UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Elaborato finale

BGP e Load Centrality

Implementazione del calcolo della centralità nel protocollo BGP

Relatore
Leonardo Maccari
Co-Relatore
Lorenzo Ghiro

Laureando
Mattia Milani

Anno accademico 2017/2018

Sommario

Il traffico di controllo generato dai protocolli di routing è fondamentale per il corretto funzionamento di ogni rete di computer. La frequenza di generazione dei messaggi di controllo è decisiva e influisce profondamente sulle performance di un protocollo. Ad esempio, incrementando tale frequenza, si possono rapidamente rilevare malfunzionamenti della rete e attivare i conseguenti meccanismi di ripristino. La durata dei disservizi viene così ridotta al costo di un elevato numero di messaggi di controllo. Il trade-off tra il volume del traffico di controllo e le performance è evidente e, nonostante decenni di ricerca ed esperienza nella configurazione di protocolli di routing, non sono molti gli studi riguardanti l'ottimizzazione di questo trade-off.

Una recente proposta [1] suggerisce, fissato un livello di overhead, di regolare la generazione dei messaggi di controllo di ogni nodo in base alla centralità di quest'ultimo all'interno della topologia di rete. Questo approccio, chiamato Pop-Routing, è stato applicato con successo inizialmente ai protocolli *Link State* (LS) e successivamente ai protocolli *Distance Vector* (DV) [2], grazie al calcolo di metriche di centralità per grafi. Nel caso dei LS i nodi, godendo di una completa conoscenza della topologia, possono sfruttare un algoritmo centralizzato per il calcolo della *Betweenness Centrality* (BC) [3]. Gli autori di [2] hanno invece proposto un algoritmo distribuito per il calcolo della *Load Centrality* (LC) per estendere Pop-Routing ai protocolli DV.

Lo scopo di questa tesi è estendere il Pop-Routing ad una categoria di protocolli finora esclusa, ovvero i protocolli *Path Vector* (PV). Per questo motivo ho studiato *Border Gateway Protocol* (BGP), il più noto ed il più diffuso fra questi protocolli, utilizzato tutti i giorni per il funzionamento di Internet. Ho successivamente definito, implementato e testato un algoritmo distribuito per il calcolo della *Destination Partial Centrality* (DPC), una metrica di centralità che ho progettato appositamente per BGP.

Ho sviluppato un simulatore ad eventi discreti per verificare la correttezza formale del nuovo algoritmo simulandolo su grafi sintetici. Ho inoltre implementato l'algoritmo in *Bird* [1], una distribuzione open source di BGP, e quindi testato il codice all'interno dell'emulatore NePa [4]. I risultati preliminari di questa tesi confermano la bontà dell'approccio Pop-Routing anche per BGP.

I contributi di questa tesi possono essere pertanto così riassunti:

- Definizione Destination Partial Centrality, un indice di centralità progettato per BGP;
- Sviluppo e implementazione dell'algoritmo in *Bird*, distribuzione open source di BGP;
- Verifica correttezza algoritmo tramite simulatore ad eventi discreti ed emulazioni con NePa [4].

[1] Bird site

Indice

1 Introduzione

1.1 Internet Oggi

Attualmente la rete Internet è composta dall'insieme degli *Autonomous System* (AS), un AS è definito [5] come un "*sistema connesso che gestisce uno o più prefissi IP utilizzati da uno o più operatori di rete che hanno una singola e ben definita politica di routing*" e si occupano di smistare il traffico da e verso l'esterno attraverso l'*Internet Service Provider* (ISP) di cui fanno parte. Mantenendo la classificazione degli AS fornita dal *Center for Applied Internet Data Analysis* (CAIDA) [1] possiamo raggruppare queste autorità in 3 macro classi:

Tabella 1.1: Classificazione AS CAIDA

Classe	Descrizione
Transit/Access	AS che sono stati ritenuti essere fornitori di transito e/o di accesso alla rete.
Content	AS che forniscono sistemi di hosting e di distribuzione di contenuti.
Enterprise	AS di varia natura, università ed aziende ai margini della rete che rappresentano principalmente utenti, o provider di accesso ad internet, transito o contenuti.

Il CAIDA definisce inoltre la tassonomia delle relazioni (collegamenti) tra gli AS e la suddivisione prevede le seguenti 3 classi:

Tabella 1.2: Classificazione collegemtnti intra-AS

Classe	Descrizione
customer-to-provider (c2p)	gli ISP dei livelli più bassi pagano gli ISP di livello più alto per il transito del traffico, e l'accesso ad internet.
peer-to-peer (p2p)	Collegamento in cui lo scambio di traffico si limita a quello proveniente da ISP customer diretti di entrambi, questo evita pagamenti ai livelli superiori.
sibling-to-sibling (s2s)	collegamento presente tra due AS appartenenti allo stesso ISP.

La topologia di internet, come la conosciamo oggi, è sostanzialmente definita dai collegamenti che sono stati stipulati attraverso contratti ed accordi commerciali. Al proprio interno gli AS possono utilizzare il protocollo di routing che preferiscono, mentre all'esterno per gestire i collegamenti tra le diverse entità tutti devono concordare sullo stesso protocollo.

Ogni AS ha delle politiche di gestione dei collegamenti, i contratti commerciali che stipula lo potrebbero limitare in alcuni aspetti o costringerlo a prendere decisioni di instradamento non sempre ottimali. Per esempio i contratti potrebbero prevedere dei costi riguardanti la quantità di traffico instradato, il che potrebbe rendere preferibile una linea meno costosa ma poco performante e mantenere una linea di backup costosa.

I normali protocolli di routing non permettono un'applicazione così accurata dipendente dai particolari accordi, perciò si necessita un protocollo che sia configurabile manualmente e che durante la scelta del percorso prenda in considerazione tutte le politiche predisposte. Negli anni sono stati scritti ed implementati diversi protocolli che permettessero lo scambio di informazioni tra AS. Quale *Exterior Gateway Protocol* (EGP) [6] il cui aggiornamento risale al 1984 e definiva il trasferimento di

[1] http://www.caida.org/data/as-classification/

informazioni tra i diversi sistemi. Successivamente è stato sviluppato BGP la cui ultima versione è la numero 4 [7], BGP è l'attuale standard de facto.

1.2 Centralità nelle reti

La centralità è un argomento di estrema importanza in diversi ambiti, dallo studio dei grafi, alle scienze sociali. È stata introdotta nell'ambito scientifico per misurare l'influenza delle persone dal punto di vista sociologico. Si sta portando questo studio anche nell'ambito del networking, perchè la conoscenza della centralità permette di gestire in modo più preciso la sicurezza [8], i servizi [9] e la rete in generale [1]. Ma è possibile migliorare le prestazioni delle reti di comunicazione [10], [11], applicando degli algoritmi di calcolo della centralità, data la loro importanza ne viene studiata l'applicazione ai protocolli di routing [2].

Gli indici di centralità applicati ad un grafo ci permettono di capire quali sono i vertici più importanti all'interno della topologia. Sono state proposte due categorie che si dividono in base a ciò che viene considerato "importante":

- In base all'**importanza del traffico** [12] in questo caso ciò che definisce quanto è importante un vertice è l'importanza di ciò che passa attraverso di lui;
- In base alla **coesione** [13] della rete, quindi l'importanza di un vertice dipende da come viene calcolata la coesione.

Una centralità appropriata ad una categoria potrebbe dare risultati errati se applicata all'altra. Una metrica di centralità può esser calcolata in base ai percorsi che attraversano un nodo oppure può dipendere dalla lunghezza del percorso utilizzato per raggiungere un nodo, alcuni indici richiedono una conoscenza della topologia maggiore rispetto ad altri, vi sono molte tecniche differenti che si adattano in base al contesto in cui vengono utilizzate.

1.3 Scopo della tesi

Questa tesi ha lo scopo di verificare la possibilità di applicare il calcolo delle metriche di centralità ad un protocollo di condivisione delle informazioni tra AS, BGP, studiarne l'implementazione ed infine trarre le conclusioni da ciò che è stato prodotto in modo da porre le basi per eventuali studi futuri che prevedano un'applicazione di questi indici. I risultati verranno poi confrontati con metriche di centralità attualmente utilizzate in altri protocolli e ne verrà discussa l'applicabilità.

Nella sezione 2 verranno date delle informazioni di background riguardanti i protocolli come BGP ed il calcolo della centralità. Nella sezione 3 verrà definito e modellato il calcolo della centralità applicato a BGP di cui nella sezione 4 verrà spiegata l'implementazione. Nel capitolo 5 verranno descritti gli esperimenti eseguiti e successivamente si avranno le conclusioni nella sezione 6.

La comunità attualmente non ha ancora raggiunto degli accordi comunemente accettati su tutte le problematiche derivanti da BGP il che lo rende un tema caldo anche se sono passati più di 10 anni dalla sua ultima versione ufficiale.

2 Background

2.1 Tipi di routing

L'operazione di routing è il procedimento che permette di scegliere l'interfaccia corretta ed il *Next Hop* (NH) verso una destinazione, ogni dispositivo che è posto tra la sorgente e la destinazione deve sapere verso quale interfaccia inoltrare il traffico. Nelle reti di computer il routing è effettuato attraverso la *Routing Table* (RT), una tabella mantenuta da ogni dispositivo che permette tramite la sua consultazione di sapere in che direzione inviare i pacchetti.

Le tabelle di routing in topologie di piccole dimensioni potrebbero essere configurate manualmente, ma al crescere della rete questa operazione diventa troppo complessa da eseguire, con probabilità di errore molto elevate. Questo ha portato alla creazione dei protocolli di routing, che permettono la condivisione tra dispositivi delle informazioni di raggiungibilità, permettendo così senza alcuna configurazione manuale di sapere il percorso che un pacchetto dovrà effettuare per raggiungere la destinazione.

Esistono diverse categorie di routing e si differenziano dalle informazioni condivise e da come vengono interpretate:

- Protocolli **DV**, prevedono l'utilizzo dell'algoritmo di Bellman–Ford. Ad ogni collegamento viene assegnato un costo. E le informazioni vengono propagate nel percorso a costo complessivamente minimo;
- Protocolli **LS**, Ogni vertice in questo protocollo dovrà calcolare localmente i percorsi a costo minimo per raggiungere qualsiasi altro vertice, ciascuno può utilizzare al proprio interno l'algoritmo che preferisce, generalmente si utilizza Dijkstra [14].

Queste due categorie hanno una differenza di base, i protocolli DV non necessitano della conoscenza completa della topologia per funzionare, mano a mano che le informazioni di raggiungibilità si propagano verranno aggiunte/sostituite delle voci nella tabella di instradamento, mentre i protocolli LS necessitano della conoscenza completa del grafo all'interno del singolo nodo, in modo che ciascuno di essi possa applicare l'algoritmo.

All'aumentare delle dimensioni della rete però questi protocolli diventano sempre più instabili e inutilizzabili, i protocolli LS richiederebbero troppe risorse locali per il calcolo dei percorsi minimi ed in più il traffico dovuto ai messaggi di flooding sarebbe eccessivo. Dunque l'uso resta limitato a reti intra-AS.

2.2 Path Vector

L'esplosione crescente di internet e l'instabilità dei protocolli precedenti ha portato alla necessità di definire un nuovo tipo di protocolli [15]. Per le comunicazioni inter-AS viene utilizzato un terzo tipo di protocolli, i **PV**, per ogni AS vengono definiti uno o più "*speaker node*", che comunicheranno tra di loro le proprie RT, in modo simile a come viene fatto dai protocolli DV. Solitamente all'interno di un PV, viene utilizzato come algoritmo di routing una versione modificata di Bellman–Ford. Ogni nodo dovrà condividere le proprie destinazioni conosciute ed anche eventuali informazioni aggiuntive. Ogni speaker comunica le destinazioni che sa raggiungere ed il percorso effettuato dalle informazioni per raggiungerle, anziché la metrica. Solitamente all'interno del path (percorso condiviso) vengono inseriti i domini attraversati. Nell'*Request For Comment* (RFC) 1322 [15] oltre a definire le informazioni che dovrebbero essere trasportate da un protocollo PV vi è anche una prima definizione per quanto riguarda l'applicazione delle politiche di preferenza delle rotte. Viene detto inoltre che i criteri di selezione utilizzati da un dominio non devono influenzare gli altri, e che viene concessa la massima libertà ai domini in fatto di policy.

