s dei processor che immettono messaggi nella rete (*senders*). Intuitivamente ciò avviene perché ogni ulteriore processor s, provoca la gestione di un ulteriore albero dei cammini minimi T_s, mentre ogni ricevente in più r provoca un aumento del numero di predicati p_r presenti nella disgiunzione associata ai rami degli alberi T_s.

Tuttavia, a seconda della topologia della rete, gli alberi radicati in due o più processor sorgenti potrebbero sovrapporsi o del tutto o solo parzialmente. Un semplice esempio è quello di una rete in cui vi sia un arco $e \in E$ che sia anche un "ponte" (cioè tale che in seguito alla sua rimozione la rete risulta partizionata in due reti separate): in questo caso tutti i processor sorgenti da un lato di e sarebbero indistinguibili dal punto di vista dei processor dall'altro lato di e e viceversa. Questo accade perché tutti i cammini che vanno da tutte le sorgenti di una parte a tutti i riceventi dell'altra parte devono obbligatoriamente passare per il ponte e quindi dal ponte in poi potrebbero usare tutti lo stesso albero. Infatti tutti gli alberi radicati nei processor indistinguibili hanno, dal ponte in poi, gli stessi archi ed ogni arco ha associati gli stessi predicati, in quanto i riceventi sono i medesimi.

L'esempio precedente mostra come PSF possa sicuramente essere ottimizzato e questo è il motivo per cui è stato introdotto lo schema di routing iPS. L'idea di iPS è che, se due processor sorgenti v_1 e v_2 sono *indistinguibili* dal punto di vista del processor u, allora u può ridurre le loro entry nella funzione F_u ad una sola, in quanto tali entry hanno gli stessi next-hop e le stesse disgiunzioni di predicati e sono perciò identiche. Come esempio estremo si può considerare una rete G, consistente in un albero: in questo caso ogni arco della rete è un ponte e perciò, dal punto di vista di ogni processor u, tutte le sorgenti raggiungibili tramite lo stesso vicino $w \in N(u)$ sono indistinguibili. Pertanto il dominio di F_u per tutti i processor u può essere ridotto da V a $N(u)$.

Verosimilmente la condizione dell'esistenza di un ponte è solo una condizione sufficiente e ci possono essere condizioni più deboli per cui due processor sorgenti si possono considerare indistinguibili. Per esempio, se si considera la rete in Figura 2.1, i processor 1, 5 e 9 sono indistinguibili dal punto di vista del processor 6 (infatti da 6 in poi gli alberi dei cammini minimi T_1, T_5 e T_9 coincidono con quello disegnato in figura).

Vediamo ora di dare una definizione un po' più formale del concetto di indistinguibilità di due sorgenti. La relazione di indistinguibilità è una relazione di equivalenza definita per ogni processor u. Ogni classe di equivalenza ha come rappresentante uno dei suoi elementi, che la identifica, e l'insieme di questi rappresentanti è denotato da $\overline{V}_u \subseteq V$. L'iPS fa uso di uno funzione $I_u : V \to \overline{V}_u$ che mette in relazione un processor sorgente v col proprio rappresentante $I_u(v)$. L'iPS

fa uso anche di una funzione $F_u : \overline{V}_u \times N(u) \to \mathcal{P}$ che è concettualmente identica a F_u definita per il PSF, tranne per il dominio che è più piccolo. La funzione Fwd_u risulta quindi simile a quella del PSF, tranne per il fatto che fa uso della funzione I_u per comprimere lo spazio dei processor sorgenti. In maniera formale, $\mathsf{Fwd}_u = \{w | m \in F_u(I_u(v), w)\}$.

Inoltre, in alcuni casi, una classe di equivalenza di processor sorgenti definita per il processor u e contenente il processor v è la stessa per tutti i processor $w \in T_v/u$. Ciò accade ogniqualvolta ci sia un ponte in una rete, ma può anche accadere in altri casi. Per esempio, sempre con riferimento alla Figura 2.1, i processor 1, 5 e 9 restano indistinguibili dalla prospettiva di tutti i processor che sono discendenti di 6 nell'albero T_1 (cioè i processor 2, 3, 4, 7, 8, 11 e 12). Infatti il processor 6 è anch'esso indistinguibile dalla prospettiva di tutti i suoi discendenti. In tali casi sembra quindi sensato riscrivere l'header del pacchetto, in modo da sostituire l'identificativo della sorgente v con il processor visitato più di recente, oltre il quale le sorgenti restano indistinguibili. In questo modo si comprime il dominio delle funzioni I_u in tutti i discendenti. A questo scopo lo schema iPS definisce una funzione di riscrittura $R_u : V_u \to \{0, 1\}$ per ogni processor u. R_u viene usata in Hdr_u per decidere se riscrivere l'header di un pacchetto entrante.
Possiamo, a questo punto, specificare più nel dettaglio le funzioni Init, Hdr e Fwd per l'iPS:

- $\mathsf{Init}_v(m) = v$, per ogni processor v e per ogni messaggio m
- $\mathsf{Hdr}_u(v) = \begin{cases} v, & \text{se } R_u(I_u(v)) = 0 \\ u, & \text{se } R_u(I_u(v)) = 1 \end{cases}$ dove v è il processor sorgente estratto dall'header h
- $\mathsf{Fwd}_u(v, m) = \{w | m \in F_u(I_u(v), w)\}$, dove v è il processor sorgente estratto dall'header h

Da ultimo definiamo la condizione che permette ad un processor u di combinare due o più sorgenti. Si ricorda che per ogni sorgente v, la funzione di forwarding F_u mette in relazione un processor vicino w con un predicato $F_u(v, w)$, che è la disgiunzione dei predicati $pred(x)$ di tutti i processor x che sono discendenti di u nell'albero T_v attraverso l'arco (u, w).

Due processor sorgenti v_1 e v_2 si dicono *indistinguibili* dal punto di vista del processor u, se valgono le due condizioni seguenti:

1. $child_u(v_1) = child_u(v_2)$, cioè ogni figlio del processor u nell'albero T_{v_1} è figlio di u anche in T_{v_2} e viceversa;

2. $\forall w \in child_u(v_1) : T_{v_1}/w = T_{v_2}/w$, cioè ogni figlio w di u ha lo stesso insieme di discendenti in T_{v_1} e in T_{v_2}.

2.2.3 PIF: Per-Interface Forwarding

PIF è uno schema di routing simile al PSF, ma in cui il dominio delle funzioni di forwarding presenti in ogni processor è limitato ai nodi vicini. Similmente allo schema di routing PSF, PIF è basato sugli alberi dei cammini minimi diretti dalla sorgente verso gli altri nodi. Si ricorda che lo schema PSF associa un predicato ad ogni arco dell'albero dei cammini minimi T_v dove v è la sorgente del messaggio. In questo modo, per ogni processor sorgente v, la funzione di forwarding F_u di PSF per il processor u mette in relazione l'arco (u, w) con il predicato $F_u(v, w)$ che è la disgiunzione di tutti i predicati dei processor discendenti di u in T_v tramite l'arco (u, w).

Lo schema PIF si basa sull'idea di associare ad ogni arco (u, w) un singolo predicato che combina i predicati dei discendenti tramite (u, w) per tutti gli alberi dei cammini minimi di tutti processor sorgenti. Per fare questo PIF mantiene per ogni processor u una funzione $F_u^* : N(u) \to \mathcal{P}$ che mette in relazione un vicino $w \in N(u)$ con la disgiunzione, su tutte le sorgenti v, dei predicati dei discendenti tramite (u, w) in T_v. In maniera più formale, riutilizzando la definizione di F_u data per il PSF ($F_u(v, w) = \bigvee_{x \in T_v/(u,w)} pred(x)$), si ha:

$$F_u^*(w) = \bigvee_{v \in V} F_u(v, w)$$

La Figura 2.3 mostra una rete in cui sono evidenziati i discendenti dell'arco

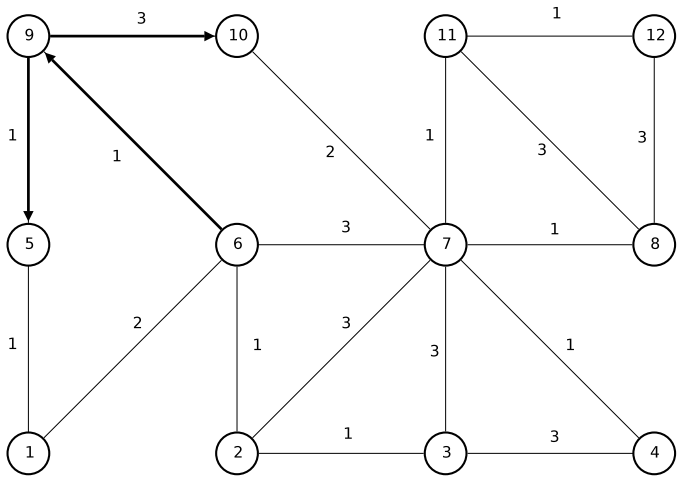


Figura 2.3: Rete di esempio con indicati i discendenti dell'arco $(6, 9)$.

F_6^* :		
prossimo-hop	$\rightarrow$	*predicato*
1	$\rightarrow$	p_1
2	$\rightarrow$	$p_2 \vee p_3$
7	$\rightarrow$	$p_4 \vee p_7 \vee p_8 \vee p_{11} \vee p_{12}$
9	$\rightarrow$	$p_5 \vee p_9 \vee p_{10}$

$T_1/(6,9) = 0$
$T_2/(6,9) = \{5, 9, 10\}$
$T_3/(6,9) = \{5, 9\}$
...
$T_7/(6,9) = \{5, 9\}$
$T_8/(6,9) = \{5, 9\}$
...

Figura 2.4: Funzione F_6^* per la rete d'esempio.

$(6, 9)$ per gli alberi dei cammini minimi di tutti i processor, mentre la Figura 2.4 mostra la funzione di forwarding F_6^* del processor 6, e in particolare di $F_6^*(9)$, elencando i discendenti dell'arco $(6, 9)$ per i vari alberi dei cammini minimi. Si fa notare che l'unione di tali discendenti determina il valore di $F_6^*(9) = p_5 \vee p_9 \vee p_{10}$.

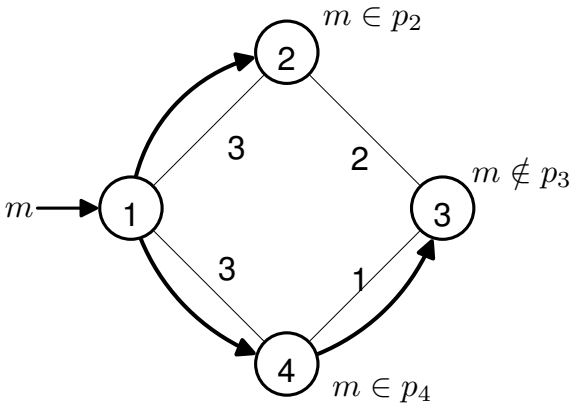
L'intento della funzione F^* è chiaramente quello di inoltrare i messaggi lungo i link associati ai predicati che li selezionano. Tuttavia, se si seguisse solamente la funzione di forwarding definita da F^*, i messaggi inizierebbero ad essere inoltrati lungo dei cicli e continuerebbero ad essere duplicati all'infinito. Infatti, sempre con riferimento alla Figura 2.3, un messaggio immesso nella rete dal processor 6 e selezionato dai predicati p_7 e p_{10}, sarebbe selezionato sia da $F_6^*(9)$ sia da $F_6^*(7)$ e perciò verrebbe inoltrato sia a 9 che a 7. La copia che raggiunge 7 verrebbe quindi inoltrata a 10, perché sarebbe sicuramente selezionata dal predicato $F_7^*(10)$. Allo stesso modo, la copia che arriva a 10 da 9 verrebbe inoltrata a 7 e così via. Per evitare tali cicli quindi, PIF deve inoltrare un messaggio m solo all'interno dell'albero T_v del processor v che ha immesso il messaggio nella rete e per farlo PIF deve memorizzare anche tutti gli alberi dei cammini minimi di tutte le sorgenti. Ciò viene realizzato memorizzando esplicitamente la funzione $child_u(v)$ in ogni processor u per ogni sorgente v.

Avendo definito in questo modo le funzioni F_u^* e $child_u$ possiamo a questo punto specificare le funzioni Init, Hdr e Fwd anche per il PIF:

- $\mathsf{Init}_v(m) = v$, per ogni processor v e per ogni messaggio m
- $\mathsf{Hdr}_u(h) = h$, per ogni processor u e per ogni messaggio m
- $\mathsf{Fwd}_u(v, m) = \{w | w \in child_u(v) \wedge m \in F_u^*(w)\}$, dove v è il processor sorgente estratto dall'header h

Quindi u inoltra un messaggio m generato da v lungo tutti gli archi (u, w) che sono sia (1) discendenti di u in T_v sia (2) associati con un predicato $F_u^*(w)$ che seleziona m.

Si fa qui notare che PIF è uno schema di routing corretto, ma non ideale. Si può facilmente notare come PIF non sia ideale tramite il controesempio mostrato in Fi-



F_4^* :		
...		
3	$\rightarrow$	$p_2 \vee p_3$
...		

Figura 2.5: Consegna dei messaggi non ideale del PIF.

gura 2.5: il messaggio m, generato dal processor 1 viene consegnato correttamente ai processor 2 e 4, che sono interessati ad esso, ma è anche inoltrato, inutilmente, al processor 3, che non è interessato ad esso. PIF spedisce il messaggio da 4 a 3 perché l'arco $(4, 3)$ appartiene all'albero dei cammini minimi T_1, ed è associato al predicato $F_4^*(3) = p_2 \vee p_3$ che seleziona m. $F_4^*(3)$ include p_2 in quanto il processor 2 è un discendente tramite il ramo $(4, 3)$ dell'albero dei cammini minimi T_4 avente radice in 4.

2.2.4 DRP: Per-Receiver Forwarding

L'ultimo schema di routing che andiamo ad analizzare si basa su un'idea differente rispetto a quella degli schemi precedenti, in quanto inoltra i messaggi sulla base dei riceventi. L'idea principale di questo schema di routing è di calcolare l'insieme dei processor interessati ad un messaggio m direttamente all'origine, quando il messaggio viene immesso nella rete. L'insieme delle destinazioni è partizionato in base ai rispettivi next-hop. Questo processo di inoltro è detto *Dynamic Receiver Partitioning*, da cui il nome DRP[3].

I due ingredienti principali dello schema DRP, che vengono memorizzati in ogni processor v, sono la funzione $pred(\cdot)$ e l'informazione di routing unicast necessaria per raggiungere ogni altro processor nella rete. Più nel dettaglio, ogni processor v memorizza una tabella che rappresenta la funzione $pred$ e una ulteriore tabella che rappresenta la funzione $unicast_v : V \rightarrow V$ tale che $unicast_v(u)$ restituisce il vicino di v che si trova sul cammino minimo da v a u.

Nel DRP quindi la funzione Init_v prende un messaggio m, ne calcola l'insieme dei processor interessati $R = \{r | m \in pred(r)\}$ dopodiché, se $R \neq \emptyset$, calcola la funzione

[3]Il DRP a sua volta è una semplificazione del protocollo *DV/DRP (Distance-Vector / Dynamic Receiver Partitioning)*, di cui ulteriori dettagli si possono trovare in [9].

di partizionamento $H_R : N(v) \to R$ che mette in relazione ogni vicino di v con un sottoinsieme di R. H_R definisce una partizione nel senso che $\bigcup_{w \in N(v)} H_R(w) = R$ e $\forall x, y : x \neq y \Rightarrow H_R(x) \cap H_R(y) = 0$. La funzione H_R è fatta in modo tale che $\forall r \in R : unicast(r) = w \Rightarrow r \in H_R(w)$. Intuitivamente H_R partiziona l'insieme dei riceventi R raggrupandoli a seconda del next-hop. L'output della funzione Init_v è un pacchetto $\langle H_R, m \rangle$ che contiene la partizione H_R come header e il messaggio m.

In un processor intermedio u la funzione Hdr_u si comporta quasi allo stesso modo di Init_u. La differenza sta nel fatto che Hdr_u non calcola l'insieme delle destinazioni usando il messaggio m e la funzione $pred(\cdot)$, ma invece estrae la funzione di partizionamento $H_{R'}$ dall'header del pacchetto e la usa per calcolare l'insieme delle destinazioni $R = H_{R'}(u)$. Se $R \neq \emptyset$ allora Hdr_u calcola la funzione H_R e assembla il pacchetto finale esattamente come in Init_u.

Infine, la funzione di forwarding Fwd_u calcola l'insieme dei next-hop esattamente allo stesso modo di come Init_u e Hdr_u calcolano la partizione H_R dell'insieme delle destinazioni R. In particolare, Fwd_u estrae la funzione di partizionamento H_R dall'header dopodiché, se $H_R(u)$ non è definita, allora ciò significa che u è il processor che ha generato il messaggio e perciò Fwd_u restituisce il dominio di H_R; altrimenti, Fwd_u legge $R = H_R(u)$ e restituisce l'insieme $\{x | \exists r \in R \wedge x = unicast(r)\}$.

L'esempio in Figura 2.7, con riferimento alla rete di Figura 2.6, illustra i concetti appena esposti. Nell'esempio il processor 6 riceve un pacchetto $\langle H_{R_1}, m \rangle$ e la funzione Hdr_6 estrae l'insieme delle destinazioni assegnate al processor 6 dalla partizione H_{R_1}, il cui risultato è $R_6 = H_{R_1}(6) = \{2, 3, 4, 8\}$. La funzione $unicast_6$, rappresentata in Figura 2.7, mette in relazione le destinazioni $\{2, 3, 4, 8\}$ con i processor next-hop 2 e 7. In particolare, gli insiemi di destinazioni $\{2, 3\}$ e $\{4, 8\}$ sono

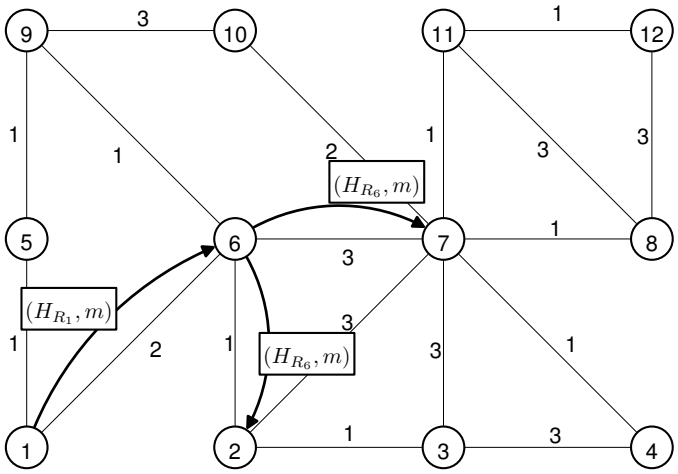


Figura 2.6: Rete d'esempio per il DRP.

$unicast_6$	
dest.	*next-hop*
...	...
2	2
3	2
4	7
...	...
8	7
...	...

$R_1 = \{1, 2, 3, 4, 8, 9\}$
$H_{R_1} = \{\ 5 \rightarrow \{9\}$
$\quad 6 \rightarrow \{2, 3, 4, 8\}\}$

$R_6 = H_{R_1}(6) = \{2, 3, 4, 8\}$
$H_{R_6} = \{\ 2 \rightarrow \{2, 3\}$
$\quad 7 \rightarrow \{4, 8\}\}$

Figura 2.7: Esempio di funzionamento del DRP.

messi in relazione rispettivamente con i next-hop 2 e 7 e questa è la definizione della partizione H_{R_6}. Il processor 6 costruisce quindi un pacchetto $\langle H_{R_6}, m \rangle$ e lo spedisce ai processor 2 e 7.

2.3 Significato della Terminologia Utilizzata

Nei paragrafi precedenti ed in quelli successivi vengono utilizzati termini come "predicato", "vincolo", "sottoscrizione", "filtro" la cui differenza di significato e di impiego è spesso sottile e poco intuitiva. A questo punto della trattazione, per evitare confusioni nel prosieguo, è bene chiarire i concetti ai quali ci si riferisce con la terminologia utilizzata in quanto ognuno dei suddetti termini nel contesto del CBR sottintende delle differenze concettuali di non poco conto ai fini della comprensione.

Nel modello del CBR ogni host esprime il proprio interesse per un certo messaggio per mezzo di una *sottoscrizione* locale. La sottoscrizione tipicamente è espressa utilizzando un linguaggio "logico" del tipo $x > 1 \wedge y = 6$. La sottoscrizione locale è quindi un *filtro*, cioè una congiunzione logica ($\wedge$) di predicati elementari, detti *vincoli*[4] ($x > 1$, $y = 6$). In questo contesto quindi i termini "sottoscrizione" e "filtro" sono sinonimi.

Nel nostro modello ogni host può esprimere più preferenze nei confronti di vari tipi di messaggi, quindi il tipico processor ricevente possiede una disgiunzione logica di filtri. Col termine *predicato* si intende appunto questa disgiunzione di filtri. Siccome poi ogni processor è anche un nodo della rete, le informazioni relative ad ogni processor che circolano sulla rete sono appunto predicati. Ogni processor è caratterizzato quindi da un singolo predicato che riassume tutti i suoi interessi. Nelle varie tabelle di forwarding degli schemi di routing si trovano quindi tali predicati o, meglio, delle disgiunzioni di tali predicati.

[4]Il termine utilizzato nella letteratura è *constraint*.

2.4 Alcune Osservazioni

Tutti e quattro gli schemi di routing content-based visti in questo capitolo sono corretti e tutti tranne uno (PIF) sono anche ideali[5]. Ciò significa che gli schemi di routing PSF, iPS e DRP inoltrano i pacchetti dalla sorgente alla destinazione lungo il cammino a latenza minima e nel farlo utilizzano il minor numero possibile di pacchetti, cioè utilizzano il minor numero di hop possibile compatibilmente con il cammino minimo scelto. Intuitivamente anche PIF si comporta allo stesso modo (in quanto anch'esso segue i cammini minimi dalla sorgente alla destinazione), con la sola differenza che, come si è visto (vedi Figura 2.5), può generare dei falsi positivi, cioè pacchetti che vengono inoltrati a processor che non sono interessati ad essi e che non devono nemmeno inoltrarli ad altri processor, generando così un numero di pacchetti che non è minimo.

Lo scopo del presente lavoro è quello di analizzare gli schemi di routing dal punto di vista delle prestazioni di rete (le misure con cui le prestazioni di rete vengono quantificate saranno esposte successivamente) e pertanto sembra utile iniziare a capire dove possano fare la differenza i quattro schemi presentati. Siccome tutti gli schemi inoltrano i pacchetti dalla sorgente ai destinatari lungo i cammini a latenza minima, sicuramente non è il percorso che segue un messaggio a fare la differenza tra uno schema e l'altro. Allo stesso modo siccome il numero di pacchetti inviati per ogni messaggio è il minimo (compatibilmente con il cammino scelto), nemmeno il numero di pacchetti inviati può fare la differenza. Quindi se è vero che il forwarding viene effettuato dai vari schemi in modo diverso, è anche vero che il risultato, dal punto di vista dei pacchetti che vengono inviati, è praticamente lo stesso. L'unica cosa che distingue uno schema dall'altro sono quindi le funzioni e le strutture dati necessarie ad effettuare il forwarding. Il throughput dei router è infatti determinato dal throughput della forwarding function, che deve essere implementata il più efficientemente possibile. In [4] vengono individuati quattro gruppi di tecniche di ottimizzazione che possono essere applicate per rendere la forwarding function più efficiente:

- algoritmi di *fast matching*;
- rappresentazioni più compatte della tabella di forwarding, in modo che possa essere manipolata più efficientemente dagli algoritmi di fast matching;
- riduzione della dimensione della tabella di forwading;

[5] In [3] ci sono dimostrazioni più o meno formali di quanto detto per tutti e quattro gli schemi.

- riduzione del traffico di pacchetti inutili (cioè falsi positivi) che passano attraverso i router.

I primi due gruppi implementano tecniche di ottimizzazione che sono relative esclusivamente al forwarding e alla sua implementazione, ovvero agli algoritmi di forwarding e di matching. Per algoritmi di forwarding si intendono quegli algoritmi che, dato un pacchetto in input, restituiscono l'insieme dei next-hop di tale pacchetto, in accordo con le regole stabilite dalla forwarding function e le informazioni presenti nella forwarding table. Per svolgere questo compito gli algoritmi di forwarding si appoggiano agli algoritmi di matching, ovvero quegli algoritmi che, dati in input un messaggio ed un predicato di selezione, dicono se il predicato seleziona tale messaggio.

Gli altri due gruppi implementano invece tecniche di ottimizzazione che dipendono dalla strategia di routing utilizzata, ovvero dallo schema di routing. Come detto, tutti gli schemi presentati ad eccezione del PIF minimizzano il numero di falsi positivi, per cui l'unica vera differenza in termini di prestazioni di rete tra i vari schemi è data dalla struttura e dimensione della tabella di forwarding.

Ai fini del presente lavoro quindi non ci occuperemo della problematica del forwarding (e del matching), ma analizzeremo e confronteremo le performance degli schemi di routing a parità di algoritmi di forwarding[6]. Questo approccio permette di confrontare esclusivamente le strategie di routing senza l'influenza di altri fattori.

Facciamo notare che nel presente lavoro non ci occuperemo nemmeno di un'altra problematica del CBR, ovvero i protocolli di routing. Tali protocolli vengono impiegati dagli schemi di routing per far circolare nella rete le informazioni con cui vengono riempite le tabelle di forwarding e definiscono quindi il modo in cui comunicano e si scambiano i predicati i vari nodi della rete[7]. Il nostro lavoro si occupa infatti di valutare il comportamento degli schemi di routing a regime, cioè con le tabelle di forwarding già riempite e in uno scenario di rete "statico", in cui cioè le sottoscrizioni per ogni host siano fisse e non cambino continuamente. Questo approccio è dettato dall'intento di valutare la bontà esclusivamente dello schema di routing e non, come avverrebbe in uno scenario dinamico, della accoppiata protocollo di routing e schema di routing.

[6]L'algoritmo di forwarding di riferimento per il presente lavoro è quello presentato in [5], dove sono presenti anche numerosi riferimenti alla problematica del forwarding e ai vari approcci al problema presenti in letteratura.

[7]Un esempio di protocollo di routing content-based si può trovare in [6], così come alcuni riferimenti alla letteratura sull'argomento.

Capitolo 3

Il Modello delle Simulazioni

In questo capitolo viene descritto il modello concreto sulla base del quale sono state implementate le simulazioni. Tale modello è a sua volta basato sul modello, più astratto, di rete content-based descritto nel capitolo precedente. Nei paragrafi seguenti viene quindi spiegato come sono stati modellati gli elementi più significativi del modello: la rete, i nodi della rete, i messaggi, le sottoscrizioni, i tempi di processing e di generazione dei messaggi. Inoltre si approfondiscono aspetti legati al modo in cui le misure devono essere raccolte, i confronti tra i protocolli ed i criteri statistici di terminazione delle simulazioni.

3.1 Il Modello della Rete

Per modellare una rete content-based bisogna modellare principalmente due elementi: i nodi della rete ed il modo in cui tali nodi sono posizionati e connessi tra di loro, ovvero la topologia della rete.

