UNIVERSITÀ DEGLI STUDI DI CATANIA

Dipartimento di Matematica e Informatica

Laurea Triennale

Martin Gibilterra

netdemic: simulazione di un'epidemia in una delay-tolerant network virtuale

Tesi

Relatori: Prof. Filippo L.M. Milotta
Prof. Filippo Stanco

Anno Accademico 2021 - 2022

Abstract

Le reti Internet convenzionali incontrano diverse difficoltà non appena la distanza tra gli host aumenta in modo significativo, ma anche quando il canale di comunicazione tra i due è instabile e la connessione intermittente. Per permettere a due utenti molto distanti fra di loro o che non riescono a connettersi stabilmente di comunicare secondo le proprie esigenze, serve un'architettura di rete capace di far fronte alle numerose problematiche che sorgono a causa di un canale di comunicazione inadeguato o troppo lungo per permettere un flusso di dati continuo tra i due host. In questo documento viene descritta un'architettura di rete a intermittenza che cerca di mitigare questi problemi, per poi successivamente dimostrare, grazie ad una simulazione grafica, gli effetti di un'epidemia all'interno di una rete a intermittenza sprovvista di qualsiasi misura di sicurezza a supporto di ogni singola connessione, le cui fondamenta sorgono su principi leggermente diversi da quelli alla base di una convenzionale rete Internet.

Contents

Dedicato alla mia famiglia, a chi mi ha reso migliore, a chi mi sta accanto giorno dopo giorno, a Samuele senza il quale niente sarebbe stato possibile, a Trilli, a Sabrina

Chapter 1

Introduzione alle delay-tolerant networks

Una *delay-tolerant network* (DTN, in italiano "rete a intermittenza" o "rete tollerante ai ritardi") è un'architettura di rete che si fonda sulla necessità di scambiare dati tra i nodi tollerando ritardi di trasmissione di molto superiori alle normali reti cablate o wireless senza grandi problemi.
Il modello DTN viene applicato alle cosiddette *challenged networks*, cioè tutte quelle reti i cui nodi componenti incorrono in tutta una serie di problematiche che, in un'architettura di rete normale, renderebbero impossibile una affidabile comunicazione end-to-end. Tra gli ostacoli più impervi si possono annoverare:

- tempi di trasmissione nell'ordine dei secondi, dei minuti o più;
- banda disponibile ridotta: ciò obbliga a limitare fortemente la quantità di dati da inviare e le ritrasmissioni in caso di errore;
- condizioni ambientali avverse e interferenze elettromagnetiche: esse minano fortemente la stabilità dei collegamenti tra i nodi causando disconnessioni frequenti;
- distanze considerevoli, a volte anche satellitari o interplanetarie;
- potenza, operatività e longevità dei dispositivi risicate.

In questo tipo di reti, il modello TCP/IP usato dalle comuni reti Internet non può funzionare in modo efficace, in quanto questo modello presuppone che esistano dei collegamenti stabili tra due nodi, che il round-trip time di un dato tra due nodi sia

relativamente basso e che non ci siano enormi perdite di trasmissione causate da agenti esterni.
Dal momento che la maggior parte delle *challenged networks* viola almeno uno di questi presupposti, questo genere di reti necessita di un'architettura diversa da quella basata sul modello TCP/IP.

1.1 Esempi di *challenged networks*

Le *challenged networks* sono costituite solitamente da dispositivi poco performanti, dislocati in ambienti ostili o di difficile accesso, oppure che parlano tra di loro a distanze enormi. In [1] vengono forniti gli esempi generici di *challenged networks* di seguito discussi.

Reti di comunicazione satellitare o interplanetaria. Nelle reti di comunicazione satellitare o interplanetaria, le trasmissioni di dati avvengono a distanze notevoli e possono essere soggette a ritardi elevati causati dalla distanza tra il dispositivo e il satellite o la sonda di destinazione, nonché da condizioni climatiche sfavorevoli e interferenze di qualsiasi tipo che possono disturbare il segnale o addirittura troncare la connessione temporaneamente. L'architettura di delay-tolerant network presentata nelle prossime sezioni si fonda su degli studi riguardo all'architettura *Interplanetary Internet* (IPN) [2], un'architettura di rete adatta per le connessioni interplanetarie.