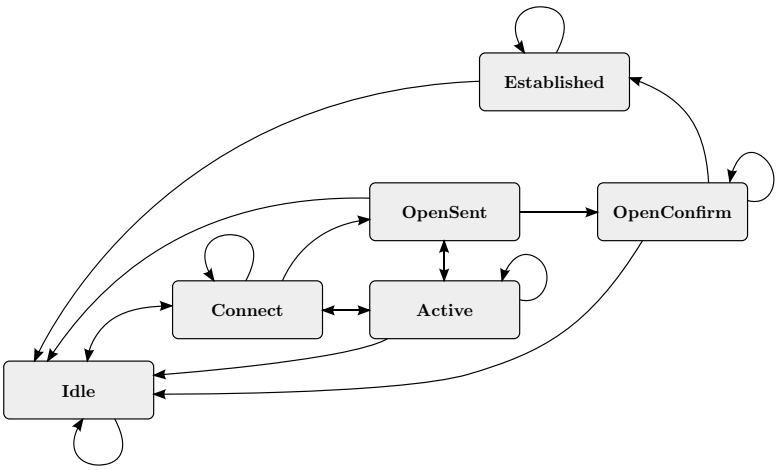


Figura 2.1: Final State Machine BGP (Johannes Rössel©)
© https://commons.wikimedia.org/wiki/User:Joey-das-WBF

Entrarono così nell'utilizzo comune inter-AS i protocolli PV, essenzialmente in due versioni:

- **BGP** un protocollo nato al di sopra dello standard TCP/IP per gestire il traffico tra AS differenti;
- ***Inter Domain Routing Protocol*** **(IDRP)** protocollo OSI progredito attraverso la standardizzazione ISO.

Ci focalizzeremo solamente su BGP in quanto è diventato lo standard attualmente utilizzato.

2.2.1 BGP

BGP raggiunse la sua quarta versione nel marzo 1995 [16], aggiornata poi nel 2006 [7]. Permetteva fin dall'inizio la conoscenza del percorso che avrebbe fatto l'informazione e l'applicazione di filtri decisionali. BGP permette agli AS di stabilire collegamenti gli uni con gli altri, ogni collegamento è detto sessione e riguarda solamente due AS, infatti gli unici tipi di connessioni supportate da BGP sono quelle di tipo punto-punto, due soli interlocutori.

Le informazioni condivise da BGP però non si limitano al solo collegamento tra AS ma prevedono anche la condivisione delle informazioni imparate dall'esterno verso l'interno del dominio d'azione, BGP si può distinguere in due parti:

- ***internal Border Gateway Protocol*** **(iBGP)** rappresenta la parte del protocollo che gestisce le informazioni internamente all'AS, oltre alla propagazione della conoscenza proveniente dall'esterno prevede anche la scoperta e la condivisione delle informazioni locali;
- ***external Border Gateway Protocol*** **(eBGP)** questa parte di protocollo riguarda il come vengono gestite le rotte imparate dall'esterno e come viene propagata la conoscenza locale al di fuori del dominio.

BGP per prendere decisioni ed eseguire delle azioni si basa sulla propria macchina a stati finiti interna, rappresentata dalla figura 2.1, che consiste in 6 stati diversi. Lo stato di partenza sarà "idle" ed ad ogni evento che si verificherà vi sarà un passaggio di stato fino a raggiungere "estabilished" in cui la sessione è attiva, in caso di errori si tornerà invece allo stato di "idle". Gli eventi che provocano le variazioni di stato sono elencati completamente dall'RFC 4271 [7] nella sezione 8.

Una volta che la sessione tra due speaker (anche detti peer) BGP è stata stabilità inizieranno a condividere tra di loro le informazioni di raggiungibilità, attraverso dei pacchetti detti di UPDATE. Per mantenere la connessione attiva invece vengono utilizzati dei pacchetti di KEEPALIVE; finché non vi saranno altre informazioni da propagare sarà sufficiente un pacchetto KEEPALIVE di dimensione minima.

Generazione degli UPDATE

Dopo aver ricevuto il pacchetto contente una nuova destinazione ne verrà valutata l'aggiunta alla propria RT. La valutazione verrò fatta in base ai filtri attualmente attivi per quella sessione, grazie ai quali ad ogni destinazione (con relativo path di raggiungibilità) verrà associato un tasso di preferenza, una volta valutate tutte le rotte che conducono alla stessa destinazione verrà scelta quella con tasso di preferenza più alto.

All'interno dei messaggi di UPDATE le informazioni riguardanti le destinazioni sono molteplici, e queste informazioni sono memorizzate sotto forma di attributi, come per esempio l'indirizzo di rete che rappresenta la rotta, il NH, l'AS_Path che definisce il percorso fino ad ora conosciuto per raggiungerla. Come definito dall'RFC 4271 [7] ci sono molti tipi di attributi (descritti brevemente nella sezione successiva). I messaggi di UPDATE vengono scambiati da un nodo con i suoi vicini in situazioni particolari:

- Quando un nodo nasce e deve far conoscere al resto della rete le proprie destinazioni;
- Quando vi è una variazione nel modo in cui si raggiunge una certa destinazione (scoperta di un percorso migliore);
- Quando vi è una modifica dei filtri del nodo, in entrata o in uscita. Andranno rifatte tutte le decisioni fatte precedentemente per tutte (o una parte) delle destinazioni, e dovranno essere comunicate ai vicini;
- Quando viene eliminato un percorso per una destinazione.

Processing degli UPDATE

Tutte le informazioni contenute dal pacchetto vanno ad influire sulle decisioni che lo speaker ricevente potrà effettuare, un peer può anche decidere di mantenere alcuni UPDATE e scartarne altri, in base alle proprie politiche.

Come prima cosa viene controllato che non vi siano loop, attraverso l'attributo AS_Path. Successivamente verranno applicate una serie di scelte in ordine di raffinamento, nel caso pessimo non vi saranno alternative, l'ultima condizione impone che solamente un percorso per destinazione venga mantenuto come ottimo.

- Preferire destinazioni con tasso di preferenza maggiore;
- Preferenza per L'AS_Path più breve (con meno salti tra AS);
- Preferenza per l'origine inferiore 0 – iBGP , 1 – eBGP, 2 – INCOMPLETE;
- Preferire destinazioni con attributo MULTI_EXIT_DISC più alto;
- Preferire eBGP rispetto a iBGP;
- Preferire destinazioni ricevute dal peer con identificativo inferiore;
- Preferire le destinazioni ricevuti dall'indirizzo inferiore.

Tutte queste regole ed altre vengono descritte più in dettaglio dall'RFC 4271 [7] sezione 9.1 in particolare la sezione 9.1.2.2.

I messaggi di UPDATE provocano mutamenti nelle tabelle di instradamento anche quando una rotta viene eliminata dalla propria RT, bisogna infatti avvertire i propri vicini utilizzando una apposita sezione del pacchetto di UPDATE, questo permetterà ai vicini di capire che quella destinazione non è più raggiungibile attraverso quel percorso, che quindi cercheranno vie alternative per raggiungerla.

Attributi dei messaggi di UPDATE

Un pacchetto di UPDATE è così formato:

Tabella 2.1: Pacchetto di UPDATE BGP

Withdrawn Routes Length (2 octets)
Withdrawn Routes (variable)
Total Path Attribute Length (2 octets)
Path Attributes (variable)
Network Layer Reachability Information (variable)

Le prime due sezioni ci danno informazioni quando una rotta non è più raggiungibile attraverso il peer da cui abbiamo ricevuto il messaggio. Nella quarta sezione, path attributes, vengono definiti diversi attributi, alcuni devono essere obbligatoriamente presenti nel messaggio di UPDATE per poterlo considerare valido, altri invece no. Gli attributi si distinguono in 4 categorie:

- Well-known mandatory;
- Well-known discretionary;
- Optional transitive;
- Optional non-transitive.

I primi due indicano attributi che devono esser obbligatoriamente conosciuti e quindi interpretabili da tutti gli speaker BGP, i primi con presenza obbligatoria nel pacchetto, i secondi a discrezione del generatore possono essere presenti oppure no.

Gli utlimi due tipi di attributi non è detto che debbano essere conosciuti da tutti i peer bgp, per questo la definizione di "opzionali", i primi, gli attributi opzionali transitivi, specificati nell'RFC4271 sezione 5 [7], possono essere interpretabili o meno per un peer, nel caso in cui non sappia interpretarli dovrà comunque includerli nel pacchetto di UPDATE che inoltrerà verso altri AS. Mentre i secondi attributi di tipo opzionale non essendo transitivi non vengono reinoltrati ma semplicemente ignorati e scartati, nel caso in cui non siano interpretabili.

Considerazioni importanti su BGP

Una cosa importante da sottolineare in BGP è che non tutti i nodi condividono delle destinazioni, alcuni AS sono solo dei punti di transito e di controllo nella rete, le loro destinazioni non appariranno mai all'interno di nessuna RT. Potremmo perciò effettuare una separazione degli speaker che compongono la rete di AS connessi tramite sessioni BGP come definito nella tabella 2.2.

Tabella 2.2: Separazione degli speaker BGP

Notazione	Significato	Eventuale simbolo
Destination node	AS che condivide delle destinazioni che sono state generate al proprio interno	
Forward node	AS che condivide destinazioni imparate a raggiungere attraverso altri AS, non ha delle destinazioni interne	

Ovviamente per la condivisione di informazioni ogni nodo si affida alle proprie policy, perciò non è neanche detto che condivida una certa destinazione con tutti i suoi vicini ma magari con solamente un sottoinsieme di essei.

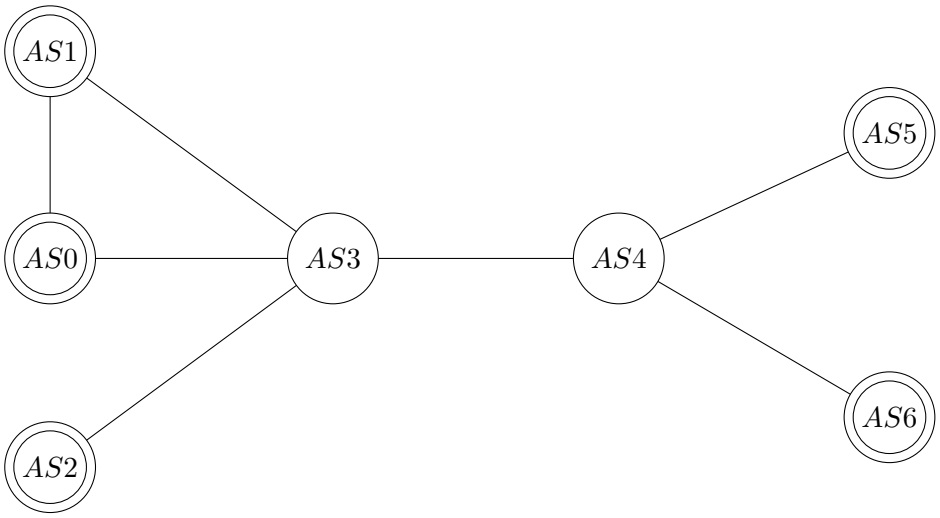


Figura 2.2: Topologia che evidenzia la separazione tra Destination node e Forward node

Come si vede dalla topologia 2.2 non tutti i nodi esportano delle destinazioni, infatti i nodi $AS3$ ed $AS4$ vengono considerati solo per la loro funzione di trasporto. Ma al loro interno non hanno nulla se non la tecnologia che gli permette di fare da tramite tra le destinazioni, nessuno vorrà mai raggiungere qualcosa che si trova all'interno di $AS3$ anche perchè $AS3$ non condivide nulla di ciò che ha all'interno.

Perciò per esempio $AS6$ saprà che per raggiungere una destinazione condivisa da $AS0$ dovrà passare attraverso un path di questo genere $\{AS4 - AS3 - AS0 - i\}$ (i sta per internal) ma non avrà mai all'interno della propria RT un percorso che termini con una destinazione interna ad $AS3$ o $AS4$.