3.1.1 I Processor

Un nodo della rete può essere visto, coerentemente col modello di rete content-based descritto nel Paragrafo 2.1, come un processor, pertanto ai fini del presente lavoro i termini "nodo" e "processor" saranno usati indistintamente. Qui di seguito vengono elencate le caratteristiche di un processor che sono state modellate:

- un processor implementa le funzioni Init, Hdr, Fwd e memorizza le tabelle di forwarding, per instradare correttamente i pacchetti, in accordo con lo schema di routing utilizzato;

- coerentemente con quanto detto nel Paragrafo 2.1, per non complicare eccessivamente il modello, un processor modella al suo interno il comportamento

di alcuni host, ognuno con le sue applicazioni, pertanto ogni processor ha associato un predicato che riassume tutti gli interessi di tutti gli host che modella. Tipicamente un predicato di un processor sarà composto da una disgiunzione di numerosi filtri (nell'ordine delle centinaia). Inoltre, siccome ogni host può anche essere una sorgente di messaggi, ogni processor genera dei messaggi applicativi e li immette nella rete;

- un processor è collegato ai suoi vicini tramite dei link bidirezionali, aventi banda e latenza definite;

- nel tentativo di modellare il comportamento di un router reale, ogni processor ha una coda di ingresso (la cui dimensione è proporzionale al numero delle sue interfacce di rete) in cui vengono messi i pacchetti in ingresso, così come i pacchetti generati dal processor stesso. Inoltre esso ha una coda in uscita per ogni interfaccia di rete, nella quale vengono accodati i pacchetti che devono essere trasmessi su quella interfaccia;

- in generale, un processor si comporta nel seguente modo: estrae dalla coda dei pacchetti in ingresso un messaggio e lo filtra per vedere se interessa i suoi host e, se è così, il processor "riceve" il messaggio. Dopodiché il processor applica le funzioni Hdr e Fwd al pacchetto inoltrandolo a quei vicini restituiti dalla funzione Fwd. Se il messaggio non va inoltrato a nessuno allora il processor scarta semplicemente il pacchetto;

- le code hanno dimensione finita, pertanto tutti i pacchetti che non riescono ad essere inseriti nelle code, perché sono piene, vengono scartati.

3.1.2 La Topologia

La topologia delle reti da simulare è un elemento critico ai fini del presente lavoro. L'obiettivo di questo lavoro è infatti quello di confrontare gli schemi di CBR nell'ottica di un loro impiego su reti di grandi dimensioni, quali quelle dell'intera Internet. Per questo motivo è necessario realizzare delle topologie che si avvicinino il più possibile alla topologia di Internet, di modo che il comportamento degli schemi di routing sulle reti simulate si avvicini il più possibile a ciò che avverrebbe utilizzando tali schemi su Internet. Ovviamente è impossibile generare manualmente topologie di rete simili a quella di Internet, pertanto per raggiungere tale obiettivo è stato utilizzato `BRITE`[1]. `BRITE` è, a detta dei suoi stessi autori, un gene-

[1] `BRITE`: Boston University Representative Internet Topology Generator è disponibile all' indirizzo Internet: *http://www.cs.bu.edu/brite/*

ratore universale di topologia. In particolare `BRITE` consente di generare topologie di rete simili ad Internet. I motivi per cui è stato preferito `BRITE` ad altri strumenti simili sono molteplici. Di seguito sono elencate le caratteristiche più significative di `BRITE` che lo differenziano, in positivo, dalla maggior parte dei suoi "concorrenti":

- consente di utilizzare diversi modelli di generazione di topologie mentre la maggior parte dei generatori in circolazione si focalizza solamente su uno. L'intento di `BRITE` è appunto quello di riunire in un unico tool i principali modelli di generazione sviluppati dalla comunità dei ricercatori, evitando così di dover utilizzare diversi tool per ogni modello;

- permette di generare diversi tipi di topologie: topologie in cui tutti i nodi sono *Autonomous system* (flat AS), topologie in cui tutti i nodi fanno parte di uno stesso AS (flat router) e topologie gerarchiche in cui i nodi sono AS, ognuno con i suoi nodi, collegati tra loro;

- è user-friendly, in quanto dispone di una comoda interfaccia grafica, attraverso la quale è possibile inserire agevolmente tutti i parametri necessari alla generazione della topologia desiderata;

- è largamente utilizzato dalla comunità scientifica, per cui non è difficile trovare lavori in cui è già stato utilizzato per simulare la topologia di Internet con i relativi parametri di configurazione;

- permette di associare ad ogni link la relativa banda e genera automaticamente le latenze dei link, a seconda della distanza tra i nodi che tale link collega;

- è perfettamente integrato col simulatore utilizzato (`OMNet++`), in quanto tra i vari tipi di file che utilizza per esportare la topologia generata, c'è anche il formato utilizzato da `OMNet++` (file `.ned`).

Ai fini del presente lavoro non è necessario entrare nei dettagli del funzionamento di `BRITE`, pertanto ci limiteremo a descriverne gli aspetti essenziali. Il processo per generare una topologia casuale può essere riassunto nel modo seguente [11]: dati in input il numero N di nodi e la dimensione del piano $n \times m$, inizialmente viene deciso dove posizionare i nodi. I nodi possono essere posizionati secondo una distribuzione uniforme o ammassati in cluster di nodi. Per ogni nodo viene deciso il suo *grado*, ovvero il suo numero di interfacce di rete disponibili ed in definitiva a quanti altri nodi deve essere connesso. Dopodiché si decide quali nodi debbano essere collegati tra loro. La probabilità che venga creato un link tra due nodi può essere uniformemente distribuita o pesata a seconda della distanza euclidea

tra loro. Tale processo continua finché tutti i nodi hanno tutte le loro interfacce collegate ed il grafo risulta connesso.

I vari modelli di generazione differiscono nel modo in cui il processo precedente viene portato a termine. Ad esempio, nel modello *Waxman* [19], il più utilizzato, i nodi vengono posizionati tutti all'inizio uniformemente, dopodiché viene calcolata la probabilità di creare un arco tra due nodi u e v, secondo la funzione di probabilità:

$$P(u, v) = \alpha e^{-d(u,v)/\beta L}$$

dove $d(u, v)$ è la distanza euclidea tra i due nodi, α è il grado medio, L è la distanza euclidea massima tra due nodi qualsiasi della rete e β determina la lunghezza media dell'arco. In altri metodi come *Barabasi Albert* (BA) [1], il processo è più complesso, ad esempio i nodi vengono connessi seguendo una legge che stimi la potenza della crescita di Internet e si possono scegliere numerosi criteri di preferenza per connettere tra loro alcuni tipi di nodi piuttosto che altri.

In questo lavoro le reti simulate sono state generate utilizzando parametri presi da lavori affini e pertanto non è necessario comprendere a fondo le dinamiche di generazione dei modelli né tanto meno approfondire ulteriormente le differenze tra un modello e l'altro. Le reti utilizzate ed i parametri di `BRITE` con cui sono state generate vengono descritti nel Paragrafo 4.4.

3.2 Lo Scenario Applicativo

Per non appesantire troppo la simulazione si è deciso di non implementare una applicazione reale sopra i processor, ma si è modellato lo scenario applicativo tramite i messaggi che vengono generati da ogni nodo ed i predicati associati ad ogni processor. In effetti ciò che interessa, ai fini delle simulazioni, è di simulare, all'interno della rete, un flusso di messaggi tipici di una applicazione reale e per fare ciò è sufficiente agire sia sui messaggi che vengono generati sia sui predicati associati ad ogni processor, che riassumono tutte le sottoscrizioni di tutti gli host "collegati" ad esso. Nei paragrafi seguenti vengono descritti i modelli dei messaggi e delle sottoscrizioni e quali sono i parametri sui quali si può intervenire per ottenere il flusso di messaggi desiderato.

3.2.1 I Messaggi e le Sottoscrizioni

Per semplificare la simulazione, senza pregiudicare i risultati dell'analisi, è stato deciso di modellare i messaggi e le sottoscrizioni come numeri interi. Pertanto

una sottoscrizione di contenuto α (con α numero intero) seleziona i messaggi di contenuto α. I predicati associati ad ogni processor, essendo una disgiunzione di sottoscrizioni, possono essere modellati come un insieme di interi (concettualmente una disgiunzione di interi). Pertanto un predicato p seleziona un messaggio m se $m \in p$.

Tale modello potrebbe sembrare troppo semplicistico, ma si ricorda che lo scopo del presente lavoro non è di valutare differenti algoritmi di matching, ma solamente le prestazioni di rete degli schemi di routing[2], pertanto ciò che importa è di creare un certo tipo di traffico e per farlo il modello di messaggi e sottoscrizioni adottato è risultato sufficiente.

Si fa qui notare che durante tutta l'esposizione, coerentemente con quanto detto nel Paragrafo 2.1, il termine "messaggio" viene usato per indicare un messaggio applicativo (cioè quello illustrato in questo paragrafo), mentre per indicare il messaggio che viaggia sulla rete, viene utilizzato il termine "pacchetto", inteso come l'insieme del messaggio e dell'header.

3.2.2 La generazione dei Messaggi e dei Predicati

Come visto, ogni nodo della rete deve possedere il suo predicato nella forma di un insieme di interi e ogni nodo della rete deve generare dei messaggi. Ovviamente la simulazione deve generare sia i messaggi sia le sottoscrizioni in maniera casuale, secondo una certa distribuzione di probabilità. Ai fini di questo lavoro i messaggi vengono generati secondo un distribuzione *uniforme* di probabilità, mentre le sottoscrizioni vengono generate secondo una distribuzione *zipf*.

La distribuzione zipf è una distribuzione di probabilità discreta che deriva dalla legge di Zipf, una legge empirica inizialmente adoperata per descrivere le occorrenze delle parole in un testo. Secondo la legge di Zipf, in un generico testo la parola che si trova più di frequente ha frequenza doppia rispetto alla seconda parola più frequente, tripla rispetto alla terza parola più frequente e così via. Dalla legge di Zipf è stata ricavata una distribuzione, la cui funzione di distribuzione è data da:

$$f(k, \alpha, N) = \frac{1/k^{\alpha}}{\sum_{n=1}^{N} 1/n^{\alpha}}$$

dove k è la posizione in base alla frequenza (1 l'elemento più frequente, 2 il secondo più frequente etc...), N è il numero di elementi e α è un esponente che caratterizza

[2]Il modo in cui è stato modellato il tempo di matching viene spiegato nel Paragrafo 3.3.2.

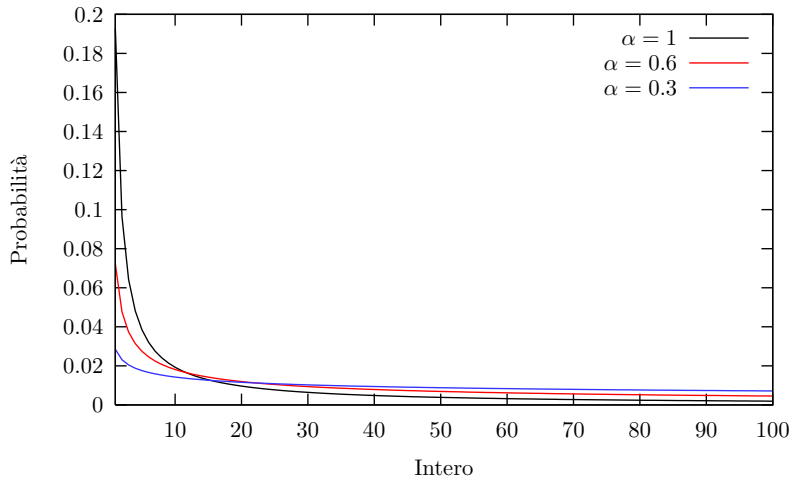


Figura 3.1: Distribuzione zipf con $N = 100$.

la distribuzione. La Figura 3.1 mostra la funzione di distribuzione[3] zipf per 100 elementi interi e l'effetto della variazione dell'esponente α. La distribuzione zipf può essere usata per modellare tutti quei fenomeni in cui alcuni eventi occorrono con probabilità maggiore mentre altri eventi occorrono con probabilità bassa, ma non nulla. Inoltre diminuendo l'esponente α si alza la probabilità degli eventi meno probabili a discapito di quelli più probabili. La distribuzione viene comunemente utilizzata per modellare fenomeni come la frequenza degli accessi alle pagine Internet, la dimensione di città e centri abitati, la distribuzione dei redditi, l'occorrenza delle note in spartiti musicali.

Più in generale, la distribuzione zipf è utilizzata per esprimere la distribuzione degli interessi di un utente. Ai fini del presente lavoro sembra quindi corretto generare i messaggi secondo una distribuzione uniforme nell'intervallo $[1, x]$, mentre gli interi che compongono ogni predicato vengono presi da una zipf nell'intervallo $[1, x]$, con esponente α in $(0, 1]$. Per comprendere la ragionevolezza di tale scelta, si pensi ad una applicazione di monitoraggio delle quotazioni azionarie: gli eventi generati sono uniformi, in quanto vengono generati eventi per tutti i titoli quotati, sia per piccole variazioni degli indici che per grandi variazioni, tuttavia vi saranno sottoscrizioni più probabili, ad esempio quelle che interessano variazioni degli indici maggiori di una certa soglia oppure i titoli più importanti e sottoscrizioni meno probabili in quanto meno interessanti.

[3]In realtà la funzione è discreta, ma è stata comunque disegnata continua, per mostrarne meglio l'andamento.

3.2.3 Simulazione di Diversi Scenari Applicativi

A questo punto risulta utile capire come differenziare diversi scenari applicativi e, in ultima analisi, diversi flussi di messaggi nella rete. In generale, i vari scenari applicativi differiscono per la percentuale media di nodi della rete che ricevono ogni messaggio generato. Ad esempio, la Figura 3.2 mostra due differenti scenari applicativi: in uno un messaggio generato ha probabilità maggiore di arrivare al 25% dei riceventi, ma anche probabilità non trascurabili di raggiungere tra il 10% e 40% di riceventi; nell'altra un messaggio generato ha probabilità maggiore di raggiungere il 3% di riceventi, ma esiste anche un'alta probabilità che un messaggio generato non interessi nessun ricevente, così come è nulla la probabilità che un messaggio arrivi a più del 30% dei riceventi. In questo modo, variando la forma e la media di tali distribuzioni si possono simulare differenti scenari applicativi. I parametri su cui si può intervenire, a tale scopo, sono tre:

- l'intervallo di generazione dei messaggi e delle sottoscrizioni, ovvero la larghezza della distribuzione uniforme e della zipf [4];
- il coefficiente α della distribuzione zipf delle sottoscrizioni;
- la dimensione dei predicati associati ad ogni processor, ovvero il numero di sottoscrizioni per predicato.

Intuitivamente infatti, allargando l'intervallo della distribuzione zipf e della generazione dei messaggi, a parità di α e di numero di sottoscrizioni, un determinato

[4] L'intervallo delle due distribuzioni è il medesimo, in quanto non ha senso generare messaggi a cui sicuramente nessuno è interessato o sottoscrizioni per messaggi che non vengono mai generati, meglio lasciare che tali situazioni accadano per ragioni del tutto casuali e non perché imposte a priori.

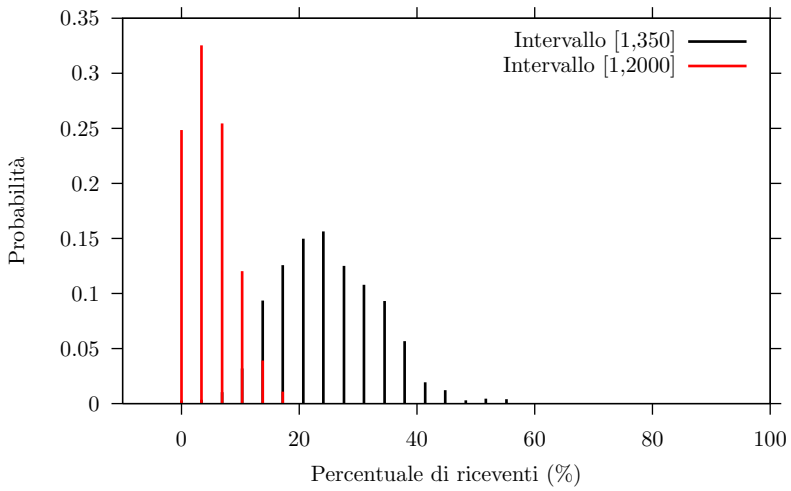


Figura 3.2: Esempi di differenti scenari applicativi.

messaggio generato corrisponderà ad un minor numero di interessi. Con riferimento ancora alla Figura 3.2 si nota infatti come, allargando l'intervallo, la percentuale dei riceventi interessati diminuisca. Allo stesso modo, diminuendo il numero di sottoscrizioni che compongono il predicato di ogni nodo, a parità di α e di intervallo, la percentuale dei riceventi diminuisce, in quanto è maggiore la probabilità che le sottoscrizioni siano diverse tra loro. Ciò accade proprio perché le sottoscrizioni sono generate secondo una distribuzione zipf ed è quindi più alta la probabilità che, aumentando il numero di sottoscrizioni per nodo, ogni nodo contenga le sottoscrizioni più probabili per quella specifica distribuzione zipf. Inoltre diminuendo l'esponente α della zipf, si otterranno curve di probabilità della percentuale di riceventi di un messaggio più piatte, ovvero in cui il picco è meno pronunciato.

Pertanto intervenendo sui tre suddetti parametri si possono generare delle distribuzioni di percentuali dei riceventi di un messaggio adeguate al tipo di scenario applicativo che si vuole simulare.

3.3 Il Modello dei Tempi

Essendo il presente lavoro orientato al confronto degli schemi di CBR dalla prospettiva delle prestazioni di rete, il modo in cui vengono modellati i tempi risulta cruciale. I tempi che si è avuto la necessità di modellare sono di due tipi: i tempi di generazione dei messaggi ed i tempi di *processing* dei nodi. L'importanza di questi due tipi di tempi è evidente: un tasso di generazione dei messaggi elevato crea un maggior traffico sulla rete, così come il tempo di processing influenza la velocità con cui i nodi smaltiscono i messaggi che generano e quelli che arrivano in ingresso.

3.3.1 I Tempi di Generazione dei Messaggi

Per tempo di generazione dei messaggi, si intende il tasso al quale vengono generati i messaggi, cioè quanti messaggi al secondo genera ogni nodo o, simmetricamente, ogni quanti secondi ogni processor genera un nuovo messaggio. Con riferimento a questa ultima definizione, i tempi di generazione dei messaggi sono stati modellati secondo una distribuzione di probabilità esponenziale. La distribuzione di probabilità esponenziale è infatti la distribuzione più naturale quando si tratta di modellare il tempo che intercorre tra due eventi indipendenti che avvengono ad un tasso mediamente costante β. Per evitare ambiguità su quale sia la distribuzione di riferimento (il parametro β è spesso indicato come $\lambda = 1/\beta$) essa viene riportata

di seguito:

$$f(x,\beta) = \begin{cases} \frac{1}{\beta}e^{-x/\beta} & , \quad x \geq 0 \\ 0 & , \quad x < 0 \end{cases}$$

D'ora in avanti, per comodità, si farà riferimento solamente al tasso di generazione medio β, sottintendendo il fatto che il rate effettivamente utilizzato nelle simulazioni proviene da una distribuzione esponenziale di parametro β.

Si fa notare inoltre che nelle simulazioni presentate si suppone che tutti i nodi della rete generino, in ogni scenario, i messaggi tutti allo stesso tasso medio β. Ciò è necessario per tenere in qualche modo sotto controllo il traffico generato, aumentando o diminuendo tale tasso a seconda delle necessità, cosa che sarebbe impossibile se tutti i nodi generassero messaggi a tassi differenti.

3.3.2 I Tempi di Processing

Per tempo di processing si intende il tempo necessario ad ogni processor per eseguire la forwarding function Fwd, ovvero il tempo necessario per decidere per ogni messaggio in arrivo a quali nodi vicini inoltrarlo, basandosi sulle informazioni contenute nelle tabelle di forwarding. All'interno di tale definizione si fa ricadere anche il tempo di filtering, ovvero il tempo necessario ad un processor per decidere se un pacchetto in ingresso va inoltrato al livello applicativo, cioè il tempo necessario per verificare se il suo predicato seleziona tale messaggio.

Il tempo in questione non può essere preso direttamente dalla simulazione, in quanto, come visto nei paragrafi precedenti, il modello dei messaggi e delle sottoscrizioni è molto semplificato rispetto alla realtà. Pertanto, è stato deciso di modellare il tempo di processing sulla base di alcuni risultati presenti in letteratura. L'articolo preso come riferimento è stato [5]: in questo articolo infatti viene proposto un algoritmo di forwarding, molto evoluto e complesso, e ne viene fatta una analisi delle performance. In particolare viene mostrato un grafico in cui vengono espressi i tempi di matching al variare del numero totale di vincoli presenti nella tabella di forwarding ed al variare del numero di interfacce presenti su un processor; tale grafico viene riportato in Figura 3.3.

Per modellare il tempo di matching si è quindi deciso di partire da tale grafico e di ricavarne un modello del tempo di processing. Per raggiungere lo scopo, dapprima si è interpolata la funzione del grafico, con un metodo di fitting, in modo da ottenere una funzione continua avente come variabili indipendenti il numero di vincoli e il numero delle interfacce, dopodiché si è proceduto a scalare la funzione ottenuta di un fattore 10. Tale riduzione dei tempi di matching, rispetto ai tempi mostrati nel grafico originale, è inclusiva sia dei miglioramenti apportati all'algorit-

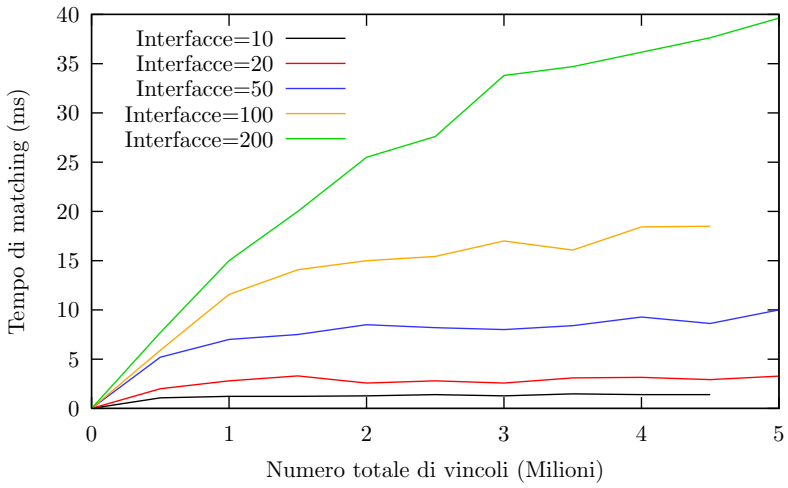


Figura 3.3: Grafico del tempo di matching così come in [5].

mo di matching negli ultimi anni sia della superiore velocità dei processori e delle memorie (le performance mostrate nel grafico originale sono infatti relative ad un processore Intel Pentium 3 (single core) con frequenza di clock di 950 Mhz). La funzione risultante è mostrata in Figura 3.4[5] ed è così definita:

$$f(v, i) = ai^{1.8} \log(bi^{-2.6}v + 1)$$

dove v è la variabile relativa al numero totale di vincoli, i quella relativa al numero di interfacce e le costanti a e b sono state ricavate applicando il metodo di fitting.

Passiamo ora a descrivere come è stato modellato il tempo di processing nelle simulazioni. Per brevità nel prosieguo chiameremo col nome *processing* la funzione

[5]La figura mostra la funzione solamente per lo stesso numero di interfacce del grafico originale, per renderne più agevole il confronto.

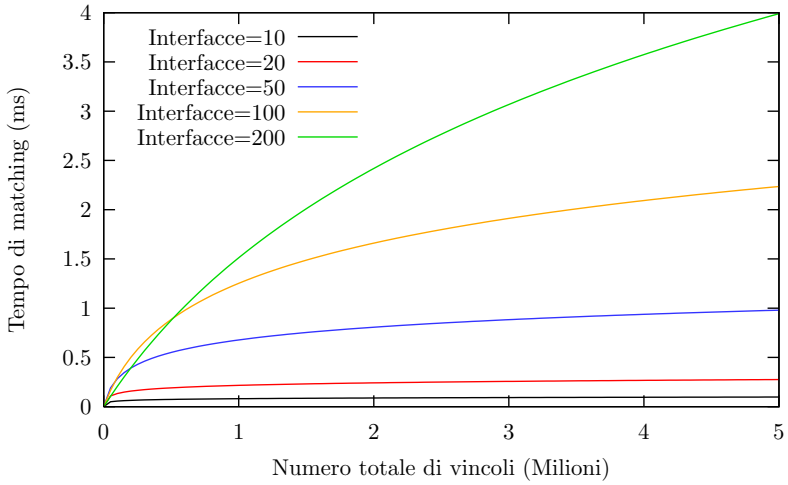


Figura 3.4: Grafico del tempo di matching utilizzato.

interpolata e scalata descritta precedentemente. Una delle due variabili di tale funzione corrisponde al numero totale di vincoli, ma nel nostro modello, come spiegato nel Paragrafo 3.2.1, esiste solamente il concetto di filtro, pertanto per rendere realisticamente più pesanti le sottoscrizioni, si è deciso di scegliere un numero η, uguale per tutti i filtri, che indica il numero di vincoli di cui è composto un singolo filtro.

Non appena arriva un messaggio, il processor dapprima esegue la fase di filtering, per verificare se il suo predicato seleziona il messaggio in ingresso. Il tempo necessario per eseguire il filtering è calcolato dalla funzione *processing* ponendo il numero di interfacce pari ad 1 ed il numero totale di vincoli pari al numero di filtri (cioè di interi) presenti nel predicato associato al processor, moltiplicati per il numero di vincoli di cui è composta una sottoscrizione. Dopodiché il processor esegue la fase di forwarding, che avviene sia per i messaggi generati dal processor stesso che per quelli in entrata. Essendo il processo di forwarding differente per ogni schema di routing, anche il tempo necessario per eseguire il forwarding viene calcolato in maniera dipendente dallo schema di routing, nel seguente modo:

- per il PSF e l'iPS, il tempo di forwarding viene calcolato direttamente dalla funzione *processing*, ponendo come numero di interfacce quelle presenti nella entry della tabella di forwarding corrispondente alla sorgente che ha inviato il messaggio e come numero totale di vincoli la somma di tutti i vincoli di tutti i filtri dei predicati associati alle interfacce individuate precedentemente. In tale modello, il tempo per scegliere la entry della tabella di forwarding corrispondente alla sorgente che ha inviato il messaggio si assume essere nullo, in quanto trascurabile rispetto al tempo di matching;

- per il PIF, il tempo di forwarding viene calcolato direttamente dalla funzione *processing*, ponendo come numero di interfacce quelle presenti nella entry della tabella *child* corrispondente alla sorgente che ha inviato il messaggio e come numero totale di vincoli la somma di tutti i vincoli di tutti i filtri dei predicati associati alle interfacce individuate precedentemente, presenti nella tabella di forwarding. Anche in questo caso il tempo necessario per scegliere la entry corrispondente alla sorgente del messaggio nella tabella *child* e quello necessario per scegliere le entry corrispondenti alle interfacce nella tabella di forwarding si considerano nulli in quanto trascurabili rispetto agli altri tempi in gioco.

- per il DRP, il tempo di forwarding viene calcolato direttamente dalla funzione *processing* solo per i pacchetti che sono stati generati dal processor stesso,

ponendo come numero di interfacce il numero di nodi presenti nella tabella *pred*, contenente i predicati di tutti i nodi, e come numero totale di vincoli la somma di tutti i vincoli di tutti i filtri di tali predicati. Per i messaggi che non sono generati dal processor stesso, e per i quali bisogna calcolare solamente la nuova partizione a partire da quella presente nell'header del pacchetto, si è scelto di modellare tale tempo con un piccolo tempo costante di due ordini di grandezza inferiore alla latenza dei link. Ciò perché il calcolo della nuova partizione non è una operazione computazionalmente così pesante quanto lo è la fase di matching dei predicati.

Come conseguenza del modello dei tempi adottato, i tempi di processing dei protocolli PSF e iPS sono gli stessi, in quanto tali tempi sono indipendenti dal numero di sorgenti presenti nella tabella di forwardig. Pertanto tali protocolli avranno le stesse prestazioni di rete. Si fa notare che questa assunzione è realistica, in quanto il minor tempo speso dall'iPS per scegliere la entry corretta nella tabella di forwarding, viene controbilanciato dal tempo necessario a stabilire la classe di equivalenza della sorgente del pacchetto da inoltrare e, anche se tali tempi non sono esattamente uguali, l'entità della loro differenza è senza dubbio trascurabile rispetto ai tempi di matching.

Un'altra conseguenza del modello dei tempi adottato è che i tempi di processing per i messaggi generati dal processor stesso sono sempre costanti, per quel processor, mentre per i messaggi in ingresso il tempo di processing dipende dalla sorgente del messaggio per tutti gli schemi, tranne che per il DRP.

