Università di Roma 'Tor Vergata'
FACOLTÀ DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica

Metodi e meccanismi per l'individuazione e l'analisi delle dipendenze nei sistemi distribuiti.

21 Febbraio 2011

Relatore:
Chiar.mo Prof. Salvatore Tucci

Correlatore:
Emiliano Casalicchio PhD

Candidato:
Antonello Paoletti
Matricola: 0094437

Anno Accademico 2009/2010

Questa pagina è stata lasciata intenzionalmente bianca

Ringraziamenti

'Whatever affects one directly, affects all indirectly. I can never be what I ought to be until you are what you ought to be. This is the interrelated structure of reality'
Martin Luther King Jr.

Alla mia famiglia, che ha dato il via a tutto questo, a Valentina che mi ha accompagnato fin qui e ai miei colleghi, il cui supporto è stato a dir poco fondamentale

Lunedì, 21 Febbraio 2011

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'oggetto di questo lavoro di tesi è stato la progettazione e l'implementazione di **DeDALO** (*DEpendency Discovery and AnaLisys using Online traffic measurement*), un framework di analisi per la rilevazione di **interdipendenze** in sistemi distribuiti. Il tool è progettato con lo scopo di automatizzare la rilevazione di dipendenze, utilizzando metodi *black-box*[1] basati sull'analisi del traffico di rete.

Studiare le dipendenze in un sistema distribuito, è fondamentale per costruire mappe logiche fra gli elementi che lo compongono, evidenziando quei componenti strategici su cui si fonda il suo corretto funzionamento. Tale analisi è definita **Dependency and Discovery Analysis** (DDA) e costituisce uno strumento utile per valutare l'impatto e la propagazione di eventi localizzati, descrivendo tramite un grafo gli elementi maggiormente coinvolti. DeDALO implementa un algoritmo di DDA orientato all'analisi delle serie temporali, caratterizzanti flussi di dati nel traffico IP.

Negli ultimi anni, le tecniche di DDA hanno ricevuto una crescente attenzione. Vi sono sia prodotti commerciali, come IBM Tivoli [IBM], EMC Smarts [EMC] o HP OpenView [HP], che risultati scientifici, [CZMB08] [KCK08] [BCG+07], in cui si suggeriscono modelli e metodologie per implementare DDA automatizzate. Il concetto di dipendenza è definito in [SJT02] come

> A linkage or connection between two infrastructures, through which the state of one infrastructure influences or is correlated to the state of the other

[1] Nella teoria dei sistemi, un modello *black-box* è un sistema che, similmente ad una scatola nera, è descrivibile solo per come reagisce

ovvero, un legame fra componenti, in cui lo stato di uno influenza lo stato dell'altro.

In [SJT02] sono identificati quattro tipi di dipendenze: *fisiche*, *cyber*, *geografiche* e *logiche*. Nel caso dei sistemi distribuiti, ne risalta una in particolare: la **cyber dipendenza**. Questo tipo di dipendenza caratterizza lo stato di un sistema, in funzione delle **informazioni scambiate al suo interno** (traffico di rete).

Ciò che è interessante, ai fini di questo lavoro, è lo **studio qualitativo e quantitativo** dell'**interconnessione** fra i componenti di un sistema distribuito, cercando di evidenziare quelle situazioni di dipendenza che potrebbero mettere il sistema a rischio.

1.1 Motivazioni

Automatizzare l'analisi delle dipendenze significa attuare delle tecniche di rilevazione tali da **osservare un sistema e descriverne le dipendenze**, operando in totale **autonomia** verso chi lo amministra o lo utilizza. La sfida, in questi ultimi anni, consiste nella definizione di metodologie *black-box*, basate cioè su informazioni rilevate a *run-time* ed indipendentemente dalla conoscenza o meno della struttura del sistema. Ovviamente, tali metodi trovano un campo di impiego più ampio e aiutano a scoprire quelle dipendenze non note, anche qualora si conoscesse la struttura del sistema.

La scelta di implementare un tool automatizzato di rilevazione è motivata dal continuo interesse che questa disciplina ha visto crescere negli anni, avvicinandosi a tematiche di **sicurezza** e **gestione del rischio**: la protezione di un sistema è un'attività che deve considerare sia le potenziali minacce **esterne**, con strumenti tradizionali che ne proteggono il **perimetro** (*firewall*, *Intrusion Detection System*, *De-Militarized Zone*), sia quelle **interne** (incompatibilità, failure, cambiamenti) con strumenti di analisi che ne studiano lo stato e ne traggono modelli di funzionamento.

I due tipi di minaccia sono in grado di causare difficoltà, malfunzionamenti o **failure di sistema**: un attacco hacker può avere le stesse conseguenze di una failure che si origina localmente e si **propaga con elevata velocità**, poichè sviluppatasi su un nodo dal quale **dipendono** migliaia di altri nodi. Capire lo stato delle dipendenze aiuta a scoprire le vulnerabilità di un sistema, costituendo una solida base per anticipare, prevenire e/o intervenire su failure di livello globale.

Uno dei problemi, in caso di failure, spesso consiste nel capire l'**origine di un guasto**, partendo da osservazioni periferiche o report generali. Con un grafo delle

dipendenze questo è possibile e semplice: analizzando quegli elementi che sono stati **maggiormente danneggiati**, è possibile risalire dalla conseguenza alla causa, applicando semplici regole della teoria dei grafi.

Le **infrastrutture critiche**, come i sistemi di controllo o di distribuzione, sono ormai altamente informatizzate e si compongono di migliaia di nodi che collaborano per garantire la funzionalità dell'intero sistema. Si parla di infrastrutture di livello **nazionale** o **regionale** che coprono i più svariati ambiti (*utility*, *trasporti*, *industria*), a cui è richiesto un altissimo livello di **affidabilità** e **continuità** nel funzionamento.

Se il sistema (informatico) di controllo fallisce, l'effetto può essere **esteso** e **catastrofico**: mancate forniture, incidenti, perdita di produzione. La DDA aiuta a confinare queste problematiche, permettendo di individuare e gestire quel sottoinsieme di elementi da cui sia il sistema di controllo, che l'intera infrastruttura non possono prescindere.

Grandi aziende, come IBM, HP o Microsoft, hanno speso considerevoli risorse nella ricerca in questo campo, rilasciando prodotti di gestione (IBM Tivoli [IBM], su tutti) per semplificare la *visione* di un amministratore di rete in relazione alle interdipendenze nel sistema.

In letteratura si trovano moltissime soluzioni, teoriche o metodologiche, per condurre DDA di vario tipo, partendo da considerazioni e rappresentazioni diverse di un sistema distribuito. Il funzionamento di DeDALO si basa su tecniche discusse in diversi articoli, analizzando vantaggi e difficoltà e arrivando a definire una **metodologia** basata sull'analisi degli **intertempi fra flussi di dati**.

Scopo di DeDALO è implementare una DDA automatizzata con un **minimo contenuto informativo** evitando, cioè, di conoscere elementi strutturali come la topologia, i sistemi operativi o i *middleware* in esecuzione. DeDALO acquisisce i tracciati di rete scambiati nel sistema, in accordo con la definizione di [SJT02], e li studia, cercando **correlazioni fra flussi di dati**, per valutare il grado di dipendenza fra coppie di elementi.

Un'analisi basata sul traffico, permette di non considerare la complessità o l'eterogeneità dei vari sotto-sistemi (*OS* diversi, macchine virtuali, contesti applicativi), in quanto si considera un elemento, come il **traffico IP**, che è definito nella sua sintassi in maniera standard e univoca.

I contributi che questo lavoro apporta sono:

1. uno **studio approfondito** della **letteratura** sui modelli di DDA

2. l'**identificazione** degli **approcci** più appropriati
3. la **progettazione** di un **framework** basato sull'approccio descritto in [CZMB08]
4. la **definizione** di una **metrica di dipendenza** fra gli elementi individuati
5. la **progettazione** di un **ambiente di simulazione** per validare il framework
6. l'**implementazione**, valutazione e validazione del **framework** di DDA

1.2 Organizzazione della tesi

La tesi sarà organizzata in **7 capitoli**:

- Nel **capitolo 2** è descritta in dettaglio la **problematica** del **Dependency discovery**, motivando la necessità di **automatizzare il processo** e descrivendo alcune soluzioni disponibili. Viene, quindi, introdotta la soluzione alla base di DeDALO, motivandone la scelta e descrivendo in dettaglio i punti di forza, rispetto a prodotti commerciali o descritti in letteratura.
- Nel **capitolo 3** viene presentato l'**approccio scelto per definire DeDALO**. Saranno descritte le modalità con cui osservare il sistema, valutare gli eventi d'interesse ed analizzarli per estrarre le dipendenze.
- Nel **capitolo 4** si descrive la **struttura applicativa di DeDALO**. Saranno presentati gli elementi di cui si compone, gli algoritmi e le singole funzionalità, descrivendo come operano ed interagiscono per attuare la DDA.
- Nel **capitolo 5** viene presentato l'**ambiente di simulazione** sviluppato per testare DeDALO. Saranno illustrate le modalità con cui condurre simulazioni per analisi DDA. Sarà poi descritto l'ambiente di simulazione, introducendo *NS3* [Lac10], i modelli protocollari sviluppati e un'interfaccia grafica per la generazione di topologie.
- Nel **capitolo 6** sono presentati i **risultati dell'esecuzione di DeDALO** su scenari applicativi simulati. Si utilizzerà una stessa topologia, di medie dimensioni, per eseguire alcune rilevazioni su applicazioni distribuite di diversa complessità. Tali scenari saranno descritti in forma di workflow, definendo i servizi utilizzati e le modalità con cui accedervi.
- Il **capitolo 7** chiude questo lavoro con conclusioni e riflessioni su sviluppi futuri.

Capitolo 2

Le dipendenze nei sistemi distribuiti e reti

In questo capitolo viene descritta in dettaglio la problematica del dependency discovery in sistemi distribuiti, motivando la necessità di automatizzare il processo e descrivendo alcune soluzioni disponibili. Viene, quindi, introdotta la soluzione alla base di DeDALO, motivandone la scelta e descrivendo in dettaglio i punti di forza rispetto a prodotti commerciali o descritti in letteratura.

2.1 Problematica

Le grandi infrastrutture di rete sono composte da migliaia di elementi, che operano ed interagiscono all'interno di ambiti applicativi **distribuiti**. Le componenti di un'applicazione sono dislocate su diversi nodi, i quali offrono e fruiscono di servizi comunicando attraverso la rete. Lo scambio di dati che ne è alla base, lega fra loro gli elementi del sistema, determinando schemi di **dipendenza** fra nodi o **servizi software**.

La ricostruzione di tali schemi, *Dependency Discovery*, è fondamentale per comprendere meglio il funzionamento del sistema e individuare gli elementi critici/-strategici da cui non può prescindere. Una cattiva conoscenza del sistema, in tal senso, rende il sistema stesso vulnerabile a congestioni e malfunzionamenti, dovuti a cattive decisioni in materia di adeguamenti, sia infrastrutturali (server, router o collegamenti di rete), che software (componenti di applicazioni distribuite).

Al contrario, una chiara visione delle interdipendenze, fa emergere quegli elementi che, ad esempio, necessitano di più risorse, strutture di replicazione o migliore

connettività. Un esempio, nel caso di un sistema distribuito, è dato dalla relazione fra **applicazioni WEB** e il **DNS**: una failure sul **DNS** pregiudica l'intera attività dell'applicazione WEB, limitando le prestazioni dei servizi offerti. Analogamente, il **DNS** è legato al **routing**, poichè se un router perde, o instrada in maniera non corretta, i pacchetti rende inaccessibile il **DNS**.

2.2 Dipendenze

Lo studio delle dipendenze nei sistemi distribuiti, è stato inquadrato in diversi lavori come un'analisi volta a caratterizzare l'interconnessione, e lo stato degli elementi, in funzione dei dati che essi scambiano. In articoli di rilievo, come [SJT02], la dipendenze in sistemi distribuiti, sono dette **cyber dipendenze**, affermando che

> An infrastructure has a cyber interdependency if its state depends on information transmitted through the information infrastructure

Volendo dare un valore meno astratto alla definizione data in [SJT02], si può definire il concetto di dipendenza fra un elemento A e un elemento B, come la necessità, da parte di B, di fruire dei servizi di A per fornire a sua volta un servizio o completare l'esecuzione di un task. La relazione è detta di **interdipendenza** se è bidirezionale, ovverosia se i due elementi sono **mutuamente dipendenti**.

Quindi, la dipendenza fra due servizi è una *formalizzazione codificata della necessità di fruire di un servizio S1 per poter accedere ad un servizio S2*

Il concetto di **cyber-dipendenza** (d'ora in poi espressa come **dipendenza**) è strettamente correlato a un'infrastruttura ICT, come un sistema distribuito. Comprendere la struttura di tali sistemi è fondamentale per caratterizzare sia le dipendenze che le metodologie per rilevarle. L'infrastruttura di un sistema distribuito può essere descritta attraverso tre livelli architetturali: livello **fisico**, livello di **rete** e livello **applicativo**.

Ogni livello è caratterizzato da funzionalità specifiche, hardware e/o software, permettendo ai livelli superiori, o all'utente di fruire di **servizi** di vario genere. I sistemi di interesse, in questa tesi, sono quelle infrastrutture basate su protocollo **IP** (Internet), i cui livelli caratteristici riflettono la struttura dello stack TCP/IP:

- Livello applicativo
- Livello IP (rete, routing, trasporto)

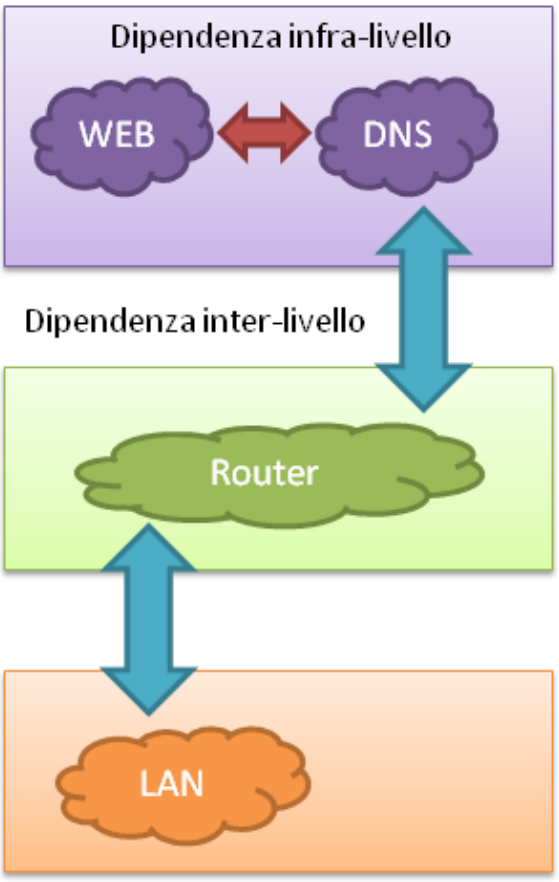

Figura 2.1: Dipendenze fra i livelli di un sistema distribuito

- Livello fisico (datalink, connetività)

all'interno del livello applicativo si ritrovano dei servizi **di supporto**, come il *naming* (**DNS**), l'*identity mangement* (**Kerberos**) e la gestione di rete (**ICMP**), che fungono da **collegamento** fra il livello applicativo e i livelli sottostanti. Ogni livello si compone di elementi hardware e/o software che interagiscono scambiando informazioni e servizi. E' possibile inquadrare le entità caratteristiche di un sistema distribuito all'interno di ognuno dei tre livelli:

- *Applicazioni software*: WEB, DNS, DBMS.. (Livello applicativo)
- *Router* (Livello IP)
- *Canali di comunicazione* (Livello fisico)

L'interazione fra elementi di uno stesso livello determina delle relazioni **infra-livello**, ad es., strutture SOA, o protocolli di routing. L'interazione fra elementi di livelli diversi è detta **inter-livello**, ad es. risoluzione di indirizzi IP (DNS) per accedere a un

server WEB. In figura 2.1 sono rappresentate alcune di queste relazioni, mostrando il caso **WEB-DNS** (relazione infra-livello) e i casi **DNS-Router** e **Router-LAN** (relazione inter-livello). Tali relazioni caratterizzano in maniera analoga le dipendenze osservabili in **dipendenze fra elementi di uno stesso livello** e **dipendenze fra elementi di livelli diversi**. Dipendenze del primo tipo sono tipiche di un ambito di sviluppo/progettazione in cui si pianifica la logica e la struttura di una piattaforma. Sorgono dipendenze infra-livello fra i singoli servizi di una struttura SOA, nel definire il routing o nel progettare reti LAN.

Dipendenze inter-livello, invece, si ritrovano nel modo in cui un elemento utilizza le risorse a sua disposizione. Tali modalità sono spesso definite dalla logica applicativa del singolo elemento e sono tanto più complesse quanto maggiore è la sua capacità di ottimizzarne l'uso.

[KK01] classifica le dipendenze utilizzando uno spazio vettoriale a sei dimensioni (mostrato in figura 2.2):

- **Spazio / Dominio**: *vicinanza* fra gli elementi in relazione di dipendenza, che può essere inter o intra dominio, sistema o package, a seconda del tipo di ambito in cui si configura la dipendenza.
- **Tipo di componente**: caratterizzazione degli elementi considerati, che possono essere hardware, entità logiche (altri sistemi) o software
- **Attività del componente**: capacità dell'elemento di agire in autonomia sul suo stato (attivo) o necessità di un *intermediario* per comuni operazioni di funzionamento (passivo)
- **Forza della dipendenza**: valutazione qualitativa di dipendenze nulle, intermittenti o stabili
- **Formalizzazione della dipendenza**: grado di formalizzazione e di automatizzazione nel rilevarla. Può essere bassa e richiedere alti costi di rilevazione o può essere alta e permettere l'attuazione di metodi codificati di rilevazione.
- **Criticità della dipendenza**: modalità con cui si deve svolgersi la cooperazione definita dalla dipendenza. Un elemento può essere considerato un 'co-requisito', un 'pre-requisito' o un 'ex-requisito' nei confronti di un altro elemento, indicando dei vincoli nell'ordine con cui tale dipendenza dev'essere soddisfatta

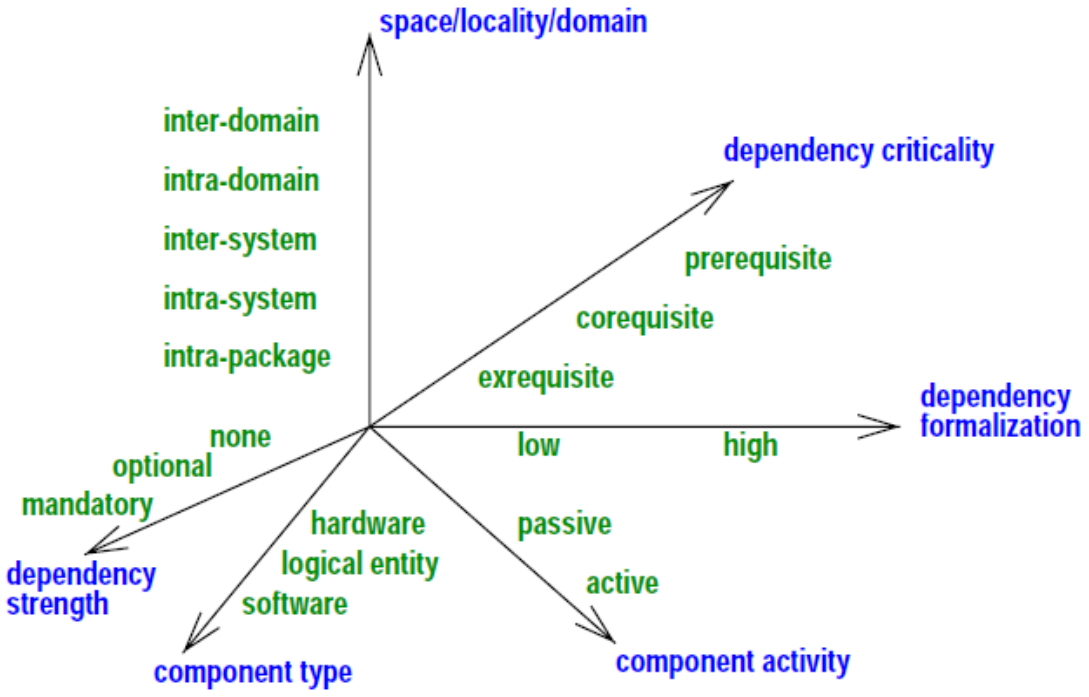

Figura 2.2: Spazio multidimensionale delle dipendenze

In aggiunta a questi aspetti, ne vanno poi considerati altri due, estranei al modello vettoriale:

- **Tempo**: Le dipendenze possono variare in funzione del ciclo di vita di un componente, attivandosi o disattivandosi in presenza di particolari eventi.
- **Ciclo di vita della dipendenza**: passaggio da dipendenza *funzionale*, catturata in fase di design a dipendenza **strutturale**, catturata in fase di esecuzione.

2.3 Rilevazione e analisi

La DDA (Dependency Discovery Analysis) è una tecnica di analisi il cui scopo è determinare un modello di dipendenze fra gli elementi di un sistema distribuito. E' utile per implementare modelli di propagazione e impatto, evidenziando gli elementi direttamente/indirettamente coinvolti da modifiche o failure localizzate.

La DDA può essere intesa sia in un'ottica proattiva che reattiva: un grafo delle dipendenze può aiutare a predire l'impatto globale di una failure (approccio proattivo) e può essere un supporto nella ricerca di cause in caso di stati anomali del sistema (approccio reattivo). La DDA si preoccupa di scoprire le cause della propagazione di

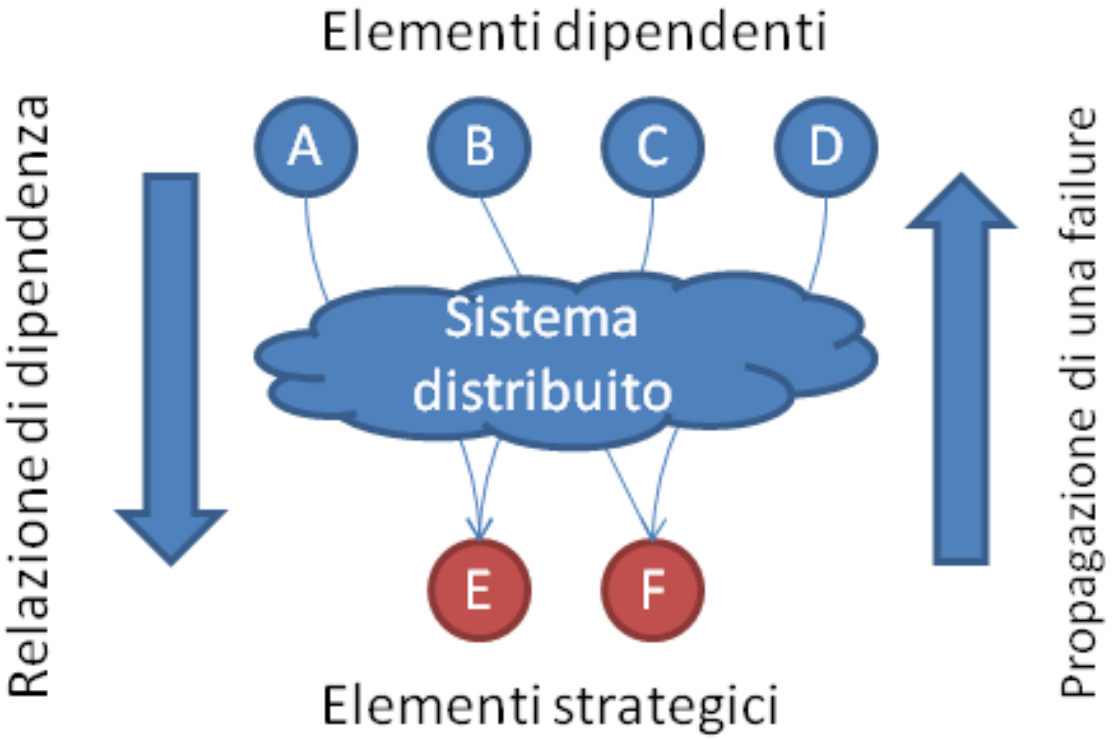

Figura 2.3: Relazione fra dipendenza e propagazione di una failure

perturbazioni critiche per il funzionamento del sistema, qualunque ne sia l'origine. Non esiste un'unica definizione di sistema distribuito, ma tutte concordano sul fatto che è necessaria una interconnessione o un coordinamento fra i suoi elementi. Fra tutte, ne risalta una, di **Leslie Lamport**, che motiva chiaramente la necessità di comprendere i modelli di interconnessione:

> A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable

Se uno o più elementi sono in grado di compromettere un intero sistema, allora è importante individuarli e individuare quegli elementi che 'dipendono' direttamente o indirettamente da loro. La DDA fa proprio questo, permettendo di ricostruire le relazioni di dipendenza in forma chiara e possibilmente automatizzata.

I passi di cui si compone la DDA sono fondamentalmente tre:

- caratterizzazione
- rilevazione
- analisi

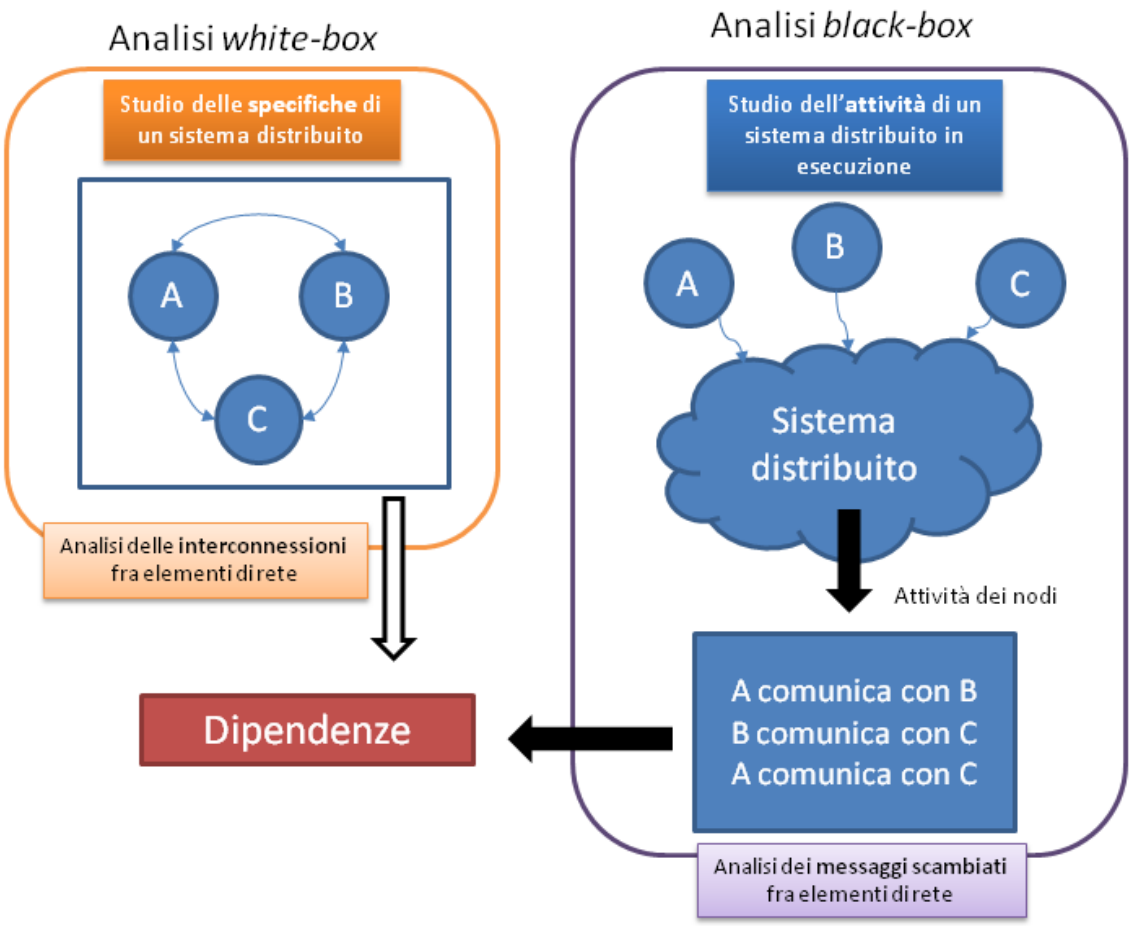

Figura 2.4: Tecniche di analisi black-box e white-box

Nella prima fase si identificano le metriche e i metodi per la rilevazione, in base al sistema e al tipo di dipendenze da rilevare. Nella seconda fase si attua la rilevazione e nella terza se ne analizzano i risultati.

In base al tipo di dipendenze ricercate, si possono estrarre linee guida per definire metodologie di rilevazione. Le dipendenze intra-livello, ad esempio, si possono ritrovare nei documenti di specifica o di progetto, valutando, ad es., la disposizione dei router o l'architettura di un'applicazione distribuita. Studi di questo tipo sono detti *white-box*, poichè si basano sulla conoscenza delle specifiche del sistema. Le tecniche duali sono dette *black-box* e si basano sullo studio delle interazioni fra gli elementi del sistema. Le tecniche *white-box* traggono ampio spunto da metodi tipici dell'**ingegneria del software**, come la *Application Code Analysis*, in cui si esegue un'ispezione del codice prima che l'applicazione venga eseguita, o la *Visual model analysis* condotta su modelli visuali del sistema. Anche le tecniche *black-box* hanno un loro corrispettivo in ingegneria del software, con tecniche come la *Message exchange sequence* volte alla rilevazione di schemi di comunicazione fra gli elementi

del sistema.

Le dipendenze inter-livello sono raramente valutabili al di fuori di un contesto di esecuzione, a causa della diversa complessità con cui i livelli interoperano. Le applicazioni *location aware* ad esempio, rompono la consueta separazione fra livelli, 'adattandosi' all'ambiente circostante pur di continuare nell'esecuzione. L'adattamento non è predicibile e l'effetto che ha, sia sul livello applicativo che sui livelli sottostanti, non si può determinare a priori. Un router può scegliere sul momento la rete più libera, utilizzando il livello fisico in base a fattori che variano continuamente, come il livello di congestione o la disponibilità di un canale. Gli studi più efficaci, in questo caso, sono orientati verso metodologie *black-box* il cui scopo è determinare comportamenti 'tipici' assunti da un elemento nel corso della sua esecuzione. Lo sviluppo di tecniche *black-box*, basate su analisi del sistema a *run-time*, costituisce una delle sfide più recenti nel campo del discovery ed è fonte sia di opportunità che di nuovi problemi.

1. E' un'opportunità perchè spesso non si hanno specifiche sul sistema e l'unico modo per sondarlo è tramite osservazioni della sua attività
2. Un'analisi basata su osservazioni può essere facilmente automatizzata e richiede un contributo minimo da parte dell'utente
3. A volte è impossibile predeterminare lo stato di un sistema analizzandone le specifiche

I problemi che introduce sono di due tipi:

1. Implementazione di modelli efficaci per tradurre **eventi** di sistema in considerazioni sulla sua struttura
2. Gestione e trattamento dei dati acquisiti, con rischi di violazione della confidenzialità.

2.4 Metodi Automatici per l'estrazione delle dipendenze

Il crescente interesse verso le tecniche di DDA, ha portato alla definizione di metodi e prodotti per automatizzare la fase di rilevazione, ognuno con la propria visione e proprie soluzioni metodologiche. L'approccio alla base di DeDALO è di

Figura 2.5: Analisi black-box

tipo *black-box* e si basa sull'analisi del traffico di rete, per rilevare dipendenze fra servizi IP.

In letteratura si possono trovare molti lavori dedicati a DDA di questo tipo, tutti orientati alla definizione di modelli algoritmici che ricevono degli eventi di sistema e restituiscono una mappa delle dipendenze. Questo tipo di approccio è ormai dominante per lo studio di quei sistemi in cui l'analisi delle specifiche è impossibile, infattibile o inutile. Automatizzare una DDA, in tal senso, significa definire:

- i **punti di osservazione** da cui rilevare l'attività (nodi, router, collegamenti...)
- i **dati da rilevare** (eventi del S.O., log di router, traffico IP...)

- gli **obiettivi dell'analisi** (dipendenze fra servizi, dipendenze fra livelli)
- un **modello di estrazione delle dipendenze** dai dati (aggregazione, correlazione...)
- i **parametri** del modello (statici, variabili...)
- **modelli di perturbazione** attiva del sistema (disattivazione di nodi, iniezione di traffico di test...)

come mostra lo schema in figura 2.6. Tutti i lavori considerati riflettono questa logica, differenziandosi sulla scelta dell'input, l'implementazione del modello e la definizione (non sempre) di modelli attivi di perturbazione. Nella prima fase si identificano gli elementi del sistema da eleggere come 'rappresentanti' della sua attività, predisponendoli all''ascolto'. Nella seconda fase si definisce il tipo di informazione da estrarre, in rapporto a diversi fattori, come il livello di pervasività dell'ascolto, gli obiettivi della ricerca e problematiche di confidenzialità.

La terza fase è quella che guida tutta l'analisi: si definisce il suo fine, indicando il tipo di dipendenze ricercate, e dando una prima caratterizzazione dei risultati attesi. La quarta fase è quella in cui si definisce la 'soluzione architetturale' scelta per la DDA. E' l'elemento chiave dell'analisi, poichè descrive quei meccanismi con cui 'si traggono conclusioni' sul sistema analizzandone il suo stato.

Tutti i lavori sull'argomento concentrano in questa fase la chiave di una DDA efficace o fallimentare. Un algoritmo di inferenza dimostra la sua efficacia nel momento in cui adatta con successo le sue politiche all'evoluzione del sistema che analizza. Nel caso di sistemi distribuiti si parla di stream di input che raramente si prestano a schemi noti o stabili nel tempo. La variabilità di un sistema è tale da generare un numero elevatissimo di stati, con un numero ancora più alto di transizioni.

La fase successiva riguarda la definizione dei parametri caratteristici del modello, caratterizzandoli in base al valore assunto, alle modalità con cui variano (se variano) e all'impatto che hanno sul modello stesso. Si distinguono due approcci, uno basato su parametri predeterminati o impostati manualmentente ed uno basato su parametri che si 'adattano' allo stato del sistema, con tecniche di Machine Learning, dopo sessioni di test o considerando dei periodi transitori di 'istruzione' del modello.

L'ultima fase è opzionale. Si definiscono modelli algoritmici con cui 'sollecitare' attivamente il sistema e valutare la sua risposta. E' una tecnica utilizzata per sondare lo stato di specifiche funzionalità e integrare i risultati sulle dipendenze

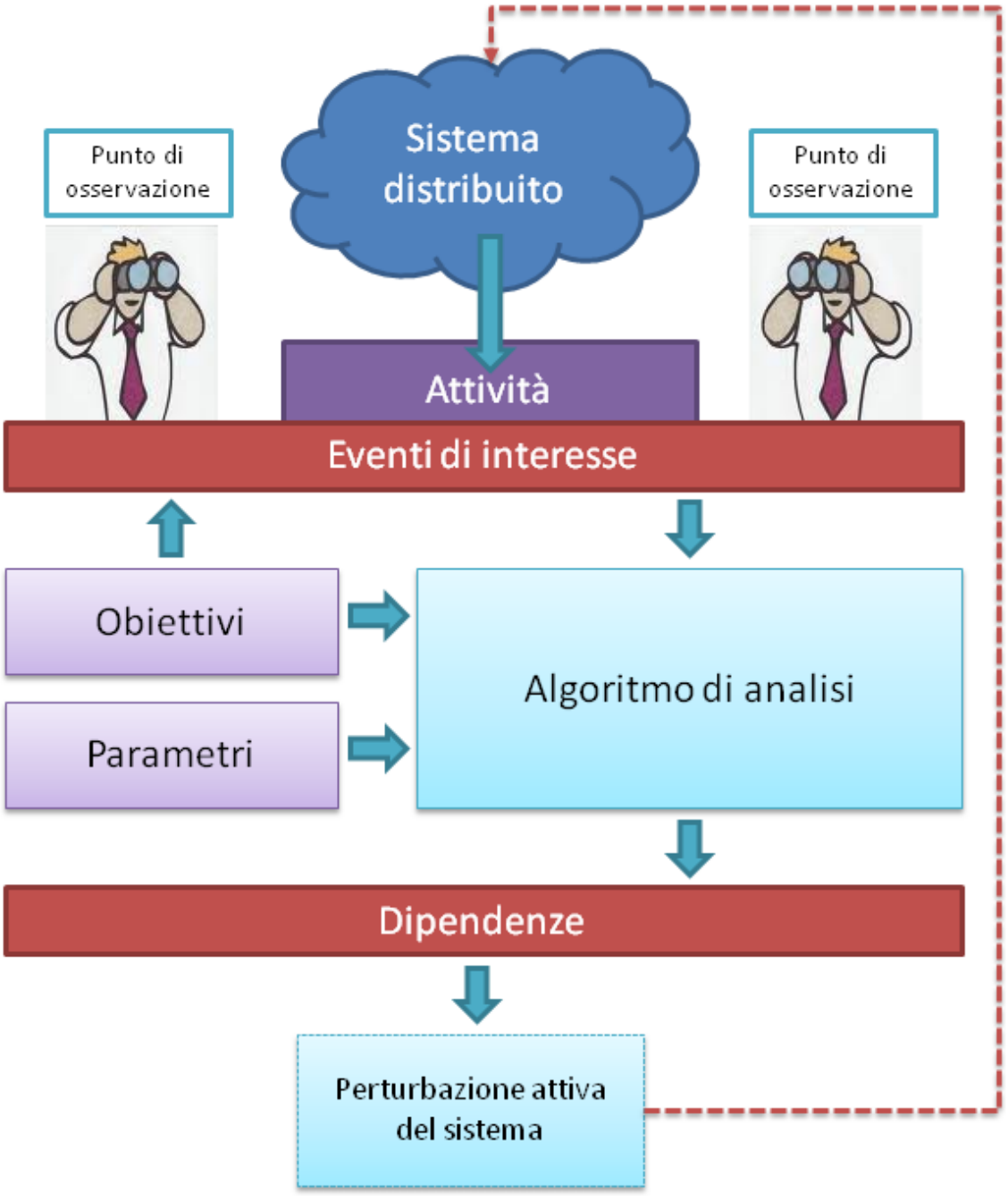

Figura 2.6: Schema di una DDA black-box automatizzata

rilevate. Perchè sia attuabile, questa fase richiede un certo grado di controllo sul sistema, con la possibilità, ad es., di iniettare traffico di test o disattivare nodi critici.