Comunicazioni militari. Le comunicazioni militari effettuate in ambienti ostili, in cui il rischio di disturbi volontari provocati da agenti esterni (ad esempio dai *jammers*) è molto alto, sono potenzialmente inclini a disconnessioni frequenti nonché a problemi di sicurezza la cui neutralizzazione è di vitale importanza per l'esistenza stessa della connessione. Le comunicazioni di questo tipo necessitano di un'architettura che garantisca sicurezza e affidabilità della connessione anche in presenza di ritardi eccessivi e disconnessioni continue. L'architettura DTN si presta molto bene ad applicazioni sul campo militare, in particolare negli scenari in cui questo modello di rete può sopperire all'assenza temporanea o alla distruzione di collegamenti satellitari fondamentali per lo svolgimento per le operazioni militari,

come quelli presentati in [3]. Esistono inoltre dei progetti in via di sviluppo che si pongono l'obiettivo di studiare l'architettura DTN a fondo per creare delle soluzioni tecniche e funzionanti per rendere sicura ed efficiente la comunicazione all'interno di reti tattico-militari: uno di questi è *MIDNet* [4], un progetto che mira a rendere possibile la fruizione, in perfetta sicurezza, di determinati servizi Internet in un ambiente di rete a intermittenza, come i browsers, le e-mail e altri servizi web che implementano sistemi militari come il *blue force tracking*[1].

Reti di sensori. Il problema principale da tenere presente in una rete di sensori è la scarsa potenza di ciascun nodo, che ha una dimensione così ridotta da non poter rendere possibile realizzare a tal proposito un hardware con prestazioni accettabili. Inoltre, un sensore è un dispositivo fabbricato su larga scala e quindi sarebbe impensabile avere reti di milioni di sensori altamente performanti, la cui produzione risulterebbe assurdamente onerosa a livello economico e di spazi necessari. Un singolo sensore ha una longevità limitata, di conseguenza la potenza utilizzata deve essere gestita al meglio, limitando le comunicazioni solo nei momenti di stretta necessità e spegnendolo quando non è operativo. Per ottimizzare il risparmio energetico, i sensori usano il *low-duty-cycle system*, una pratica che permette di usare il sensore soltanto all'interno dell'arco temporale del suo *duty-cycle*, cioè quella frazione di tempo in cui il dispositivo "lavora", calcolata in proporzione al tempo totale di riferimento; nelle reti di sensori, ogni sensore ha un *duty-cycle* molto ridotto, il che aumenta la longevità del dispositivo e ottimizza la quantità di energia effettivamente utilizzata. Esistono dunque diverse sfide da superare per costruire nel mondo reale una rete di sensori che lavori in modo efficiente. Ciò potrebbe essere possibile applicando i principi fondamentali di una generica architettura DTN ad un insieme di sensori, come approfondito anche in [5].

Date le numerose possibili applicazioni in campo pratico e considerate le limitazioni hardware dei dispositivi di una rete di tale natura, soprattutto dal punto di vista dello storage, nel tempo sono stati ideati diversi modelli di architettura di rete, anche diversi dall'architettura di delay-tolerant network classica, che affrontano

[1] Il *blue force tracking* è un sistema di tracciamento militare che, grazie all'uso della geolocalizzazione e di dispositivi GPS compatibili, fornisce ai comandanti e alle forze militari informazioni sulla posizione delle forze militari amiche. Il nome deriva dalla simbologia militare della NATO, dove il blu indica tipicamente le forze amiche.

il problema grazie ad approcci diversi. Un esempio è il modello *SWIM* per reti di sensori ad-hoc usate per applicazioni biologiche [6], che si pone l'obiettivo di connettere dispositivi poco performanti e poco capienti in termini di storage, garantendo tuttavia un ritardo di comunicazione bilanciato alle capacità dell'hardware, oppure *ZebraNet* [7], una rete sperimentale *peer-to-peer* situata in Kenya e costituita da sensori di tracciamento inseriti all'interno di speciali collari che, una volta indossati da un animale, tenevano traccia della sua posizione all'interno del territorio, per poi comunicare i dati raccolti ad una stazione mobile creata ad-hoc dai ricercatori coinvolti nell'esperimento soltanto quando essi erano sul posto.

