UNIVERSITÀ
degli STUDI
di CATANIA

DTN Network: Epidemic - Prophet

a project authored by

Isgrò Santino 1000000617

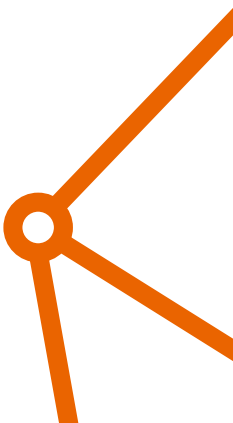
Maggio Alessandro W82000208

Merola Fabio W82000188

Teaching:

PEER TO PEER AND WIRELESS NETWORKS E LABORATORIO

31 March, 2021



Contents

Chapter 1

Introduzione

In questo studio presentiamo un'analisi comparativa delle prestazioni tra due differenti protocolli di rete di tipo DTN[1] quali *Epidemic* e *Prophet*.
Una volta studiato il funzionamento di tali protocolli, il passo successivo è stato quello di definire uno scenario ideale sul quale realizzare le simulazioni. I protocolli sono stati in una prima fase implementati, tramite l'utilizzo del software *NS-3 Network Simulator*[2], e successivamente sono state analizzate le prestazioni tramite precise simulazioni.
L'analisi è stata realizzata basandoci su 3 differenti metriche:

1. Tempi di consegna;
2. Percentuale di consegna;
3. Carico della rete.

Tali simulazioni sono state realizzate manipolando *TTL, Numero e posizione dei Nodi.* I risultati e tutte le considerazioni fatte sono disponibili nell'ultimo capitolo dell'elaborato.

1.1 DTN

Delay Tolerant Network o DTN (rete tollerante ai ritardi) è un'architettura di rete, che si propone come obiettivo la possibilità di far comunicare tra loro reti indipendenti, mutuamente incompatibili e non appartenenti ad Internet (definita in questo contesto come una rete globale continuamente interconnessa).
Nelle reti di dati tradizionali, come Internet, è garantito almeno un percorso end-to-end

continuo tra un nodo di origine e un nodo di destinazione attraverso il quale viaggiano i pacchetti. Nelle DTN, tuttavia, non è garantito un percorso end-to-end e i pacchetti vengono consegnati da un nodo di origine ad un nodo di destinazione tramite routing basato sullo *store and forward message*. Con questo meccanismo un nodo di origine o un nodo intermedio memorizza i pacchetti e li trasporta mentre si sposta. Questi pacchetti vengono inoltrati ad altri nodi in base a criteri predefiniti e infine vengono consegnati a un nodo di destinazione dopo un determinato numero di hop.
Le DTN sono anche definite *reti opportunistiche* poiché pensate per essere applicate in scenari difficili con densità di nodi sparsa, raggio di trasmissione breve, connettività intermittente e partizionamento frequente. Ad esempio, nelle aree non sviluppate o in scenari di disastro in cui non è disponibile una connessione ad Internet, o in reti di sensori che monitorano campi naturali, militari, o ancora reti opportunistiche mobili composte da veicoli in movimento o pedoni.

1.2 Epidemic

Epidemic è un protocollo di routing molto utile nel caso in cui si abbia a che fare con reti sparse e/o altamente mobili in cui potrebbe non esserci un percorso contemporaneo dalla sorgente alla destinazione. Adotta il paradigma precedentemente descritto dello *store-and-forward*.

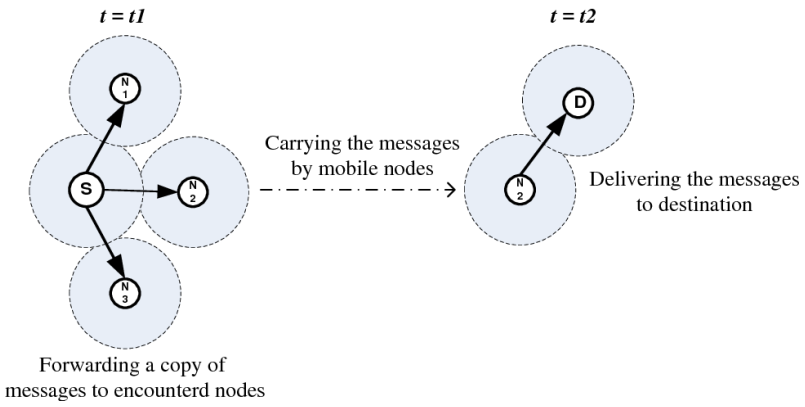


Fig. 1.1: Esempio sul funzionamento di Epidemic

In Fig.1.1 viene mostrata una sorgente S che desidera inviare un messaggio alla destinazione D, attualmente è fuori dalla sua portata poiché non vi è nessun percorso connesso

da S a D. Seguendo il protocollo Epidemic S trasmette i suoi messaggi ai vicini *N1, N2 e N3* tramite comunicazione diretta al tempo *t1*.
In un secondo momento, ovvero al tempo *t2*, *N2* entrerà in contatto con un altro nodo o nel caso della Fig.1.1 con la destinazione stessa al quale recapiterà il messaggio.

Analogamente alla diffusione delle malattie infettive, ogni volta che un nodo portatore di pacchetti incontra un nodo che non ha una copia di quel pacchetto, si dice che il corriere infetti il nuovo nodo trasmettendo una copia del pacchetto; che a sua volta si comporterà in modo identico verso tutti i nodi incontrati. Si verrà a creare così un *flooding* di pacchetti che nel caso in cui il traffico è molto basso, è in grado di ottenere tempi di consegna minimi a scapito di un maggiore utilizzo di risorse come larghezza di banda e potenza di trasmissione.