2.3 Centralità

Ai grafi di rete può essere applicata una metrica di centralità estremamente importante, la BC che ha un'applicazione nel posizionamento dei servizi [9] miglioramento del routing [17],[1] ed altri usi. L'applicazione di metriche di centralità a protocolli di routing è, in definitiva, un promettente campo di ricerca per migliorare i protocolli esistenti. La BC si applica bene ai protocolli di routing LS perchè per essere calcolata richiede una conoscenza completa dei cammini minimi tra ogni coppia di vertici, e viene generalmente calcolata off-line. L'algoritmo più veloce che permette un calcolo della BC conosciuto in letteratura è quello di Brandes [3].

Lo scopo di questa tesi è verificare che il calcolo della centralità possa essere effettuato in modo distribuito anche all'interno di protocolli PV, tra i quali viene preso in considerazione BGP.

Gli algoritmi di routing DV non prevedono la completa distribuzione della topologia della rete, il che è un problema per il calcolo della centralità, una conoscenza parziale porterebbe ad un calcolo non congruente con l'effettiva BC. Si presentano anche problemi di natura computazionale per i protocolli LS, il calcolo in locale richiede troppo tempo, anche se il fenomeno è attenuabile attraverso delle tecniche euristiche [18].

Possiamo però considerare un'alternativa alla BC che è la LC [19], [20], che non richiede le restrizioni nella conoscenza.

2.3.1 Load Centrality

La LC può essere definita nel seguente modo: sia $G = (V, E)$ un grafo dove V è l'insieme dei vertici ed E l'insieme degli archi che lo compongo.

definition 2.3.1. (Load centrality LC) *dato un grafo G(V,E) ed un algoritmo per definire il/i (potenzialmente multipli) cammino/i a costo minimo tra ogni possibile coppia di vertici (s, d), sia $\theta_{s,d}$ l'unità di traffico inviata dal vertice s al vertice d. Assumiamo che la quantità di carico sia passata al*

Next Hop (NH) attraverso il cammino a costo minimo, e nel caso in cui vi siano più NH, il carico è diviso equamente tra i NH. Chiamiamo $\theta_{s,d}(v)$ il carico complessivamente inoltrato dal vertice v.

$$LC(v) = \sum_{s,d \in V} \theta_{\mathrm{s,d}}(v)$$

Si assume che $s \neq d, s \neq v, d \neq v$ e se $\theta_{\mathrm{s,d}}$=1 la LC è una proprietà definita interamente dalla struttura del grafo, e dall'algoritmo usato per scoprire i path a costo minimo. In un grafo qualsiasi il numero massimo di cammini che potranno attraversare il nodo v per il trasferimento della quantità di carico tra s e d sarà pari a $(n-1)(n-2)$ assumendo che il percorso da s a d e da d ad s vengano considerati come due percorsi separati, ed in più i contributi non vengono sommati ne' alla sorgente ne' alla destinazione, dunque la LC può essere normalizzata a:

$$\overline{LC}(v) = \frac{1}{(N-1)(N-2)} \sum_{s,d \in V} \theta_{s,d}(v)$$

2.3.2 Calcolo distribuito della centralità

Il calcolo in locale sarebbe comunque computazionalmente complesso per reti di dimensioni troppo elevate, perciò si può pensare ad un approccio distribuito, ogni vertice v dovrà calcolare la propria centralità in base alle informazioni in suo possesso in quel momento, per poi condividerla fino a che tutti i vertici non avranno raggiunto la convergenza.

Sono già presenti in letteratura algoritmi distribuiti che permettono di calcolare la LC [2]. Si basa sul passaggio del carico della LC, ogni vertice genera un carico unitario per tutte le possibili destinazioni e viene poi passato al/ai NH (tramite un messaggio con degli attributi specifici), mantenendo una lista aggiornata dei NH e dei *Previus Hop* (PH) per raggiungere la destinazione d.

L'algoritmo proposto all'interno dell'articolo è stato applicato ad un protocollo DV con successo, perciò con le adeguate modifiche potrà essere applicato anche ad un protocollo PV.

3 Calcolo della centralita a contributo parziale

La LC non si adatta completamente alla natura di BGP, in quanto presenta una diversità sostanziale, nella LC tutti i nodi vengono considerati come possibili destinazioni, punti di arrivo attraverso un cammino a costo minimo, cosa che non è vera in BGP. I nodi che condividono delle destinazioni saranno gli unici raggiungibili, e le sorgenti del traffico possono essere solamente dei nodi che hanno al loro interno delle rotte che vengono esportate. Inoltre le connessioni tra i peer BGP sono ottenute tramite TCP/IP in cui il flusso di informazioni è sempre bidirezionale, in quanto l'apertura di una connessione prevede lo scambio di informazioni tra i due partecipanti. Per questi motivi si rende necessario definire una nuova metrica di centralità.

3.1 Modello

Consideriamo $G = (V, E)$ un grafo dove V è l'insieme dei vertici ed E l'insieme degli archi che lo compongono.

Definiamo $D \subseteq V$ come l'insieme delle destinazioni possibili: ovvero l'insieme dei nodi che sono raggiungibili attraverso cammini nel grafo.

e definiamo come C l'insieme che rappresenta tutte le possibili coppie di vertici appartenenti all'insieme D, ogni coppia rappresenta una sorgente ed una destinazione di traffico, così definito:

$$C = \{(s, d) | s \in D, d \in D, s \neq d\}$$

Grazie a questi insiemi possiamo definire una nuova metrica di centralità che si adatterà in modo migliore a BGP.

3.1.1 Destination Partial Centrality

definition 3.1.1. DPC *Dato un grafo $G(V, E)$ ed un algoritmo per definire il/i (potenzialmente multipli) cammino/i a costo minimo tra ogni possibile coppia di vertici appartenenti a C ($\forall c \in C$), sia θ_c l'unità di carico inviata dal vertice $s \in c$ Al vertice $d \in c$. assumiamo che la quantità di carico sia passata al NH attraverso un cammino a costo minimo, e nel caso in cui vi siano più NH, il carico venga diviso equamente tra i NH. Chiamiamo $\theta_c(v)$ il carico complessivamente inoltrato dal vertice v.*

$$dpc(v) = \sum_{c \in C} \theta_c(v)$$

Come nella definizione della LC nella sezione 2.3.1, il vertice v non deve appartenere alla coppia di vertici c considerata, nel caso in cui $D = V$ e si stia considerando un grafo non direzionato possiamo normalizzare la DPC nel seguente modo:

$$\overline{DPC}(v) = \frac{1}{(N-1)(N-2)} \sum_{c \in C} \theta_c(v) = \overline{LC}(v)$$

Altrimenti, non possiamo però normalizzare come fatto nella definizione della LC, perché la centralità non è più definita dalla sola struttura del grafo, ma anche da come vengono formati gli insiemi di partenza, non è detto che vi siano ancora $(N-1)(N-2)$ percorsi che attraversino v. Ora la Centralità dipenderà anche da quanti vertici apparterranno agli insiemi D, C, dunque avremo la seguente normalizzazione:

$$\overline{DPC}(v) = \frac{1}{(|D|-1)(|D|-2)} \sum_{c \in C} \theta_c(v)$$

Se D contiene tutti i vertici non vi sono variazioni dal punto di vista dei valori di LC rilevati sui grafi, in quanto verrebbero comunque considerate tutte le coppie di vertici. Prendiamo la topologia mostrata in figura 3.1

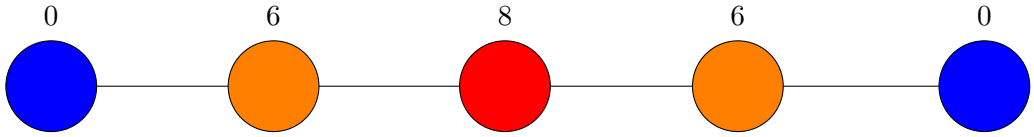


Figura 3.1: Topologia di esempio di calcolo della LC, con LC per nodo

La LC considera tutte le 20 coppie di vertici, e anche nel caso in cui solo alcuni di essi condividessero informazioni verrebbero comunque considerate tutte. Infatti osservando la figura 3.2 possiamo notare un netta differenza nei valori di centralità, che sono però dovuti al fatto che la DPC considererà solamente 2 coppie di vertici.

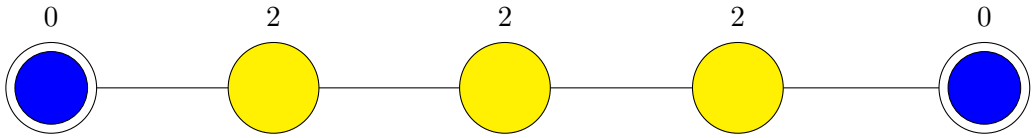


Figura 3.2: Topologia di esempio di calcolo della DPC

3.1.2 Modifiche al Protocollo BGP

Sfruttando gli elementi ed i concetti già presenti all'interno del protocollo è stato possibile procedere al calcolo della centralità con la semplice aggiunta di alcuni elementi ai pacchetti di UPDATE.

Dopo aver visto gli attributi definiti da BGP nella sezione 2.2.1 passiamo a discutere gli attributi transitivi opzionali che andranno aggiunti per permettere il calcolo della centralità, andranno aggiunti ad ogni pacchetto di UPDATE. Gli attributi che sono stati aggiunti sono: NH e ASLoadList.

Nome attributo	Contenuto	
NH	NH_Number (2 octets)	NH_AS (variable)
ASLoadList	AS_Load_Number (2 octets)	AS_ID + AS_Load + AS_State (variable)

Tabella 3.1: Tabella attributi aggiuntivi

Il primo attributo aggiuntivo presente nella tabella 3.1, NH, è una lista dei NH che un nodo v potrà utilizzare per raggiungere la destinazione condivisa, utilizzato per fare in modo che gli altri nodi possano sapere quanti cammini minimi conosce v per la destinazione d. L'attributo sarà suddiviso in due parti, la prima (16 bit) indicherà il numero di NH presenti, nella seconda parte, avremo poi una parte a dimensione variabile (32 bit per ogni NH presente). Per rappresentare un NH si può utilizzare l'identificativo dell'AS.

ASLoadList, il secondo attributo aggiuntivo nella tabella 3.1 è utilizzato per distribuire il carico di tutti gli AS conosciuti, anche di quelli che non condividono destinazioni. Il carico di un determinato AS è un'informazione che viene mantenuta globalmente nel peer BGP e non localmente alla sessione, perciò devo porre attenzione sul come aggiornare questo attributo. Ogni peer è responsabile dell'aggiornamento della propria centralità nella lista degli AS_Load conosciuti, in questo modo ognuno diventa responsabile di mantenere globalmente aggiornato il proprio carico.

L'attributo ASLoadList sarà formato essenzialmente da 4 elementi:

- AS_Load_Number che indicherà quanti elementi sono presenti all'interno dell'ASLoadList inviata (16 bit);
- AS_ID indicherà l'AS a cui si riferiranno i dati seguenti (32 bit);
- AS_Load carico rilevato per l'AS (32 bit);

- AS_State valore che indica il livello di aggiornamento della metrica di centralità, ogni speaker è responsabile di aggiornare questo valore nel caso in cui vi siano variazioni del proprio AS_Load (32 bit).

Un peer utilizza questi 4 attributi uniti all'algoritmo 3.1 ed al resto del pacchetto di update per calcolare la propria centralità ed aggiornare quelle che conosce.

Un'altra modifica da applicare al protocollo è quella riguardante la ricezione dei Withdraws, i messaggi che segnalano la non raggiungibilità di una destinazione, in questo caso andranno rimosse le informazioni di centralità influenzate dalla destinazione non più raggiungibile, e successivamente bisognerà aggiornare la propria centralità.

Oltre a queste modifiche dei pacchetti di UPDATE un'altra parte modificata all'interno del protocollo riguarderà l'impiego dell'algoritmo distribuito per il calcolo ed il mantenimento della centralità [2]. Questo però non può essere applicato senza modifiche, infatti quell'algoritmo punta a calcolare la LC mentre a noi interessa la DPC. Le modifiche però sono minime e riguarderanno solamente l'insieme dei nodi considerato ed il carico inoltrato.

Algorithm 3.1: Eseguito da v per il calcolo distribuito della DPC

```
1  Inizializzazione:
2      foreach d ∈ D − v do
3          ...
4  Ripeti ogni δ s:
5      ... foreach d ∈ D − v do
6          if v ∈ D then
7              loadOut[d] = 1 + Σ_{u∈PH[d]} loadIn^u[d];
8          else
9              loadOut[d] = 0 + Σ_{u∈PH[d]} loadIn^u[d];
10         ...
11     ...
12 Alla ricezione < u, NH^u, contrib^u > da u fai:
13     foreach d ∈ D − v do
14         ...
```

L'insieme ora considerato sarà D, righe [2-5-13] in modo che solamente le unità di carico riguardanti delle destinazioni vengano distribuite. Oltre a questo solamente i vertici appartenenti a D potranno aggiungere la propria unità di carico, righe [da 6 a 9].