3.4 Gli Schemi di Routing Confrontati

Alla luce di quanto detto finora, per inquadrare meglio il modello delle simulazioni, è utile trattare quali sono gli schemi di routing che si sono confrontati. Se inizialmente, infatti, i protocolli da confrontare dovevano essere i quattro descritti nel capitolo precedente, in fase di simulazione si è deciso di operare dei confronti differenti.

Innanzitutto, come si è visto nel paragrafo precedente, le prestazioni di rete dei protocolli PSF e iPS sono identiche, di conseguenza si è deciso di eliminare lo schema iPS dai confronti sulle prestazioni di rete. Esso è stato simulato solo per operare dei confronti sulla occupazione di memoria, unico aspetto che lo differenzia dal PSF.

Inoltre si è deciso di verificare come si comportano gli schemi di routing su una topologia ad albero (topologia spesso utilizzata per creare delle overlay network).

Per farlo si è deciso di operare nel modo seguente: per ogni rete e per ogni scenario, dapprima si confrontano gli schemi di routing PSF, PIF e DRP. Dopodiché, si estrae dalla rete utilizzata per tali confronti il minore albero ricoprente[6] e si simula su tale albero un protocollo di *subscription forwarding*. Il protocollo di subscription forwarding, che nel seguito verrà indicato come *SFwd*, è concettualmente identico al PIF, con la differenza che, essendo la topologia aciclica, esso non necessita della tabella *child*.

Infine si è deciso di simulare, sul minimo albero ricoprente della rete, un semplice protocollo di flooding su albero, al quale nel prosieguo si farà riferimento col nome di *Flood*. Il protocollo Flood è estremamente semplice: ogni processor, quando genera un messaggio, lo immette nella rete inviandolo su tutte le sue interfacce e ogni processor che riceve un pacchetto in ingresso lo inoltra su tutte le sue interfacce, tranne quella da cui ha ricevuto il pacchetto. Ovviamente tale protocollo funziona solamente su una topologia aciclica. Il protocollo Flood è stato scelto appositamente per la sua semplicità, per poter confrontare gli schemi di routing content-based, caratterizzati da un tempo di processing consistente e dalla minimizzazione del traffico di rete, con un protocollo diametralmente opposto, ovvero caratterizzato da un tempo di processing trascurabile e dalla generazione del massimo traffico di rete possibile.

Per quanto riguarda il modello dei tempi, si assume che il tempo impiegato dal protocollo Flood per inoltrare i messaggi sia nullo, in quanto il processor non deve processare i messaggi ma semplicemente inviarli sulle interfacce di rete, mentre il tempo di filtering viene modellato come per gli altri processor.

3.5 Le Misure Raccolte

Stabilito il modello delle simulazioni, risulta ora necessario definire le misure che vengono raccolte durante le simulazioni stesse. Le misure raccolte possono concettualmente essere classificate in misure "principali", ovvero quelle misure necessarie per misurare le prestazioni di rete di uno schema di routing e per capire quale sia lo schema di routing migliore, e misure "ausiliarie", ovvero misure che di per se stesse non misurano la performance di rete di uno schema di routing, ma che possono essere utili sia per capire cosa sta succedendo nella rete durante la simulazione sia per operare confronti su aspetti più marginali degli schemi di routing in questione oppure per evidenziare le differenze tra i vari scenari applicativi, o di rete, simulati.

[6] Per minore albero ricoprente si intende il *minimum spanning tree*, che nel lavoro in questione è stato calcolato con l'algoritmo di *Kruskal*.

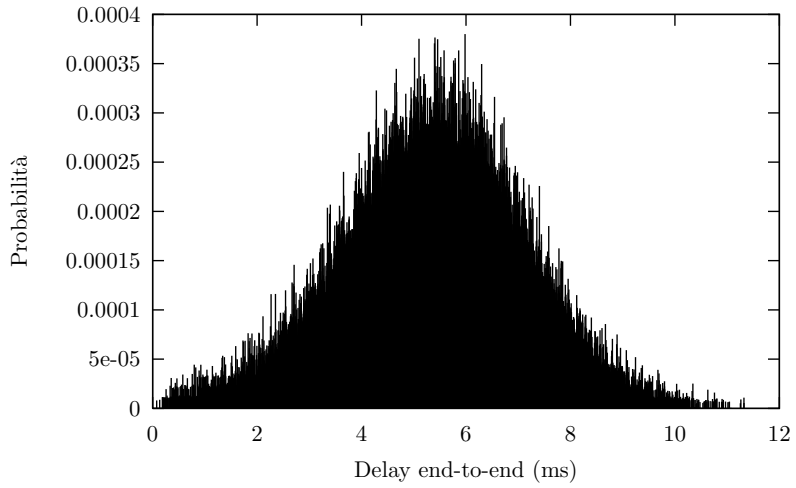


Figura 3.5: Esempio di distribuzione dei delay end-to-end.

3.5.1 Le Misure delle Prestazioni di Rete

Innanzitutto occorre definire cosa si intende per prestazioni di rete. Per definire le prestazioni di rete di uno schema di routing si fa riferimento alla definizione di efficienza di uno schema di routing data nel Paragrafo 2.1.3. Per essere efficiente uno schema di routing deve quindi consegnare un messaggio ai destinatari interessati il più velocemente possibile, e nel farlo deve generare il minor traffico di rete possibile. Pertanto le misure utilizzate per misurare la prestazioni di rete di uno schema di routing sono le seguenti:

delay end-to-end la prima misura, nonché la più importante, utilizzata per misurare la performance di rete di uno schema di routing content-based è quella dei *delay end-to-end*. Per delay end-to-end si intende il tempo che intercorre tra la generazione di un messaggio e la sua ricezione dall'host interessato (quindi dopo che è avvenuto il processo di filtering). Tipicamente quindi, per ogni messaggio generato, verranno raccolte le misure di tanti delay quanti sono i riceventi interessati a tale messaggio. Ai fini dei nostri confronti i delay end-to-end raccolti vengono utilizzati per calcolare, e disegnare, una distribuzione dei delay, ovvero dato un messaggio generato qual'è la probabilità che esso venga consegnato con un determinato delay, e per calcolare delle misure statistiche come il delay minimo, massimo, la media, la mediana e la moda dei delay. La distribuzione dei delay, di cui un tipico esempio si può vedere in Figura 3.5, serve più che altro ad un confronto visivo, mentre i valori statistici sono più utili per operare un confronto quantitativo. La media dei delay end-to-end, in particolare, quando calcolata su un numero signifi-

cativo di campioni[7], fornisce un valore fondamentale per poter confrontare gli schemi di routing a parità di rete e di scenario applicativo. Una media dei delay end-to-end minore è infatti sinonimo di uno schema di routing più efficace, in quanto tale schema riesce, mediamente, a consegnare i messaggi ai riceventi interessati più velocemente. Questa misura sarà quindi la misura più importante ai fini dei confronti operati, essendo anche la più interessante dal punto di vista degli utilizzatori del servizio di rete content-based.

pacchetti per messaggio generato un'altra misura di notevole interesse è quella dei *pacchetti per messaggio generato*, alla quale d'ora in avanti ci si riferirà col nome di PPM. Per PPM, si intende il numero totale di link attraversati da un messaggio nel suo percorso dalla sorgente a tutti i riceventi interessati. Tale misura coincide quindi col numero totale di pacchetti inviati sulla rete per ogni messaggio generato[8]. A differenza del caso precedente, in questo caso viene raccolta una misura di PPM per ogni messaggio generato. Anche in questo caso i dati raccolti vengono poi utilizzati per tracciarne una distribuzione e per calcolarne valore minimo e massimo, media, mediana e moda. La Figura 3.6 mostra la distribuzione dei PPM di due schemi di routing nello stesso scenario di rete e applicativo. Dalla figura si evince chiaramente come lo schema 1 sia più efficiente dello schema 2, in quanto quest'ultimo genera quasi il doppio dei pacchetti per ogni messaggio generato. Anche in questo caso quindi la media dei PPM aiuta a quantificare quanto uno schema sia

[7] Cosa si intende per numero significativo di campioni, viene spiegato nel Paragrafo 3.6.

[8] Nel nostro modello la dimensione del messaggio viene sempre considerata inferiore alla dimensione massima consentita dal livello di rete, pertanto un messaggio viaggia sempre sulla rete incapsulato in un solo pacchetto, senza dar luogo a frammentazione.

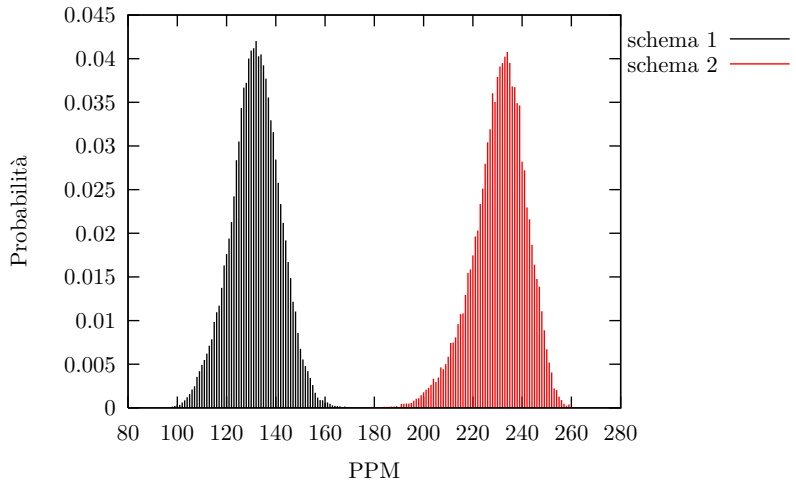


Figura 3.6: Esempio di confronto tra distribuzioni di PPM.

migliore di un altro in termini di traffico generato. Come nota finale si fa notare che, per quanto visto nel Paragrafo 2.4, le distribuzioni del traffico generato per gli schemi PSF, iPS e DRP coincidono, in quanto questi schemi generano tutti il traffico minimo, compatibilmente con la definizione di traffico minimo data nel suddetto paragrafo[9].

throughput della rete per throughput della rete si intende il numero totale di messaggi ricevuti in un secondo da tutti i destinatari interessati su tutta la rete. Il throughput di rete viene calcolato contando, in un secondo, per tutti i nodi della rete, quanti sono i messaggi che vengono ricevuti, cioè i messaggi selezionati dal predicato associato al nodo. Tale misura ci da un'idea di quanti messaggi riesce a consegnare globalmente il servizio di rete. Ciò che più interessa di questa misura, ai fini dei confronti tra i protocolli, non è la misura in se, ma come essa varia al variare del tasso di generazione dei messaggi. Per brevità chiameremo nel seguito tale misura *curva di throughput*. La Figura 3.7 mostra il confronto della curva di throughput tra due schemi di routing a parità di scenario applicativo e di rete. Dalla figura si nota chiaramente come lo schema 2 sia nettamente inferiore in termini di throughput. Si nota anche che, ad esempio, il throughput massimo dello schema 2 si aggira attorno ai 700mila messaggi al secondo, mentre quello dello schema 1 è destinato a superare 1,5 milioni. La curva di throughput fornisce numerose informazioni riguardo agli schemi di routing. Innanzitutto descrive come gli schemi reagiscono quando il traffico aumenta a dismisura e come si comportano quando vengono "stressati". Sempre con riferimento alla figura si vede

[9]Non è infatti detto che il traffico minimo, per la definizione data, sia il minor traffico possibile.

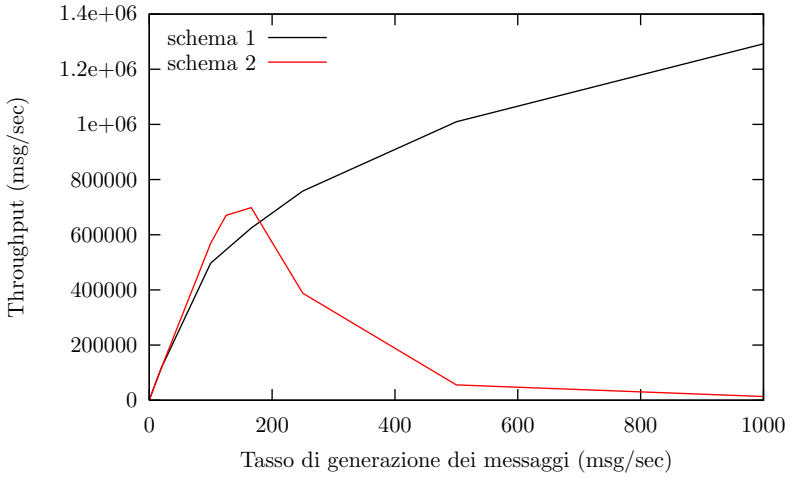


Figura 3.7: Esempio di confronto tra curve di throughput.

come lo schema 2 sia degenere da questo punto di vista, in quanto aumentando il traffico esso inizia a perdere tutti i pacchetti, finché non riesce più a consegnarne nessuno. Lo schema 1 è invece più stabile, in quanto nonostante il traffico aumenti riesce ad aumentare il numero di messaggi consegnati. La curva di throughput rivela anche un'altra informazione importante, ovvero il punto in cui la rete inizia a perdere pacchetti, che coincide con il punto in cui la retta che parte dall'origine inizia a flettere. Da questo punto di vista lo schema 2 è migliore, in quanto garantisce un throughput senza perdita di pacchetti maggiore.

3.5.2 Le Misure Ausiliarie

Come detto precedentemente, le misure ausiliarie vengono raccolte per capire meglio cosa avviene nelle reti simulate. Lo scopo delle misure ausiliarie è quindi, da una parte quello di aiutare a comprendere i motivi per cui, eventualmente, gli schemi differiscono per le loro prestazioni di rete e dall'altra quello di fornire ulteriori dettagli per analizzare meglio lo scenario applicativo e di rete simulati. Alcune di queste misure sono inoltre ridondanti, in modo da poter verificare che le misure raccolte siano coerenti tra loro. Le misure ausiliarie sono elencate qui di seguito.

Conteggi Globali

Queste misure servono per operare dei confronti quantitativi sul traffico generato dai vari schemi di routing e dai vari scenari.

numero di messaggi generati è il numero totale di messaggi generati da tutti i nodi della rete, durante tutta la durata della simulazione.

numero di messaggi scartati all'origine tutti gli schemi, ad eccezione del protocollo Flood, scartano direttamente all'origine i messaggi che non interessano nessun nodo della rete. Questa quantità esprime appunto il numero totale di tutti i messaggi scartati all'origine da tutti i nodi della rete, durante tutta la durata della simulazione. Tale misura è utile per capire lo scenario applicativo: un numero elevato di tali messaggi indica uno scenario in cui gli interessi dei nodi sono molto pochi rispetto alla varietà dei messaggi generati. La differenza tra questa misura e quella precedente da inoltre un'idea di quanti siano i messaggi effettivi che hanno causato il traffico sulla rete.

traffico totale generato è il numero totale di tutti i pacchetti scambiati sulla rete, durante tutta la durata della simulazione. Tale numero serve soprattutto per quantificare il traffico generato da uno schema in un determinato scenario.

numero di messaggi ricevuti indica il numero totale di tutti i messaggi ricevuti (cioè selezionati dal predicato associato al nodo) da ogni nodo, durante tutta la durata della simulazione.

numero di messaggi consegnati indica il numero totale di messaggi generati che sono stati correttamente ricevuti da tutti i destinatari interessati, per tutti i nodi della rete, per tutta la durata della simulazione. In pratica, ogni volta che un messaggio generato viene ricevuto da tutti i nodi interessati tale quantità viene incrementata di uno. La differenza tra il numero di messaggi generati non scartati all'origine e questa quantità, da il numero di messaggi generati che non sono stati correttamente consegnati a tutti i destinatari interessati (quindi anche quei messaggi che sono stati consegnati solo ad una parte del totale dei riceventi interessati).

numero di pacchetti persi sulla code in entrata indica il numero totale dei pacchetti persi da tutti i processor della rete, per tutta la durata della simulazione, in quanto la coda dei pacchetti in ingresso era piena. Questa misura da un'idea della "lentezza" di un processor nello smaltire il traffico entrante: se il traffico entrante è troppo per la capacità di processing del processor, esso inizia a perdere i pacchetti in ingresso o quelli generati dal processor stesso.

numero di pacchetti persi sulle code in uscita misura analoga alla precedente, ma per le code in uscita. Questa misura da invece un'idea di quanto il traffico circolante sulla rete sia superiore alla capacità della rete stessa di smaltirlo: un traffico elevato, infatti, causerà il riempimento delle code associate alle interfacce in uscita (in attesa che i link in uscita siano liberi per poter trasmettere i pacchetti) e, pertanto, se il numero di pacchetti inviati sui link da un processor è maggiore della capacità dei link stessi, tali code perdono pacchetti.

numero di messaggi persi all'origine indica il numero totale, per tutti i nodi della rete, per tutta la durata della simulazione, dei messaggi che sono stati persi prima ancora di essere immessi nella rete, in quanto la coda in ingresso dei processor era piena. Facendo la differenza tra i pacchetti persi sulle code in entrata e questa quantità, si può capire quanti dei pacchetti persi siano quelli provenienti dai link in entrata e quanti dai messaggi generati.

numero di falsi positivi indica il numero totale, per tutti nodi della rete, per

tutta la durata della simulazione, dei pacchetti giunti ad un processor senza che questo debba né inoltrarli ad altri nodi né sia interessato al loro contenuto.

Misure Relative allo Scenario Applicativo

Queste misure servono per operare dei confronti sugli scenari applicativi simulati.

percentuale di riceventi per ogni messaggio generato da ogni nodo della rete, per tutta la durata della simulazione, viene annotata la percentuale di riceventi interessati a tale messaggio. In questo modo si può calcolare una distribuzione della percentuale di riceventi, come illustrato nel Paragrafo 3.2.3.

Dimensioni della Rete

Queste misure servono per comprendere meglio lo scenario di rete della simulazione.

grado dei nodi il grado, o fanout, di un nodo corrisponde al numero di interfacce che possiede. Per ogni rete simulata, per ogni nodo della rete, ne viene calcolato il grado. Si può così calcolare la distribuzione dei gradi e ricavare misure statistiche come il grado minimo, massimo e medio della rete.

diametro della rete il diametro di una rete corrisponde alla lunghezza del cammino minimo di lunghezza massima tra tutte le coppie di nodi della rete. Quindi per ogni coppia di nodi della rete viene calcolato il cammino a latenza minima e, di tutti questi cammini, viene scelto quello costituito da più hop. Il numero di hop di tale cammino è il diametro della rete.

Occupazione di Memoria delle Tabelle di Forwarding

Queste misure servono per dare un'idea della occupazione di memoria, locale ad ogni processor, delle strutture dati necessarie ai vari schemi di routing per calcolare la funzione di forwarding.

numero di nodi tutti i processor degli schemi che annotano l'indicazione del nodo sorgente di un messaggio (PSF, iPS e PIF) hanno nelle tabelle di forwarding una entry per i possibili nodi sorgente. Per questi tipi di processor, questa misura si riferisce al numero di nodi sorgente presenti in tabella, misurato per ogni processor della rete. Per il DRP, invece, tale numero si riferisce al numero di nodi presenti nella tabella *pred*.

numero di interfacce tutti gli schemi che associano un predicato alla interfacce (PSF, iPS, PIF), hanno per tali interfacce delle entry nelle tabelle di forwarding. Questo numero si riferisce al numero totale di interfacce presenti nelle tabelle di forwarding, misurato per ogni processor della rete.

numero di filtri questa misura indica il numero totale di filtri presenti nelle tabelle di forwarding, misurato per ogni processor della rete.

3.6 Validità delle Simulazioni

La fase di progettazione del modello delle simulazioni è la parte più importante di un lavoro quale quello qui presentato, affinché i risultati ottenuti con le simulazioni siano validi. Ciò perché il modello deve riuscire a cogliere gli aspetti più importanti della realtà da simulare, di modo che, nonostante le necessarie semplificazioni, esso vi si avvicini il più possibile. Questo è l'intento del modello delle simulazioni descritto sin qui. Tuttavia, a prescindere dalla bontà del modello, affinché i risultati delle simulazioni siano credibili, non bisogna dimenticare che le simulazioni sono simulazioni stocastiche, in quanto in esse sono simulati processi casuali, e che pertanto le simulazioni vanno trattate in tutto e per tutto come degli esperimenti statistici [13]. Ciò implica che bisogna risolvere tre problematiche principali: la scelta di una fonte di casualità appropriata, la scelta del momento in cui iniziare a raccogliere le misure dalla simulazione e la scelta del numero di campioni da raccogliere.

Per quanto riguarda la scelta della fonte di casualità, essa viene data da degli pseudo-generatori di numeri casuali (*PRNG*), che devono essere sufficientemente potenti. Il problema di gran parte degli PRNG, infatti, è che essi hanno un periodo con lunghezza dell'ordine di 2^{32}. Ciò significa che se usati per simulazioni di grosse reti o se usati per simulazioni troppo lunghe, essi esauriscono velocemente, data la potenza dei processori odierni, tutti i numeri a disposizione, generando una ciclicità nei risultati. Fortunatamente, il simulatore scelto (`OMNet++`), utilizza il PRNG *Mersenne twister*, caratterizzato da un periodo di $2^{19937} - 1$ [12]. Tale PRNG è quindi più che sufficiente per qualsiasi tipo di simulazione [13].

La scelta del momento in cui iniziare a raccogliere le misure, così come il criterio di terminazione della simulazione, sono invece dei problemi molto complessi, in quanto richiedono l'uso di tecniche statistiche abbastanza elaborate [13]. Una trattazione esaustiva di tali problematiche esula dallo scopo del presente lavoro per tre motivi principiali: le tematiche affrontate sarebbero più adeguate per un lavoro di tipo statistico-matematico; la vastità dell'argomento è tale che richiede-

rebbe una trattazione a se[10]; ai fini di questo lavoro, non è stata implementata appositamente nessuna soluzione ai suddetti problemi, ma viene utilizzato un apposito tool per risolverli: `Akaroa2` [7]. `Akaroa2`, che viene descritto nel dettaglio nel Paragrafo 4.5, implementa, per risolvere i suddetti problemi, i metodi descritti in [14]. Nei prossimi paragrafi vengono quindi affrontate le problematiche della ricerca della fine del transitorio e del criterio di terminazione della simulazione da un punto di vista abbastanza di alto livello, in modo da fornire una panoramica dei principali approcci al problema, approfondendo brevemente i metodi che sono stati scelti per le simulazioni del presente lavoro[11]. Lo scopo della trattazione è da una parte quello di inquadrare il problema e dall'altra di fornire tutti gli strumenti necessari a comprendere l'utilizzo di `Akaroa2`.

3.6.1 L'individuazione della Fine del Transitorio

Poiché le simulazioni nel presente lavoro sono orientate a confrontare i vari schemi di routing a regime, la scelta del momento in cui iniziare a raccogliere le misure della simulazione è cruciale, in quanto bisogna essere certi di iniziare le misurazioni a regime. Infatti, le misure prese nel periodo transitorio (ovvero durante la creazione delle tabelle oppure quando le code non sono ancora piene, etc...) non sono significative e rischiano di "sporcare" il calcolo delle distribuzioni relative alle misure raccolte. Ad esempio se ci si mette nell'ottica di raccogliere i delay end-to-end dei messaggi scambiati sulla rete, nella fase iniziale in cui la rete inizia a riempirsi tali delay saranno relativamente più bassi rispetto a quelli a regime, in quanto le code dei processor sono inizialmente vuote. Esistono principalmente quattro approcci generali per risolvere questo problema:

- far girare la simulazione raccogliendo i dati sin dall'inizio, dopodiché identificare, quando la simulazione è ormai a regime, i campioni iniziali appartenenti al periodo transitorio e scartarli;

- impostare le condizioni iniziali del modello, in modo che le simulazioni partano già a regime;

- far girare la simulazione per un periodo abbastanza lungo, tale da rendere ininfluente il rumore causato dai dati iniziali;

[10] In [16] ad esempio vengono confrontati ben 46 metodi per ricercare la fine del periodo transitorio, mentre in [14] vengono illustrati una ventina di metodi tra euristici e statistici per trovare la fine del periodo transitorio e otto metodi relativi al criterio di terminazione di una simulazione.

[11] In [14] i metodi utilizzati sono spiegati approfonditamente utilizzando un rigoroso formalismo matematico-statistico, comprensivo di dimostrazioni e riferimenti alle teorie statistiche utilizzate.

- stimare i parametri della simulazione a regime utilizzando una breve simulazione di prova.

L'approccio più naturale ed anche il più utilizzato nella maggioranza delle simulazioni è il primo [16]. Tuttavia, il problema di identificare i campioni appartenenti al transitorio è un problema molto complesso, per risolvere il quale sono stati proposti moltissimi metodi, i quali si possono classificare in cinque categorie:

- metodi grafici, basati sull'ispezione delle serie temporali dei campioni prodotti dalla simulazione;
- metodi euristici, che forniscono delle regole empiriche per capire quando il rumore dovuto ai campioni iniziali è stato eliminato;
- metodi statistici;
- metodi basati sui test del rumore di inizializzazione: questi metodi impostano un test di ipotesi statistica, per cui l'ipotesi nulla è che il rumore dovuto ai campioni iniziali è stato eliminato. Pertanto questi metodi non individuano la durata del transitorio, ma piuttosto il momento in cui i campioni appartenenti al transitorio sono stati eliminati.
- metodi ibridi, costituiti da una qualche composizione dei metodi precedenti.

Sia [16] che [14] concordano sulla particolare efficienza e bontà di un metodo in particolare: il test di *Schruben*[12]. Tale metodo fa parte dei metodi basati sui test del rumore di inizializzazione. Il metodo di identificazione della fine del transitorio utilizzato da `Akaroa2` è anch'esso basato su tale test, pertanto è stato scelto il test di Schruben per le simulazioni del presente lavoro e viene quindi brevemente spiegato.
Il test in questione si basa sulla seguente assunzione [14]:

> I campioni iniziali dovuti al periodo transitorio sono stati eliminati da una sequenza di osservazioni, se la sequenza (standardizzata) determinata a partire dalle osservazioni rimanenti si comporta in maniera consistente con un processo stocastico (stazionario) standard.

In altre parole, viene impostato un test di verifica di ipotesi, in cui l'ipotesi nulla è che una data sotto-sequenza di osservazioni è stazionaria, o equivalentemente, che il periodo di transitorio iniziale non è incluso nelle osservazioni.

[12]Esistono diversi test di Schruben, quello a cui ci si riferisce in questo lavoro è il cosiddetto *optimal test* presentato in [17].

Tale test è basato sulla fatto che la sequenza delle somme parziali:

$$S_k = \overline{X}(n) - \overline{X}(k), \qquad k = 0, 1, 2, \ldots \qquad S_0 = S_n = 0$$

è particolarmente sensibile alla presenza del rumore di inizializzazione in $\overline{X}(n)$. $\overline{X}(n)$ e $\overline{X}(k)$ sono le medie campionarie, calcolate rispettivamente su n e sulle prime k osservazioni. In base a questa proprietà, il test di Schruben usa come statistica test, per verificare l'ipotesi nulla, la sequenza standardizzata $\{T(t)\}$, $0 < t \leq 1$ delle somme parziali S_k, dove:

$$T(t) = \frac{\lfloor nt \rfloor S_{\lfloor nt \rfloor}}{\hat{\sigma}\left[\bar{X}(n)\right]\sqrt{n}}, \qquad 0 < t \leq 1, \qquad T(0) = 0$$

Utilizzando il metodo delle serie temporali standardizzate [14], la statistica test $\{T(t)\}$ può essere confrontata con una distribuzione t, per rifiutare o accettare l'ipotesi nulla.

In `Akaroa2` tale metodo è utilizzato per creare una procedura sequenziale (descritta sempre in [14]), che consente di inizializzare correttamente tutti i parametri necessari al metodo di Schruben per funzionare e di scartare gradualmente i campioni iniziali finché l'ipotesi nulla non viene accettata, iniziando in tale caso la raccolta delle misure relative al periodo a regime.