1.3 Prophet

Prophet[3] è un protocollo di routing DTN che mira a utilizzare le conoscenze ottenute da incontri passati con altri nodi per ottimizzare la consegna dei pacchetti. Diversamente da Epidemic che sfrutta un grande quantitativo di risorse e non prevede nessun metodo per eliminare pacchetti duplicati; Prophet si basa sul principio che gli incontri tra i nodi, soprattutto in un ambiente realistico, non siano del tutto casuali. I nodi muovendosi tendono ad avere maggiori probabilità di incontrare determinati nodi rispetto ad altri. In Prophet ogni nodo mantiene un vettore di stime di prevedibilità della consegna e lo usa per decidere se inoltrare il pacchetto ad un determinato nodo incontrato. Le stime di prevedibilità vengono aggiornate seguendo tre regole ben definite:

1. Quando un nodo M incontra un nodo E la prevedibilità di E aumenta:

$$P(M,E)_n = P(M,E)_o + (1 - P(M,E)_o) * L$$

   dove L rappresenta una costante di inizializzazione.

2. Le probabilità verso tutte le destinazioni D diverse da E sono "invecchiate":

$$P(M,D)_n = P(M,D)_o * \gamma^{\kappa}$$

dove γ rappresenta la costante di invecchiamento e κ rappresenta il tempo trascorso dall'ultimo "invecchiamento".

3. Le probabilità vengono scambiate tra M ed E e viene utilizzata la proprietà "transitiva" della probabilità per gestire il caso in cui due nodi si incontrano raramente, ma esiste un altro nodo che frequentemente incontra entrambi questi nodi:

$$P(M,D)_n = P(M,D)_o + (1 - P(M,D)_o) * P(M,E) * P(E,D) * \beta$$

dove β è una costante di ridimensionamento.

Nel seguente elaborato sono stati assegnati i seguenti valori alle costanti.

Costante	**Valore**
L	0.75
γ	0.98
β	0.25

Chapter 2

Scenario utilizzato

Passo fondamentale affinché la simulazione dia dei risultati il più attendibili possibile, al fine di testare le prestazioni dei protocolli analizzati, è stato quello di creare uno scenario di applicazione.

La seguente tabella mostra tutte le configurazioni utilizzate.

Area di simulazione	5000 x 5000 (m)
Tempo di simulazione	21600s
Protocollo di routing	Epidemic, Prophet
Numero di nodi	15, 25, 35, 50, 75, 100, 150, 200, 300
Numero di pacchetti	200
Delay prima dell'invio	30 (s)
Intervallo tra l'invio dei pacchetti	100 (s)
TTL	4, 5, 6, 7, 8, 9, 10, 11, 12
Mobility Model	Random Waypoint
Seed	10, 118, 177
Dimensione pacchetto	1024 bytes
Velocità spostamento nodi	18 k/h

Table 2.1: Configurazioni utilizzate per la simulazione

Chapter 3

Implementazione

Tutto il materiale descritto in questo capitolo è reperibile nella nostra repository[4] di GitHub.
Entrambi i protocolli sono stati sviluppati utilizzando le classi e gli helper messi a disposizioni da NS-3. La tipologia di helper può essere classifica in due gruppi:

- Per la creazione del canale di comunicazione sono stati utilizzati i seguenti helper: *WifiHelper, YansWifiPhyHelper, YansWifiChannelHelper, WifiMacHelper*
- Per la gestione dei nodi invece: *NodeContainer, NetDeviceContainer, InternetStackHelper*

Ovviamente, anche classi come Socket, Ipv4AddressHelper, etc, hanno giocato un ruolo importante per la corretta implementazione ed in particolare per l'invio e la ricezione dei pacchetti.

Data la complessità con la quale è stato sviluppato NS-3 diventa abbastanza antipatico e macchinoso estendere una classe per poi aggiungerne degli attributi o metodi, nel caso vi fosse la necessità di modificare in qualche modo la classe Node e Packet. Il workaround utilizzato in entrambi i protocolli è stato quello di creare una classe dedicata denominata **NodeHandler** e di dichiarare un vettore globale di tipo NodeHandler in modo tale che ogni nodo avesse la sua locazione di memoria e non vi fossero eventuali problemi di concorrenza. Ogni qualvolta viene inizializzata un'istanza di Node, NS-3 assegna in maniera automatica un identificativo, il quale coincide esattamente con la posizione dedicatagli all'interno del vettore. Dunque, ad esempio il nodo con id 0 troverà informazioni e metodi

aggiuntivi in *nodeHandlerArray[0]*. Nonostante entrambi i protocolli condividano la classe NodeHandler sono state realizzate due implementazioni differenti in quanto ci si è accorti che purtroppo le necessità erano abbastanza diverse.