Applicata alla DDA, questa tecnica si rivela utile se i suoi obiettivi sono ben definiti, puntuali e orientati allo studio di singoli elementi. E' il caso, ad es., di rilevazioni mirate a conoscere lo stato o la connettività di nodi strategici.

2.4.1 Utilizzo di automi a stati finiti

Un primo approccio consiste nel modellare il sistema come una macchina a stati finiti, definendo un numero di stati in rapporto agli 'eventi' d'interesse (eventi di SO, log di VM...) e identificando quelle catene di transizioni che conducono a conclusioni codificate. E' richiesta un'altissima conoscenza del sistema, oltre a definizioni codificate degli stati, delle conclusioni e dei percorsi che li legano. Nuovi eventi conducono a nuovi stati e nuove catene, rendendo un modello statico facilmente obsoleto. Un sistema di grandi dimensioni presenta una quantità di stati, variabili nel tempo, tale da richiedere modelli definiti a run time e massive capacità di calcolo. Un even-

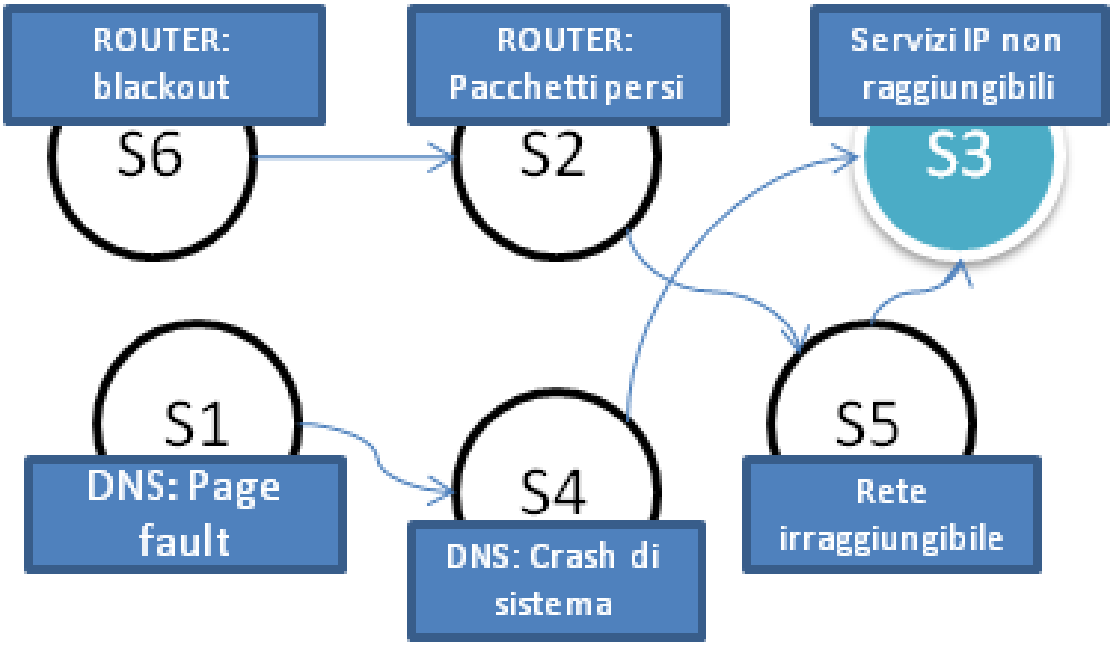

Figura 2.7: Modello di rilevazione basato su un automa a stati finiti

to/stato può essere definito con diversi livelli di astrazione, considerando lo stato di singoli elementi e lo stato del sistema. Nell'immagine di esempio, si correla il crash di un DNS al mancato funzionamento di servizi IP, determinando una relazione di dipendenza.

2.4.2 Tecniche di apprendimento di regole di comunicazione

Un secondo approccio si basa sull'individuazione di regole o pattern di comunicazione nel flusso di eventi analizzati. Si basa su tecniche guidate per 'apprendere' metodologie di analisi e adattarle a comportamenti 'tipici' del sistema. Richiede una

minima conoscenza del sistema analizzato ma, senza un'adeguata preparazione del modello, è fortemente esposto a problematiche di falsi positivi/negativi.

L'assunto alla base di questo modello, come per quelli che seguiranno, è che la maggior parte degli eventi di rete, traffico soprattutto, sono generati da applicazioni (web, email, p2p) e tendono ad essere governati da 'regole' sottostanti. Questo modello è mutuato da modelli di prevenzione contro spam e intrusione, definendo dei classificatori in grado di apprendere, e riconoscere, specifici 'pattern' dal traffico o dagli eventi analizzati. E' ovviamente richiesta una fase preliminare di 'tuning', tramite cui acquisire informazioni dal sistema e 'istruire' il modello. Un esempio è dato da [KCK08], in cui si analizzano flussi di dati per categorizzarli e inferire la logica che li genera e li correla.

2.4.3 Correlazione di eventi entro finestre di tempo

Alcuni lavori come [BCG+07] orientano l'analisi verso lo studio delle modalità con cui occorrono gruppi di eventi, considerandoli 'espressione' dello stato di un elemento. Se due elementi generano eventi con coordinazione, soprattutto temporale, è presumibile che siano legati da relazioni di dipendenza. L'apporto dell'utente è limitato a semplici operazioni di setup, da effettuare in fase di deploying. Questo metodo è meno complesso del precedente ma non meno effice e può essere messo in atto sia definendo staticamente i suoi parametri, sia raffinandoli con tecniche di Machine Learning. In [BCG+07], ad es., l'oggetto di studio è il traffico di rete, analizzato in funzione dei nodi coinvolti e dei tempi con cui occorre ogni comunicazione. Si assume che due flussi di dati (e quindi gli elementi coinvolti) siano in relazione di dipendenza se co-occorrono frequentemente entro piccole finestre di tempo. Questa semplice definizione rimanda a un'idea meno stretta di 'co-requisito', focalizzandosi sulla coordinazione temporale di due eventi di sistema. Le problematiche di questo approccio sono diverse e tendono a variare a seconda del modello scelto per rappresentare la finestra temporale.

C'è poi un problema di ambiguità nel motivare i risultati: l'assunto è che se due flussi co-occorrono molte volte, allora è presumibile che siano legati. La debolezza di questo approccio si mostra nel momento in cui flussi molto popolari o molto lunghi tendano a prevalere su flussi meno frequenti, oscurandone l'importanza a favore della mera numerosità. Può accadere, per assurdo, che flussi stabilmente accoppiati, ma poco frequenti risaltino meno di flussi casuali ma molto frequenti.

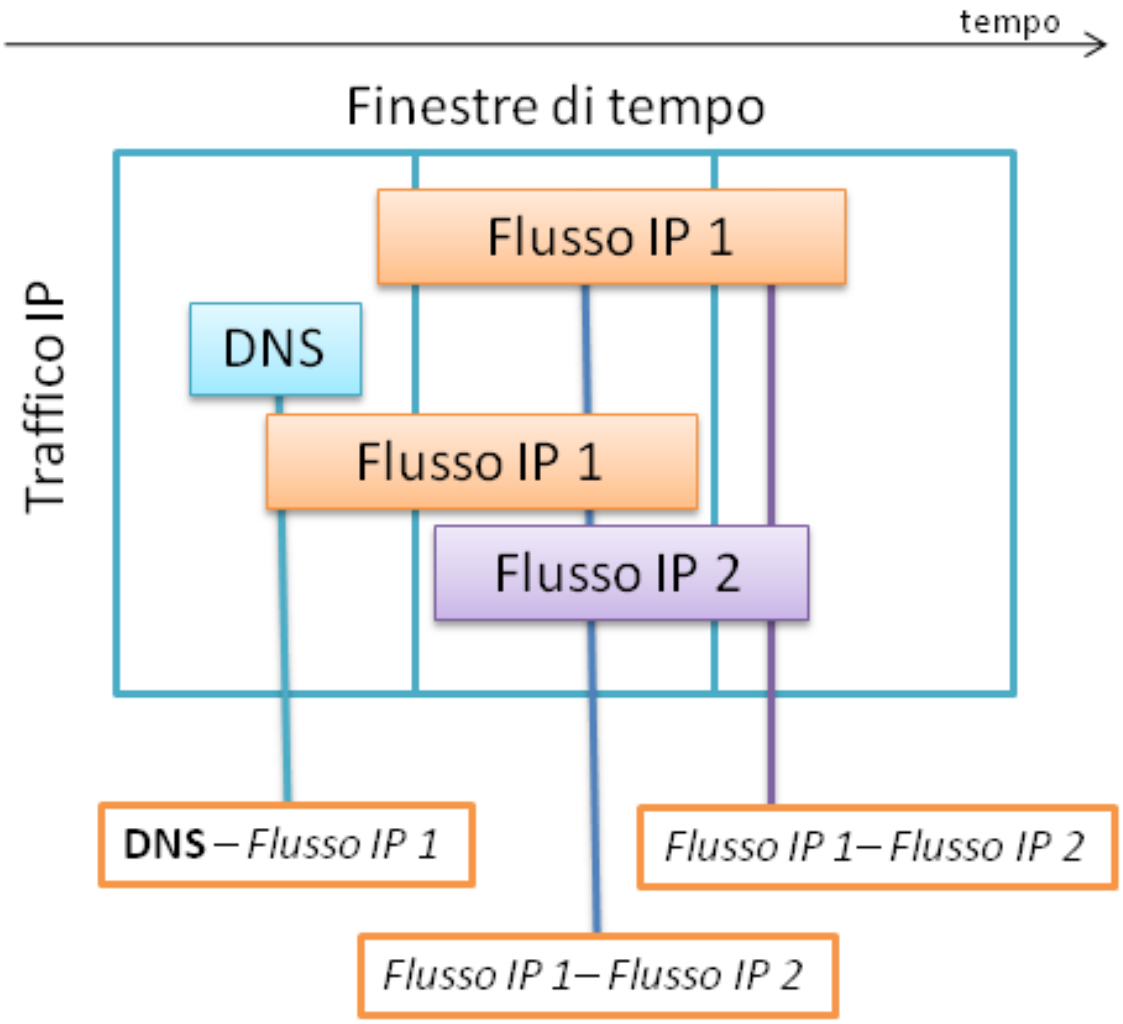

Figura 2.8: Modello di correlazione temporale

2.4.4 Studio della sistematicità nella correlazione di eventi

Un ultimo modello, proposto in [CZMB08], e applicato in DeDALO, si basa sullo studio della 'stabilità' con cui co-occorrono coppie di eventi. La base metodologica è molto simile a quella del modello precedente, ma si basa su assunti tipici del dominio applicativo, ritrovando nel concetto di 'sistematicità' la caratteristica che più rimanda al concetto di dipendenza. Questo approccio supera agilmente le problematiche di parametrizzazione e ambiguità, tipiche del modello di [BCG+07], assumendo che la reale dipendenza fra due servizi è tale se sussistono tre condizioni: *due flussi di dati, generati da una logica applicativa (**Immutabilità**), co-occorrono per un numero elevato di volte (**Ripetitività**) con intertempi simili (**Sistematicità**).*

L'ordine di grandezza con cui considerare la **Ripetitività** e la **Sistematicità** è tipico di un ambiente applicativo, in cui una stessa sequenza di operazioni viene

ripetuta per centinaia o migliaia di volte, con intertempi rilevati che abbiano differenze dell'ordine del millesimo di secondo. Questo approccio divide nettamente le

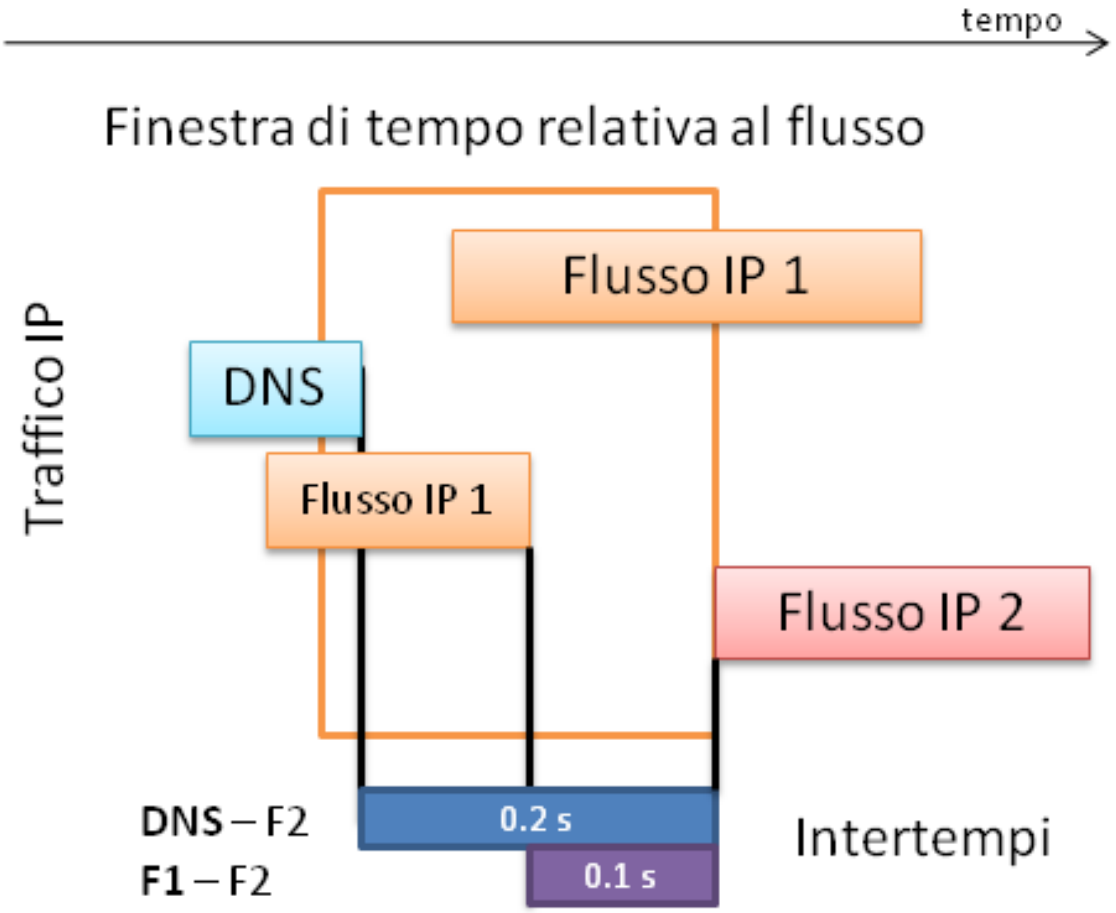

Figura 2.9: Modello di analisi degli intertempi

correlazioni 'sistematiche', di natura applicativa, da quelle 'casuali', solitamente di natura 'umana', con l'assunto che la dipendenza fra due servizi è tanto più forte quanto è maggiore la 'stabilità' con cui vengono utilizzati. La classica dipendenza 'WEB - DNS', ad es., si può ritrovare nella sistematica sequenza di flussi WEB e flussi DNS, intervallati frequentemente da uno stesso intertempo. Più gli intertempi fra due flussi sono 'stabili', più la dipendenza è forte.

Questo modello ha molti vantaggi e permette, caso unico, di stabilire valutazioni quantitative sulle dipendenze. Mutuando concetti dalla **Application Code Analysis** [Pre10] e, più in generale delle analisi white-box, si rileva una sovrapposizione fra il concetto di **dipendenza** e il grado di **accoppiamento** (**coupling** cfr. [Pre10]) fra componenti applicativi. Più questo grado è elevato, più i due elementi saranno **dipendenti**, al punto che il malfunzionamento di uno causerà problemi ad entrambi.

Portare questa evidenza da un'analisi delle specifiche, a un'analisi dell'attività, significa identificare un'invariante che si possa ritrovare sia nei documenti di progetto che negli eventi di un sistema. Tale invariante consiste nel fatto che una logica applicativa poco complessa si compone di 'sequenze' di istruzioni o accessi a servizi, eseguite con tempistiche predicibili o comunque stabili, poichè immutabili nel tempo e automatizzate. Queste due caratteristiche distinguono nettamente un'applicazione distribuita (in cui possono sussistere dipendenze) da un comportamento 'umano' (estraneo al normale funzionamento del sistema)

Capitolo 3

Identificazione ed analisi delle dipendenze

In questo capitolo viene presentato l'approccio scelto per definire DeDALO, in accordo con la struttura di **DDA** *black-box* descritta nel capitolo 2. Saranno descritte le modalità con cui **osservare** il sistema, estrarre **eventi** d'interesse e **analizzarli** per rilevare dipendenze fra servizi IP.

3.1 Concetti di base e definizioni

In questa sezione sono introdotti quei concetti che saranno utili nel seguito per descrivere il funzionamento di DeDALO.

Si definisce un **flusso** come una sequenza ininterrotta di dati scambiati fra due nodi di rete. Un **flusso** ha origine per iniziativa di un nodo **sorgente**, il quale interpella un nodo **destinazione** per avviare una comunicazione TCP o UDP.

I dati scambiati si compongono di interazioni fra i due nodi, allo scopo di negoziare la fornitura di un **servizio** IP fra il nodo **sorgente** e il nodo **destinazione**.

Un **servizio** IP consiste in specifiche funzioni software che un nodo offre tramite la rete. Un esempio di flussi e servizi è dato dall'interazione fra un nodo client **A** e un **DNS** server (mostrata in figura 3.2) per risolvere un nome host:

- **Inizio flusso A-DNS**
- **A** interpella il **DNS** per risolvere un nome host (*richiesta* di un **servizio**)

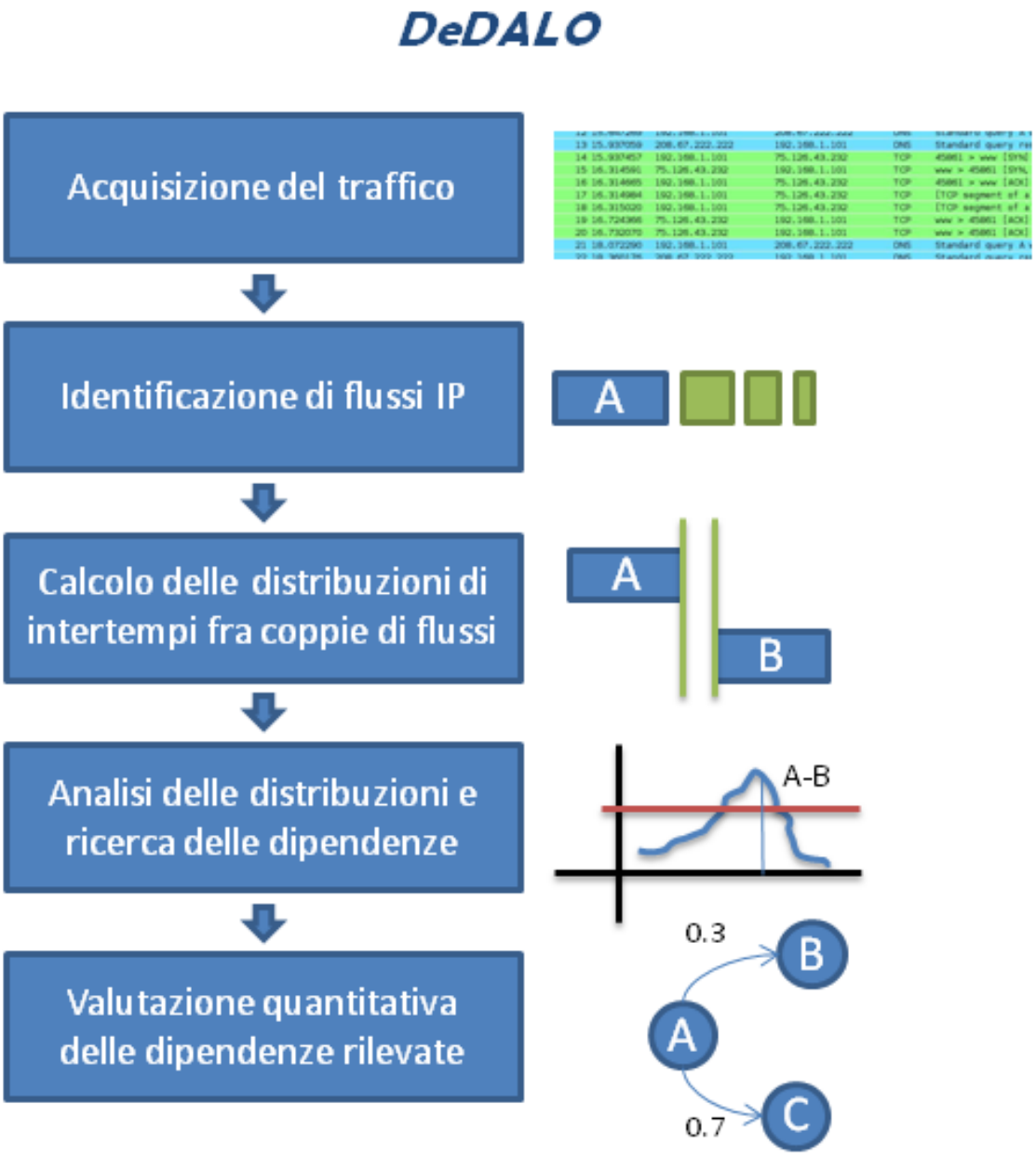

Figura 3.1: Struttura di DeDALO

- Il DNS riceve la richiesta, risolve il nome e invia l'IP ad **A** (*fornitura* di un **servizio**)
- **Fine flusso A-DNS**

Un flusso è contraddistinto da quattro informazioni fondamentali:

- nodo **sorgente**
- nodo **destinazione**

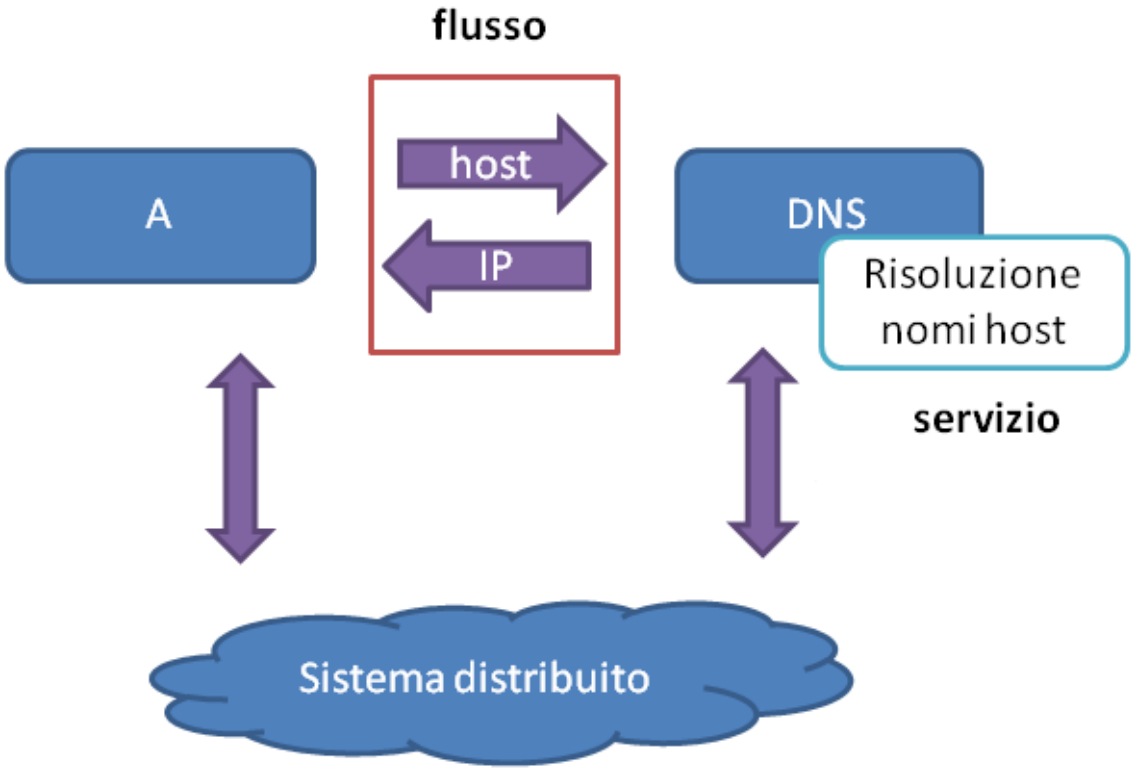

Figura 3.2: Rappresentazione di un flusso e di un servizio

- tempo di avvio del flusso
- tempo di fine flusso

Due flussi **F1** ed **F2**, originati da uno stesso nodo, possono occorrere in sequenza separati da un **tempo di interarrivo** o **intertempo** (differenza fra il tempo di chiusura di **F1** e il tempo di avvio di **F2**). Nell'esempio in figura 3.3, si considera un nodo **A** che attiva un primo flusso verso un DNS e, successivamente, un secondo flusso verso un WEB server.

E' possibile studiare l'interazione fra i due flussi, calcolando la **distribuzione di frequenze** degli intertempi: tale distribuzione rappresenta il numero di occorrenze con cui si verifica un dato intertempo nella sequenza di accessi ai due servizi ed è identificata dai due nodi **destinazione**.

La sequenza di flussi mostrata in figura 3.3, **A - DNS** e **A - WEB**, dà luogo alla distribuzione **DNS - WEB**, con un intertempo di **0.1** secondi rilevato **quattro** volte. Se due nodi client, **A** e **B**, attivano sequenzialmente **flussi DNS** e **flussi WEB**, si avrà un'unica distribuzione **DNS - WEB**, composta sia dai quattro intertempi dei flussi di A (DNS e WEB), che dai quattro intertempi dei flussi di B (DNS e WEB), per un totale di otto rilevazioni.

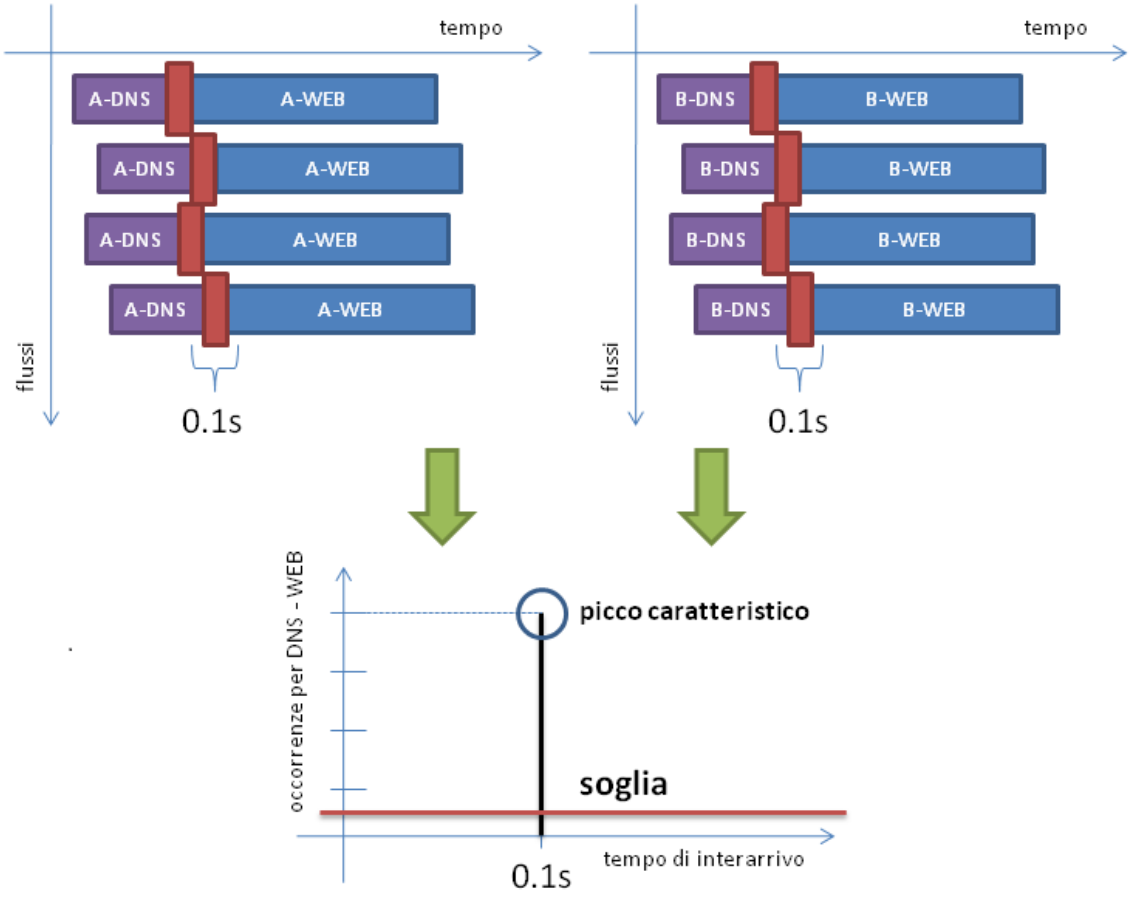

Figura 3.3: Aggregazione di intertempi fra flussi

Si definisce **picco caratteristico** della distribuzione, un intertempo con frequenza molto elevata, superiore a una data **soglia** calcolata in funzione della media e della varianza della distribuzione. In questo contesto, definiamo come **stabile** una distribuzione caratterizzata da una **bassa variabilità di intertempi**, che esibisce almeno un **picco**. E' definita **instabile** se gli intertempi esibiscono un'ampia variabilità e non si verificano **picchi**.

3.2 Soluzione applicata in DeDALO

L'approccio di DDA utilizzato in DeDALO sfrutta un modello di rilevazione basato sull'analisi dei tempi di interarrivo fra flussi di dati IP. Gli eventi d'interesse sono i pacchetti IP scambiati nel sistema, rilevati dai singoli nodi o tramite dispositivi hardware passivi. L'obiettivo della ricerca consiste nella rilevazione di dipendenze fra coppie di servizi IP, analizzando le modalità e i tempi con cui vengono acceduti sequenzialmente dai nodi client. L'assunto alla base di DeDALO, ripreso da Orion

[CZMB08], è che la **dipendenza** fra due servizi IP è una condizione che si configura in caso di un utilizzo sistematico e sequenziale dei due servizi all'interno di contesti applicativi.

Un esempio di logica applicativa consiste nell'accesso a un server **DNS**, effettuato da un **WEB browser**, prima di richiedere una **pagina WEB**. Il browser genera due flussi IP: uno verso il **DNS** e uno verso il **WEB server**. Questa sequenza **si ripropone immutata in ogni sua istanza**, causando un **ritardo costante fra i due flussi**, poichè guidata da un'applicazione.

Se una sequenza di **flussi** (caratterizzanti accessi a servizi) è **ripetutamente intervallata da uno stesso tempo**, allora è possibile affermare che i due flussi (e quindi i due servizi) sono **legati da una relazione di dipendenza**. Ad es., se un flusso WEB è sistematicamente preceduto da un flusso DNS, con intertempi molto simili, si può affermare che il servizio WEB dipende dal servizio DNS, poichè esiste una logica applicativa che prevede una sequenzialità fra i due e se il DNS dovesse smettere di funzionare, entrambi i servizi risulterebbero non funzionanti.

3.2.1 Dati di input e obiettivi

DeDALO opera con l'obiettivo di rilevare coppie di servizi IP in relazione di dipendenza. Un servizio IP è identificato da un software server che opera su una macchina fisica, per fornire una qualche funzione (di calcolo o memorizzazione) ad altri nodi della rete, ad es., un servizio di risoluzione dei nomi (DNS) o un server HTTP. Scopo di questa ricerca sarà analizzare le comunicazioni che intercorrono fra i nodi di un sistema, cercando coppie di **flussi** che occorrono, ripetutamente, in sequenza con stessi intertempi.

Il dato di partenza è il singolo pacchetto IP, scambiato fra un nodo client C e un nodo server S. Le informazioni di interesse consistono nei soli header TCP-UDP/IP, rappresentati dalla tupla (*TCP:IPc, TCP:IPs, SYN, FIN, Tempo*), nel caso TCP, o dalla tupla (*UDP:IPc, UDP:IPs, Tempo*) nel caso UDP. Gli elementi di ogni tupla sono:

- **TCP-UDP/IPc**: porta e indirizzo IP del nodo client
- **TCP-UDP/IPs**: porta e indirizzo IP del nodo server
- **SYN**: valore del flag SYN (caso TCP)
- **FIN**: valore del flag FIN (caso TCP)

- **Tempo**: istante di tempo in cui il pacchetto è stato generato

scartando il payload, in quanto inutile ai fini dell'analisi.

DeDALO analizza i singoli pacchetti per ricostruire i flussi IP che intercorrono fra due nodi. Un flusso è caratterizzato dalla tupla (*TCP:IPc, TCP:IPs, TempoInizio, TempoFine*) o (*UDP:IPc, UDP:IPs, TempoInizio, TempoFine*), di cui:

- **TCP-UDP/IPc**: porta e indirizzo IP del nodo **sorgente**
- **TCP-UDP/IPs**: porta e indirizzo IP del nodo **destinazione**
- **TempoInizio**: istante di tempo in cui è stato generato il primo pacchetto
- **TempoFine**: istante di tempo in cui è stato generato l'ultimo pacchetto

Occorre fare una precisazione sul modo in cui DeDALO gestisce i tempi. Un **contesto distribuito** presenta **problematiche di sincronizzazione**, ed è spesso necessario attuare politiche di coordinamento fra i nodi, per evitare casi di *clock skew*[1].

In questo caso il problema non si pone, poichè la caratterizzazione temporale di un **flusso** ha senso solo in relazione al nodo che lo genera. DeDALO considera questi valori temporali per pre-processare i dati ed estrarre delle **differenze** fra tempistiche **coerenti** (cfr. capitoli successivi) in quanto riferite uno stesso nodo.

Un flusso può trovarsi in tre stati: **Nuovo**, **In corso** e **Terminato**, come mostrato in figura 3.4. Si identifica un nuovo flusso nel momento in cui si rilevano procedure di *3-way handshake* (pacchetti con flag SYN attivo) o se nessun flusso **In corso** può accogliere il pacchetto analizzato (caso UDP). Un flusso si definisce **Terminato** nel momento in cui si rilevano procedure di *4-way handshake*, nel caso TCP, o dopo un timeout prefissato, nel caso UDP. Se un pacchetto non conclude un flusso nè lo avvia, allora il flusso è considerato **In corso**. Ogni flusso è contraddistinto da un tempo di inizio, estratto dal suo primo pacchetto e da un tempo di fine, estratto dal pacchetto più recente.

La raccolta dati, in DeDALO si compone di due funzioni che operano a stretto contatto: una si preoccupa di acquisire i pacchetti e l'altra di gestire l'analisi dei flussi. Il prodotto di questa fase è una sequenza di flussi, ordinata temporalmente in base al tempo di inizio di ogni flusso.

[1]Discrepanza più o meno forte fra i valori di *real time clock* esibiti dei nodi

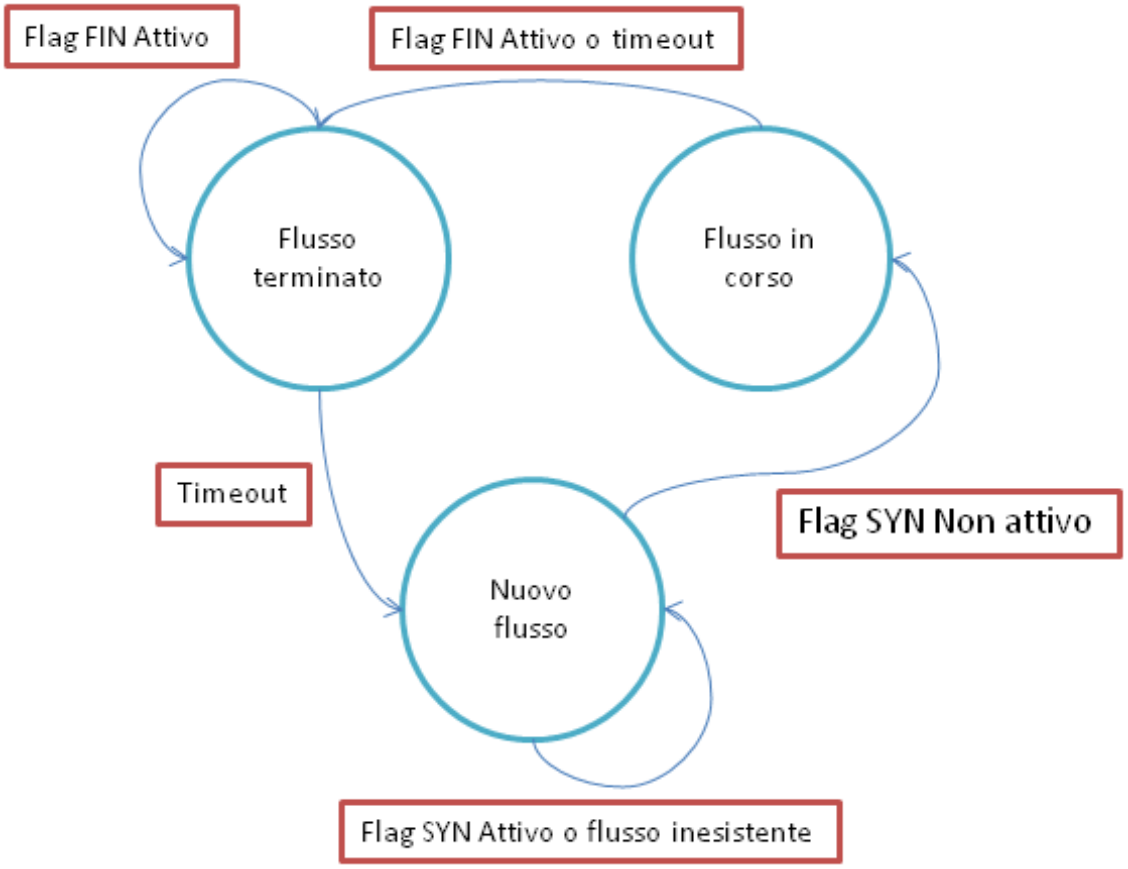

Figura 3.4: Stati di un flusso

3.3 Modello di estrazione delle dipendenze

Scopo di questo modello è determinare delle triple del tipo (*TCP:IPs1, TCP:IPs2, Grado*), partendo dai flussi identificati in fase di presa dati. La tupla rappresenta due servizi IP, S1 ed S2, acceduti sequenzialmente da uno stesso nodo **origine** entro una finestra di tempo. DeDALO analizza coppie di flussi sequenziali 'contando' il numero di volte in cui fra due flussi si è determinato uno stesso intertempo e costruendo delle distribuzioni di frequenze per ogni coppia di servizi analizzati.