3.6.2 Criterio di Terminazione della Simulazione

Una volta che sono stati eliminati i campioni relativi alla fase di transitorio, il problema successivo è quello di decidere quando terminare la simulazione. Tale problema è strettamente legato al numero di campioni che sono stati raccolti durante la simulazione. Se il numero di campioni raccolti è troppo piccolo, calcolare delle statistiche a partire da tali dati, come ad esempio la media, può portare a dei risultati erronei. Ponendosi quindi nell'ottica di calcolare la media di una serie di osservazioni ricevute da una simulazione, quello che si deve fare è raccogliere abbastanza campioni in modo che la media di tali campioni possa essere calcolata con un errore statistico relativo dell'intervallo di confidenza inferiore ad una certa soglia e con un accettabile livello di confidenza. Si fa notare che d'ora in avanti la quantità da stimare alla quale ci si riferisce sarà sempre la media dei campioni raccolti e, in particolare, la media dei delay end-to-end. Come visto è questa, infatti, la misura principale che si vuole calcolare ed è quindi per l'intervallo di confidenza di tale misura che vengono richiesti un alto livello di confidenza ed un basso errore statistico relativo.

Esistono due principali tipologie di metodi per affrontare tale problema: quelli che analizzano l'output della simulazione alla fine e verificano se i livelli di confidenza e di precisione desiderati sono stati raggiunti, obbligando però in caso negativo a ripetere la simulazione per collezionare un numero maggiore di campioni, e quelli cosiddetti sequenziali, in cui l'esecuzione della simulazione è intervallata da checkpoint, ad ognuno dei quali viene stimato se la accuratezza della media misurata è adeguata. In caso positivo la simulazione viene fermata, altrimenti si continua fino al checkpoint successivo, finché le condizioni stabilite non sono state raggiunte. Ovviamente il secondo approccio è il più efficiente, in quanto col primo si rischia di dover ripetere la simulazione troppe volte se non si riesce a stimare correttamente il numero di campioni necessari, pertanto ci concentreremo sulla seconda classe di metodi.

La maggior parte di tali metodi si basa su un criterio di terminazione del tipo $\epsilon \leq \epsilon_{max}$, con:

$$\epsilon = \frac{\Delta_x}{\overline{X}(n)} \qquad 0 < \epsilon < 1$$

dove Δ_x è metà della lunghezza dell'intervallo di confidenza per la media di livello $(1-\alpha)$, con α dato. Tale rapporto ϵ è detto *precisione relativa dell'intervallo di confidenza*. La simulazione viene quindi fermata al primo checkpoint per cui $\epsilon \leq \epsilon_{max}$. Si fa notare che:

$$1-\alpha \leq P\left(\left|\overline{X}(n)-\mu_x\right| \leq \epsilon\left|\overline{X}(n)\right|\right) \leq P\left(\frac{\left|\overline{X}(n)-\mu_x\right|}{|\mu_x|} \leq \frac{\epsilon}{1-\epsilon}\right)$$

dove $\frac{\left|\overline{X}(n)-\mu_x\right|}{|\mu_x|}$ è detto errore relativo dell'intervallo di confidenza e pertanto si può calcolare ϵ a partire dall'errore relativo massimo desiderato.

I vari metodi differiscono quindi tra loro per le statistiche test usate per calcolare gli intervalli di confidenza descritti. Sebbene in [14] si illustrino otto metodi usati per stabilire il criterio di terminazione di una simulazione, i più usati in letteratura sono principalmente due:

- il metodo basato sulla *spectral analysis*;

- il metodo basato sulle *batch means*.

Entrambi questi metodi sono anche implementati da `Akaroa2`. La scelta fatta in questo lavoro è stata quella di utilizzare il metodo delle batch means. Tale scelta è dovuta alle seguenti ragioni: innanzitutto il metodo delle batch means ha una teoria relativamente semplice e comprensibile, pertanto si può intervenire facilmente sui parametri di configurazione di tale metodo e nel metodo delle batch

means implementato in `Akaroa2` vengono richiesti solamente tre parametri di configurazione (e i loro valori di default vanno bene per la maggioranza dei casi). Di contro il metodo della spectral analysis fa uso di strumenti matematici e statistici più complessi, pertanto risulta difficile avere una padronanza della teoria di tale tecnica tale da poter intervenire agevolmente sui parametri di configurazione di questo metodo. Inoltre non sono state trovate in lettura analisi che confermino o smentiscano la maggiore efficacia di un metodo piuttosto che l'altro, pertanto, mancando motivi significativi per scegliere il metodo più complesso si è optato, ragionevolmente, per scegliere quello più semplice.

Il principale problema di stimare l'intervallo di confidenza per la media dei campioni prodotti da una simulazione stocastica risiede nel fatto che tali campioni sono altamente correlati tra loro. Pertanto il calcolo della varianza della media campionaria, utilizzata per stabilire l'intervallo di confidenza, risulta estremamente complicato a causa della presenza dei coefficienti di autocorrelazione. Al contrario, nel caso in cui i campioni siano indipendenti, il calcolo dell'intervallo di confidenza della media risulta molto semplice, in quanto si può usare uno stimatore della varianza della media campionaria la cui forma è molto semplice.

Il metodo delle batch means si basa sull'idea di indebolire la correlazione presente nei campioni raccolti dalla simulazione, in modo da potersi ricondurre al caso in cui i campioni siano indipendenti. Per farlo esso divide i campioni raccolti in batch di lunghezza fissa e calcola la media campionaria di ogni batch: se i batch sono abbastanza grandi, le medie di tali batch si possono considerare non correlate (o poco correlate) tra loro. Pertanto si può stimare la media di tali medie (che corrisponde alla media di tutti i campioni) e calcolarne facilmente l'intervallo di confidenza, in quanto si ricade nel caso in cui i campioni di partenza (in questo le caso le medie dei batch) sono tra loro indipendenti.

Più formalmente, se sono stati raccolti n campioni (esclusi ovviamente quelli appartenenti al transitorio), $x_1, x_2, \ldots, x_n$ essi vengono divisi in $k = n/m$ batch (non sovrapposti) di lunghezza m, $(x_{1,1}, x_{1,2}, \ldots, x_{1,m})$, $(x_{2,1}, x_{2,2}, \ldots, x_{2,m})$, $\ldots$, $(x_{k,1}, x_{k,2}, \ldots, x_{k,m})$. Di tali batch vengono calcolate le medie $\overline{X}_1(m)$, $\overline{X}_2(m)$, $\ldots$, $\overline{X}_k(m)$. La media μ_x viene quindi stimata come:

$$\overline{X}(n) = \overline{\overline{X}}(k, m) = \frac{1}{k} \sum_{i=1}^{k} \overline{X_i}(m)$$

Pertanto se la dimensione dei batch m è sufficientemente grande, le medie di tali batch si possono considerare come non correlate tra loro, in quanto calcolate su campioni molto distanti tra loro nel tempo, e si può calcolare l'intervallo di

confidenza per la media μ_x di livello $(1-\alpha)$ come:

$$\overline{\overline{X}}(k,m) \pm t_{k-1,1-\frac{\alpha}{2}} \hat{\sigma} \left[\overline{\overline{X}}(k,m)\right]$$

dove

$$\hat{\sigma}^2 \left[\overline{\overline{X}}(k,m)\right] = \sum_{i=1}^{k} \frac{\left(\overline{X_i}(m) - \overline{\overline{X}}(k,m)\right)^2}{k(k-1)}$$

è lo stimatore della varianza di $\overline{\overline{X}}(k,m)$ e la distribuzione t con $k-1$ gradi di libertà può essere utilizzata, in quanto, grazie al teorema centrale del limite, le medie dei batch possono essere approssimate come appartenenti ad una distribuzione normale.

In questo metodo risulta quindi cruciale decidere la dimensione dei batch. Tuttavia, `Akaroa2` utilizza un metodo sequenziale, per cui la dimensione dei batch viene continuamente aumentata, automaticamente, finché le medie dei batch non superano un test di autocorrelazione, risultando non correlate. A questo punto si utilizza il metodo delle batch means con la dimensione dei batch trovata, calcolando ad ogni checkpoint l'intervallo di confidenza e l'errore relativo massimo, così come illustrato precedentemente. Occasionalmente, se l'algoritmo fa fatica a convergere ai valori desiderati, la dimensione dei batch può essere aumentata ulteriormente. Per funzionare, il metodo implementato ha bisogno di tre parametri di configurazione: la lunghezza iniziale dei batch (la lunghezza finale trovata è un multiplo della lunghezza iniziale voluta), la lunghezza della sequenza di medie dei batch da testare per l'autocorrelazione ed il livello di significatività al quale i coefficienti di autocorrelazione delle batch means sono testati per determinare la scelta della dimensione dei batch.

Capitolo 4

L'implementazione delle Simulazioni

In questo capitolo viene descritto l'aspetto implementativo delle simulazioni, ossia viene spiegato come è stato realizzato il modello visto nel capitolo precedente. Innanzitutto viene fornita una panoramica sui simulatori ad eventi discreti e sul simulatore scelto, `OMNet++`, quindi vengono passate in rassegna le principali scelte implementative effettuate, come l'architettura della simulazione, i parametri del modello, le reti utilizzate, ed infine viene descritto `Akaroa2`, il software utilizzato per dare una validità ai risultati ottenuti, i cui aspetti teorici sono stati affrontati nel Paragrafo 3.6.

4.1 `OMNet++`

Lo scopo di questa presentazione di `OMNet++` è di fornirne una panoramica, in modo da poterne capire i punti di forza, il modello di programmazione e i componenti principali, senza entrare troppo nei dettagli che possono essere facilmente reperiti nel manuale utente [18].

Innanzitutto `OMNet++` è un simulatore ad eventi discreti, pertanto risulta necessario capire cosa si intende per simulatore ad eventi discreti.

4.1.1 I Simulatori ad Eventi Discreti

Un sistema ad eventi discreti è un sistema in cui i cambiamenti di stato (gli eventi) avvengono in istanti di tempo discreti. Tali eventi sono istantanei, per cui è nullo il tempo impiegato per un cambiamento di stato. Si assume quindi che nulla (o meglio nulla di interessante) accada tra due eventi consecutivi, cioè nel sistema non si hanno cambiamenti di stato tra gli eventi. Ciò è contrario a quanto avviene nei sistemi continui, in cui i cambiamenti di stato sono continui.

Quei sistemi che possono essere visti come sistemi ad eventi discreti possono essere modellati usando una simulazione ad eventi discreti (*DES*, *Discrete Event Simulation*). Per esempio, una rete di computer può essere vista come un sistema ad eventi discreti, di cui alcuni esempi di eventi possono essere: l'inizio della trasmissione di un pacchetto, la fine della trasmissione di un pacchetto, l'estrazione dalla coda di un pacchetto. Ciò implica che tra due eventi come l'inizio della trasmissione di un pacchetto e la fine della trasmissione, non succeda nulla di interessante.

Il tempo al quale accadono gli eventi è comunemente detto *timestamp*, mentre si può distinguere tra: il tempo all'interno del modello simulato, detto *tempo della simulazione* o *tempo virtuale*; il tempo trascorso all'esterno del modello, ovvero il tempo che indica il tempo di vita del programma che ha simulato il sistema, detto *tempo reale*; il tempo durante il quale tale programma ha utilizzato il processore, detto *tempo di CPU*.

Nelle simulazioni ad eventi discreti, l'insieme degli eventi futuri viene tenuto in una struttura dati comunemente chiamata *FES* (*Future Event Set*). Per simulatore si intende un programma che si comporta secondo il seguente pseudo-codice:

---

inizializza – ciò include costruire il modello ed inserire gli eventi iniziali nel FES
while (*il FES non è vuoto e la simulazione non è completata*) **do**
    prendi il primo evento dal FES
    $t \leftarrow$ *timestamp di questo evento*
    processa l'evento
    (*processare l'evento può causare l'inserimento di nuovi eventi nel FES*
    o la cancellazione di alcuni già presenti)
end while
termina la simulazione (scrivi i risultati statistici, etc...)

Inizialmente quindi il simulatore si occupa di inizializzare tutte le strutture dati del modello della simulazione, chiamare eventuali funzioni di inizializzazione fornite dall'utente e inserire gli eventi iniziali nel FES, in modo che la simulazione possa partire. In seguito viene eseguito il ciclo principale, in cui vengono presi gli eventi dal FES e vengono processati. Gli eventi sono processati strettamente in ordine di timestamp, di modo che la causalità tra gli eventi sia mantenuta e che nessun evento possa incidere su eventi già passati. Si fa notare che processare un evento significa chiamare del codice fornito dall'utente. Per esempio in una rete di computer, processare un evento "timeout scaduto" significa dover rispedire una copia del pacchetto di rete, aggiornare il contatore dei tentativi, fissare un nuovo evento "timeout" e così via. Il codice fornito dall'utente può anche rimuovere degli eventi dal FES, per esempio quando viene cancellato un timeout.

La simulazione si ferma quando non sono rimasti più eventi nel FES, oppure quando è stato raggiunto il tempo massimo di esecuzione stabilito, oppure perché le statistiche raccolte hanno raggiunto il desiderato livello di accuratezza. Prima che il programma di simulazione termini completamente, vengono registrate, su memoria di massa, tutte le statistiche, o altri dati, voluti dall'utente.

4.1.2 I Punti di Forza di `OMNet++`

Prima di iniziare a descrivere più dettagliatamente gli aspetti principali di `OMNet++`, vengono qui di seguito elencati i punti di forza di tale simulatore, per cui esso è stato preferito, senza alcun dubbio, ad altri simulatori esitenti:

- è gratuito, per scopi di ricerca accademica[1], ed è opensource.
- il kernel del simulatore è chiaramente separato dal modello della simulazione tramite una semplice interfaccia, per cui l'unica preoccupazione, da parte dell'utente, è di implementare il modello secondo le API del kernel, senza preoccuparsi di come questo funziona;
- è altamente modulare: i componenti del modello della simulazione possono essere assemblati in una gerarchia, teoricamente infinita, di moduli, per cui è facile incapsulare un modulo dentro un altro per aggiungergli nuove funzionalità senza stravolgere l'architettura della simulazione. Tali moduli sono inoltre riutilizzabili e combinabili tra loro;
- ha una architettura tale per cui il modello è nettamente separato dalle simulazioni, nel senso che tutti i parametri del modello sono configurabili in un file esterno al modello. Inoltre il modello è separato dalla topologia della rete. Pertanto risulta immediato passare da una rete all'altra o da uno scenario all'altro, o cambiare alcuni parametri della simulazione, il tutto senza dover ricompilare nulla;
- è in grado simulare efficientemente topologie di rete molto grandi;
- presenta una interfaccia di programmazione molto matura e pulita, e fornisce classi ausiliare e funzionalità accessorie per ogni evenienza, come: code di pacchetti, generatori di numeri casuali, generazione di numeri casuali secondo numerose distribuzioni di probabilità, classi per raccogliere i dati, moduli statistici per elaborare i dati raccolti, gerarchie di pacchetti, incapsulamento di pacchetti;

[1] per scopi commerciali, bisogna acquistare una licenza di `OMNest`, la sua controparte a pagamento.

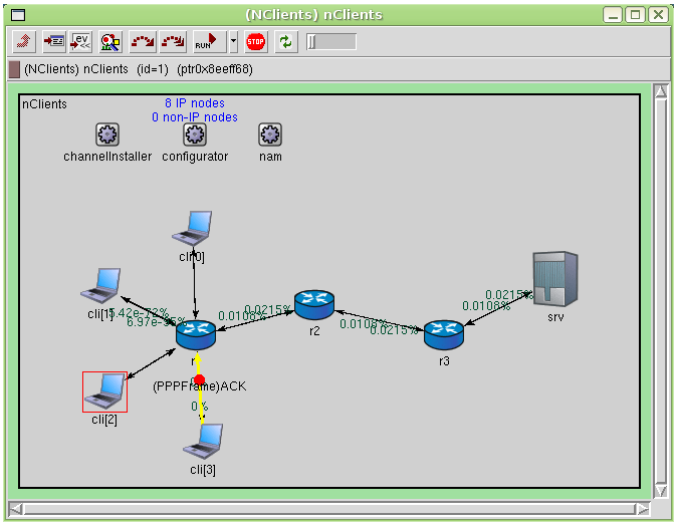


Figura 4.1: Interfaccia grafica di `OMNet++`.

- ha una straordinaria interfaccia grafica (vedi Figura 4.1) in grado di visualizzare ogni singolo dettaglio del modello implementato, rendendo in questo modo estremamente semplice effettuare il debug di protocolli o algoritmi distribuiti;

- oltre agli strumenti necessari a sviluppare ed eseguire le simulazioni dispone di un vero e proprio pacchetto applicativo, costituito da applicazioni grafiche per creare le topologie di rete e per visualizzare i risultati prodotti dalle simulazioni creando vari tipi di grafici;

- ha un manuale utente chiaro e sempre aggiornato, così come la documentazione delle API di programmazione. Inoltre dispone di una serie di tutorial, veramente ben fatti, che, assieme all'ottima fattura del manuale utente, rende estremamente semplice imparare ad usare il simulatore sfruttandone appieno tutte le funzionalità;

- dispone di numerose implementazioni di protocolli per implementare reti TCP/IP, peer-to-peer, LAN, MAN, ad-hoc, wireless e sensor network. Di queste la più imponente è sicuramente l'`INET Framework` che modella fino al più piccolo dettaglio l'intero stack TCP/IP, così come la maggior parte dei protocolli utilizzati su Internet;

- dispone di una grossa comunità di utilizzatori, per lo più ricercatori, per cui non è difficile, iscrivendosi alla mailing list di `OMNet++` trovare risposte a qualunque tipo di domanda, relativa sia all'utilizzo del simulatore sia a questioni più teoriche riguardanti argomenti di ricerca;

- supporta l'esecuzione di simulazioni distribuite, ovvero il modello da simulare può essere diviso, agendo semplicemente su un file di configurazione, in più parti, ognuna delle quali viene poi eseguita, parallelamente alle altre e comunicando con le altre, su una macchina diversa.

4.1.3 Le Caratteristiche Principali

I Moduli e la loro Gerarchia

Un modello di `OMNet++` consiste in una gerarchia di moduli annidati, che comunicano passandosi messaggi tra loro. Come esemplificato nella Figura 4.2, il modulo a capo della gerarchia è il *system module*, il quale contiene dei sotto-moduli, che a loro volta possono contenere altri sotto-moduli e così via, senza limiti alla profondità dell'annidamento, in modo da poter riflettere nel modello la struttura logica del sistema da simulare.

I moduli che contengono sotto-moduli sono detti *compund module*, mentre quelli che sono alla fine della gerarchia sono detti *simple module*. I simple module sono i moduli che contengono gli algoritmi del modello e sono implementati dall'utente in `C++`, estendendo una opportuna classe `cSimpleModule`.

Il modello definito dall'utente viene descritto definendo i vari "tipi" di moduli che verranno utilizzati. Si parte definendo i simple module fino a definire, tramite questi, tipi di moduli via via più complessi. Il tipo di modulo definito che contiene tutti gli altri viene istanziato come system module e tutti gli altri moduli definiti dall'utente vengono istanziati come suoi sotto-moduli o sotto-sotto-moduli (e così via) come definito dai loro tipi.

Visti esternamente sia i simple module che i compound module sono identici, pertanto l'utente può facilmente decidere di reimplementare la funzionalità di un simple module all'interno di un compound module e viceversa, senza cambiare la struttura del modello.

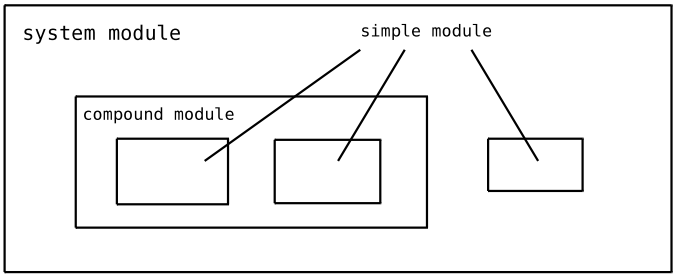


Figura 4.2: Gerarchia dei moduli di `OMNet++`.

I tipi dei moduli possono essere memorizzati esternamente al modello in cui vengono utilizzati, pertanto si possono creare vere e proprie librerie di componenti.

I Messaggi, le Interfacce e i Link

I moduli comunicano tra loro scambiandosi messaggi. Tali messaggi in una simulazione reale possono essere rappresentati da pacchetti di rete, ma non solo, in quanto i messaggi possono contenere qualunque struttura dati di qualunque complessità. I simple module possono spedire messaggi direttamente alla loro destinazione (chiamando una funzione della destinazione) oppure attraverso un cammino predefinito, utilizzando le interfacce e le connessioni, come in una normale rete.

Il tempo "logico" locale di un modulo avanza quando il modulo riceve un messaggio. Il messaggio può arrivare da un altro modulo oppure dal modulo stesso, pertanto se un modulo vuole implementare un timeout esso manda un messaggio a se stesso, in modo che arrivi appena il timeout è scaduto.

Un modulo ha interfacce di output e di input, i messaggi sono spediti tramite le interfacce di output e arrivano attraverso le interfacce di input.

Ogni link, o connessione, tra i moduli viene creata all'interno di un singolo livello della gerarchia dei moduli. Pertanto, all'interno di un compound module si possono creare connessioni "orizzontali", tra le interfacce di due suoi sotto-moduli (Figura 4.3(a)), oppure "verticali" tra l'interfaccia di un sotto-modulo e quella del compound module (Figura 4.3(b)).

Essendo la struttura del modello gerarchica, i messaggi, per partire da un simple-module ed arrivare ad un altro simple module per essere processati, tipicamente viaggiano attraverso una serie di connessioni, sia verticali sia orizzontali. La serie di connessioni che porta da un simple module ad un altro è detta *route*. All'interno di una route i compound module agiscono semplicemente come dei relay, facendo passare i messaggi dal loro interno al loro esterno.

(a) Connessione tra sotto-moduli

(b) Connessione tra livelli gerarchici

Figura 4.3: Tipi di connessione tra moduli

Il Modello di Trasmissione dei Pacchetti

Alle connessioni tra i moduli possono essere assegnati tre parametri che facilitano la modellazione di reti di comunicazione: il ritardo di propagazione (*delay*), il tasso di errore nell'inviare i bit ed il tasso di trasmissione (cioè la banda). Tutti e tre questi parametri sono opzionali e possono essere specificati per ogni singola connessione, oppure all'interno di definizioni di tipi di link che possono poi essere riutilizzate in tutto il modello.

I tre parametri sono definiti come segue: il ritardo di propagazione è la quantità di tempo che passa da quando viene spedito il primo bit a quando tale bit giunge a destinazione, ovvero indica di quanti secondi viene ritardato l'arrivo di un messaggio per il fatto che viaggia sul canale. Il tasso di errore specifica la probabilità che un bit non venga trasmesso correttamente, e permette di simulare link non affidabili. Il tasso di trasmissione è specificato in bit/sec ed è usato per calcolare il tempo di trasmissione di un pacchetto, dove per tempo di trasmissione si intende per quanto tempo il canale resta occupato a causa della trasmissione di un messaggio.

Parametri dei Moduli

Ogni modulo può avere dei parametri e i valori di tali parametri possono essere assegnati sia nella definizione del modulo, sia in un file di configurazione particolare, il file `omnetpp.ini`. Tale file ha una sintassi propria e viene letto dal simulatore quando la simulazione viene lanciata, pertanto utilizzando questo file per assegnare i valori ai parametri questi possono essere cambiati senza ricompilare il modello. Un esempio di file `omnetpp.ini` si può vedere in Figura 4.4.

```
#File Omnetpp.ini

[General]
network=dfn
num-rngs = 4

[Parameters]
*.nodetype = "PSFProcessor"

*.node*.processor.max_subscriptions = 100
*.node*.processor.app_zipf_length = 1800
*.node*.processor.zipf_alpha = 0.1
*.node*.processor.zipf_rng = 1
*.node*.processor.gen_min_value = 1
*.node*.processor.gen_max_value = 1800
*.node*.processor.gen_delay = exponential(0.025, 3)
*.node*.processor.packet_size = 1000
*.node*.processor.queue_length = 50
*.node*.processor.constraints_per_subscription = 1000
*.node*.queue[*].capacity = 50
```

Figura 4.4: Esempio di file `omnetpp.ini`

I parametri possono essere stringhe, numeri, valori di verità o persino alberi XML. Ai parametri numerici possono essere assegnati valori statici, oppure valori generati da un generatore di numeri casuali secondo una certa distribuzione di probabilità oppure generati da funzioni definite dall'utente.

Descrizione della Topologia

La struttura del modello, ovvero come sono interconnessi tra loro i moduli (simple o compound) sia verticalmente, cioè lungo la gerarchia sia orizzontalmente, cioè tra di loro, è definita in un file detto `NED` file, di cui un esempio si può vedere in Figura 4.5.
Tramite il `NED` file si possono quindi definire i simple module, i parametri dei moduli, i moduli compound (come una connessione di altri moduli precedentemente definiti, siano essi simple o compound) e si può definire anche la topologia della rete. La rete pertanto è costituita da un compound module, il quale a sua volta ha al suo interno tutti i suoi nodi connessi tra loro a parità di livello gerarchico. I nodi stessi della rete possono essere a loro volta dei compound module.

Il Modello di Programmazione

I simple module contengono gli algoritmi, implementati come funzioni `C++`. Chi scrive la simulazione può usare tutta la potenza e la flessibilità offerti da tale linguaggio di programmazione, come ad esempio i concetti della programmazione object oriented (ereditarietà, polimorfismo, etc...) ed i design pattern, per estendere le funzionalità del simulatore.

```
import "processors";
import "queue";

module PSFContainer
  gates:
    in: in[ ];
    out: out[ ];
  submodules:
    processor: PSFProcessor;
      gatesizes:
        in[sizeof(in)],
        out[sizeof(out)];
    queue: DropTailQueue[sizeof(out)];
    server: Server[sizeof(out)];
  connections:
    for i=0..sizeof(in)-1 do
      processor.out[ i ] --> queue[ i ].in;
      queue[ i ].out --> server[ i ].in;
      server[ i ].out --> out[ i ];
      processor.in[ i ] <-- in[ i ];
    endfor;
endmodule
```

Figura 4.5: Esempio di file `NED`

Tutti gli oggetti usati da una simulazione, come i messaggi, i moduli, le code, sono rappresentati come delle classi C++. Essi sono stati progettati per cooperare insieme efficientemente, creando un potente framework di programmazione delle simulazioni. Pertanto la libreria delle classi utilizzabili dalla simulazione, fornita da OMNet++, è costituita da classi come i moduli, le interfacce, le connessioni, i parametri, i messaggi, le code, classi per raccogliere i dati prodotti dalla simulazione, classi statistiche e classi che stimano le distribuzioni. Tutte queste classi sono implementate in maniera tale per cui è molto semplice esaminare gli oggetti durante l'esecuzione della simulazione, per visionarne le caratteristiche come il nome, le variabili di stato oppure il contenuto di code e array. Grazie a tale caratteristica OMNet++ è anche in grado di fornire una interfaccia grafica con cui è possibile esaminare la simulazione in maniera approfondita.

Il modello della simulazione può essere implementato per mezzo di un modello di programmazione *event-driven*, oppure in uno stile simile ai processi dei sistemi operativi, dove il flusso della simulazione viene esplicitamente passato da un processo all'altro. Lo stile simile ai processi è espressamente sconsigliato dall'autore di OMNet++ e viene fornito solo per modellare quei casi in cui lo stile event-driven risulta incompatibile col modello da simulare. Pertanto lo stile utilizzato nel presente lavoro è stato lo stile event-driven, che viene brevemente approfondito qui di seguito.

Come detto precedentemente, in una simulazione gli eventi avvengono nei simple module. Pertanto i simple module racchiudono tutto il codice C++ che genera gli eventi e quello che reagisce agli eventi, ovvero i simple module implementano il comportamento del modello.

Nello stile event-driven, un simple module viene implementato estendendo una opportuna classe cSimpleModule. Tale classe non offre particolari funzionalità, ma presenta tre funzioni che devono essere ridefinite per implementare il comportamento del modello che si vuole simulare. Tali funzioni sono le seguenti:

- `void initialize();`

- `void handleMessage(cMessage * msg);`

- `void finish();`

La funzione initialize() ha il significato seguente: nella fase di inizializzazione, OMNet++ costruisce la rete, istanzia i simple e compound module necessari e li connette come definito nel NED file, infine chiama per ogni modulo istanziato la funzione initialize().

