ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

Campus di Cesena - Scuola di Scienze

Corso di Laurea in Scienze e Tecnologie Informatiche

ANALISI DEL PROTOCOLLO TOR E DEL SIMULATORE SHADOW

Relazione finale in Reti di Calcolatori

Relatore
GABRIELE D'ANGELO

Presentata da
DAVIDE ALESSANDRINI

Terza sessione
Anno Accademico 2012/2013

Prefazione

Agli inizi degli anni ’60 alcune delle maggiori potenze mondiali misero a punto, durante la guerra fredda, un nuovo sistema di difesa e controspionaggio. Esso si basava sulla prima divulgazione scientifica in cui si alludeva alla possibilità di creare una rete di computer collegati tra loro, pubblicata dagli statunitensi Joseph C.R. Licklider e Welden E. Clark, ricercatori del Massachusetts Institute of Technology. Licklider e Clark pubblicarono “*On-line man computer communication*” nel 1962, assegnando anche un nome alla rete da loro teorizzata: “*Intergalactic Computer Network*”. Tuttavia fu il progetto “*ARPANET*” a gettare le basi della moderna Internet, sviluppato negli stessi anni dal Ministero della Difesa statunitense. A questo periodo inoltre, risalgono i primi studi sui moderni protocolli che vengono attualmente utilizzati, come il modello client/server e i principali applicativi dell’epoca, Telnet ed FTP[1]. Alla fine degli anni ’70 la rete contava quattro nodi che collegava le principali università americane. Nel giro di dieci anni fu eseguito il primo collegamento transoceanico. L’Italia fu il terzo paese europeo, dopo Norvegia e Inghilterra, a connettersi ad ARPANET nel novembre del 1986. [**1**]

[1] File Transfer Protocol

In seguito nel 1991 presso il CERN di Ginevra, il ricercatore Tim Berners-Lee definì il protocollo HTTP[2], un sistema che permette la lettura ipertestuale di documenti grazie all'uso dei cosiddetti link (collegamenti). Pochi anni più tardi grazie al primo browser, Mosaic, nacque il Word Wide Web. Da quel momento in avanti, anche grazie alla diffusione dei personal computer nella vita di tutti i giorni, Internet passò dall'essere una rete a esclusivo utilizzo di pochi utenti per usi accademici o militari, a diventare la principale fonte di comunicazione moderna; attualmente possiamo stimare circa 2.1 miliardi di utenti connessi da tutto il mondo, 3.146 miliardi di indirizzi email e 800 milioni di siti Internet. [**2**]

Fin dall'inizio, l'idea di base era che chiunque fosse collegato poteva liberamente accedere ai contenuti e reperire informazioni. Ovviamente con l'aumentare del numero di utenti e dei servizi offerti si è reso necessario affrontare problemi importanti, come sicurezza e privacy. La sicurezza è un tema attualmente molto rilevante, in cui lo scopo principale è quello di fornire un sistema di comunicazione rapido, sicuro ed affidabile, in cui ogni utente possa fornire dei dati sensibili senza pericolo che questi vengano intercettati da terze persone. Basti pensare ai moderni sistemi di e-commerce, in cui sono comunicati codici di carte di credito e password. Informazioni di questo genere in mano a persone malintenzionate potrebbero provocare danni ingenti. Altro punto non meno importante riguarda la privacy. Sempre più frequentemente gli utenti vengono a conoscenza del fatto che i loro dati trasmessi vengono filtrati ed analizzati per realizzare ricerche di mercato al fine di estrapolare informazioni sensibili utili per ricavare denaro.

L'obiettivo che si pone questa tesi è quello di analizzare alcune soluzioni esistenti riguardo gli ultimi due quesiti appena descritti. Nello specifico sarà illustrato il programma Tor[3], un sistema di comunicazione grazie al quale si garantisce l'anonimato in Internet. Sponsorizzato inizialmente dall'US Naval Research Laboratory, all'origine veniva

[2] HyperText Transfer Protocol
[3] The Onion Router

utilizzato per poter effettuare comunicazioni militari criptate; è stato successivamente un progetto della Electronic Frontier Foundation ed ora è gestito da The Tor Project, un'associazione senza scopo di lucro che si occupa del relativo sviluppo. Le ricerche e gli sviluppi riguardanti questa tecnologia vengono tuttavia resi difficili da problemi di scalabilità e dalla difficoltà di riprodurre risultati affidabili. Nel corso della tesi saranno illustrati gli studi riguardanti il simulatore Shadow, un progetto Open Source gestito da The Tor Project, che permette la simulazione del protocollo Tor. I risultati ottenuti dai test effettuati sul simulatore, possono essere riapplicati in seguito alla rete reale Tor, poiché grazie a Shadow è possibile testare ed ottenere risultati attendibili circa il comportamento e la corretta esecuzione del software Tor.

Indice

Capitolo 1

Introduzione

Internet è un ecosistema complesso, che ha sostanzialmente trasformato, se non addirittura rivoluzionato il nostro modo di comunicare, di scambiare informazioni e organizzare attività di natura sociale, politica ed economica. È il luogo principale in cui si scambiano pensieri, conoscenze e in quanto tale è una risorsa inestimabile per la ricerca e lo sviluppo. È il motore dell'economia globale e la sua infrastruttura è fondamentale al pari delle reti dell'acqua, elettricità e trasporti. Internet sta rivoluzionando il modo in cui gli utenti vivono, migliorando l'offerta di beni e servizi e rendendo più efficiente complessi sistemi burocratici. Per queste ragioni è importante preservare la struttura della rete e con essa i dati riguardanti milioni di persone. Inoltre, proprio perché società, imprese e governi utilizzano ormai Internet per svolgere funzioni essenziali, è diventato prioritario affrontare le minacce alla sicurezza.

1.1 CONCETTI CHIAVE

Privacy e sicurezza sono due termini che sempre più spesso ritroviamo insieme nel mondo virtuale, in cui le nostre informazioni possono essere

violate in qualunque momento ed, in effetti, spesso è così avviene nella realtà. Questo accade, anche se i nostri dati dovrebbero essere protetti da sistemi sicuri e da normative che tutelano le informazioni personali. Il 1° gennaio del 2004 è entrata in vigore in Italia una nuova legge chiamata a riordinare e regolare quanto necessario in riferimento alla privacy, ovvero alla protezione dei dati personali. Il testo in questione, passato poi immediatamente alla storia come "Testo unico sulla privacy" è il "Decreto legislativo 30 giugno 2003, n. 196 Codice in materia di protezione di dati individuali". [**3**] Le nuove procedure, il cui controllo è stato affidato al Garante della Privacy, mirano a limitare i danni di eventuali "*data breach*"[4] sia per gli utenti sia per le aziende. Inoltre viene fornita una maggiore tutela dei consumatori contro lo spam[5] e l'utilizzo dei cookie per monitorare la navigazione.

Occorre tuttavia ricordare che senza un adeguato sistema di protezione strutturato e sicuro ogni dato è in pericolo. Gli attacchi informatici rappresentano una minaccia sempre più importante, soprattutto per le aziende e governi che hanno l'obbligo di proteggere i propri dati. Lo standard ISO 27002:2007 stabilisce i concetti fondamentali riguardo la sicurezza informatica, nei termini di integrità, riservatezza e disponibilità. Si tratta nel dettaglio di tutte le tecniche e le modalità operative che dovrebbero essere seguite per mettere in sicurezza la struttura della propria rete informatica.

Da sempre, l'anonimato ha rappresentato una condizione fondamentale per la libertà di parola e di espressione, diritto importantissimo e riconosciuto da tutti i governi democratici. Esso differisce tuttavia dalla privacy. L'anonimato rappresenta la segretezza dell'identità delle parti coinvolte in una conversazione o nella trasmissione di dati di qualsiasi genere. La privacy, invece, riguarda la segretezza delle informazioni inviate durante una conversazione o una trasmissione. Tutelare l'anonimato significa consentire agli interlocutori coinvolti di comunicare senza rilevare la propria identità, indipendentemente dalla

[4] Violazione dei dati personali
[5] Invio di messaggi indesiderati attraverso la posta elettronica

segretezza dei contenuti. Grandi aziende e non solo cercano di estrapolare continuamente informazioni dal traffico generato, utili alla creazione di un modello di business. Rendere anonima una sorgente ha quindi l'obiettivo, oltre a cercare di aumentare la privacy, di inviare meno informazioni possibili sulla nostra identità.

1.2 MINACCE ALL'ANONIMATO

Per capire quali obiettivi si pongono gli Anonymity System[6], bisogna comprendere quali sono le tipologie di minacce che si possono presentare. Ad esempio in uno scenario di spionaggio industriale i motivi che spingono questi soggetti a controllarci sono molteplici: sapere cosa pensa l'utente, raccogliere delle statistiche, conoscere gli interessi dell'utente, al fine, per esempio, di proporgli della pubblicità mirata. Esistono aziende specializzate che sfruttano queste tecniche per registrare quali pagine sono state visitate e costruire un profilo degli interessi del navigatore al fine di proporgli della pubblicità personalizzata. Alternativamente un'azienda può usare l'analisi del traffico per monitorare quando un'azienda concorrente visita il suo sito e tracciare le pagine a cui è interessata. L'analisi del traffico è quindi una delle principali minacce all'anonimato. Le prime tecniche di analisi sono state sviluppate in ambito militare, le quali avevano lo scopo di intercettare gli ordini di battaglia al fine di capire eventuali mosse strategiche del nemico in anticipo. Al giorno d'oggi l'esempio più concreto riguarda le trasmissioni wireless. Essendo le informazioni inviate tramite l'etere, è possibile che un individuo riesca a recuperare informazioni semplicemente analizzando il traffico inviato in chiaro.

Ottenere queste informazioni, come la maggior parte delle cose, ha sempre un lato positivo e un lato negativo. Il rimedio principale contro queste tipologie di problemi è rappresentato dagli Anonymity System. Questi sistemi si suddividono principalmente in due categorie: *high-*

[6] Sistemi che garantiscono l'anonimato

latency e *low-latency*. I primi cercano di massimizzare l'anonimato, introducendo un notevole ritardo nelle comunicazioni. Il vantaggio di queste reti nel resistere agli attacchi, viene pagato al prezzo di non permettere l'utilizzo delle normali applicazioni interattive, come browser, chat o connessioni SSH[7]. I sistemi low-latency vengono invece caratterizzati dalla possibilità di utilizzare senza problemi i comuni programmi, riducendo tuttavia il grado di resistenza agli attacchi di analisi del traffico.

1.3 TOR PROJECT

Tor è un sistema di comunicazione che protegge gli utenti dall'analisi del traffico, aumentando così privacy e sicurezza della navigazione. Per fornire l'anonimato ai suoi utenti, il traffico viene instradato attraverso tre server scelti casualmente all'interno della rete (detti relay). La Figura 1-1 mostra come viaggiano i pacchetti attraverso la rete Tor. I router verdi rappresentano i relay collegati all'interno della rete tramite tunnel crittografici usando Onion Routing [**4**] [**5**] (instradamento a cipolla). Un client sceglie un percorso attraverso la rete e costruisce un circuito. Tor rende anonima l'origine del vostro traffico e cripta i dati all'interno della rete, ma non può criptare le comunicazioni tra la rete Tor e la sua destinazione finale, qualora si voglia raggiungere un host sulla normale rete Internet. [**6**] In questo caso, l'host di destinazione avrà come mittente l'ultimo nodo della rete Tor (exit relay). Ciò nonostante, se il destinatario dovesse essere all'interno della rete Tor, dal momento che i dati vengono cifrati dinamicamente, non c'è modo di conoscere l'origine dei pacchetti.

[7] Secure SHell

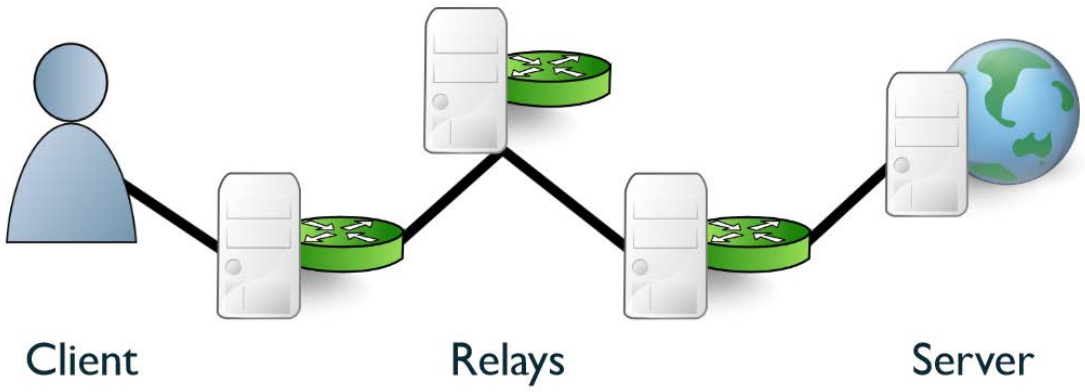


Figura 1-1: Modello client-server attraverso la rete Tor [**6**]

La rete fornisce essenzialmente due servizi:

- connessioni anonime in uscita
- fornitura di servizi nascosti

Come altri sistemi, Tor mira ad impedire ad un attaccante di collegare in qualche modo un mittente con un destinatario, oppure di collegare comunicazioni multiple provenienti o ricevute da un singolo utente. Può essere utilizzato per "anonimizzare" qualunque tipo di servizio, dalla semplice navigazione web alla posta elettronica passando per la messaggistica istantanea. L'utilizzo più gettonato risulta essere quello riguardo la navigazione web.

Attualmente Tor è utilizzato da più di 5 milioni di persone, (in Italia l'utilizzo è stimato a 200000 utenti, circa il 3,97% rispetto all'utilizzo totale). [**7**] Il futuro della rete passa attraverso l'usabilità e la sicurezza della stessa. Infatti se l'usabilità aumenta, attrarrà più utenti che aumenteranno le possibili sorgenti e destinazioni di connessione, incrementando così la sicurezza. "*Tuttavia gli interessi politici e tecnologici minacciano l'anonimato, minacciano la possibilità di leggere e parlare liberamente online. Come se non bastasse, questa situazione*

mette a rischio anche la sicurezza nazionale e le infrastrutture tecnologiche che al giorno d'oggi sorreggono tutto il sistema mondiale. [**8**]"

1.4 SIMULATORE SHADOW

Shadow è uno strumento accurato, efficiente e scalabile in grado di fornire risultati attendibili per la simulazione della rete Tor. [**9**] La maggior parte degli studi su Tor riguardano l'analisi di potenziali attacchi e l'implementazione di un nuovo design sempre più performante. Tuttavia effettuare sperimentazioni sulla rete reale provoca in primo luogo problemi riguardanti la privacy e in secondo luogo, i test sul design del software causano la modifica di centinaia di client e relay sparsi per il mondo il cui tempo di elaborazione non può essere trascurato. È necessario quindi un nuovo metodo per la sperimentazione. Uno degli approcci provati è stato quello di configurare una rete parallela privata, usando macchine fisiche, oppure utilizzare una piattaforma come *PlanetLab*. [**10**] Tuttavia le ricerche sia su macchine fisiche che su PlanetLab non riflettono perfettamente le condizioni effettive della rete pubblica di Tor; inoltre avendo a disposizione un numero limitato di macchine su cui operare non sarà possibile valutare correttamente la scalabilità delle modifiche, nonché problemi di gestione del sistema. Si è quindi pensato di creare un'ambiente di simulazione che dia la possibilità di effettuare i test riducendo tuttavia la precisione dei risultati. Il simulatore Shadow offre la possibilità di eseguire Tor su una singola macchina Linux senza necessitare dei permessi di Root, fornendo un'ambiente sicuro dove effettuare simulazioni con migliaia di nodi, come se fossimo sulla rete reale. Shadow è in grado di eseguire realmente l'applicazione Tor, modellare la topologia di rete, nonché la latenza e larghezza di banda, simulare nodi virtuali e usare la crittografia per cifrare i messaggi, come avviene nella realtà.