In questo modo viene garantito che le uniche coppie considerate per il calcolo della centralità saranno quelle che condividono informazioni di raggiungibilità.

Nel caso in cui $D = V$ le modifiche non avrebbero effetto e l'algoritmo convergerebbe alla LC.

4 Centralità in BGP: Implementazione e convergenza

Attualmente non vi è un sistema che in BGP implementi il calcolo della centralità. Viene proposto quindi un'estensione che implementa l'algoritmo distribuito modificato in 3.1.2, richiedendo l'aggiunta di alcuni dati e l'implementazione all'interno dei pacchetti di UPDATE degli attributi presentati nella sezione 3.1.2.

4.1 Assunzioni semplificative

Un'applicazione che però comprenda ogni sfaccettatura di BGP risulterebbe troppo complessa per questa tesi, perciò sono state applicate alcune assunzioni semplificative, che ci aiuteranno nella comprensione del problema e nella modellazione della soluzione proposta.

Le prime assunzioni si pongono sul grafo considerato e sulla concezione di condivisione:

- Non ci poniamo il problema di quale router eBGP sia collegato a quale AS ma consideriamo un unico grande router eBGP all'interno dell'AS con più interfacce ciascuna collegata all'AS corrispondente corretto;
- Per mantenere la propria RT ad uno stato corretto BGP utilizza le proprie policy ed i filtri basati su delle decisioni prese a monte dall'amministratore di rete. Non è possibile prevedere quali siano questi filtri, dunque assumeremo di avere dei filtri il più generici possibili, ogni nodo esporterà tutta la sua conoscenza con tutti ed importerà e valuterà tutto ciò che riceverà, da qualsiasi fonte.

Abbiamo stabilito su quale grafo andremo ad implementare ed eseguire il nostro algoritmo, ma avremo bisogno di semplificare la concezione del nodo BGP le cui funzionalità sono moltissime. A partire dall'RFC originale [7] sono state prodotte molte RFC che aggiungono degli attributi o modificano delle componenti [21] [22] dunque utilizzeremo i seguenti vincoli:

- Sappiamo che in base al numero ed alle dimensioni delle destinazioni esportate un AS potrebbe essere considerato più importate di un'altro, come è giusto che sia. Per semplicità questo concetto non è stato applicato/studiato a fondo, nei nostri test consideriamo tutti gli AS uguali che condividono un carico indifferente dal numero/dimensione delle destinazioni, questo lascia spazio a studi futuri sull'argomento, con minime modifiche all'algoritmo;
- Un'altro concetto che abbiamo deciso di ignorare è l'aggregazione dei path, definito nell'RFC originale di BGP [7], che venivano passati al nodo successivo. Non consideriamo l'aggregazione ma invece prevediamo che in caso di destinazioni uguali raggiungibili attraverso più percorsi l'AS prenda la decisione di seguire un percorso invece dell'altro in base ai propri filtri o in base al proprio meccanismo decisionale interno a BGP.

4.2 Implementazione in BGP dell'algoritmo

L'algoritmo proposto è stato creato partendo dall'algoritmo 1 presente nella sezione 3.1.2. Bisognerà apportare delle modifiche alla RT gestita da BGP la voce *loadIn* verrà aggiunta dall'algoritmo 4.1:

Tabella 4.1: Routing Table estesa

Campo	notazione	Descrizione
Destinazione	d	identifica la destinazione presa in considerazione nella RT
Next-Hop	NH	il NH che ho scelto per raggiungere la destinazione d
Load in ingresso	$loadIn$	Dizionario che mappa ogni vicino u che appartiene ai PH per la destinazione d, il proprio load condiviso

Oltre ai campi aggiunti alla RT presentati dalla tabella 4.1 è stata aggiunta una struttura dati che viene mantenuta dallo speaker BGP ed è globale all'interno del peer.

Tabella 4.2: Struttura dati globale

Campo	notazione	Descrizione
AS-Load-dict	$ASLoadDict$	Viene utilizzata come dizionario per mantenere globalmente i carichi attuali degli AS conosciuti, all'interno viene mantenuto anche il carico dell'AS locale, è formato da elementi di questo tipo ($Load, ASLoadCounter$).

Per comodità l'algoritmo applicato è stato diviso nelle sue 3 fasi principali:

Algorithm 4.1: Inizializzazione BGP eseguito da v

```
1  Init:
2      foreach d ∈ v do
3          RT[d].NH = [];
4      ComplessiveLoad = 0;
5      ASLoadDict = { 'v.AS' : [ComplessiveLoad, 1]};
```

Nella prima fase dell'algoritmo 4.1 è presente l'inizializzazione della struttura dati e di ciò che verrà utilizzato all'interno del protocollo, le righe [2-3] indicano che per ogni destinazione appartenente al nodo v, andrà creato un elemento nella RT con una lista dei NH vuota. Nelle righe [4-5] viene creato il primo elemento all'interno di ASLoadDict, quello che rappresenterà l'as locale, $v.AS$ sta ad indicare il codice dell'AS locale e viene usato come chiave all'interno del dizionario.

la parte di algoritmo 4.1 andrà applicata quando lo speaker si troverà nello stato di "idle", le successive parti invece andranno applicate con una connessione già stabilità, lo stato corrispondente sarà "estabilished", riferito alla figura 2.1.

Algorithm 4.2: Parte dell'algoritmo relativa alla propagazione, eseguito da v

```
1  Ripeti ogni δ s:
2      ****** Aggiornamento ASLoadDict *******
3      if ASLoadDict(v.AS).Load ≠ ComplessiveLoad then
4          ASLoadDict.update(v.AS) = [ComplessiveLoad, actual + 1];
5      ***** Aggiornamento loadOut *******
6      foreach d ∈ RT() do
7          if v ∈ D then
8              loadOut = 1;
9          else
10             loadOut = 0;
11         foreach u ∈ RT[d].loadIn() do
12             loadOut+ = RT[d].loadIn[u];
13         //Update da inoltrare con gli attributi aggiuntivi, sfruttato dal route refresh 4.3
14         send < UpdateInfo, RT[d].NH, loadOut, ASLoadDict > to neighbors;
```

In questa seconda fase, algoritmo 4.2, sfruttando la ricondivisione degli update del protocollo ci occuperemo della distribuzione delle informazioni ai nostri vicini. Avremo due parti fondamentali prima di poter inoltrare il pacchetto ai vicini:

- L'aggiornamento del dizionario degli AS. Andrà aggiornato il load locale che sarà successivamente esportato. Nel caso in cui vi sia stato un aggiornamento del carico locale allora la struttura dati viene aggiornata, e viene aumentato anche il counter che si occupa di discriminare tra le informazioni più o meno recenti;
- Successivamente per ogni destinazione viene aggiornato il carico in uscita utilizzando quello in ingresso più il proprio, ponendo attenzione ad aggiungere un'unità di carico solamente se si appartiene all'insieme dei nodi destinazione D.

Algorithm 4.3: Integrazione con BGP eseguito da v

```
1  On recive < d, NH^u, loadOut^u, ASLoadList^u > from u do:
2      if d ∉ RT i filtri lo permettono then
3          RT[d].NH = u;
4      else
5          if Nuovo percorso ottimo per raggiungere d then
6              RT[d].NH = u;
7      ******* Gestione dei contributi di load *********
8      if v ∈ NH^u then
9          RT[d].loadIn[u] = loadOut^u/|NH^u|;
10     else
11         RT[d].loadIn.remove(u);
12     ****** Update del load locale ********
13     ComplessiveLoad = 0;
14     foreach d ∈ RT() − {v} do
15         foreach u ∈ RT[d].loadIn() do
16             ComplessiveLoad+ = RT[d].loadIn[u];
17     ******* Update degli ASLoad ricevuti *******
18     foreach asLoad ∈ ASLoadDict^u do
19         if asLoad ∉ ASLoadDict then
20             ASLoadDict.add(discovered asLoad);
21         else
22             if asLoad.latestThen(ASLoadDict(asLoad.key)) then
23                 ASLoadDict.update(asLoad.key) = asLoad;
```

Nella terza ed ultima fase 4.3 dell'algoritmo avremo la gestione dei messaggi ricevuti da parte dei nostri peer vicini. Alla riga 2 vi è una prima considerazione, nel caso in cui non conoscessimo la destinazione che ci viene inoltrata (ed i nostri filtri lo permettono) l'informazione va aggiunta alla RT ed il NH andrà impostato con il valore dello speaker u che ci ha inoltrato la rotta.

Al contrario, se invece quella destinazione è già presente nella RT andranno fatte delle valutazioni aggiuntive, BGP in base ai suoi filtri deciderà se utilizzare il path presente nel nuovo update come miglior path oppure mantenere quello vecchio, nel caso in cui prenda la prima decisione andrà aggiornato anche il NH presente nella RT. Tutto questo perchè abbiamo deciso di utilizzare un sistema monopath, viene considerato solamente il percorso migliore e non tutte le possibili variati, per non incorrere in situazioni di instabilità.

Dopo aver fatto ciò dovremo gestire il contributo ricevuto, nelle righe da 8 a 11 viene valutato il carico in ingresso e/o l'eventuale rimozione di esso. Successivamente andrà aggiornato il carico locale, sommando tutti i carichi in ingresso, provenienti dai PH, che vengono rilevati scorrendo la RT. Come ultimo punto aggiorniamo il dizionario con i carichi degli AS che ci sono stati passati, e manteniamo solamente il valore più aggiornato per ogni AS presente nella nostra struttura dati globale. Utilizziamo

questo sistema di condivisione dei carichi per fare in modo che venga propagato anche il carico dei nodi che non condividono destinazioni.

NH singolo

Una nota importante da fare per quanto riguarda l'algoritmo 4.3 riguarda le righe dalla 2 alla 6, come si può notare queste sono le uniche righe in cui viene gestito il valore del NH all'interno della struttura dati, ed i NH non si sommano in una lista incrementale, ma bensì si sostituiscono l'uno con l'altro. Questo perché è stato deciso di mantenere solamente il NH appartenente al percorso ottimo per quella destinazione d.

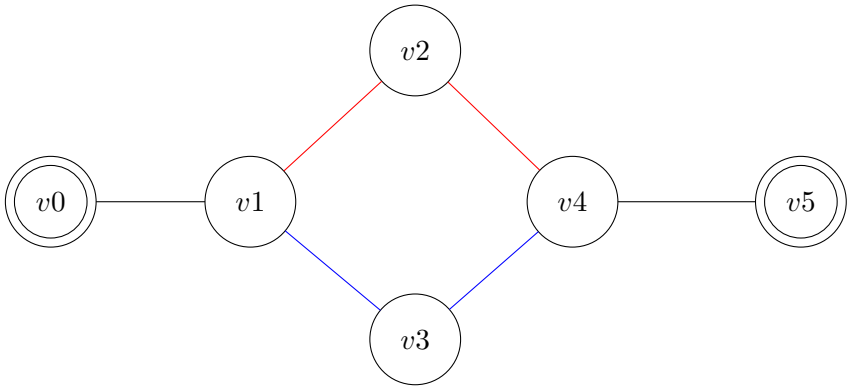


Figura 4.1: Più path possibili con lo stesso numero di hop

Come illustrato in figura 4.1 potrebbero esserci due cammini a costo minimo che conducano verso $v5$ partendo da $v0$. Il primo, mostrato dal cammino rosso, passerà attraverso il vertice $v2$ mentre il secondo cammino, quello blu, passerà attraverso $v3$.

Il nodo BGP $v1$ attuerà le proprie politiche ed inserirà nella RT solamente il percorso che considererà migliore, ed il traffico verrà instradato solamente verso quella destinazione, per questo motivo è stato deciso di non supportare la presenza di NH multipli.

Convergenza dell'algoritmo

Vi saranno essenzialmente tre momenti di convergenza rilevabili all'interno della nuova implementazione in bgp, evidenziati dalla figura 4.2.