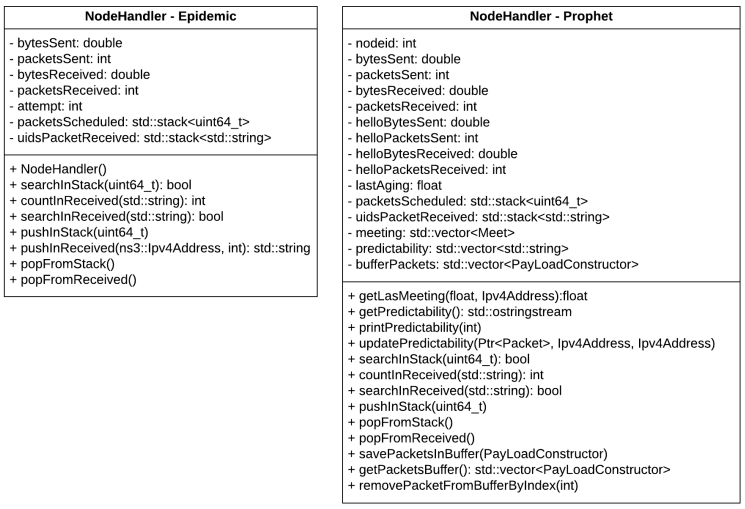


Fig. 3.1: UML - Classe NodeHandler per i protocolli Prophet ed Epidemic

Tutti gli attributi privati della classe sono ovviamente dotati di metodi getter e setter ma non sono stati inseriti all'interno dell'UML. La maggior parte degli attributi sono stati comunque inseriti esclusivamente per facilitarne la raccolta di dati statistici.

3.1 Epidemic

3.1.1 NodeHandler

La classe NodeHandler per Epidemic ha come attributi principali sicuramente i due stack denominati *packetsScheduled* e *uidsPacketReceived*. All'interno di packetsScheduled è possibile trovare tutti gli id dei pacchetti che sono stati ricevuti e successivamente inoltrati nella rete, di fatti, il metodo *searchInStack* controlla se l'id passato come argomento della funzione è presente nello stack e ritorna un valore booleano. La stessa funzionalità ha il metodo *searchInReceived* il quale però controlla i payload presenti in uidsPacketReceived. Infine, il metodo *countInReceived* riceve come argomento una stringa (con il formato IP;ID) e conta quante volte l'id del pacchetto in questione è presente in uidsPacketReceived.

3.1.2 Gestione dei pacchetti e relativo payload

Il payload dei pacchetti non è nient'altro che una banale stringa. All'interno di essa troviamo:

- Indirizzo di destinazione;
- TTL del pacchetto;
- ID del pacchetto.

Le informazioni di cui sopra sono state poi unite utilizzando il delimitatore "*;*", il risultato finale sarà dunque: *IP;TTL;UID*. Per facilitare la gestione delle stringhe è stata sviluppata la classe *PayLoadConstructor*.

3.1.3 ReceivePacket e GenerateTraffic

Ad ogni nodo è stata associata una socket collegata in broadcast e alla socket è stata collegata la *ReceivePacket* callback (utilizzando ovviamente i metodi MakeCallback e SetRecvCallback di NS-3). La ReceivePacket verrà ovviamente triggerata ogni qualvolta il nodo in questione riceva un pacchetto, inoltre, verranno eseguite delle semplici operazioni come:

1. Estrazione del payload dal pacchetto;
2. Se il pacchetto non è mai stato ricevuto lo si inserisce all'interno dello stack;
3. Se l'indirizzo di destinazione estratto dal payload fa riferimento al nodo correte l'esecuzione termina e ne vengono saltavi i dati;
4. Altrimenti, viene eseguito un controllo sul pacchetto il quale, deve avere TTL maggiore di 0 e la sua ricezione deve risultare assente nel passato;
5. Nel caso in cui le condizioni siano favorevoli si procede con l'invio del pacchetto tramite il metodo *GenerateTraffic*.

Il metodo GenerateTraffic prende in input parecchi argomenti:

- socket, di tipo Ptr<Socket>;

- packet, di tipo Ptr<Packet>;
- UID, di tipo numerico;
- previousAddressUid, di tipo stringa;
- TTL, di tipo numerico;

I nomi di tutte le variabili dovrebbero essere abbastanza esplicativi, eccetto *previousAddressUid*. Quest'ultima è una stringa composta da: *INDIRIZZO_IP;UID*, dove, con indirizzo IP si fa riferimento all'indirizzo del nodo dalla quale si è precedentemente ricevuto il pacchetto. Prima di procedere all'effettivo invio del pacchetto si controlla nuovamente che il pacchetto non sia già stato inviato e che non si abbia ricevuto il relativo ack per almeno 2 volte. Di fatti, 60 secondi dopo l'invio del pacchetto, GenerateTraffic chiama se stesso in maniera ricorsiva e continuerà a farlo fin quando l'ack non supera il valore di 2. Questo controllo è stato realizzato in modo tale da aumentare la percentuale di consegna dei pacchetti poiché ad un invio purtroppo non corrisponde sempre una ricezione (magari perchè il canale è occupato). Dato che i pacchetti sono inviati in broadcast l'ack in questo caso non è nient'altro che il pacchetto stesso.