1.4.1 MODELLO DI SIMULAZIONE

Il primo passo per introdurre Shadow è creare un modello di simulazione per l'esperimento, ovvero una rappresentazione della realtà che vogliamo andare ad analizzare e successivamente a simulare. Nel nostro caso il file che rappresenta il modello, in formato standard XML[8], specifica quando Shadow deve creare i nodi virtuali e per ognuno di essi quale software eseguire. Vengono inoltre definite le informazioni sulla topologia di rete e le sue proprietà come la latenza e i tassi di perdita dei pacchetti.

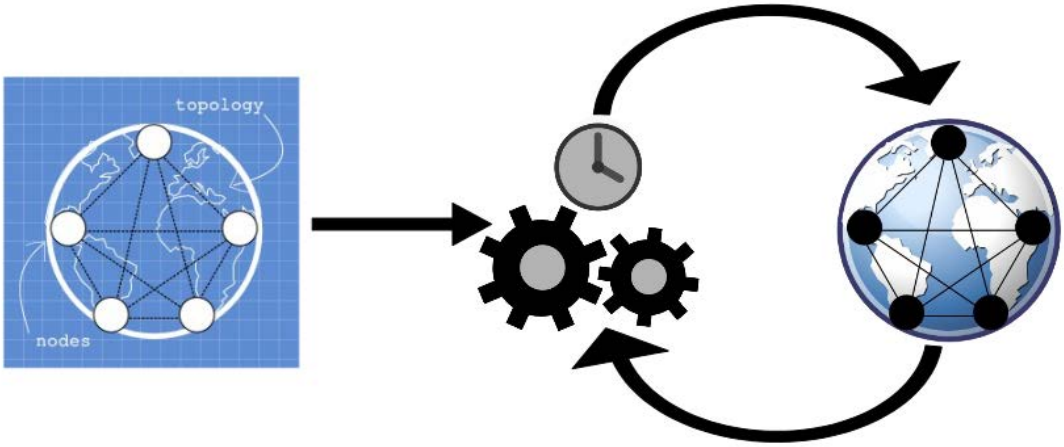


Figura 1-2: Schema del modello di simulazione in Shadow [**8**]

Il modello di simulazione in formato XML è necessario per l'esecuzione del simulatore. Nella Figura 1-2 è visibile come questo serva per la corretta configurazione del simulatore e dei suoi nodi, non solo durante la fase di inizializzazione ma per tutta la durata della simulazione.

1.4.2 SIMULAZIONE DI EVENTI DISCRETI

Con il termine simulazione intendiamo una tecnica algoritmica per effettuare esperimenti su sistemi che si evolvono nel tempo. La creazione di un modello di simulazione ci permettere di ottenere le

[8]EXtensible Markup Language

conoscenze sul sistema in analisi che possono portare ad un suo miglioramento. Inoltre grazie ai test effettuati, abbiamo la possibilità di valutare le prestazioni di tale sistema, ancora prima che questo venga messo in funzione. Le variabili di stato di un sistema, ovvero quelle variabili che corrispondono ad attributi o proprietà di tale sistema, definiscono in che modo può cambiare lo stato del sistema. Un evento provoca la modifica istantanea di una o più variabili di stato del sistema. [**11**] Shadow crea una serie di eventi dopo aver estratto le informazioni dal modello di configurazione XML. Ognuno di questi vengono eseguiti in un istante di tempo discreto durante l'esperimento; inoltre gli eventi causano l'esecuzione da parte del nodo virtuale del software specifico. Shadow tiene traccia del tempo che ogni nodo spende per l'elaborazione all'interno dell'applicazione e dei ritardi della Cpu virtuale. Nella Figura 1-3 è possibile avere una rappresentazione di come i pacchetti vengono inviati all'interno del simulatore. Poiché le applicazioni inviano dati tra loro, per trasferire i pacchetti interni di Shadow vengono trasferiti i puntatori tra le varie code. Questo processo implica l'uso di buffer per l'invio e la ricezione dei pacchetti tra i nodi virtuali e limitazioni per garantire ad ogni nodo la larghezza di banda desiderata.

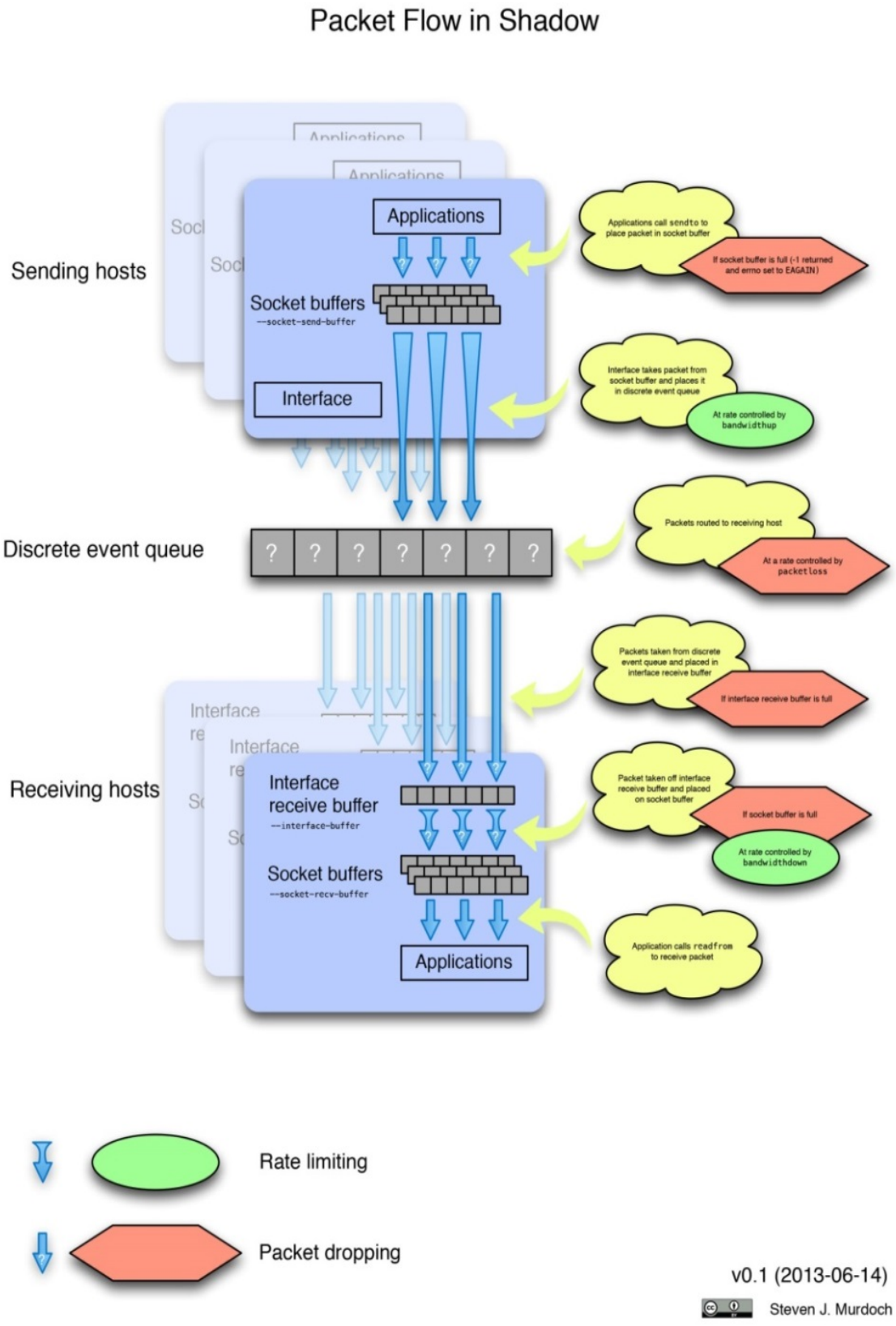


Figura 1-3: Scambio di pacchetti in Shadow (© Steven J. Murdoch)

1.4.3 GESTIONE DEI NODI VIRTUALI

Shadow esegue il software vero e proprio tramite i plug-in. Per eseguire più nodi virtuali deve tenere traccia dello stato di molte informazioni relative ai vari nodi contemporaneamente. Per supportare questa funzione, il motore del simulatore memorizza una copia di tutte le variabili di stato esistenti per ogni plug-in. In questo modo ogni volta che viene eseguito un evento per l'applicazione di un singolo nodo, avviene un "cambio di contesto". La Figura 1-4 ci mostra esattamente questo procedimento.

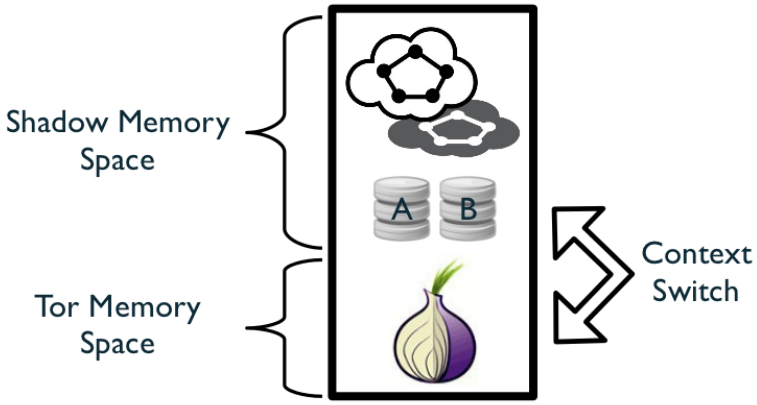


Figura 1-4: Gestione della memoria [**8**]

1.4.4 PLUG IN

I plug-in sono librerie che collegano il simulatore all'applicazione reale Tor. Shadow dinamicamente carica queste librerie per eseguire il codice ed effettua delle chiamate al sistema per consentire una perfetta integrazione dell'applicazione con l'ambiente simulato. In questo modo l'applicazione potrebbe non accorgersi che è in esecuzione nel simulatore, funzionando così come se fosse in un ambiente UNIX standard. Il simulatore, come abbiamo già detto, permette non solo la simulazione di Tor, ma anche di sistemi distribuiti o peer-to-peer, questo grazie ai vari plug-in che possono essere sviluppati ad hoc.

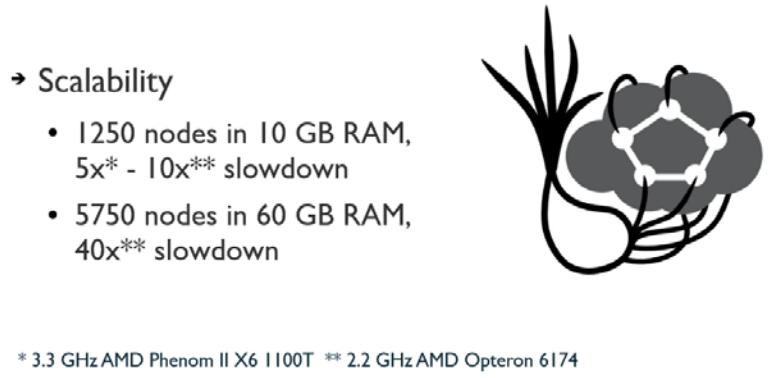


Figura 1-5: Scalabilità in Shadow [**8**]

Shadow mette a disposizione diverse librerie per effettuare le simulazioni; in particolar modo verranno analizzate le librerie "Scallion", che sono quelle utilizzate per la simulazione di Tor. "Scallion" integra Tor dentro Shadow, in modo tale da permettere la comunicazione tra le librerie e il simulatore, sfruttando la funzionalità unica di Shadow per consentire la prototipazione rapida[9] e la sperimentazione. Sono inoltre presenti degli script che generano topologie di rete per eseguire i test e aiutano ad analizzare i risultati ottenuti.

[9] Tecnica focalizzata a realizzare il cosiddetto prototipo

Capitolo 2

Il progetto Tor

Tor, attualmente giunto alla seconda generazione, è la porta per il cosiddetto *deep web*[10]. Si tratta di un mondo parallelo, irraggiungibile dai normali motori di ricerca, in cui chiunque può navigare in modo completamente anonimo o allestire un server la cui localizzazione risulta sconosciuta. Tali possibilità fanno si che molti criminali sfruttino queste caratteristiche per compiere azioni illecite o per scambiarsi materiale illegale. Tuttavia il progetto Tor è nato per tutt'altro scopo, ossia quello di salvaguardare la privacy degli utenti e dare la possibilità di superare le censure imposte in molti paesi. Infatti sempre più frequentemente l'utilizzo di questa tecnologia risulta l'unico modo per accedere a contenuti da paesi in cui esistono regimi dittatoriali che negano l'accesso completo alle informazioni su Internet, oppure per diffondere la propria opinione in totale sicurezza su temi di carattere delicato. La domanda che sorge spontanea è se l'utilizzo di Tor sia veramente sicuro. Dalle recenti indagini svolte dalla NSA[11], agenzia governativa americana che si occupa della sicurezza nazionale, sembra che la rete sia sotto parziale controllo, anche se ciò non è stato provato. [**12**]

[10] Web sommerso
[11] National Security Agency

Indichiamo ora le linee guida che caratterizzano il protocollo Tor:

- *deployability*: il design su cui si basa deve avere un deploy[12] semplice, facile da usare nel mondo reale. Per questo non deve richiedere risorse ingenti per essere eseguito.
- *usability*: l'esecuzione del programma non deve richiedere modifiche ai software installati o al sistema operativo. Non deve introdurre molto ritardo nelle operazioni e non deve essere difficile da configurare. Inoltre deve essere multipiattaforma[13].
- *flexibility*: il protocollo deve essere flessibile e ben strutturato, in modo tale da poter essere riutilizzato anche in futuro per fini diversi.
- *simple design:* il design e i parametri di sicurezza devono essere chiari e ben strutturati.

2.1 DESIGN

La rete che forma Tor è un *overlay network*, ovvero una rete formata da nodi (chiamati anche relay) che vengono eseguiti sopra la normale rete Internet. I due principali componenti della rete sono:

- *onion router* (OR): vengono eseguiti processi user-level senza bisogno di privilegi speciali. Inoltre ogni OR mantiene una connessione *Transport Layer Security* (TLS) verso tutti gli altri OR della rete.
- *onion proxy* (OP): vengono eseguiti dagli utenti sulla rete; sono in grado di creare un circuito e gestire le connessioni dalle applicazioni che l'utente intende utilizzare.