La funzione `handleMessage()`, invece, viene chiamata quando il modulo in questione deve processare un evento, pertanto è tramite questa funzione che viene implementato il comportamento del modello. Il parametro preso dalla funzione in questione è l'evento stesso, che altro non è che un messaggio. Questa funzione viene quindi chiamata dal kernel del simulatore, quando il modulo riceve un messaggio. Per processare il messaggio il simple module in questione può utilizzare tutte le funzioni che vuole, cambiare le proprie strutture dati e, chiamando opportune funzioni fornite dal kernel del simulatore, può spedire messaggi a se stesso o, tramite le sue interfacce di rete, ad altri moduli.

Infine la funzione `finish()` viene chiamata per ogni modulo quando la simulazione termina correttamente e tipicamente serve per scrivere le statistiche finali in memoria di massa.

Pertanto il tipico comportamento di un simulatore, descritto nel Paragrafo 4.1.1, viene implementato da `OMNet++` nel seguente modo:

---

chiama `initialize()` *per tutti i moduli*
while (*il FES non è vuoto e la simulazione non è completata*) **do**
    prendi il primo evento dal FES
    $t \leftarrow$ *timestamp di questo evento*
    $m \leftarrow$ *modulo che contiene questo evento*
    m->`handleMessage(evento)`
end while
chiama `finish()` *per tutti i moduli*

I Componenti di una Simulazione

Il modello di una simulazione in `OMNet++` consiste nei seguenti componenti:

- un insieme di file `NED` in cui vengono definiti i moduli (simple o compound) della simulazione, i loro parametri, le interfacce di rete, e le connessioni tra i moduli, cioé la topologia della rete (con indicate banda e latenza dei link);

- un file `omnetpp.ini` in cui vengono dati i valori a tutti i parametri dei moduli del modello;

- un insieme di file `.msg` in cui vi sono le definizioni dei messaggi usati nel modello, scritti con una particolare sintassi;

- i file sorgenti, scritti in `C++`, dei simple module, ovvero le classi che implementano i simple module, così come tutte le ulteriori classi necessarie ai simple module per implementare il comportamento del modello.

Il sistema di simulazione fornisce, invece, i seguenti componenti:

- il kernel della simulazione, che contiene il codice necessario per gestire la simulazione e la libreria delle classi della simulazione. Esso è implementato come una libreria scritta in `C++`;

- le interfacce utente, grafiche e testuali, necessarie per facilitare il debug, l'esecuzione dimostrativa o l'esecuzione in batch delle simulazioni;

La simulazione viene costruita mettendo insieme tutti i componenti precedenti, quindi dapprima i file `.msg` e `NED` vengono trasformati in classi `C++`, dopodiché tutti i file sorgenti vengono compilati e linkati col kernel della simulazione e l'interfaccia utente scelta (grafica o testuale).

La Simulazione

Il file eseguibile della simulazione, creato come visto al punto precedente, è un programma a sé stante, che può essere eseguito anche su macchine senza `OMNet++` installato e senza i file che descrivono il modello. Quando il programma viene lanciato, esso innanzitutto legge il file di configurazione `omnetpp.ini` che contiene tutte le impostazioni che definiscono come la simulazione viene eseguita, ovvero con quali interfacce utente, se in modalità batch oppure interattiva e imposta i valori dei parametri.

L'output della simulazione viene scritto in file di dati particolari chiamati *vector* e *scalar*, che contengono rispettivamente i singoli campioni delle misure di interesse raccolti durante tutta la simulazione e valori numerici di altre quantità costituite da un singolo numero (cioè aggregate e non un insieme di campioni). Inoltre l'utente può decidere di produrre, come output della simulazione, ulteriori file da lui stesso definiti. `OMNet++` fornisce anche dei programmi grafici per poter visualizzare e filtrare i dati contenuti nei file vector e scalar, per esempio per mezzo di diagrammi, ma tali file sono scritti in un formato facilmente esportabile per essere usato con altri programmi.

La simulazione può essere eseguita in modalità grafica ed in tale modalità è possibile, tramite l'interfaccia grafica, ispezionare tutti i dettagli relativi ai moduli della simulazione, osservare il flusso dei messaggi ed intervenire sui parametri a runtime. Come detto, questa opportunità è utilissima in fase di debug o a scopo dimostrativo, per mostrare come funziona esattamente il modello. Se le performance sono importanti la simulazione può essere eseguita in modalità testuale.

Infine, il programma della simulazione può contenere diversi modelli indipendenti al suo interno e l'utente può simulare un modello piuttosto che un altro

semplicemente specificando nel file di configurazione `omnetpp.ini`, quale modello intende eseguire.

4.2 Architettura della Simulazione

Le simulazioni oggetto del presente lavoro sono state realizzate implementando i processor e la rete da simulare come moduli `OMNet++`. In particolare la rete è il modulo padre al cui interno vi sono i moduli processor collegati tra loro. Il modo in cui i processor sono collegati, ovvero la topologia della rete, viene definito in un file `NED` generato da `BRITE`, dei cui dettagli ci si occupa nel Paragrafo 4.4. Qui viene invece analizzato come sono stati implementati i processor, i messaggi che questi si scambiano e la raccolta delle misure.

4.2.1 I Processor

Il modulo di un processor è in realtà un compound module detto `ProcessorCompound`. Esso è strutturato come mostrato nella figura 4.6: ha delle interfacce di ingresso che sono collegate alle interfacce di ingresso del modulo `Processor`, il quale ha le proprie interfacce di uscita collegate ad un modulo `OutputQueue`, il quale a sua volta ha le interfacce di uscita collegate a quelle di uscita del modulo `ProcessorCompound`. Il modulo `Processor` è il processor vero e proprio, che implementa le funzioni di router e di generatore e ricevente dei messaggi, mentre il modulo `OutputQueue` implementa la coda associata ad ogni interfaccia di uscita del modulo `Processor`. Pertanto il modulo `ProcessorCompound` rappresenta un processor, così come modellato nel Paragrafo 3.1.1. Si fa notare che, nonostante le interfacce di input e di output siano differenti, per il fatto che `OMNet++` richiede di differenziarle, i link entranti ed uscenti da tali interfacce possono essere visti come un unico link bidirezionale, in quanto per ogni link entrante avente determinate banda e latenza, ve ne è uno uscente con le medesime caratteristiche. Inoltre ogni

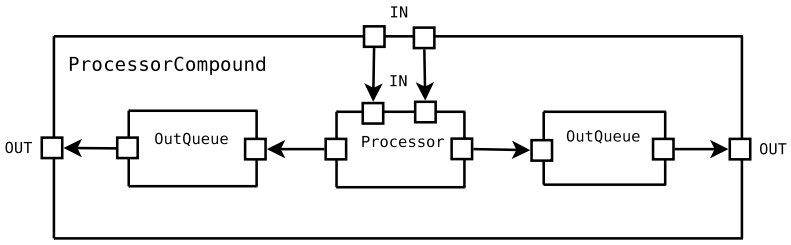


Figura 4.6: Struttura di un modulo `ProcessorCompound`.

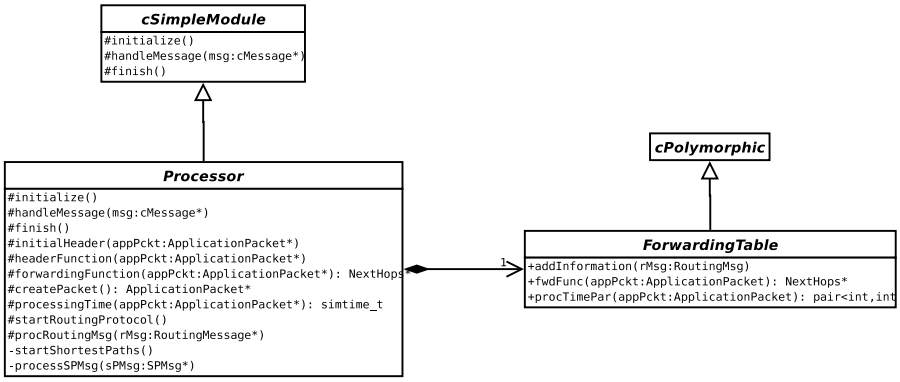


Figura 4.7: Class Diagram UML delle classi astratte.

modulo `ProcessorCompound` ha associato un identificativo numerico, che funge da indirizzo di rete.

Sia il modulo `Processor` che `OutputQueue` sono stati implementati come dei simple module e sono pertanto delle classi `C++` che estendono la classe `cSimpleModule`. L'implementazione del modulo `OutputQueue` è abbastanza banale: esso mette i pacchetti in arrivo dal modulo `Processor` in una coda FIFO e, continuamente, non appena il canale di trasmissione diventa libero, trasmette il primo messaggio della coda sul canale. L'implementazione del modulo `Processor` è decisamente più complessa, pertanto risulta necessario entrare maggiormente nei dettagli. Innanzitutto il modulo `Processor` è in realtà una astrazione, in quanto nella simulazione reale viene effettivamente istanziato il modulo che implementa uno specifico schema di routing: pertanto esistono dei moduli `PSFProcessor`, `IPSProcessor`, `PIFProcessor`, `DRPProcessor` e `FloodProcessor`. Tali moduli sono tutti specializzazioni di una classe astratta `Processor` il cui class diagram UML viene mostrato in Figura 4.7. Tale classe astratta si appoggia inoltre ad un'altra classe astratta, la classe `ForwardingTable`, che rappresenta la tabella di forwarding del processor[2]. La classe `Processor` innanzitutto ridefinisce i metodi della classe `cSimpleModule`: il metodo `initialize()` inizializza opportunamente le strutture dati, gli eventi iniziali da inserire nel FES, legge i parametri della simulazione dal file di configurazione `omnetpp.ini` e genera le sottoscrizioni associate al processor (secondo la distribuzione zipf specificata); il metodo `finish()` scrive le statistiche relative ad ogni nodo; mentre il metodo `handleMessage()` implementa il comportamento generico di un processor e per farlo utilizza tutte le altre funzioni definite.

Innanzitutto `handleMessage()`, chiama la funzione `startShortestPaths()`,

[2]Affinché la classe `ForwardingTable` si integri correttamente con la simulazione e possa essere ispezionata a runtime, essa estende la classe `cPolymorphic` di `OMNet++`.

che serve per iniziare il protocollo per la costruzione della tabella di routing unicast di ogni nodo. A tal fine la funzione `procSPMsg()` viene chiamata da `handleMessage()` ogni volta che viene ricevuto un messaggio di tale protocollo. Terminato tale protocollo, `handleMessage()` chiama le funzioni `startRoutingProtocol()` e `procRoutingMsg()` in maniera analoga alle due precedenti, ma questa volta il protocollo in questione è quello che serve per riempire le tabelle di forwarding con le informazioni necessarie per eseguire il routing content-based. La funzione `procRoutingMsg()` inoltre chiama la funzione `addInformation()` della classe `ForwardingTable` per fare in modo che le informazioni contenute nei messaggi del protocollo di routing siano inserite opportunamente nella tabella di forwarding. L'implementazione delle ultime tre funzioni illustrate varia a seconda dello schema di routing utilizzato, essendo le strutture dati e le informazioni locali necessarie ad ogni nodo dipendenti dallo schema di routing, pertanto essi sono metodi astratti che vengono ridefiniti opportunamente nelle specializzazioni della classe `Processor`. Ulteriori considerazioni riguardo ai protocolli di routing vengono espresse nel Paragrafo 4.2.3.

Una volta che le tabelle di forwarding sono state riempite, i nodi iniziano a generare, inoltrare e ricevere i messaggi. Pertanto la funzione `handleMessage()` viene chiamata solamente in tre circostanze: quando riceve un evento che indica la generazione di un nuovo messaggio[3], quando riceve un messaggio applicativo, oppure quando riceve un evento che indica di processare il primo messaggio presente nella coda in ingresso. Nel primo caso `handleMessage()` chiama la funzione `createPacket()` che genera un nuovo messaggio applicativo (secondo la distribuzione uniforme specificata) e lo mette nella coda in ingresso. Nel secondo caso mette direttamente il messaggio nella coda in ingresso, mentre nell'ultimo caso prende il primo messaggio dalla coda di ingresso, lo processa per vedere se è selezionato dal predicato associato al processor, dopodiché, se il messaggio è stato generato dal nodo stesso, chiama la funzione `initialHeader()` (corrispondente alla funzione Init), altrimenti chiama la funzione `headerFunction()` (corrispondente alla funzione Hdr) per impostare l'header del pacchetto da spedire. Infine viene chiamata la funzione `forwardingFunction()` (corrispondente alla funzione Fwd), che a sua volta chiama la funzione `fwdFunc()` della classe `ForwardingTable`, per ottenere l'insieme dei vicini a cui inoltrare il pacchetto. Nel processare un messaggio, la funzione `handleMessage()` segue il modello dei tempi illustrato nel Paragrafo 3.3.2 e per farlo chiama la funzione `processingTime()` che implementa la funzione del tempo di processing. Tale funzione chiama la funzione `procTimePar()` della clas-

[3]Tale evento arriva mediamente ogni β secondi, come illustrato nel paragrafo 3.3.1.

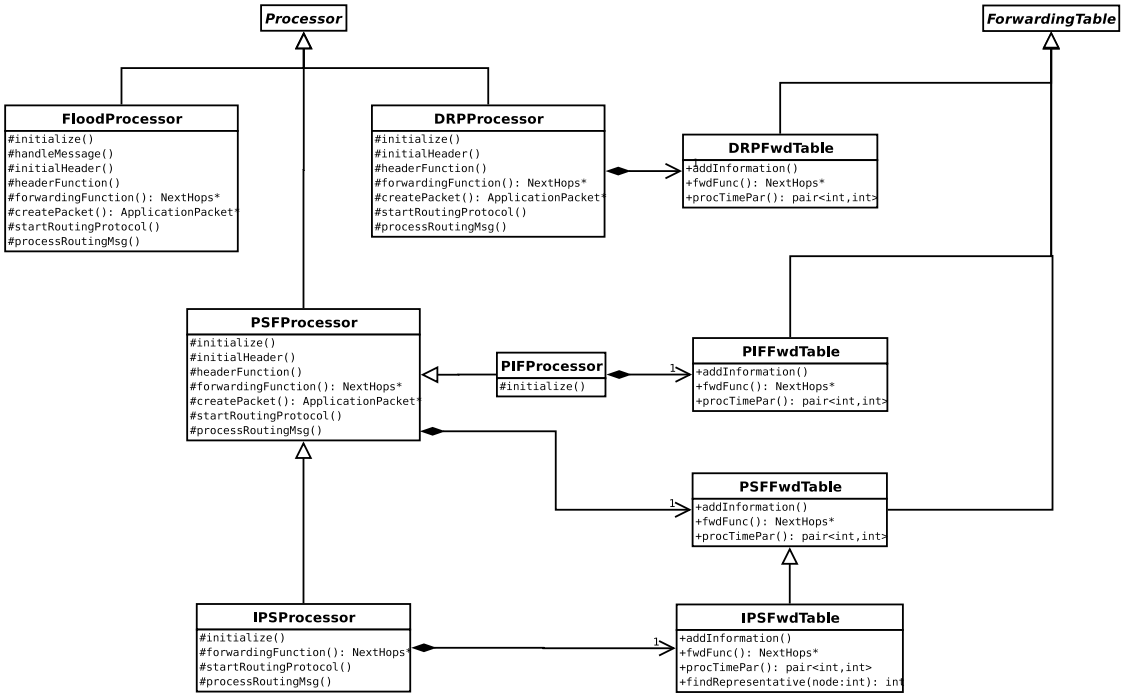


Figura 4.8: Class Diagram UML della gerarchia delle classi.

se `ForwardingTable` che restituisce il numero di interfacce ed il numero totale di vincoli necessari per il calcolo del tempo di processing.

A seconda dello schema di routing implementato le varie specializzazioni della classe `Processor` ridefiniscono alcune delle funzioni descritte. Tipicamente la funzione `handleMessage()` non viene ridefinita, in quanto implementa il comportamento di alto livello comune ad ogni processor. Unica eccezione la classe `FloodProcessor` che, seguendo una logica totalmente differente da quella degli altri schemi di routing, ridefinisce anche la funzione `handleMessage()`. La Figura 4.8 mostra il diagramma UML della gerarchia delle classi `Processor` e `ForwardingTable`, in essa si usa la convenzione per cui i metodi che vengono ridefiniti nelle specializzazioni sono mostrati, mentre quelli ereditati no.

4.2.2 I Pacchetti

I pacchetti implementati nella simulazione sono di tre tipi: i pacchetti scambiati nella fase di costruzione delle tabelle di routing unicast, quelli scambiati nella fase di costruzione delle tabelle di forwarding e i pacchetti applicativi, scambiati durante la parte della simulazione che effettivamente interessa ai fini del presente lavoro. I primi due tipi di pacchetti vengono, come detto precedentemente, affrontati nel Paragrafo 4.2.3.

I pacchetti applicativi seguono la gerarchia illustrata in Figura 4.9. Innanzi-

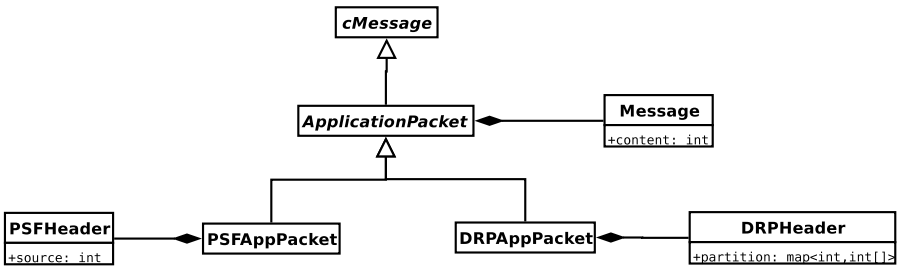


Figura 4.9: Class Diagram UML della gerarchia dei messaggi.

tutto ogni classe che rappresenta un pacchetto in `OMNet++` deve obbligatoriamente estendere la classe `cMessage`. Tale classe fornisce alcuni campi che tutti i pacchetti possiedono, come il tempo della simulazione in cui il pacchetto è stato creato, oppure la dimensione del pacchetto. Un generico pacchetto applicativo è implementato dalla classe `ApplicationPacket` che è stata implementata come una classe astratta, per garantire la necessaria generalizzazione all'interno della classe `Processor`. Tale classe contiene una classe `Message`, che implementa il contenuto di un messaggio, che nel presente contesto è un intero[4]. I pacchetti usati dagli schemi PSF, iPS, PIF e Flood sono istanze della classe `PSFAppPacket`, mentre lo schema DRP usa pacchetti del tipo `DRPAppPacket`. La differenza tra i due tipi di pacchetti sta solo nella classe che ne definisce l'header. La classe `PSFHeader` ha come informazione contenuta nell'header un intero, contenente l'identificativo della sorgente del messaggio, mentre la classe `DRPHeader` contiene la funzione di partizione dei nodi riceventi in una opportuna struttura dati.

Si fa notare che i pacchetti hanno una dimensione espressa in byte, che è la somma della dimensione del messaggio applicativo, definita nel file di configurazione, e la dimensione dell'header. La dimensione dell'header è di 4 byte per il `PSFHeader`, cioè la dimensione di un indirizzo di rete, mentre per il `DRPHeader` la dimensione varia in base alla dimensione della partizione contenuta nell'header. In particolare per ogni identificativo di rete presente nella partizione vengono sommati 4 byte, più altri 4 byte per ogni partizione presente (per modellare l'overhead della struttura dati utilizzata).

4.2.3 I Protocolli di Routing

Nei paragrafi precedenti si è accennato alle funzioni e ai messaggi che implementano i protocolli di routing. Essi vengono affrontati molto brevemente, in quanto tali

[4]Si è scelta questa struttura, invece che inserire direttamente un intero nella classe `ApplicationPacket`, nella prospettiva di ampliamenti futuri alla simulazione, per simulare contesti con un modello dei messaggi più complesso.

protocolli non hanno altro scopo se non quello di riempire le tabelle di forwarding utilizzate dagli schemi di routing content-based. Pertanto i protocolli in questione sono stati implementati nella maniera più sbrigativa possibile e non hanno nessuna pretesa di simulare protocolli di routing reali, che necessitano di un livello di ottimizzazione ben maggiore. La scarsa importanza data ai protocolli di routing deriva dal fatto che le simulazioni presentate hanno, come detto in precedenza, lo scopo di confrontare gli schemi di routing a regime in uno scenario statico, in cui cioè le sottoscrizioni dei nodi restano stabili così come le latenze dei link. Tale assunzione deriva dal fatto che ciò su cui ci si vuole concentrare è la bontà dello schema di routing in se e non dell'accoppiata schema di routing e protocollo di routing.

Si fa notare che si è scelto comunque di implementare tali protocolli come protocolli di rete, non utilizzando soluzioni centralizzate (come ad esempio l'impiego di un modulo che tramite chiamate di funzione riempisse le tabelle dei processor) sia in prospettiva di ampliamenti al modello che includano la simulazione di tali protocolli sia in vista di una esecuzione delle simulazioni in un ambiente distribuito. In quest'ultimo caso infatti `OMNet++` non permette che vi siano chiamate di funzione tra moduli che sono simulati su macchine diverse e quindi tutta l'informazione scambiata tra i nodi deve passare attraverso messaggi che viaggiano sui link.

4.2.4 Le Misure

Per raccogliere le misure della simulazione è stato invece scelto un approccio centralizzato[5]. Pertanto è stato implementato un apposito modulo `Statistics` che si incarica di raccogliere tutte le misure della simulazione e di memorizzarle in memoria di massa utilizzando le classi ausiliarie fornite da `OMNet++`. Il modulo `Statistics` è un simple module, non collegato alla rete, la cui classe presenta l'interfaccia mostrata in Figura 4.10. Tale interfaccia consente ai moduli `Processor` di chiamare una funzione del modulo `Statistics` per comunicargli ogni evento locale che avviene. In questo modo il modulo `Statistics` non solo raccoglie tutte le misure necessarie, ma implementa anche una sorta di "oracolo" che vede dall'alto tutto ciò che succede nella rete ed è in grado di dire se tutti i messaggi giungono veramente a tutti i destinatari interessati, funzionalità quest'ultima essenziale per verificare la correttezza dell'implementazione degli schemi di routing.

La classe `Statistics` si occupa anche di terminare la simulazione. La simu-

[5]Tale affermazione potrebbe sembrare in contrasto con quanto sostenuto nel paragrafo precedente, ma, a differenza del caso dei protocolli di routing, in una simulazione distribuita il modulo che raccoglie le statistiche può essere replicato su ogni macchina che partecipa alla simulazione e le misure raccolte possono poi essere aggregate al termine della simulazione.

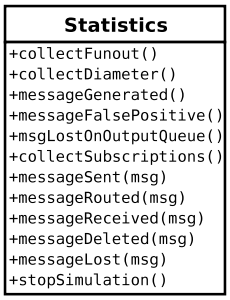


Figura 4.10: Class Diagram UML della classe `Statistics`.

lazione può essere terminata in due modi: impostandone la durata massima o facendola terminare da `Akaroa2` (vedi Paragrafo 4.5). In entrambi i casi è la classe `Statistics` che si occupa di far terminare la simulazione in maniera appropriata.

4.3 I Parametri del Modello

Durante l'implementazione sono anche stati scelti i parametri della simulazione. Tali parametri sono stati implementati in modo da poterli impostare nel file `omnetpp.ini`. Come detto in precedenza, ciò permette di poter cambiare i valori dati a tali parametri agevolmente, senza dover ricompilare la simulazione e con il vantaggio di poter intervenire in un unico punto su tutti i parametri del modello. Tali parametri sono elencati qui di seguito:

- il nome dello specifico modulo `Processor` da utilizzare. Tale parametro determina quindi lo schema di routing che viene simulato;

- capacità delle code in entrata: rappresenta la capacità delle code associate concettualmente alle interfacce di input del processor. La capacità della coda di ingresso del processor viene calcolata moltiplicando tale valore per il numero di interfacce del processor;

- capacità delle code in uscita;

- la topologia da simulare: ogni topologia disponibile ha un nome, tale parametro specifica il nome della rete che si vuole simulare;

- l'ampiezza della distribuzione zipf e della distribuzione uniforme con cui vengono generati rispettivamente le sottoscrizioni ed i messaggi applicativi;

- il numero di sottoscrizioni massimo per processor: indica il numero di sottoscrizioni che vengono generate per ogni processor. Le sottoscrizioni vengono generate da una zipf, pertanto è possibile che siano generate sottoscrizioni

identiche e il numero reale di sottoscrizioni sia minore del valore specificato da tale parametro;

- tasso medio di generazione dei messaggi: indica il parametro della distribuzione esponenziale che modella il tempo di generazione dei messaggi. Tale valore è in sec/msg;

- il numero di vincoli di cui è composta una sottoscrizione;

- la dimensione di un messaggio, espressa in byte. Come detto precedentemente, la dimensione del pacchetto che viaggia sulla rete è data dalla somma di questo valore, più il valore della dimensione dell'header del pacchetto, che viene assegnato dal processor a seconda del tipo di header;

- i generatori di numeri casuali utilizzati: tale parametro indica quali sono i generatori di numeri casuali che vengono utilizzati. Ogni generatore di numeri casuali utilizza lo stesso algoritmo di generazione dei numeri, ma produce stream di numeri generati differenti. Ogni generatore è identificato con un numero intero;

- la durata della simulazione, espressa in secondi.

In Appendice A.1 sono mostrati i valori tipici assegnati a tali parametri nella simulazioni presentate.

4.4 Le Reti Utilizzate

Come detto precedentemente gli schemi di routing vengono simulati su reti di dimensioni geografiche, create utilizzando `BRITE`. Innanzitutto, affrontando tale argomento occorre precisare che gli schemi di routing non vengono simulati su una overlay network, ma vengono simulati concettualmente a livello di rete. Pertanto la topologia generata da `BRITE` stabilisce esattamente come sono posizionati e collegati tra loro i soli processor.

Le reti che sono state impiegate nelle simulazioni sono reti generate sul modello di reti di Autonomous System (AS). Pertanto i nodi delle reti simulate sono in realtà degli AS. Ciò è coerente con il modello presentato, in quanto i processor simulano il comportamento di numerosi host e possono pertanto essere visti come degli AS. Inoltre l'unico modo per simulare la topologia di Internet è quello di simulare una rete di AS, in quanto la stessa Internet è costituita da tanti AS connessi tra loro.

I parametri dati a BRITE per generare le topologie delle reti simulate sono stati presi da [10] e da [11]. In [10] vengono confrontati alcuni generatori di topologie, tra cui BRITE, per verificare quale tra questi riesce a generare la topologia che si avvicina maggiormente alla topologia di due reti di ISP reali: il backbone continentale di AT&T US, da 154 nodi, ed una più piccola rete di ricerca tedesca (DFN G-Win) da 30 nodi. In [11] invece, vengono ancora confrontati diversi generatori di topologie, ma senza far riferimento a reti reali. In questo caso le reti che vengono generate sono confrontate tra loro per vedere quale si avvicina maggiormente alla topologia di Internet e vengono generate reti che vanno da alcune centinaia fino ad alcune migliaia di nodi.

Seguendo i parametri e le indicazioni fornite negli articoli citati, sono state generate cinque topologie che sono state utilizzate nelle simulazioni: due di queste sono quelle presentate in [10], mentre le restanti tre sono reti, generate sul modello di quelle presentate in [11], da 300, 600 e 1000 nodi. Ogni topologia è stata generata da BRITE in un NED file. I NED file così prodotti sono stati elaborati da un tool Java, appositamente implementato, che ne ha estratto il minimo albero ricoprente, secondo l'algoritmo di Kruskal. Le caratteristiche principali delle topologie generate sono mostrate in Tabella 4.1.

Si fa notare che le latenze dei link vengono anch'esse generate automaticamente da BRITE a seconda della posizione dei nodi che collegano (esse variano da alcuni microsecondi a un massimo di poco meno di 4 millisecondi). La banda dei link è invece costante ed è stato scelto di porla a 155 Mbit/sec, in quanto velocità realistica dei collegamenti tra AS.

In Appendice A.2 vengono riportati i parametri con cui è stato configurato BRITE per generare le reti in questione.