3.3.1 Correlazione fra due flussi

L'analisi viene attivata contestualmente all'individuazione di un nuovo flusso **F1**, avviato al tempo T_i0. DeDALO cerca tutti i flussi in stato di **Terminato**, con lo stesso nodo **sorgente** di **F1** e che abbiano un tempo di fine flusso superiore a T_i0 **- 3 secondi**. Il motivo di questa limitazione sta nel fatto che una finestra di tre se-

condi è sufficientemente ampia da considerare sequenze generate da un'applicazione, considerando anche i risultati sperimentali presentati in [CZMB08].

Non esistono problematiche di *corretto dimensionamento* della finestra, poichè in questa fase si selezionano i possibili flussi correlati ad **F1** e ciò che interessa è **non escludere** potenziali correlazioni. Questa considerazione, ripresa anche da Orion [CZMB08], è sufficiente a garantire una selezione grossolana ma efficace per prevenire la rilevazione di falsi negativi.

Se il sistema ha individuato uno o più flussi con questi criteri, li seleziona e li confronta con **F1**. Tali flussi, *Fp1..FpX* sono detti **antecedenti** [KK01], ed hanno un tempo di fine flusso pari a $Tp_f1..Tp_fX$. Per ogni coppia **F1 - FpN** (ennesimo flusso 'antecedente'), DeDALO calcola l'intertempo fra i due flussi, sottraendo Tp_fN (tempo di fine dell'ennesimo flusso 'antecedente') a T_i0 e generando la tupla (*TCP-UDPs1, IPs1, TCP-UDPsN, IPspN, Intertempo*) con:

- **TCP-UDPs1**: porta del servizio (sul nodo **destinazione**) contattato in F1
- **TCP:IPs1**: indirizzo IP del servizio (sul nodo **destinazione**) contattato in F1
- **TCP-UDPsN**: porta del servizio (sul nodo **destinazione**) contattato in FpN
- **TCP:IPsN**: indirizzo IP del servizio (sul nodo **destinazione**) contattato in FpN
- **Intertempo**: tempo di interarrivo rilevato

il cui significato è: *E' stata rilevata un'istanza della sequenza FpN - F1, con un dato Intertempo fra i due flussi.* Ogni tupla viene associata a una distribuzione di intertempi, caratterizzata da (*TCP-UDPs1, IPs1, TCP-UDPsN, IPspN*) a cui viene aggiunta un'occorrenza sull'intertempo rilevato. Il risultato di ogni passaggio sarà una tupla del tipo (*TCP-UDPs1, IPs1, TCP-UDPsN, IPspN, distribuizione*). E' importante caratterizzare questa **distribuzione**:

- sono considerati tutti gli intertempi nell'**intervallo]0,3[secondi**
- i valori rilevati sono semplificati tramite *binning* al centesimo di secondo
- ogni distribuzione considera **trecento** intertempi
- la **frequenza media** è calcolata come rapporto fra il numero totale di campioni e il numero totale di intertempi (300)

- la **deviazione standard** consiste nello scostamento medio delle frequenze di ogni intertempo

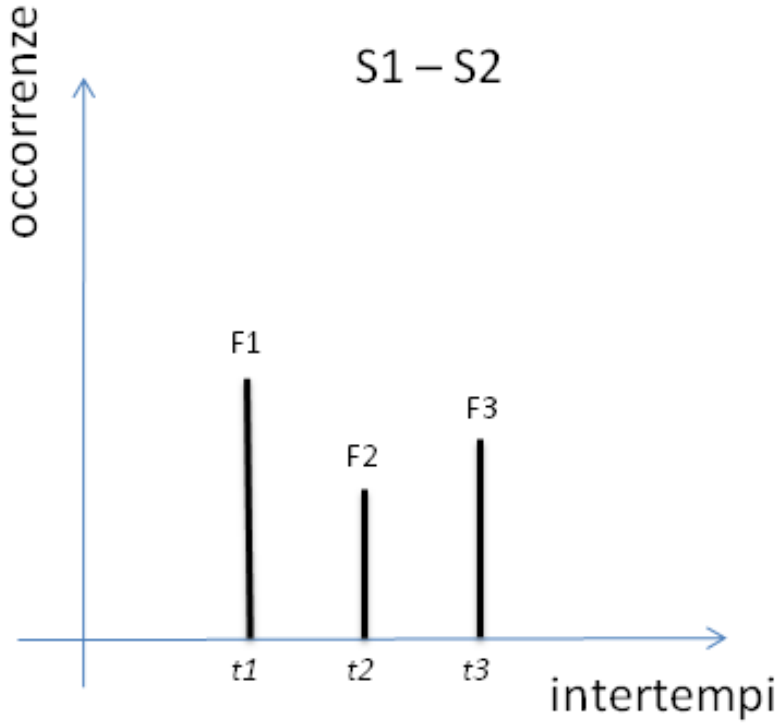

Figura 3.5: Distribuzione di intertempi fra accessi a due servizi IP

La distribuzione rappresenta la frequenza con cui uno o più nodi client accedono sequenzialmente a due servizi IP con un dato intertempo. Si possono ottenere distribuzioni con forme diverse, a seconda del numero di rilevazioni e della variabilità degli intertempi.

Consideriamo, ad es., la distribuzione in figura 3.5: sono mostrati gli intertempi fra due accessi a servizi IP, S1 ed S2. La distribuzione evidenzia tre intertempi, t1, t2 e t3, rilevati rispettivamente con frequenza F1, F2 ed F3. Il numero totale di rilevazioni è dato dalla somma delle tre frequenze.

La media è calcolata come:

$$Media = \frac{F1 + F2 + F3}{300} \tag{3.1}$$

e la varianza equivale a

$$D.st = \sqrt{\frac{(F1 - Media)^2 + (F2 - Media)^2 + (F3 - Media)^2 + (297 \times Media^2)}{300}} \tag{3.2}$$

considerando sia i tre intertempi rilevati che i restanti 297 non rilevati.

La **deviazione standard** cresce man mano che uno o più intertempi presentano frequenze distanti dalla **frequenza media**. E' il caso ad es, di un **picco** o di quegli intertempi con frequenza pari a zero. Nel caso opposto, invece, la deviazione tende a diminuire in caso di più intertempi che abbiano un numero simile di osservazioni.

Per fare un esempio, confrontiamo due distribuzioni:

- una con 600 campioni su un singolo intertempo
- una con 600 campioni disposti equamente sui 300 intertempi (hanno tutti frequenza pari a 2)

La **deviazione standard** assume, nel primo caso valore 17 e, nel secondo caso valore 1. Per evitare confusione è bene distinguere le metriche che interessano le frequenze, dalle metriche orientate ai tempi: *non siamo interessati a determinare l'intertempo medio, ma la frequenza media con cui un qualsiasi intertempo può verificarsi.* E' importante fare questa distinzione, poichè il tipo di analisi su cui si basa DeDALO, va alla ricerca di ripetizioni 'sistematiche' di una stessa sequenza di accessi, cercando le tracce di tale 'sistematicità' nella 'stabilità' con cui due eventi sono intervallati da uno stesso tempo, non importa quale.

3.3.2 Identificazione di una dipendenza

Ottenuta una distribuzione, DeDALO analizza i suoi valori per determinare se i due servizi sono in relazione di **dipendenza**. L'analisi viene condotta cercando uno o più intertempi che abbiano una frequenza 'particolarmente elevata' in grado, cioè, di superare una *soglia di dipendenza.* Tali intertempi sono detti **picchi caratteristici** o 'spike' (cfr.[CZMB08]). La soglia è determinata in funzione della **frequenza media** e della **deviazione standard**. In lavori come Orion [CZMB08], si suggerisce di calcolare la soglia come:

$$Soglia = Media + K \times DeviazioneStandard \tag{3.3}$$

con K = 3. Per ogni distribuzione, viene calcolata la soglia di 'dipendenza' e valutata, rispetto ad essa, la frequenza di ogni intertempo rilevato. Se la distribuzione presenta uno o più **picchi caratteristici**, DeDALO considera i rispettivi sevizi come **dipendenti**. Il **picco** si configura come la ripetizione sistematica di due accessi a servizi **con uno stesso intertempo**. Esso determina l'evidenza di un contesto

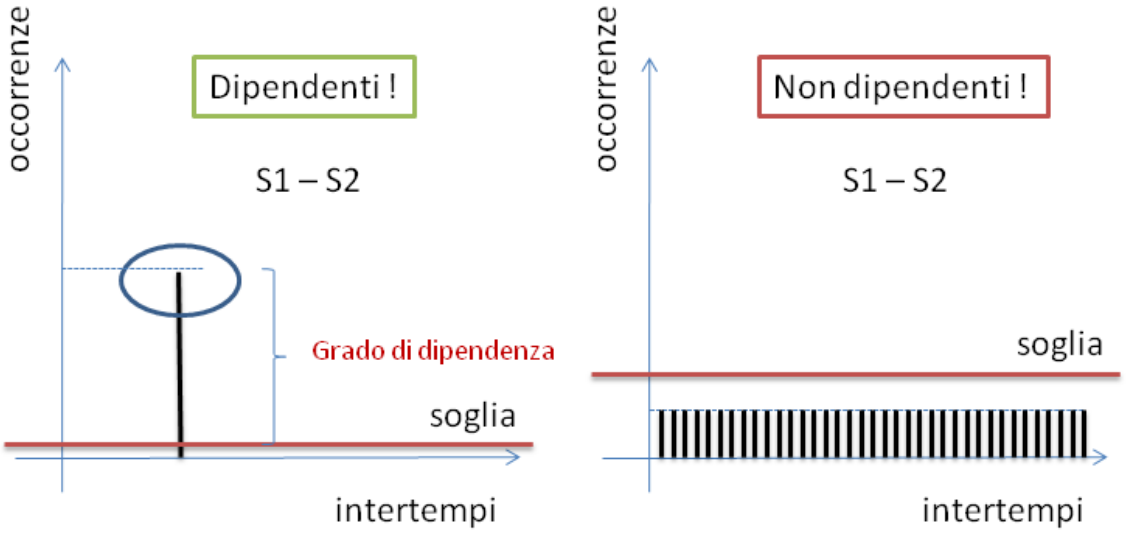

Figura 3.6: Massimo e minimo grado di dipendenza

applicativo che accede sistematicamente ai due servizi e che quindi li lega da una relazione di dipendenza.

3.4 Calcolo del grado di dipendenza

Una volta identificati due servizi dipendenti, DeDALO calcola il **grado** di correlazione fra i due. Due distribuzioni con stesso numero di campioni, possono caratterizzare diversi livelli di dipendenza, a seconda della disposizione dei **picchi**, del loro numero e del numero di rilevazioni su ogni picco. Si afferma che il **grado di dipendenza fra due servizi** sia tanto più forte quanto maggiore è la concentrazione di campioni su un singolo picco, il che significa che un'applicazione sta eseguendo un flusso logico sempre uguale, sistematico e non soggetto a condizionamenti. Se un servizio fallisce, allora è presumibile che anche il servizio **dipendente** risulterà non funzionante.

Il grado di dipendenza è una metrica che rappresenta la *probabilità* che un servizio S2 dipenda, per il suo funzionamento, da un servizio S1. La variabilità di tale metrica è soggetta due fattori:

- complessità applicativa nell'accedere ai due servizi
- sistemi di caching, load-balancing e replicazione

Maggiore è la complessità applicativa, nel definire una sequenza fra due accessi a servizi, maggiore sarà la dispersione negli intertempi rilevati. Per fare un esempio, se

un software accede a un servizio S1, effettua dei calcoli complessi e poi accede a un servizio S2, sarà difficile ripetere uno stesso intertempo fra S1 ed S2 con sistematicità. In questo caso, è poco probabile che S1 sia 'vitale' per l'accesso ad S2 e si avranno distribuzioni con buona numerosità ma alta variabilità negli intertempi.

L'impatto di sistemi di caching o load-balancing è analogo: se un nodo client accede per due volte a un servizio S1 (in un breve arco di tempo), accederà solo la prima volta al DNS, memorizzando localmente la coppia nome host - IP. In questo caso, analizzando i due flussi S1 in relazione al DNS, si avrà che:

- La prima istanza della coppia **Flusso DNS-flusso S1** avrà un intertempo pari a **T1**
- La seconda istanza della coppia **Flusso DNS-flusso S1** avrà un intertempo pari a $\mathbf{T2} > T1$

dando luogo a una distribuzione con due campioni su due intertempi.

Altro fattore di variabilità è dato da servizi replicati o soggetti a bilanciamento del carico, per cui uno stesso accesso a un servizio S1, genera più flussi con diversi nodi **destinazione**, selezionati volta per volta da un elemento intermediario. Ai fini del calcolo del grado, è importante determinare una metrica che riesca a tracciare i diversi livelli di variabilità in fuzione dei fattori appena visti. Il dato di partenza è la distribuzione degli intertempi, caratterizzata da:

- un numero totale di rilevazioni
- un numero di intertempi rilevati
- una soglia di dipendenza
- uno o più **picchi**

Si assume che il massimo grado di dipendenza, si verifichi per distribuzioni con N osservazioni totali concentrate su un singolo intertempo, mentre il grado minimo si ha per distribuzioni con N osservazioni distribuite fra tutti i possibili intertempi.

Partendo dai due casi estremi, si nota come la differenza fra un **picco** (se presente) e la soglia di dipendenza rifletta la 'forza' con cui due servizi sono relazionati. In particolare, si avrà un basso grado di dipendenza per soglie elevate (alto numero di intertempi) e **picchi** con poche osservazioni. Al contrario, si avrà un alto grado di dipendenza per soglie molto basse (pochi intertempi rilevati) e **picchi** con molte osservazioni. Un esempio è dato dalla figura 3.7

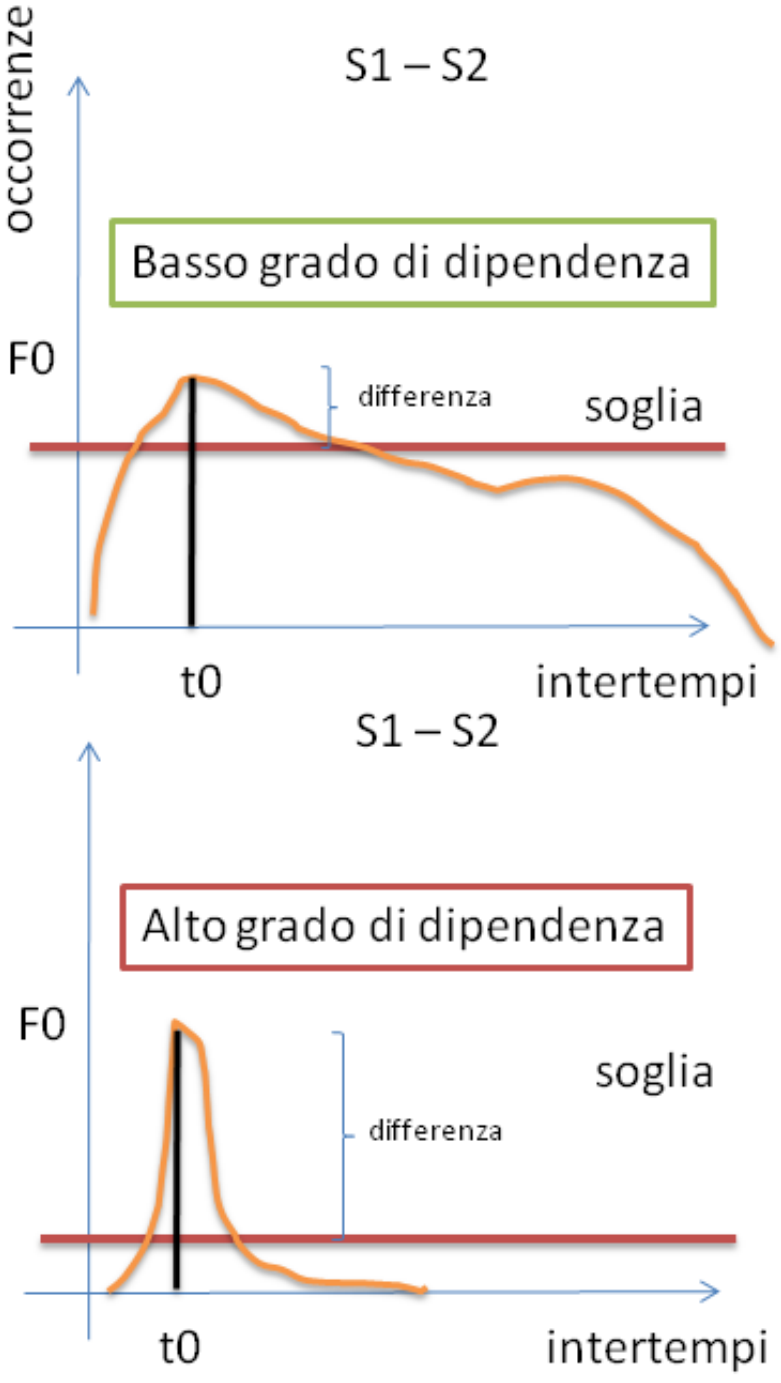

Figura 3.7: Grado di dipendenza

Formulando quest'analisi in termini matematici, si ottiene la formula probabilistica del grado di dipendenza fra due servizi:

$$Grado = 1 - (\frac{Soglia}{FrequenzaPiccoMassimo}) \tag{3.4}$$

Si considera il complemento a uno del rapporto fra la soglia e la frequenza del pic-

co massimo (se presente). Tale rapporto varia nell'intervallo]0,1] e assume valore minimo (massimo grado di dipendenza) quando la soglia tende a zero e tutti i campioni sono concentrati su un tempo (massima sistematicità). Assume, invece valore 1 (minimo grado di dipendenza) quando la soglia coincide con il **picco** massimo.

3.5 Parametrizzazione del modello e pulizia dei dati

Il modello di analisi si contraddistigue, rispetto a molti altri per la particolarità di non richiedere alcun intervento da parte dell'utente. La solidità degli assunti, la loro semplicità e le considerazioni sul dominio, rendono DeDALO totalmente avulso da attività di tuning o setup.

La finestra di tempo ha un impatto quasi irrilevante sul risultato ed è impostata per default a 3 secondi solo per stabilire un valore massimo. La scala di intertempi a cui siamo interessati è dell'ordine del decimo di secondo e non c'è rischio di stabilire finestre troppo ampie o troppo strette.

Ogni coppia di flussi viene valutata in virtù della distribuzione caratteristica degli intertempi. Può accadere, a volte, che fattori esterni turbino il traffico al punto da far variare gli intertempi in modo non trascurabile, dando luogo a distribuzioni più sparse e con soglie più alte. Questo può portare ad errori nella valutazione del grado o, addirittura, alla non rilevazione di una dipendenza.

Generalmente, l'impatto di un fattore di disturbo ha effetto su tempi non rilevanti, determinando delle fluttuazioni nelle occorrenze tali a volte da creare dei picchi 'non caratteristici' che si affiancano a quelli 'caratteristici'. Analizzando queste fluttuazioni nel dominio della frequenza, si nota come l'impatto di fattori di disturbo generi delle distribuzioni con uno spettro più ampio, di quanto non sia nel caso di distribuzioni non 'turbate'.

E' possibile, quindi, attuare delle tecniche di pulizia dei dati, applicando filtri passa-basso alle distribuzioni degli intertempi, che sono trattate come dei segnali finiti. Applicando dei tagli a frequenze fissate, si riesce ad escludere queste fluttuazioni senza perdere dati.

3.6 Ambiti applicativi

DeDALO è un tool di analisi basato sull'evoluzione del traffico di rete; per operare necessita di una sorgente da cui estrarre pacchetti IP, come una scheda di rete in attività o un file con il log di attività precedenti. Nel primo caso si parla di analisi 'Online', effettuata sul sistema man mano che questo evolve (e con esso i dati scambiati al suo interno). Nel secondo caso, invece, si parla di analisi 'Offline'. Entrambe presentano sia vantaggi che difficoltà.

L'analisi online è tipica di quei sistemi su cui si ha un certo controllo, disponendo di uno o più nodi da cui 'osservarne' lo stato ed eventualmente perturbarlo con traffico di test. In questo caso, DeDALO può essere installato su uno o più macchine e connesso ad una NIC per 'ascoltare' e analizzare il traffico in transito. I vantaggi di un'analisi di questo tipo consistono nella possibilità di ottenere dati 'freschi', che riflettono lo stato del sistema permettendo di intervenire in breve tempo in caso di situazioni critiche. La difficoltà di un approccio online sta nel fatto che servono molti nodi di osservazione, in modo da superare la compartimentazione in Autonomous System e soprattutto lo switching, che limita la 'visibilità' del traffico al solo nodo di osservazione. L'approccio online più efficace richiederebbe l'installazione di sistemi di ascolto direttamente sui router o su nodi in reti broadcast (senza switch) ma entrambe le soluzioni sono strutturalmente infattibili. La soluzione online proposta per DeDALO consiste nell'installazione del tool in quanti più nodi possibile e centralizzare le osservazioni per effettuare un'analisi globale.

La soluzione di tipo offline si basa su tracciati di attività rilevati con un certo ritardo rispetto all'analisi, per questioni di privacy, difficoltà logistiche o politiche di amministrazione della rete. E' anche tipica di rilevazioni su sistemi simulati, in cui si utilizzano file di log per tracciare il traffico generato dal simulatore.

Per testare DeDALO su ampi scenari, si è scelto di attuare un'analisi offline di questo tipo, implementando dei modelli simulativi per replicare topologie ed applicazioni distribuite, con livelli di dipendenza variabili.

Capitolo 4

Il framework di analisi delle dipendenze

In questo capitolo si descrive la struttura applicativa di DeDALO. Saranno presentati gli elementi di cui si compone, gli algoritmi e le singole funzionalità, descrivendo come operano ed interagiscono per attuare la **DDA**.

4.1 Struttura

Il framework di analisi si compone di due moduli:

- *backend* **locale** di pre-processamento del traffico
- *frontend* **centralizzato** di aggregazione e presentazione dei risultati

La struttura è progettata a due livelli per disaccoppiare la componente di rilevazione dalla componente di presentazione. I due componenti possono essere distribuiti su di una rete e permettono di implementare delle strutture di analisi basate sull'utilizzo contemporaneo di più *backend*.

Il *frontend*, in questo caso, funge da **hub**, ricevendo le informazioni da più fonti e aggregandole per poi analizzarle. Il backend ha il compito di acquisire il traffico IP, identificare i diversi flussi applicativi, ed estrarre distribuzioni di intertempi, che invierà al frontend per l'analisi delle dipendenze.

Il backend opera in quattro fasi:

- Lettura del traffico

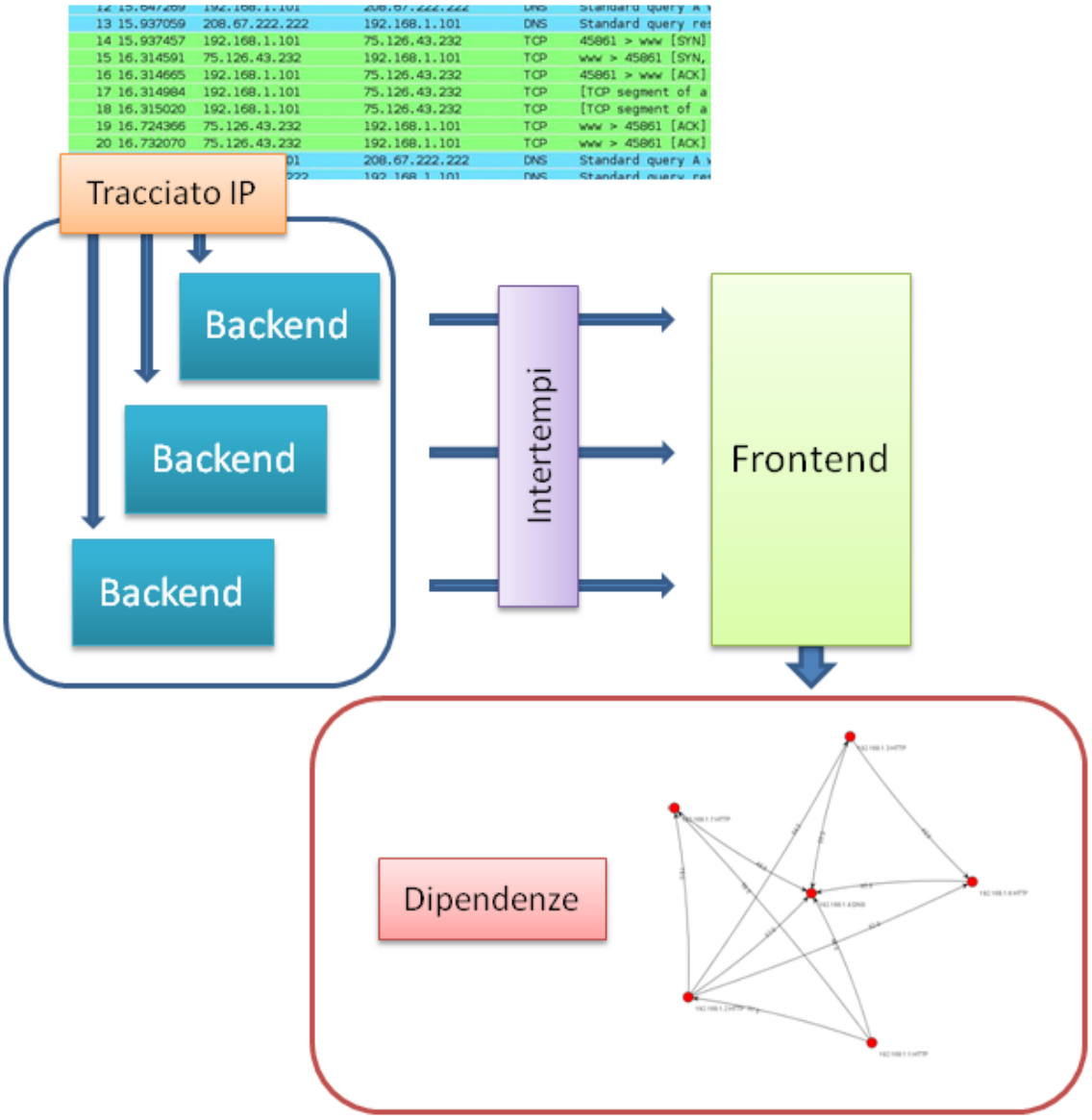

Figura 4.1: Struttura di DEDALO

- Identificazione dei flussi
- Calcolo delle distribuzioni di intertempi
- Invio delle distribuzioni al frontend

Il funzionamento del frontend si articola in cinque fasi:

- Ricezione delle distribuzioni dai backend
- Aggregazione dei dati ricevuti
- Analisi delle distribuzioni e ricerca delle dipendenze

- Calcolo del grado di dipendenza
- Presentazione dei risultati

4.2 Backend di rilevazione

Il backend ha il compito di acquisire un tracciato IP e raccogliere statistiche sulle distribuzioni di intertempi fra coppie di accessi a servizi IP. Questa operazione si basa sulla ricostruzione dei flussi di comunicazione che intercorrono fra due nodi di rete. La prima fase, fra le quattro del backend, consiste nell'acquisizione del traffico, accedendo a interfacce di rete o leggendo tracciati acquisiti in precedenza.

L'algoritmo di estrazione accede alla sorgente di traffico IP e decodifica ogni pacchetto in forma di strutture dati. Il pacchetto viene privato del payload e 'semplificato' nella sua struttura, lasciando solamente quegli header utili a identificare i nodi coinvolti, il tempo d'invio e i flag di controllo.

La struttura risultante viene passata all'algoritmo di individuazione dei flussi. Un flusso è costituito da una serie di pacchetti scambiati ininterrottamente fra due nodi. Lo scopo di DeDALO, in questa fase, è ricostruire i flussi IP a partire dai singoli pacchetti. Ogni flusso sarà catalogato da DeDALO memorizzando il nodo **sorgente** (solitamente un client), il nodo **destinazione** (un nodo server), il *Tempo di inizio* del flusso (preso dal suo primo pacchetto) e il *Tempo di fine* flusso.

Per ogni flusso che DeDALO identifica, viene avviato l'algoritmo di calcolo della distribuzione degli intertempi. Questo si preoccupa di valutare il nuovo flusso in relazione ai flussi terminati in precedenza, contando il numero di volte in cui l'intertempo fra ogni coppia di flussi ha assunto un certo valore. Il backend raccoglie queste statistiche, salvando una tripla composta dai due flussi e da un vettore di occorrenze per gli intertempi, che poi invia al frontend.

4.2.1 Acquisizione dei pacchetti

In questa prima fase, l'algoritmo di DDA accede a una fonte di traffico ed estrae sequenzialmente dei pacchetti IP. L'algoritmo si articola tramite due cicli annidati: il ciclo più esterno ha il compito di mantenere il tool 'attivo', stabilendo dei meccanismi di attesa per 'interrogare' la fonte sullo stato dei pacchetti acquisiti. Il ciclo più interno ha lo scopo di scorrere la lista di pacchetti disponibili, indirizzando il primo pacchetto utile. All'interno di questo ciclo hanno luogo tutte quelle operazioni

Algoritmo 1 Estrazione di un pacchetto

```
while not timeout do
  while pacchettoIP ← acquisisciPacchetto() do
    if pacchettoValido(pacchettoIP) then
      src ← estraiIPSorgente(pacchettoIP)
      dst ← estraiIPDestinazione(pacchettoIP)
      porta ← estraiPorta(pacchettoIP)
      tempo ← estraiTempoInvio(pacchettoIP)
      syn ← estraiSYN(pacchettoIP)
      fin ← estraiFIN(pacchettoIP)
      trasp ← estraiProtocolloTrasporto(pacchettoIP)
      gestioneFlussi(src, dst, porta, tempo, syn, fin, trasp)
    end if
  end while
end while
```

volte a manipolare il pacchetto per renderlo adatto all'analisi, inquadrandolo in una struttura composta da:

- Indirizzo IP e porta del nodo **origine**
- Indirizzo IP e porta del nodo **destinazione**
- Protocollo di trasporto
- Tempo di invio del pacchetto (relativo al nodo)
- Flag SYN (se previsto)
- Flasg FIN (se previsto)

La struttura ottenuta è sottoposta a una serie di controlli per evitare:

- Frame provenienti da reti fisiche non gestite
- Pacchetti inviati in broadcast
- Pacchetti su interfacce di loopback
- Pacchetti estranei all'ambiente monitorato
- Protocolli di trasporto diversi dal TCP o UDP

A questo punto, la struttura identifica quei dati fondamentali che descrivono un pacchetto unicast, TCP o UDP, inviato su rete IP da un nodo A a un nodo B.

4.2.2 Identificazione di un flusso

Questo algoritmo riceve in input un pacchetto e lo analizza per identificare il flusso corrispondente. Le informazioni che caratterizzano un flusso sono cinque:

- Indirizzo IP del primo nodo
- Indirizzo IP del secondo nodo
- Porta TCP o UDP
- Tempo di *Inizio flusso*
- Tempo di *Fine flusso*

DeDALO mantiene due liste di flussi che indicizza in base ai nodi **end-point** e alla porta TCP-UDP:

- la prima lista traccia i flussi attualmente **In corso**
- la seconda lista traccia i flussi **Terminati** entro 3 secondi

Entrambe memorizzano, per ogni entry, il *Tempo di inizio* flusso, rilevato dal suo primo pacchetto, e il *Tempo di fine* flusso, rilevato dal suo ultimo pacchetto. Per ogni pacchetto estratto (inviato da un nodo A a un nodo B), DeDALO interroga la lista di flussi **In corso**, cercandone uno che sia già attivo fra A e B, per la data porta TCP-UDP, senza considerare l'ordine dei nodi. Se la ricerca ha esito positivo, DeDALO estrae la entry e analizza il pacchetto. La prima operazione consiste nell'aggiornare il tempo di *Fine flusso* della entry con il tempo di invio del pacchetto. Successivamente, si analizzano i flag e il protocollo di trasporto:

- **Pacchetto TCP con flag FIN non attivo**: DeDALO considera il flusso ancora **In corso**
- **Secondo pacchetto TCP con flag SYN attivo**: DeDALO considera il flusso ancora **In corso**
- **Pacchetto TCP con flag FIN attivo**: DeDALO sposta la entry dalla lista di flussi **In corso** alla lista di flussi **Terminati**
- **Pacchetto UDP inviato entro 3 secondi dal tempo di *Fine flusso***: DeDALO considera il flusso ancora **'In corso'**

Algoritmo 2 Identificazione dei flussi

```
flusso ← null
if esisteFlussoInCorso(src, dst, porta) then
  if pacchettoUDP(trasp) then
    timeout ← tempoFineFlusso(flusso) + 3sec
    if tempo > timeout then
      spostaEntrySuFlussiTerminati(flusso)
      flusso ← creaNuovaEntry(src, dst, porta)
    else
      flusso ← ottieniFlussoInCorso(src, dst, porta)
    end if
  else
    flusso ← ottieniFlussoInCorso(src, dst, porta)
  end if
else
  if pacchettoUDP(trasp) then
    flusso ← creaNuovaEntry(src, dst, porta)
  else
    if finAttivo(fin) then
      spostaEntrySuFlussiTerminati(flusso)
    else
      if synAttivo(syn) then
        analizzaIntertempi(flusso)
      end if
    end if
  end if
end if
```

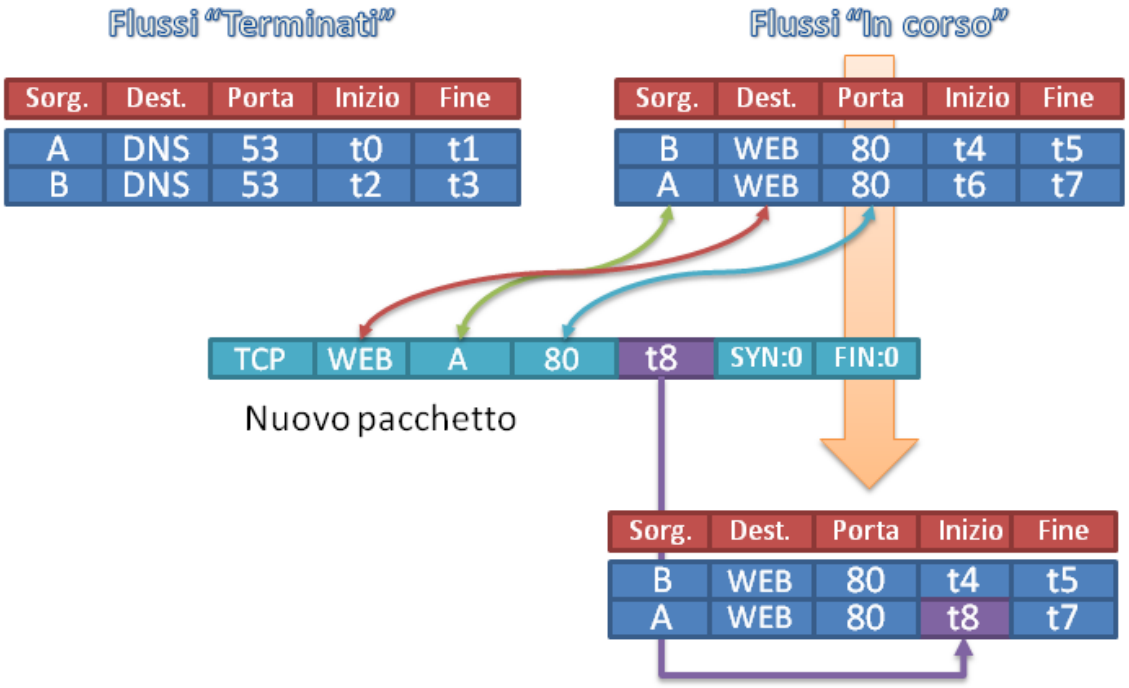

Figura 4.2: Ricostruzione di un flusso

- **Pacchetto UDP inviato oltre 3 secondi dopo il tempo di *Fine flusso***: DeDALO sposta la entry dalla lista di flussi **In corso** alla lista di flussi **Terminati** e istanzia una nuova entry nella lista di flussi **In corso**, con tempo di *Inizio flusso* pari al tempo di invio del pacchetto

quest'ultimo caso è particolare, poichè il protocollo UDP non permette di identificare la fine di una comunicazione, come invece avviene con il flag FIN nel caso TCP. Si assume, quindi, che i pacchetti di uno stesso flusso UDP debbano essere separati da non più di tre secondi, altrimenti il pacchetto viene considerato come parte di un nuovo flusso.