L'OP accetta connessioni *Transmission Control Protocol* (TCP) e provvede ad instradarle su di un circuito precedentemente creato. Ogni

[12] Messa in produzione
[13] Possibilità di essere eseguito su diversi sistemi operativi

OR che fa parte del circuito instrada i pacchetti, eseguendo operazioni di cifratura e decifratura fino alla destinazione. Inoltre ogni OR possiede due tipologie di chiavi: una *identity key* e una *onion key*. La prima è utilizzata per firmare i certificati TLS (protocolli crittografici che permettono di effettuare una comunicazione sicura) mentre la seconda viene usata nella fase iniziale per instaurare un circuito e per negoziare le chiavi di cifratura. Quest'ultime vengono generate periodicamente in modo da limitare il pericolo di compromettere la comunicazione. [**9**]

2.2 RICHIESTA DI CONNESSIONE

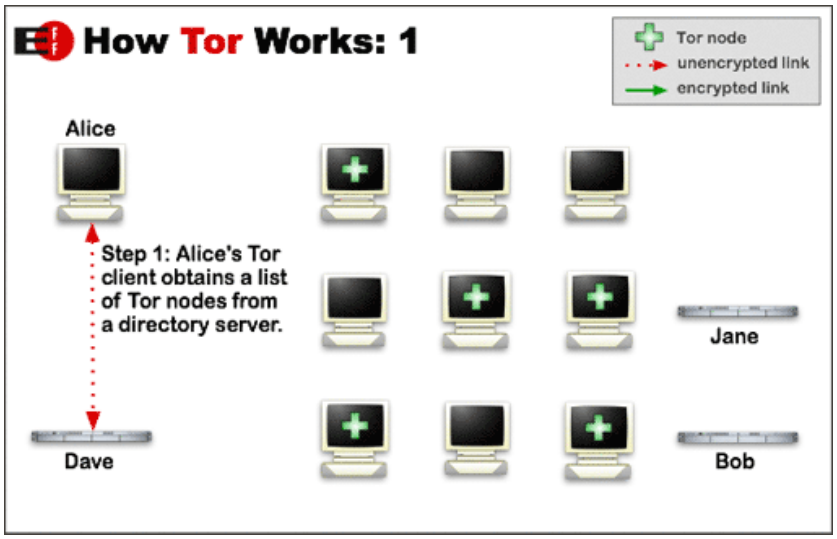


Figura 2-1: Richiesta di connessione a Tor [**6**]

I nodi con un simbolo + verde sono relay della rete Tor, *Jane* (J) e *Bob* (B) sono dei server pubblicamente raggiungibili da Internet mentre *Alice* (A) sta utilizzando Tor in modalità client. (Figura 2-1) D è un particolare server pubblico chiamato *Tor directory server*; lui e i suoi simili forniscono la lista pubblica di relay Tor e la mantengono periodicamente aggiornata. Il primo passo compiuto dal client Tor di A è, appunto, quello di contattare un directory server per ottenere la lista dei nodi marchiati con un simbolo + verde. A questo punto A contatta il server B, per esempio perché vuole visitare una pagina web ospitata li

sopra. Il nodo Tor di A si preoccupa di costruire una catena di nodi attraverso i quali invierà la richiesta, come mostrato nella Figura 2-2.

Nella terminologia di Tor, i nodi prendono un particolare nome a seconda della loro posizione nel circuito: il primo nodo si chiama *guard node* (nodo di guardia, GN), il secondo *middleman node* (nodo intermediario, MN) e il terzo e*xit node* (nodo di uscita, EN). La scelta dei nodi di una catena viene effettuata dal client di A in base alle informazioni ricevute dal directory server.

2.3 SCHEMA DI CONNESSIONE

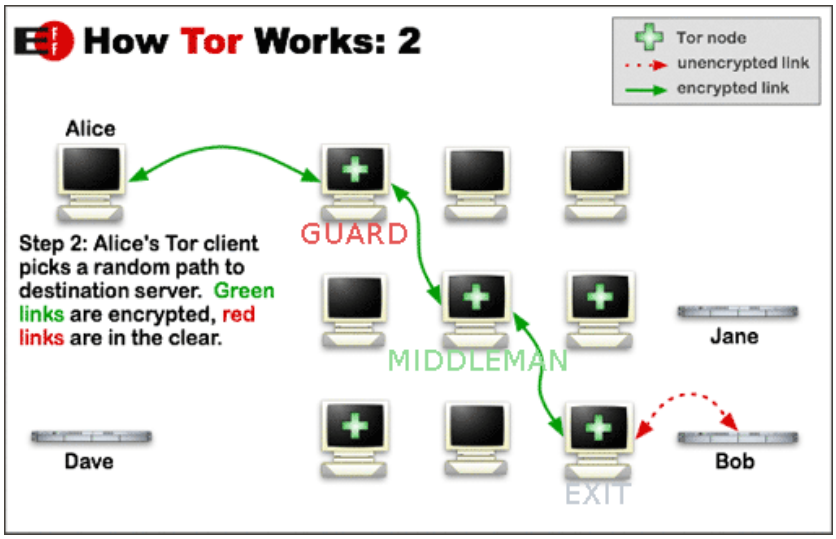


Figura 2-2: Creazione di un circuito all'interno della rete [**6**]

Ogni relay di Tor quando si registra presso i directory server invia il proprio indirizzo IP[14] e una serie di caratteristiche. I nodi di uscita sono i più importanti: contattano loro direttamente le macchine esterne alla rete Tor. Tuttavia, anche i nodi di guardia è necessario che siano sicuri, in quanto affidiamo loro il primo passo per entrare nella rete. Tutti i rimanenti relay sono considerati nodi intermediari.

[14] Internet Protocol address

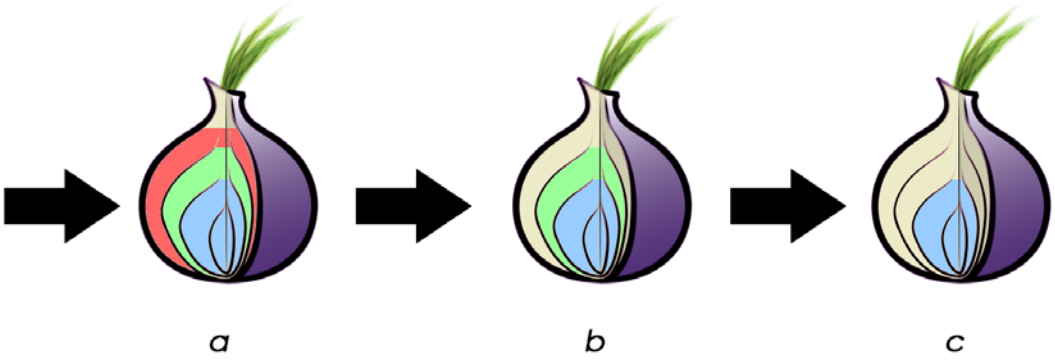


Figura 2-3: Cifratura e decifratura a “cipolla”

Il circuito criptato viene esteso un hop (salto) alla volta, e ogni relay lungo il percorso conosce solo quale relay gli ha dato le informazioni, e verso quale relay inoltrarle, grazie alla cifratura. Nessun relay conosce il percorso completo che il pacchetto ha preso. Per capire come mai questo protocollo è resistente agli attacchi di analisi del traffico è utile analizzare cosa accade nella Figura 2-3. Quando il pacchetto viene inviato da A a GN, il contenuto della buccia rossa è protetto con le chiavi di GN, quindi solo lui può leggere cosa c’è scritto. (Figura 2-3 a) Leggendo le istruzioni contenute in questo strato, GN capisce che deve inviare il resto del contenuto al MN. La Figura 2-3 b mostra il contenuto visto da MN; come nel caso precedente la buccia di colore verde è leggibile solo dal destinatario. In questo strato MN legge che deve spedire il resto del contenuto a EN. Come potete notare, già a questo livello è sparita ogni indicazione riguardante A. Questo è l’ultimo passaggio: EN riceve da MN quello che rimane della cipolla (Figura 2-3 c), legge il contenuto decifrabile solo da lui e scopre che deve fare richiesta di una pagina web presso il server B. Una volta che B ha trasmesso il contenuto necessario ad EN questo impacchetterà tutto e spedirà a MM il quale, vedendo che gli sta arrivando una risposta da EN, passerà a GN e questi chiuderà il circuito inviando la risposta ad A.

Il software negozia un nuovo insieme di chiavi crittografiche per ogni hop lungo il circuito, per assicurarsi che ciascun nodo non possa tracciare queste connessioni durante il passaggio. Per ragioni di efficienza, Tor utilizza lo stesso circuito per le connessioni che avvengono nell'arco di dieci minuti. Le richieste successive sono fornite a un nuovo circuito, per evitare che qualcuno possa collegare le azioni precedenti con le successive. Nella Figura 2-4 è possibile vedere questo funzionamento nel dettaglio. Un altro punto importante da osservare riguarda il colore delle frecce. La comunicazione che avviene all'interno della rete Tor è completamente cifrata, questo impedisce a un attaccante che riesca a intercettare uno qualsiasi di quei messaggi, di ricostruire tutto il flusso. L'ultimo collegamento (tra Exit e Bob) non è crittato ma questo dipende dal tipo di richiesta che ha effettuato Alice a monte del circuito: nel nostro esempio abbiamo ipotizzato una pagina web servita con HTTP, se Alice avesse usato HTTPS anche l'ultimo link sarebbe stato verde. Va ribadito, comunque, che l'ultima comunicazione è estranea alla trasmissione di informazioni all'interno del protocollo Tor.

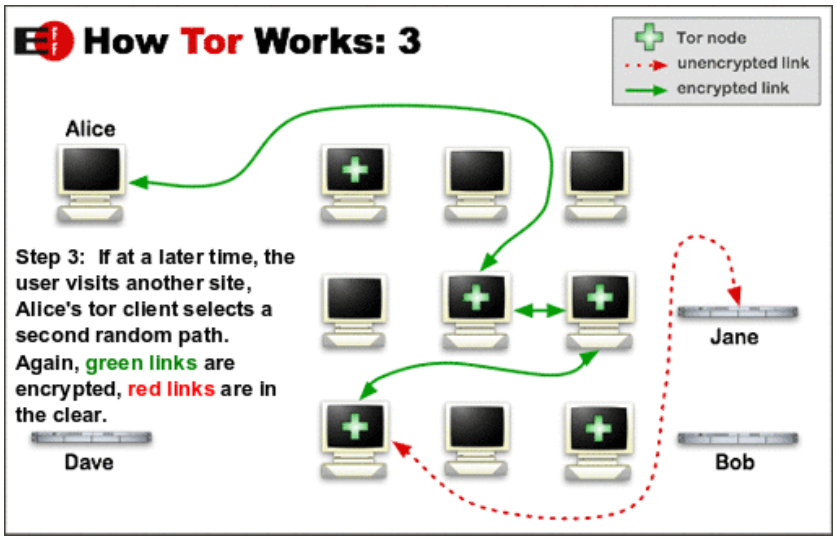


Figura 2-4: Nuovo circuito instaurato dallo stesso client [**6**]

2.4 STRUTTURA DEI PACCHETTI

Ogni onion router comunica con tutti gli altri attraverso connessioni sicure TLS. Il traffico che passa in questa connessione è formato da pacchetti (chiamati anche celle) di grandezza fissata. Ognuno ha una dimensione di 512 byte, ed è formato da:

- *header:* include un *circuit identifier* (CircID) che specifica a quale circuito il pacchetto appartiene e un *command* (CMD) che descrive cosa fare con il payload.
- *payload:* contiene i dati effettivi da inviare.

Basandoci sul CMD presente nel pacchetto distinguiamo due tipologie di pacchetti:

- *celle di controllo:* rappresentano i dati per un nodo intermedio, con il compito di ricevere il pacchetto dal suo predecessore ed instradarlo lungo il circuito.
- *relay cell*: rappresentano i dati per un exit relay dalla rete. Hanno il compito di smistare i dati in arrivo da diversi circuiti. Queste tipologie di pacchetti possiedono dei dati supplementari:
  - *streamID*: identificano il singolo stream visto che possono essere instradati diversi circuiti sullo stesso relay.
  - *digest:* esegue il controllo dell'integrità (checksum[15]).
  - *len*: indica la lunghezza del payload.

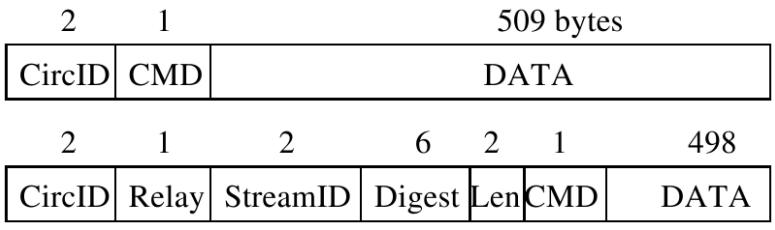


Figura 2-5: Struttura delle celle di controllo e relay cell [**9**]

[15] Sistema utilizzato per verificare l'integrità di un dato

2.5 CONNESSIONI ATTRAVERSO LA RETE

Le operazioni che possono essere eseguite dai vari relay sono molteplici, e ognuna presenta uno schema prefissato. Vediamo ora nel dettaglio ogni possibile sequenza.

2.5.1 COSTRUIRE UN CIRCUITO

Di seguito verrà spiegato come viene instaurato un nuovo circuito attraverso diversi OR. La Figura 2-6 mostra i vari passaggi. Un utente che vuole inizialmente creare un circuito costruisce in maniera incrementale dei collegamenti negoziando una chiave simmetrica con ogni altro OR sul circuito. [**9**]

a) Per iniziare a creare un nuovo circuito, l'OP invia una *create cell* al primo nodo (OR1) del circuito che ha scelto. Il payload della cella contiene la prima parte del Diffie-Hellman[16] handshake (g^{x1}) cifrato con la onion key dell'OR1.
b) OR1 risponde alla richiesta con una *created cell* contenente la seconda parte del Diffie-Hellman handshake (g^{y1}) e l'hash della chiave negoziata K1 = g^{x1y1}.
c) Una volta instaurato un circuito tra OP e OR1, queste due componenti sono predisposte per scambiarsi informazioni, utilizzando la chiave concordata; a questo punto bisogna espandere il circuito attraverso un *extended relay cell* inviato da OP a OR1 specificando l'indirizzo del prossimo OR (OR2) e la prima parte del Diffie-Hellman handshake g^{x2} cifrata con la onion key di OR2.
d) OR1 copia la prima metà dell'handshake[17] in una *create cell* e la inoltra a OR2.

[16] Protocollo crittografico che consente di definire una chiave condivisa e segreta tramite un canale di comunicazione insicuro. Tale chiave verrà successivamente utilizzata per cifrare la comunicazione tramite uno schema di crittografica simmetrica.
[17] Momento iniziale di verifica della connessione

e) OR2 risponde a OR1 come prima con una *created cell* contenente tutte le informazioni necessarie ad estendere il circuito.
f) OR1 inoltra le informazioni all'OP attraverso una *extended cell*.
g) Ora il circuito è stato esteso e procederà di questo passo fino a raggiungere l'ultimo nodo del circuito scelto dall'OP.

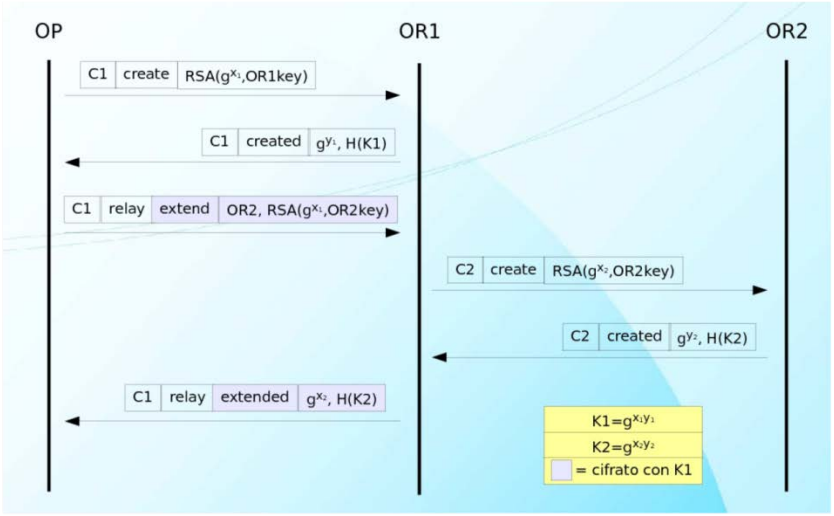


Figura 2-6: Creazione di un nuovo circuito [**6**]

2.5.2 INVIO E RICEZIONE DI UN PACCHETTO

Abbiamo appena descritto come avviene l'apertura di un nuovo circuito. Vediamo ora come vengono inviati i pacchetti sul circuito precedentemente creato tra mittente e destinatario. [**9**]

a) L'OP seleziona l'OR destinatario e calcola il *digest* (checksum) per quest'ultimo.
b) Iterativamente l'OP cifra il *payload* con la chiave simmetrica di ogni hop del circuito fino ad arrivare all'OR destinatario.
c) Ogni OR del circuito che riceve il pacchetto, decifra il *payload* con la propria chiave di sessione e verifica il *digest* della cella.
d) Se il *digest* corrisponde, significa che è lui il destinatario e quindi processa il pacchetto, altrimenti lo inoltra lungo il circuito al prossimo OR.