Reti mobili. Anche le reti cellulari possono essere un esempio di rete di questo genere, in quanto la distanza che percorre il segnale dall'antenna a un dispositivo mobile può essere non indifferente e la connessione tra questi due agenti può essere esposta a dispersione, interferenze e quindi anche a disconnessioni, dal momento che il canale di comunicazione non è stabile come in una rete cablata. La simulazione proposta in questo documento viene eseguita su una ipotetica rete wireless.

1.2 Perché progettare una nuova architettura di rete

Il modello TCP/IP non è adatto a supportare una rete di questo tipo per tanti motivi: ad esempio, per il routing dei pacchetti in una DTN, IP è inutilizzabile in quanto il modo in cui i router inoltrano i pacchetti, richiedendo un nodo di destinazione immediatamente attivo a cui affidare il pacchetto, si scontrerebbe con la precaria stabilità dei collegamenti tra i nodi caratteristica delle DTN, scartando potenzialmente la maggior parte dei pacchetti trasmessi; dal punto di vista delle applicazioni di rete, esse sono solitamente sviluppate con l'idea di lavorare con latenze non eccessivamente alte e le caratteristiche intrinseche di una DTN non sono compatibili con questo scenario, necessitando di un adattamento per lavorare con ritardi elevati, il quale non è sempre possibile o conveniente.

Una prima soluzione potrebbe essere quella di inserire un intermediario tra un nodo e una *challenged network*, che "traduca" il modello TCP/IP in un modello compatibile con le caratteristiche delle DTN, operando sul flusso dei dati: ciò è

stato realizzato in passato tramite l'introduzione di *Performance Enhancing Proxies* (PEPs) [8] e *protocol boosters* [9], degli agenti hardware o software che riescono ad adattare i protocolli TCP/IP ad una rete a intermittenza facendo credere loro di trovarsi in una comune rete a bassa latenza. Tuttavia, questi agenti potrebbero diventare dei colli di bottiglia e dei *single point of failure*, perché tutto il traffico da e verso la *challenged network* passerebbe da loro, il che sarebbe un bel problema nei casi in cui i pacchetti vengano instradati tramite routing asimmetrico[2], perché il transito sarebbe limitato su un singolo path. Dall'analisi di queste problematiche scaturisce il bisogno di creare un nuovo tipo di architettura di rete, diversa da quella gestita dal modello TCP/IP, che riesca a svolgere il suo compito anche nelle condizioni estreme imposte da una *challenged network*.

1.3 Elementi caratteristici di un'architettura di delay-tolerant network

La generica architettura di *delay-tolerant network* in esame è progettata tenendo in mente il concetto che essa deve garantire l'interoperabilità tra reti classiche e *challenged networks*, tra due o più *challenged networks* collegate fra loro e all'interno di una stessa *challenged network*. L'architettura DTN si definisce di "overlay", in quanto opera al di sopra dei protocolli già esistenti nelle reti che essa collega.

1.3.1 Regioni

La singola rete interconnessa, nell'architettura DTN, viene definita *regione* e ognuna di esse può utilizzare uno stack protocollare differente, pertanto questa architettura deve fornire una serie di protocolli generici utilizzabili da ogni singola regione, indistintamente.
In particolare, una regione è una zona i cui nodi componenti comunicano attraverso lo stesso stack protocollare e non hanno bisogno di nessun intermediario per comunicare tra di loro. Tuttavia, potrebbe avvenire che due nodi di due regioni diverse

[2]Il routing asimmetrico è un tipo di instradamento che prevede la possibilità che un pacchetto segua due diversi percorsi per l'andata e il ritorno tra due reti diverse.