Se DeDALO non identifica un flusso **In corso**, adatto ad accogliere il pacchetto, possono verificarsi tre casi:

- **Pacchetto TCP con flag SYN non attivo**: DeDALO scarta il pacchetto
- **Primo pacchetto TCP con flag SYN attivo**: DeDALO istanzia una nuova entry per mappare un flusso **In corso**, con tempo di *Inizio flusso* pari al tempo di invio del pacchetto
- **Pacchetto UDP**: DeDALO istanzia una nuova entry per mappare un flusso **In corso**, con tempo di *Inizio flusso* pari al tempo di invio del pacchetto

4.2.3 Analisi fra coppie di flussi

Questa fase si attiva contestualmente all'identificazione di un flusso F1 ed ha lo scopo di 'contare' le frequenze degli intertempi fra F1 e le istanze dei flussi terminati. In questa fase si compie un passo basilare per condurre la DDA: si analizza la 'sistematicità' del traffico IP. Per ogni nuovo flusso F1, viene interrogata la lista di flussi **Terminati**, cercando quei flussi che abbiano lo stesso nodo **sorgente** di F1, in modo da esser certi che fra i due ci sia effettivamente un legame.

Il risultato di questa ricerca consiste in una serie di flussi, potenziali 'ex-requisiti', da correlare con il F1. La correlazione avviene considerando il **Tempo di interarrivo** fra il *Tempo di inizio* di F1 e il *Tempo di fine* dell'ennesimo flusso **Terminato**. Per ogni coppia di flussi analizzati, si estrae una tupla consistente nei due nodi **destinazione** e nell'intertempo T. DeDALO mantiene una lista apposita per salvare

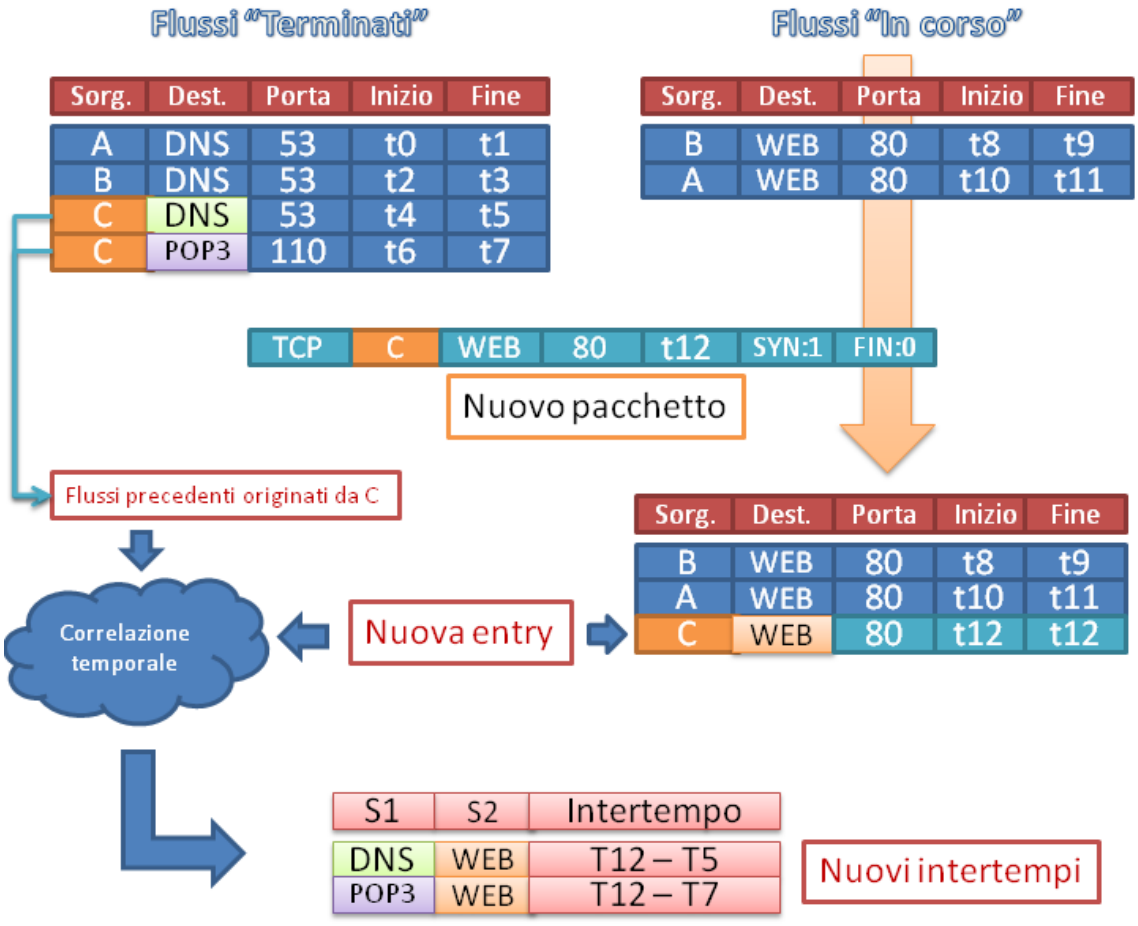

Figura 4.3: Correlazione temporale di un nuovo flusso

questi risultati: tale lista identifica ogni entry in base ai nodi **destinazione** dei due flussi, a cui viene associato un vettore di intertempi fra 0 e 3 secondi. Per ogni

coppia analizzata, DeDALO estrae i due nodi **destinazione** e interroga la lista per cercare l'array corrispondente. Se la ricerca ha successo, DeDALO salva il risultato:

- tronca T al centesimo di secondo (0.021 diventa 0.02)
- moltiplica T troncato per 100 (0.02 diventa 2) e ottiene un indice I
- incrementa di uno il valore dell'array in corrispondenza di I

i possibili intertempi sono in tutto 300 (cento per ogni secondo, per un totale di tre secondi). Nel caso in cui la ricerca fallisse, DeDALO crea una nuova entry. Il

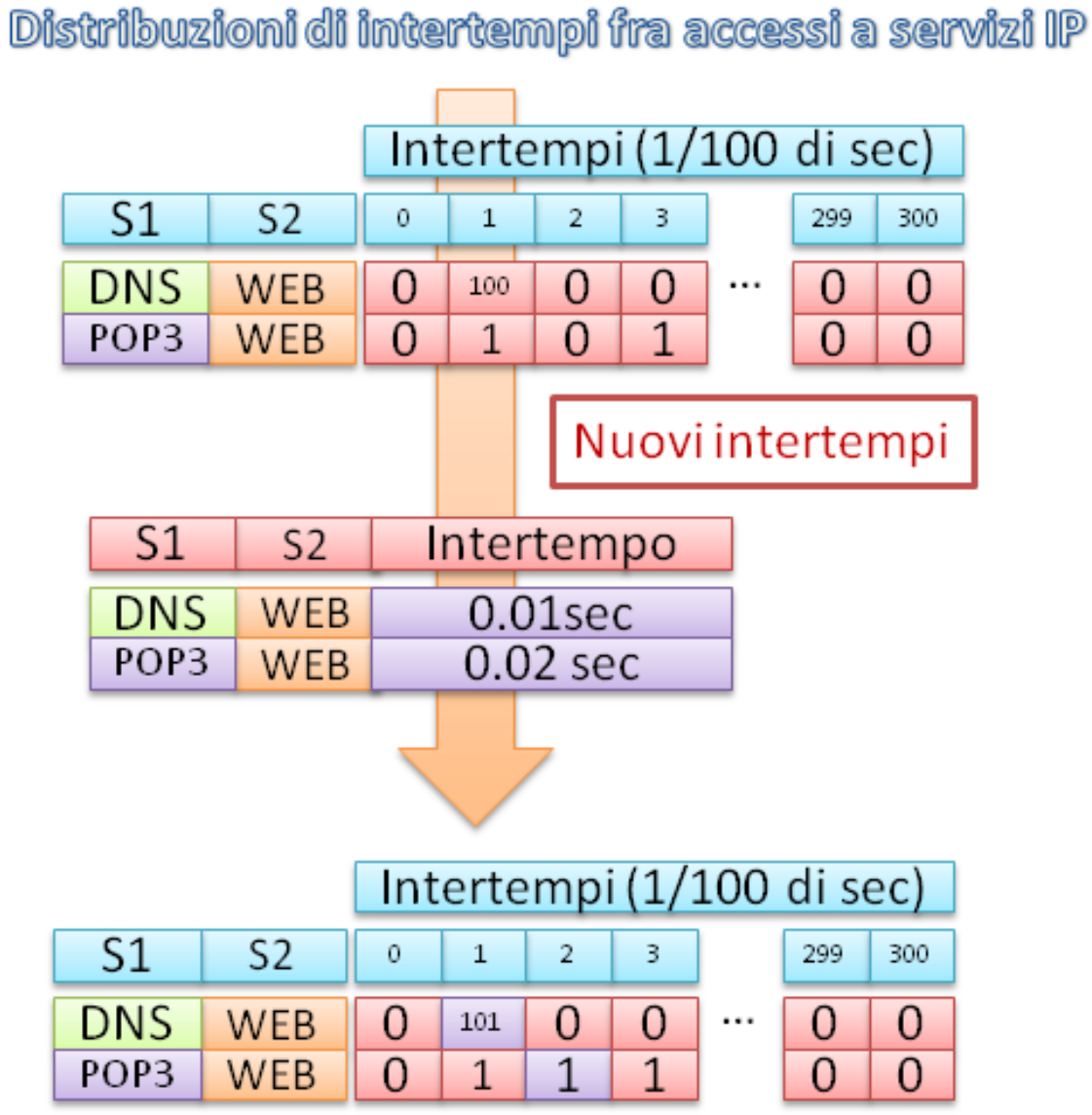

Figura 4.4: Aggiunta di un intertempo alla distribuzione

risultato di questa fase è una lista composta da coppie di nodi e un array di frequenze di intertempi.

Algoritmo 3 Analisi dei flussi

```
inizio ← estraiTempoInizio(F1)
sorgente ← estraiNodoSorgente(F1)
dest1 ← estraiNodoDestinazione(F1)
finestra ← inizio − 3sec
flussi ← estraiFlussiTerminati(finestra, sorgente)
while F2 ← estraiProssimoFlusso(flussi) do
  fine ← estraiTempoFine(F2)
  dest2 ← estraiNodoDestinazione(F2)
  T ← fine − inizio
  intertempi ← null
  if esisteArrayIntertempi(dest1, dest2) then
    intertempi ← estraiArrayIntertempi(dest1, dest2)
  else
    intertempi ← creaArrayIntertempi(dest1, dest2)
  end if
  T ← troncaAlCentesimoDiSec(T)
  I ← T × 100
  intertempi[I] ← intertempi[I] + 1
end while
```

4.2.4 Invio dei risultati

In quest'ultima fase si comunica al frontend il risultato ottenuto. Sono inviate tutte le distribuzioni di intertempi, caratterizzate dai rispettivi servizi IP, che abbiano almeno cento campioni e una deviazione standard minima pari a 5.

Algoritmo 4 Invio dei risultati

```
distribuzioni ← estraiDistribuzioni()
while distribuzione ← estraiProssimaDistribuzione(distribuzioni) do
  occorrenze ← estraiOccorrenze(distribuzione)
  devSt ← estraiDevSt(distribuzione)
  if occorrenze ≥ 100 or devSt ≥ 5 then
    S1 ← estraiPrimoServizio(distribuzione)
    S2 ← estraiSecondoServizio(distribuzione)
    intertempi ← estraiIntertempi(distribuzione)
    inviaDistribuzione(S1, S2, intertempi)
  end if
end while
```

4.3 Frontend di aggregazione e analisi

Il frontend ha il compito di ricevere, aggregare e analizzare i dati ricevuti dai backend, e calcolare il grafo globale delle dipendenze. E' composto da un server TCP, per la ricezione delle distribuzioni, da un motore di rilevazione, per estrarre le dipendenze e da un motore grafico per mostrare i risultati.

4.3.1 Aggregazione dei dati

In questa fase si procede all'aggregazione di una distribuzione ricevuta da un **backend** a dati già noti al *frontend*. E' mantenuta, a tal proposito, una lista di distribuzioni di intertempi per coppie di servizi S1 ed S2. Per ogni distribuzione ricevuta, il frontend interroga la lista alla ricerca di una precedente distribuzione fra S1 ed S2. Se la ricerca va a buon fine (come mostrato in figura 4.5), il *frontend*

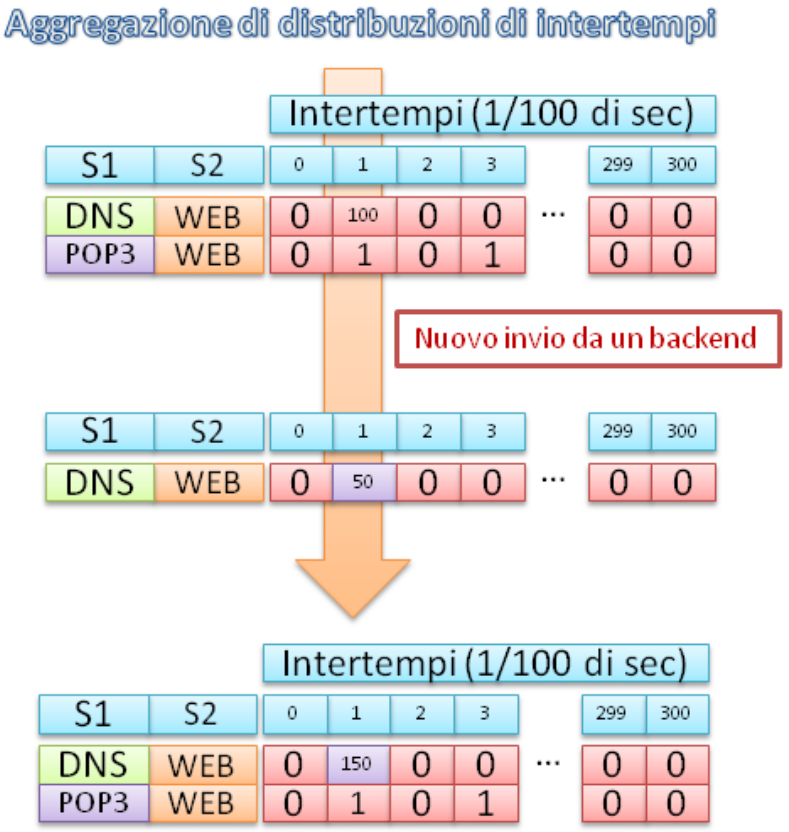

Figura 4.5: Aggregazione di dati ricevuti da un backend

somma le frequenze ricevute alla distribuzione trovata e salva i nuovi valori. Se la ricerca fallisce, viene creata una nuova entry, valorizzata con la distribuzione ricevuta

e associata alla coppia di servizi. L'algoritmo è identico a quello del backend per l'analisi di coppie di flussi:

Algoritmo 5 Aggregazione di distribuzioni

```
distribuzione ← null
if esisteDistribuzione(S1, S2) then
  distribuzione ← estraiDistribuzione(S1, S2)
else
  distribuzione ← creaDistribuzione(S1, S2)
end if
sommaIntertempi(distribuzione, datiRicevuti)
```

4.3.2 Selezione delle coppie e valutazione delle dipendenze

In questa fase, attivata ad ogni arrivo, si verifica la possibilità che fra due flussi ci sia effettivamente una dipendenza. Le condizioni sono due, una preliminare e una di analisi. Sono considerate, ai fini dell'analisi, tutte e sole le distribuzioni che abbiamo almeno 300 campioni. Se una distribuzione rispetta questo requisito, viene passata all'algoritmo di valutazione.

Algoritmo 6 Ricerca delle dipendenze

```
media ← estraiMedia(distribuzione)
devSt ← estraiDevSt(distribuzione)
soglia ← media + 3 × devSt
picchi ← estraiPicchi(distribuzione, soglia)
if conteggio(picchi) ≥ 1 then
  frequenzaMassima ← estraiFrequenzaMassima(picchi)
  percentuale ← (1 − (soglia/frequenzaMassima))
  if percentuale > 0 then
    S1 ← ottieniPrimoNodo(distribuzione)
    S2 ← ottieniSecondoNodo(distribuzione)
    aggiungiDipendenza(S1, S2, percentuale)
  end if
end if
```

L'algoritmo opera calcolando la soglia oltre cui valutare la presenza o meno di 'picchi'. Se una distribuzione esibisce uno o più picchi, si considera una relazione di dipendenza fra i servizi, S1 ed S2, che la contraddistinguono. In tal caso, viene calcolata la percentuale di dipendenza, rapportando la soglia di dipendenza alla frequenza del picco massimo. Il risultato di questa fase è una lista di tuple composte da **Servizio IP 1**, **Servizio IP 2**, **Percentuale di dipendenza**.

Capitolo 5

Ambiente di sperimentazione

Presentiamo, in questo capitolo, l'ambiente di simulazione sviluppato per testare DeDALO. E' basato sul simulatore *Network Simulator 3* (**NS3**), erede del più noto **NS2**, storica piattaforma utilizzata in ambito accademico e industriale. DeDALO è

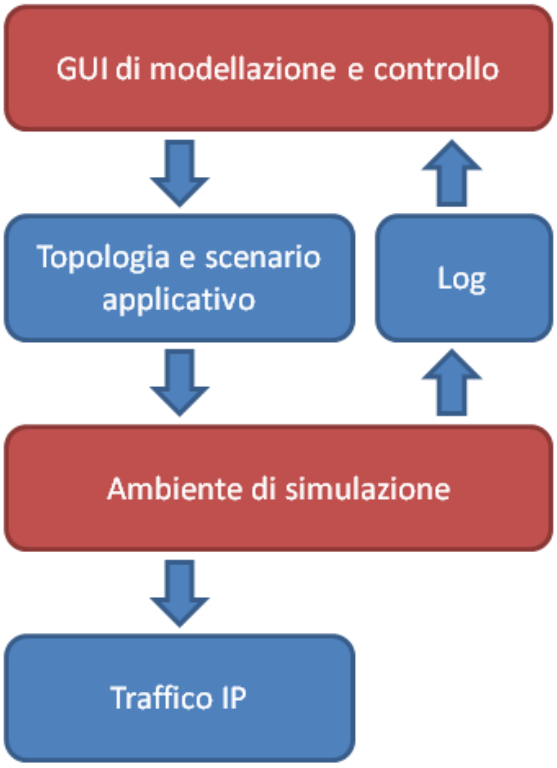

Figura 5.1: Struttura dell'ambiente di sperimentazione

in grado di operare su contesti applicativi che spaziano dalle piccole reti locali alle grandi infrastrutture geografiche e, per meglio testarne il funzionamento, si è scelto di implementare modelli simulativi di comuni protocolli IP. Tali modelli rappresentano

lo strato applicativo di un sistema distribuito e sono composti da sequenze più o meno complesse di interazioni richiesta - risposta.

L'ambiente di sperimentazione è composto da un *backend*, dedicato alla simulazione, e da un *frontend*, per generare topologie e scenari applicativi. La topologia è definita graficamente attraverso una GUI e viene esportata al backend tramite un linguaggio di descrizione che contempla sia la struttura fisica che il routing e il livello applicativo. Il modulo di decodifica, lato backend, si preoccupa di istanziare la topologia in NS3, acquisendo le definizioni da un file testo, e allocando le risorse per descriverla in RAM.

Saranno illustrate le modalità con cui condurre simulazioni per analisi DDA. Sarà poi descritto l'ambiente di simulazione, introducendo NS3, i modelli protocollari e il frontend.

5.1 Simulazione e analisi

Modellare un sistema distribuito significa descrivere i meccanismi che regolano i tre livelli di cui si compone:

- Strato applicativo
- Routing
- Canali di comunicazione

Il termine 'topologia', in questo ambito, si riferisce al modo in cui sono interconnessi gli elementi di un sistema distribuito, definendo una struttura di tipo 'grafo' composta da nodi (calcolatori) e archi (collegamenti). Strutturalmente, si tratta di rappresentare reti fisiche compartimentate (*Autonomous System*) e nodi di interscambio (*router*), descrivendo i due livelli più bassi di un sistema distribuito. Nella definizione di una topologia, si possono utilizzare diversi approcci, basati su metodologie e tool che assistono l'utente nel dislocare e interconnettere nodi di rete.

Esistono generatori come BRITE [MLMB01], che applicano modelli parametrici di crescita ai nodi delle sottoreti (ad es. il modello di routing Albert-Barabasi [AB02]): l'utente imposta i parametri, definendo ad es., il numero di AS, il numero di nodi e la banda disponibile, e il generatore applica un modello matematico, il cui risultato è un grafo o una matrice d'incidenza.

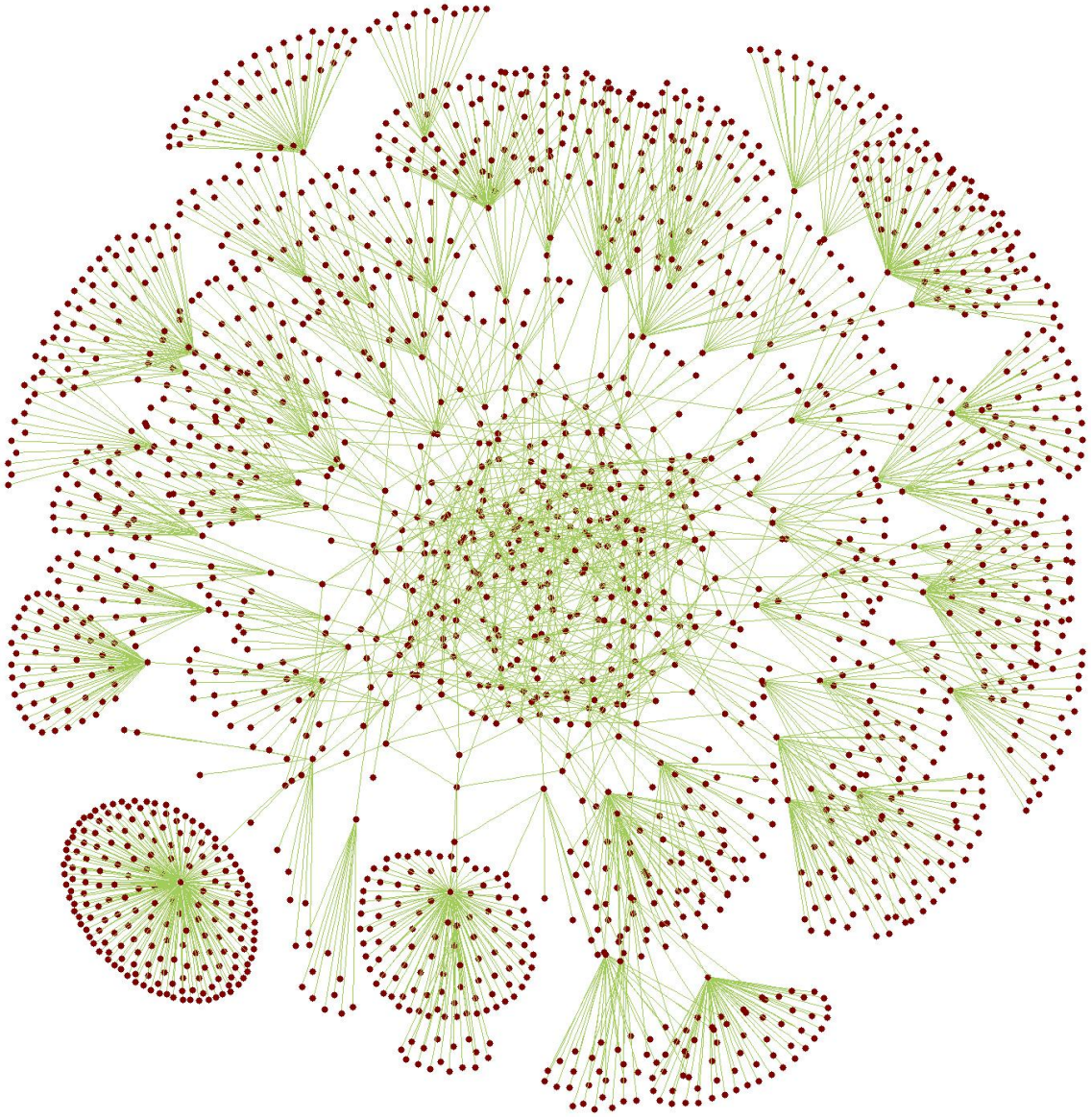

Figura 5.2: Esempio di topologia di rete

Definita la topologia, occorre caratterizzare i singoli nodi, definendovi dei *modelli di generazione del carico* (**workload**), il cui scopo è generare flussi IP in funzione di specifiche logiche protocollari. Tali modelli consistono in sequenze, più o meno complesse, di interazioni fra coppie di nodi, basate su comuni protocolli IP come **HTTP**, **DNS** o **DBMS**. (vedi figura 5.3) Lo scopo di un *modello di generazione del carico* è quello di **attivare** e **guidare** la **generazione di traffico IP** fra coppie di nodi nel sistema, permettendo la raccolta di tracciati da analizzare tramite DDA.

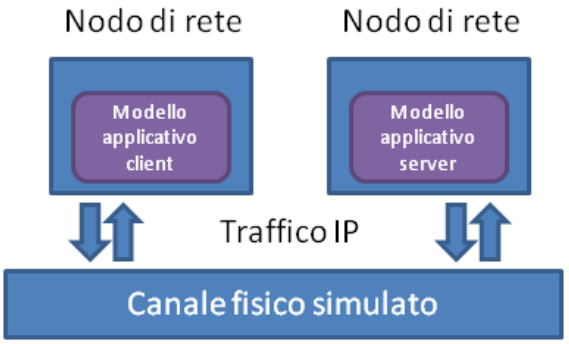

Figura 5.3: Generazione di traffico IP tramite modelli applicativi

5.1.1 Validazione dell'analisi

Un ambiente controllato, com'è quello simulativo, semplifica di molto le operazioni di validazione del tool di analisi. La validazione consiste nel confronto fra i *modelli di generazione del carico*, definiti sui nodi, e le rilevazioni fatte da DeDALO (figura 5.4), **contando** il numero di dipendenze rilevate correttamente. Una dipendenza sarà modellata come una sequenza più o meno sistematica fra due accessi a nodi server. I parametri su cui agire sono tutti di natura probabilistica, come:

- l'intertempo fra i due accessi
- la probabilità di eseguire o meno i singoli accessi
- il grado di casualità con cui scegliere i nodi server (d'ora in poi, 'servizi')

la dipendenza più forte si ha in condizioni di:

- intertempi definiti a priori
- certezza di effettuare l'accesso ad un servizio
- scelta dei servizi a compile-time

Ad esempio, si definisce un modello di carico client tale da collegarsi a un servizio A e a un servizio B, con un intertempo costante di 0.5 secondi. Eseguendo sistematicamente queste operazioni, per un numero molto alto di volte, si crea una situazione di *dipendenza* fra i due servizi, poichè le risposte di A saranno (con grande probabilità) necessarie per fruire dei servizi di B. Da un punto di vista formale, questa considerazione soddisfa le condizioni di **Sistematicità**, **Ripetitività** e **Immutabilità** definite nel capitolo 2. Per definire modelli di carico con bassi livelli di dipendenza, si può agire:

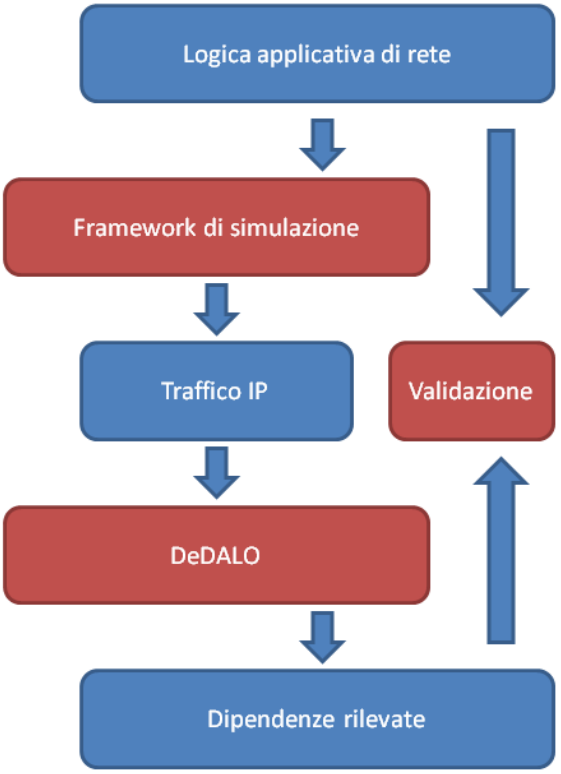

Figura 5.4: Interazione fra l'ambiente di simulazione e l'ambiente di analisi

- definendo intertempi generati da variabili aleatorie
- imponendo basse probabilità di effettuare i singoli accessi
- selezionando i servizi a cui accedere a run-time

5.2 Il Network simulator versione 3

Il Network Simulator versione 3 [Lac10], è un simulatore a eventi discreti per sistemi di rete, sviluppato nel 2009 da Mathieu Lacage, presso i laboratori dell'INRIA[1]. Malgrado il nome, la versione 3 costituisce qualcosa di più di una major release, infatti il simulatore è stato rivisto e riscritto rispetto alla più nota versione 2. E' stato introdotto un alto grado di realismo grazie a una totale reingegnerizzazione, che ne ha cambiato la struttura e introdotto nuove funzionalità:

- Core software estensibile
- Alto grado di realismo

[1] http://www.inria.fr/

- Integrazione e virtualizzazione
- Statistiche e tracing
- Gestione degli attributi

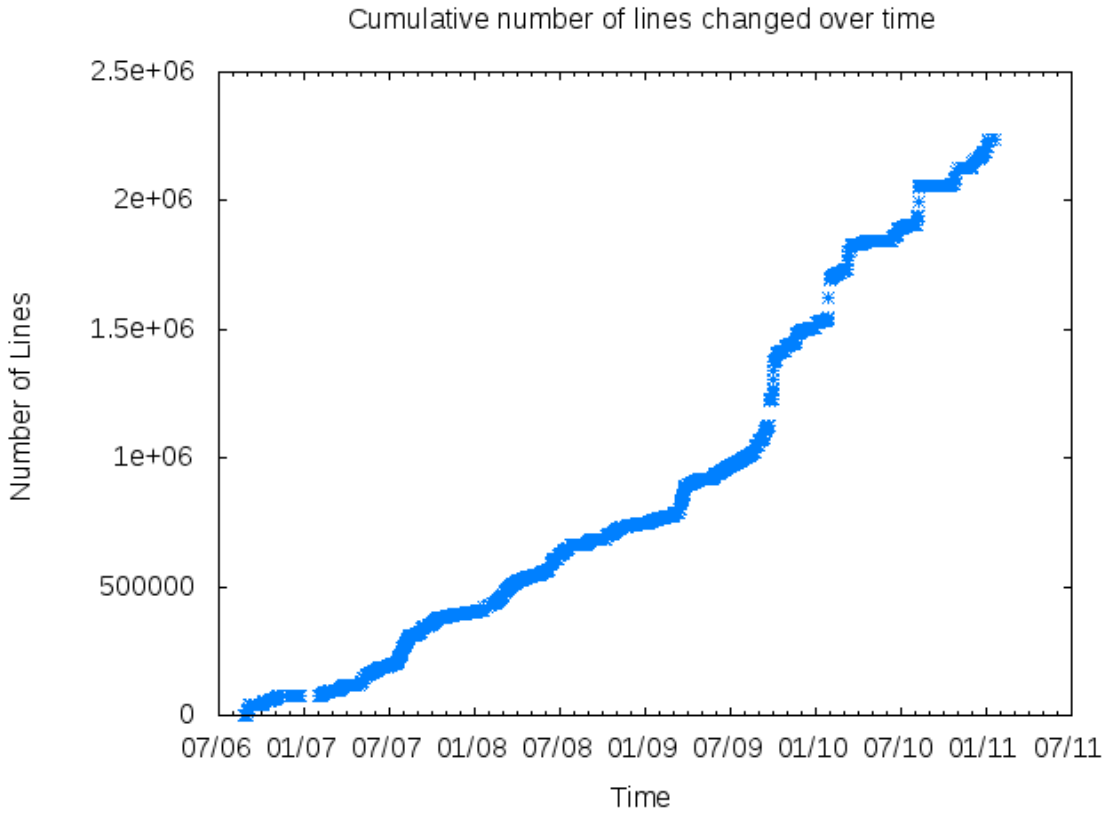

Figura 5.5: Linee di codice in NS3. Fonte: www.http://www.nsnam.org

L'ultima versione è la 3.10 del 5 gennaio 2011. La roadmap di sviluppo prevede rilasci con cadenza trimestrale e la popolarità del framework è in continua crescita, sia per il numero di download, che per il numero di linee di codice (LOC) scritte (figura 5.5).

NS3 è sviluppato in C++ con binding opzionali per Python, superando il dualismo C++ - OTcl che caratterizzava NS2. Il progetto viene gestito attraverso HG[2], un sistema pubblico di repository e versionamento: il tree ufficiale è mantenuto dall'INRIA, ma la comunità di utenti è incoraggiata a definire modelli e funzionalità da condividere con sviluppatori e altri utenti. NS3 supera alcune delle problematiche tipiche di NS2, come l'interoperabilità fra i modelli e la necessità di fare debugging

[2]http://mercurial.selenic.com/wiki/HgSubversion

su due linguaggi. L'utilizzo di Python, in questo caso, è puramente facoltativo e, al pari di Otcl, si propone come linguaggio di sviluppo 'più semplice' rispetto al C++.

NS3 prevede soluzioni che incoraggiano il riuso del software, permettendo una gestione più attenta dell'ereditarietà e della memoria. Gli oggetti sono aggregabili, in modo da creare entità specializzate definite a run time. E' posta particolare attenzione sul realismo delle simulazioni, modellando ogni nodo come un vero calcolatore composto da interfacce, stack IP e API applicative. Il supporto all'integrazione è garantito dall'utilizzo di codifiche e interfacce largamente diffuse, sia per la generazione di log e tracciati che per la struttura interna del simulatore. NS3 può operare con Wireshark, ad es., per visualizzare la sequenza di pacchetti generati da un run di simulazione.

La figura 5.6 schematizza la struttura di un modello di sistema distribuito in NS3. Il simulatore rappresenta il sistema come una composizione di oggetti **Node** (pc, server, router...) interconnessi da link di rete a cui accedono tramite oggetti di tipo **Interface**. Il compito di questi oggetti è inoltrare sul canale il carico generato dal nodo, guidato da oggetti di tipo **Application**.