È stato mostrato come viene inviata una cella di relay sul circuito. Al momento della risposta le operazione che vengono eseguite sono l'esatto contrario di quelle effettuate nel momento dell'invio.

a) L'OR mittente cifra il *payload* e calcola il *digest* della cella.
b) Ogni OR sul circuito cifra il contenuto del *payload*.
c) L'OP che riceve il pacchetto, iterativamente, lo decifra con la chiave relativa ad ogni OR del circuito, dal più vicino al più lontano.
d) Ad ogni decifratura l'OP verifica il *digest* e se è corretto significa che la cella proviene dall'OR corrispondente all'ultima chiave utilizzata.

2.5.3 APERTURA E CHIUSURA DELLO STREAM

Quando un'applicazione vuole aprire una connessione TCP verso un determinato indirizzo ed una porta, effettua una richiesta all'OP (attraverso SOCKS). [**9**] L'OP sceglie l'ultimo circuito aperto oppure ne crea uno nuovo a seconda delle esigenze. Successivamente vengono effettuate le seguenti operazioni, illustrate anche nella Figura 2-7.

a) L'OP apre uno stream inviando un *relay begin cell* all'exit relay, usando un random *streamID*.
b) L'exit relay ricevuto il pacchetto provvede ad effettuare l'handshake TCP con l'host destinatario alla porta richiesta.
c) Successivamente l'exit node risponde con un *relay connected cell* facendo percorrere al pacchetto il percorso inverso nel circuito.
d) A questo punto l'OP invierà dati dall'applicazione inoltrandoli all'exit node attraverso delle *reley data cell*.

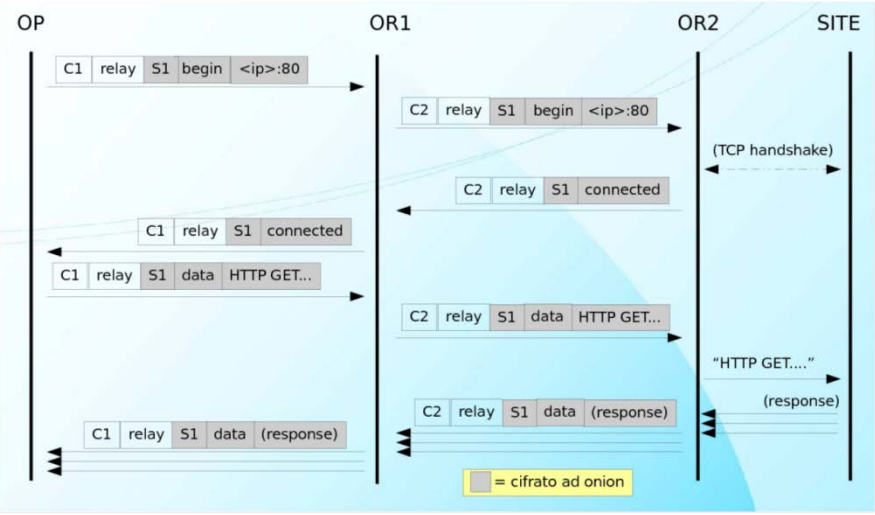


Figura 2-7: Apertura di uno stream sul circuito [**6**]

La chiusura di uno stream TCP viene effettuata in maniera analoga con la sola differenza che al posto di una *relay connect cell* e una *relay connected cell* viene inviata una *relay end cell* e una *relay ended cell*. Uno stream TCP può essere chiuso anche in maniera brusca, quindi l'exit node in tal caso sarà costretto ad inviare a ritroso nel circuito fino all'OP una *relay teardown cell* che segnalerà all'OP la chiusura inaspettata dello stream TCP.

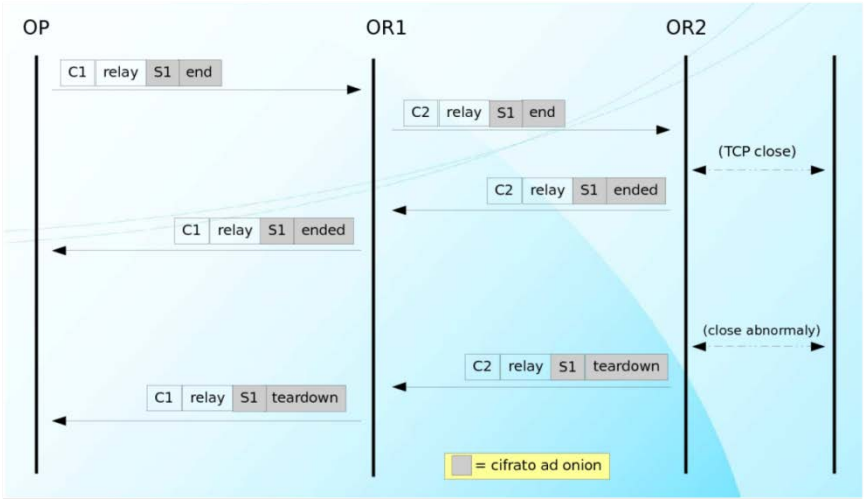


Figura 2-8: Chiudere uno stream sul circuito [**6**]

2.5.4 CHIUSURA DI UN CIRCUITO

La distruzione di un circuito avviene inviando una cella di controllo con il comando *destroy* e il numero del circuito che si vuole distruggere. Inizializzata dall'OP questa operazione con l'ID del circuito che lega l'OP e il primo OR, sarà successivamente eseguita a catena da ogni OR che provvederà ad inviare la medesima cella con il numero del circuito corrispondente al prossimo hop del circuito totale. Questo fino a raggiungere l'ultimo nodo del circuito e dichiarare chiuso il circuito. [**9**] Oltre ad essere distrutto, un circuito può essere anche troncato:

a) L'OP sceglie il punto in cui troncare il circuito, quindi seleziona un OR.
b) Calcola il digest della *relay truncate cell* che invierà sul circuito e iterativamente cifra il payload con le chiavi in ordine inverso degli OR che attraverserà la cella.
c) Ogni OR provvederà a decifrare il payload e a verificare il digest.
d) Nel caso quest'ultimo corrisponda allora l'OR provvederà ad eseguire un operazione di *destroy* per il resto del circuito (come descritta sopra) e inoltrerà all'OP una relay *truncated cell*, che sta a segnalare il buon fine dell'operazione di troncaggio.
e) Se il digest della cella non corrisponde allora l'OR inoltrerà al prossimo hop la cella.

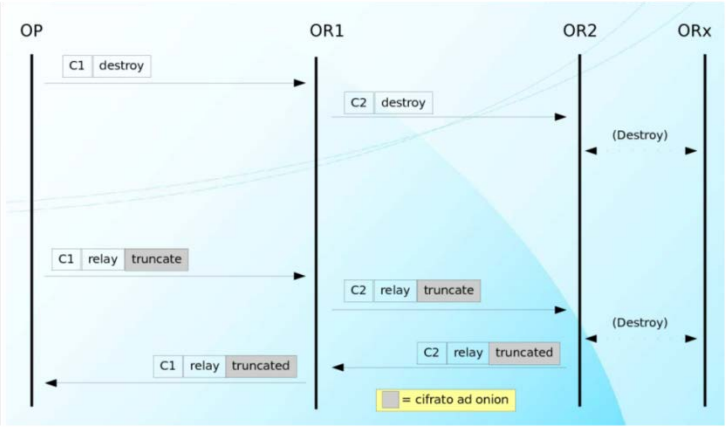


Figura 2-9: Modifica o distruzione di un circuito [**6**]

2.6 SERVIZI NASCOSTI

Tor consente agli utenti di nascondere la loro posizione quando offrono vari servizi, come pubblicazioni sul web o sistemi di messaggistica. Utilizzando i *rendezvous point* (punti di incontro) di Tor, gli altri utenti Tor possono connettersi a questi servizi nascosti, ciascuno senza conoscere l'identità di rete dell'altro. La funzionalità dei servizi nascosti permette agli utenti di Tor di creare un sito web in cui pubblicare materiale senza preoccuparsi della censura. Nessuno è in grado di determinare chi sta fornendo il sito, e nessuno che fornisca un sito può sapere chi sta scrivendo su di stesso.

Affinché i client possano contattarlo, un hidden service deve anzitutto rendere nota la sua esistenza nella rete Tor. Per questo il servizio sceglie alcuni relay a caso, stabilisce dei circuiti verso di essi e chiede loro di fungere da *introduction point* comunicandogli la sua chiave pubblica.

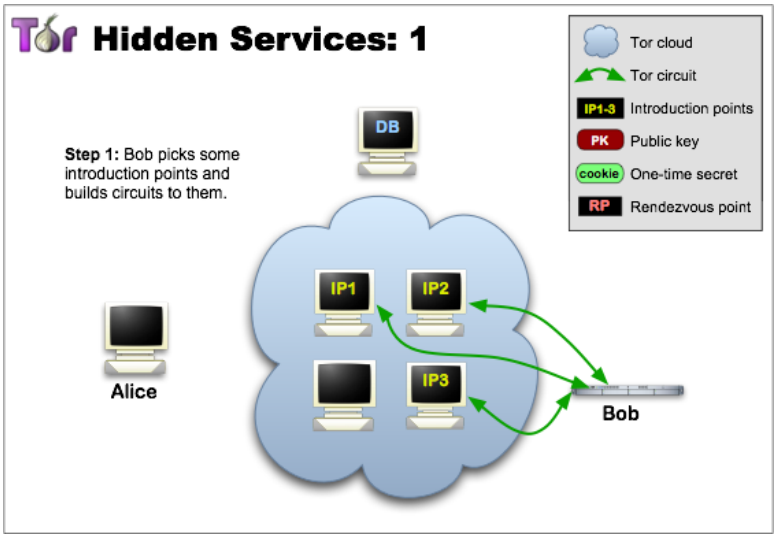


Figura 2-10: Creazione di un circuito con gli introduction point [**6**]

Secondo passo: l'hidden service costruisce un *hidden service descriptor*, contenente la sua chiave pubblica ed un sommario degli introduction point. Invia il descriptor a un gruppo di directory server, contenente il descriptor e l'indirizzo IP dell'hidden server. Il descriptor verrà trovato dai client che richiederanno XYZ.onion, dove XYZ è un nome di 16 caratteri. Dopo questo passo, l'hidden service è attivo.

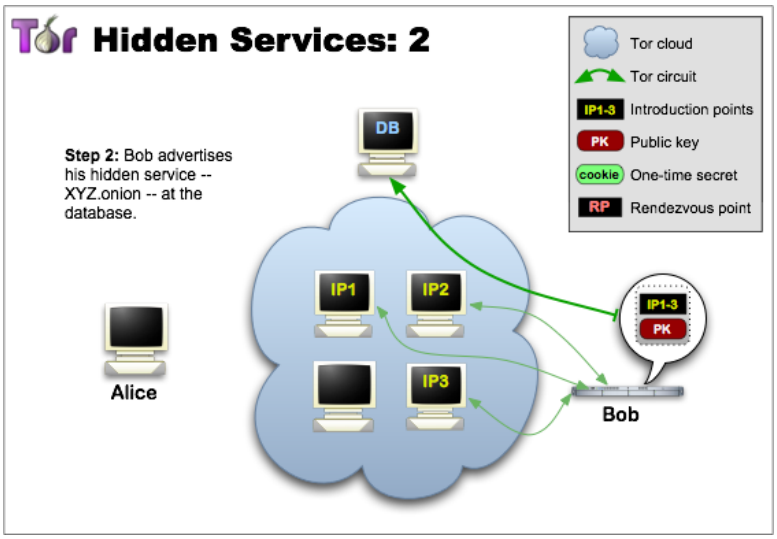


Figura 2-11: Attivazione del servizio [**6**]

Terzo passo: quando un client desidera contattare un hidden service, deve conoscere prima il suo indirizzo. Dopodiché il client può iniziare a stabilire la connessione scaricandone il descrittore dai directory server. Se esiste un descrittore per XYZ.onion presente nel database, il client viene a conoscenza del gruppo di introduction point e la corretta chiave pubblica dell'hidden service. In questo momento il client crea anche un circuito verso un altro relay scelto a caso e gli chiede di fungere da *rendezvous point*.

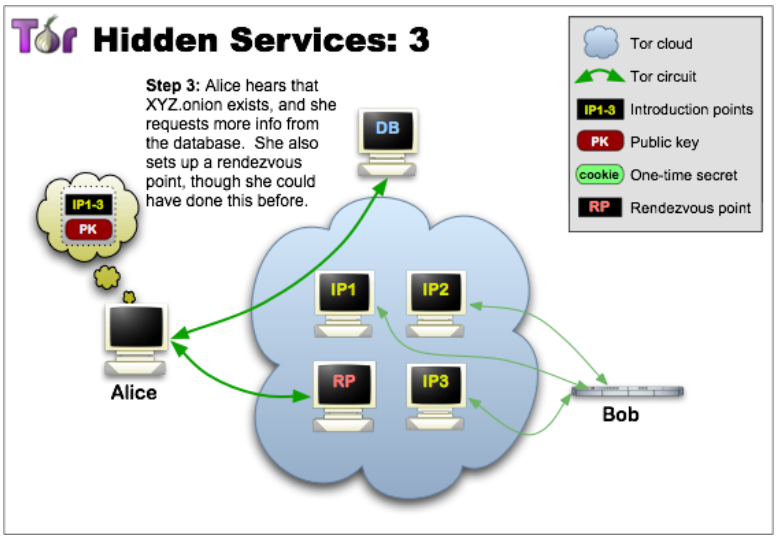


Figura 2-12: Collegamento al rendezvous point [**6**]

Quarto passaggio: una volta presente il descriptor e pronto il rendezvous point, il client costruisce un *introduce message* contenente l'indirizzo del rendezvous point. Il client invia questo messaggio a uno degli introduction point, chiedendo che venga consegnato all'hidden service. La comunicazione avviene sempre tramite un circuito Tor: in questo modo nessuno può collegare l'invio dell'introduce message all'indirizzo IP del client, ed il client rimane così anonimo.