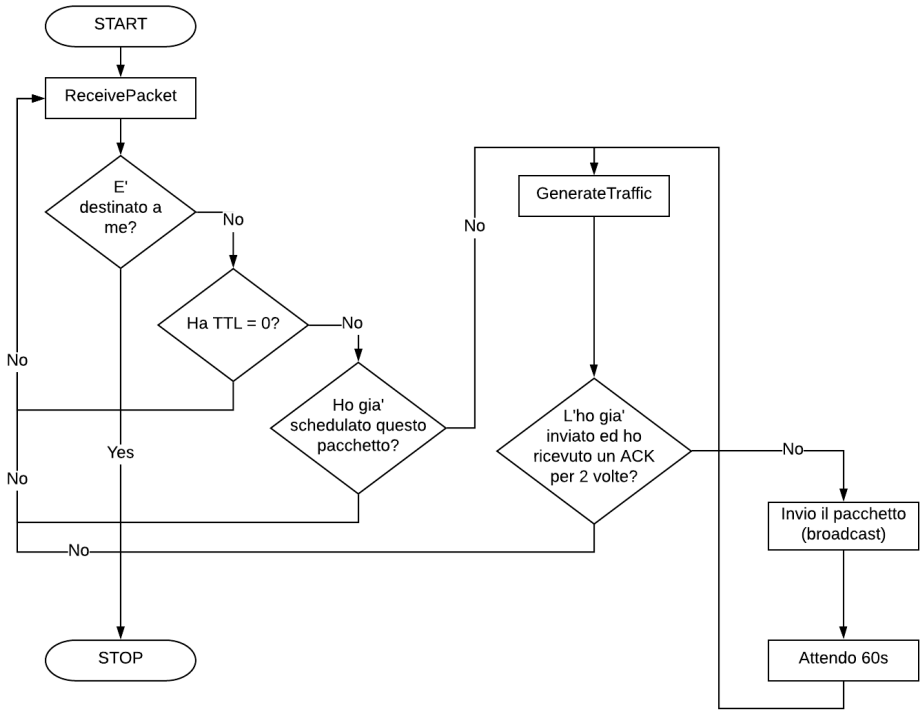


Fig. 3.2: Diagramma di flusso - Epidemic

3.2 Prophet

3.2.1 NodeHandler

Nella classe NodeHandler per Prophet oltre agli attributi e metodi descritti prima vi è la presenza di attributi e metodi aggiuntivi. Di particolare importanza il *float lastAging*, il *vector meeting di tipo Meet* e il *vector di stringhe denominato predictability.* Il tipo *Meet* è una semplice *typedef struct* contenente un *Ipv4Address* IP e un *float* time_meet. A supporto dell'array e della struct, NodeHandler è in possesso del metodo *getLasMeeting(float now, Ipv4Address ip)*, il quale, cerca all'interno dell'array meeting se vi è già stato un precedetene incontro tra il nodo attuale e quello passato come argomento, in caso affermativo aggiorna il valore presente in array e ne ritorna il delta time dall'ultimo incontro, altrimenti fa un inserimento all'interno dell'array. Il cuore del protocollo gira principalmente su due metodi, tra cui uno appartenente al NodeHandler, ovvero *updatePredictability(Ptr<Packet>packet, Ipv4Address ip, Ipv4Address currentIp).* Questo metodo viene richiamato ogni qualvolta si riceve un pacchetto e vi è appunto la necessità di aggiornare i valori di *predictability* della quale ogni nodo è a conoscenza. Come già descritto nel capitolo precedente le regole di aggiornamento sono tre: *diretto, invecchiamento, transitivo* e, in base alla tipologia di pacchetto ricevuto viene eseguita una determinata operazione.

Pacchetto	**Aggiornamento predict**
HELLO	Diretto
HELLO_ACK	Invecchiamento, transitivo
HELLO_ACK2	Transitivo

Table 3.1: Aggiornamento in base al tipo di HELLO ricevuto

L'aggiornamento di tipo 'diretto' avviene alla ricezione di un HELLO , esso si occupa di eseguire l'update del valore di prevedibilità di incontro del singolo nodo con il quale è appena avvenuta la comunicazione. La ricezione di un pacchetto di tipo HELLO_ACK prevede l'aggiornamento sia per 'invecchiamento' che 'transitivo'. Durante l'aggiornamento per invecchiamento tutti i nodi della quale si ha già un dato in tabella verranno invecchiati in quanto, incontrando nuovi nodi la prevedibilità di incontrare nuovamente gli stessi scende. Il 'meet' di nuovi nodi pero' oltre ad avere un fattore negativo come quello appena discusso permette la conoscenza di percorsi alternativi (magari anche più brevi/veloci) e la

presenza di ulteriori nodi, il tutto avviene grazie alla proprietà transitiva. Se due nodi A e B si incontrano e B conosce una prevedibilità per C, allora anche A ne verrà a conoscenza. A livello implementativo dunque vi sono due cicli for, uno prevede l'iterazione della tabella di prevedibilità contenuta nel pacchetto ricevuto e l'altro l'iterazione della propria prevedibilità, se entrambi i nodi conoscono un determinato nodo X e il valore transitivo è favorevole viene conservato altrimenti scartato. I nuovi nodi saranno ovviamente tutti conservati. Infine, come già scritto NodeHandler è previsto di un vettore *bufferPackets* il cui scopo è quello di contenere tutti i pacchetti che si stanno attualmente trasportando (carry). Tutte le prevedibilità contenute nel vector predictability sono stringhe il cui formato è il seguente: IP:PREDICTABILITY (lo stesso formato è utilizzato anche nei payload).

3.2.2 Gestione dei pacchetti e relativo payload

Come già anticipato questo protocollo prevede la presenza di diverse tipologie di pacchetti. Tutti i payload sono di tipo stringa e le varie informazioni formattate tramite l'uso di speciali delimitatori. Ad ogni tipologia di pacchetto vi è stato assegnato un id tramite enumeratore.