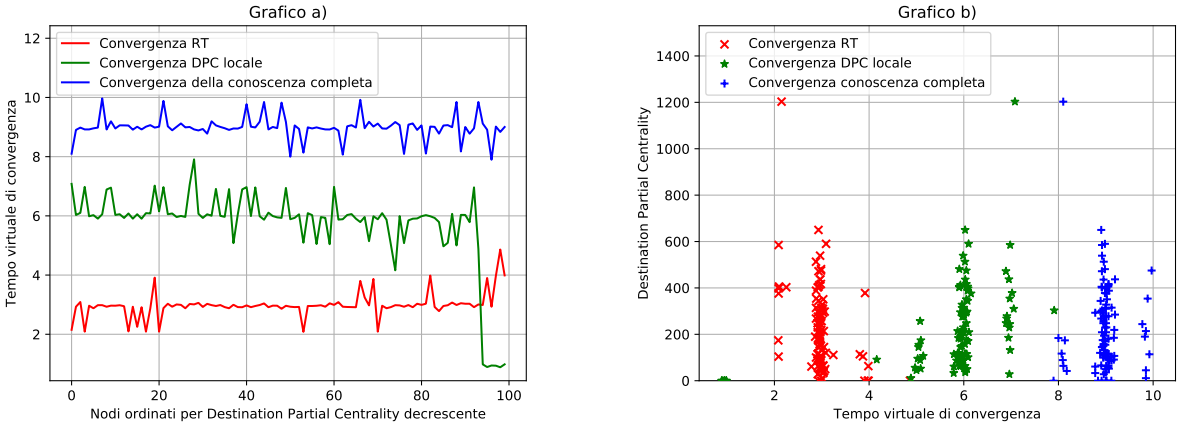


Figura 4.2: Confronto convergenza della centralità

Il primo momento evidenziato in rosso, nei grafici in figura 4.2, riguarda la convergenza della RT, una volta che questa non varierà più significherà che tutti sapranno come raggiungere i nodi destinazione. Successivamente vi sarà il calcolo del valore locale di centralità, nella figura 4.2, grafico

a, nella parte finale avremo un picco, questo è dovuto alla presenza di alcuni nodi con centralità 0, si possono notare anche nel grafico b, in cui alcuni punti sono molto vicini allo 0, se la loro centralità è 0. Il terzo momento di convergenza riguarda la conoscenza completa di tutte le centralità corrette di tutti i nodi, questa è dovuta allo scambio di informazioni tra i nodi.

Le informazioni della figura 4.2 sono state ottenute da un grafo di Erdos di diametro 6 contenente 100 nodi di cui il 100% condivideva destinazioni.

4.3 Route Refresh

Il protocollo BGP come abbiamo visto nella sezione 2.2.1 non permette ad un peer di generare ulteriori messaggi di update a meno che non vengano scoperti dei cammini migliori per raggiungere delle destinazioni, il che però potrebbe impedire all'algoritmo descritto nella sezione precedente 4.2 di raggiungere la conoscenza completa e corretta della centralità degli altri nodi, Per questo motivo si è reso necessario l'utilizzo del route refresh.

Il route refresh viene utilizzato in modo leggermente improprio rispetto alla definizione data dall'RFC originale [23]. Questa funzionalità prevede l'inoltro delle informazioni riguardanti le destinazioni conosciute attraverso pacchetti di UPDATE ai vicini anche se sono già aggiornati. Il suo scopo originale era quello di permette ad un nodo, nel caso in cui cambiasse i propri filtri di rivalutare le destinazioni provenienti dai suoi vicini, in modo tale che potesse variare la propria conoscenza in base ai nuovi filtri.

Forziamo un'operazione di route refresh senza cambiare i filtri, verranno inoltrate tutte le informazioni attuali di un nodo tramite messaggio di UPDATE, il che forzerà l'utilizzo del codice dell'algoritmo 4.2, in particolar modo la riga 14, con gli attributi di centralità aggiornati, fino a raggiungere la convergenza della struttura dati ASLoadDict per tutti i nodi facenti parte della topologia. Quindi ogni nodo della nostra topologia, ad intervalli regolari richiederà a tutti i suoi vicini di effettuare un route refresh verso di lui, in modo che lui possa aggiornare le proprie strutture dati.

Questa operazione potrebbe essere eseguita per esempio una volta al giorno con i proprio peer vicini, in modo da ottenere nuovi aggiornamenti sulla centralità degli altri AS.

5 Esperimenti

Per poter validare la modifica proposta nel capitolo precedente sono stati prodotti degli esperimenti che implementano l'algoritmo per il calcolo della centralità distribuito descritto nella sezione 4.2.

5.1 L'ambiente di sviluppo

L'ambiente di sviluppo si compone essenzialmente di tre elementi:

- **NePa Test** ambiente di gestione e generazione dei test [4];
- **Mininet** Ambiente di virtualizzazione di reti;
- **Bird** Demone per simulare il protocollo;
- Simulatore ad eventi discreti.

Figura 5.1: Gerarchia implementativa

I primi 3 ambienti cooperano per raggiungere il risultato, gli ambienti più interni sono controllati dal modulo più esterno che configura e gestisce l'andamento del componente sottostante come rappresentato in figura 5.1, il modulo più esterno deve essere configurato appositamente dal programmatore.

5.1.1 NePa

NePa Test [4] è un ambiente utilizzato per eseguire test su delle topologie di rete, una volta decisa la topologia NePa sarà in grado di creare degli ambienti virtuali per ogni nodo assegnandovi indirizzi ip e collegamenti in modo che sia possibile poi eseguire comandi su di ogni singolo nodo. Viene utilizzato Mininet per virtualizzare la rete in base alla topologia proposta.

L'ambiente può essere configurato dal programmatore tramite dei file di inizializzazione che forniscono le specifiche, come per esempio che file topologico utilizzare, che test eseguire ed altri parametri che possono essere ricavati all'interno del test.

I file topologici possono essere scritti manualmente tramite file .edges oppure tramite file .json opportunamente costruiti. Una volta ottenuta la topologia NePa avvia i nodi e li collega secondo la specifica, a questo punto ogni nodo avrà un proprio ambiente virtuale in esecuzione.

Ora inizierà il test, si compone di codice Python opportunamente scritto per eseguire lo studio necessario, grazie alla comunicazione con l'host tramite mininet possiamo controllare tutti gli host e lanciare comandi specifici per ciascuno di essi.

Prima di poter eseguire tutti i demoni che comunicheranno all'interno del nostro ambiente sarà necessario creare tutti i file di configurazione necessari, che andranno scritti dinamicamente in modo da eseguire *Bird* su di ogni nodo, All'interno vi saranno informazioni riguardanti il singolo nodo, come per esempio l'indirizzo di una interfaccia ed l'interfaccia con cui si è collegati, come si può vedere nella figura di esempio 5.2.

```
protocol bgp h0_0_h1_1 {
    local 10.0.0.1 as 1;
    neighbor 10.0.0.2 as 2;
    hold time 3;
    ipv4{
        import filter bgp_in;
        export all;
    };
    direct;
}
```

Figura 5.2: Esempio file di configurazione

Una volta avviato il demone che eseguirà la versione modificata di BGP sarà sufficiente attendere il tempo necessario al raggiungimento della convergenza per poi poter valutare i risultati ottenuti, che verranno salvati su degli appositi file di log.

5.1.2 Bird

Bird è un demone che permette di eseguire vari protocolli di rete, può essere inserito all'interno dell'ambiente di rete per gestire messaggi provenienti dall'esterno. È un software distribuito tramite licenza GNU [1] [2], ed al suo interno presenta il supporto per i seguenti protocolli:

- Both IPv4 and IPv6;
- Multiple routing tables;
- BGP;
- OSPF;
- BFD;
- Ecc..

È stato scelto bird perchè presenta un supporto per moltissimi RFC riguardanti BGP ed in più ha una gestione migliore della ram rispetto ai suoi concorrenti, la community è relativamente attiva per fornire supporto.

Essendo però un progetto open source presenta delle limitazioni dovute alla motivazione degli utenti che decidono di scrivere delle funzionalità ed implementarle all'interno di esso, non tutto degli RFC è supportato, per esempio non vi è alcun supporto per l'aggregazione delle rotte, ed in più alcuni timer sono mancanti, dato il dibattito della comunità sul loro utilizzo. Ma per il nostro progetto fornisce tutto ciò di cui abbiamo bisogno.

5.1.3 Simulatore ad eventi discreti

Per le simulazioni su larga scala è stato utilizzato un simulatore ad eventi discreti in Python che implementasse tutti i messaggi di UPDATE che sarebbero stati scambiati tra i vari nodi di una topologia data. Il simulatore all'interno della logica di valutazione e inoltro dei messaggi di UPDATE include il codice presente nell'algoritmo 4.1, 4.2 e 4.3.

Il simulatore utilizza una propria temporizzazione degli eventi, basati su di un timer virtuale $\vartheta = 1$ e convergerà al risultato dopo un certo periodo $k\vartheta$, al timer di generazione dei pacchetti viene aggiunta un piccola variazione casuale per evitare una perfetta sincronia.

Una simulazione termina solamente quando l'ultimo nodo raggiunge la convergenza, il che avviene quando il suo stato non varia ulteriormente. Lo stato di un nodo è composto dalla sua RT dal proprio carico calcolato localmente di DPC e la conoscenza relativa alla DPC di tutti gli altri nodi.

5.1.4 Topologie utilizzate

Inizialmente sono state utilizzate topologie semplici, che permettevano un rapido controllo dei risultati ottenuti con quanto ci si aspettasse, le topologie sono visibili in figura 5.3.

Su queste topologie sono stati poi decisi i nodi che sarebbero appartenuti all'insieme D, dovranno essere scelti almeno due nodi per topologia, i nodi evidenziati in blu nella figura 5.3 sono quelli scelti, dopo un paio di minuti di simulazione l'algoritmo sarà arrivato a convergere ad un risultato, che confronteremo con ciò che ci viene restituito dal simulatore ad eventi discreti, tabella 5.1.

Tabella 5.1: Risultati dalle topologie semplici

Topologia	Risultato Aspettato (Sim. Ev. Disc.)	Risultato Ottenuto (NePa)
Topologia a	$\{v_0, v_1, v_2, v_3, v_4 : 0, 2, 2, 2, 0\}$	$\{v_0, v_1, v_2, v_3, v_4 : 0, 2, 2, 2, 0\}$
Topologia b	$\{v_0, v_1, v_2, v_3, v_4 : 0, 12, 0, 0\}$	$\{v_0, v_1, v_2, v_3, v_4 : 0, 12, 0, 0\}$
Topologia c	$\{v_0, v_1, v_2, v_3, v_4 : 0, 2, 0, 2, 0\}$	$\{v_0, v_1, v_2, v_3, v_4 : 0, 2, 0, 2, 0\}$

[1] Bird site

[2] GNU General Public License

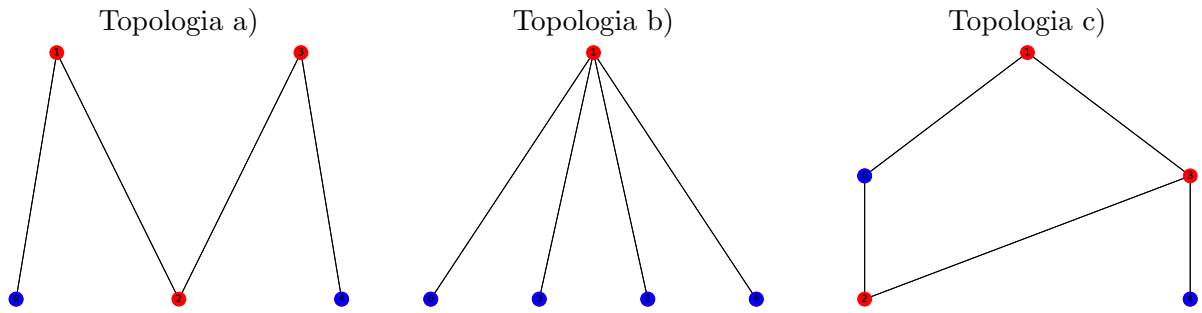
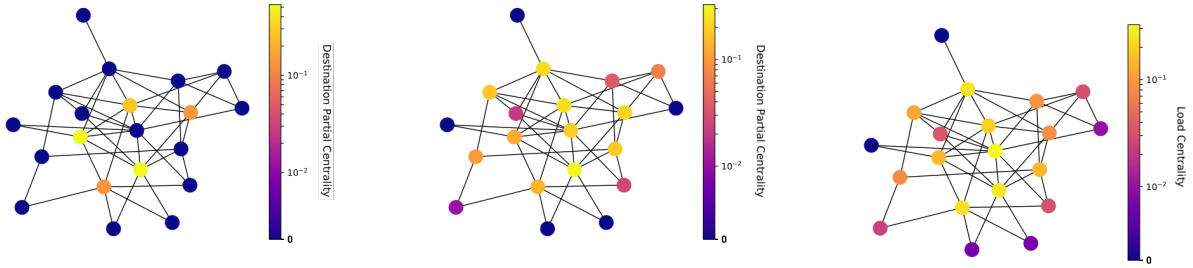


Figura 5.3: Topologie semplici

Simulazioni su larga scala

Una volta arrivati ad uno stato accettabile nello sviluppo dell'implementazione dell'algoritmo all'interno del demone si è passati a valutare topologie più complesse, e più simili alla realtà, ottenute dalla produzione di topologie attraverso software come NetworkX[3]. Tutti i risultati seguenti, presentati nelle figure 5.5, 5.6 e 5.7, sono stati ottenuti attraverso il simulatore ad eventi discreti.