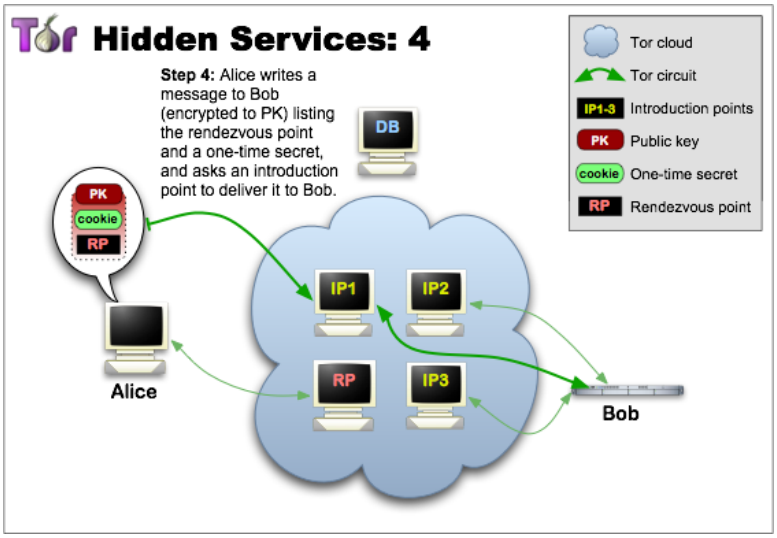


Figura 2-13: Creazione del circuito tra il client e l'hidden service tramite uno dei tre introduction point [**6**]

Quinto passaggio: l'hidden service decifra il messaggio del client e scopre l'indirizzo del rendezvous point. A questo punto l'hidden service crea un circuito verso il rendezvous point.

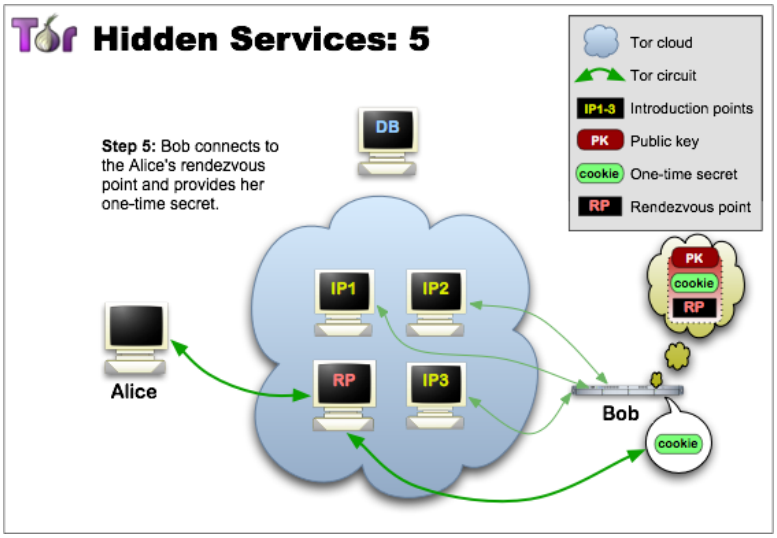


Figura 2-14: Collegamento diretto tramite il rendezvous point [**6**]

Nell'ultimo passaggio, il rendezvous point notifica al client che la connessione è stata stabilita con successo. Dopodiché il client e l'hidden service possono usare i loro circuiti verso il rendezvous point per comunicare tra di loro. Il rendezvous point inoltra semplicemente i messaggi cifrati dal client al service e viceversa. Uno dei motivi per non usare l'introduction point per le successive comunicazioni è che nessun singolo relay deve sembrare responsabile per un certo hidden servce. Ecco perché il rendezvous point non viene mai a conoscere l'identità dell'hidden service. In generale il collegamento completo tra client ed hidden service consiste in 6 relay: 3 di essi vengono scelti dal client, il terzo dei quali è il rendezvous point, gli altri 3 vengono scelti dall'hidden service.

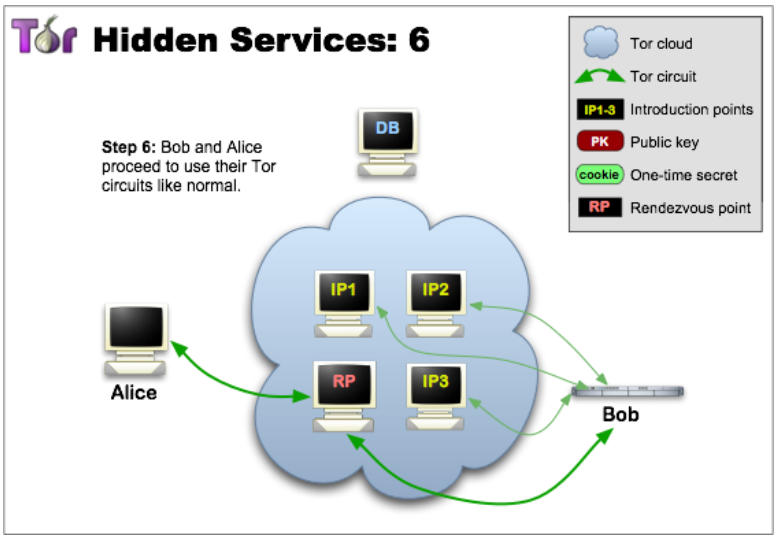


Figura 2-15: Schema finale di connessione [**6**]

Capitolo 3

Architettura del simulatore Shadow

Al fine di produrre risultati che siano consistenti ed accurati, utilizzando una rappresentazione il più accurata possibile della rete Tor si è cercato di modificare Shadow il meno possibile. Tuttavia Tor spende molte risorse per operazioni di buffering e per la crittografia dei dati; questo in un'ambiente simulato potrebbe portare a risultati sbagliati causati dall'elevato lavoro svolto per compiere queste operazioni e misurazioni inconsistenti. Per riprodurre le condizioni reali presenti all'interno della rete occorre inoltre determinare parametri quali latenza, affidabilità dei collegamenti, topologia di rete, larghezze di banda e velocità della CPU. Di seguito verranno descritti i principali componenti.

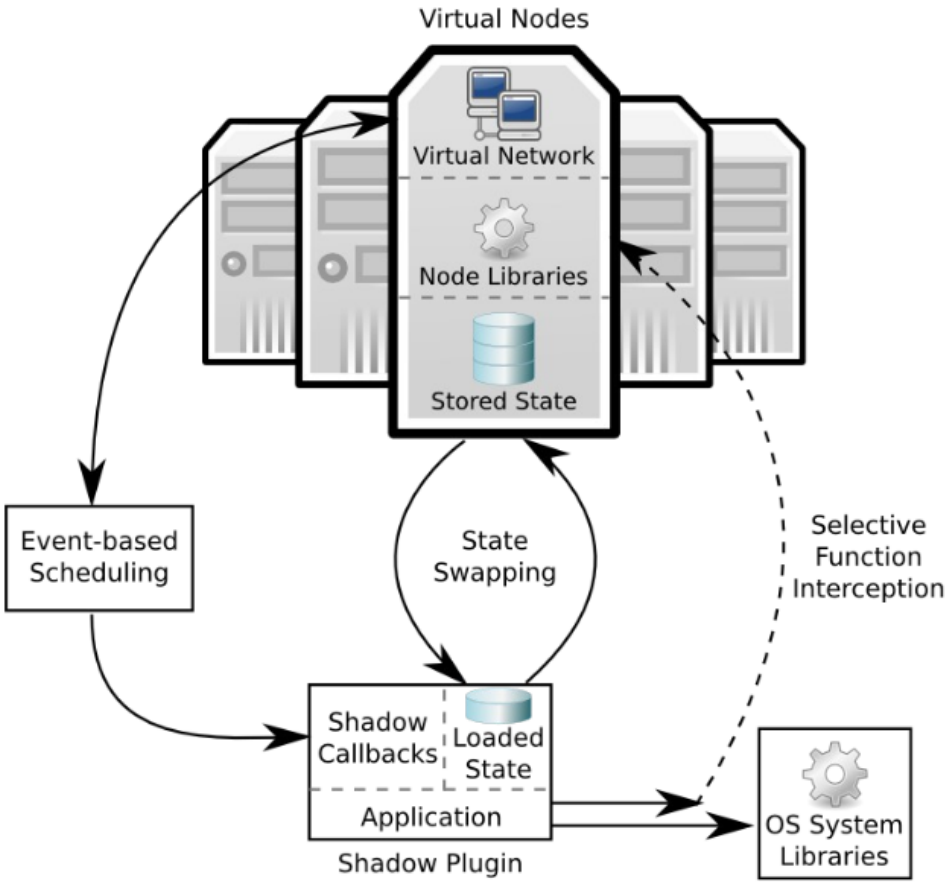


Figura 3-1: Architettura di Shadow [**8**]

3.1 SHADOW PLUG-IN

Shadow in origine era un simulatore *Distributed Virtual Network* (DVN) [**4**] a cui sono state aggiunte circa 18000 linee di codice. I plug-in, ovvero quelle librerie che consentono il collegamento tra il simulatore e l'applicazione reale, vengono caricate dinamicamente e istanziate all'interno di ogni nodo virtuale. Essi implementano le interfacce necessarie per la corretta comunicazione tra i vari componenti; nella Figura 3-2 sono rappresentati i principali componenti che costituiscono l'architettura del simulatore e come avviene la comunicazione.

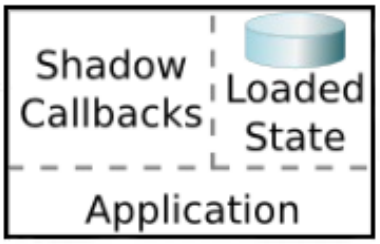


Figura 3-2: Shadow Plug-in

Per essere eseguita nel simulatore l'applicazione deve essere asincrona, ovvero deve usare chiamate di I/O non-bloccanti, in modo da prevenire il deadlock [**13**] durante l'esecuzione del codice. Le librerie utilizzate per gestire gli eventi asincroni sono le *libevent-2.0*. [**14**] Shadow deve essere eseguito in un singolo processo e singolo thread; sotto-processi o thread fork potrebbero provocare dei cambiamenti all'ambiente di simulazione e quindi produrre risultati errati. Il plug-in inoltre deve memorizzare tutte le variabili di stato dell'applicazione in Shadow per facilitare l'esecuzione della stessa da parte dei nodi virtuali; in particolare il simulatore riesce a soddisfare tale requisito passando i puntatori delle aree di memoria allo specifico nodo. Ogni variabile deve quindi essere globalmente visibile durante il processo di inizializzazione.

3.2 NODI VIRTUALI

All'interno del simulatore, i nodi virtuali rappresentano i singoli host simulati. Ognuno di essi contiene le informazioni necessarie per una corretta comunicazione di rete, come ad esempio l'indirizzo IP che consente di individuare univocamente il nodo all'interno della rete simulata. I nodi virtuali memorizzano lo stato dell'applicazione al loro interno; quando devono essere processati, viene effettuato prima un cambio di contesto per passare il controllo dell'esecuzione al plug-in.

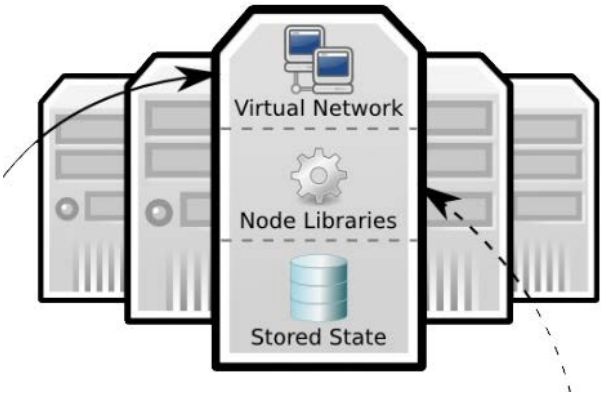


Figura 3-3: Nodo virtuale

3.2.1 VIRTUAL NETWORK

Per rete virtuale intendiamo l'interfaccia principale attraverso la quale i nodi possono comunicare tra di essi. Al momento della creazione di ogni nodo, viene assegnato loro un indirizzo e larghezza di banda in download e upload, come definito nel modello di simulazione. Algoritmi come *Token Bucket* vengono utilizzati per controllare la quantità di dati inviata all'interno della rete. Questi algoritmi effettuano un controllo sia per i pacchetti in entrata, sia per quelli in uscita. Inoltre sono in grado di bloccare la trasmissione nel caso in cui la quantità di informazioni immessa nella rete sia superiore al limite. Questo evento provocherà una ritrasmissione del pacchetto; diversamente i pacchetti vengono inviati attraverso il socket alle interfacce le quali si occupano di inviare i dati alle code di eventi discreti.

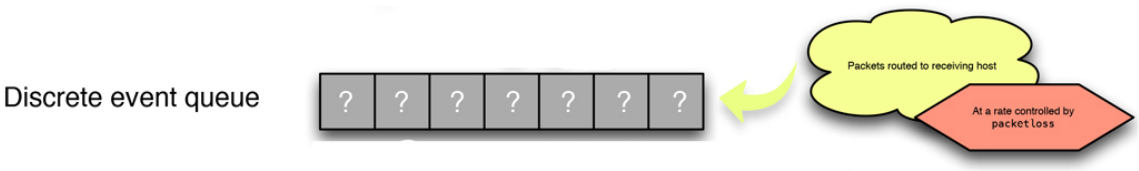


Figura 3-4: Coda degli eventi discreti

3.2.2 NODE LIBRARIES

Ogni nodo virtuale implementa le funzioni di sistema necessarie per gestire la rete, eventi, crittografia e molto altro. Le *function interposition* sono funzioni utilizzate per reindirizzare le chiamate del sistema e delle librerie allo specifico compartimento di Shadow. Queste funzionalità sono inserite nella libreria *ld_preload* che è la prima invocata ogni volta che avviene una chiamata ad una funzione.

Le librerie di sistema implementano delle chiamate che provocano la modifica dell'ambiente di simulazione. Esse sono configurate in modo tale da ritornare il nome dell'host attualmente in elaborazione, indirizzo nonché il tempo di simulazione. È possibile ottenere il tempo di elaborazione del modulo della Cpu virtuale considerando il ritardo prodotto dall'applicazione. Utilizzando questi due fattori, Cpu virtuale e ritardo di processing riusciamo ad accurare che ogni dato viene processato in un istante di tempo discreto nella simulazione. In particolar modo quando un nodo legge o scrive dati tra l'applicazione e Shadow, la Cpu virtuale produce un ritardo per processare i dati che è memorizzato come tempo impiegato per l'esecuzione di un evento di un nodo virtuale; gli eventi programmati dal singolo nodo vengono ritardati ed eseguiti successivamente. Ovviamente se ogni nodo legge o scrive una quantità di dati rilevante, il tempo di attesa per il prossimo evento aumenterà.

Il cuore del nodo virtuale è rappresentato dalle librerie del socket. Esse implementano tutte le funzionalità più importanti come la creazione, apertura e chiusura del socket, invio buffering e ricezione dei dati, nonché utilizzo dei protocolli di rete *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP). Shadow intercetta e reindirizza le funzioni tra le interfacce dei socket. Quando un'applicazione vuole inviare dei dati li incorpora in un pacchetto che viene inviato attraverso l'interfaccia. Ogni socket si occupa anche della ritrasmissione, di utilizzare metodi per il controllo della congestione, gestione degli *Acknowledge* (ACK) e *TCP Auto-Tuning*, una modalità che consente di

gestire il buffer in modo dinamico calcolando la dimensione in base alla velocità di connessione.