Pacchetto	ID	Payload
STANDARD	0	TYPE;DESTINATION_IP;HOP;UID
HELLO	1	TYPE
HELLO_ACK	2	TYPE;PREDICTABILITY
HELLO_ACK2	3	TYPE;PREDICTABILITY
PKTREQ	4	TYPE;DESTINATION_IP;HOP;UID
PKTACK	5	TYPE;DESTINATION_IP;HOP;UID

Table 3.2: Tipologia di pacchetti e relativo payload

La tipologia del pacchetto sarà dunque identificabile tramite il primo valore estratto dal payload (ovviamente dopo aver fatto uno split di ;).

Ogni 60 secondi i nodi inviano broadcast un pacchetto di tipo HELLO avente come payload solamente '1'. Chiunque riceva questo pacchetto risponde al diretto interessato con un pacchetto di tipo HELLO_ACK avente al suo interno un payload d'esempio '2;C:0.56;D;0.34'. Allo stesso modo chi riceve il pacchetto di tipo HELLO_ACK risponde con HELLO_ACK2 completando così lo scambio di pacchetti di tipo HELLO. Se volessimo dare un'interpretazione "umana" a questa sequenza, la discussione tra i due nodi potrebbe

essere la seguente:

1. "A: Ciao B"

2. "B: Ciao A volevo informarti che incontro spesso C con un valore di 0.56 e D con un valore di 0.34"

3. "A: Ciao B, grazie a te adesso anche io conosco un possibile percorso per C e D. Inoltre, io conosco E ed F, tieni magari ti fa comodo."

Completato lo scambio di pacchetti HELLO ogni nodo verifica il suo buffer e dunque se sta attualmente trasportando dei pacchetti, in caso positivo e nel caso in cui il nodo appena incontrato ha un percorso favorevole per portare il pacchetto a destinazione viene inviato un PKTREQ. Alla ricezione di un PKTREQ corrisponde l'invio di un PKTACK per confermare che effettivamente il pacchetto oltre ad esser stato ricevuto è stato anche preso in carico. Anche qui la sequenza potrebbe essere "umanizzata" nel modo seguente:

1. "B: A, ho visto che incontri spesso C. Ho un pacchetto destinato a lui con HOP 5. Puoi consegnarlo tu per me?"

2. "A: si certo B, me la vedo io"

Un esempio di payload è il seguente: 4;192.168.1.14;5;1.

3.2.3 ReceivePacket , GeneratePacket e GenerateHello

Proprio come per Epidemic anche in Prophet ad ogni nodo è stata associata una socket e a sua volta una callback denominata *ReceivePacket*. Questo metodo come precedentemente spiegato viene triggerato ogni qualvolta si riceve un pacchetto. Nel caso di Prophet in base alla tipologia di pacchetto ricevuto viene eseguito l'updatePredictability del nodo corrispondente e prosegue con il pacchetto di conferma. Nel caso in cui il pacchetto ricevuto è di tipo HELLO_ACK o HELLO_ACK2 vengono eseguiti tutti i controlli sul buffer per valutare un possibile inoltro dei pacchetti attualmente in carico. Principalmente il compito associato a questo metodo è quello di ricevere il pacchetto, analizzarlo e proseguire con l'invio del successivo per poter rispettare le chains legate agli HELLO o PKT. I metodi di *GeneratePacket* e *GenerateHello* sono pressoché simili; ogni metodo prepara un

pacchetto con payload di tipo *STANDARD* o *HELLO* rispettivamente e, nel caso di GeneratePacket viene semplicemente conservato nel proprio buffer. Nel caso di GenerateHello il pacchetto viene immediatamente inviato via broadcast.