5.2.1 Architettura

L'architettura software di NS3 (figura 5.7) è composta da nove moduli:

- Core: gestione degli attributi, delle callback, del tracing e dei generatori pseudo-casuali
- Common: modellazione di un pacchetto IP, gestione stream di output e funzionalità comuni
- Simulator: gestione degli eventi, gestione dello scheduler e del tempo simulato
- Node: definizione dei nodi, interfacce fisiche, stack di rete e socket
- Mobility: gestione della mobilità di un nodo per utilizzi in ambiti wireless
- Routing: gestione dello strato di rete in topologie simulate
- Internet Stack: gestione della (de) frammentazione dei dati, creazione di PDU
- Devices: gestione dei canali fisici di rete
- Helper: API di programmazione

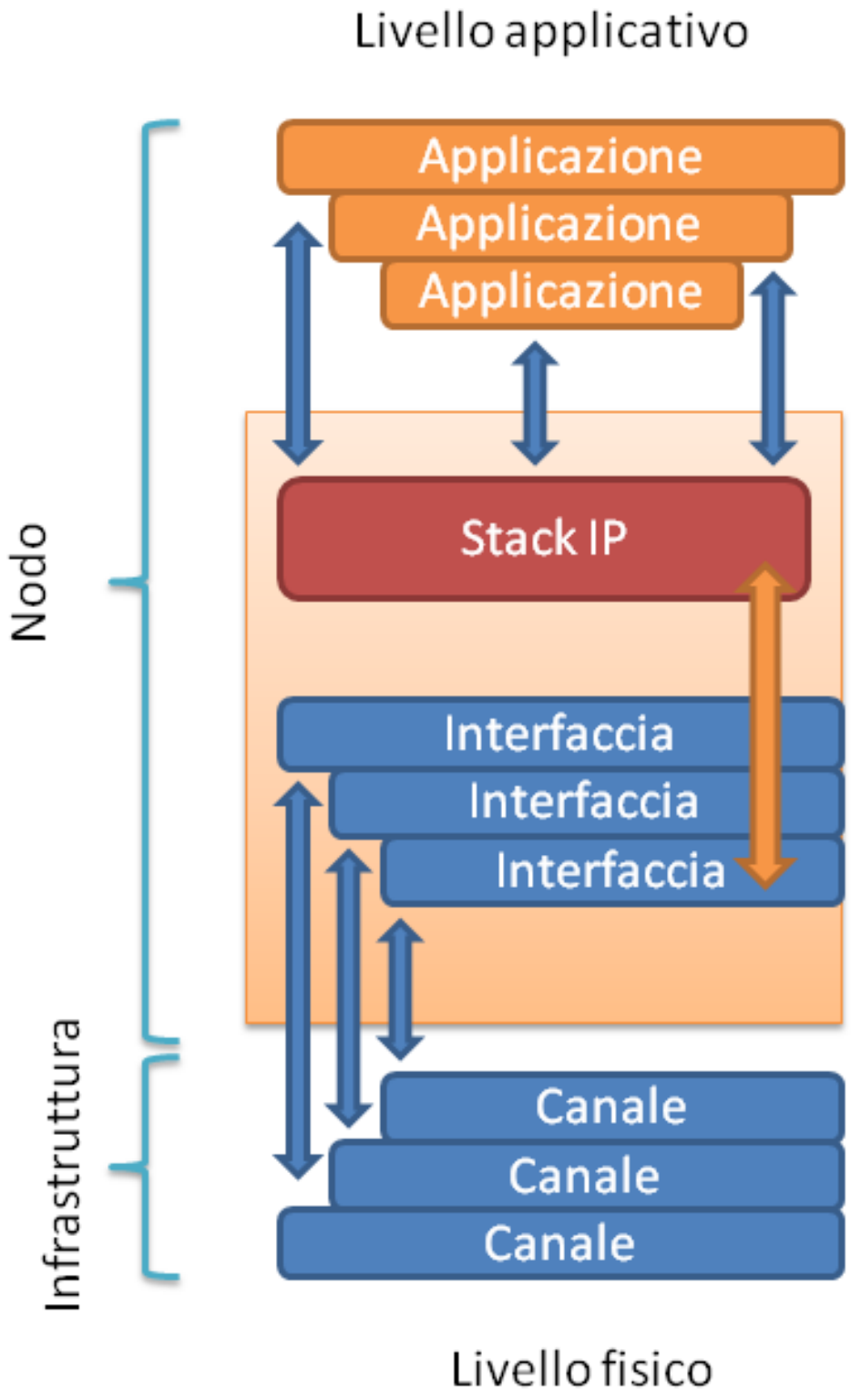

Figura 5.6: Modello di un sistema distribuito in NS3

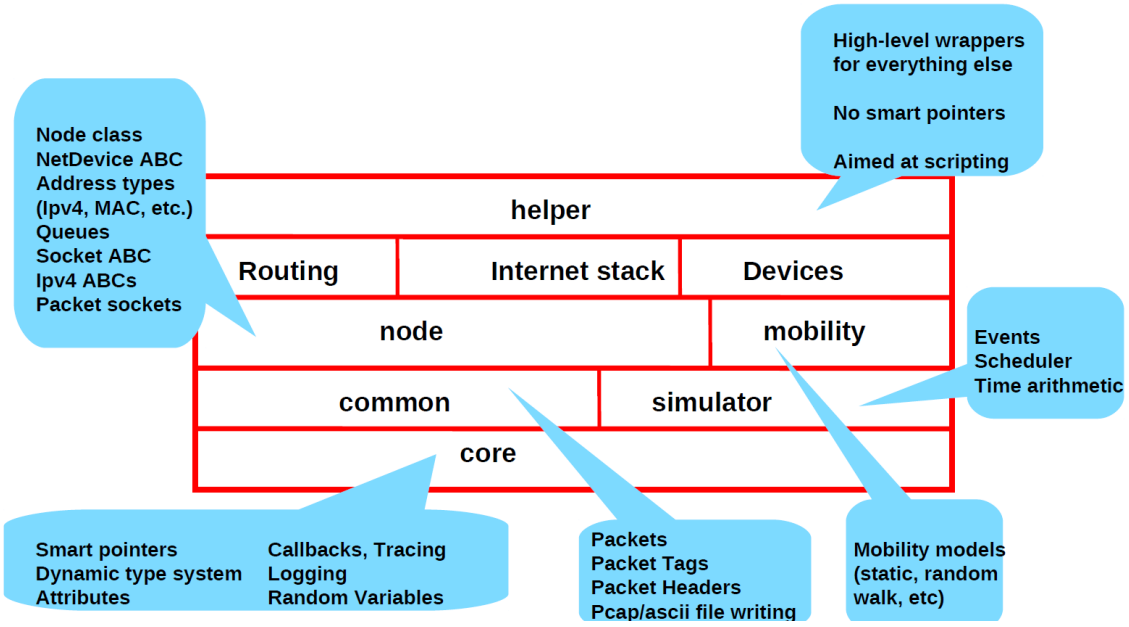

Figura 5.7: Struttura di NS3

Sviluppare in NS3, significa definire modelli simulativi di

- Canali di rete
- Protocolli di routing
- Applicazioni distribuite

NS3 nasce come sistema di simulazione e test per reti di trasporto, considerando il livello applicativo come un semplice 'carico' per testare la rete sottostante. I modelli più diffusi, infatti, riguardano i due livelli più bassi, permettendo di simulare le più comuni infrastrutture wireless e wired.

5.2.2 Modelli di generazione del carico

NS3 dispone di due rudimentali *modelli di generazione del carico* (figura 5.8):

- Generatore di traffico **on/off** su protocollo UDP
- Sistema **echo**, client/server, su protocollo UDP

La logica del modello **on/off** prevede una generazione continua e incontrollata di traffico, con:

- Un client che genera stream casuali su protocollo UDP

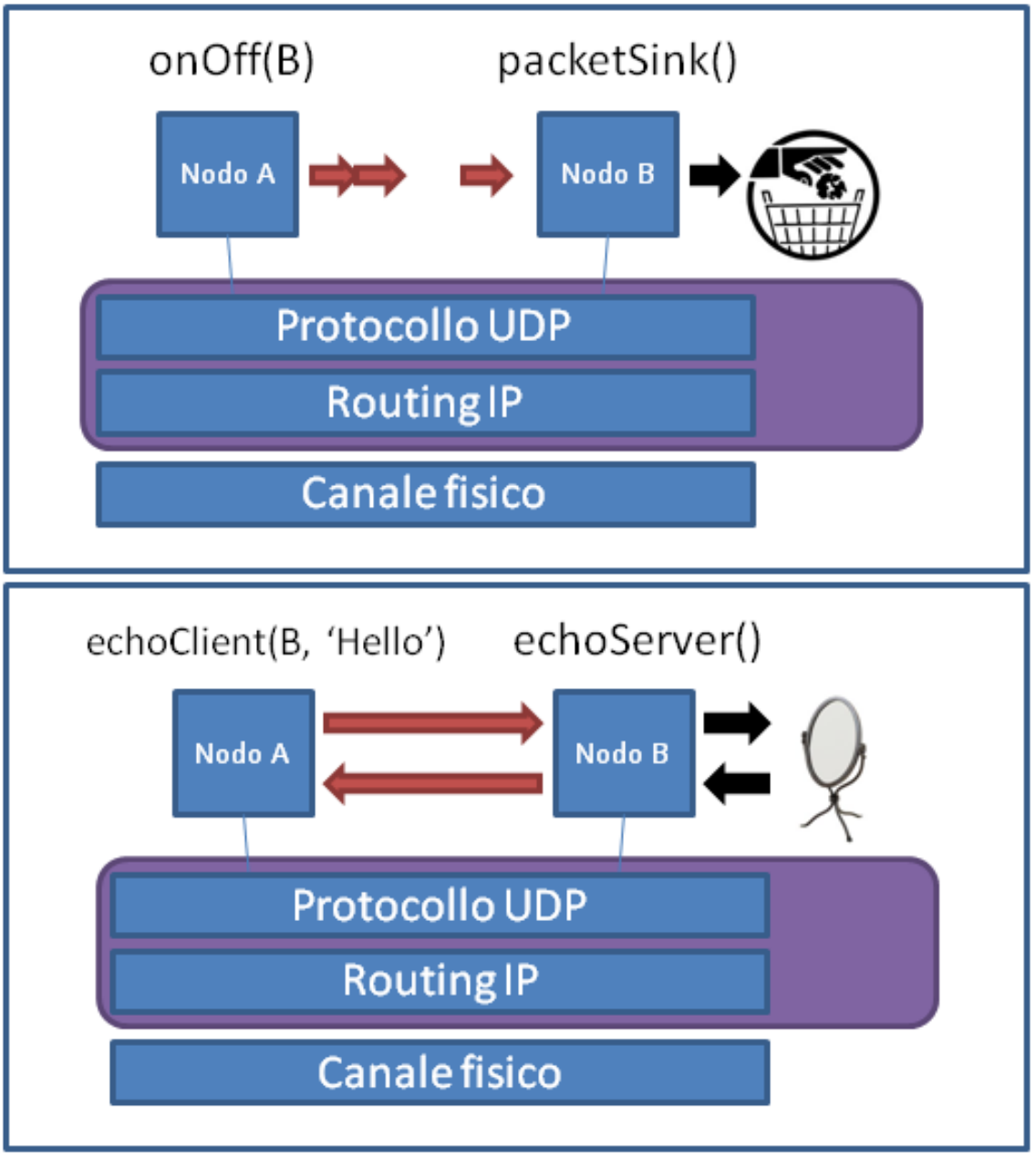

Figura 5.8: Modelli applicativi presenti in NS3

- Un server, definito *sink*, che riceve gli stream e li cancella senza nemmeno leggerli

Lo scopo di questo modello è generare un carico intermittente per valutare l'impatto sul routing e sui canali di rete. Il modello **echo** è leggermente più complesso, poichè sia client che server condividono un minimo **workflow**. I dati scambiati non rispettano alcuna **sintassi** nè **semantica** protocollare, ma entrambe le parti agiscono in maniera coordinata:

- Il client invia una stringa a un server
- Il server riceve la stringa e la rimanda al client, come un'eco (**echo**)

La semplicità di questi modelli li rende del tutto inutili ai fini di uno studio del traffico IP: nell'ambito di questa tesi sono stati sviluppati modelli di generazione del carico tali da rappresentare situazioni applicative più complesse.

5.2.3 Simulazione virtualizzata

Una delle novità di NS3 consiste nella possibilità di integrare le entità simulate all'interno di contesti reali, veicolando il traffico simulato su testbed o reti enterprise. E' possibile, ad esempio, validare modelli e protocolli, studiandone l'effetto sia su topologie simulate che su reti fisiche: il simulatore governa il traffico ma le operazioni link layer e di instradamento sono gestite in maniera ibrida.

Un vantaggio notevole consiste nel non dover modellare intere topologie, concentrando lo sviluppo sui quei nodi che dovranno testare il protocollo o i modelli. Sarebbe, di fatto, molto difficile riprodurre in maniera verosimile l'attività di migliaia di nodi, richiedendo un'elevata (e inutile) capacità di calcolo. E' possibile studiare sia l'evoluzione del traffico sintetico su reti reali che l'evoluzione del traffico reale su topologie simulate come mostrato in figura 5.9. NS3 permette questo at-

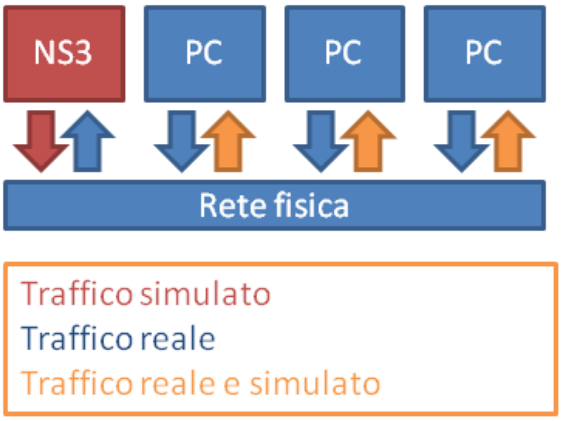

Figura 5.9: Traffico IP in reti virtualizzate

traverso l'utilizzo di macchine virtuali (QEMU) e pseudo-interfacce di rete (TAP), che agiscono localmente ma comunicano con l'esterno sfruttando interfacce fisiche reali (figura 5.10). Questa caratteristica, ancora in versione beta, è una delle più interessanti di NS3 e ne rappresenta uno dei punti di forza rispetto a prodotti simili.

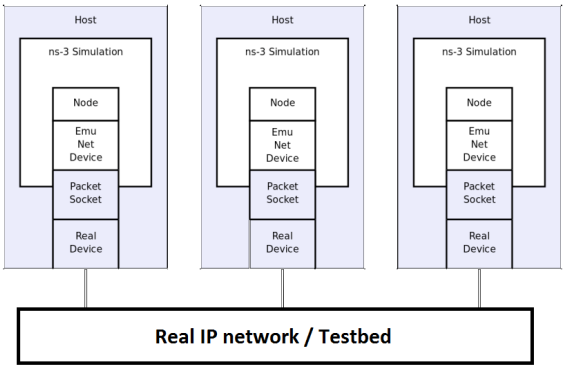

Figura 5.10: Integrazione di NS3 con reti fisiche reali

5.2.4 Performance e gestione delle risorse

La gestione degli oggetti in C++ è ancora piuttosto limitata se paragonata a JAVA o ad altri linguaggi Object Oriented. In particolare, la mancanza di un Garbage Collector, della classe Object, e di supporti sofisticati all'ereditarietà, rendono necessari alcuni accorgimenti per evitare fault applicative. NS3 dispone di un sistema di gestione della memoria che permette di superare questi problemi, sgravando l'utente da operazioni di controllo e pulizia del codice. Sono definiti dei puntatori

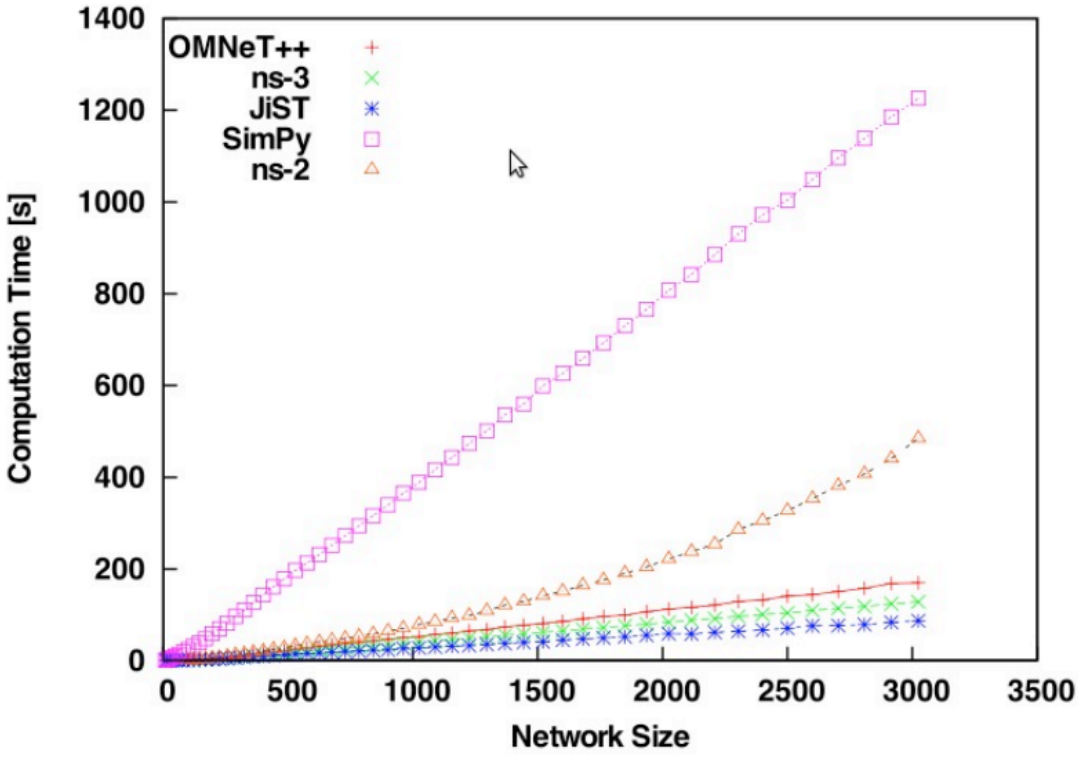

Figura 5.11: Tempo di calcolo in relazione ai nodi simulati. Fonte: A performance comparison of recent network simulators

detti 'smart' che permettono di tenere il conto delle referenze, in modo da attuare politiche di Garbage Collecting. Sono poi introdotte delle classi di generazione di oggetti, molto simili alle Factory Java, utilizzate ampiamente in NS3 per ottenere oggetti o gruppi di oggetti senza dover gestire definizioni e istanziazioni. Questi accorgimenti permettono ad NS3 di operare con alti livelli di performance e un'occupazione limitata di memoria, come riportato in [WW09], e mostrato in figura 5.12. La definizione di un proprio albero di ereditarietà permette a tutti gli oggetti NS3

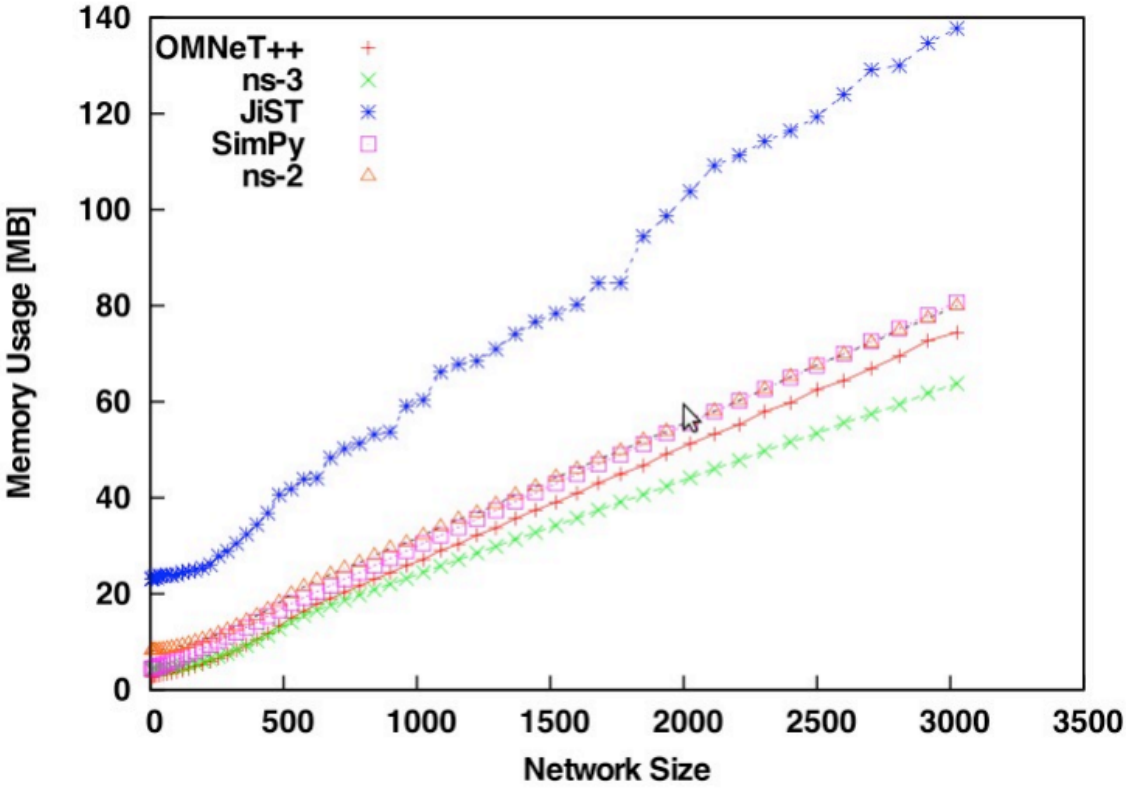

Figura 5.12: Memoria utilizzata in relazione ai nodi simulati. Fonte: A performance comparison of recent network simulators

di discendere da un unico tipo, dando modo al programmatore di creare metodi con parametri di tipo Object. Questo facilita il riuso e incoraggia la definizione di classi (modelli, ma non solo) di uso generale da estendere per utilizzi di dominio.

5.3 Modelli di protocolli IP

Nell'ambito di questa tesi, sono stati sviluppati dei modelli applicativi basati su tre comuni protocolli IP:

- Modello HTTP (WEB)

- Modello DNS
- Modello DBMS

ogni modello implementa il **workflow**, la **sintassi** e la **semantica** del rispettivo protocollo, definendo sia la sequenza di operazioni da svolgere, che le regole con cui comporre i messaggi scambiati. Ogni modello è composto da due entità distinte, **client** e **server**, che comunicano secondo il paradigma **richiesta - risposta** (figura 5.13). Il modello WEB, ad es., si compone di un oggetto **client** e un oggetto

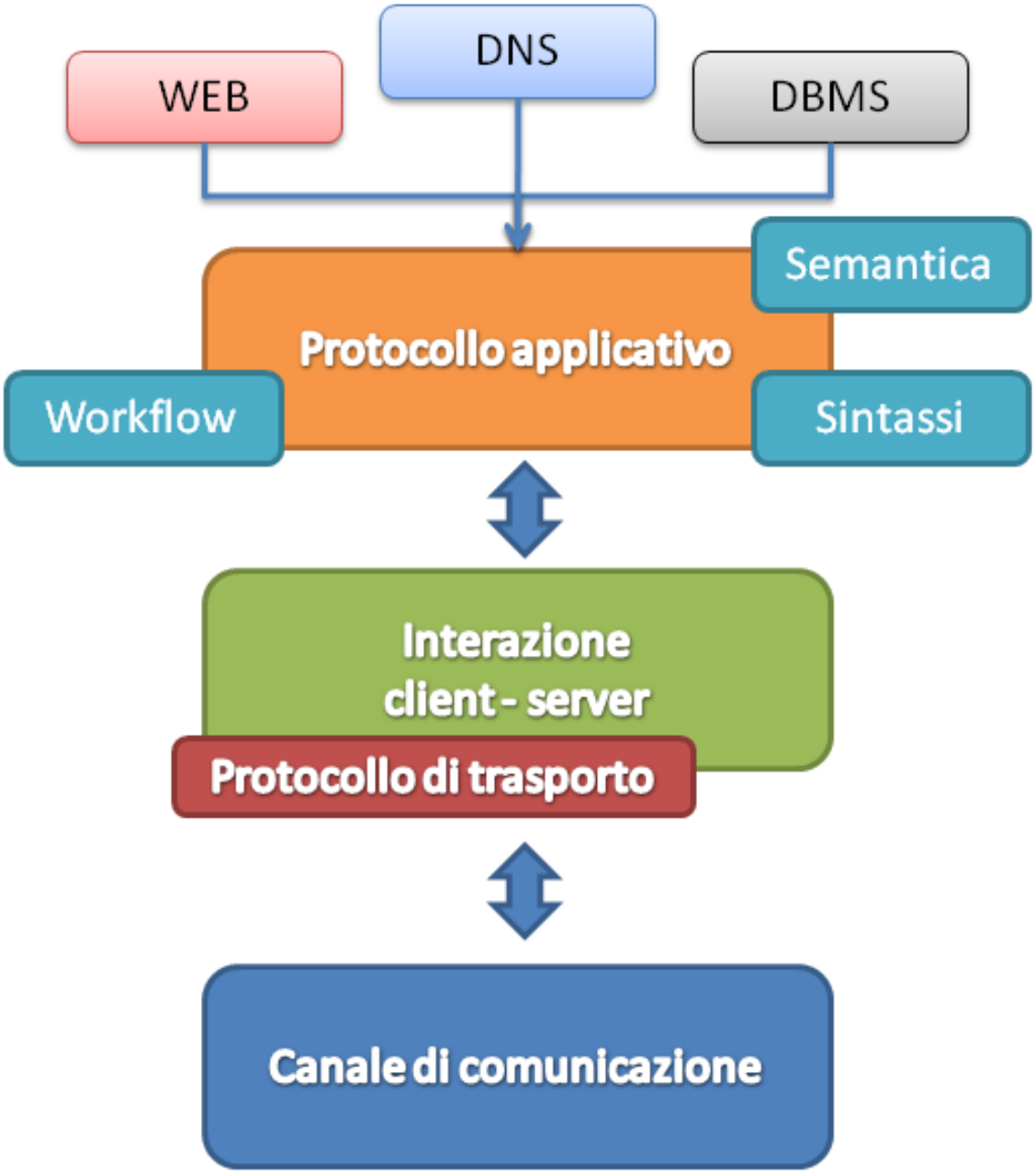

Figura 5.13: Struttura di un protocollo IP client - server

server che operano su nodi fisici (simulati), e interagiscono tramite flussi IP per trasferire pagine web.

Il modello compone la **richiesta** (lato client) e la **risposta** (lato server) secondo i vincoli imposti dal protocollo, e interagisce con i livelli sottostanti per inoltrare il messaggio sul canale. La componente **client**, in particolare, ha lo scopo di

- simulare un periodo di *think-time*
- effettuare la connessione a un nodo server
- generare la richiesta di una pagina WEB: **GET / HTTP/1.1**
- inviare la richiesta
- attendere la risposta
- ricevere la risposta

La componente **server** è più complessa, e si occupa di

- gestire le richieste di connessione dei client
- accogliere le richieste di pagine WEB
- simulare un tempo di servizio
- generare la risposta, es. **HTTP/1.1 200 OK**...
- inviare la risposta

I modelli **DNS** e **DBMS** operano in maniera analoga, differenziandosi nelle modalità di generazione della richiesta, della risposta, del *think-time* e del tempo di servizio. Lo schema di un generico modello **client/server** è rappresentato in figura 5.14. Sono definite due tipologie di operazioni, dette **operazioni client-side** e **operazioni server-side**, che le due componenti eseguono prima di comporre e inviare il messaggio. Lo scopo di queste due fasi è duplice:

- introdurre ritardi più o meno variabili
- far eseguire operazioni sistematiche al modello (es. accesso al filesystem)

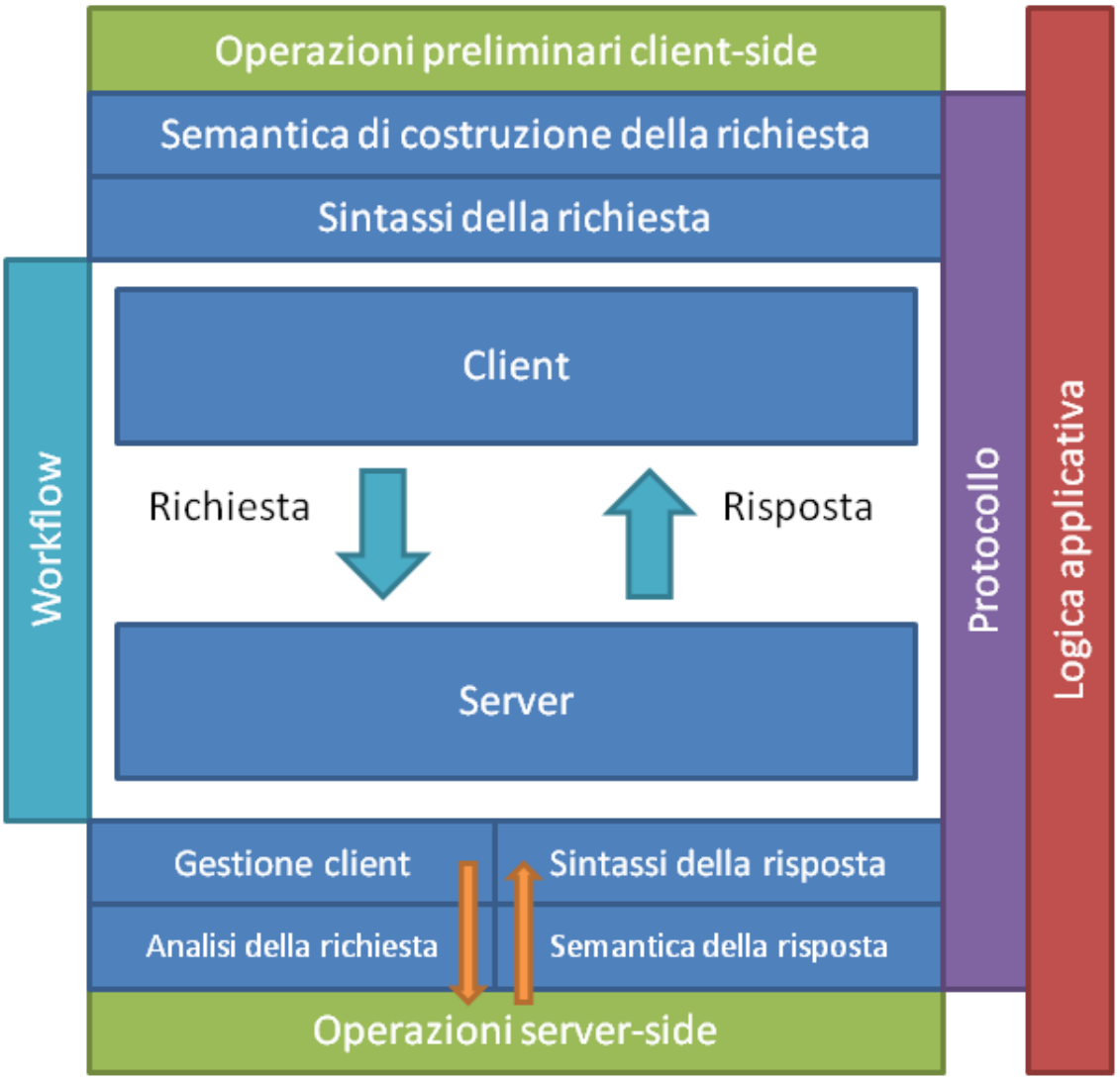

Figura 5.14: Struttura di un modello applicativo e protocollare

Per fare un esempio, un modello client WEB può essere impostato in modo da attendere 0.3 secondi prima di richiedere una pagina al server, oppure, un modello server WEB può essere impostato per leggere un file da disco prima di inviare la risposta al client. In queste fasi si possono definire dei semplici cicli di attesa o sequenze di task da far eseguire al modello.

Un modello viene **istanziato** definendo la coppia di nodi, **C** ed **S**, che agiranno da **client** e **server**, e definendo alcuni parametri caratteristici:

- operazioni **client-side** o valore del *think-time*
- operazioni **server-side** o valore del *tempo di servizio*

Uno stesso modello può essere istanziato più volte su uno stesso nodo, permettendo ad es., a un client **C** di attivare richieste sia verso un server **S1** che verso server **S2**.

Ogni istanza di un modello **client** è configurata per eseguire ciclicamente una stessa richiesta, ed è possibile parametrizzare il modello in modo da definire:

- il numero N di iterazioni, $[1, \infty)$ ovvero, finchè non termina la simulazione
- il tempo T di attesa fra un'iterazione e la successiva

la definizione di un tempo può avvenire impostando sia un valore **costante** (es. 0.4 secondi), sia una **variabile aleatoria** da cui estrarlo (es. *Normale* con media 0.3 e varianza 0.1). Per fare un esempio, un modello di **client WEB**, istanziato su un nodo **C**, viene configurato per eseguire **1000** iterazioni, separate da un tempo di attesa definito con probabilità *Uniforme* fra 0.3 secondi e 0.6 secondi.

Un modello **client** accede a uno specifico nodo server definito a compile-time, ma è possibile configurare il modello in modo da definirlo anche a run-time, indicando un set di nodi server fra cui scegliere. Ad es., il modello **client WEB** può essere configurato in modo da accedere ai server **S1** o **S2**, entrambi con probabilità **0.5**. Il simulatore sceglierà il servizio ad ogni esecuzione, guidando il modello client verso il server S1 o il server S2.

Due modelli possono cooperare per creare strutture **multi-tier** (es. WEB-DBMS in figura 5.15), tramite l'interazione di modelli client e server istanziati su uno stesso nodo. Nel caso in figura 5.15, si considera un modello WEB fra i nodi A e B e un modello DBMS fra i nodi B e C, definendo sul nodo B un **collegamento** fra l'istanza del server WEB e l'istanza del client DBMS. Tale **collegamento** potrà concretizzarsi, ad es., definendo operazioni **server-side** sul modello server WEB di B, tali da interrogare il server DBMS all'arrivo di ogni richiesta WEB da A.

5.3.1 Utilizzo dei modelli applicativi per definire un modello di carico

I tre modelli IP possono essere utilizzati per implementare un *modello di generazione del carico* attraverso quattro fasi:

- identificazione dei nodi client **NC**
- identificazione dei nodi server **NS**

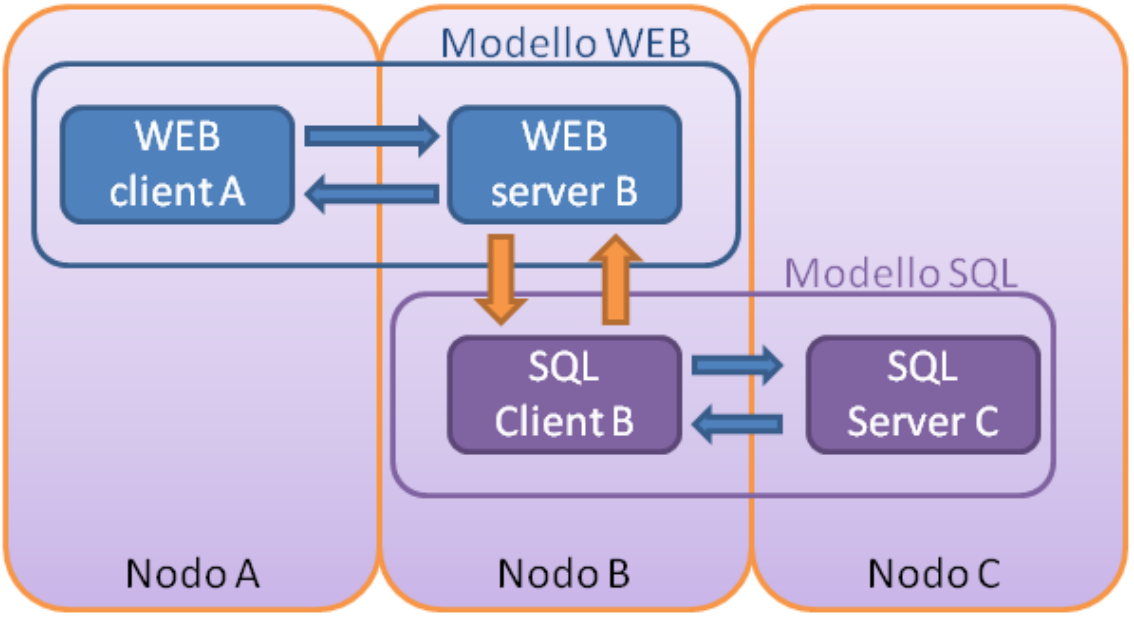

Figura 5.15: Due modelli in una struttura multi-tier

- **formalizzazione** del *workflow applicativo*
- selezione dei *modelli server*
- istanziazione dei modelli client e server sui nodi

Formalizzare un *workflow applicativo* significa definire un **flusso logico** composto da sequenzi di accessi a nodi server. Ad es., si può definire un *workflow* sul nodo **C**, composto da un accesso al server **S1** e da un accesso al server **S2**. Per **accesso**, si intende un'interazione client/server guidata da uno specifico modello applicativo.

Un nodo client può essere dotato di un *workflow applicativo* che utilizza sequenzialmente più modelli o più istanze di uno stesso modello, per generare flussi IP. Si può replicare, ad es., il workflow di un browser WEB, installato sul nodo **C**, istanziando un modello client DNS, un modello client WEB e legando i due da una relazione di **sequenzialità**. Nell'algoritmo 7, si definisce un ciclo di esecuzione che invia una richiesta DNS, attende 0.1 secondi e poi invia una richiesta WEB. Il ciclo viene eseguito finchè la simulazione non viene interrotta ed interpone un tempo estratto da una variabile Normale (4, 0.3) fra un'iterazione e la successiva.

Il risultato è una sequenza di coppie di flussi DNS (**C-S1**) e HTTP (**C-S2**) con intertempi costanti di **0.1** secondi.

Affinchè tutto funzioni, ovviamente, occorre che i modelli server, **DNS** e **WEB**, siano definiti e attivati sui nodi **S1** ed **S2**. La definizione del *workflow* può essere

Algoritmo 7 Definizione di un workflow applicativo

```
clientWEB ← istanziaModelloClient(WEB, S1)
clientDNS ← istanziaModelloClient(DNS, S2)
impostaIterazioni(clientWEB, 1)
impostaIterazioni(clientDNS, 1)
installaModello(C, clientWEB)
installaModello(C, clientDNS)
iterazione ← 1
iterazioni ← ∞
while iterazione ≤ iterazioni do
  attivaModello(C, clientDNS)
  attendi(0.1 sec)
  attivaModello(C, clientWEB)
  attendi(Normale, 4 sec, 0.3)
  iterazione ← iterazione + 1
end while
```

articolata in modo da definire sequenze di accessi su base probabilistica ed eseguire path applicativi definiti a run-time. Per fare un esempio, si può modellare l'effetto di una cache DNS definendo la probabilità di *cache-hit* e decidendo se accedere o meno al DNS. Nell'algoritmo 8 si simula il comportamento di un client WEB, che usa una cache DNS con probabilità di *cache-hit* pari a 0.2, ovverosia, in 8 casi su 10, una richiesta WEB sarà preceduta da una richiesta DNS.

Algoritmo 8 Accesso probabilistico ai servizi

```
cacheHit ← 0.2
while iterazione ≤ iterazioni do
  P ← estraiValore(0, 1)
  if P ≥ cacheHit then
    attivaModello(C, clientDNS)
    attendi(0.1 sec)
  end if
  attivaModello(C, clientWEB)
  attendi(Normale, 4 sec, 0.3)
  interazione ← interazione + 1
end while
```

Anche i parametri dei modelli, e i modelli stessi, possono essere selezionati su base probabilistica. Proseguendo con l'esempio di un browser WEB, è possibile simulare l'accesso a nodo server selezionato all'interno di un set di nodi in mutua esclusione. L'algoritmo 9 simula una sequenza di accessi WEB, con *cache-hit* pari a 0.2, estraendo il nodo server dal set *(S1, S2, S3)*, con probabilità di *(0.2, 0.4, 0.4)*.