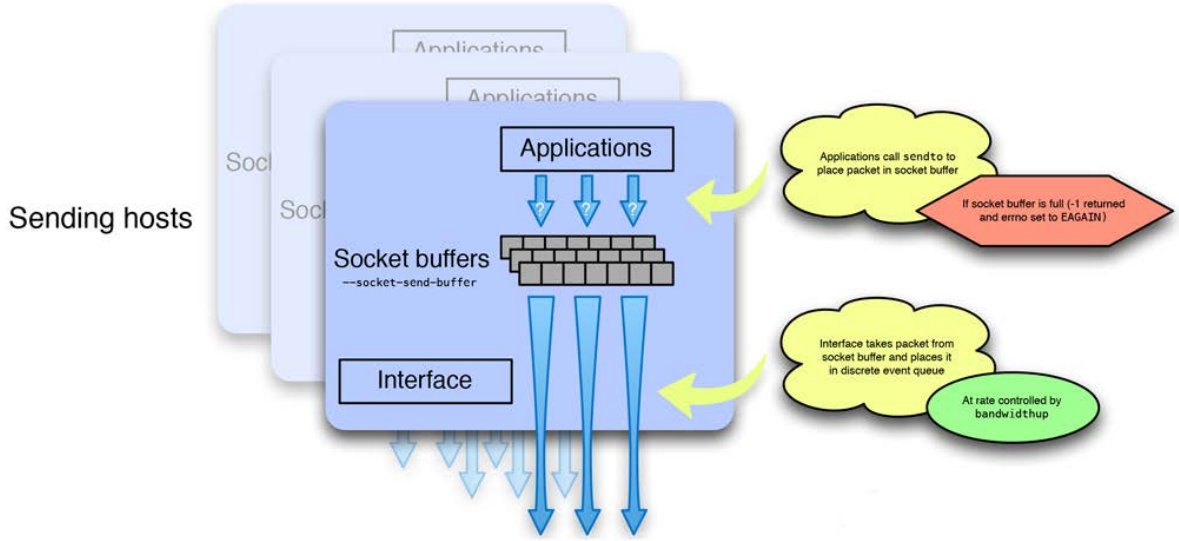


Figura 3-5: Host mittente

L'ambiente supporta l'uso di libevent-2.0 per facilitare l'asincronismo delle applicazioni. Mentre Tor non richiede l'uso di queste librerie, Shadow intercetta e reindirizza le funzioni tramite le interfacce di queste librerie.

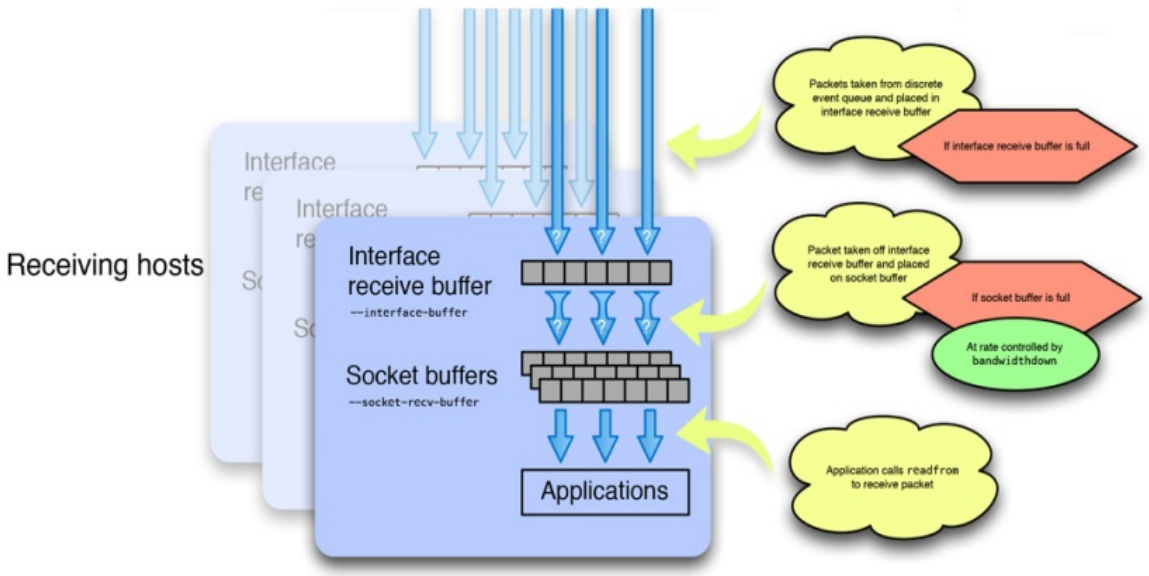


Figura 3-6: Host ricevente

Come abbiamo già detto, Tor spende molto tempo per operazioni riguardanti la crittografia dei dati. Simulare questi meccanismi causerebbe ritardi nel tempo di elaborazione e una possibile inconsistenza dei risultati. Poiché i dati vengono inviati tra i nodi all'interno del simulatore non è necessario che essi vengano criptati; tuttavia se vogliamo inviare informazioni direttamente su internet, si ha la necessità di utilizzare un metodo sicuro. Shadow permette di utilizzare i protocolli *SSL/TLS* sfruttando le librerie *OpenSSL*. [**15**] Queste funzioni eseguono solo operazioni di crittografia a basso livello, il che non provoca effetti sulle funzionalità dell'applicazione. Nel grafico è possibile comprendere l'influenza dell'uso della crittografia all'aumentare del numero di host.

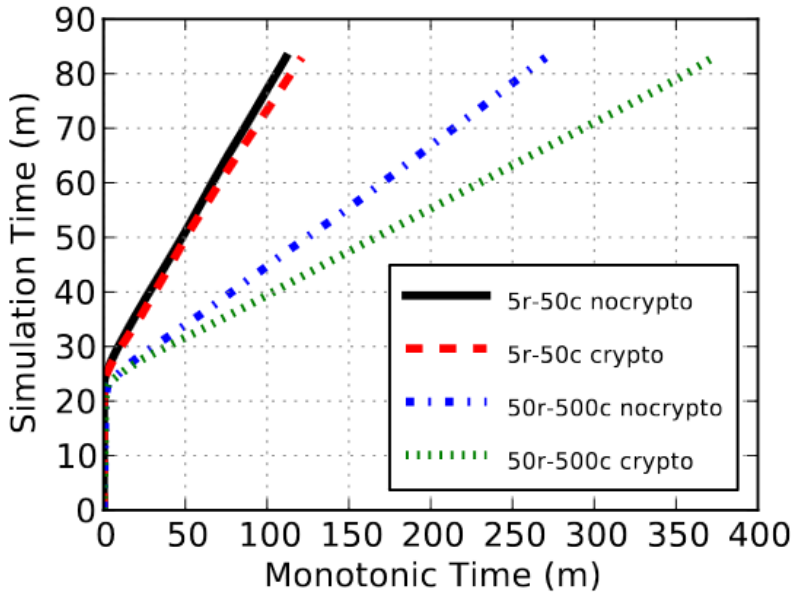


Grafico 3-1: Tempo di elaborazione per l'uso della crittografia [**16**]

3.2.3 STORED STATE

Multipli nodi virtuali possono eseguire lo stesso plug-in. Anziché duplicare l'intero plug-in in memoria per ogni nodo, Shadow duplicherà solo le variabili e lo stato dell'applicazione. In questo modo il simulatore determina le aree di memoria per ogni nodo virtuale, tenendo insieme quelle relative allo stesso plug-in, riducendo il consumo di memoria totale per ciascun plug-in. La memoria risparmiata può risultare significativa per simulazioni in cui consideriamo un numero elevato di host.

3.3 PLUG-IN SCALLION

Shadow è stato sviluppato su misura per simulare l'applicazione Tor sfruttando un plug-in denominato Scallion. Ogni nodo virtuale deve eseguire questo plug-in per simulare un host. Occorre precisare però che Tor è un'applicazione multi-threading, mentre Scallion implementa una versione un po' differente a causa di Shadow, che richiede solamente un singolo thread per l'esecuzione del codice dell'applicazione. All'interno della rete Tor, esistono una serie di script che vengono eseguiti in automatico e servono per determinare in modo dinamico parametri come la larghezza di banda. Scallion contiene un componente che approssima questa funzionalità sfruttando i dati della rete reale, tuttavia non è necessario utilizzarlo in quanto queste informazioni vengono definite all'interno del modello di simulazione e rimangono invariate per tutto il periodo della simulazione. Tor utilizza questo componente per generare dei report all'interno di ogni nodo. Ogni relay della rete Tor tiene traccia della sua larghezza di banda utilizzata recentemente e sfrutta questo valore per mantenere equità tra i vari nodi disponibili. I dati che si considerano si basano sulle quantità di informazioni trasferite di recente. Questi dati vengono aggiornati ogni venti minuti, solamente se non è avvenuto un cambiamento significativo. Ciò causa, utilizzando questa metodologia, che un nuovo relay appena aggiunto alla rete avrà

larghezza di banda pari a 0 per i primi venti minuti della simulazione, provocando scelte sbagliate su come instradare i dati tra i vari nodi. Per ovviare a questo problema durante la simulazione, come detto precedentemente, si è scelto di definire staticamente nel modello la larghezza di banda da utilizzare, senza considerare la quantità dei dati trasmessi come invece avviene nella realtà.

Sebbene Shadow minimizzi i requisiti di memoria necessari all'esecuzione, istanze complesse di Tor richiedono ancora quantità di memoria notevoli. Pertanto le simulazioni devono essere configurate in modo appropriato, con una dimensione che riesca a fornire risultati attendibili e con parametri paragonabili alla rete reale. Ad esempio, alcuni nodi con larghezza di banda limitata avranno un throughput[18] differente rispetto a nodi la cui banda risulta più elevata. Inoltre ripartire correttamente la banda tra i nodi risulta difficile. Il throughput dipende anche dal numero di client configurati e dalla quantità dei dati che si vuole inserire in rete. Quando si generano dei modelli di simulazione di una rete è essenziale che i parametri siano paragonabili a statistiche reali di Tor. Possiamo utilizzare degli script che ci definiscono il modello di simulazione, utilizzando i parametri presi da statistiche sulla rete reale. Lo script genera topologie scalabili, e migliora drasticamente l'usabilità del software all'interno del simulatore.

[18] Frequenza massima alla quale un nodo può smaltire dei dati

Capitolo 4

Analisi delle performance del simulatore Shadow

Gli aspetti riguardanti il design di Shadow sono stati discussi nei capitoli precedenti; in questo capitolo verranno esposte delle statistiche per verificare l'accuratezza del simulatore. Compareremo i risultati ottenuti in Shadow con quelli relativi ad un esperimento effettuato su PlanetLab e con quelli forniti dalla pubblicazione. [**16**] Il test su PlanetLab consiste nell'aprire connessioni http per trasferire dei file tra un client ed un server attraverso la rete. Ecco di seguito i parametri utilizzati: 361 client scaricano una serie di file da 20 server http, scegliendo un nuovo server in modo random per ogni nuovo download. 18 di 361 client richiedono dei file da 5MB, mentre i restanti 343 utilizzano dei file di 320KB. Larghezza di banda, latenza e velocità della CPU sono prese dalla rete reale.

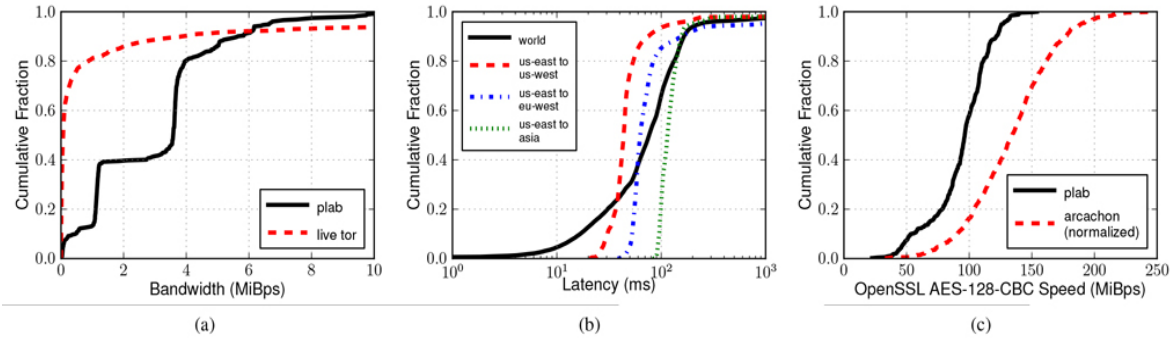


Grafico 4-1: Dati utilizzati per l'esperimento in shadow. (a) Larghezza di banda misurata su PlanetLab e rete Tor. (b) Latenza misurata su PlanetLab. (c) Cpu throughput all'interno di un nodo su PlanetLab [**16**]

4.1 SHADOW E PLANETLAB

Il Grafico 4-2 mostra i risultati riguardo PlanetLab. L'analisi si è concentrata principalmente su due aspetti: il *time to receive the first byte* (TTFB) ed il *time to complete* (DT). Con la prima metrica riusciamo a comprendere i ritardi associati all'invio di una richiesta che può essere passata a diversi nodi prima del raggiungimento della destinazione; ad esempio può rappresentare il tempo minimo che un utente deve attendere per visualizzare un'informazione sul browser. Le prestazioni generali sul tempo di download vengono invece studiate attraverso la seconda metrica. I grafici (c) ed (e) mostrano la metrica TTFB sia in PlanetLab che in Shadow. Il tempo finale di download di un file attraverso Tor è invece illustrato nei grafici (d) ed (f). Bisogna ricordare che effettuare il download di un file attraverso Tor richiede sicuramente più tempo rispetto alla rete internet, in quanto i dati devono essere processati e crittografati da più relay. Nel grafico (a) viene mostrano il numero di dati elaborati per ogni relay nell'esperimento di un'ora. La media è leggermente superiore per Shadow a causa della condivisione delle risorse. Nel grafico (b) è possibile vedere i tempi di attesa. In questo caso i valori misurati in Shadow e PlanetLab sono molto vicini.

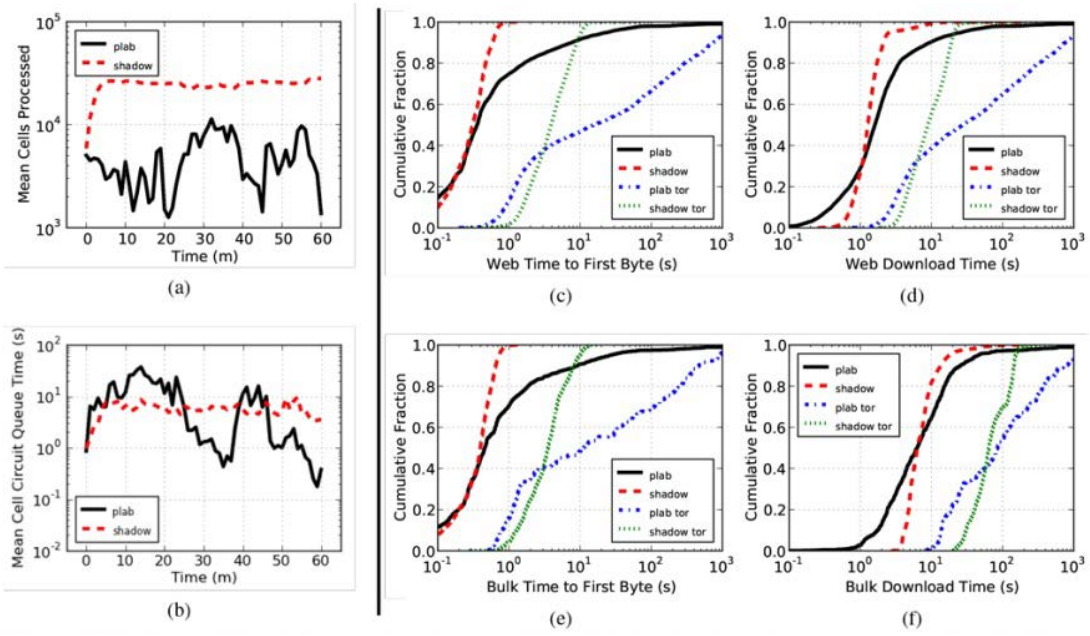


Grafico 4-2: Shadow e PlanetLab performance. (a) ed (b) evidenziano come Shadow approssimi le prestazioni di PlanetLab. (c) ed (e) mostrano il TTFB di un file di 320KB, mentre (d) ed (f) mostrano il DT di un file di dimensioni 5MB [**16**]

4.2 SHADOW E LIVE TOR

Anche se i risultati appena descritti mostrano come Shadow approssimi PlanetLab, essi non mostrano come esattamente Shadow può simulare correttamente la rete reale Tor. Inoltre attualmente la rete è formata da migliaia di relay e centinaia di migliaia di utenti distribuiti in tutto il mondo. Pertanto di seguito verranno esposti dei dati comparativi tra il simulatore e Tor per testare le potenzialità di Shadow di avvicinare le condizioni della rete Tor; le statistiche sulla rete sono documentante da The Tor Project [**7**] [**16**]. L'esperimento effettuato in Shadow è simile a quello analizzato precedentemente su PlanetLab, in cui dei client scaricano dei file da dei server all'interno della rete. I file utilizzati sono però di tre dimensioni, 50KB, 1MB e 5MB, mentre la configurazione della rete viene approssimata in base alle risorse disponibili in tempo reale su Tor. Sono stati usati 50 relay, 950 web client[19] , 50 bulk client[20]

[19]Client standard che scarica dei file dalla rete
[20] Client che scarica in modo continuato dei file per congestionare la rete

e 200 server. La larghezza di banda è stata impostata a 1MB in download e 3.5MB in upload. La latenza è configurata come in PlanetLab. Nel Grafico 3-1 è possibile vedere l'accuratezza di Shadow durante la simulazione. L'area grigia rappresenta il primo e terzo quartile e la linea tratteggiata indica il tempo medio di download. Si evidenzia che il tempo medio di download è praticamente identico per i due file da 50KB e 1MB, mentre è maggiore del 10% per i file di 5MB.