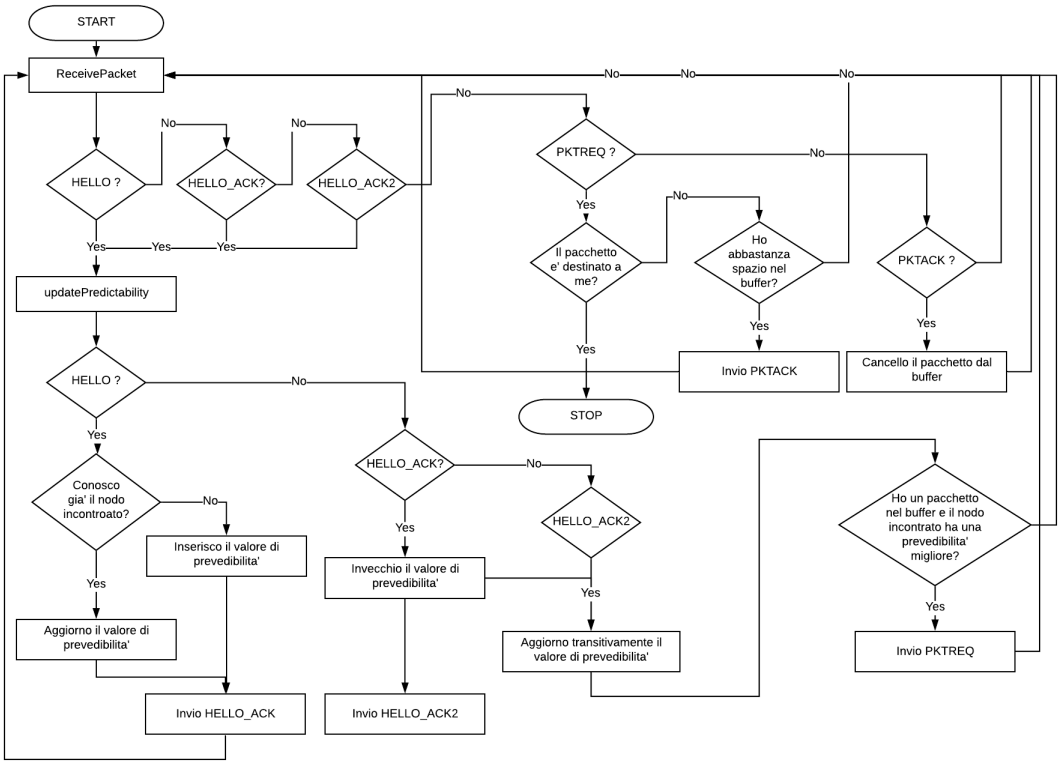


Fig. 3.3: Diagramma di flusso - Prophet

Chapter 4

Conclusioni

Come anticipato nell'introduzione, le analisi svolte si sono soffermate sulle seguenti variabili:

1. Tempi di consegna;
2. Percentuale di consegna;
3. Carico della rete.

Per motivi computazionali, ogni analisi è basata sulla media dei risultati di tre serie di simulazioni, nelle quali la posizione di partenza di ogni singolo nodo varia dipendentemente da quale dei tre seed randomici è stato utilizzato. Questa scelta è stata presa in considerazione per evitare che il risultato delle analisi fosse eccessivamente influenzato dalla posizione iniziale di mittente e destinatario.

4.1 Tempi di consegna

Questi sono stati considerati solo in funzione del numero di nodi presenti nella simulazione.

Nel grafico in Fig.4.1 le linee continue rappresentano il tempo medio impiegato dai pacchetti per arrivare a destinazione, nel dettaglio abbiamo in blu i tempi relativi al protocollo Epidemic, in arancio quelli di Prophet.

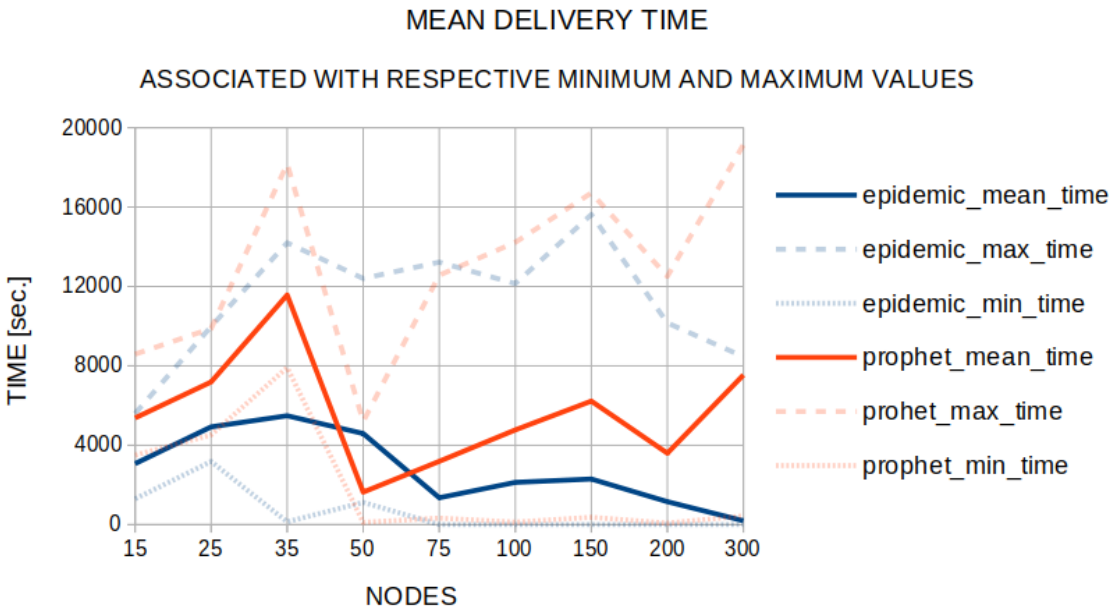


Fig. 4.1: Tempi medi di consegna - numero di nodi variabile

Come previsto, Epidemic mostra una risposta migliore rispetto a Prophet per la quasi totalità delle situazioni testate, addirittura si nota come il loro comportamento diverga oltre la soglia dei 50 nodi.

4.2 Percentuali di consegna

In questo caso l'analisi è stata svolta sia in funzione del numero di nodi presenti che del limite di HOP permesso(TTL). Precisazione necessaria per la comprensione della seconda analisi, quella in cui si impone un TTL, è che la nostra implementazione di Prophet non prevede un numero massimo di HOP per il pacchetto quindi, ogni percentuale è stata calcolata cumulando il numero di pacchetti consegnati con uno specifico numero di HOP ai precedenti.