Algoritmo 9 Selezione probabilistica dei servizi

```
clientWEB ← istanziaModelloClient(WEB)
aggiungiNodoServer(clientWEB, S1, 0.2)
aggiungiNodoServer(clientWEB, S2, 0.4)
aggiungiNodoServer(clientWEB, S3, 0.4)
clientDNS ← istanziaModelloClient(DNS, S2)
impostaIterazioni(clientWEB, 1)
impostaIterazioni(clientDNS, 1)
installaModello(C, clientWEB)
installaModello(C, clientDNS)
iterazione ← 1
iterazioni ← ∞
cacheHit ← 0.2
while iterazione ≤ iterazioni do
  P ← estraiValore(0, 1)
  if P ≥ cacheHit then
    attivaModello(C, clientDNS)
    attendi(0.1 sec)
  end if
  estraiNodoServer(clientWEB)
  attivaModello(clientWEB)
  attendi(Normale, 4 sec, 0.3)
  interazione ← interazione + 1
end while
```

L'effetto di questo *workflow* consisterà in sequenze di tre diversi flussi HTTP (C-S1, C-S2 e C-S3), preceduti nell'80% dei casi da flussi DNS (C-DNS) con intertempo pari a 0.1 secondi.

5.3.2 Esempi di workload con dipendenze

Ai fini dell'esecuzione di DeDALO, occorre definire dei *modelli di generazione del carico* (o **workload**) tali da esibire livelli variabili di **dipendenze**. Un esempio di **workload** con dipendenze è costituito da una sequenza, ripetuta molte volte, di accessi a servizi IP definiti a compile-time, con intertempo costante (algoritmo 10). Un esempio di **workload** privo di dipendenze è costituito da sequenze di accessi

Algoritmo 10 Workload con dipendenze

```
iterazione ← 1
iterazioni ← ∞
while iterazione ≤ iterazioni do
  attivaModello(DNS)
  attendi(0.1 sec)
  attivaModello(WEB)
  attendi(Normale, 4 sec, 0.3)
  iterazione ← iterazione + 1
end while
```

a servizi IP, determinati a run-time da un **set molto ampio di candidati** e con la possibilità di eseguire o meno i singoli accessi. Le possibili sequenze di flussi, generate dall'algoritmo 11 ad ogni iterazione, sono più di **10000**, con intertempi che variano uniformemente fra 0 e 2 secondi. Sarà difficile replicare più volte una stessa sequenza di flussi (**C-S1**, **C-S2**) con **stesso intertempo**. Considerando questi esempi, è possibile definire vari livelli di dipendenza in funzione delle **possibili sequenze di flussi** generate dal *modello di generazione del carico*.

5.4 Interfaccia di generazione della topologia

Il frontend è implementato come ambiente grafico per la definizione della topologia. Il sistema è sviluppato in JAVA/SWING e sfrutta la libreria JUNG per rappresentare e modellare dei grafi. Scopo di questo sistema è fornire una semplice interfaccia utente per dislocare nodi, collegamenti e applicazioni su cui eseguire simulazioni con NS3. Si basa su delle API High-Level scritte in JAVA, concepite per modellare un sistema distribuito, riflettendo ed estendendo la logica di API di NS3.

Algoritmo 11 Workload senza dipendenze

```
modelloClientWEB1 ← istanziaModelloClient(WEB)
aggiungiNodoServer(WEB1, S1, 0.01)
aggiungiNodoServer(WEB1, S2, 0.01)
...
aggiungiNodoServer(WEB1, S100, 0.01)
modelloClientWEB2 ← istanziaModelloClient(WEB)
aggiungiNodoServer(WEB2, S1, 0.01)
aggiungiNodoServer(WEB2, S2, 0.01)
...
aggiungiNodoServer(WEB2, S100, 0.01)
impostaIterazioni(modelloClientWEB1, 1)
impostaIterazioni(modelloClientWEB2, 1)
installaModello(C, modelloClientWEB1)
installaModello(C, modelloClientWEB2)
iterazione ← 1
iterazioni ← ∞
probW1 ← 0.3
probW2 ← 0.6
while iterazione ≤ iterazioni do
  P1 ← estraiValore(0, 1)
  if P1 ≥ probW1 then
    estraiNodoServer(WEB1)
    attivaModello(WEB1)
    attendi(Uniforme, 0 sec, 2 sec)
  end if
  P2 ← estraiValore(0, 1)
  if P2 ≥ probW2 then
    estraiNodoServer(WEB2)
    attivaModello(WEB2)
    attendi(Uniforme, 0 sec, 2 sec)
  end if
  attendi(Normale, 4 sec, 0.3)
  interazione ← interazione + 1
end while
```

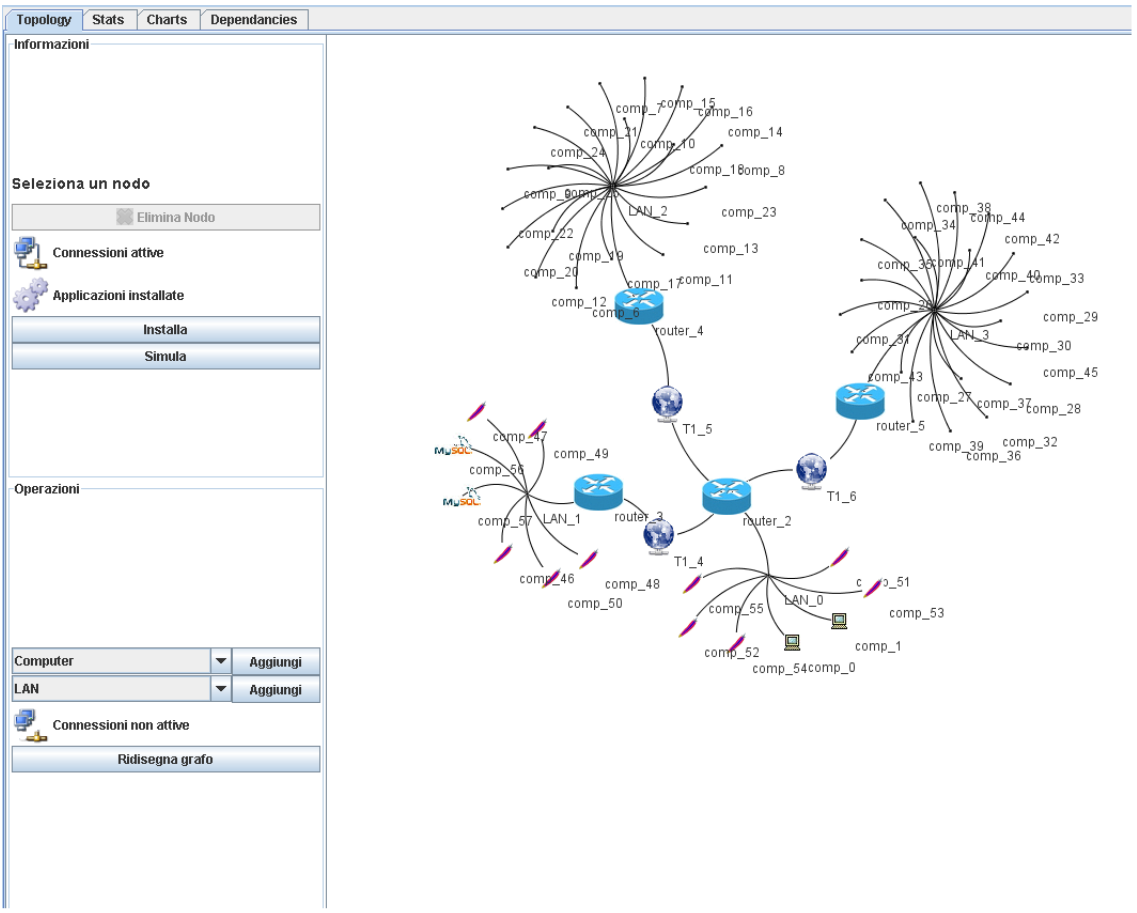

Figura 5.16: Ambiente di generazione della topologia

E' possibile gestire oggetti 'Nodi', 'Link' e 'Applicazioni' (lato JAVA) utilizzando uno stesso name-space fra GUI e ambiente di simulazione. L'interfaccia è composta da un pannello SWING suddiviso in due aree. A sinistra si trovano i pulsanti per aggiungere, eliminare o interconnettere elementi. A destra si trova il piano di lavoro, su cui viene mostrata la topologia man mano che evolve. Il tool permette di esportare la topologia utilizzando un formato di codifica che ne cattura ogni aspetto, dal livello fisico a quello applicativo, ed è in grado di rilevarne lo stato mentre il simulatore vi opera.

Capitolo 6

Risultati Sperimentali

Sono mostrati, in questo capitolo, i risultati dell'esecuzione di DeDALO su scenari applicativi simulati. Si utilizzerà una stessa **topologia** di medie dimensioni, per simulare il comportamento di applicazioni distribuite di diversa complessità. Ogni applicazione sarà descritta definendo la logica applicativa e i servizi utilizzati.

Per ogni simulazione saranno definiti due *modelli di generazione del carico*: uno composto da sequenze predeterminate di accessi a servizi (**carico con dipendenze**) ed uno composto da sequenze casuali (**carico senza dipendenze**), il cui scopo è introdurre 'rumore di fondo'.

Si mostrerà l'efficacia di DeDALO nel saper individuare e valorizzare le **dipendenze** fra i vari servizi IP utilizzati nei tre scenari.

6.1 Ambiente di sperimentazione

Sia la simulazione che DeDALO, sono eseguiti su una macchina HP monoprocessore (8-core) con 6 GB di RAM. I due ambienti utilizzano aree comuni di memorizzazione, in modo da stabilire delle relazioni produttore - consumatore attraverso i tracciati generati a run-time.

6.2 Caso di studio

Gli esperimenti sono condotti su una **topologia** composta da:

- **6** *Autonomous System* su rete **LAN Gigabit**

- **6** link **T1** da **1 Mbps**

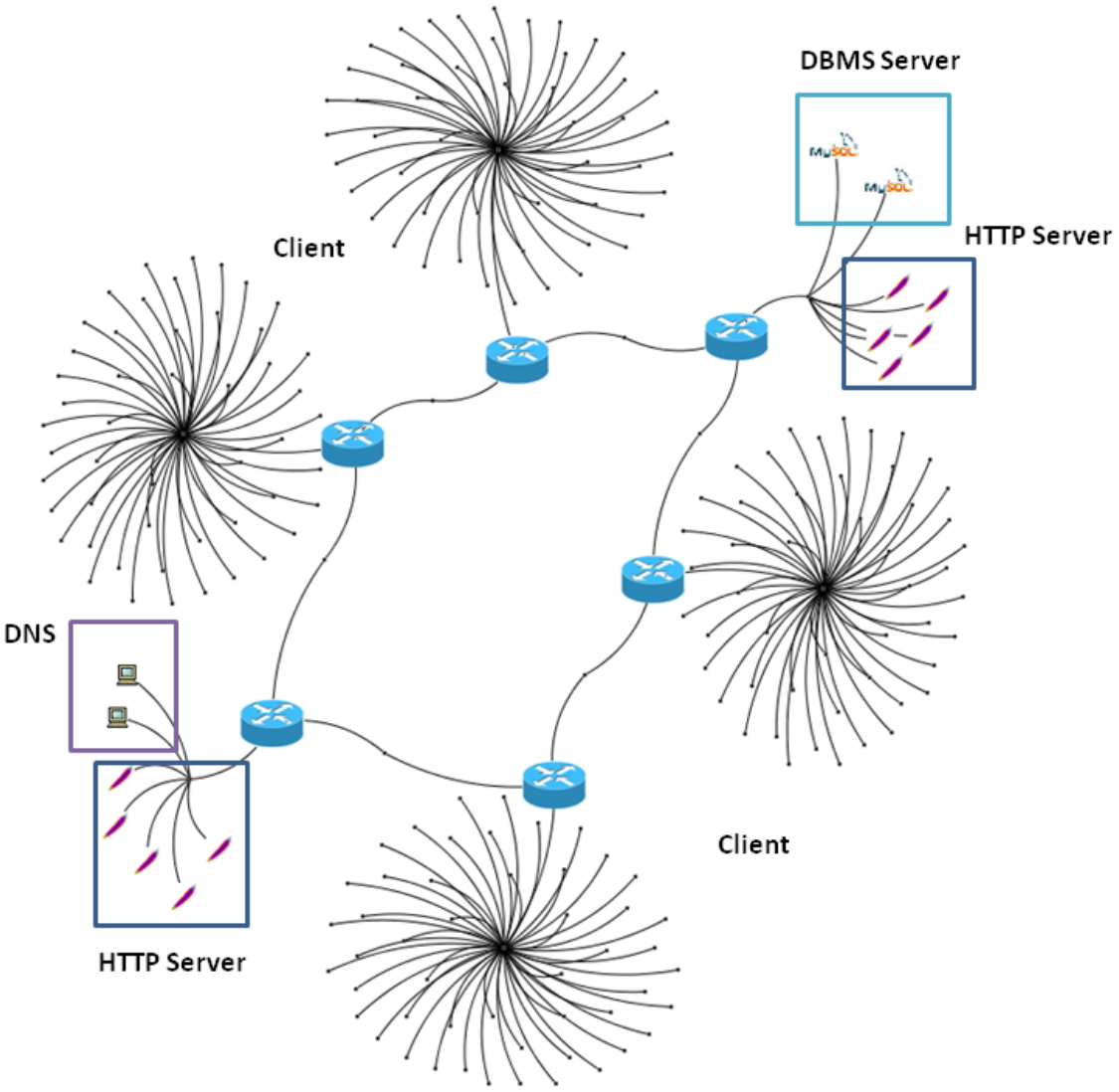

Figura 6.1: Topologia e suddivisione in Autonomous System

Ogni *Autonomous System* è dotato di un proprio **router** che lo collega ad altri due *Autonomous System*. I nodi **client** sono disposti su quattro *Autonomous System*, ognuno con un proprio indirizzo di rete:

1. **192.168.2.0/24** (**50** nodi)
2. **192.168.3.0/24** (**50** nodi)
3. **192.168.4.0/24** (**50** nodi)
4. **192.168.5.0/24** (**50** nodi)

I nodi **server** sono disposti su due *Autonomous System*:

1. **192.168.0.0/24** (**2 DNS server**, **5 WEB server**)
2. **192.168.1.0/24** (**2 DBMS server**, **5 WEB server**)

L'indirizzo IP di ogni nodo viene assegnato a run-time dal simulatore, partendo dall'indirizzo di rete del rispettivo *Autonomous System*.

6.3 Modelli di generazione del carico

Le simulazioni sono condotte dividendo i nodi client in due gruppi, (client di **applicazioni distribuite** e client *desktop*) così dislocati:

- **25** client *desktop* sull'*Autonomous system* 192.168.3.0/24
- **25** client *desktop* sull'*Autonomous system* 192.168.4.0/24
- **25** client di **applicazioni distribuite** sull'*Autonomous system* 192.168.3.0/24
- **25** client di **applicazioni distribuite** sull'*Autonomous system* 192.168.3.0/24
- **50** client *desktop* sull'*Autonomous system* 192.168.5.0/24
- **50** client di **applicazioni distribuite** sull'*Autonomous system* 192.168.6.0/24

in modo da dedicare un intero *Autonomous system* alla generazione di traffico applicativo e due *Autonomous system* alla generazione di carico misto (casuale e applicativo).

Per ogni gruppo sarà definito un diverso *modello di generazione del carico*: uno generato a *run-time*, definito sui nodi *desktop* e uno generato a *compile-time*, definito sui client di **applicazioni distribuite**.

Lo scopo del primo modello è creare 'rumore di fondo', generando traffico IP composto da sequenze casuali di accessi a servizi. Lo scopo del secondo modello, invece, è generare delle sequenze di accessi a servizi che abbiano un grado più o meno forte di correlazione, esibendo, cioè, delle **dipendenze**. Tale modello sarà definito, nel seguito, come **modello sistematico di generazione del carico**, e sarà descritto per ogni esperimento definendo:

- i servizi coinvolti (modelli IP utilizzati)

- il flusso logico (sequenza di accessi a nodi server)
- i parametri caratteristici (tempi di interarrivo e probabilità)
- l'effetto del modello (sequenze di flussi IP generati)
- i servizi in relazione di dipendenza

Due servizi IP, **S1** ed **S2** saranno in relazione di **dipendenza** *forte* se il modello li interroga centinaia di volte in **sequenza**, con intertempi che abbiamo uno scarto dell'ordine del **centesimo di secondo**. La relazione sarà *debole* se lo scarto fra gli intertempi è dell'ordine del **decimo secondo** e se i due flussi occorrono sequenzialmente per poche volte.

L'operato di DeDALO sarà valutato sia in funzione del numero di dipendenze rilevate, considerando i falsi positivi e negativi, che in funzione del grado di dipendenza. DeDALO identifica una dipendenza *forte* nel caso di un grado superiore al 45%, altrimenti la considera *debole*.

6.4 Convenzioni adottate

Gli algoritmi descritti nelle sezioni che seguono adottano alcune **primitive** per rappresentare gruppi di operazioni svolte dai modelli. Le primitive utilizzate nell'ambito di un modello **client** sono:

- *istanza del modello* **istanziaModelloClient**(*nome modello*, *nodo server*) : crea un'istanza del modello client impostato per connettersi al dato nodo server
- **aggiungiNodoServer**(*istanza del modello*, *nodo server*, *probabilità*) : aggiunge un nodo server al modello client indicando un valore probabilistico di scelta
- **estraiNodoServer**(*istanza del modello*) : estrae un nodo server fra quelli aggiunti al modello
- **inviaRichiesta**(*nodo client*, *istanza del modello*) : attiva il modello client inviando una richiesta al nodo server
- **indirizzoInCache**(*probabilità di cache hit*) : valuta la possibilità di risolvere un indirizzo IP accedendo alla cache DNS
- **erroreDNS**(*probabilità di failover*) : probabilità di failure sul DNS primario

- *istanza del modello* **selezionaModelloClient**(*nodo client*) : seleziona casualmente un modello fra quelli installati su un nodo
- *risposta* **riceviRisposta**(*nodo client*, *istanza del modello*) : ricezione di una risposta dal server

Le primitive utilizzate nell'ambito di un modello server sono:

- *istanza del modello* **istanziaModelloServer**(*nome modello*, *porta*) : crea un'istanza del modello server impostato per lavorare sulla data porta
- *nodo client* **accettaConnessione**() : connessione di un nodo client
- *richiesta* **riceviRichiesta**(*nodo client*, *istanza del modello*) : riceve la richiesta da un nodo client
- *risposta* **componiRisposta**(*istanza del modello*) : compone la risposta in base al modello indicato
- **inviaRisposta**(*nodo client*, *istanza del modello*, *risposta*) : invia la risposta al nodo client usando il modello selezionato

Le primitive comuni sono:

- **simulazioneInCorso**() : controlla che la simulazione sia in corso
- **installaModello**(*nodo*, *istanza del modello*) : installa un modello su un nodo
- **attendi**(*tempo*) : pone il nodo in attesa di un tempo costante
- **attendi**(*variabile aleatoria*, *media*, *varianza*) : pone il nodo in attesa di un tempo variabile

6.5 Primo esperimento

In questo primo esperimento è simulata l'attività di una semplice **applicazione distribuita**, composta da **cinque servizi WEB** offerti da cinque diversi nodi server.

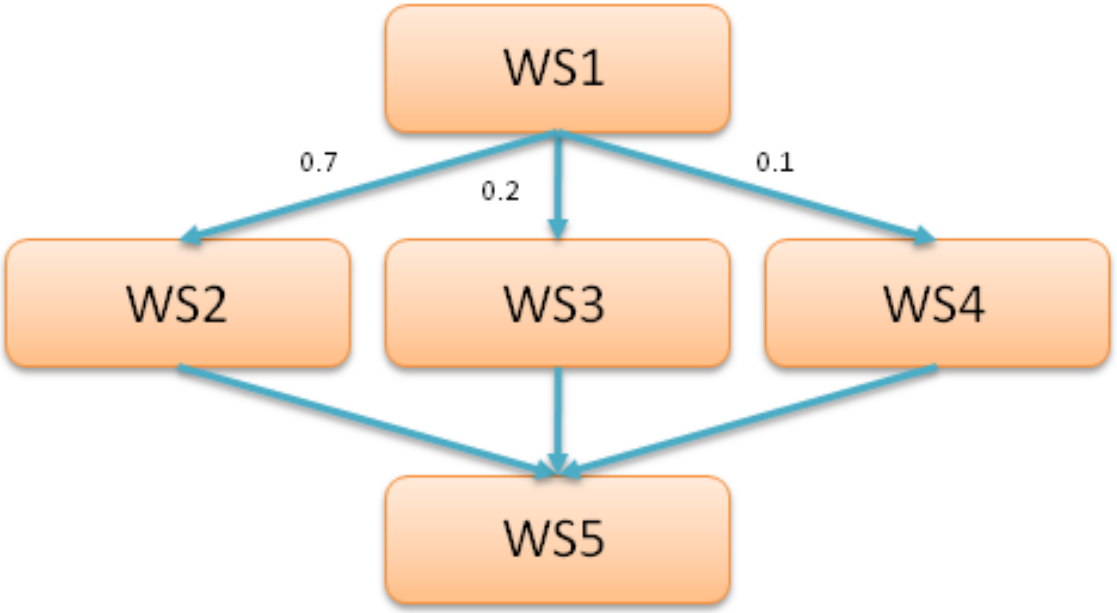

Figura 6.2: Modello di generazione del carico (Primo esperimento)

6.5.1 Contesto rappresentato

Il *modello sistematico di generazione del carico*, (mostrato in figura 6.2 si compone di una sequenza di **tre accessi a servizi WEB**, di cui il primo è selezionato a *compile-time* (**WS1**), il secondo a *run-time*, scelto fra un set di tre:

- **WS2** con probabilità *0.7*
- **WS3** con probabilità *0.2*
- **WS4** con probabilità *0.1*

e l'ultimo selezionato a *compile-time* (**WS5**). Tutti i flussi **HTTP** sono preceduti da flussi **DNS**, con intertempi **costanti** di **0.1** secondi. L'intertempo fra i primi due accessi (**WS1** $\rightarrow$ **WS2..WS4**) è determinato da una variabile aleatoria **Uniforme** fra **0.3** secondi e **0.4** secondi.

L'intertempo fra il secondo e il terzo accesso (**WS2..WS4** $\rightarrow$ **WS5**) è di **1** secondo. Ogni iterazione è separata dalla successiva da **3** secondi, per evitare correlazioni fra flussi di diverse iterazioni. La figura 6.3 mostra un possibile percorso di esecuzione attivato dal client **C**. Il modello risultante è illustrato nell'algoritmo 12. I servizi **WS1..WS5** sono installati sui nodi server dell'*Autonomous system* con indirizzo **192.168.1.0/24**. I due **DNS** hanno indirizzo **192.168.0.6** e **192.168.0.7**.

Le **dipendenze** introdotte sono:

Algoritmo 12 Primo modello di generazione del carico

```
clientWS1 ← istanziaModelloClient(WEB, WS1)
clientWS2 ← istanziaModelloClient(WEB)
clientWS3 ← istanziaModelloClient(WEB, WS5)
aggiungiNodoServer(clientWS2, WS2, 0.7)
aggiungiNodoServer(clientWS2, WS3, 0.2)
aggiungiNodoServer(clientWS2, WS4, 0.1)
clientDNS ← istanziaModelloClient(DNS, DNS1)
installaModello(C, clientWS1)
installaModello(C, clientWS2)
installaModello(C, clientWS3)
installaModello(C, clientDNS)
while simulazioneInCorso() do
  inviaRichiesta(C, clientDNS)
  attendi(0.1 sec)
  inviaRichiesta(C, clientWS1)
  attendi(Uniforme, 0.3 sec, 0.4 sec)
  inviaRichiesta(C, clientDNS)
  attendi(0.1 sec)
  estraiNodoServer(clientWS2)
  inviaRichiesta(C, clientWS2)
  attendi(1 sec)
  inviaRichiesta(C, clientDNS)
  attendi(0.1 sec)
  inviaRichiesta(C, clientWS3)
  attendi(3 sec)
end while
```

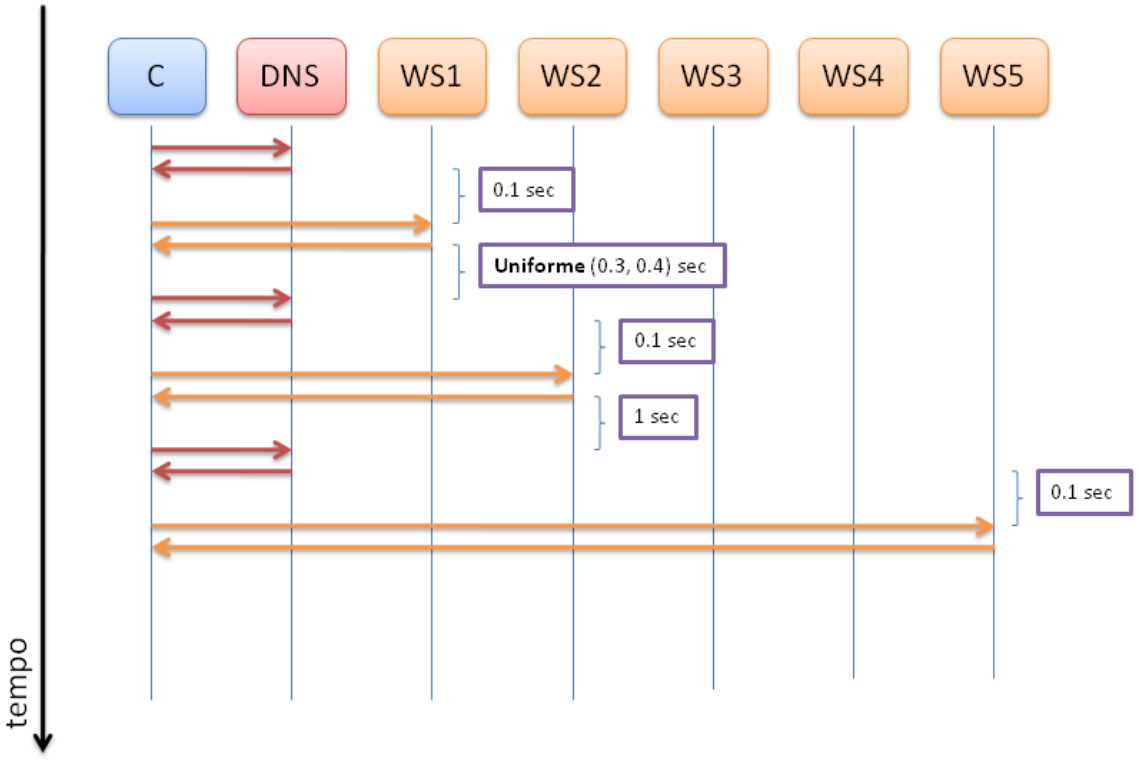

Figura 6.3: Diagramma delle interazioni (Primo esperimento)

- dipendenza *forte* fra **WS1**, **WS2**, **WS3**, **WS4** e il **DNS** (**4 dipendenze**)
- dipendenza *debole* fra **WS2..WS3** e **WS1** (**2 dipendenze**)
- dipendenza *debole* fra **WS5** e **WS2..WS3** (**2 dipendenze**)

Le possibili sequenze di flussi IP sono:

- **DNS** → **WS1** → **DNS** → **WS2** → **DNS** → **WS5**, con probabilità del **70%**
- **DNS** → **WS1** → **DNS** → **WS3** → **DNS** → **WS5**, con probabilità del **20%**
- **DNS** → **WS1** → **DNS** → **WS4** → **DNS** → **WS5**, con probabilità del **10%**

6.5.2 Esecuzione della simulazione

La simulazione viene condotta sulla topologia per **un'ora di attività simulata**. Il simulatore assegna dinamicamente gli indirizzi IP, caratterizzando i cinque servizi WEB, **WS1..WS5**, come:

- **WS1**: 192.168.1.5

- **WS2**: 192.168.1.2
- **WS3**: 192.168.1.3
- **WS4**: 192.168.1.1
- **WS5**: 192.168.1.4

Figura 6.4: Log di un run di simulazione su un nodo

L'immagine 6.4 mostra il log di dell'attività di un nodo che genera un workload **con dipendenze**. Da notare l'interazione con il **DNS**, la selezione del nodo server a run-time e i due *think-time*, valorizzati in **0.3913 secondi** e **un secondo**. La simulazione ha richiesto circa 15 minuti, con un'occupazione media di **CPU** del **25%** e di **RAM** del **5%**. I tracciati generati ammontano a circa **40 MB** per ogni *Autonomous System*, per un totale di **160 MB**. Il tracciato in figura 6.5 mostra una sequenza

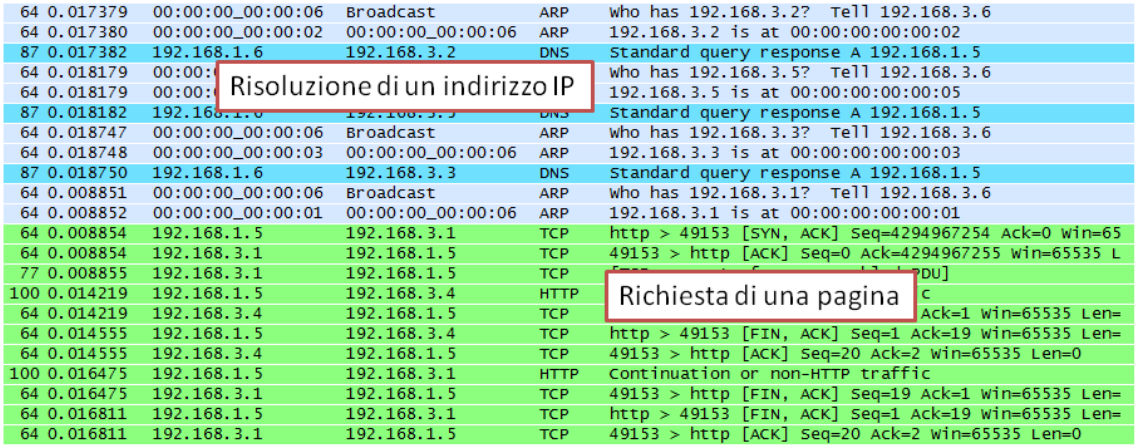

Figura 6.5: Tracciato IP

di pacchetti generati dai client caratterizzati da un carico **con dipendenze**: è possibile individuare una sequenza di risposte DNS (dal server **192.168.1.6** al client **192.168.3.2**) e una sequenza di accessi HTTP (dal client **192.168.3.1** al server **192.168.1.5**)

6.5.3 Analisi mediante DeDALO

Per eseguire DeDALO sono stati monitorati i router dei 4 *Autonomous System* client, acquisendo i tracciati IP del traffico in transito. Per ogni tracciato IP è stata avviata un'istanza del *backend* di DeDALO, eseguita su un processo a parte, per l'analisi e il pre-processamento dei pacchetti.

Ogni *backend* comunica i suoi risultati al *frontend*, che si preoccupa di aggregare i valori e mostrare il grafo delle dipendenze. Per operare, ogni backend ha richiesto circa lo **0.06%** di **CPU** e circa lo **0.83%** di **RAM**. Il *backend* ha il compito di

Nodo client	Intertempo	Servizio IP 1	Servizio IP 2
192.168.6.5	[2 csec]	192.168.1.2:80	192.168.1.6:53
192.168.3.3	[1 csec]	192.168.1.5:80	192.168.1.6:53
192.168.5.2			
192.168.3.5	[56 csec]	192.168.1.6:53	192.168.1.5:80
192.168.3.5			
192.168.3.5	[58 csec]	192.168.1.2:80	192.168.1.5:80
192.168.3.3	[52 csec]	192.168.1.2:80	192.168.1.6:53
192.168.3.3	[50 csec]	192.168.1.2:80	192.168.1.5:80
192.168.3.2	[159 csec]	192.168.1.4:80	192.168.1.6:53
192.168.3.2	[157 csec]	192.168.1.4:80	192.168.1.5:80
192.168.3.2	[101 csec]	192.168.1.4:80	192.168.1.3:80

Figura 6.6: Log di un backend (Primo esperimento)

identificare coppie di **flussi IP**, originati da uno stesso client, e correlarli analizzando gli intertempi. Il log in figura 6.6 mostra una serie di intertempi rilevati fra coppie di flussi IP. Nel caso evidenziato, DeDALO rileva un intertempo di **56 centesimi di secondo** fra un **flusso DNS** e un **flusso HTTP**, generati dal client **192.168.3.5**.

6.5.3.1 Rilevazione delle dipendenze

Le informazioni acquisite dal *backend* consistono in **distribuzioni di intertempi** fra coppie di **flussi IP**. Ogni distribuzione viene inviata al *frontend* descrivendo i servizi coinvolti, **S1** ed **S2**, e gli intertempi rilevati con le relative occorrenze. Il

frontend riceve i dati dai quattro *backend* e aggrega le occorrenze in base alla tripla (*S1, S2, intertempo*).

I risultati sono mostrati in forma di statistiche (vedi figura 6.7), evidenziando ad es. che il **backend** sul nodo **192.168.3.1** ha rilevato per **17** volte un intertempo di **159** centesimi di secondo fra un flusso IP verso **192.168.1.4** e un flusso IP verso **192.168.1.5**. Le figure 6.8 e 6.9 rappresentano le distribuzioni in forma di grafici,

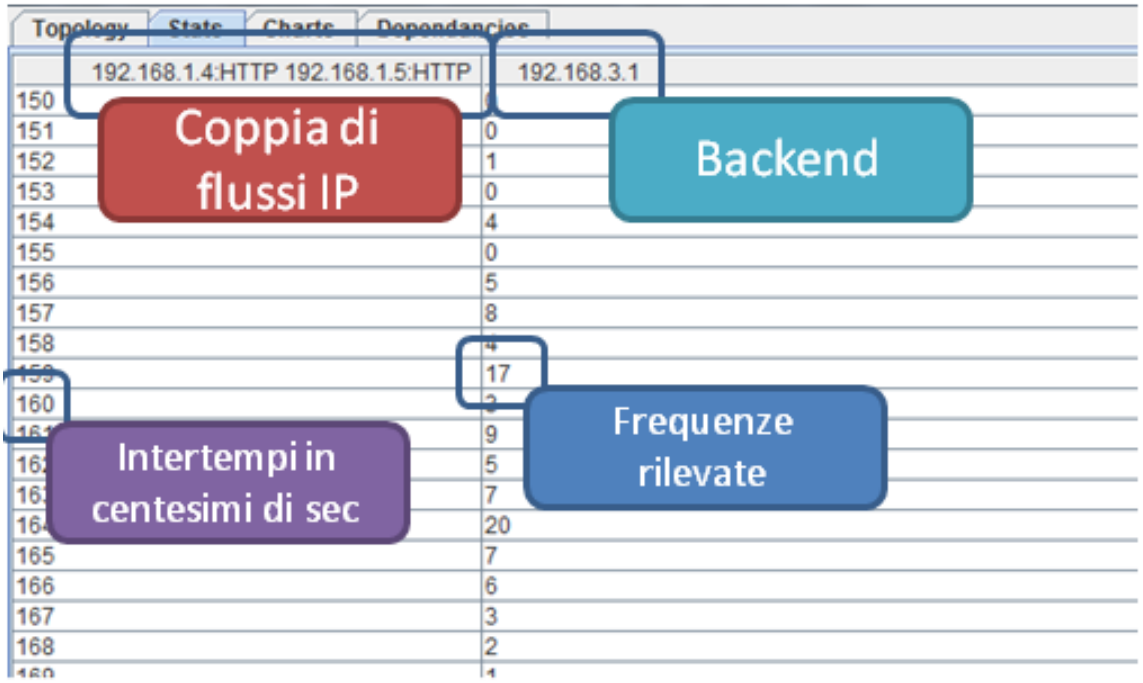

Figura 6.7: Statistiche sugli intertempi

evidenziando in blu la **soglia di dipendenza** e dando una valutazione quantitativa del **grado di dipendenza** fra i servizi considerati.