Rete	**Nodi**	**Grado dei nodi**			**Diametro**
		Min	Max	Medio	
DFN	30	3	23	9	4
DFN-albero	30	1	6	1,933	11
ATT	154	2	25	3,961	9
ATT-albero	154	1	10	1,987	19
INET300	300	2	17	4	12
INET300-albero	300	1	8	1,993	28
INET600	600	2	81	3,99	12
INET600-albero	600	1	41	1,996	33
INET1000	1000	4	30	8	12
INET1000-albero	1000	1	9	1,998	67

Tabella 4.1: Caratteristiche delle reti simulate.

4.5 Akaroa2

Come visto nel Paragrafo 3.6, per dare una validità statistica ai risultati delle simulazioni è stato utilizzato `Akaroa2`[6]. `Akaroa2` è un software, che consente di fermare la simulazione quando la stima della media delle misure di interesse raggiunge un determinato livello di precisione ad un certo livello di confidenza.

Il funzionamento di `Akaroa2` è molto semplice ed è basato su delle semplici API che possono essere invocate dalla simulazione. Ogni volta che la simulazione raccoglie un campione di una delle misure da stimare, essa lo invia, tramite una chiamata di funzione, ad un componente di `Akaroa2`, detto analizzatore locale, che raccoglie le misure prodotte dalla simulazione, le analizza e, quando queste hanno raggiunto i livelli di precisione e confidenza desiderati, invia alla simulazione un segnale per farla terminare.

Se si utilizza `OMNet++`, non c'è nemmeno bisogno di preoccuparsi di quali siano le API di `Akaroa2`, in quanto `Akaroa2` si integra perfettamente con `OMNet++`. Pertanto è sufficiente dichiarare nel file `omnetpp.ini` quali sono le misure che si intende raccogliere per essere analizzate da `Akaroa2` e la simulazione gli invierà automaticamente i campioni relativi a tale misura.

`Akaroa2` è utile anche per velocizzare le simulazioni, in quanto esso mette a disposizione la possibilità di stimare la media della misura di interesse tramite l'approccio *Multiple Replications In Parallel* (*MRIP*). L'approccio MRIP consiste nel lanciare diverse simulazioni in parallelo, ognuna delle quali attinge da un differente generatore di numeri casuali, e raccogliere contemporaneamente le misure fornite da tutte le simulazioni, in modo che a parità di tempo venga raccolto un maggior numero di campioni. Il vantaggio consiste nel fatto che, essendo tali campioni causati da generatori di numeri casuali differenti, si possono considerare indipendenti. L'architettura di `Akaroa2` che consente di implementare tale approccio è mostrata in Figura 4.11: su ogni host[7] che partecipa alla simulazione viene lanciato un processo `akslave` che si occupa di raccogliere le misure prodotte dalla simulazione localmente. Periodicamente tale processo invia le stime locali di tali misure al processo `akmaster`, che può essere eseguito su un host a sé, oppure su uno degli host su cui vengono eseguite le simulazioni. Il processo `akmaster` raccoglie le varie stime locali e, quando esse hanno raggiunto il desiderato livello di confidenza e precisione, pone fine alle simulazioni.

`Akaroa2` per funzionare richiede di impostare alcuni parametri che vengono qui

[6] `Akaroa2` è un progetto del Department of Computer Science, University of Canterbury, Christchurch, New Zealand. Esso è disponibile gratuitamente per scopi accademici di ricerca all'url: *http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/*

[7] Gli host possono anche essere differenti processori appartenenti ad una stessa macchina.

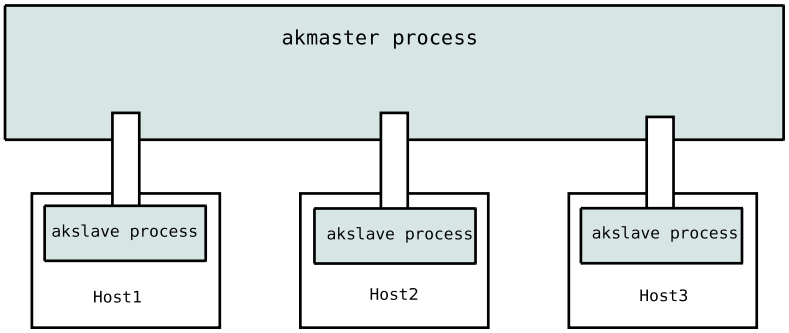


Figura 4.11: Architettura di `Akaroa2`.

di seguito elencati, e di cui ne vengono forniti i valori tipici utilizzati nel presente lavoro:

- il metodo di analisi che, come visto nel Paragrafo 3.6, è quello delle Batch Means;
- l'errore relativo massimo dell'intervallo di confidenza per la media delle misure raccolte, che nelle simulazioni effettuate varia tra l'1% e 5%;
- il livello di confidenza dell'intervallo di confidenza per la media, che nelle simulazioni effettuate varia tra 95% e 99%;

Inoltre il metodo delle Batch Means richiede di impostare la dimensione iniziale dei batch e la lunghezza della sequenza iniziale di batch means da analizzare per determinare la dimensione dei batch. Valori tipici utilizzati sono da 50 a 250 come dimensione iniziale dei batch e da 100 a 500 come lunghezza della sequenza iniziale.

4.6 L'ambiente di Esecuzione delle Simulazioni

Le simulazioni presentate sono state implementate ed eseguite su un sistema `Debian GNU/Linux 4.0` a 32-bit, con `kernel 2.6.18` e `gcc 4.1.2`. Le versioni dei software utilizzati sono:

- `OMNet++ 3.3`;
- `Akaroa 2.7.6`;
- `BRITE 2.1`.

Capitolo 5

I Risultati delle Simulazioni

Nel presente capitolo vengono mostrati i risultati ottenuti dalle simulazioni. Dopo una breve panoramica sul metodo con cui sono state effettuate le simulazioni, vengono analizzate le misure principali ottenute e vengono confrontate tra loro per i diversi schemi di routing. Per ogni misura vengono mostrati, tramite dei grafici significativi, i risultati ottenuti e tali risultati vengono poi analizzati con l'intento di comprenderne le indicazioni e le cause.

5.1 Il Metodo delle Simulazioni

Le simulazioni sono state effettuate sulle cinque reti presentate nel Paragrafo 4.4. Per brevità nel prosieguo ci si riferirà ad esse con i nomi DFN, ATT, INET300, INET600, INET1000[1]. Per ognuna di queste reti sono stati simulati cinque scenari applicativi differenti, caratterizzati ognuno da una diversa distribuzione della percentuale dei riceventi di un messaggio. In particolare, tali distribuzioni sono caratterizzate da una diversa moda della percentuale dei riceventi. Pertanto sono stati simulati scenari applicativi in cui la moda della percentuale dei riceventi è il: 25%, 10%, 5%, 1% e 0,5%. Ciò significa che, ad esempio, nello scenario caratterizzato dalla moda del 25%, alla fine della simulazione il maggior numero di messaggi generati sarà stato ricevuto dal 25% dei nodi della rete. Per la particolare forma di queste distribuzioni, nella quasi totalità dei casi la moda corrisponde anche alla media della distribuzione, pertanto nel suddetto scenario si può ipotizzare che mediamente un messaggio finisca al 25% dei nodi della rete. A titolo d'esempio in Appendice B vengono mostrate le distribuzioni degli scenari applicativi utilizzati per le simulazioni sulla rete INET300.

[1] si ricorda che le dimensioni di queste reti sono rispettivamente di 30, 154, 300, 600 e 1000 nodi

Una volta in possesso dei parametri necessari per simulare tutti e cinque gli scenari per ogni rete, si è potuto procedere alla raccolta delle misure delle simulazioni. Pertanto per ogni rete e per ogni scenario applicativo sono stati simulati i tre schemi di routing PSF, PIF, DRP ed i due protocolli su albero Flood e SFwd. Ognuna di queste simulazioni è stata eseguita sotto il controllo di `Akaroa2`. In particolare sono state affidate al controllo di `Akaroa2` le misure dei delay end-to-end. Questa scelta è stata fatta in quanto la media dei delay end-to-end è la misura più interessante per i confronti effettuati e pertanto si richiede, giustamente, che il suo valore sia calcolato in maniera sufficientemente accurata.

Per ogni simulazione svolta sotto il controllo di `Akaroa2` sono stati inoltre annotati il tempo simulato al quale la simulazione è stata fermata e la durata del transitorio. Dopodiché, per ogni rete e per ogni scenario è stata individuata la durata maggiore, tra tutti gli schemi, sia del transitorio che della simulazione. Una volta in possesso di questi dati, per ogni rete e per ogni scenario sono stati simulati nuovamente tutti gli schemi, ma stavolta tutti a parità di durata della simulazione e di transitorio, secondo i tempi precedentemente individuati[2]. In questo modo sono state raccolte anche tutte le altre misure, principali e ausiliarie, che richiedono tempi di simulazione uguali per tutti gli scenari. In tal modo si è potuto ottenere un confronto diretto, a parità di durata e quindi di traffico e di messaggi generati, tra i vari schemi.

Si fa notare che nelle simulazioni fin qui descritte il tasso di generazione dei messaggi è stato regolato per ogni rete in modo che la rete stessa fosse "scarica", ovvero in modo che nessuno degli schemi simulati potesse perdere pacchetti a causa del riempimento delle code dovuto al troppo traffico generato.

Infine si è proceduto a eseguire le simulazioni per calcolare la curva di throughput. Tale curva è stata calcolata solamente per le reti ATT e INET300, a causa dell'elevata quantità di tempo e capacità computazionale necessarie[3]. Per tali reti, per ogni scenario e per ogni schema di routing, sono state effettuate 25 simulazioni, aumentando ad ogni esecuzione il tasso di generazione dei messaggi. Le simulazioni sono state eseguite per un tempo sufficiente a garantire che le reti simulate fossero a regime, e per ogni simulazione sono state raccolte le misure del throughput a regime. In questo modo è stato possibile per ogni schema e per ogni scenario applicativo avere 25 punti con cui poter calcolare l'andamento della curva di throughput per poter operare i confronti.

[2]In realtà la durata della simulazione è stata aumentata di alcuni secondi, per maggiore sicurezza.

[3]Il tempo necessario per calcolare la curva di throughput sulla rete INET300 per tutti e cinque gli schemi, per un solo scenario ammonta, su una macchina equipaggiata con un moderno processore Intel Core 2 Duo T7200 a più di 50 ore.

Si fa notare che nel prosieguo lo scenario di riferimento sarà quello composto dalla rete INET300 con il 10% dei nodi della rete interessati alla maggior parte dei messaggi generati. Tale scenario è quello che meglio rispecchia tutte le peculiarità emerse analizzando i risultati ottenuti su tutte le reti simulate e su tutti gli scenari. Eventuali "eccezioni" rispetto allo scenario di riferimento vengono mostrate laddove necessario.

5.2 I Delay

La prima misura analizzata sono i delay end-to-end, in quanto, come detto precedentemente, misura più importante per valutare le prestazioni di rete degli schemi di routing in questione. I grafici mostrati da Figura 5.1 a Figura 5.5 evidenziano, per ogni rete, i delay end-to-end per i cinque schemi di routing nello scenario in cui la percentuale media di riceventi per messaggio generato è del 10%[4]. Nei grafici, partendo dal basso, le barrette orizzontali indicano, nell'ordine, il valore minimo misurato, il 5° percentile, la media, il 95° percentile e il valore massimo. In Figura 5.6 inoltre si può vedere l'andamento della media dei delay, in funzione del numero di nodi della rete.

Dai grafici in questione si possono già iniziare a cogliere alcune indicazioni significative. Ad esempio, si vede chiaramente come, aumentando la dimensione della rete, il DRP ottenga dei risultati molto inferiori dal punto di vista dei delay end-to-end rispetto a tutti gli altri schemi. Se infatti per la rete DFN i valori dei delay del DRP sono ancora confrontabili con quelli degli schemi PSF e PIF e

[4]Per quanto riguarda i delay end-to-end ci si limiterà ad analizzare lo scenario col 10% di riceventi, in quanto su differenti scenari non sono emerse particolari differenze.

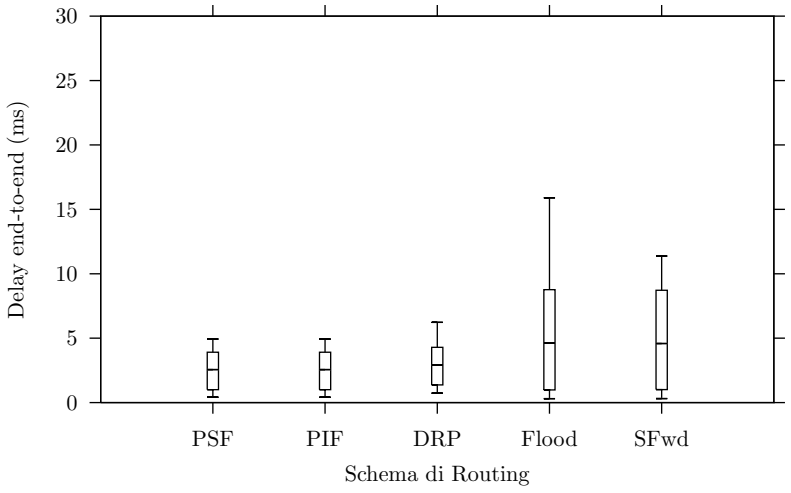


Figura 5.1: Comparazione dei delay end-to-end per la rete DFN.

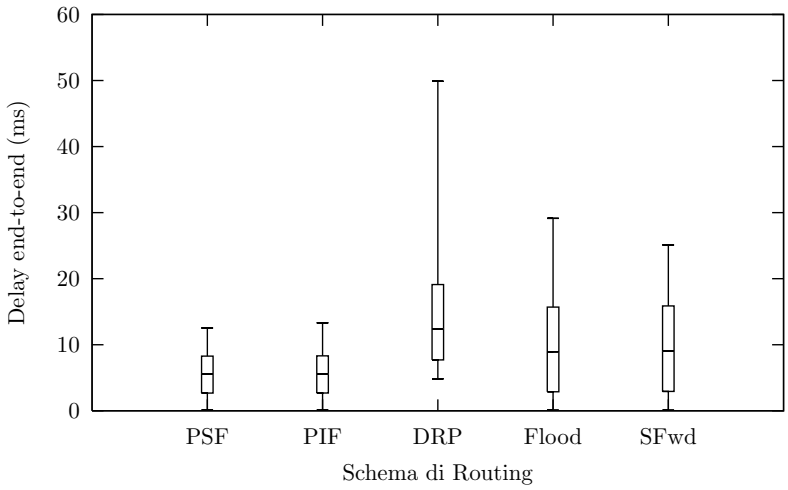


Figura 5.2: Comparazione dei delay end-to-end per la rete ATT.

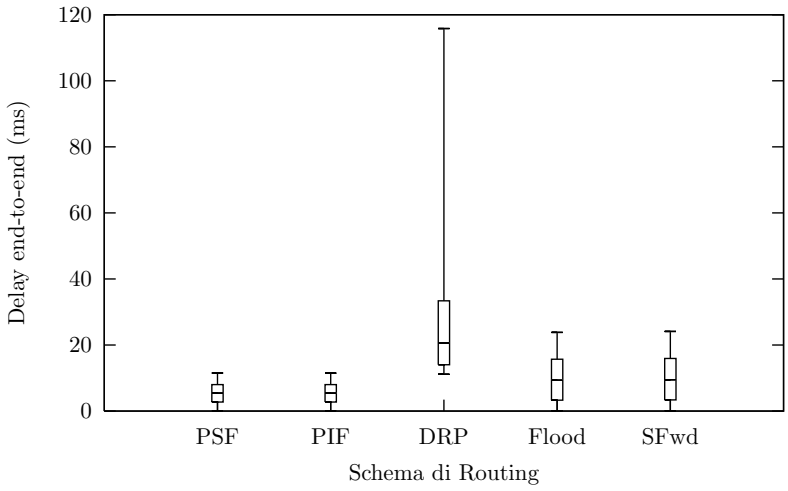


Figura 5.3: Comparazione dei delay end-to-end per la rete INET300.

inferiori a quelli dei protocolli Flood e SFwd, già per la rete ATT tali valori iniziano ad essere i peggiori. A partire dalla rete INET300 il 5° percentile del DRP risulta a pari livello, o maggiore del 95° percentile degli altri schemi e, se confrontato con i soli schemi PSF e PIF, il valore minimo dei delay del DRP risulta sempre superiore al 95° percentile di tali schemi. La causa di una prestazione così inferiore del DRP va imputata al tipo di funzione di forwarding utilizzata. Nel DRP infatti, ogni processor sorgente calcola, per ogni nuovo messaggio che deve immettere nella rete, la partizione iniziale analizzando la tabella *pred*() che contiene i predicati di tutti i nodi della rete. Il DRP deve quindi confrontare se un nuovo messaggio generato è selezionato dal predicato di un nodo per tutti i nodi e, all'aumentare dei nodi della rete, tale operazione diventa evidentemente troppo costosa in termini computazionali e rallenta eccessivamente i processor. Teoricamente, la pesantezza del calcolo della partizione iniziale dovrebbe essere mitigata dalla velocità con cui

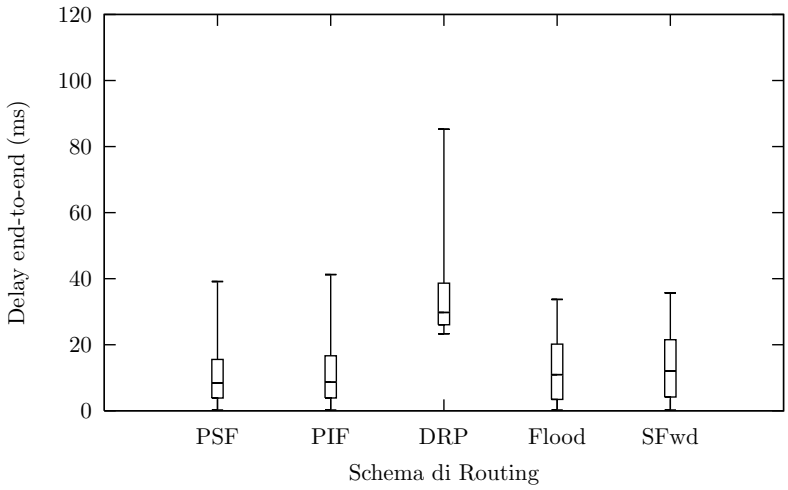


Figura 5.4: Comparazione dei delay end-to-end per la rete INET600.

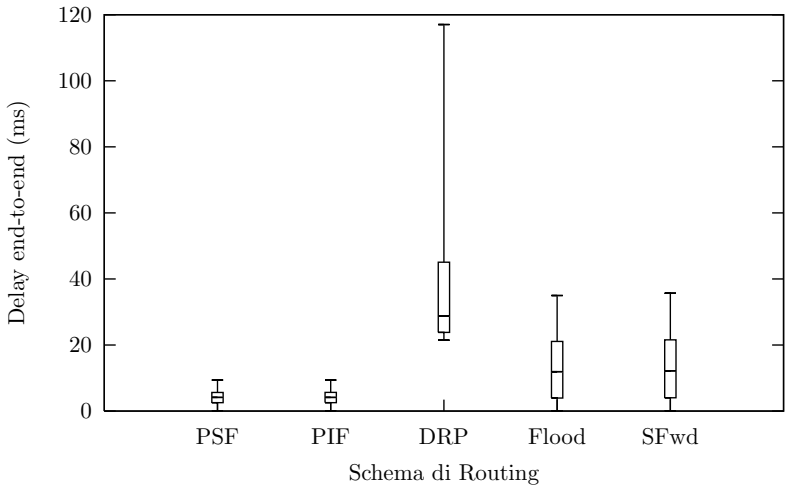


Figura 5.5: Comparazione dei delay end-to-end per la rete INET1000.

i processor intermedi eseguono il forwarding dei messaggi in ingresso, ma, come si vede dai grafici, nella pratica la lentezza del calcolo della partizione iniziale incide pesantemente sui valori dei delay end-to-end. Anche perché, sebbene il forwarding per i processor intermedi sia estremamente veloce, i pacchetti devono comunque restare nelle code mentre aspettano che il processor abbia calcolato la partizione iniziale per un nuovo messaggio. Una ulteriore conferma del degradare delle prestazioni del DRP all'aumentare dei nodi della rete si ha osservando il grafico di Figura 5.6. Per rendere l'entità dei valori in questione si fa notare che, dai risultati ottenuti, per la rete ATT i valori della media dei delay end-to-end del DRP sono superiori di circa il 120% rispetto a quelli del PSF, nelle reti INET300 e INET600 l'incremento raggiunge il 280% circa, fino ad arrivare alla rete INET1000 dove la media dei delay è superiore del 600% circa a quella del PSF.

Analizzando il comportamento degli altri schemi di routing si nota invece una

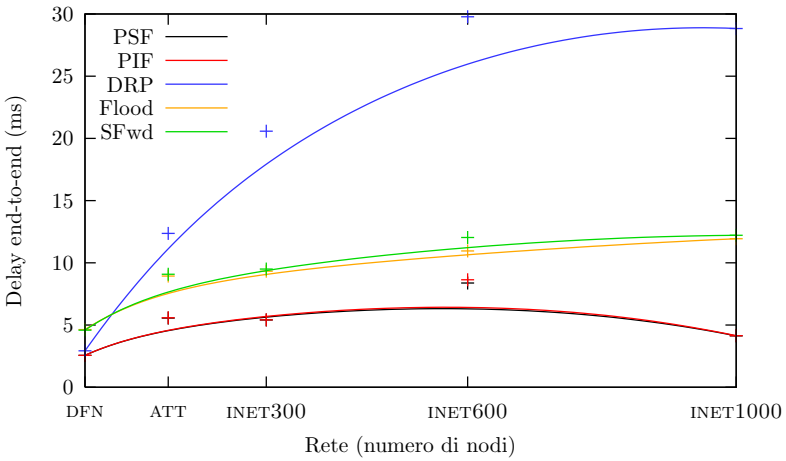


Figura 5.6: Andamento della media dei delay end-to-end.

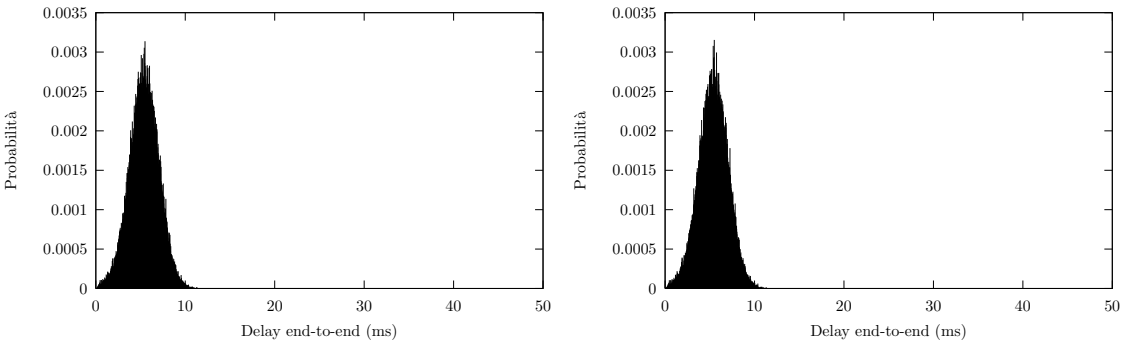
certa omogeneità nei risultati al variare delle reti. Innanzitutto si nota come gli schemi PSF e PIF siano molto simili tra loro. Essi utilizzano infatti due funzioni di forwarding molto simili, basate sul fatto che i predicati dei nodi sono associati alle interfacce. I falsi positivi generati dal PIF, così come il maggior numero di predicati associati alle interfacce[5] non sembrano incidere eccessivamente sui valori della media dei delay end-to-end: dalle misure raccolte risulta che, nella maggior parte degli scenari, tali valori sono superiori a quelli del PSF in percentuali minori dell'1%, tali da rendere la differenza trascurabile. Tuttavia, in alcuni scenari caratterizzati da una percentuale media di riceventi molto bassa (e pertanto in cui i nodi hanno predicati molto diversi tra loro), la maggiore quantità di predicati associati alle interfacce influisce in maniera più marcata, provocando una media dei delay fino al 10% superiore al PSF. La quantità di falsi positivi prodotta dal PIF invece non sembra incidere più di tanto in quanto nella misurazione dei delay la rete è relativamente scarica.

Un'ulteriore osservazione si può fare su come gli schemi PSF e PIF siano costantemente migliori degli schemi Flood e SFwd. I delay medi del Flood infatti sono risultati mediamente il 70% superiori a quelli del PSF, tranne che per la rete INET1000, dove l'incremento è risultato essere del 180%. Queste prestazioni peggiori sono dovute al fatto che tali schemi operano su un albero della rete, pertanto mediamente un pacchetto sull'albero deve attraversare più nodi per giungere a destinazione. Questa spiegazione è supportata dal fatto che proprio dove la differenza tra il diametro della rete e quello dell'albero è maggiore, ovvero per la rete INET1000, la differenza di prestazioni risulta anch'essa maggiore. A parità di albero invece il protocollo SFwd ottiene dei valori della media dei delay mediamente

[5]Si ricorda che il PIF non differenzia le interfacce a seconda della sorgente, pertanto le interfacce hanno mediamente associati più predicati rispetto al PSF.

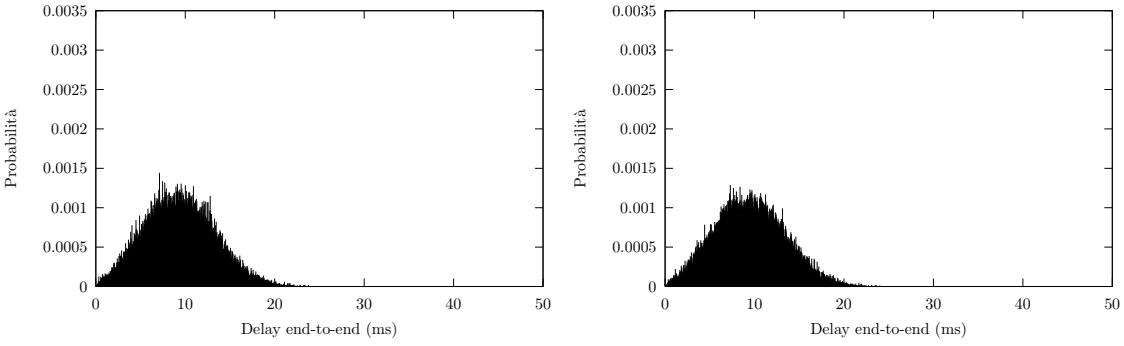
superiore del 20% rispetto al Flood. I processor del Flood infatti inoltrano i messaggi in un tempo nullo, mentre il SFwd deve confrontare i messaggi con i predicati associati alle proprie interfacce. Anche in questo caso, come per il PIF, il maggior traffico generato dal Flood non sembra incidere più di tanto, molto probabilmente a causa del fatto che la rete è relativamente scarica.

Si può notare inoltre come l'andamento delle medie dei delay non sia direttamente legato alla dimensione della rete (cioè al numero di nodi) per gli schemi in questione. Ad esempio, per gli schemi PSF e PIF la media dei delay della rete INET600 è circa il doppio di quella per la rete INET1000, nonostante quest'ultima abbia molti più nodi e nonostante il diametro delle due reti sia lo stesso. La spiegazione di questo risultato risiede nel fatto che la rete INET600 presenta un nodo con un grado pari ad 81. Tale nodo, avendo un così gran numero di interfacce è sicuramente un passaggio obbligato per molti cammini e fa da collo di bottiglia della rete, in quanto il tempo per calcolare la funzione di forwarding del PSF e del PIF è largamente influenzato dal numero di interfacce sulle quali bisogna operare i confronti.



(a) PSF (b) PIF

Figura 5.7: Distribuzione dei delay end-to-end (PSF - PIF).



(a) Flood (b) SFwd

Figura 5.8: Distribuzione dei delay end-to-end (Flood - SFwd).