30% 100%

Figura 5.4: Topologie colorate in base alla centralità

Nelle figura 5.4 possiamo notare 3 grafi di Erdos di diametro 4, contenti 20 nodi, colorati in base ai valori di centralità normalizzati, nelle prime due immagini è stata calcolata la DPC con due diverse composizioni di nodi destinazione, nella prima solamente il 30% dei nodi apparteneva all'insieme D, mentre nella seconda il 100% dei nodi apparteneva a D. Successivamente i valori di centralità sono stati normalizzati secondo la formula di normalizzazione della DPC 3.1.1. Nella terza immagine è presente invece lo stesso grafo su cui però è stata calcolata la LC normalizzata utilizzando la formula presentata in 2.3.1.

Tutti i seguenti risultati (figure 5.5, 5.6, 5.7) rappresentano ciò che è stato ottenuto applicando il calcolo della DPC ad un grafo di Erdos di diametro 6 contenente 100 nodi, sono stati eseguiti diversi esperimenti con percentuali di nodi appartenenti a D differenti, e per ogni percentuale sono stati eseguiti 50 esperimenti, successivamente i valori di ogni singolo test sono stati utilizzati per ricavare dei risultati medi per ogni percentuale.

Tutti questi esperimenti rappresentati dalla figura 5.5 mostrano il calcolo della DPC applicato al grafo, e come la variazione del numero di nodi appartenenti a D influisca sulla quantità di carico distribuita.

Nella figura 5.6 viene presentato un confronto tra la LC calcolata su tutto il grafo rispetto alla DPC calcolata sfruttando la sola percentuale di nodi destinazione presenti. Il grafo utilizzato è lo stesso della figura 5.5 e si utilizza sempre la media data da 50 esperimenti con percentuale di nodi destinazione ottenuta casualmente dall'insieme V.

Nella figura 5.7 vengono normalizzate le centralità confrontate e presentate nella figura 5.6. La normalizzazione è stata calcolata per la LC secondo la formula 2.3.1 mentre la DPC è stata normalizzata attraverso 3.1.1.