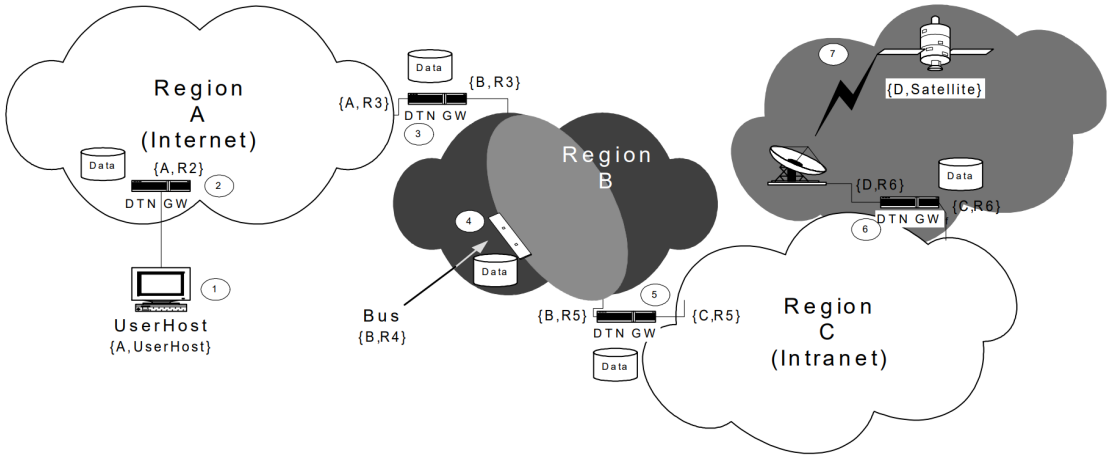

Figure 1.1: Struttura generica di una DTN: si noti che i gateways DTN possono interconnettere regioni con discrepanze tra loro in termini di protocolli utilizzati [1]

vogliano comunicare: in questo caso, serve un elemento di mezzo per far comunicare le due regioni tra di loro, detto *gateway*.

1.3.2 DTN gateways

Due regioni che comunicano con protocolli diversi possono mettersi in contatto attraverso un *DTN gateway* (o più di uno nel caso in cui le regioni non siano adiacenti), i quali esercitano la funzione di *store-and-forward* dei pacchetti: quando un pacchetto non può essere inoltrato verso la regione di destinazione perché non c'è collegamento con il nodo destinatario, il *DTN gateway* non scarta il pacchetto ma lo conserva per poi inviarlo non appena rileva che il collegamento è disponibile.
Il *DTN gateway* ha la duplice funzione di fare sia da firewall che da router: ogni pacchetto che entra o esce da una regione deve obbligatoriamente passare dal *DTN gateway* per motivi di sicurezza e controllo di flusso ed esso si occupa di instradarlo verso la regione di destinazione. I *DTN gateways* sono costruiti come delle entità divise in due metà e poste in mezzo a due regioni adiacenti, dove ogni metà si trova, figurativamente, all'interno della rispettiva regione, come avviene per le interfacce di rete di un normale router che si affaccia su più di una LAN.

1.3.3 Tuple identificative

Per instradare un pacchetto verso una regione, i *DTN gateways* sfruttano delle *tuple identificative* di valori che fanno da etichetta per ognuno dei gateway. Ogni gateway conserva in una tabella di routing le informazioni sulle tuple identificative di tutti gli altri gateway della rete.

Le tuple sono formate da $t + 1$ valori ($t \in \mathbb{Z}$), uno per il nome della regione R e da 1 a t valori per identificare univocamente l'entità gateway E, grazie ai quali il gateway calcola il percorso migliore da scegliere per arrivare alla regione di destinazione. Nella figura 1.1 la tupla identifica un gateway tramite la generica coppia di valori (N_R, N_E), dove N_R è il nome della regione in cui risiede l'interfaccia del gateway e N_E è il nome stesso dell'entità gateway, ma questa tupla potrebbe contenere anche altre metriche di routing, come ad esempio un livello minimo di priorità che il pacchetto deve avere per passare da quel gateway, la capacità delle sue code di storage e altri elementi che possono ottimizzare il routing (se noti). Le tuple identificative potrebbero a un primo impatto essere paragonate agli hostname nei DNS, tuttavia ci sono alcune differenze: contrariamente al protocollo DNS, le tuple non rispettano nessun tipo di gerarchia e inoltre non conservano nessun dato circa gli indirizzi fisici delle singole entità, perché il routing è basato esclusivamente sulla scelta di un path tramite nome di entità.