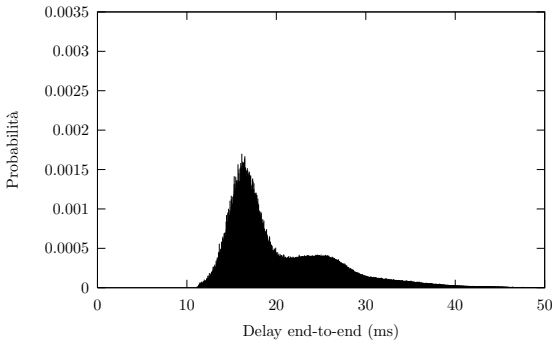


Figura 5.9: Distribuzione dei delay end-to-end (DRP).

Infine nelle figure da 5.7 a 5.9 sono mostrate le distribuzioni dei delay per lo scenario di riferimento. In esse si nota ancora più chiaramente la somiglianza tra PSF e PIF e tra Flood e SFwd, così come la netta inferiorità dello schema DRP, la cui distribuzione dei delay oltre che essere nettamente spostata destra ha anche una coda molto ampia.

5.3 I Pacchetti per Messaggio Generato

Nel trattare i risultati relativi ai PPM[6], l'approccio seguito è stato quello di confrontare tra loro gli schemi PIF, Flood, SFwd. Tali schemi infatti generano un numero di PPM non minimo per diversi motivi: il PIF in quanto genera falsi positivi, il SFwd poiché opera su una topologia ad albero, mentre il Flood sia perché opera su una topologia ad albero sia perché genera falsi positivi. Ciò che è stato confrontato è la percentuale media di PPM in più rispetto al valore minimo, ovvero quello degli schemi PSF e DRP (per i quali il numero di PPM coincide). Pertanto nei grafici da Figura 5.10 a Figura 5.12 viene mostrato il comportamento di questi tre schemi su tre differenti topologie, confrontando la percentuale di PPM in più rispetto al minimo.

Si nota immediatamente come il Flood generi un numero di pacchetti per messaggio generato molto maggiore rispetto al minimo, soprattutto per quegli scenari in cui la percentuale media di riceventi è bassa. In questi casi il numero di link attraversati da un messaggio è fino a 50 volte superiore rispetto al minimo. Ciò è naturale, in quanto il Flood manda ogni messaggio generato a tutti i nodi della rete, pertanto in scenari in cui il numero dei riceventi per messaggio è basso la differenza si nota molto di più. A questo si somma il fatto che il Flood opera su una topologia ad albero. L'effetto provocato dalla topologia ad albero si può notare

[6]Si ricorda che i PPM sono i pacchetti per messaggio generato, così come definito nel Paragrafo 3.5.1

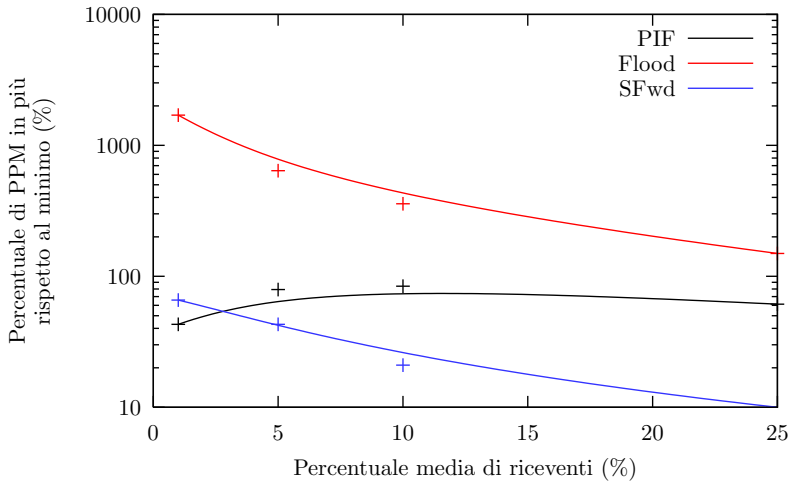


Figura 5.10: Andamento dei PPM rispetto al minimo sulla rete ATT.

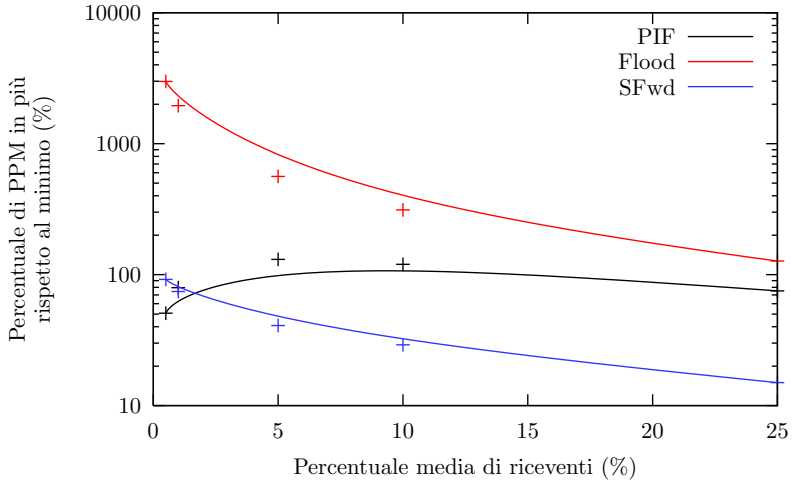


Figura 5.11: Andamento dei PPM rispetto al minimo sulla rete INET300.

analizzando l'andamento del protocollo SFwd, in quanto esso è a traffico minimo (sulla topologia ad albero) e i PPM in più sono dovuti solamente alla diversa topologia. Quello che si nota è che, all'aumentare della percentuale di riceventi, i PPM in più sono in percentuale meno significativi rispetto allo scenario con una percentuale minore di riceventi. Per esempio, sulla rete INET300 la percentuale in più di PPM passa da circa il 90% per lo scenario con lo $0,5\%$ medio di riceventi per messaggio generato, al 15% circa per lo scenario con il 25% di riceventi. In teoria ciò che avviene su una topologia ad albero è che il maggiore diametro della rete causa un maggior numero di PPM, essendo mediamente maggiore il numero di link tra una sorgente e un destinatario. Tuttavia l'incidenza di tale effetto si riduce in presenza di una alta percentuale di riceventi interessati. Infatti, se su una topologia ad albero un messaggio interessa molti nodi, i cammini dalla sorgente a tutti i riceventi si sovrappongono in molti punti, provocando un numero totale di

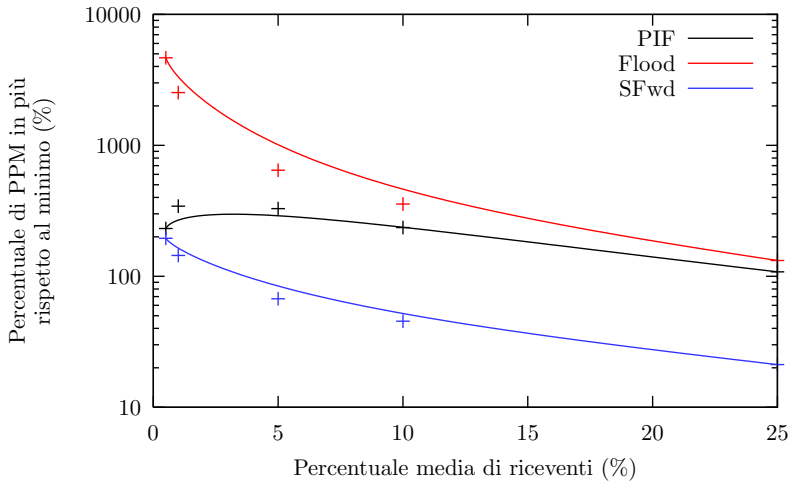


Figura 5.12: Andamento dei PPM rispetto al minimo sulla rete INET1000.

pacchetti generati minore.

Per quanto riguarda il PIF la percentuale di PPM in più rispetto al minimo non sembra essere legata allo scenario applicativo secondo un andamento ben definito. In effetti la causa dei falsi positivi prodotti dal PIF è nota, ma non è facile predire in quali scenari essa si materializza. Tali falsi positivi sono infatti il risultato di diversi fattori: la topologia della rete, lo scenario applicativo e persino il valore delle sottoscrizioni di ogni nodo. Si fa notare che per il PIF nelle simulazioni effettuate la percentuale di PPM in più varia tra il 5% ed il 350%. Il dato più stupefacente però si ottiene osservando come il PIF si comporti sempre peggio, rispetto al SFwd e come la situazione sia sempre più grave man mano che lo scenario applicativo è caratterizzato da una maggiore percentuale di riceventi. Ciò ci fa capire che l'incidenza dei falsi positivi del PIF non è trascurabile, in quanto uno schema su albero riesce a generare un traffico minore.

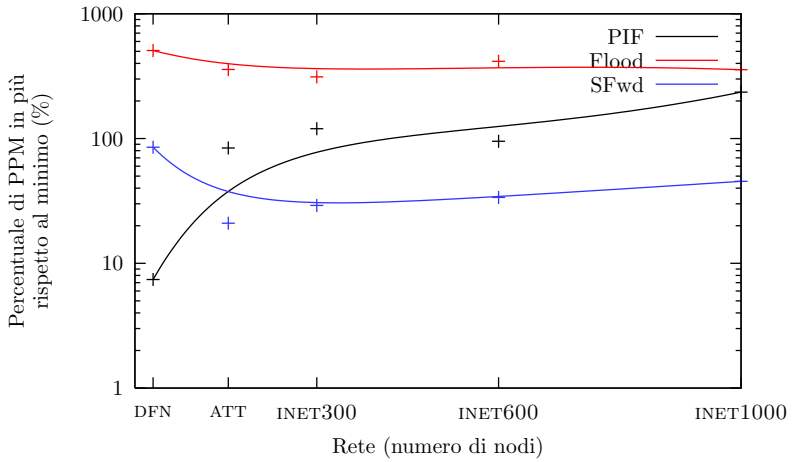


Figura 5.13: PPM rispetto al minimo col 10% dei riceventi.

Infine la Figura 5.13 mostra l'andamento della percentuale di PPM in più rispetto al minimo in funzione della dimensione della rete. Anche in questo caso, non si nota un andamento ben definito, a sostegno del fatto che una correlazione va ricercata piuttosto nello scenario applicativo o nella particolare topologia della rete.

5.4 Il Traffico Totale

Una misura strettamente legata al numero di pacchetti per messaggio generato è il traffico totale. Tale misura rappresenta il numero totale di pacchetti scambiati su tutta la rete per tutta la durata della simulazione. Nei grafici da Figura 5.14 a Figura 5.16 vengono mostrati i valori raccolti per le reti ATT, INET300, INET1000. In tali grafici le misure relative allo schema DRP non vengono mostrate, in quanto identiche a quelle dello schema PSF. Si fa notare inoltre che non ha senso confrontare tra loro le misure raccolte su reti differenti, in quanto ogni rete è stata simulata per una durata differente.

Dai grafici proposti si può innanzitutto osservare, da un punto di vista differente, come incidono quantitativamente le percentuali di PPM illustrate nel paragrafo precedente. Si può ad esempio notare come negli scenari caratterizzati da un'alta percentuale di riceventi il PIF generi una grossa mole di traffico in più rispetto al PSF e come il traffico generato dal PIF si avvicini al traffico generato dal Flood. Allo stesso modo si può notare come il SFwd si comporti meglio del PIF dal punto di vista del traffico generato. Per tutte queste osservazioni (e per altre simili che si possono fare) valgono le considerazioni illustrate nel paragrafo precedente.

L'analisi di questi dati consente però di porre l'attenzione anche su un altro

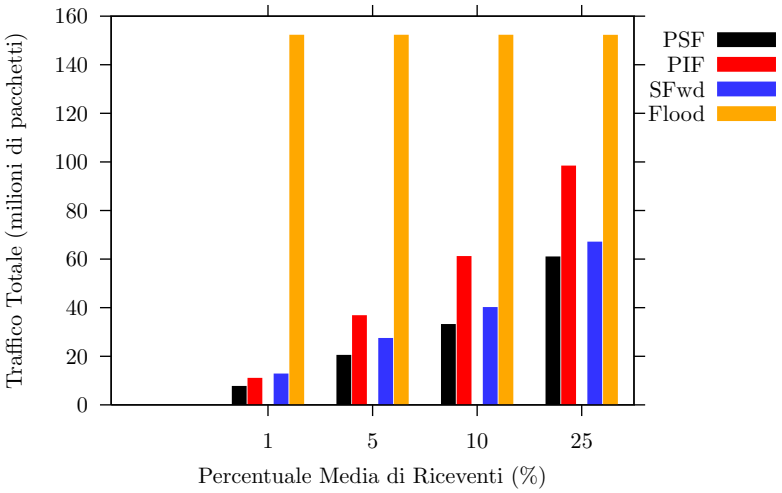


Figura 5.14: Traffico totale generato sulla rete ATT.

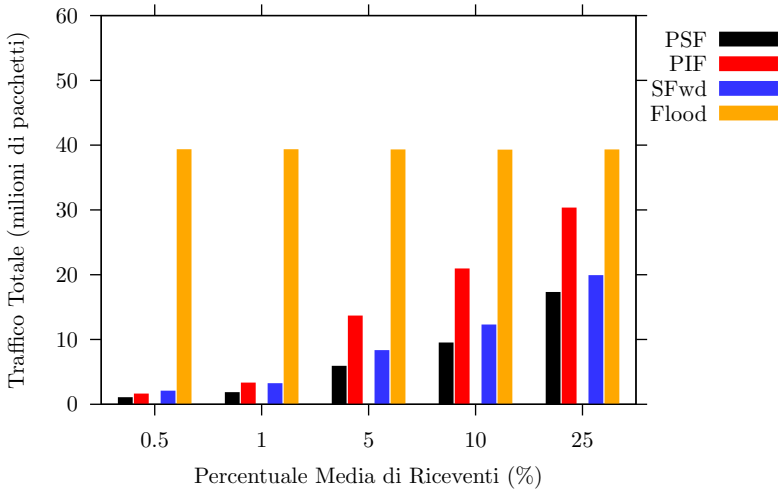


Figura 5.15: Traffico totale generato sulla rete INET300.

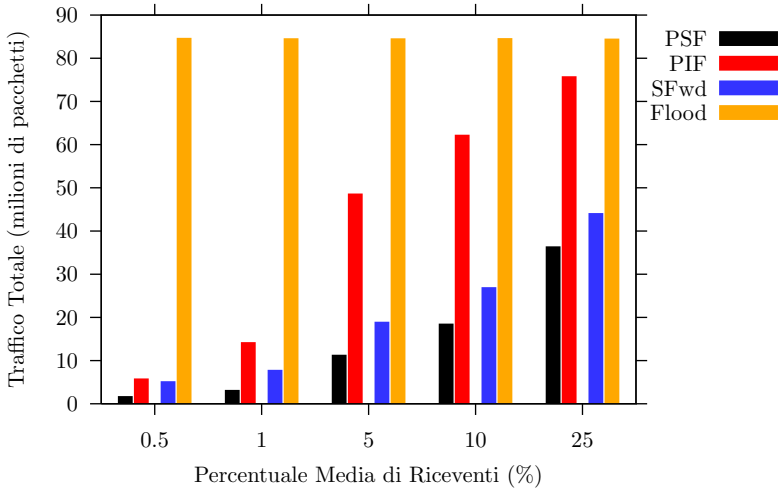


Figura 5.16: Traffico totale generato sulla rete INET1000.

aspetto da analizzare. Come si vede, nei grafici le coppie PSF - PIF e Flood - SFwd sono state disegnate staccate tra loro. Ciò che si vuole enfatizzare è in effetti un confronto interno a tali coppie. Il PSF infatti è uno schema a traffico minimo e anche il SFwd è a traffico minimo sulla topologia ad albero. Ciò significa che tali schemi non generano falsi positivi, ovvero che tutti i messaggi raggiungono solo i destinatari interessati oppure nodi che devono inoltrare tali messaggi a dei riceventi interessati. Dai grafici in questione emerge chiaramente a quanto ammonta il numero totale di pacchetti "inutili" scambiati sulla rete dal PIF (per differenza col PSF) e dal Flood (per differenza col SFwd). Tale quantità serve soprattutto per chiarire a quanto ammonta il numero di link mediamente percorsi da un falso positivo. Si fa notare infatti che una delle misure ausiliarie raccolte durante le simulazione è quella relativa al numero di falsi positivi. Nel paragrafo 3.5.2 un

falso positivo viene definito come un messaggio che raggiunge un processor il quale non è né interessato a tale messaggio né deve inoltrarlo ad altri processor. I falsi positivi tuttavia non rappresentano il totale dei pacchetti inutili. Il numero di falsi positivi, secondo la definizione appena data, rappresenta infatti solamente quanti sono i pacchetti inutili dei quali i nodi della rete si sono accorti. Per esemplificare il concetto, si ipotizzi il seguente scenario: il nodo A invia un messaggio al nodo B il quale pur non essendo interessato al messaggio lo inoltra al nodo C che non è né interessato al messaggio né lo deve inoltrare e lo scarta. In tale scenario solamente il nodo C può accorgersi che il messaggio inviatogli è inutile e pertanto in tale scenario il numero di falsi positivi viene incrementato di uno. In realtà però il numero di pacchetti inutili è pari a due, in quanto anche il pacchetto inviato da A a B non interessa nessuno. Il nodo B non può però conteggiare tale pacchetto come falso positivo, in quanto esso lo inoltra verso C e pertanto dal punto di vista di B tale pacchetto è "utile", in quanto è stato inoltrato.

In una rete reale risulta quindi difficile avere una misura di quanti sono i pacchetti inutili. Ciò perché in una rete reale i nodi si possono solo accorgere dei falsi positivi come prima definiti. Risulta quindi utile, per farsi un'idea dell'ammontare dei pacchetti inutili, sapere quanti sono mediamente i link attraversati da un falso positivo o detto in altre parole, quanti sono mediamente i pacchetti inutili scambiati sulla rete per ogni falso positivo rilevato. Nel caso delle simulazioni effettuate, avendo a disposizione il numero totale di pacchetti inutili calcolato come spiegato precedentemente, si è potuta calcolare facilmente tale quantità, dividendo il numero di pacchetti inutili per il numero di falsi positivi.

Il grafico di Figura 5.17 mostra l'andamento del numero di link attraversati mediamente da un falso positivo per la rete INET300. Come si può notare il

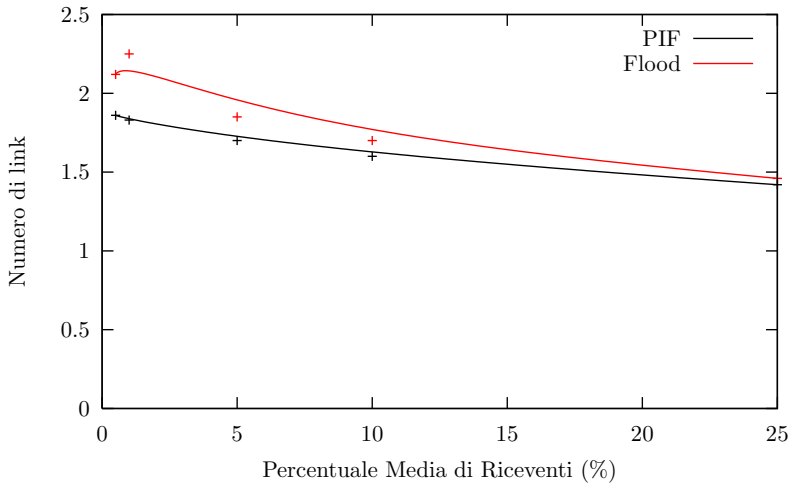


Figura 5.17: Numero medio di link attraversati da un falso positivo.

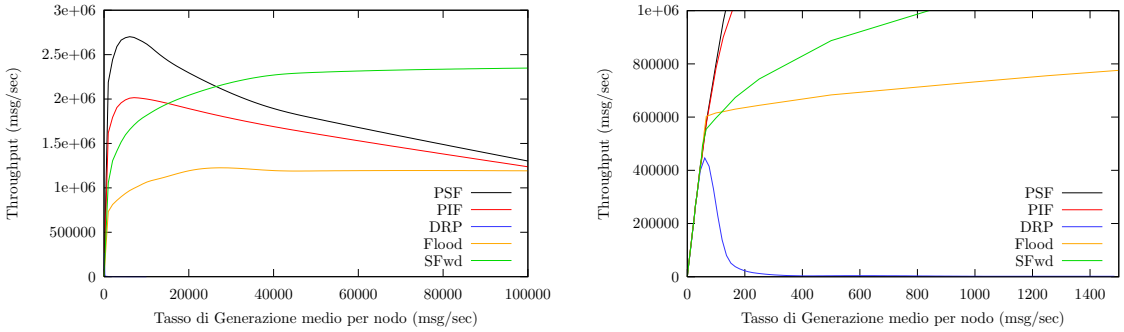
valore di tale quantità varia grosso modo tra 2 e 1,5, pertanto mediamente per ogni falso positivo conteggiato viene scambiato un ulteriore pacchetto inutile sulla rete. Dal grafico si nota anche come all'aumentare della percentuale di riceventi per messaggio il numero di link attraversati da un falso positivo si riduce. Tale considerazione è abbastanza ovvia, infatti essendoci meno nodi non interessati ad un determinato messaggio è meno probabile che un messaggio inutile passi attraverso un numero elevato di nodi non interessati.

Si fa infine notare come per il protocollo Flood i soli processor che possono rendersi conto del fatto che un messaggio è un falso positivo sono le foglie dell'albero. Infatti tali processor sono gli unici che non inoltrano messaggi e pertanto conteggiano come falsi positivi tutti i messaggi ricevuti ai quali non sono interessati. Tutti gli altri processor non foglia invece inoltrano sicuramente tutti i messaggi che ricevono.

5.5 Il Throughput

Nella Figura 5.18(a) viene mostrata la curva di throughput per lo scenario di riferimento. Si ricorda che il throughput rappresenta il numero di messaggi ricevuti globalmente da tutti i nodi della rete per unità di tempo, così come definito nel Paragrafo 3.5.1. Come già spiegato in tale paragrafo la curva di throughput da un'idea di come si comportano i vari schemi di routing quando vengono stressati sottoponendoli ad un elevato carico di traffico. Tale curva può fornire utili informazioni sulla capacità dei vari schemi di far fronte a picchi elevati di traffico e in generale da un'idea della rapidità con cui i vari schemi perdono pacchetti all'aumentare del traffico di rete[7].

[7]Si ricorda che per aumentare il traffico di rete, nelle simulazioni presentate si agisce sul tasso al quale i singoli processor generano messaggi e che tale tasso è lo stesso per tutti i processor.



(a) Generale (b) Particolare

Figura 5.18: Curva di throughput per lo scenario di riferimento.

L'andamento della curva di throughput è però importante soprattutto perché riassume tutte le peculiarità dei vari schemi e ne amplifica i pregi ed i difetti. Il suo andamento infatti è strettamente legato alle caratteristiche dei vari schemi analizzate nei paragrafi precedenti: in generale, schemi che presentano un delay end-to-end mediamente più basso, riescono a consegnare i messaggi più rapidamente e riescono quindi a consegnare più messaggi per unità di tempo, ottenendo un throughput mediamente più elevato. D'altra parte il risultato dei delay non tiene conto della quantità di traffico generato dai vari schemi, aspetto che incide invece sulla curva di throughput. Infatti tutto il traffico inutile generato dagli schemi non minimi, oltre ad avere l'effetto di saturare la banda ed aumentare quindi il numero di pacchetti persi sulle code in uscita, obbliga i processor a sprecare capacità computazionale per il filtering ed il forwarding di messaggi inutili.

Analizzando la Figura 5.18(a) si nota subito come da questo punto di vista il PSF sia lo schema migliore. Esso riesce infatti ad ottenere un throughput massimo superiore a quello di tutti gli altri schemi. Un'altra osservazione che si può fare riguarda l'andamento della curva di throughput dall'origine al throughput massimo. Nel caso del PSF e del PIF la curva ha, nella parte iniziale, un andamento molto ripido, indice del fatto che tali schemi si avvicinano al throughput massimo perdendo molti meno pacchetti rispetto ai protocolli Flood e SFwd. L'andamento della curva infatti indica come si comportano gli schemi all'aumentare del traffico. Se per uno schema la curva è quasi orizzontale significa che il numero di messaggi al secondo consegnati globalmente su tutta la rete resta più o meno costante, pertanto tutti i messaggi in più prodotti aumentando il tasso di generazione vengono persi. Si nota quindi come sia preferibile l'andamento del PIF piuttosto che quello del SFwd, in quanto quest'ultimo, pur ottenendo un throughput massimo più elevato, per raggiungere il suo throughput massimo ha perso molti più pacchetti.

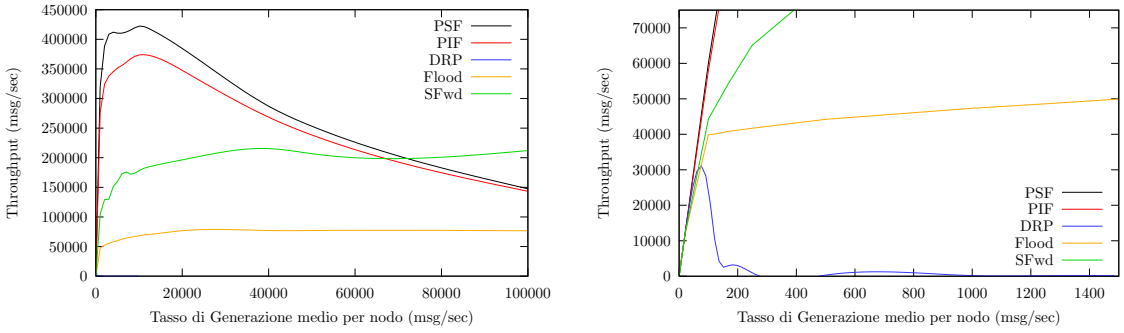
Si può notare inoltre come per gli schemi PSF e PIF la curva decresca dal massimo in poi, mentre per il Flood ed il SFwd resti più o meno costante. Tale comportamento è dovuto al fatto che nel PSF e nel PIF ad un certo punto i processor esauriscono la loro capacità computazionale, non riuscendo più a far fronte alla mole di messaggi generati, e iniziano a perdere molti più pacchetti sulle code in ingresso, provocando così una diminuzione del numero di messaggi ricevuti per unità di tempo. In generale ciò che succede è che quando il tasso di generazione dei messaggi è troppo elevato le code in ingresso sono piene di messaggi da immettere sulla rete. I processor non riescono a smaltire abbastanza velocemente tutti questi messaggi e pertanto i messaggi che arrivano dagli altri processor trovano le code in ingresso piene e vengono scartati. Nel Flood invece il carico computazionale è costituito solo dalla fase di filtering, pertanto i processor che adottano tale sche-

ma riescono sempre a processare tutti i messaggi in arrivo e tutti quelli generati. Tuttavia il Flood genera molto più traffico, pertanto esaurisce velocemente tutta la banda a disposizione perdendo tutti i pacchetti sulle code in uscita. Il particolare comportamento del SFwd è invece dovuto al fatto che esso genera molto meno traffico rispetto al Flood, pertanto satura la banda più lentamente, riuscendo a parità di pacchetti generati al secondo a consegnarne un numero più elevato. Inoltre esso ha una funzione di forwarding più leggera rispetto al PSF in quanto deve operare i confronti su un numero di interfacce minore (essendo la topologia ad albero caratterizzata da un grado medio dei nodi inferiore) e pertanto la capacità computazionale dei suoi processor è tale che essi riescono a processare abbastanza messaggi da saturare la banda. Conseguenza di ciò è che pertanto la curva non decresce dal massimo in poi.

Il minore rendimento del PIF rispetto al PSF è invece sicuramente imputabile alla maggiore quantità di traffico generato, dovuto ai numerosi falsi positivi.

La Figura 5.18(b) mostra invece un ingrandimento del grafico precedente in cui si vede il comportamento degenere del DRP. Il rendimento del DRP è nettamente inferiore rispetto agli altri schemi in quanto i suoi processor esauriscono subito la capacità computazionale. Infatti, come già osservato precedentemente, nel DRP la fase più pesante è quella di immissione di nuovi pacchetti nella rete e aumentando il tasso a cui i nuovi pacchetti vengono generati non si fa altro che peggiorare la situazione, in quanto i processor sono impegnati a calcolare la partizione per i nuovi messaggi e non riescono più a ricevere pacchetti, che vanno quindi persi sulle code in entrata.