[3] `https://networkx.github.io/`

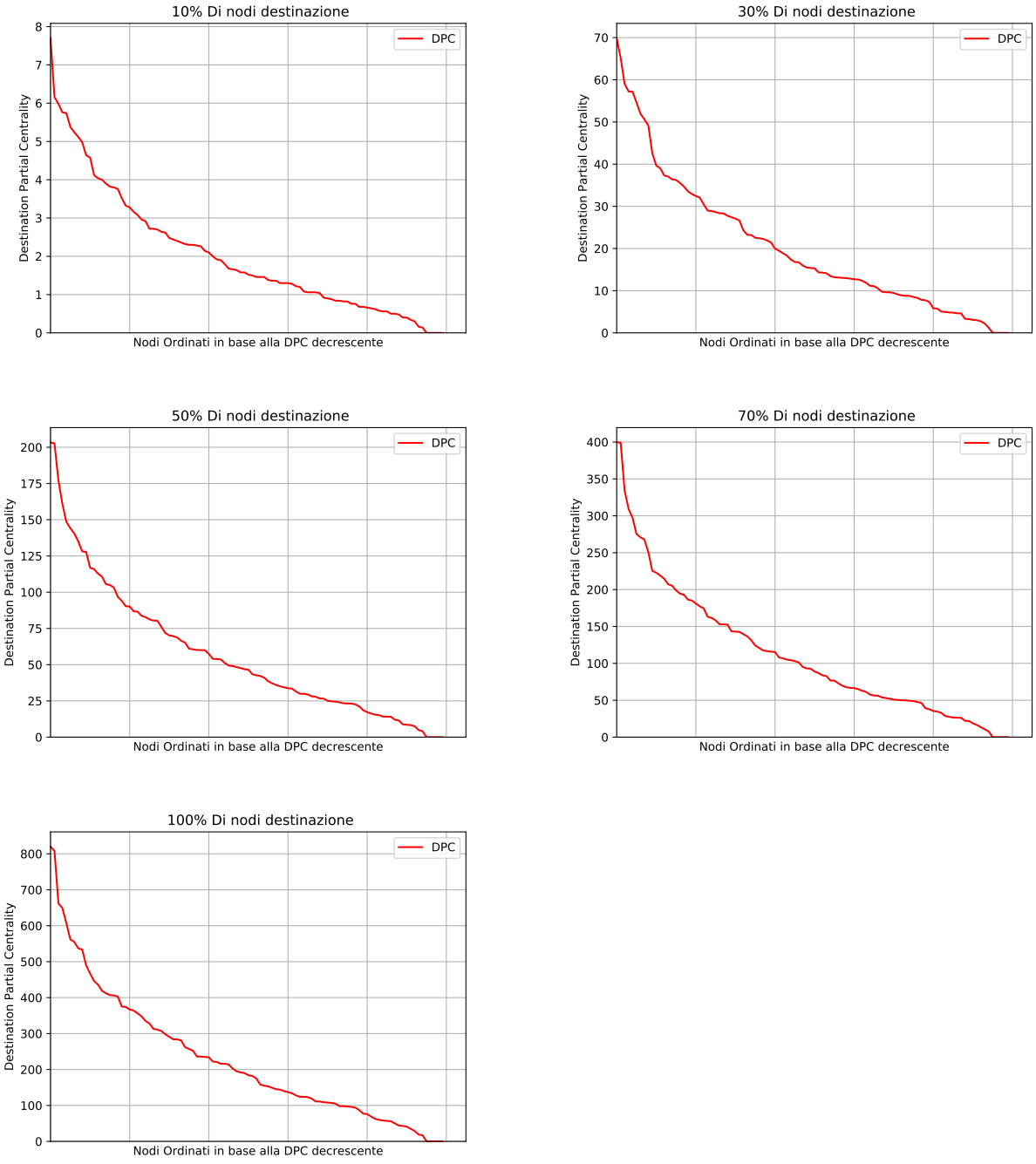


Figura 5.5: DPC In base alla percentuale di nodi

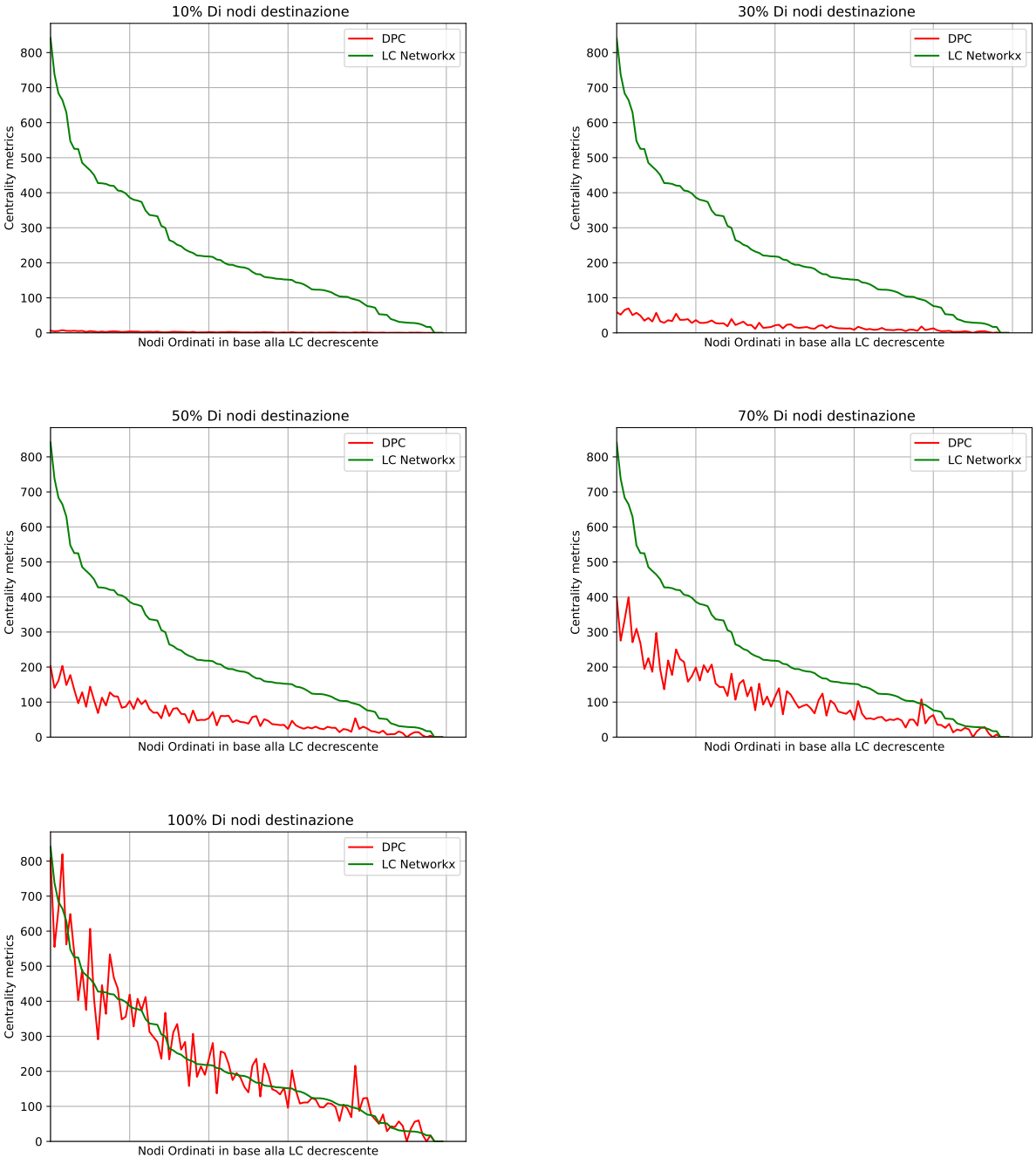


Figura 5.6: DPC VS LC In base alla percentuale di nodi

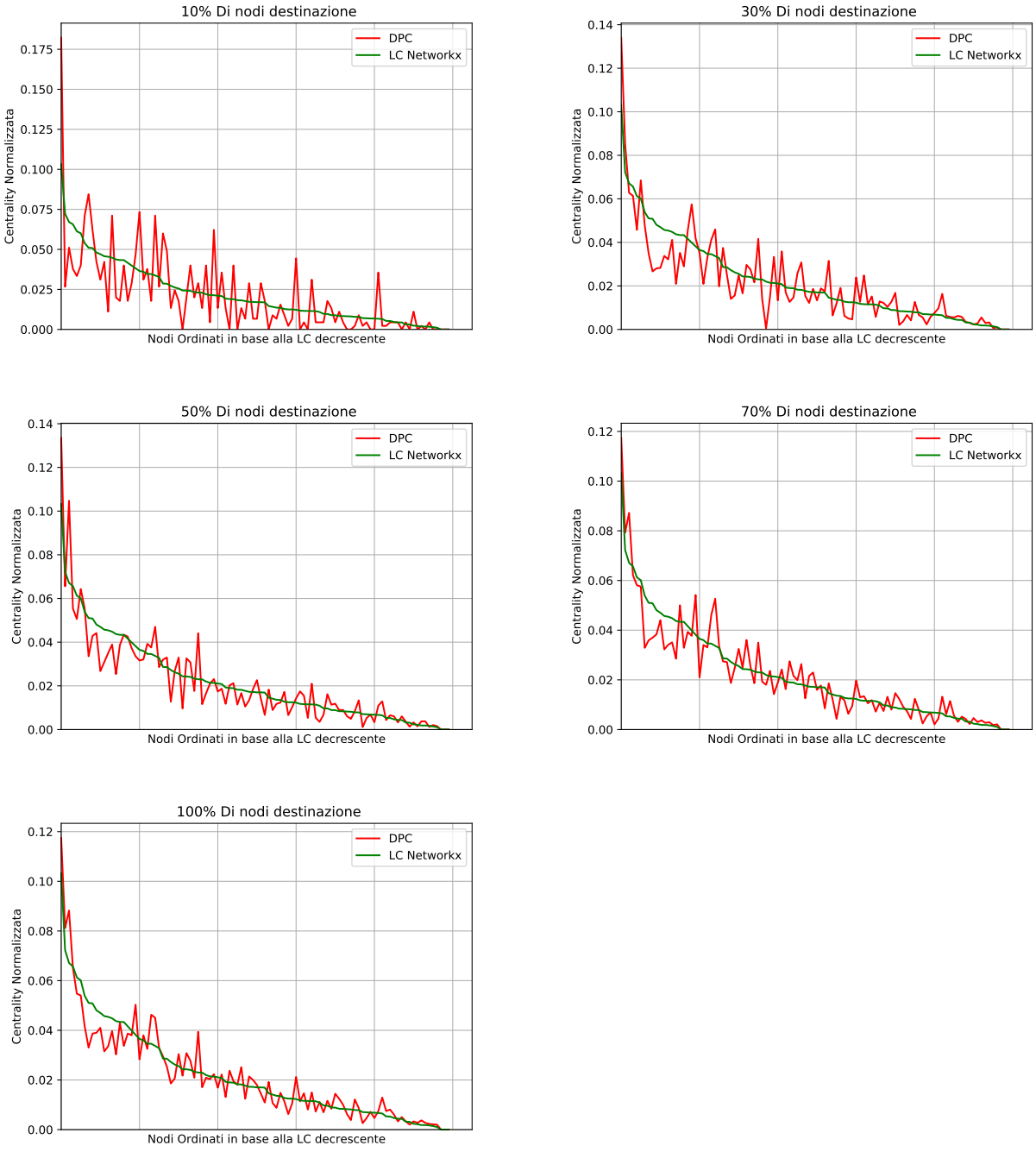


Figura 5.7: DPC vs LC In base alla percentuale di nodi, normalizzata

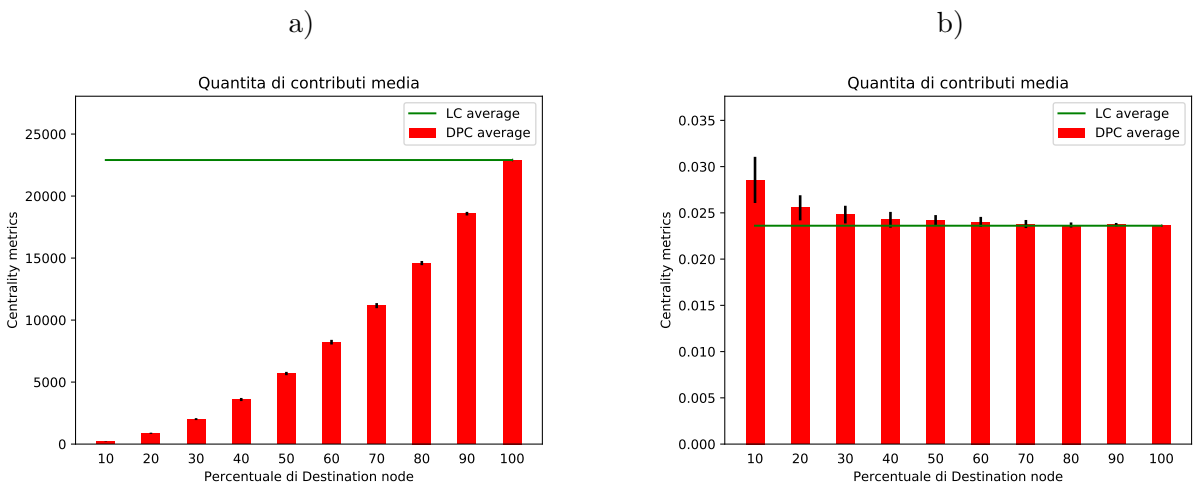


Figura 5.8: DPC vs LC considerando la quantità di contributi

5.2 Commenti e considerazioni

Dalla figura 5.4 si può notare la somiglianza tra il grafo mappato attraverso la DPC ed il grafo che utilizza la LC. Le uniche differenze sono dovute alla percentuale di Destination node utilizzati ed al fatto che la LC calcolata prevedeva l'utilizzo del multipath, cosa che invece il nostro algoritmo applicato a BGP non fa, si ricorda la sotto sezione 23. La LC nel caso in cui si trovi ad avere più percorsi con la stessa preferenza per raggiungere la destinazione dividerà la quantità di traffico in modo equo tra i diversi percorsi, mentre nella nostra implementazione della DPC questo non viene fatto.

All'interno della figura 5.7 si può notare come l'andamento in base alla percentuale di peer che esportano le proprie rotte tenda ad avvicinarsi all'andamento della LC con percentuali elevate. Il fatto che le due curve non convergano ad un'unica curva è dovuto all'applicazione del monopath, dunque è un problema dipendente dalla definizione, ma tutto funziona correttamente in BGP anche senza lo splitting dei contributi.

Su topologie che non contemplino il multipath infatti la nostra implementazione della DPC converge alla LC.

Se calcolassimo l'integrale al di sotto delle curve della DPC nella figura 5.6 potremmo notare un andamento come quello presentato nel grafico a della figura 5.8, il grafo considerato è lo stesso dei test precedenti, sono stati ripetuti, per ciascuna percentuale di Destination node, 50 test scegliendo casualmente i nodi che avrebbero condiviso delle destinazioni, successivamente è stata calcolata la quantità media di carico distribuita per ciascuna simulazione, e ciò che viene presentato nell'istogramma sono le medie della quantità di carico derivanti da tutte le simulazioni.

La particolarità sta nel fatto che nel grafico a della figura 5.8 la quantità di carico media trasmessa all'interno della rete da parte della DPC tende alla quantità trasmessa dalla LC. Questo perchè sia nella LC che nella DPC la quantità di traffico che un nodo inoltra per ogni altra destinazione è pari ad 1, solamente che nel caso di più cammini la LC separa il carico, mentre nella nostra implementazione della DPC questo non viene fatto.

Nel grafico b della figura 5.8 possiamo notare come anche normalizzando i contributi che vengono trasmessi, all'aumentare della quantità di nodi appartenenti all'insieme D la quantità di carico complessiva trasmessa da parte della DPC tende a quella trasmessa dalla LC. Dunque l'unica differenza sta nel come viene spartita questa quantità di carico, come possiamo vedere dalle immagini presenti nella figura 5.7 infatti da parte della DPC alcuni nodi vengono considerati più centrali di altri a cui invece la LC assegnerebbe un valore più alto. Tutto questo è dovuto sempre al fatto che l'implementazione della DPC applicata non considera il multipath, ma ciò non comporta una mancanza dal punto di vista funzionale in BGP.

Tutto il codice prodotto per gli esperimenti è presente nel seguente ropository: Redmine BGP repo

6 Conclusioni

I risultati ottenuti ci mostrano che la DPC potrebbe essere una buona metrica di centralità per protocolli che prevedono delle situazioni particolari come quella presa in considerazione da BGP, permetterebbe di avere una percezione migliore rispetto alla LC dei nodi che sono effettivamente centrali all'interno della rete.

All'interno della tesi sono stati posti diversi vincoli, che se rilassati potrebbero portare a situazioni di malfunzionamento o che porterebbero al completo non funzionamento.

Unificazione eBGP router

Per permettere il funzionamento dell'algoritmo all'interno di un AS con più eBGP router bisognerebbe modificare la parte relativa alla condivisione delle rotte all'interno dell'AS, il che è fattibile, nei pacchetti di update destinati all'interno della rete andrebbero aggiunte le strutture dati che vengono inviate verso l'esterno. Ma se non vengono apportate queste modifiche i due peer interni non entreranno a conoscenza delle informazioni mantenute dall'altro, se non attraverso domini esterni che ricondividono le informazioni.

Filtri di condivisione

Nel caso in cui vi fossero filtri più mirati all'interno dell'AS per determinate sessioni l'algoritmo continuerebbe a funzionare, ma per certe topologie con determinate configurazioni di filtri non si potrà raggiungere la convergenza ed in più le informazioni di centralità saranno parziali. Alcune porzioni di rete potrebbero non entrare mai a conoscenza dell'esistenza di alcune destinazioni. Il che porterebbe ad un calcolo della DPC non accurato.

Aggregazione dei path

Se considerassimo l'aggregazione l'algoritmo continuerebbe a funzionare perchè non utilizza il path derivante dal pacchetto di update. Ma potrebbero presentarsi problemi concettuali, come considero il NH se il mio prossimo passo contiene una aggregazione?

6.1 Future works

Questa tesi apre le porte a tanti possibili studi, sia per quanto riguarda l'applicazione di questa metrica di centralità che l'utilizzo dei risultati ottenuti su BGP. come viene detto nella sezione 1.3 la comunità scientifica non ha raggiunto delle decisioni per quanto riguarda alcuni aspetti, come per esempio l'impiego di un timer detto *Minimum Route Advertisement Interval* (MRAI) [24] che potrebbe portare a dei ritardi esponenziali nella propagazione delle informazioni [25] se impostato in modo non corretto. L'utilizzo invece della centralità per controllarlo in modo dinamico potrebbe portare a dei miglioramenti, si potrebbero configurare dei timer specifici per sessioni, facendo in modo che le informazioni provenienti da fonti più centrali si propaghino più velocemente, come da lavori già svolti in quest'ambito [1], un primo studio è presente nell'allegato A.

Oltre a questo un prossimo studio potrebbe essere la valutazione del livello di adattabilità nel protocollo modificato, per esempio alla morte di un nodo, il tempo impiegato per ritornare a convergere alla DPC con il nodo mancante.

Un ulteriore campo esplorativo potrebbe essere confrontare la centralità calcolata dalla DPC con altre metriche, per esempio la stress centrality [19], e/o modificare l'algoritmo in modo da puntare alla convergenza con questa metrica.

Bibliografia

[1] L. Maccari and R. L. Cigno, "Pop-routing: Centrality-based tuning of control messages for faster route convergence," pp. 1–9, 2016.

[2] L. Maccari, L. Ghiro, A. Guerrieri, A. Montresor, and R. L. Cigno, "On the Distributed Computation of Load Centrality and Its Application to DV Routing," in *IEEE International Conference on Computer Communications (INFOCOM), Honolulu, USA*, 2018.

[3] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of mathematical sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[4] L. Baldesi and L. Maccari, "NePA TesT: network protocol and application testing toolchain for community networks," in *Wireless On-demand Network Systems and Services (WONS), 2016 12th Annual Conference on*. IEEE, 2016, pp. 1–8.

[5] J. Hawkinson and T. Bates, "Guidelines for creation, selection, and registration of an Autonomous System (AS)" rfc1930," *BBN Planet/MCI, March*, p. 10, 1996.

[6] D. Mills, "Exterior Gateway Protocol Formal Specification, DARPA Network Working Group Report RFC-904," M/A-COM Linkabit, Tech. Rep., 1984.

[7] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," Tech. Rep., 2005.

[8] P. Zilberman, R. Puzis, and Y. Elovici, "On network footprint of traffic inspection and filtering at global scrubbing centers," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 5, pp. 521–534, 2017.

[9] P. Pantazopoulos, M. Karaliopoulos, and I. Stavrakakis, "Distributed placement of autonomic internet services," *IEEE transactions on parallel and distributed systems*, vol. 25, no. 7, pp. 1702–1712, 2014.

[10] D. Katsaros, N. Dimokas, and L. Tassiulas, "Social network analysis concepts in the design of wireless ad hoc network protocols," *IEEE network*, vol. 24, no. 6, 2010.