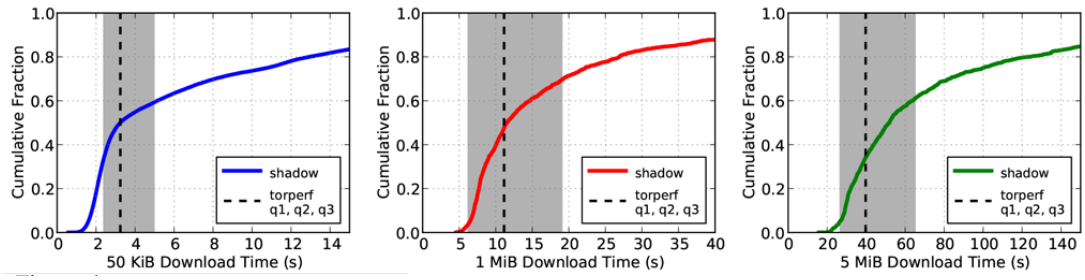


Grafico 4-3: Comparazione tra Shadow e la rete Tor [**16**]

4.3 SIMULAZIONE

In questo paragrafo verranno mostrati i risultati delle simulazioni effettuate. Sono state utilizzate due configurazioni per effettuare i test. Durante il primo studio, che chiameremo s*imulazione small* (Grafico 4-4) sono stati configurati 50 relay e 500 client che comunicano con 50 server tramite il protocollo HTTP. Anche in questo caso, abbiamo utilizzato file da 320 KB e da 5MB. Rispetto alla reale dimensione della rete Tor, questa configurazione è 50 volte più piccola. Quello che ci aspettiamo di visualizzare è che il TTFB dovrebbe essere indipendente dalla dimensione dei file. Nel secondo test, definito *simulazione large* visibile nel Grafico 4-5, abbiamo incrementato la portata configurando il simulatore con 100 relay e 1000 client che comunicano con 100 server tramite il protocollo HTTP. In questo caso la dimensione è di circa 25 volte più piccola rispetto alla rete reale. Ci aspettiamo che la metrica DT sia più accurata in questa simulazione rispetto alla prima.

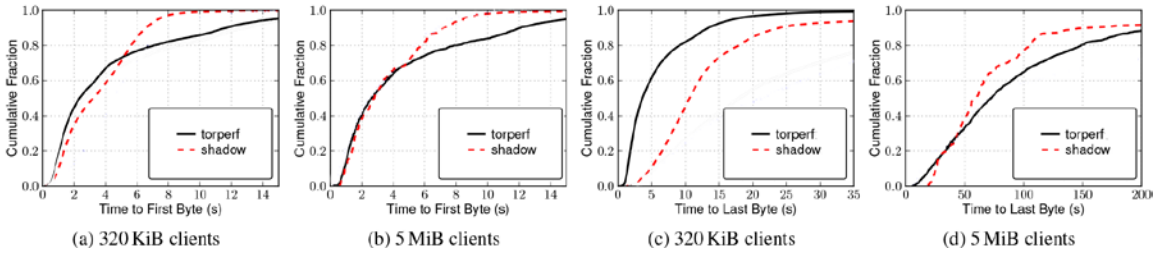


Grafico 4-4: Simulazione small. (a) ed (b) mostrano la metrica TTFB mentre (c) e (d) mostrano la metrica DT

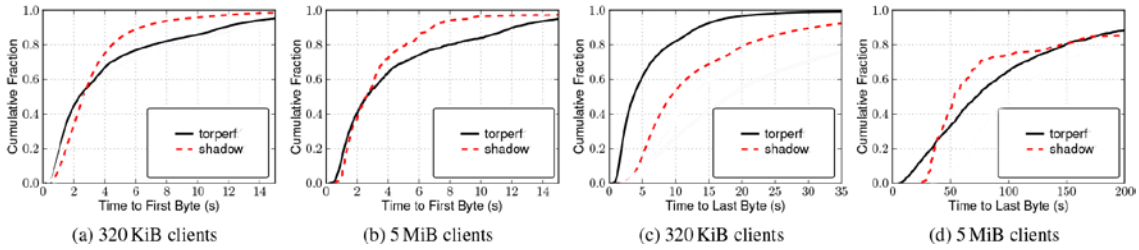


Grafico 4-5: Simulazione large. (a) ed (b) mostrano la metrica TTFB mentre (c) e (d) mostrano la metrica DT

Dai grafici è possibile vedere come Shadow approssimi ragionevolmente bene i risultati presi dalle statistiche sulla rete Tor. Grazie a questi modelli è possibile determinare e prevedere il comportamento della rete Tor. Questo modello di simulazione può fornire uno strumento valido in futuro per aumentare la scalabilità per studiare i possibili comportamenti che potrebbero verificarsi.

4.4 ALGORITMO DI SCHEDULING

Originariamente Tor era stato progettato utilizzando un algoritmo di scheduling Round Robin. Recentemente è stato progettato ed integrato nelle nuove versioni di Tor un nuovo algoritmo basato su *Exponentially Weighted Moving Average* (EWMA) [**17**]. In pratica di fronte alla scelta di due canali da privilegiare, l'EWMA sceglie quello con il minor numero di dati da inviare.

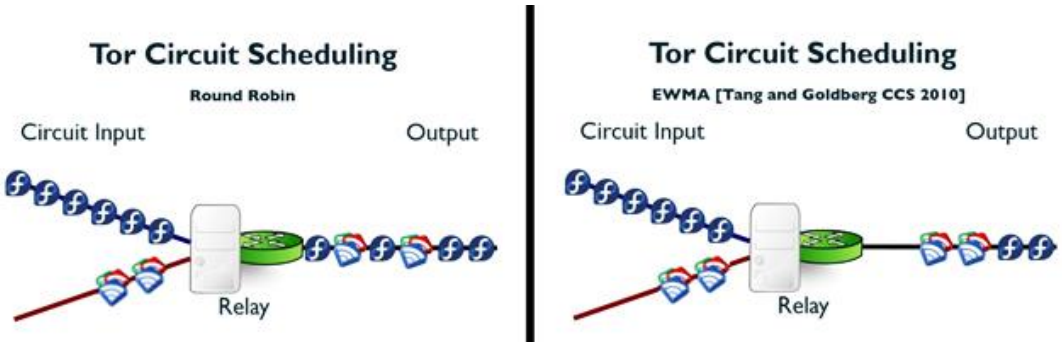


Figura 4-1: Algoritmi di scheduling a confronto

4.4.1 EWMA IN BOTTLENECK

Tuttavia occorre testare il comportamento dell'algoritmo in una topologia che presenta dei colli di bottiglia. Per validare l'EWMA è possibile costruire una rete congestionata anche in Shadow e vedere come si comporta. L'esperimento consiste nel creare un circuito con un unico ingresso, un nodo centrale ed un singolo relay di uscita. Due client scaricano continuamente dei file da 5MB per congestionare il circuito. Altri due client vengono incaricati dopo dieci minuti di avviare il download di un nuovo file all'interno della rete congestionata. Il tempo di attesa per l'avvio di questo download, secondo i dati rilasciati da *Tang e Goldberg* [**17**] è di 11 secondi. La struttura della rete appena descritta è visibile nella figura sottostante.

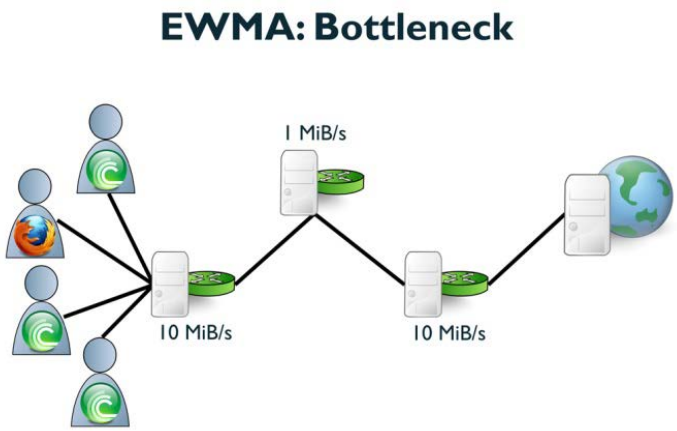


Figura 4-2: EWMA in bottleneck (collo di bottiglia)

Illustriamo ora delle statistiche sulla rete congestionata. L'obiettivo è quello di comparare il comportamento dei due algoritmi di scheduling. Le statistiche riguardanti il carico di lavoro dei relay sono riportati nel Grafico 4-6. Si nota il significativo aumento del traffico dopo dieci minuti. Il grafico (a) mostra il numero di celle elaborate che è simile per ogni relay. Il grafico (b) mostra come aumentano le code dei nodi di uscita perché devono elaborare le informazioni finali. I grafici (c) ed (d) espongono i dati relativi alle metriche TTFB che DT di un client web. Nei grafici sottosanti (e) ed (f) è possibile invece vedere le stesse metriche ma per un download di massa. Si noti come l'EWMA riesca a dare una risposta più rapida rispetto al round-robin utilizzato in predenza. Occorre tuttavia notare che l'utilizzo dell'EWMA per un download di massa, come è possibile vedere nel grafico DT (f), incrementa il tempo di attesa di un 40%. Questo aumento può provocare un ritardo notevole.

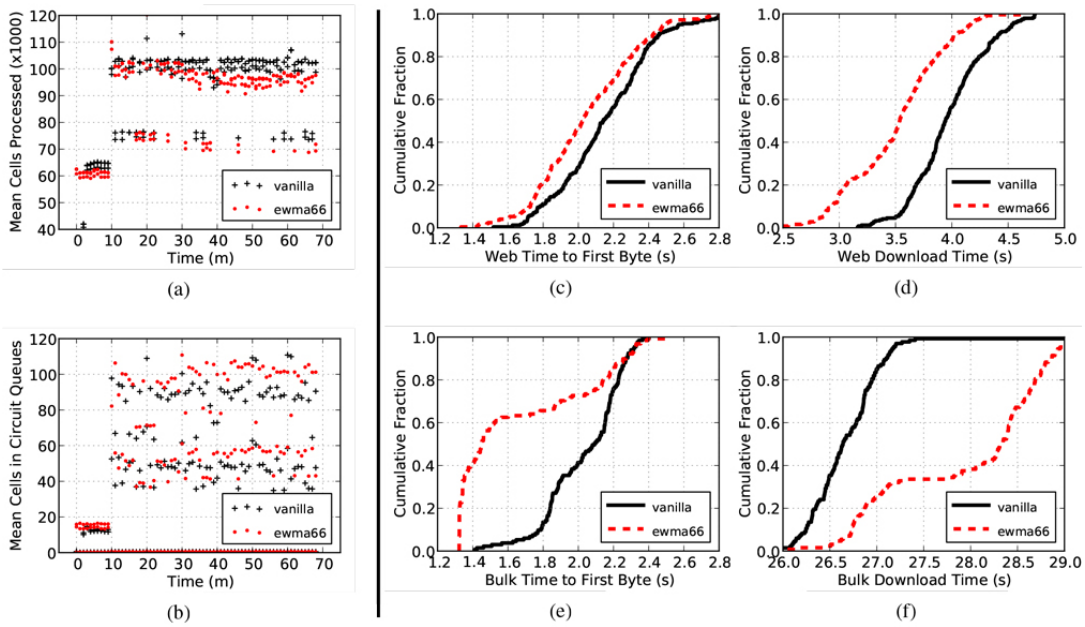


Grafico 4-6: Performance in una rete congestionata. Il numero di celle elaborate (a) e in coda (b) aumenta dopo 10 minuti. l'EWMA migliora la reattività per il traffico normale (c), (d) ed (e) ma contrariamente alle ipotesi diminuisce le prestazioni per i download di massa (f) [**16**]

4.4.2 EWMA IN SHADOW

Analizziamo ora i risultati ottenuti dallo scheduler EWMA attraverso l'intera rete di Shadow. Sono stati analizzati tre casi, in cui si analizzano 25, 50 e 100 client presi in modo random all'interno della rete. Per ridurre il tasso di errore ogni esperimento è stato effettuato cinque volte. I Grafico 4-7 (a)-(c) mostrano che lo scheduler EWMA, per una porzione di 25 client, riduce le prestazioni rispetto allo scheduler di default utilizzato in Tor. Con l'aumento del carico per 50 client (d)-(f), le prestazioni non migliorano significativamente. Gli unici miglioramenti sono ottenuti grazie due configurazioni dell'EWMA in (d) ed (e). Le prestazioni in un dowload di massa (f) con l'EWMA non riscontrano miglioramenti. Infine nei grafici (g)-(i) analizziamo il traffico con 100 client. Sotto pesante carico l'EWMA sembra comportarsi decisamente meglio rispetto a prima. Tuttavia per i download di massa non si riscontrano miglioramenti rispetto al normale scheduler di Tor.