Analizzando ancora la Fig. 1.1, conservando soltanto le tuple dei gateway circostanti, il gateway $R2$ della regione A non sa nulla sulla regione B, se non il suo nome e quello del gateway che deve usare per trasmettere qualcosa a qualcuno in quella regione: tutto ciò garantisce una maggiore sicurezza nel trasporto, anche perché i gateways della regione B sono gli unici a poter tradurre correttamente i metadati del pacchetto, in modo tale da poterlo indirizzare verso il nodo locale di destinazione (si parla infatti di interpretazione locale o *late binding* delle tuple). Questa interpretazione viene fatta quando la connessione è già aperta, mentre la risoluzione dell'host operata da DNS è il primo passo fondamentale senza il quale una connessione end-to-end non può nemmeno essere aperta: questo modo diverso di procedere è dovuto alle alte latenze che una comunicazione in una DTN deve fronteggiare, latenze che non permetterebbero di fare delle richieste DNS, ricevendo una risposta e inviando una richiesta di apertura di una connessione nei brevissimi tempi che si è soliti vedere utilizzando la rete Internet.

1.3.4 Bundles e Bundle Protocol (BP)

Un nodo facente parte di una DTN non invia dei semplici pacchetti, ma aggregati di messaggi chiamati *bundles*, formati principalmente da due elementi:

- un *header* che contiene dei metadati necessari al *DTN gateway* per eseguire le operazioni di *store-and-forward* verso la regione designata come destinazione, nonché le informazioni di routing che portano il bundle a destinazione;
- il *payload*, costituito dal pacchetto originale che sarebbe stato inviato da solo nel caso di una trasmissione in una rete non-delay-tolerant.

Per gestire la costruzione dei bundles, in cima allo stack protocollare utilizzato da un DTN gateway c'è un protocollo aggiuntivo: il *Bundle Protocol* (BP). Questo protocollo viene utilizzato per costruire il bundle e inserirvi le informazioni di instradamento, in modo simile a quanto fa IP inserendo la propria frame nel pacchetto.

Tuttavia, c'è una grossa differenza tra IP e BP: mentre IP funziona se e solo se esiste tra i due nodi un path attivo senza interruzioni, BP è costruito in modo tale da ovviare a disconnessioni ed errori frequenti caratteristici di una *delay-tolerant network*, cercando potenziali strade alternative per far arrivare il *bundle* a destinazione oppure ordinando di conservarlo in caso di irreperibilità del destinatario. Per semplicità di collegamento con nozioni già ampiamente conosciute, in questo documento il bundle avrà la denominazione più generica di *pacchetto*.

1.3.5 Livelli di convergenza

In un'architettura di *delay-tolerant network*, il *Bundle Protocol* è in cima alla pila di protocolli che vengono usati localmente alla regione in cui un gateway risiede. Per rendere possibile la comunicazione tra questi protocolli e il BP, un gateway può implementare uno o più *convergence layers*, o livelli di convergenza. I livelli di convergenza, comunicando sia con i livelli sottostanti che con il *Bundle Protocol layer*, aiutano quest'ultimo a confezionare il pacchetto in maniera corretta fornendogli un livello di astrazione aggiuntivo utile ad adattare i dati provenienti dai livelli sottostanti ad un formato leggibile e utilizzabile dal BP layer.

I livelli di convergenza vengono implementati tenendo a mente la tipologia e le caratteristiche della rete in esame, nonché dei dispositivi che ne fanno parte;