[11] M. Kas, S. Appala, C. Wang, K. M. Carley, L. R. Carley, and O. K. Tonguz, "What if wireless routers were social? approaching wireless mesh networks from a social networks perspective," *IEEE Wireless Communications*, vol. 19, no. 6, 2012.

[12] S. P. Borgatti, "Centrality and network flow," *Social networks*, vol. 27, no. 1, pp. 55–71, 2005.

[13] S. P. Borgatti and M. G. Everett, "A graph-theoretic perspective on centrality," *Social networks*, vol. 28, no. 4, pp. 466–484, 2006.

[14] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[15] D. Estrin, Y. Rekhter, and S. Hotz, "A Unified Approach to Inter-Domain Routing," Tech. Rep., 1992.

[16] Y. Rekhter and T. Li, "RFC 1771," *A Border Gateway Protocol*, vol. 4, pp. 1–54, 1995.

[17] S. Dolev, Y. Elovici, and R. Puzis, "Routing betweenness centrality," *Journal of the ACM (JACM)*, vol. 57, no. 4, p. 25, 2010.

[18] R. Puzis, P. Zilberman, Y. Elovici, S. Dolev, and U. Brandes, "Heuristics for speeding up betweenness centrality computation," in *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*. IEEE, 2012, pp. 302–311.

[19] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation," *Social Networks*, vol. 30, no. 2, pp. 136–145, 2008.

[20] K.-I. Goh, B. Kahng, and D. Kim, "Universal behavior of load distribution in scale-free networks," *Physical Review Letters*, vol. 87, no. 27, p. 278701, 2001.

[21] Q. Vohra and E. Chen, "BGP support for four-octet Autonomous System (AS) number space," 2012.

[22] J. Dong, M. Chen, and A. Suryanarayana, "Subcodes for BGP Finite State Machine Error," Tech. Rep., 2012.

[23] E. Chen, "Route refresh capability for BGP-4," 2000.

[24] P. Jakma, "Revisions to the bgp'minimum route advertisement interval'," 2011.

[25] A. Fabrikant, U. Syed, and J. Rexford, "There's something about MRAI: Timing diversity can exponentially worsen BGP convergence," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 2975–2983.

Ringraziamenti

Indubbiamente un ringraziamento speciale va a Giulia, che durante questi 3 anni di università mi ha accompagnato, supportato e sopportato non poco, è sempre stata un punto di riferimento per me. Vorrei poi ringraziare la mia famiglia, per tutto l'amore che mi hanno dato e per avermi spinto a proseguire il mio percorso di studi fino a raggiungere questo importante traguardo, senza di loro non sarei qui.

Indubbiamente ringrazio gli amici, quelli che da più di 10 anni mi accompagnano, quelli che mi hanno sempre aiutato, che c'erano nei momenti difficili, quelli che bastava un messaggio e si presentavano sotto casa per andare a bere e parlare assieme, quelli con cui ho vissuto innumerevoli avventure e sbronze. Ma allo stesso tempo voglio ringraziare anche tutti gli amici che ho incontrato qui a Trento, gli aperitivi, le nostre cene tipiche, i nostri pomeriggi passati in saletta, tra un'imprecazione ed un esame, un pomeriggio al lago ed una serata a ballare, una birra ed un cocktail.

Grazie a tutti voi per avermi reso ciò che sono oggi, ed essere le splendide persone che siete.

Per ultimi ma non per importanza vorrei ringraziare tutto il team ANS, in particolar modo il Prof. Renato Lo Cigno, il mio relatore, L'Ass. Prof. Leonardo Maccari per aver reso possibile questa tesi ed avermi proposto questo progetto che è stato per me estremamente istruttivo, ed un ringraziamento speciale a Lorenzo Ghiro ed alla sua pazienza nei miei confronti, il cui supporto è stato fondamentale per la comprensione e lo sviluppo di questa tesi.

Allegato A Implementazione del Timer MRAI con Pop-Routing

A.1 Timer MRAI

Il timer MRAI è un argomento caldo all'interno della comunità scientifica perché non si è ancora arrivati ad un comune accordo sul suo utilizzo, che se configurato erroneamente potrebbe portare a dei ritardi di propagazione esponenziali [25]. Il che ha portato lentamente all'abbandono di questo timer ed il suo inutilizzo sia per i messaggi di withdraw che per i messaggi di UPDATE in cui è stata lasciata più libertà per la configurazione da parte dell'amministratore.

A.1.1 Funzionalità del timer

Il timer MRAI è nato per diminuire la mole di messaggi che potrebbero essere inviati per aggiornare i propri vicini sulla situazione di alcune rotte, non potrà essere inviato un messaggio riguardante una certa destinazione prima che questo timer sia scaduto. In questo modo se mentre il timer è ancora attivo trovo un percorso migliore per quella destinazione allo scadere del timer mi basterà inviare un singolo messaggio contente il percorso migliore per raggiungere la destinazione. Non viene più utilizzato per i messaggi di withdraw per fare in modo che in caso di malfunzionamento le informazioni si propaghino più velocemente.

Questa soluzione del non considerare il timer ovviamente può portare a delle situazioni in cui si hanno molti messaggi di UPDATE sul canale che magari potrebbero essere evitati, ed il ricevente dovrà comunque interpretare quei messaggi per poi ogni volta scartare le modifiche al percorso per utilizzare solamente le informazioni contenute nel messaggio di UPDATE più recente.

A.2 Obbiettivo

L'obbiettivo a cui si punta è quello di poter applicare l'approccio Pop-Routing [1] sfruttando la metrica di centralità DPC per poter settare correttamente il timer MRAI in modo da non creare ritardi esponenziali, ma di garantire una propagazione delle informazioni più veloce se proveniente da nodi centrali e più lentamente se a distribuire l'aggiornamento sono i nodi marginali della rete. Applicare inoltre questo approccio per migliorare la convergenza del protocollo nel caso di malfunzionamenti.

A.3 Sviluppo attuale

Attualmente sto sviluppando una modifica del demone Bird 5.1.2 per permettere l'utilizzo di questo timer all'interno del protocollo BGP in quanto il disaccordo della comunità scientifica aveva portato alla sua mancata implementazione. Come primo test è stato implementato un timer MRAI parziale, in quanto non è stato creato un timer per ogni singola destinazione conosciuta, dato che la mole di lavoro per poterlo implementare non avrebbe permesso di inserire del materiale prima della data ultima di presentazione di questa tesi, ma invece è stato inserito un timer sulle singole connessioni attive.

Dunque potremmo ridefinire il timer come *Minimum Connection Advertisement Interval* (MCAI), che impedisce l'inoltro di pacchetti di UPDATE su di una determinata connessione/sessione BGP prima della sua scadenza. Con la consapevolezza che una nuova destinazione, di cui si entra a conoscenza prima della fine del timer, non verrebbe ricondivisa fino alla scadenza successiva. Per ovviare a questa problematica il timer andrà implementato come definito dall'RFC 4271 [7] in cui il timer MRAI va ad agire direttamente sulle destinazioni conosciute.

Dal punto di vista implementativo di Bird, il timer attualmente è all'interno della struttura dati che controlla la connessione agli altri peer, viene inizializzato quando la connessione diventa attiva, stato estabilished. I pacchetti all'interno di Bird hanno due stati, non schedulato e schedulato, i pacchetti

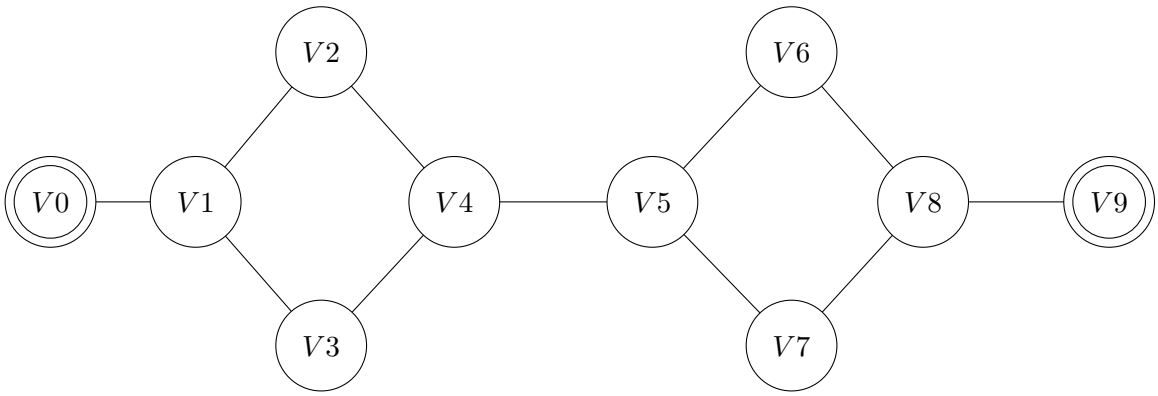


Figura A.1: Semplice topologia di test

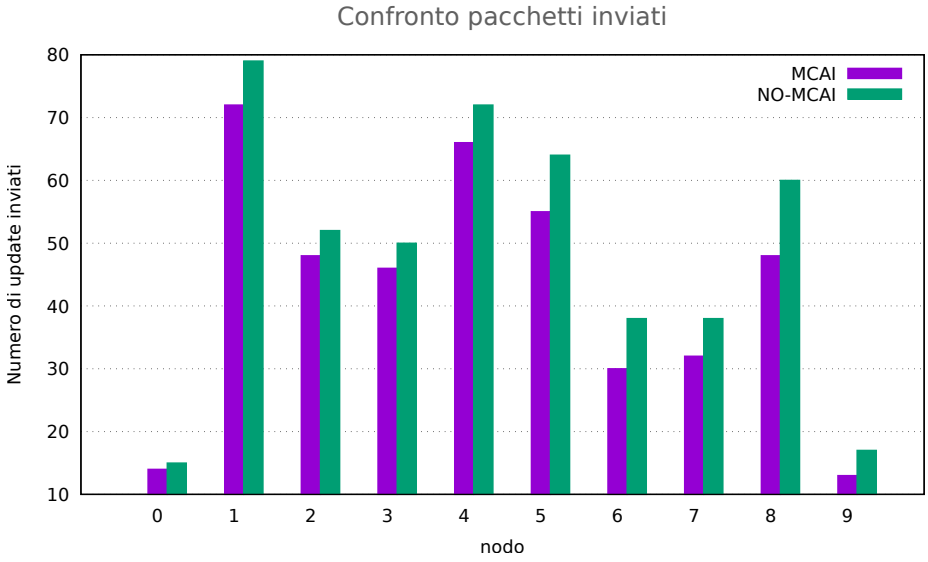


Figura A.2: Risultati sulla topologia di test

schedulati sono pronti per essere inoltrati mentre quelli non schedulati non vengono considerati per l'inoltro. Il timer va ad agire su questo punto, i pacchetti di UPDATE potranno entrare nello stato di schedulati solamente se il timer è scaduto al momento della valutazione, altrimenti i pacchetti verranno comunque aggiunti al buffer ma non verranno segnalati come inoltrabili.

Ora il timer è stato fissato a 5 secondi, non vi è ancora l'adattamento in base alla centralità per motivi di tempo implementativo. I test eseguiti già permettono di avere però dei risultati, riguardanti il numero di pacchetti inoltrati, la convergenza ai valori di centralità viene sempre raggiunta in ogni caso, ma quanti pacchetti di UPDATE sono necessari per raggiungerla?

Il test è stato eseguito su di una rete semplice dalla dimensione di 10 nodi con alcuni percorsi multipli come visibile in figura A.1

Su questa topologia sono stati presi due nodi appartenenti all'insieme D, ovvero il nodo $V0$ ed il nodo $V9$. Dopo di che sono stati confrontati il numero di pacchetti di UPDATE necessari a raggiungere la convergenza, tra il demone BGP con la modifica relativa all'MCAI ed il demone senza questa modifica, il confronto è visibile in figura A.2.

Come si può notare dalla figura A.2 già su una topologia così piccola con un timer fisso si hanno dei leggeri vantaggi sul numero di pacchetti inoltrati, mediamente si può considerare un guadagno di circa l'8/9% rispetto al numero di UPDATE inviati dal demone senza la modifica.

Questo studio continuerà però su topologie più complesse e con l'applicazione di Pop-Routing [1] al timer.