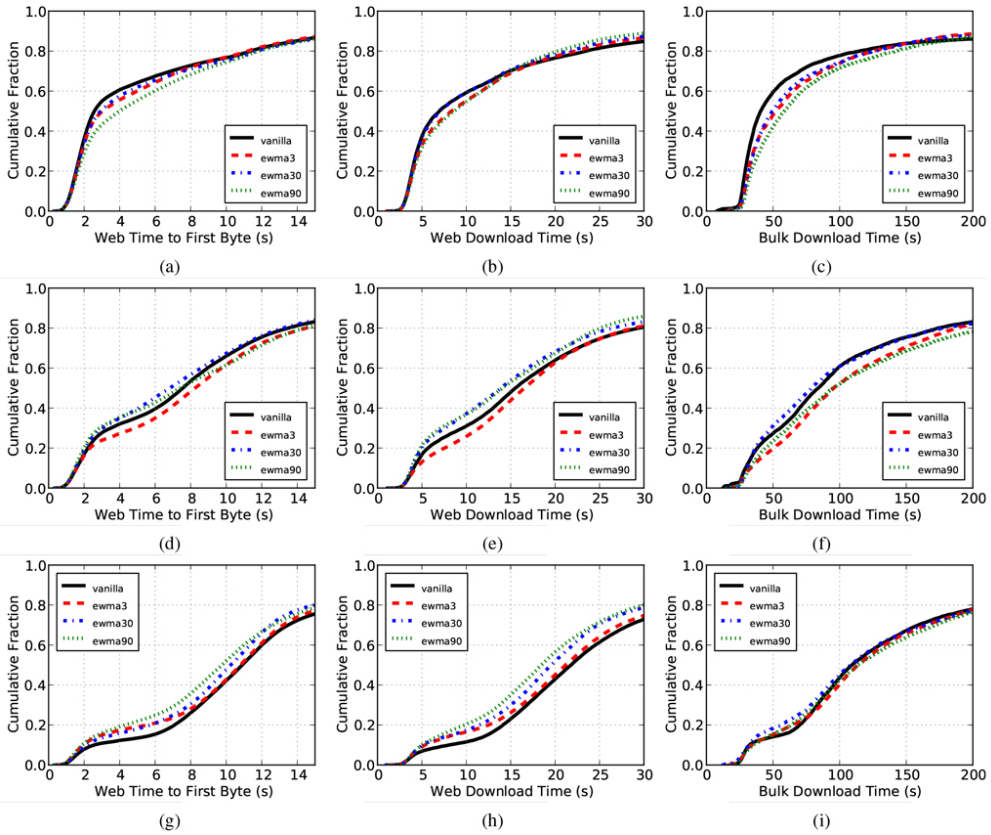
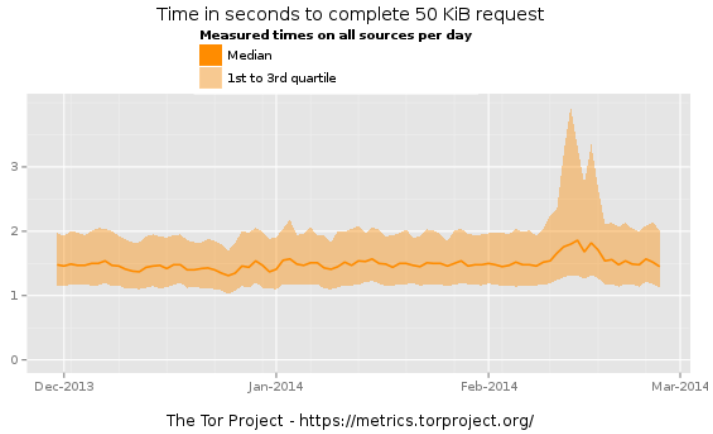


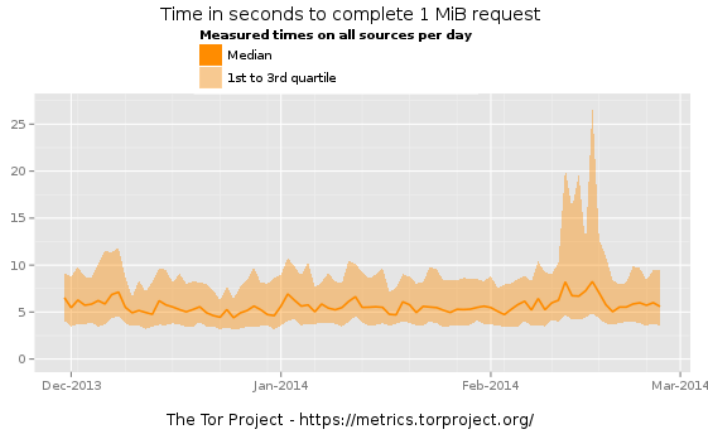
Grafico 4-7: EWMA e Vanilla in Tor. Il carico di lavoro è generato da 25 client scelti casualmente (a)-(c), 50 client (d)-(f) e 100 client (g)-(i) [**16**]

In conclusione si ritiene che non sia necessario utilizzare lo scheduler EWMA in quanto non è chiaro come le prestazioni possano migliorare. Sicuramente, se ci dovessero essere miglioramenti, questi potrebbero avvenire solo in una rete con un numero elevato di client. Tuttavia risultano invariati dei miglioramenti per quanto riguarda il download di massa.

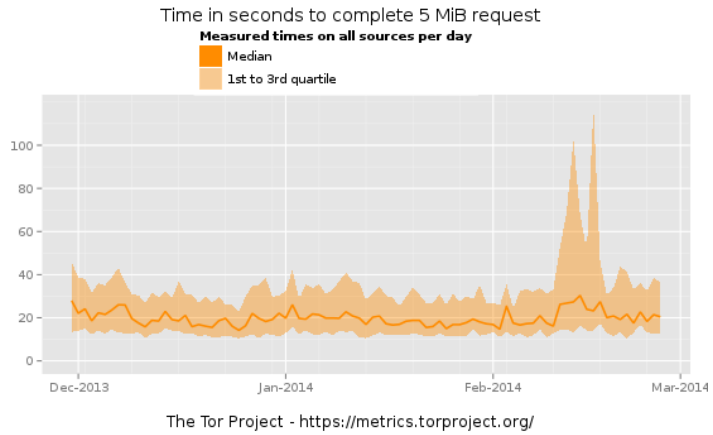
I grafici seguenti mostrano il tempo medio di download di tre file di diverse dimensioni. Mediane e quartili sono calcolati più volte al giorno. I grafici vengono aggiornati ogni 24-36 ore, non è quindi possibile visualizzare i dati in tempo reale, ma bisogna attendere l'aggiornamento relativo al giorno precedente.



(a)



(b)



(c)

Grafico 4-8: Tempo di download di tre file sulla rete Tor [7]

Conclusioni

Nel corso dell'ultimo decennio si è verificato un incredibile sviluppo tecnologico. Abbiamo assistito alla nascita e lo sviluppo del Web 2.0 e grazie all'avvento sul mercato di nuovi dispositivi come smartphone e tablet, il numero degli utenti che utilizzano Internet è cresciuto notevolmente. Aggiungendo a queste nuove tecnologie un sostanziale cambiamento nei modi di fare dovuto all'evoluzione del sistema sociale, troviamo che attualmente gli utenti connessi sono più di 2 miliardi. L'uso di social network, posta elettronica, chat e portali web fanno si che tantissime informazioni ogni secondo vengano instradate tra i vari router.

La grande maggioranza degli utenti, e questo non solo in ambito informatico, utilizzano un servizio o una tecnologia senza però capirne il funzionamento. Uno degli obiettivi di questa tesi è quello di spiegare il perché oggigiorno vengono utilizzati dei sistemi come Tor in grado di proteggere i nostri dati. Un normale utilizzatore di Internet non si preoccupa che i propri dati possano essere letti e utilizzati da persone che non hanno il diritto di farlo. Sempre più spesso inoltre grandi società si arricchiscono tramite informazioni estrapolate dalle nostre connessioni, il tutto a nostra insaputa.

Tuttavia i recenti eventi che riguardano la sicurezza informatica e la sicurezza dei dati personali, hanno acceso il focus su questi temi. Negli ultimi anni l'incremento degli utilizzatori di Anonymity System come Tor è cresciuto del 40%. Sembra quasi un paradosso, ma tutt'oggi troviamo difficile in determinate parti del mondo esporre la propria opinione in totale sicurezza e sincerità. Ecco perché, insieme a chi vuole salvaguardare la propria privacy, l'utilizzo di queste tecnologie risulta molto importante.

Occorre però constatare, che in un "mondo" in cui non sono presenti controlli e in cui chiunque può dire o pubblicare qualunque informazione, vi è la possibilità di imbattersi in criminali e malintenzionati. Proprio per la natura stessa della tecnologia, sempre più spesso la criminalità si nasconde dietro la rete Tor.

Ecco le motivazioni che mi hanno spinto a scrivere questa tesi, ed analizzare di conseguenza sia il funzionamento di Tor che quello di Shadow, un ottimo strumento con cui effettuare ricerche per lo sviluppo futuro della rete Tor. Trovo di fondamentale importanza avere la possibilità di utilizzare il simulatore come ambiente in cui fare ricerca, soprattutto perché effettuare sperimentazioni direttamente sulla rete risulta molto pericoloso. Ciò nonostante, grazie alla struttura stessa di Shadow, possiamo utilizzarlo non solo per la simulazione di una rete Tor, ma anche in altri ambiti. Questa strada potrebbe portare ad estendere l'utilizzo di questo software anche in altri ambiti di ricerca.

Appendice A

Installazione di Shadow

Shadow funziona principalmente in due modalità:

- installazione su macchina UNIX
- esecuzione su istanza del server Amazon EC2[21]

Il vantaggio di eseguire il codice su un server Amazon già configurato, ci sgrava dalla verifica delle librerie necessarie e software ausiliari che dovremmo installare manualmente qualora volessimo avere una versione locale sulla nostra macchina. Ovviamente è preferibile avere una versione del simulatore installata localmente, anche se la simulazione di topologie complesse richiede risorse hardware gravose.

SHADOW SU AMAZON EC2 SERVER

Amazon offre servizi di cloud computing ad aziende e privati con risorse potenzialmente illimitate. Per effettuare esperimenti su larga scala è possibile collegarsi ad un'istanza basata sulla distribuzione Linux Ubuntu-12.04 LTS in cui è già preinstallato e configurato Shadow.

[21]Amazon Elastic Compute Cloud

Ovviamente il servizio è a pagamento (le tariffe variano in base alle diverse configurazioni hardware che si richiedono).

Per eseguire Tor su EC2 in pochi minuti bisognerà:

1. fare login su Amazon EC2
2. configurare una versione di Shadow-cloud AMI basata su Linux Ubuntu-12.04 LTS
3. selezionare il tipo di modello (plug-in) su cui si vuole basare la simulazione
4. impostare un nuovo keypair, nel caso non sia presente, per il collegamento SSH

Quando avremo concluso questi quattro passi avremo un'istanza attiva a cui dovremmo collegarci tramite SSH.

```
1       ssh -i your-key.pem ubuntu@your-public-dns.amazonaws.com
```

SHADOW SU MACCHINA LINUX

Il software è stato installato su una macchina con un sistema operativo Linux Ubuntu 12.04-LTS, processore AMD Athlon™ 2 P320 Dual-Core, 3 Gb Ram DDR2, Scheda Video AMD Radeon HD 4000, Hard Disk 320 GB Sata2.

DIPENDENZE

Shadow richiede software e librerie specifiche per un corretto funzionamento. È opportuno verificare le versioni dei software indicate tra le parentesi al fine di evitare errori di configurazione.

Lista dei software richiesti:

- clang, llvm (3.2)
- glib (>= 2.32.0)
- cmake (>= 2.8.8)
- make

- python (= 2.7)
- xz-utils
- gcc (scallion plug-in only)
- automake (scallion plug-in only)
- autoconf (scallion plug-in only)
- zlib (scallion plug-in only)
- libtidy (browser plug-in only)

Per effettuare l'installazione dei programmi necessari è possibile eseguire il seguente comando.

```
1    sudo apt-get -y install gcc xz-utils make automake autoconf cmake
     zlib1g-dev zlib1g-dev tidy libtidy-dev libglib2.0 libglib2.0-dev
     dstat pdftk python2.7 python-matplotlib python-numpy python-
     scipy htop screen libxml2-dev libxslt-dev
```

Tuttavia sono raccomandati anche i seguenti software:

- htop
- screen
- dstat
- numpy
- scipy
- matplotlib
- pdftk

Per l'installazione di clang/llvm è necessario procedere manualmente, in quanto i pacchetti installati automaticamente del sistema operativo non includono dei moduli necessari al simulatore.

```
1    wget http://www.llvm.org/releases/3.2/llvm-3.2.src.tar.gz
2    wget http://www.llvm.org/releases/3.2/clang-3.2.src.tar.gz
3    tar xaf llvm-3.2.src.tar.gz
4    tar xaf clang-3.2.src.tar.gz
5    cp -R clang-3.2.src llvm-3.2.src/tools/clang
6    cd llvm-3.2.src
7    mkdir build
8    cd build
```

```
9    cmake -DCMAKE_INSTALL_PREFIX=/home/username/.local ../.
10   make -jN
11   make install
```

Note: occorre modificare ‘username’ con l’user dell’utente con cui effettuiamo l’installazione, e ‘N’ con il numero di thread paralleli da utilizzare.

SCARICARE SHADOW

Per effettuare il download del codice è possibile visitare il seguente indirizzo: https://shadow.cs.umn.edu/download/ oppure utilizzando git[22] tramite il seguente codice:

```
1    git clone https://github.com/shadow/shadow.git
2    cd shadow
3    git checkout release
```

In questa pagina (https://github.com/shadow/shadow/tags) è possibile visualizzare le Releases del software, con le relative modifiche apportate.

INSTALLAZIONE

Posizionarsi nella cartella superiore in cui abbiamo scaricato Shadow, e iniziamo l’installazione tramite i seguenti comandi:

```
1    ./setup dependencies
2    ./setup build
3    ./setup install
```

[22] Sistema software di controllo di versione distribuito

Per maggiori dettagli è possibile visualizzare la guida tramite il comando --help

CONFIGURAZIONE DI SISTEMA

Di default Linux limita il numero di file che si possono aprire. Se ogni nodo virtuale del simulatore apre dei file dal socket, bisognerà incrementare il limite al fine di evitare errori causati dalle chiamate open() o socket(). Per fare questo bisogna andare a modificare delle impostazioni nei file di configurazione aumentando i parametri:

ulimit -n
cat /proc/sys/fs/file-max
cat /proc/sys/fs/inode-max

Bibliografia

[1] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Main_Page

[2] Giancarlo Livraghi. (2014) Rete e Comunicazione. [Online]. http://gandalf.it/data/

[3] "Testo unico sulla privacy," Repubblica Italiana, D.Lgs 196/2003 Giugno 30 Giugno 2003.

[4] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding Routing Information," in *Proceedings of Information Hiding Workshop*, 1996, pp. 137-150.

[5] M. Reed, P. Syversion , and D. Goldschlag, "Anonymous connection and onion routing," *IEEE journal on Selected Areas in Communications*, vol. 16, no. 4, pp. 482-494, 1998.

[6] (2013) The Tor Project. [Online]. www.torproject.org

[7] (2013) Tor Metrics. [Online]. https://metrics.torproject.org/index.html

[8] (2013) The Shadow Simulator. [Online]. http://shadow.github.io/

[9] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation Onion Router," in *Proceedings of the 13th USENIX Security Symposium*, 2004.

[10] B Chun, D Culler, T Roscoe, A Bravier, and L. Peterson, "Planetlab: An overlay testbed for broad-coverage services," in *SIGCOMM Computer Communication Review*, 2003, pp. 33:3-12.

[11] Ross S, *Simulation*, Terza ed. San Diego: Academic Press, 2002.

[12] Fabio Chiusi. (2013, Ottobre) Repubblica.it. [Online]. http://chiusinellarete-messaggeroveneto.blogautore.repubblica.it/tag/scandalo-nsa/

[13] A. Silberschatz, P. Baer Galvin, and G. Gagne, *Sistemi operativi. Concetti ed esempi*, 6th ed., Marra V, Ed.: Pearson Education Italia, 2002.

[14] version 2.0 The LibEvent notification library. [Online]. http://monkey.org/libevent

[15] The OpenSSl cryptographic library. http://www.openssl.org/.

[16] R. Jansen and N. Hopper, "Shadow: Running Tor in a Box for accurate and Efficient Esperimantion," Proc of the 19th Network and Distributed System Security Symposium 2012.

[17] C. Tang and I. Goldberg, "An improved algorithm for Tor circuit scheduling," in *Proceedings of the 17th ACM Conference on Computer and Communication Security*, 2010, pp. 329-339.

Indice delle figure

Indice dei grafici