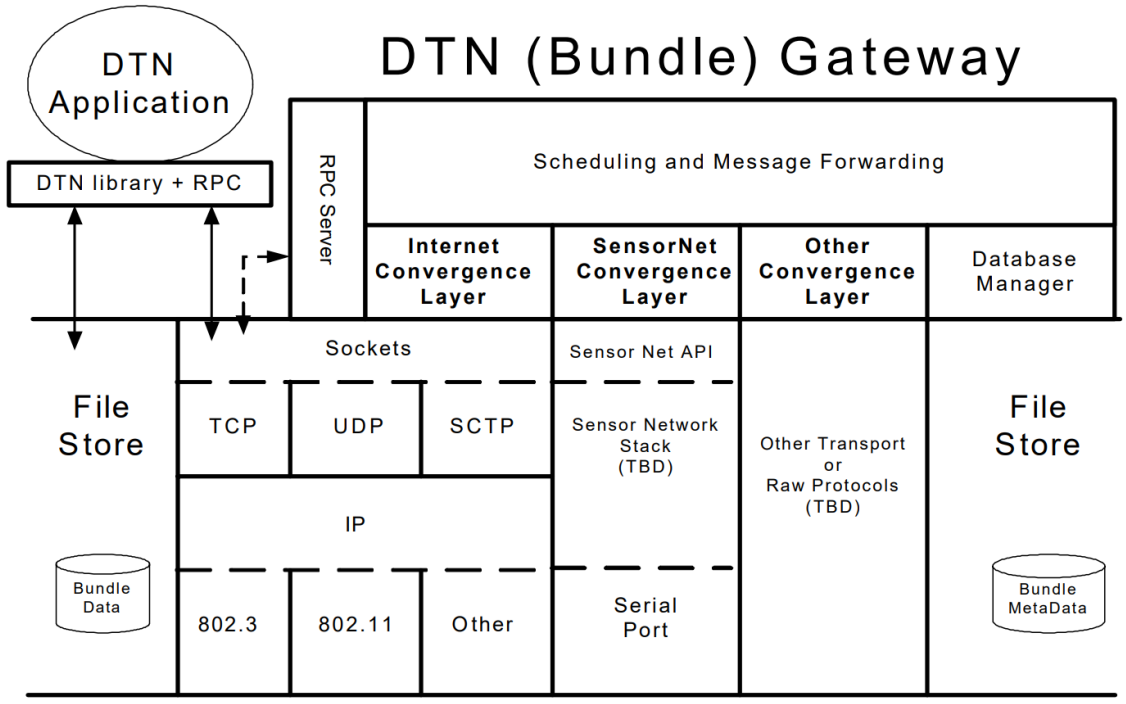

Figure 1.2: *Struttura di un generico DTN gateway con tre livelli di convergenza [1]*

all'interno di uno stesso gateway possono conviverne più di uno: questo accade nel caso in cui una regione contenga dispositivi di diversa natura. Nella Fig. 1.2, il gateway possiede tre livelli di convergenza: uno per i protocolli Internet sottostanti, uno per i protocolli *SensorNet* per la comunicazione tra sensori, un terzo per dei protocolli di genere non specificato. In particolare, un livello di convergenza svolge l'importante ruolo di raccogliere le feature di tutti i protocolli sottostanti e unirle in un solo layer: questo è molto utile nel caso in cui nello stack protocollare ci siano n protocolli, uno dei quali implementa delle feature che gli altri non possiedono e che potrebbero essere molto importanti per ottimizzare la connessione o per avere determinate funzionalità, in certi casi fondamentali, come i *message boundaries*[3].

Uno scenario d'esempio banale sarebbe quello in cui per trasportare dati si usa TCP, ma TCP ha il difetto di inviare flussi di byte e non singoli messaggi; pertanto, un *convergence layer* soprastante TCP potrebbe implementare una funzionalità che

[3]Per *message boundary* si intende la netta separazione tra due messaggi inviati tramite un qualsiasi protocollo di trasporto dati. UDP utilizza questa proprietà, TCP invece no, in quanto invia un flusso di byte continuo.

permette di sezionare i byte trasmessi in modo tale da distinguere i messaggi in arrivo.

1.3.6 Trasferimento di custodia

I nodi gateway di una DTN possono essere classificati in due macrocategorie:

- nodi di tipo **P** (*persistent*): sono nodi che possono conservare una quantità non indifferente di messaggi e hanno un tempo di uptime più alto della media, rispetto al totale del tempo di riferimento;
- nodi di tipo **NP** (*non-persistent*): sono nodi molto meno potenti dei nodi P, solitamente non sono adatti a conservare messaggi in quanto hanno uno storage ridotto e, essendo poco stabili, sono soggetti a periodi di downtime molto più alti dei nodi P.