La Figura 5.19 mostra invece l'andamento della curva di throughput per la rete INET300 con mediamente lo $0,5\%$ di riceventi per messaggio. Si nota innanzitutto come i valori del throughput siano inferiori rispetto al grafico precedente, ma ciò è abbastanza ovvio, in quanto i riceventi per ogni messaggio sono molti meno. Più



(a) Generale (b) Particolare

Figura 5.19: Curva di throughput per la rete INET300 con lo $0,5\%$ di riceventi.

interessante è invece notare l'incidenza del traffico generato dai vari schemi sul throughput. Se infatti l'andamento dei vari schemi presi singolarmente è abbastanza simile al caso precedente, ciò che è diverso è la distanza, in termini di throughput, tra loro. Si può notare infatti come nello scenario con lo $0,5\%$ di riceventi i protocolli Flood e SFwd siano più staccati dagli altri schemi rispetto al caso col 10% di riceventi e come invece il PIF sia più vicino al PSF. Una spiegazione di questo fatto si può avere confrontando i due grafici delle curve di throughput col grafico delle percentuali di PPM rispetto al minimo mostrato in Figura 5.11. Si nota ad esempio come i protocolli Flood e SFwd nello scenario con lo $0,5\%$ di riceventi generano una percentuale di PPM maggiore rispetto allo scenario col 10%. Tale differenza si riflette nella curva di throughput, in quanto il maggior traffico generato dagli schemi Flood e SFwd nello scenario con lo $0,5\%$ di riceventi aumenta la differenza di throughput rispetto al PSF, arrivando a saturare prima le code in uscita. Allo stesso modo il PIF nello scenario con lo $0,5\%$ di riceventi ha una percentuale di PPM rispetto al minimo inferiore rispetto allo scenario col 10% ed infatti nello scenario con lo $0,5\%$ ha una curva di throughput che si avvicina maggiormente a quella del PSF.

Infine in Tabella 5.1 sono mostrati, per i vari schemi analizzati, i valori del throughput massimo senza che vi siano perdite di pacchetti e del relativo tasso di generazione dei messaggi al quale tali valori sono stati ottenuti. I valori in questione corrispondo al punto in cui la retta della curva di throughput che parte dall'origine inizia a flettere. Dalla tabella si vede come per lo scenario col 10% di riceventi gli schemi su albero si comportino in maniera migliore. Evidentemente in tale scenario e ad un tale tasso di generazione dei messaggi, la rete è relativamente scarica e pertanto il maggiore traffico prodotto da tali schemi non incide in maniera negativa sul throughput. Viene quindi premiata la maggiore velocità della funzione di forwarding del Flood rispetto a tutti gli altri schemi e quella del SFwd rispetto al PSF e al PIF. Tuttavia, se si osserva la Figura 5.18(b) si nota come, pur essendo vero che le curve di throughput del Flood ed del SFwd iniziano a flettere dopo il PSF ed il PIF, la flessione avviene in maniera più netta. Pertanto anche in

Schema	**riceventi 10%**		**riceventi 0.5%**	
	rate (msg/s)	throughput (msg/s)	rate (msg/s)	throughput (msg/s)
PSF	45,45	419679	100	**59246**
PIF	41,67	386037	83,33	49776
DRP	18,18	168305	52,63	31898
Flood	58,82	**544918**	58,82	35205
SFwd	52,63	491327	62,5	37382

Tabella 5.1: Throughput massimi senza perdite (rete inet300).

questo scenario il PSF ed il PIF garantiscono un comportamento più stabile, in quanto aumentando di poco il tasso di generazione dei messaggi essi perdono pochi pacchetti, mentre gli schemi su albero ne perdono molti di più.

Analizzando infine lo scenario con lo $0,5\%$ di riceventi si osserva che tutto torna alla "normalità", con i throughput senza perdite dei vari schemi che seguono l'andamento delle rispettive curve. Come già sottolineato in precedenza, in questo scenario il maggiore traffico generato dagli schemi su albero incide negativamente sul throughput e di conseguenza anche su quello senza perdite.

5.6 Indicazioni dei Risultati

I risultati ottenuti dalle simulazioni e la loro analisi, illustrati nei paragrafi precedenti, danno delle indicazioni molto chiare. Innanzitutto lo schema di routing content-based migliore tra quelli analizzati per quanto riguarda le prestazioni di rete è sicuramente il PSF. Esso garantisce delay end-to-end mediamente minori rispetto agli altri schemi, è a traffico minimo in quanto non genera falsi positivi ed è di conseguenza anche lo schema che garantisce migliori prestazioni in termini di throughput.

Il PIF ha delle prestazioni molto simili al PSF per quanto riguarda i delay end-to-end e delle prestazioni vicine a quelle del PSF in termini di throughput. Tuttavia esso genera molto più traffico rispetto al PSF a causa dei falsi positivi. L'aspetto peggiore del PIF è che il numero di falsi positivi generato varia a seconda della topologia e del valore delle sottoscrizioni dei nodi e pertanto risulta quasi impossibile tenere sotto controllo o rendere prevedibile la quantità di traffico generato a causa dei falsi positivi.

Per quanto riguarda gli schemi su albero, il Flood presenta generalmente dei delay end-to-end più elevati rispetto al PSF, ma comunque accettabili in quanto al massimo sono tre volte più alti. Esso ha lo svantaggio di generare una considerevole quantità di traffico inutile soprattutto in scenari caratterizzati da messaggi che interessano pochi destinatari e in tali scenari esso satura facilmente la banda a disposizione. Il SFwd invece riesce a garantire un traffico minimo sull'albero anche se le prestazioni in termini di delay end-to-end sono lievemente inferiori rispetto al Flood. Tuttavia l'impiego di tali protocolli, soprattutto se considerata la loro estrema semplicità, risulta accettabile in scenari caratterizzati da messaggi che interessano un'alta percentuale di riceventi interessati. In tali scenari infatti la quantità di traffico generata da tali protocolli e soprattutto dal SFwd si avvicina a quella del PSF. Ovviamente però operando su una topologia ad albero è più

facile che a parità di traffico sull'albero si possa saturare la banda a disposizione. Pertanto su reti caratterizzate da una ridotta disponibilità di banda, l'impiego del PSF risulta ancora una volta il più indicato.

I risultati ottenuti dal DRP non lasciano invece adito a dubbi, nonostante esso sia a traffico minimo, le pessime prestazioni della sua funzione di forwarding causano dei delay end-to-end molto più elevati rispetto a tutti gli altri schemi e di conseguenza un throughput nettamente inferiore, rendendo pertanto tale protocollo inadeguato già per reti la cui dimensione superi i 100 nodi. A tal proposito si fa notare però che la scelta di avere un'unica coda sia per i messaggi generati sia per i messaggi in arrivo tende a penalizzare particolarmente uno schema come DRP caratterizzato da un tempo necessario ad inoltrare i messaggi molto diverso da quello necessario per immettere nella rete un nuovo messaggio.

Una nota finale riguarda il consumo di memoria locale di tali schemi di routing. Come dimostrato in [3] e come intuitivamente ci si aspetterebbe, il consumo di memoria del PSF è in teoria molto più elevato del PIF, in quanto le tabelle di forwarding del PSF dividono i predicati a seconda della sorgente. Le misure relative alla occupazione di memoria effettuate nelle simulazioni presentate confermano questa ipotesi, indicando che il consumo di memoria locale medio (in termini di numero di sottoscrizioni presenti nelle tabelle) del PSF rispetto al PIF varia mediamente tra il 5% in più, per lo scenario con lo $0,5\%$ di riceventi, fino al 50% in più per lo scenario col 25% di riceventi, con una punta massima del 70% di sottoscrizioni in più in tabella per i processor della rete INET1000. Tali misure possono solamente dare un'indicazione molto approssimativa della situazione, in quanto il modello dei messaggi e delle sottoscrizioni adottato è molto semplice ed è per questo motivo che i risultati relativi alla occupazione di memoria delle tabelle di forwarding ottenuti non vengono analizzati approfonditamente così come fatto per le altre misure raccolte. Verosimilmente infatti nella realtà la situazione è più complessa, in quanto la complessità dei predicati logici consente di operare differenti tipi di compressione sui dati presenti nelle tabelle. Tuttavia l'indicazione data da tali risultati è che, negli scenari dove i messaggi interessano mediamente un'alta percentuale di nodi della rete, l'occupazione di memoria in più del PSF diventa considerevole.

A tal proposito risulta invece utile confrontare il numero di nodi sorgente presenti nelle tabelle dei processor del PSF con quelli dell'iPS. Questa misura è infatti attendibile, a differenza della precedente, in quanto la relazione di indistinguibilità tra i nodi sorgente, sulla quale si fonda l'iPS, è basata esclusivamente sulla topologia della rete. Pertanto i risultati ottenuti possono dare delle indicazioni utili per capire cosa può avvenire su una topologia reale. Dalle misure raccolte, risulta che

mediamente le tabelle dei processor dell'iPS contengono il 75% di nodi sorgente in meno rispetto al PSF, con punte per la rete INET1000 dell'80%. L'ottimizzazione dell'iPS sembra quindi portare dei vantaggi notevoli in termini di occupazione di memoria locale ai processor. Verosimilmente, come spiegato nel Paragrafo 3.3.2 le prestazioni di rete dell'iPS sono le medesime del PSF pertanto, alla luce dei risultati ottenuti sulla occupazione di memoria, lo schema iPS risulta essere il migliore tra tutti gli schemi di routing content-based analizzati nel presente lavoro.

Capitolo 6

Conclusioni

Nel presente lavoro di tesi sono stati analizzati e confrontati tramite simulazione quattro schemi di routing content-based, i quali sono ad oggi oggetto di ricerca nell'ambito della comunicazione content-based. Tali schemi di routing sono stati inoltre confrontati con due ulteriori protocolli che operano su una topologia di rete ad albero: un protocollo di subscription forwarding (SFwd) e un protocollo di flooding su albero (Flood). Di questi sei schemi analizzati, quattro (PSF, iPS, PIF e SFwd) basano la funzione di forwarding dei router sull'idea di associare le sottoscrizioni dei destinatari dei messaggi alle interfacce di rete attraverso le quali tali destinatari sono raggiungibili. Inoltre due (PSF e iPS) di questi quattro distinguono ulteriormente le sottoscrizioni associate alle interfacce in base ai mittenti dei messaggi da inoltrare. Uno schema di routing (DRP) basa invece la sua funzione di forwarding sull'idea di calcolare tutti i nodi interessati ad un messaggio al momento di immettere tale messaggio nella rete. Infine il protocollo di flooding su albero invia semplicemente i messaggi a tutti i nodi della rete.

L'ottica con la quale sono stati eseguiti i confronti è quella delle prestazioni di rete. Per operare i confronti è stato ideato un modello in grado di rappresentare al meglio la realtà da simulare, ovvero i router, i messaggi, le sottoscrizioni, le sorgenti dei messaggi, i destinatari dei messaggi, la rete e gli scenari applicativi. Sono state inoltre generate delle topologie realistiche di reti geografiche sulle quali simulare il comportamento degli schemi di routing e sono state decise delle misure da raccogliere durante le simulazioni per valutare le prestazioni di rete ed altri aspetti più marginali degli schemi in questione. Il modello e le reti sono stati in seguito implementati utilizzando un simulatore ad eventi discreti (`OMNet++`). Gli schemi in questione sono quindi stati simulati su cinque differenti topologie ed in cinque differenti scenari applicativi. Per dare credibilità ai risultati ottenuti sono stati impiegati degli strumenti statistici in grado di garantire che la durata delle

simulazioni fosse tale da produrre misure accurate.

I risultati ottenuti dalle simulazioni hanno decretato che globalmente le migliori prestazioni di rete sono state ottenute dagli schemi PSF (Per-Source Forwarding) e dalla sua versione ottimizzata, l'iPS (Improved Per-Source Forwarding) e hanno inoltre messo in luce aspetti positivi e negativi di tutti e sei gli schemi di routing analizzati.

Le indicazioni fornite da tali risultati servono innanzitutto per capire quali sono gli schemi di routing su cui concentrare gli sforzi di ricerca, nell'ottica di implementare in scenari reali tali protocolli.

Tuttavia le problematiche aperte restano ancora molte. Le nostre simulazioni non hanno toccato l'aspetto dei protocolli di routing, ovvero di come i vari nodi della rete si scambiano tra loro le informazioni necessarie agli schemi di routing per compiere l'inoltro dei messaggi dalle sorgenti ai destinatari interessati. Tali protocolli sono critici per le prestazioni degli schemi di routing analizzati, soprattutto in scenari dinamici in cui le sottoscrizioni e la topologia della rete variano continuamente. Sono quindi richiesti sforzi per ideare protocolli di routing efficienti e per analizzare come tali protocolli incidono sulle prestazioni di rete. Ulteriori sforzi poi sono richiesti per capire quale sia effettivamente il consumo di memoria locale ad ogni router delle strutture dati necessarie per eseguire la funzione di forwarding dei messaggi e quali siano le strutture dati più efficienti per ottimizzare tale consumo di memoria. Le nostre simulazioni infatti insinuano il sospetto che le strutture dati richieste dal PSF siano troppo dispendiose in termini di consumo di memoria e pertanto indicano come schema su cui orientare gli sforzi futuri l'iPS, caratterizzato da un minor consumo di memoria e le cui prestazioni di rete sono identiche a quelle del PSF. Tuttavia la relazione di indistinguibilità dei nodi sorgenti introdotta dall'iPS sembrerebbe complicare considerevolmente i protocolli di routing. Pertanto risulta necessario investigare quanto possa essere complicato un protocollo di routing per l'iPS e eventualmente se non sia più ragionevole utilizzare il PSF ed orientare gli sforzi verso una implementazione ottimizzata delle sue strutture dati, in grado di diminuirne l'occupazione di memoria.

Più in generale, nell'ottica di portare la comunicazione content-based su reti geografiche come Internet, poiché entrambi gli schemi iPS e PSF distinguono nelle loro tabelle di forwarding le sorgenti dei messaggi, risulta improponibile un loro impiego su una rete con moltissimi nodi. Servono quindi dei meccanismi efficienti per portare anche sulle reti content-based i concetti di subnetting e supernetting, tipici della struttura gerarchica di Internet, che hanno permesso al protocollo IP di scalare efficientemente su reti geografiche. In tal senso la direzione della ricerca sembra orientata verso l'impiego di *bloom filter* sui border router in grado di ve-

locizzare enormemente la funzione di forwarding, pur producendo dei falsi positivi che eventualmente vengono filtrati dai router delle sotto-reti. In tal senso sembra utile come passo successivo analizzare come si sposano gli schemi PSF e iPS con questo modello di comunicazione.

Infine, sebbene l'impiego dei protocolli PSF ed iPS a livello di rete, così come pensato nelle simulazioni presentate, rappresenta una sfida stimolante nell'ottica di rendere la comunicazione su Internet più efficiente e più adeguata alle esigenze di numerose applicazioni distribuite, tuttavia l'impiego più probabile, almeno nell'immediato futuro, per i protocolli di CBR resta quello su overlay network. Pertanto sembra opportuno analizzare il comportamento di tali schemi sopra un protocollo di trasporto come TCP o UDP. Si fa notare che l'architettura delle simulazioni presentate è stata progettata per poter facilmente essere estesa in tal senso[1].

Concludendo, il presente lavoro ha ristretto il numero di alternative possibili per la scelta di uno schema di routing content-based su cui puntare gli sforzi di ricerca. Tuttavia la strada da percorrere per portare il Content-Based Routing su reti geografiche, quale la stessa Internet, risulta ancora lunga anche se tutt'altro che impraticabile.

[1] L'architettura è stata pensata per essere estesa utilizzando l'`INET Framework` di `OMNet++` per simulare lo stack TCP/IP.

Appendice A

Parametri

A.1 Valori dei Parametri delle Simulazioni

In Tabella A.1 sono mostrati, per ogni parametro delle simulazioni, il valore (o il range di valori) tipico utilizzato nelle simulazioni e l'unità di misura.

Parametro	**Valore Tipico**	**unità di misura**
Nome dello schema	PSF, IPS, PIF, DRP, Flood	-
Capacità code in entrata	50 - 1000	pacchetti
Capacità code in uscita	50 - 1000	pacchetti
Nome della rete	dfn, att, inet300, inet600, inet1000	-
Ampiezza zipf e uniforme	165 - 18000	-
Numero di sottoscrizioni per processor	50 - 100	sottoscrizioni
Tasso di generazione dei messaggi	1 - $1 \cdot 10^{-6}$	sec/msg
Numero di vincoli per sottoscrizione	100 - 1000	vincoli
Dimensione di un messaggio	1000	byte
Generatori di numeri casuali	1 - 4	id del generatore
Durata della simulazione	1 - 1000	secondi simulati

Tabella A.1: Valori dei parametri delle simulazioni.

A.2 Parametri di BRITE

La Tabella A.2 mostra i parametri con cui BRITE è stato configurato per generare le reti DFN e ATT (generate con un tipo di topologia *Bottom-up*), mentre la Tabella A.3 mostra quelli per le reti INET300, INET600 e INET1000 (generate con un tipo di topologia *AS Only*).

Parametri		**DFN**	**ATT**
Bottom-up	**Grouping Model**	Random Pick	Random Pick
	AS Assignement	Constant	Constant
	Number of AS	17	31
	Inter BW Dist	Constant	Constant
	BW	155	155
Router	**HS**	1000	1000
	LS	100	100
	N	30	154
	Model	GPL	BA
	Node Placement	Random	Random
	Growth Type	Incremental	Incremental
	Pref. Conn.	None	None
	p(add)	0.45	0.45
	beta	0.64	0.64
	m	3	2
	Bandwidth Dist	Constant	Constant
	BW	45	45

Tabella A.2: Valori dei parametri di BRITE per le reti DFN e ATT.

Parametri	**INET300**	**INET600**	**INET1000**
HS	1000	1000	1000
LS	10	10	10
N	300	600	1000
Model	Waxman	BA	Waxman
Node Placement	Random	Random	Random
Growth Type	Incremental	Incremental	Incremental
Pref. Conn.	ON	ON	ON
alpha	0.0015	-	0.0015
beta	0.6	-	0.6
m	2	2	4
Bandwidth Dist	Constant	Constant	Constant
BW	155	155	155

Tabella A.3: Valori dei parametri di BRITE per le reti INET.

Appendice B

Scenari Applicativi - Rete INET300

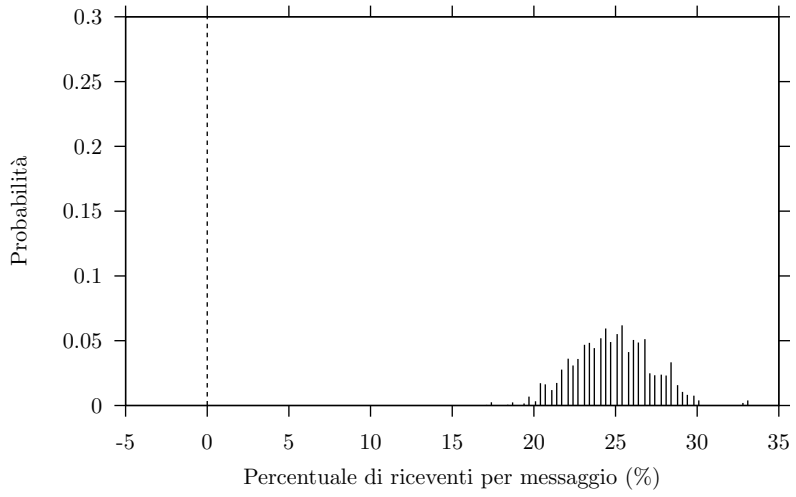


Figura B.1: Scenario applicativo con moda del 25%.

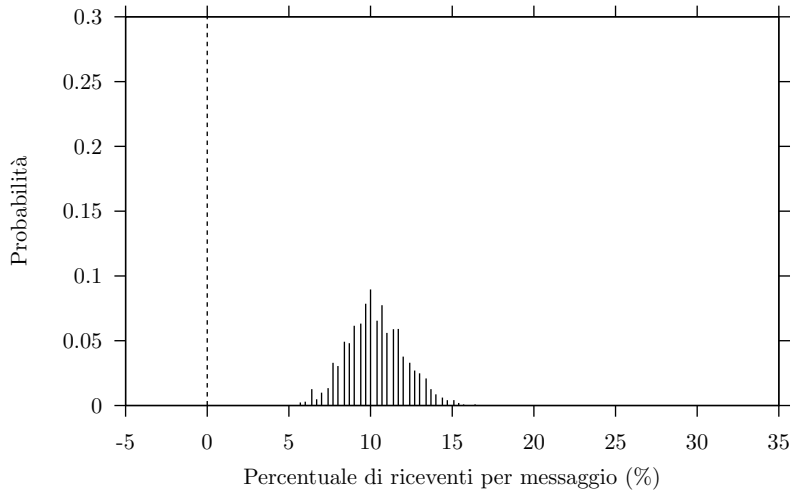


Figura B.2: Scenario applicativo con moda del 10%.

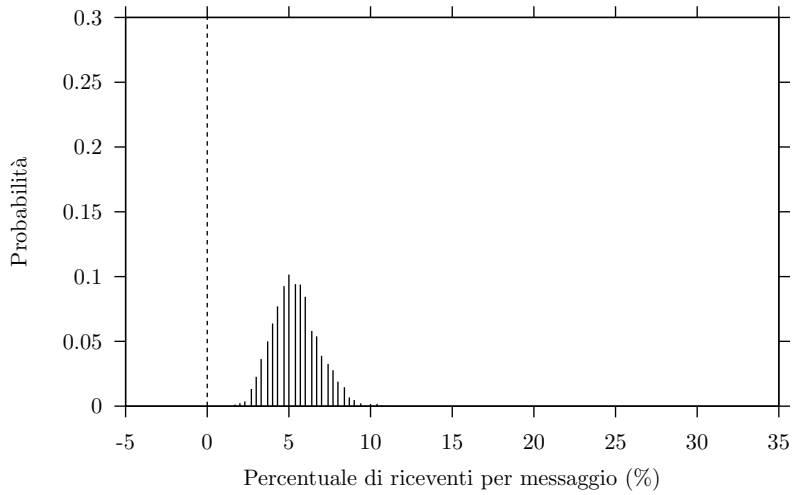


Figura B.3: Scenario applicativo con moda del 5%.

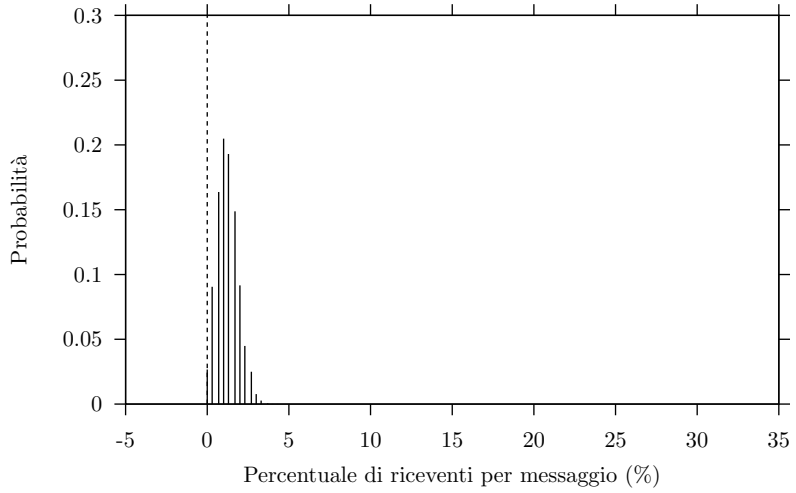


Figura B.4: Scenario applicativo con moda dell'1%.

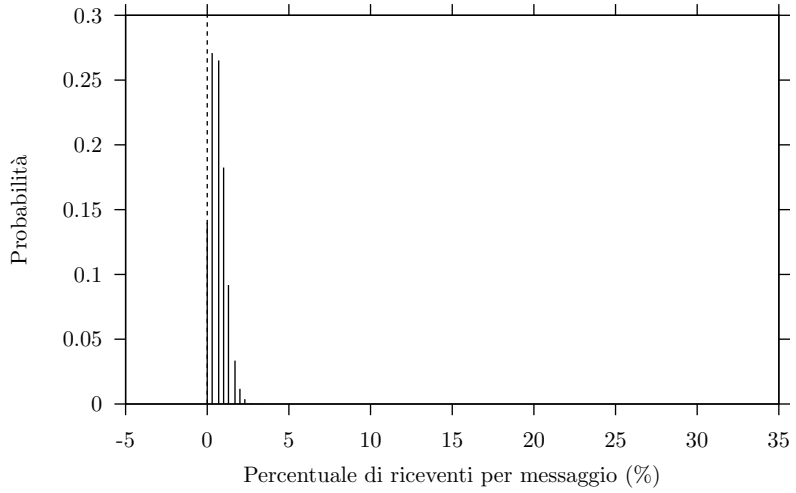


Figura B.5: Scenario applicativo con moda dello 0,5%.

Bibliografia

[1] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.

[2] A. Carzaniga, M. Rutherford, and A. Wolf. A routing scheme for content-based networking. Technical Report CU-CS-953-03, Department of Computer Science, University of Colorado, June 2003.

[3] Antonio Carzaniga, Aubrey J. Rembert, and Alexander L. Wolf. Understanding Content-Based Routing Schemes. Technical Report 2006/05, University of Lugano, September 2006.

[4] Antonio Carzaniga and Alexander L. Wolf. Content-based networking: A new communication infrastructure. In *IMWS '01: Revised Papers from the NSF Workshop on Developing an Infrastructure for Mobile and Wireless Systems*, pages 59–68. Springer-Verlag, 2002.

[5] Antonio Carzaniga and Alexander L. Wolf. Forwarding in a content-based network. In *Proceedings of ACM SIGCOMM 2003*, pages 163–174, August 2003.

[6] R. Chand and P. Felber. A scalable protocol for content-based routing in overlay networks. Technical Report RR-03-074, EURECOM, February 2003.

[7] G. Ewing, K. Pawlikowski, and D. McNickle. Akaroa 2.8 user's manual, April 2003.

[8] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. Meghdoot: content-based publish/subscribe over p2p networks. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 254–273. Springer-Verlag New York, Inc., 2004.

[9] Cyrus P. Hall, Antonio Carzaniga, Jeff Rose, and Alexander L. Wolf. A content-based networking protocol for sensor networks. Technical Report CU-

CS-979-04, Department of Computer Science, University of Colorado, August 2004.

[10] O. Heckmann, M. Piringer, J. Schmitt, and R. Steinmetz. How to use topology generators to create realistic topologies. Technical Report 07/2002, Department of EEITCS, Darmstadt, Germany, December 2002.

[11] C. Jin, Q. Chen, and S. Jamin. Inet: Internet topology generator. Technical Report CSE-TR443 -00, Department of EECS, University of Michigan, 2000.

[12] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[13] K. Pawlikowski, H. D. J. Jeong, and J. S. R. Lee. On credibility of simulation studies of telecommunication networks. *Communications Magazine, IEEE*, 40(1):132–139, 2002.

[14] Krzysztof Pawlikowski. Steady-state simulation of queueing processes: survey of problems and solutions. *ACM Comput. Surv.*, 22(2):123–170, 1990.

[15] Costin Raiciu, David S. Rosenblum, and Mark Handley. Revisiting content-based publish/subscribe. In *ICDCSW '06: Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems*, page 19. IEEE Computer Society, 2006.

[16] Stewart Robinson and Agis Ioannu. The problem of the initial transient: Techniques for estimating the warm-up period for discrete-event simulation models. Warwick Business School, University of Warwick, Coventry, UK, 2007.

[17] L. W. Schruben, H. Singh, and L. Tierney. Optimal tests for initialization bias in simulation output. *Operations Research*, 31:1167–1178, 1983.

[18] András Varga. Omnet++ 3.2 user manual.
Available at: http://www.omnetpp.org.

[19] Bernard M. Waxman. Routing of multipoint connections. *Broadband switching: architectures, protocols, design, and analysis*, pages 347–352, 1991.