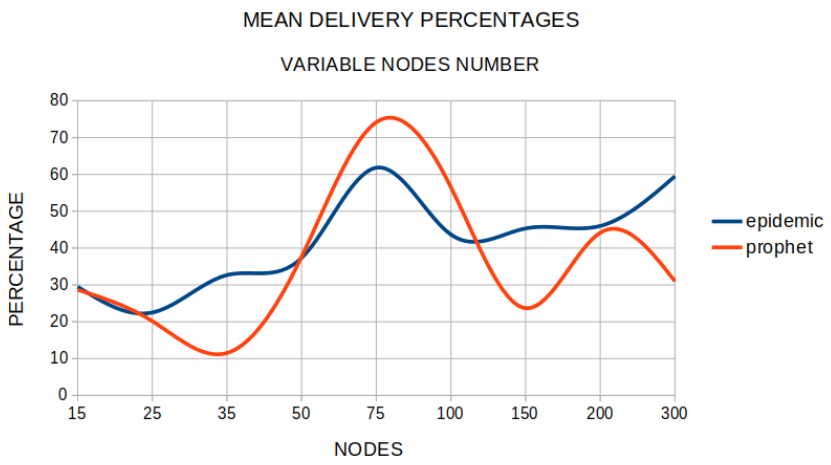


Fig. 4.2: Percentuali medie di consegna - numero di nodi variabile

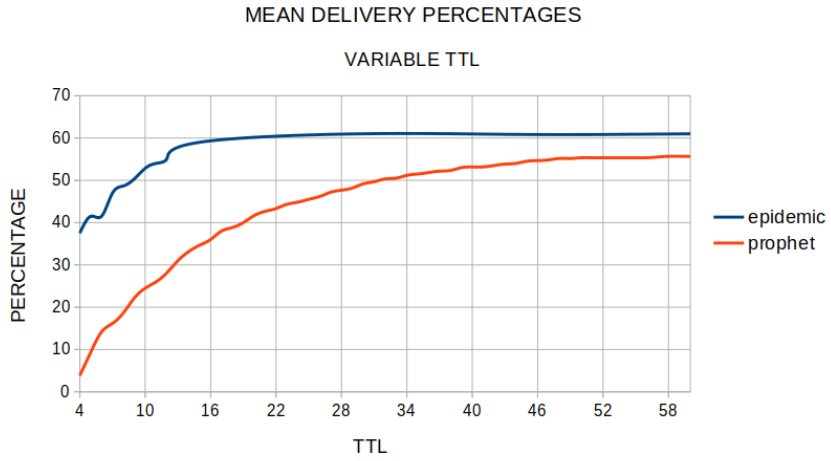


Fig. 4.3: Percentuali medie di consegna - TTL variabile

Il primo grafico, in Fig.4.2, denota come la reazione di Epidemic ad un aumento della popolazione dei nodi sia stabile se messa in relazione a quella di Prophet. Inoltre, è chiaro come, a differenza di Prophet, il protocollo Epidemic rispetti mediamente un rapporto di proporzionalità diretta tra le due variabili in questione.

Il secondo grafico, in Fig.4.3, invece mostra come, in queste condizioni di test, sia sufficiente adoperare un TTL pari a 16 per Epidemic per avere prestazioni, riguardanti i termini utilizzati in questa analisi, migliori rispetto a qualsiasi scenario che implichi Prophet.

4.3 Carico della rete

L'analisi in questione si sofferma sullo studio di due aspetti ritenuti fondamentali:

1. Relazione tra pacchetti immessi nella rete e dimensione degli stessi;
2. Comparativa tra i due protocolli in funzione del numero di nodi presenti e del TTL.

Stavolta, non potendo materialmente limitare il numero di HOP massimi di Prophet, che comunque si attesta mediamente sui 60 HOP, la comparativa avverrà utilizzando una media dei valori finali d'interesse presi dalle tre simulazioni con seed differente.

Carico medio della rete - Numero di nodi variabile

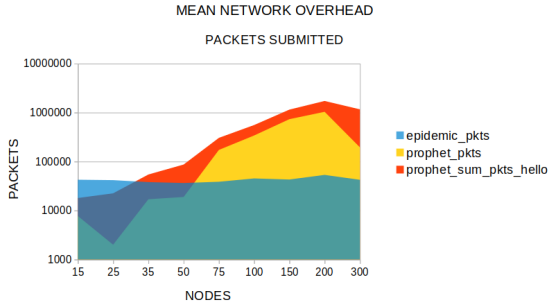


Fig. 4.4: Pacchetti immessi nella rete
Numero di nodi variabile - TTL = 6

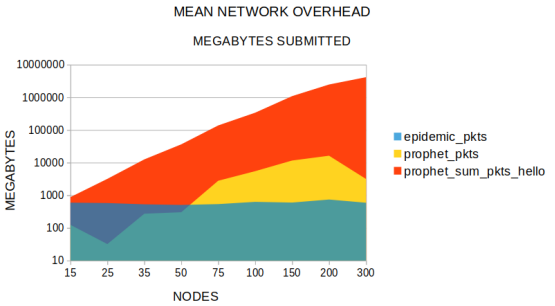


Fig. 4.5: Megabytes immessi nella rete
Numero di nodi variabile - TTL = 6

Carico medio della rete - TTL variabile

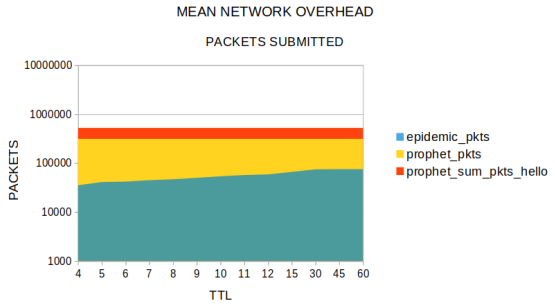


Fig. 4.6: Pacchetti immessi nella rete
TTL variabile - Nodi = 100

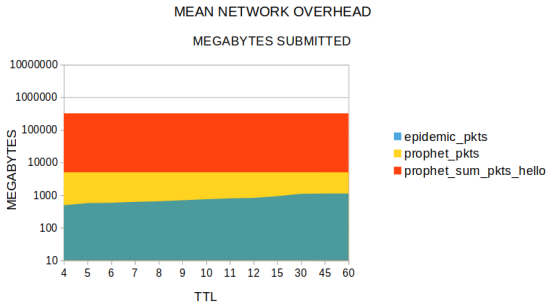


Fig. 4.7: Megabytes immessi nella rete
TTL variabile - Nodi = 100

Differentemente da quanto previsto, nelle nostre implementazioni dei due protocolli, risulta esserci un palese carico della rete superiore utilizzando Prophet, rispetto a Epidemic.