6.5.3.2 Discussione dei risultati

Le dipendenze rilevate da DeDALO sono presentate tramite un grafo pesato (figura 6.10) in cui si evidenziano due concentrazioni di dipendenze: una verso il **DNS** (192.168.1.6) ed una verso il servizio **WS1** (192.168.1.5). Le percentuali più alte sono definite per le relazioni con il **DNS**, come dimostra la figura 6.11, in cui si mostra la distribuzione di intertempi fra un flusso HTTP (verso **192.168.1.5**) e un flusso DNS (verso **192.168.1.6**), rilevati in sequenza per più di **2000** volte con una variabilità di intertempi di **pochi centesimi di secondo**. Si nota una concentrazione di dipendenze sul servizio **192.168.1.5** (**WS1**), verso cui convergono tre servizi IP:

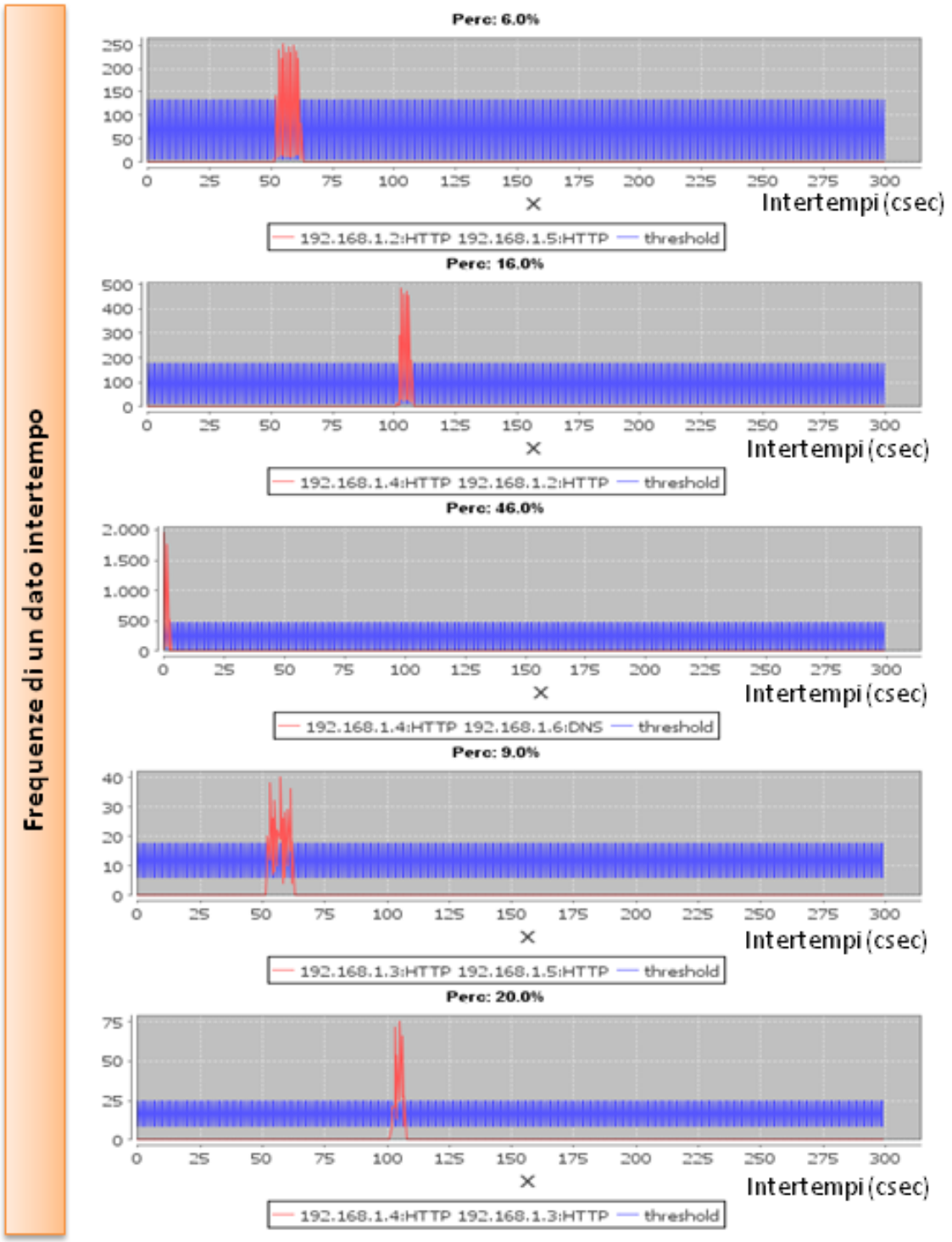

Figura 6.8: Distribuzioni di intertempi (Primo Esperimento) - (prima immagine)

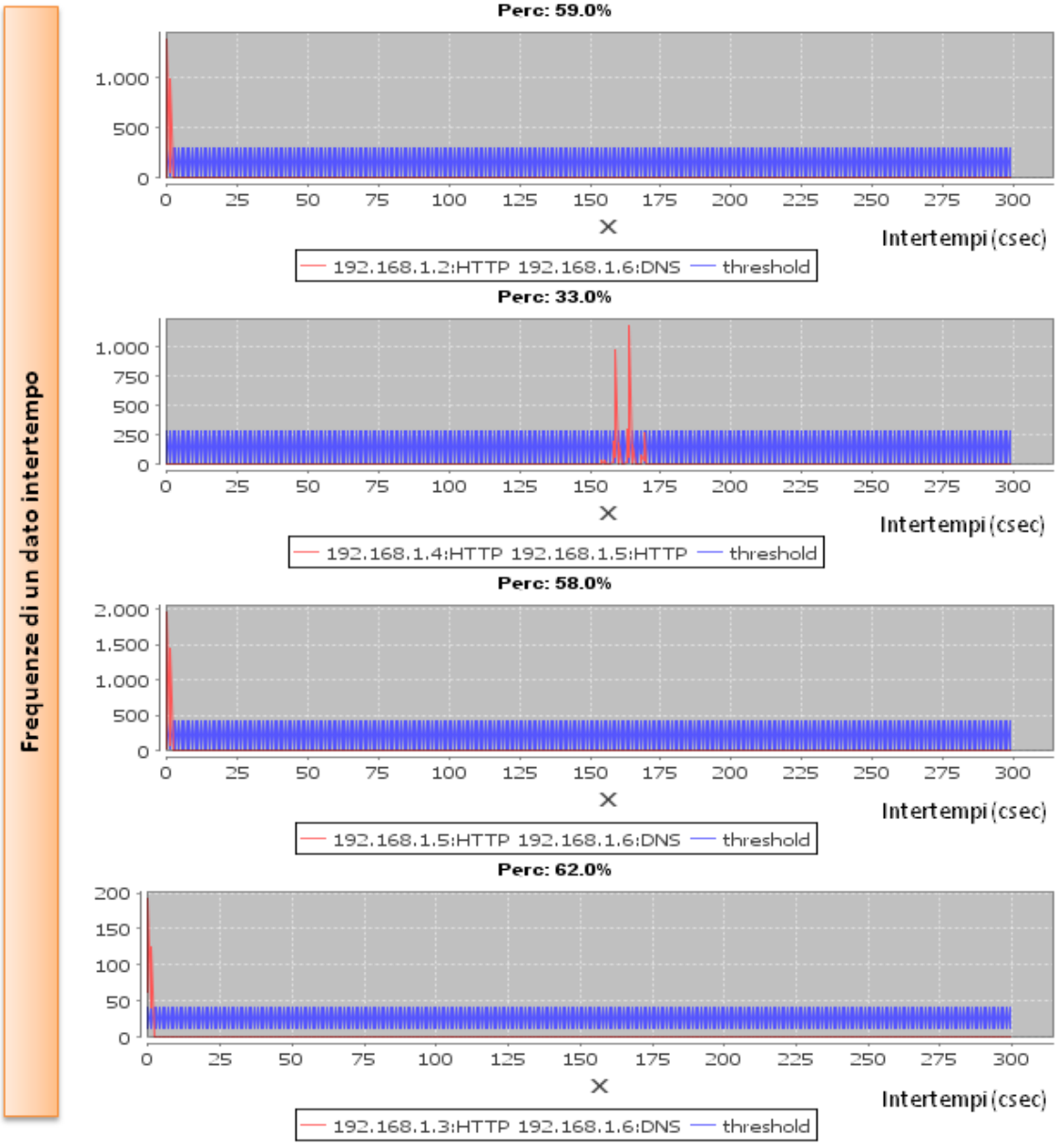

Figura 6.9: Distribuzioni di intertempi (Primo Esperimento) - (seconda immagine)

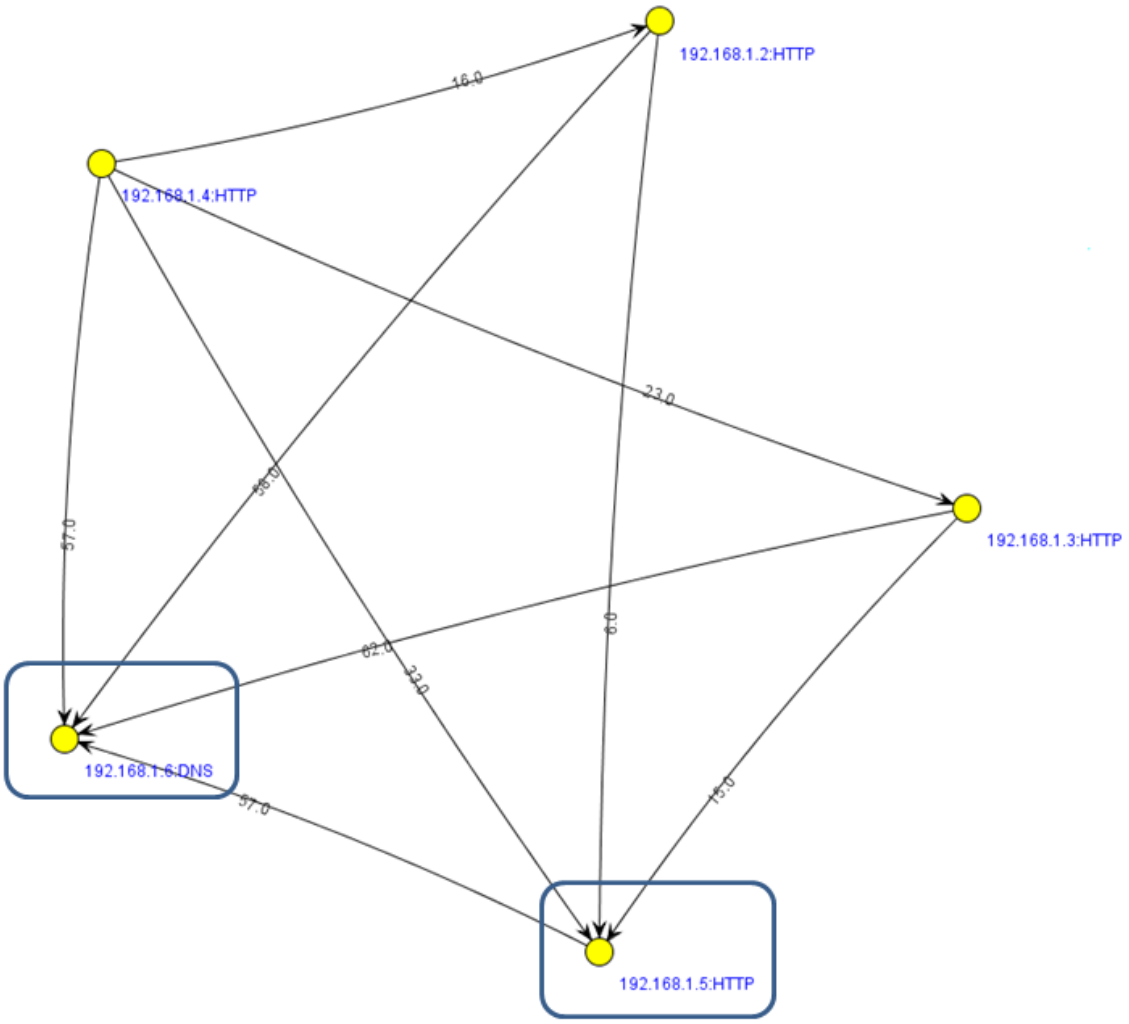

Figura 6.10: Grafo delle dipendenze (Primo esperimento)

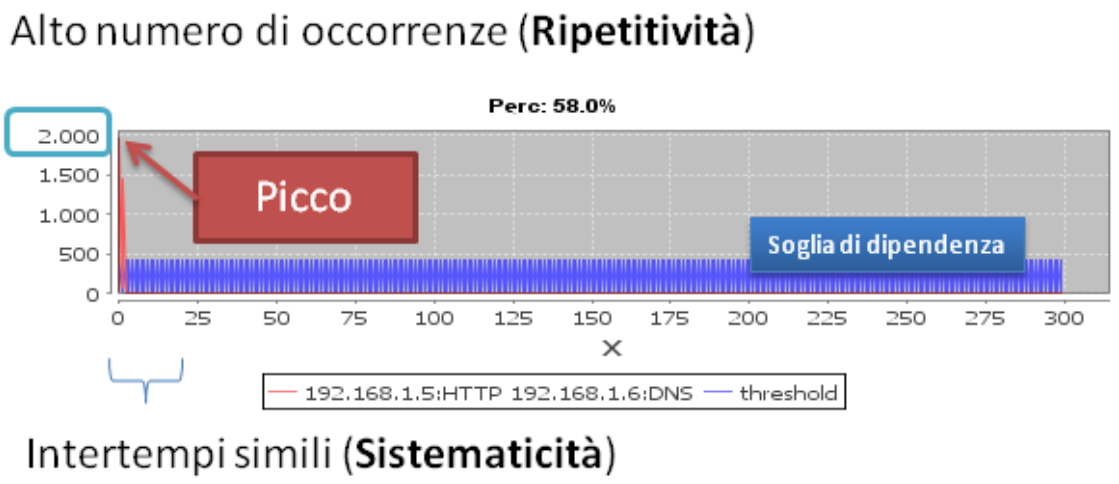

Figura 6.11: Distribuzione di intertempi fra due flussi

- **192.168.1.4** (**WS5**), con probabilità del **33%**
- **192.168.1.2** (**WS2**), con probabilità del **6%**
- **192.168.1.3** (**WS3**), con probabilità del **15%**

DeDALO rileva le dipendenze *deboli* in corrispondenza dei servizi selezionati in maniera probabilistica (**WS2** e **WS3**), escludendo però **WS4** a causa dalle bassa frequenza con con cui appare il relativo flusso IP (condizione di **Ripetitività** non rispettata). Fra gli elementi dipendenti da **WS1**, appare anche **WS5** (**192.168.1.4**), con una percentuale più alta. Questo caso è particolare, poichè nel workflow non vi è una stretta sequenzialità fra **WS5** e **WS1**, ma entrambi i servizi appaiono stabilmente accoppiati entro la finestra di tre secondi. Questo è sufficiente per definirli dipendenti, dato che lo scopo di DeDALO non è quello di ricostruire i path applicativi, ma identificare schemi di accoppiamento fra flussi di dati.

Un caso d'interesse è dato da **192.168.4** (**WS5**), il quale dipende da tutti gli altri nodi, poichè nel workflow è considerato per ultimo. La relazione di dipendenza è più *forte* nel caso di **WS1** (**192.168.1.5**), per lo stesso motivo visto sopra, e più *debole* per quei servizi selezionati su base probabilistica (**WS2** e **WS3**).

Confrontando il *modello di generazione del carico* con il grafo estratto, si può concludere che:

- Sono state rilevate le **4 dipendenze** *forti* dal **DNS**
- Sono state rilevate le **2 dipendenze** *deboli* (+1) da **WS1**
- Sono state rilevate le **2 dipendenze** *deboli* (+1) di **WS5**

6.6 Secondo esperimento

In questo secondo esperimento è simulata l'attività di un'**applicazione distribuita**, composta da **otto servizi WEB** offerti da otto diversi nodi server, dislocati su due *Autonomous system*. I nodi client operano simulando una **cache DNS**, con probabilità di *cahe-hit* del **50%** e probabilità di *failover* (failure sul **DNS** primario e inoltro della richiesta al **DNS** secondario, impostato sul client) del **10%**

6.6.1 Contesto rappresentato

Il *modello di generazione del carico* si compone di **cinque accessi a servizi WEB**, forniti da due gruppi di server (dislocati su diversi *Autonomous system*): **WSA** e **WSB**. Cinque accessi sono definiti a compile-time (**WSA2**, **WSA1** e **WSB3**) e due a run-time. La particolarità di questo scenario, consiste nell'utilizzo

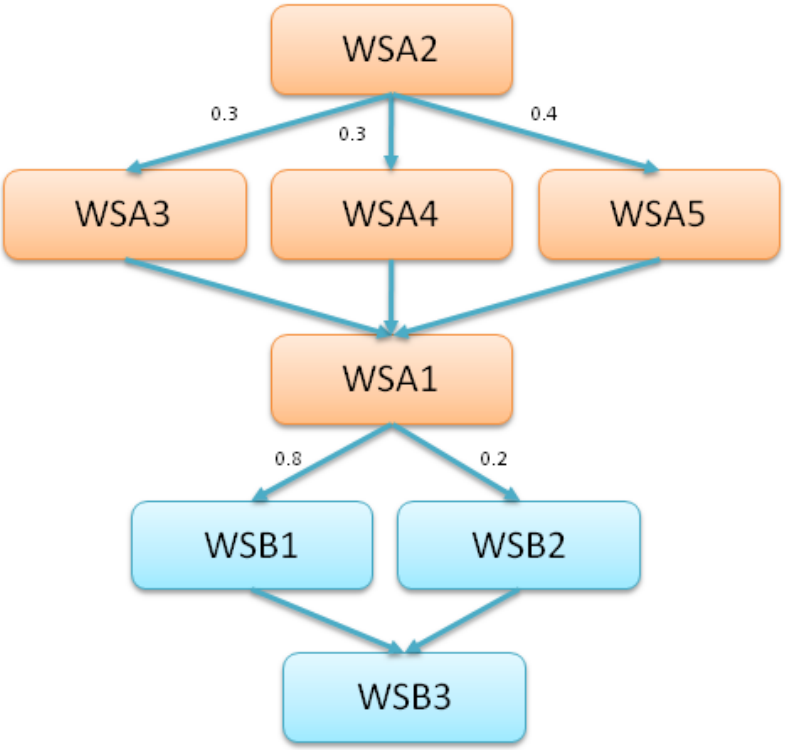

Figura 6.12: Modello di generazione del carico (Secondo esperimento)

di una cache DNS da parte dei nodi client: è definita una probabilità di *cache-hit* pari al 50%, il che significa che un flusso HTTP su due, non sarà preceduto da un flusso DNS. E' anche definita una probabilità di *failover* del 10%: questo valore determina la possibilità di accedere al DNS secondario (**DNS2**) simulando una failure sul primario (**DNS1**).

L'algoritmo 13 illustra il funzionamento del modello DNS così parametrizzato. Le **dipendenze** introdotte sono:

Algoritmo 13 Secondo modello di generazione del carico

```
...istanziazione dei modelli WEB...
clientDNS1 ← istanziaModelloClient(DNS, DNS1)
clientDNS2 ← istanziaModelloClient(DNS, DNS2)
...installazione dei modelli WEB...
installaModello(C, clientDNS1)
installaModello(C, clientDNS2)
cacheHit ← 0.5
failOver ← 0.1
while simulazioneInCorso() do
  if ¬indirizzoInCache(cacheHit) then
    inviaRichiesta(C, clientDNS1)
    if erroreDNS(failOver) then
      inviaRichiesta(C, clientDNS2)
    end if
    attendi(0.1 sec)
  end if
  ...invio delle richieste WEB...
  attendi(3 sec)
end while
```

- dipendenza *forte* fra tutti i servizi **WEB** e il **DNS** (**8 dipendenze**)
- dipendenza *forte* fra **WSA3..WSA5** e **WSA2** (**3 dipendenze**)
- dipendenza *debole* fra **WSB1..WSB2** e **WSA1** (**2 dipendenze**)
- dipendenza *forte* fra **WSB3** e **WSB1** (**1 dipendenza**)
- dipendenza *debole* fra **WSB3** e **WSB2** (**1 dipendenza**)

le possibili sequenze di flussi HTTP, senza considerare il **DNS**, (esemplificate nella figura 6.13) sono:

- **WSA2** → **WSA3** → **WSA1** → **WSB1** → **WSB3**, con probabilità del **24%**
- **WSA2** → **WSA3** → **WSA1** → **WSB2** → **WSB3**, con probabilità del **6%**
- **WSA2** → **WSA4** → **WSA1** → **WSB1** → **WSB3**, con probabilità del **24%**
- **WSA2** → **WSA4** → **WSA1** → **WSB2** → **WSB3**, con probabilità del **6%**

- **WSA2** → **WSA5** → **WSA1** → **WSB1** → **WSB3**, con probabilità del **32%**
- **WSA2** → **WSA5** → **WSA1** → **WSB2** → **WSB3**, con probabilità dell'**8%**

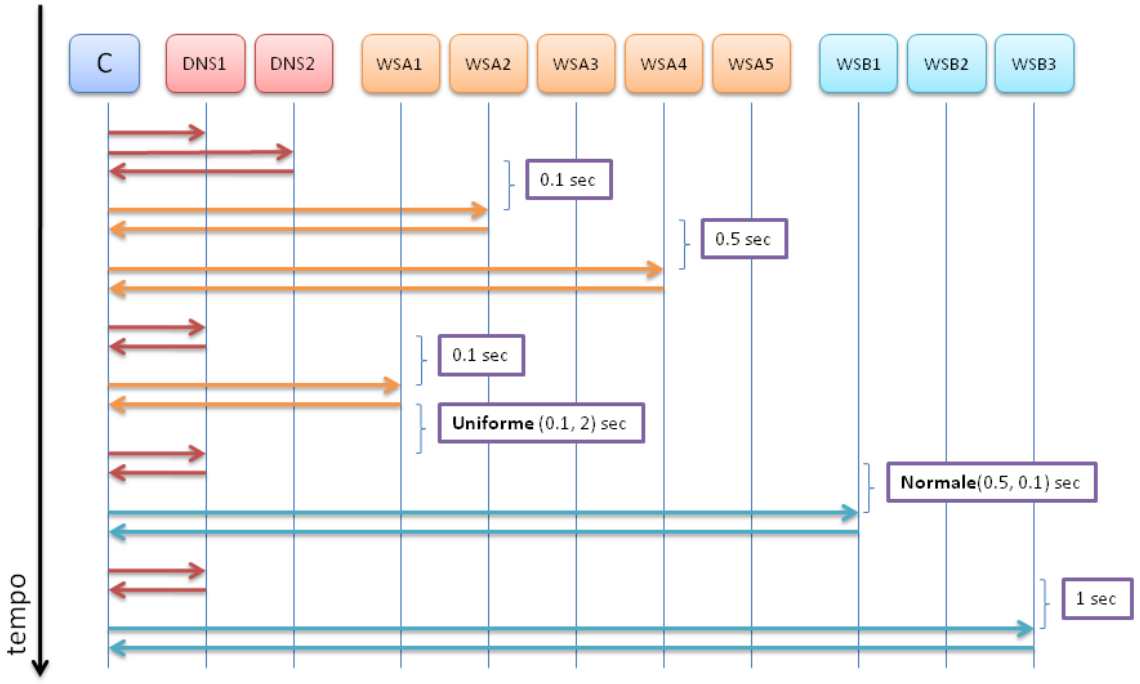

Figura 6.13: Diagramma delle interazioni (Secondo esperimento)

6.6.2 Esecuzione della simulazione

La simulazione viene condotta sulla topologia per **un'ora di attività simulata**. Il simulatore assegna dinamicamente gli indirizzi IP, caratterizzando gli otto servizi WEB, **WSA1..WSA5** e **WSB1..WSB3**, come:

- **WSA1**: 192.168.1.5
- **WSA2**: 192.168.1.2
- **WSA3**: 192.168.1.3
- **WSA4**: 192.168.1.1
- **WSA5**: 192.168.1.4
- **WSB1**: 192.168.2.1
- **WSB2**: 192.168.2.7

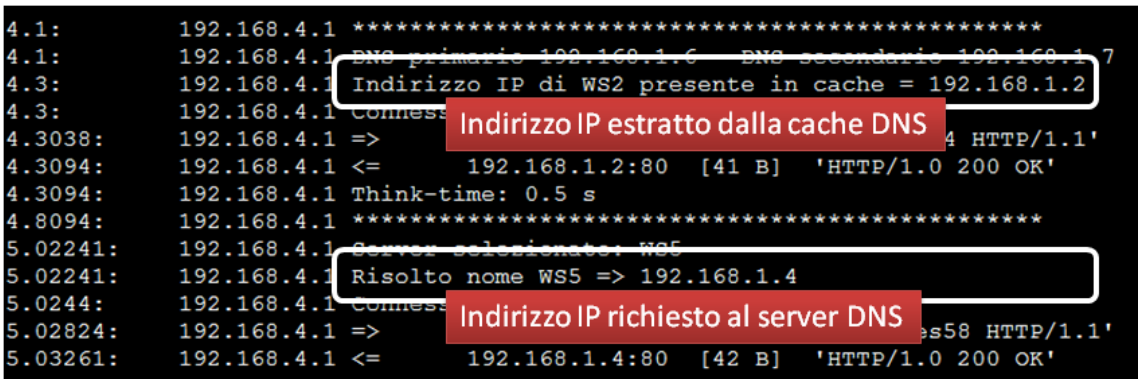

Figura 6.14: Effetto della cache DNS

- **WSB3**: 192.168.2.6

L'immagine 6.14 mostra un caso di *cache-hit*, con la risoluzione di un indirizzo IP tramite la cache DNS del client. La simulazione ha richiesto circa 25 minuti, con un'occupazione media di **CPU** del **27%** e di **RAM** del **5%**. I tracciati generati ammontano a circa **75 MB** per ogni *Autonomous System*, per un totale di **300 MB**. Il tracciato in figura 6.15 mostra un tentativo di accesso al **DNS** primario

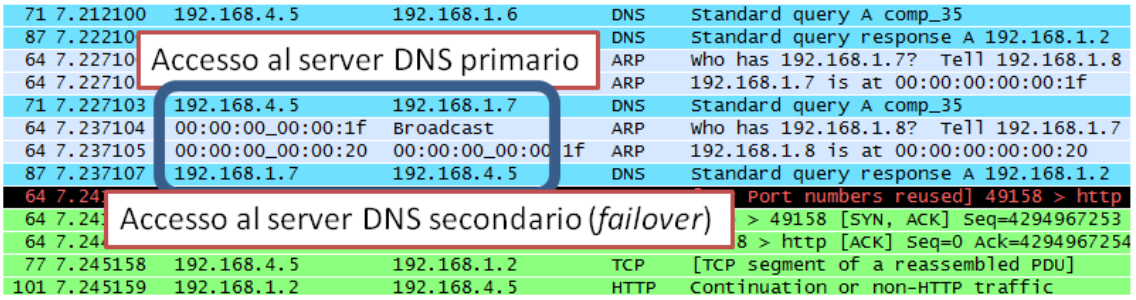

Figura 6.15: Effetto del failorver sul DNS

192.168.1.6, da parte di **192.168.4.5**, seguito da un accesso al **DNS** secondario **192.168.1.7** in seguito al *failover*.

6.6.3 Analisi mediante DeDALO

L'esecuzione di DeDALO, sui tracciati IP simulati, ha dato come output una serie di distribuzioni di intertempi mostrate nelle figure che seguono. I grafici evidenziano andamenti diversi delle curve, causati da intertempi estratti da diverse variabili aleatorie.

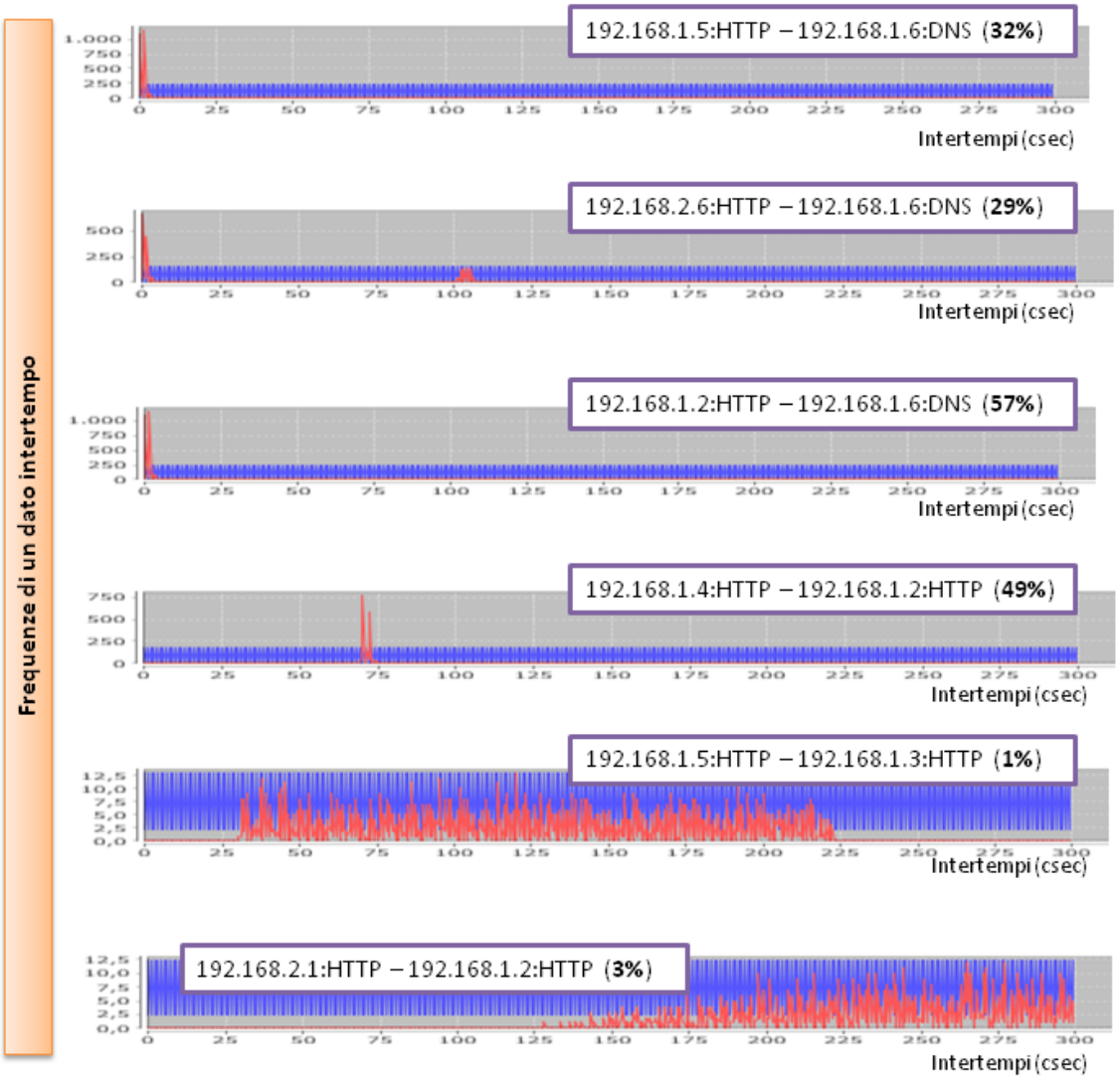

Figura 6.16: Distribuzioni di intertempi (Secondo Esperimento) - (prima immagine)

6.6.3.1 Discussione dei risultati

Il grafo rilevato da DeDALO è mostrato in figura 6.20. Si notano quattro nodi con concentrazioni di dipendenze:

- il **DNS**, da cui dipendono tutti gli 8 servizi
- **192.168.1.2** (**WSA2**) da cui dipendono 5 servizi
- **192.168.1.4** (**WSA5**) da cui dipendono 3 servizi
- **192.168.1.5** (**WSA1**) da cui dipendono 3 servizi

Anche in questo caso, le percentuali più alte sono definite per le relazioni con il **DNS**. I valori sono attenuati, rispetto alle percentuali del primo esperimento, a causa dei

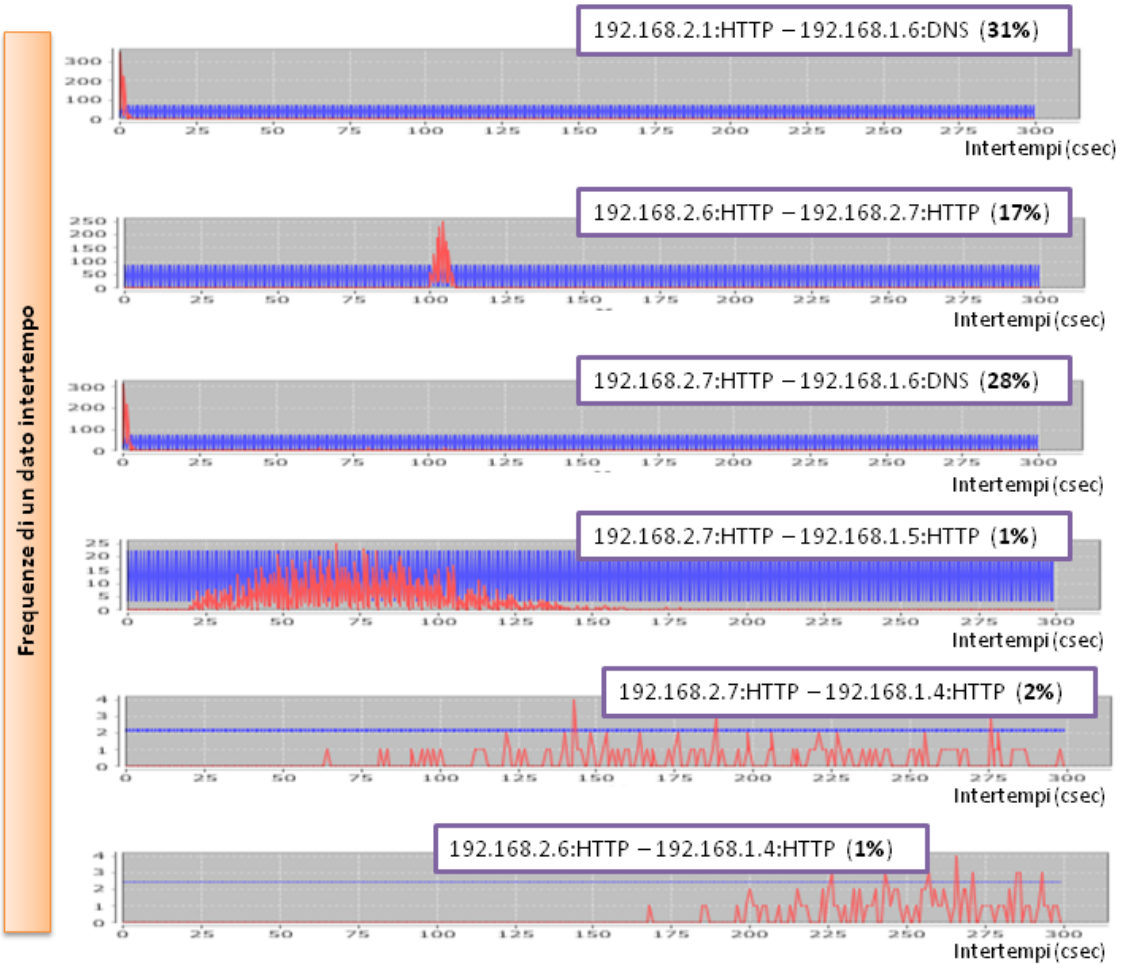

Figura 6.17: Distribuzioni di intertempi (Secondo Esperimento) - (seconda immagine)

vari *cache-hit* verificatisi sui client. DeDALO rileva anche tre servizi HTTP con una discreta concentrazione di servizi dipendenti. Il primo è **192.168.1.2** (**WSA2**), verso cui convergono cinque servizi IP:

- **192.168.1.3** (**WSA3**), con probabilità del **49%**
- **192.168.1.1** (**WSA4**), con probabilità del **49%**
- **192.168.1.4** (**WSA5**), con probabilità del **48%**
- **192.168.2.1** (**WSB1**), con probabilità del **3%**
- **192.168.2.7** (**WSB2**), con probabilità dell'**1%**

Le dipendenze più nette sono rilevate sui tre servizi che si alternano dopo l'accesso a **WSA2**. DeDALO rileva anche una minima situazione di dipendenza con i servizi

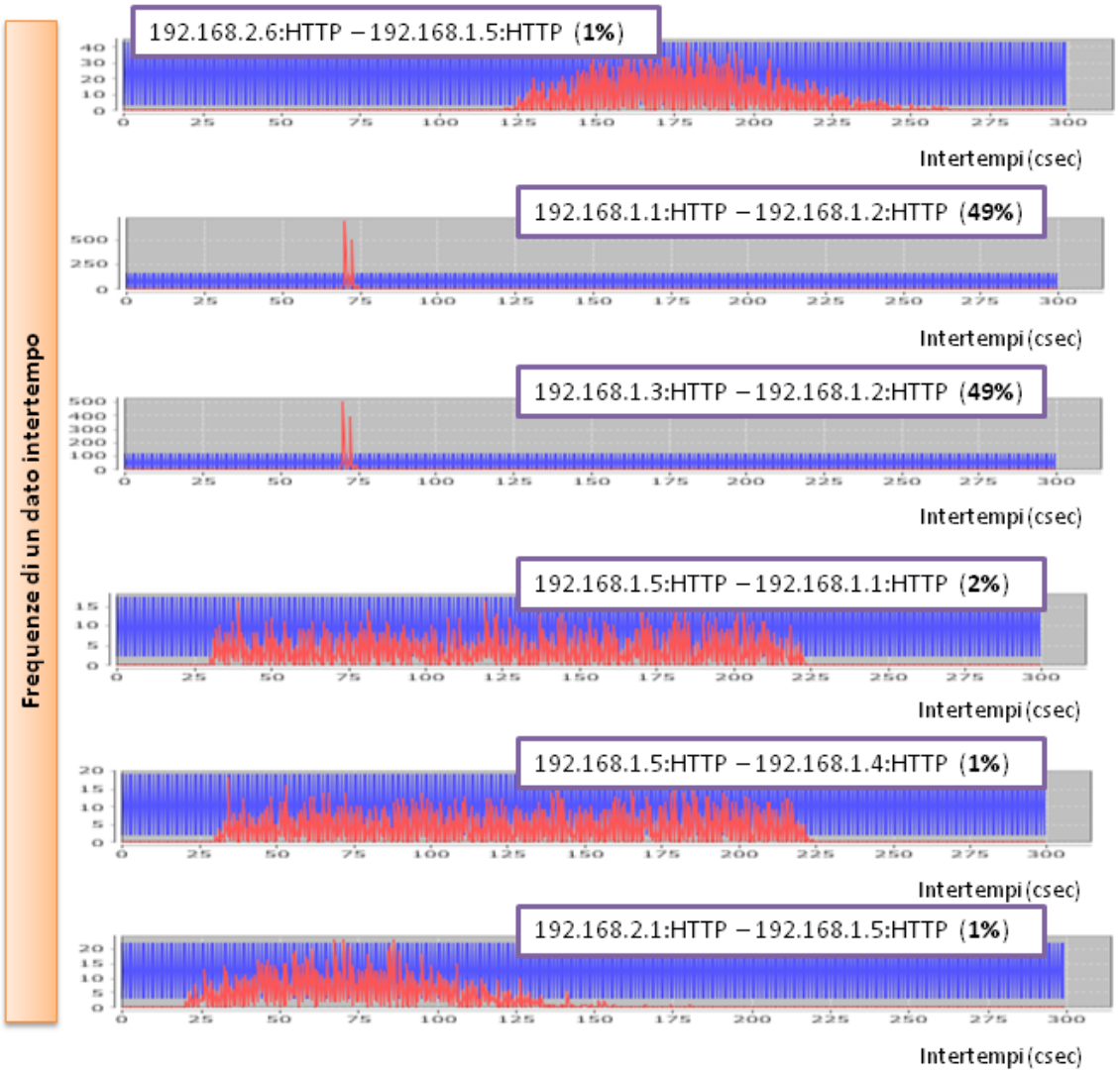

Figura 6.18: Distribuzioni di intertempi (Secondo Esperimento) - (terza immagine)

del secondo *Autonomous system*, ma è talmente debole da risultare quasi trascurabile. Il secondo servizio, evidenziato da DeDALO è **192.168.1.5** (**WSA1**), da cui dipendono tre servizi IP:

- **192.168.2.1** (**WSB1**), con probabilità del **1%**
- **192.168.2.7** (**WSB2**), con probabilità dell'**1%**
- **192.168.2.6** (**WSB3**), con probabilità dell'**1%**

anche in questo caso i risultati sono quasi trascurabili evidenziando delle situazioni al limite della **dipendenza**. Il terzo servizio d'interesse è **192.168.1.4** (**WSA5**) da cui dipendono tre servizi IP:

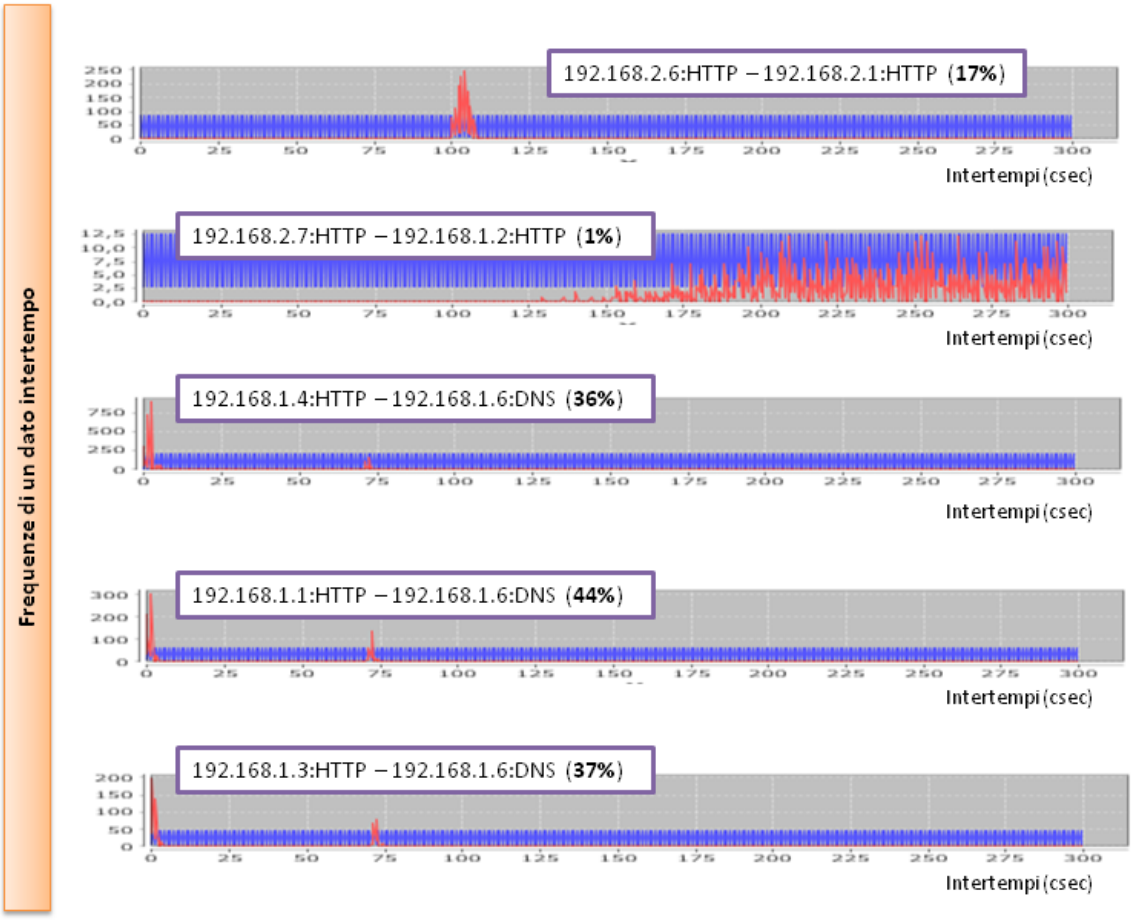

Figura 6.19: Distribuzioni di intertempi (Secondo Esperimento) - (quarta immagine)

- **192.168.1.2** (**WSA2**), con probabilità del **48%**
- **192.168.1.5** (**WSA1**), con probabilità dell'**1%**
- **192.168.2.6** (**WSB3**), con probabilità dell'**1%**

con l'evidenza di una **dipendenza** *forte* espressa da **WSA2**. Vi sono tre servizi che hanno un servizio dipendente ciascuno:

- **192.168.1.1** (**WSA4**), da cui dipende **192.168.1.5** (**WSA1**) con probabilità del **48%**
- **192.168.2.1** (**WSB1**), da cui dipende **192.168.2.6** (**WSB3**) con probabilità dell'**17%**
- **192.168.2.7** (**WSB2**), da cui dipende **192.168.2.6** (**WSB3**)con probabilità dell'**2%**

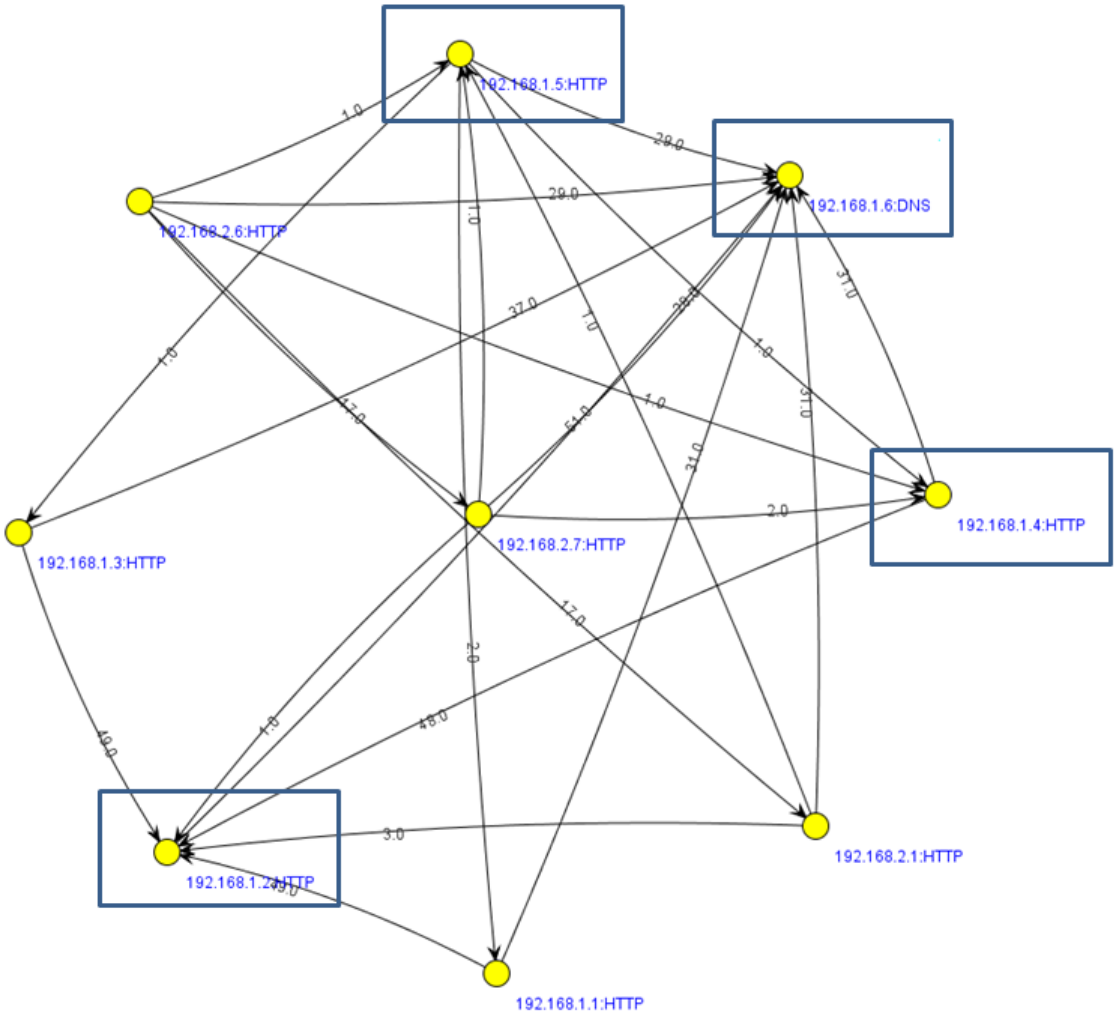

Figura 6.20: Grafo delle dipendenze (Secondo esperimento)

Si nota, infine, come **192.168.2.6** (**WSB3**) dipenda da cinque servizi IP.

Confrontando il *modello di generazione del carico* con il grafo estratto, si può concludere che:

- Sono state rilevate le **8 dipendenze** *forti* dal DNS
- Sono state rilevate le **3 dipendenze** *forti* (+2) da **WSA2**
- Sono state rilevate le **2 dipendenze** *deboli* (+1) di **WSA1**
- E' stata rilevata la **dipendenza** *debole* di **WSB1**
- E' stata rilevata la **dipendenza** *debole* di **WSB2**

in questo caso, DeDALO ha identificato tre **falsi positivi**, commettendo comunque un errore minimo, ma nessun **falso negativo**, identificando e valorizzando correttamente tutte le dipendenze.

6.7 Terzo esperimento

In questo terzo esperimento è simulata l'attività di un'**applicazione distribuita** con architettura *multi-tier*, basata su **cinque server WEB** e **due DBMS server**. Due server WEB saranno configurati per accedere al DBMS in corrispondenza di ogni richiesta dei client, dando luogo a sequenze di flussi annidati. Tutti i nodi con funzionalità *client* fanno uso della cache **DNS** con probabilità di *cache-hit* del **70%** e probabilità di *failover* del 50%, utilizzando, in maniera minima, entrambi i server **DNS**

6.7.1 Contesto rappresentato

Il *modello di generazione del carico* si compone di **tre accessi a servizi WEB**, forniti da tre gruppi di server (dislocati su diversi *Autonomous system*): **WSA** e **WSB**, su protocollo HTTP e **DBMS**, su protocollo SQL. Due accessi sono definiti a compile-time (**WSA1** e **WSB3**) e due a run-time. La particolarità di questo scenario, consiste nell'implementazione di una struttura **multi-tier** in cui due server **WEB** interrogano dei server **DBMS** per comporre la risposta da inviare ai client. L'algoritmo 14 illustra il funzionamento di un nodo server **S** su cui opera un server **WEB** così configurato. Le **dipendenze** introdotte sono:

Algoritmo 14 Terzo modello di generazione del carico

```
serverWS ← istanziaModelloServer(WEB, 80)
clientDBMS ← istanziaModelloClient(DBMS, DBMS1)
clientWS ← istanziaModelloClient(WEB, WSA2)
installaModello(S, serverWS)
installaModello(S, clientDBMS)
installaModello(S, clientWS)
while simulazioneInCorso() do
  C ← accettaConnessione(serverWS)
  richiesta ← riceviRichiesta(C, serverWS)
  attendi(0.5 sec)
  risposta ← null
  if selezionaModelloClient() = clientDBMS then
    ...risoluzione dell'indirizzo IP...
    inviaRichiesta(S, clientDBMS)
    risposta ← riceviRisposta(S, clientDBMS)
  else
    ...risoluzione dell'indirizzo IP...
    inviaRichiesta(S, clientWS)
    risposta ← riceviRisposta(S, clientWS)
  end if
  risposta ← componiRisposta(risposta)
  inviaRisposta(C, serverDBMS, risposta)
end while
```

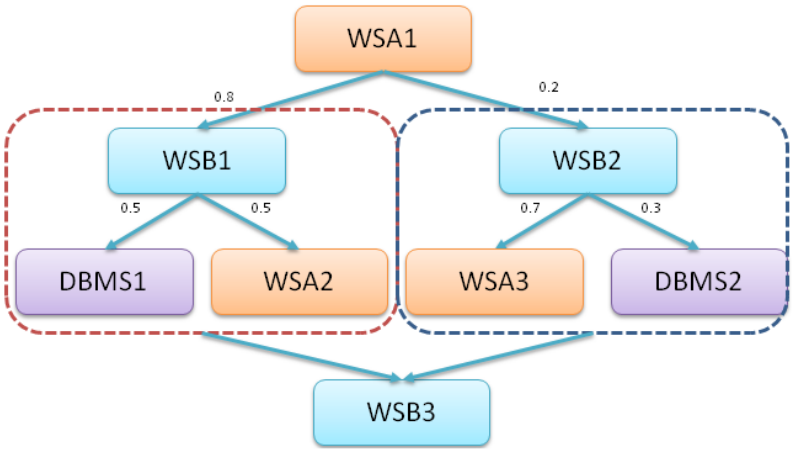

Figura 6.21: Modello di generazione del carico (Terzo esperimento)

- dipendenza *debole* fra **WSB1** e **WSA1** (**1 dipendenza**)
- dipendenza *debole* fra **WSB2** e **WSA1** (**1 dipendenza**)
- dipendenza *forte* fra **WSB3** e **WSB1..WSB2** (**2 dipendenze**)

Le possibili sequenze di flussi IP (esemplificate nella figura 6.22) sono:

- **WSA1** → **WSB1** → *DBMS1* → **WSB3**, con probabilità del **40%**
- **WSA1** → **WSB1** → *WSA2* → **WSB3**, con probabilità del **40%**
- **WSA1** → **WSB2** → *WSA3* → **WSB3**, con probabilità del **14%**
- **WSA1** → **WSB2** → *DBMS2* → **WSB3**, con probabilità del **6%**

6.7.2 Esecuzione della simulazione

La simulazione viene condotta sulla topologia per **un'ora di attività simulata**. Il simulatore assegna dinamicamente gli indirizzi IP, caratterizzando i sei servizi WEB, **WSA1..WSA3** e **WSB1..WSB3**, e i due database server **DBMS** come:

- **WSA1**: 192.168.1.5
- **WSA2**: 192.168.1.2
- **WSA3**: 192.168.1.3
- **WSB1**: 192.168.2.1

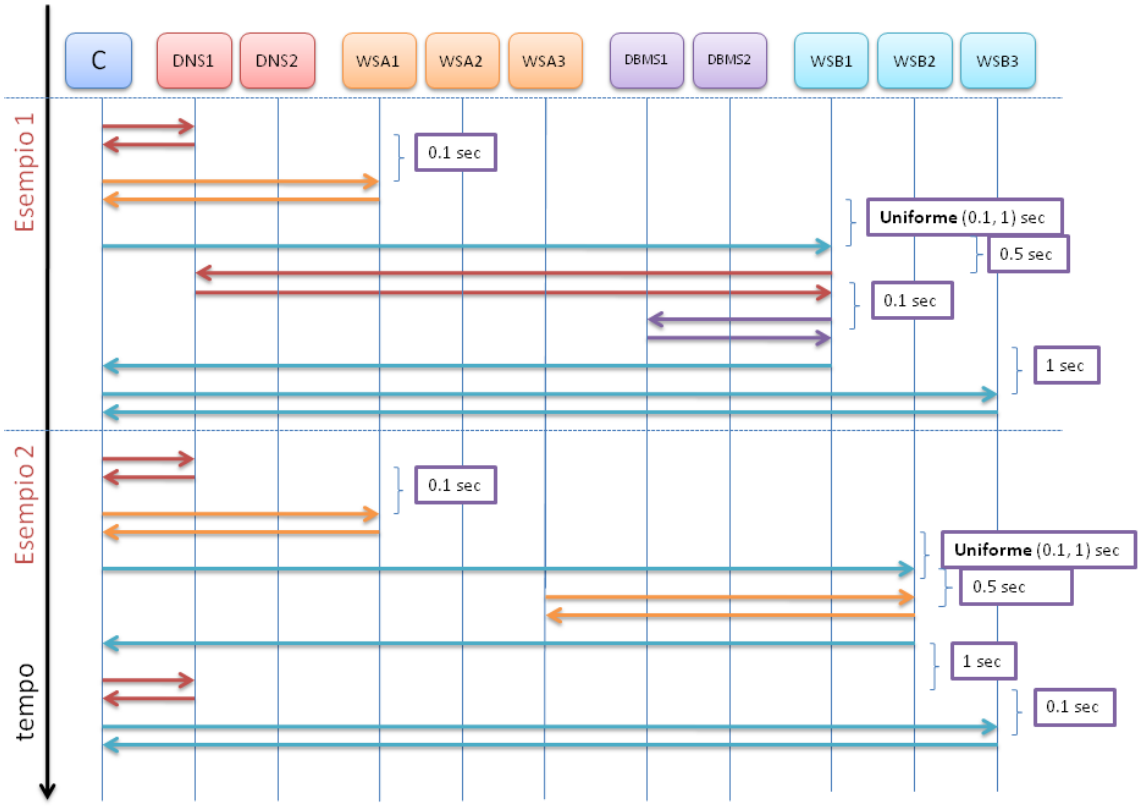

Figura 6.22: Diagramma delle interazioni (Terzo esperimento)

- **WSB2**: 192.168.2.7
- **WSB3**: 192.168.2.6
- **DBMS1**: 192.168.2.4
- **DBMS2**: 192.168.2.3

La simulazione ha richiesto circa 20 minuti, con un'occupazione media di **CPU** del **20%** e di **RAM** del **7%**. I tracciati generati ammontano a circa **30 MB** per ogni *Autonomous System*, per un totale di **120MB**.

6.7.3 Analisi mediante DeDALO

L'esecuzione di DeDALO, sui tracciati IP simulati, ha dato come output una serie di distribuzioni di intertempi mostrate nelle figure che seguono. I grafici evidenziano andamenti diversi delle curve, causati da intertempi estratti da variabili aleatorie **Normali** e **Uniformi**. Il log in figura 6.25 mostra una serie di intertempi rilevati fra coppie di flussi IP di tipo HTTP e DBMS.

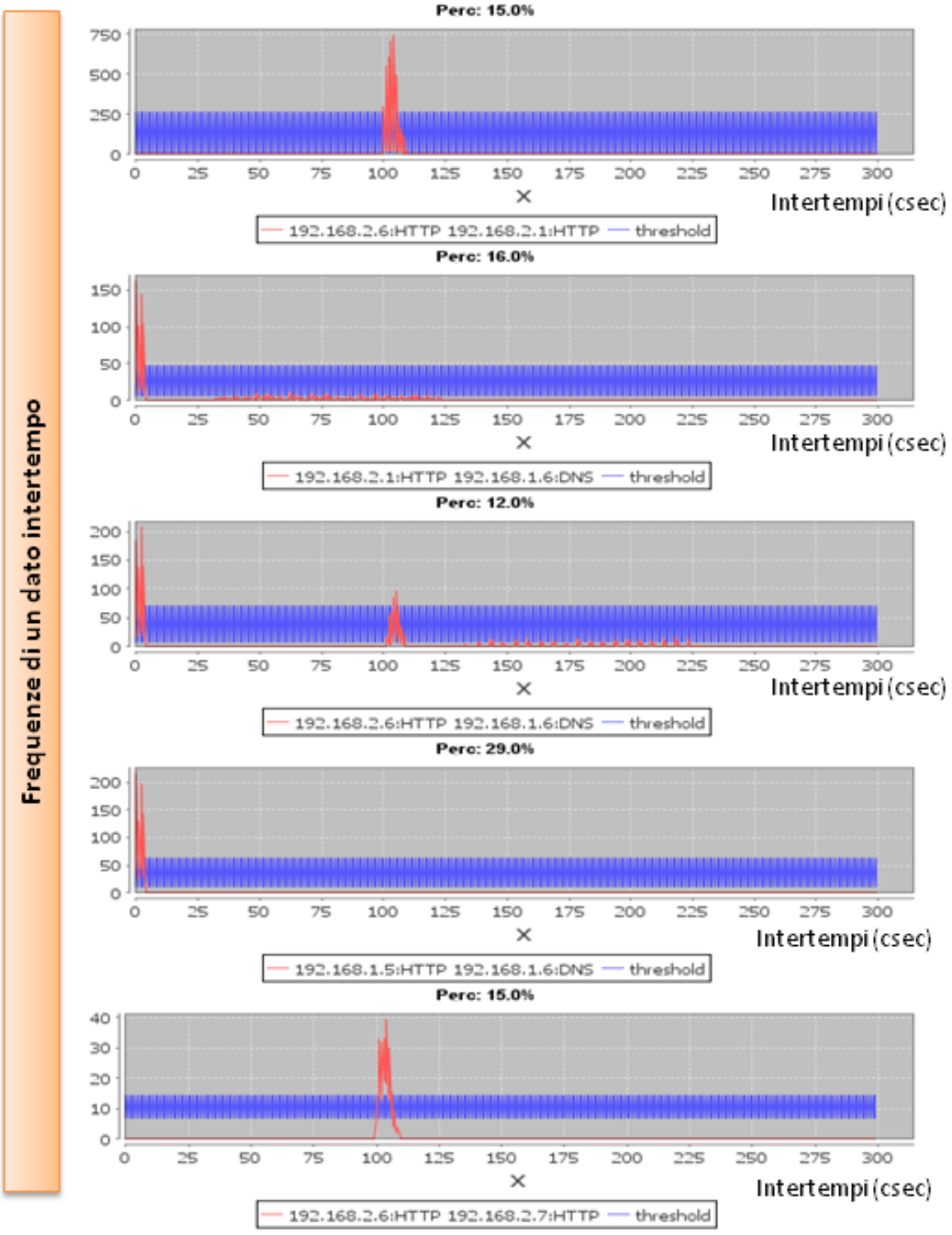

Figura 6.23: Distribuzioni di intertempi (Terzo Esperimento) - (prima immagine)

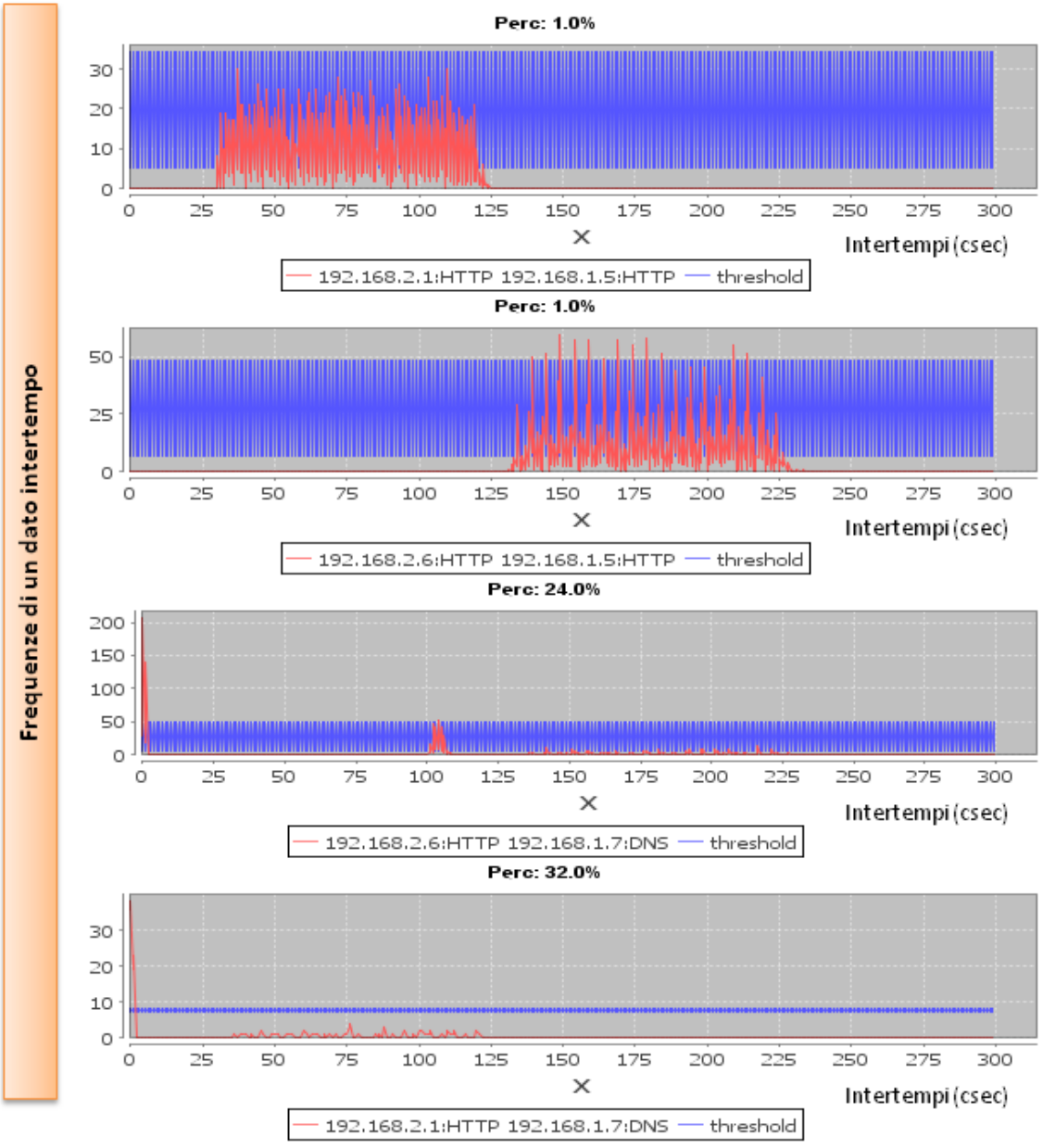

Figura 6.24: Distribuzioni di intertempi (Terzo Esperimento) - (seconda immagine)

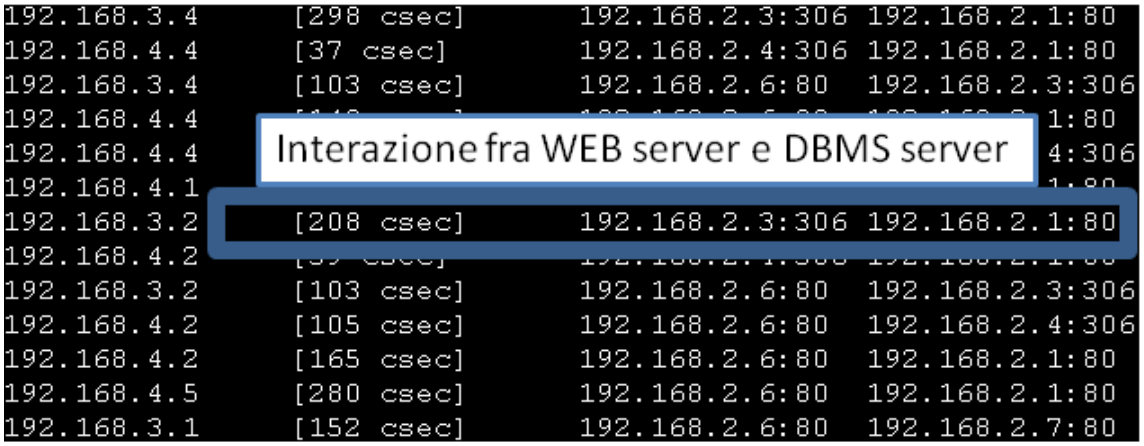

Figura 6.25: Log di un backend (Terzo esperimento)

6.7.3.1 Discussione dei risultati

Il grafo rilevato da DeDALO è mostrato in figura 6.26. Si notano tre nodi con concentrazioni di dipendenze, di cui due sui DNS

- il **DNS** primario (**192.168.1.6**), con 3 servizi dipendenti
- il **DNS** secondario (**192.168.1.7**), con 3 servizi dipendenti

e una su **WSA1** (**192.168.1.5**) da cui dipendono 2 servizi:

- **WSB1** (**192.168.2.1**), con percentuale dell'**1%**
- **WSB3** (**192.168.1.6**), con percentuale dell'**1%**

vi è poi da sottolineare una concentrazione di **dipendenze** da **WSB3** (**192.168.2.6**) che, essendo l'ultimo elemento del workflow, risulta dipendente sia da **WSA1** (**192.168.1.5**) che da **WSB1** (**192.168.2.1**). Le percentuali di dipendenza verso il **DNS** sono molto variabili e DeDALO rileva alcuni casi di **dipendenza** ma con grado dissimile. Confrontando il *modello di generazione del carico* con il grafo estratto, si può concludere che:

- E' stata rilevata la **dipendenza** *debole* fra **WSB1** e **WSA1**
- Non è stata rilevata la **dipendenza** *debole* fra **WSB2** e **WSA1**
- E' state rilevata, ma con grado non corretto, la **dipendenza** fra **WSB3** e **WSB1**

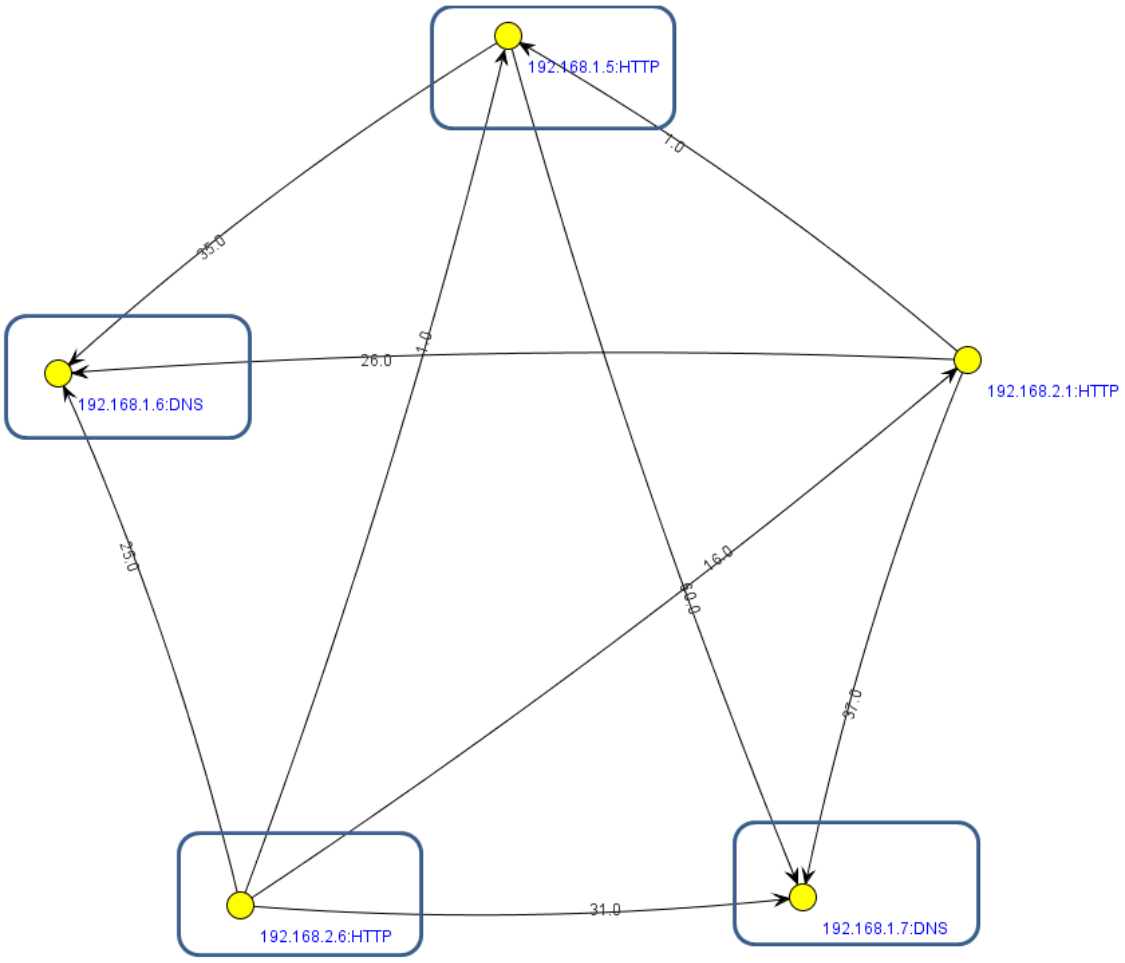

Figura 6.26: Grafo delle dipendenze (Terzo esperimento)

- Non è stata rilevata la **dipendenza** *debole* fra **WSB2** e **WSB1**

In questo esperimento DeDALO commette due errori, non rilevando alcune casistiche che, seppur rare, potrebbero comunque esibire delle **dipendenze**. Nel caso specifico, l'esecuzione in mutua esclusione di servizi, l'utilizzo di cache DNS e intertempi molto variabili dà luogo a sequenze di flussi che occorrono con una frequenza talmente bassa da essere ignorate o considerate casuali per osservazioni su archi di tempo ridotti. Si è comunque scelto di limitare l'osservazione a un'ora per valutare l'operato di DeDALO su brevi intervalli di tempo, accettando qualche **falso negativo** (di minimo impatto sul risultato) in cambio di una 'diagnosi' veloce sul sistema.

Capitolo 7

Conclusioni e Sviluppi Futuri

L'obiettivo iniziale di questa tesi è stato lo studio e l'implementazione di un sistema automatizzato (**DeDALO**) per la rilevazione di dipendenze in sistemi distribuiti.

E' stata introdotta e descritta la problematica delle **dipendenze** fra componenti di un sistema distribuito, motivando la necessità di **rilevare** e **valorizzare** queste relazioni, determinando degli **schemi logici di dipendenza** fra gli elementi analizzati.

Sono stati **considerati** i lavori più importanti nel campo del *Dependency discovery*, puntualizzando il concetto di **dipendenza** e valutando diversi approcci per la rilevazione in modalità *black-box*. Ne è stato selezionato uno, *Orion* [CZMB08], su cui basare il modello algoritmico di rilevazione implementato in DeDALO. Tale modello analizza la **sistematicità** con cui due **flussi** di dati co-occorrono in finestre di tempo.

E' stata, quindi, presentata la struttura applicativa di DeDALO, contributo primario di questa tesi, descrivendo l'implementazione delle componenti di rilevazione e analisi. E' stata definita una **metrica** inedita per la valutazione **quantitativa** del **grado di dipendenza** fra gli elementi individuati.

Per testare il funzionamento di DeDALO, e' stato progettato un **ambiente di simulazione**, basato su NS3, implementando dei modelli applicativi di comuni protocolli IP. Tali modelli costituiscono un'innovazione importante in un campo dominato dalla modellazione dei soli livelli fisici.

La **validazione** di DeDALO è stata condotta simulando **tre casi di studio** di diversa complessità, valutando la capacità del tool di rilevare e valorizzare le dipendenze introdotte da applicazioni distribuite. I risultati ottenuti hanno mostrato

un'**elevata efficacia** di DeDALO, sia nel determinare le relazioni di dipendenza, che nel valorizzarle, presentando casi molto limitati di **falsi positivi** e **falsi negativi**.

I possibili sviluppi futuri riguardano tre diversi ambiti:

- **modello di rilevazione**: estendere l'algoritmo definendo delle tecniche di analisi per applicazioni **multi-tier** o *peer-to-peer*
- **architettura di DeDALO**: creare una struttura gerarchica, definendo dei nodi intermedi fra *backend* e *frontend*
- valutazione del **grado di dipendenza**: raffinare la formula per abbattere il numero di **falsi positivi** nelle rilevazioni

Bibliografia

[AB02] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks, 2002. Department of Physics, University of Notre Dame.

[BCG+07] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A. Maltz, and Ming Zhang. Towards highly reliable enterprise network services via inference of multi-level dependencies, 2007. Microsoft Research.

[CZMB08] Xu Chen, Ming Zhang Z., Morley Mao, and Paramvir Bahl. Automating network application dependency discovery: experiences, limitations, and new solutions, 2008. Microsoft Research, University of Michigan.

[EMC] EMC. It operations intelligence. http://www.emc.com/products/family/.

[HP] HP. Network management center. https://h10078.www1.hp.com/cda/.

[IBM] IBM. Tivoli. http://www-01.ibm.com/software/tivoli/.

[KCK08] Srikanth Kandula, Ranveer Chandra, and Dina Katabi. Whats going on? learning communication rules in edge networks, 2008. Microsoft Research.

[KK01] Alexander Keller and Gautam Kar. Determining service dependencies in distributed systems, 2001. IBM T.J. Watson Research Center, 1101 Kitchawan Rd., Route 134, Yorktown Heights, N.Y. 10598.

[Lac10] Mathieu Lacage. The network simulator 3, 2010. INRIA, Sophia Antipolis - Méditerranée, 2004 route des Lucioles - BP 93 06902 Sophia Antipolis Cedex.

[MLMB01] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation, 2001. Boston University.

[Pre10] Roger S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill Higher Education, 7th edition, 2010.

[SJT02] Rinaldi S.M, Peerenboom J.P, and Kelly T.K. Identifying, understanding, and analyzing critical infrastructure interdependencies, 2002. Argonne National Laboratorry, 9700 S. Cass Ave., Bldg. 900, Argonne.

[WW09] E. Weingartner and H. Lehn K. Wehrle. A performance comparison of recent network simulators, 2009. RWTH Aachen University, Aachen, Germany.