In riferimento a Prophet è possibile inoltre notare, paragonando i grafici a sinistra(Fig.4.4, Fig.4.6) con quelli a destra(Fig.4.5, Fig.4.7), che nonostante i pacchetti di HELLO(in arancio) siano in minoranza rispetto a quelli contenenti i messaggi tra mittente e destinatario(in giallo), all'atto pratico abbiano dimensioni decisamente maggiori. Nel dettaglio, prendendo in esame il grafico in Fig.4.5, notiamo come non solo è verificato quanto appena detto, ma in più la crescita delle dimensioni dei pacchetti di HELLO risulta sempre positiva e quasi lineare, indipendentemente dal fatto che essi aumentino o diminuiscano in numerosità. Questo fenomeno è sicuramente causato dell'accrescersi continuo delle tabelle di prevedibilità trasmesse nei suddetti pacchetti.

Un'altra importante informazione ricavabile dai grafici riguarda il numero di pacchetti e conseguentemente i Megabyte immessi nella rete utilizzando Epidemic. Infatti come si può osservare nel grafico in Fig.4.4, con un TTL fisso pari a 6, il carico sottoposto alla rete risulta essere evidentemente limitato in quanto, indipendentemente dal numero di nodi coinvolti, il numero di pacchetti immessi resta praticamente invariato. Questo trend, con tutta probabilità, verrebbe replicato anche settando un valore di TTL compreso nel range che va da 4 a 16 perché, come ci suggerisce il grafico in Fig.4.6, il protocollo causa abbastanza inoltri del singolo pacchetto da saturarne sempre il suo TTL fino alla soglia di TTL pari a 16, oltre la quale, sembra convergere per altre motivazioni smettendo di crescere. Tali motivazioni, data la regione geometrica in cui i nodi sono liberi di muoversi randomicamente, potrebbero essere attribuite all'alta densità di questi ultimi combinata con l'elevato guadagno delle antenne in ricezione e trasmissione, il che porta ad avere tutti i nodi interpellati già al sedicesimo HOP.

4.4 Sviluppi futuri

Di seguito sono esposte alcune implementazioni o analisi aggiuntive interessanti che potrebbero essere sviluppate in futuro allo scopo di arricchire e migliorare questo progetto.

Nel dettaglio, per il protocollo Epidemic si propone:

- Studio della relazione che sussiste tra TTL, densità della popolazione di nodi e area geometrica a disposizione;
- Implementazione di un metodo che analizzi messaggi inviati con differenti TTL,

basandosi sui risultati dello studio proposto sopra, al fine ultimo di stimare:

  - Miglior TTL(considerati tutti gli aspetti prestazionali);
  - Popolazione di nodi;
  - Area geometrica di competenza.

Invece per il protocollo Prophet si propone:

- Implementazione di un sistema di pulizia della tabella di prevedibilità che ne implichi comunque una dimensione dinamica, ma sicuramente inferiore;
- Implementazione di un sistema di TTL come visto per Epidemic;
- Implementazione dell'intervallo temporale che si interpone tra gli invii dei messaggi di HELLO in maniera dinamica, analizzando magari la variabilità delle tabelle ricevute;
- Provare a considerare l'utilizzo di due tipologie di nodi d'appoggio:
  - Nodi pozzo;
  - Nodi messaggero.

  I primi posizionati in punti strategici, ad esempio disposti in circolo al centro dell'area di interesse, i secondi, in costante movimento, si dirigono verso i confini per poi tornare al centro, facendo quindi da tramite tra nodi comuni e pozzi. I pozzi sono abbastanza vicini da essere in costante contatto tra di loro. Concettualmente quindi i messaggeri raccolgono messaggi dalle periferie, li portano ai pozzi e questi inoltrano ogni messaggio al nodo comune od al messaggero che rispetta al meglio le regole già descritte per Prophet. Questo procedimento dovrebbe creare una maggiore coesione della popolazione di nodi, atta ad aumentare la percentuale di consegne e diminuirne le tempistiche.

References

[1] RFC4838 - Delay Tolerant Network.
[https://tools.ietf.org/html/rfc4838].
Ultimo accesso: 3 Maggio 2020.

[2] NS-3 - Network Simulator.
[https://www.nsnam.org/].
Ultimo accesso: 3 Maggio 2020.

[3] Prophet routing protocol.
[https://en.wikipedia.org/wiki/PRoPHET_routing_protocol].
Ultimo accesso: 3 Maggio 2020.

[4] GitHub repository: DTN-Networks-ns3 .
[https://github.com/SantinoI/DTN-Networks-ns3].
Ultimo accesso: 3 Maggio 2020.