Qualsiasi nodo, in base alle proprie capacità elaborative e trasmissive, concorre ai processi di trasferimento di custodia di un pacchetto. Il trasferimento di custodia è il processo con il quale un nodo, dopo aver ricevuto e conservato un pacchetto, lo inoltra al successivo DTN hop delegandogli la responsabilità di inoltrarlo in modo affidabile all'hop successivo continuando la *chain of trust*, come se l'hop successivo sottoscrivesse un contratto di responsabilità che lo vincola ad effettuare quella operazione [10]. I nodi P sono quelli che concorrono nella maggior parte ai processi di trasferimento di custodia (*custody transfer*) di un pacchetto da un DTN hop a un altro rispetto ai nodi NP. Tuttavia, si noti bene che tutti i nodi possono sia ricevere che inviare messaggi, che essi siano di tipo P o NP, e quindi tutti i nodi possono far parte della catena di custodia: gli elementi che distinguono un nodo di tipo P da uno di tipo NP sono infatti soltanto la quantità di memoria disponibile e il tempo di attività, da cui si evince la "frequenza" con la quale i nodi riescono a trasmettere i pacchetti. I nodi P possono inviare e ricevere messaggi più frequentemente in quanto sono più potenti, avendo delle code di storage di messaggi molto capienti e dei tempi di attività superiori alla media, a differenza dei nodi NP che, come discusso sopra, hanno poca potenza elaborativa e poca memoria, rendendoli più raramente parte di un processo di trasferimento di custodia. Il concetto del trasferimento di custodia è fondamentale in quanto aiuta a limitare le perdite, individuando eventuali nodi colpevoli, e solleva il singolo nodo da una grossa parte di responsabilità in caso di

inoltro fallito, soprattutto nel caso in cui capitasse che la custodia venga trasferita a nodi poco potenti con risorse limitate. Trasferendo subito la responsabilità ad un altro nodo, viene limitata di molto la possibilità che la custodia rimanga ad un nodo facilmente intasabile, più incline a periodi di downtime. Tuttavia, dare un eccessivo peso al processo di trasferimento di custodia non sempre garantisce che il pacchetto venga innanzitutto ricevuto, in quanto un pacchetto potrebbe rimanere fermo per molto tempo in attesa di un nuovo nodo a cui affidarne la custodia e il prosieguo del suo cammino.

Chapter 2

Fondamenti teorici alla base di una simulazione ideale di delay-tolerant network

Dal momento che la tecnologia DTN è una tecnologia sperimentale in via di sviluppo e standardizzazione, non esiste una metodologia standard per costruire un sistema che implementi le funzionalità necessarie ad una rete che possiede tali caratteristiche. In questo capitolo, vengono presentati alcuni fondamenti teorici sui quali una simulazione epidemica di una delay-tolerant network ideale potrebbe essere costruita, che compongono lo schema teorico di un approccio che si può utilizzare per la sua implementazione.

2.1 La rete come un sistema di particelle

Una rete delay-tolerant può essere pensata come una rete wireless peer-to-peer in cui diversi host dislocati in varie zone interagiscono fra di loro e si muovono all'interno delle proprie regioni di appartenenza. Ogni regione può essere rappresentata come un sistema di particelle che si muovono casualmente o meno. Nel caso in cui fosse presente un sistema di infezione (sia che si manifesti come singolo nodo attaccante o come un numero arbitrario di nodi malevoli), anche questo sistema sarebbe definibile come un sistema di particelle, che si spostano e interagiscono col sistema chiuso rappresentato dalla rete in esame.

In generale, un sistema di particelle è un sistema di n elementi puntiformi che possono interagire tra di loro o meno, nel quale i punti non si muovono tutti nella

stessa direzione, ma possono muoversi in modo casuale o secondo determinati criteri. Il movimento non uniforme fa sì che il centro di massa del sistema di particelle, cioè quel punto che rappresenta la posizione del sistema se esistesse una sola particella con massa uguale alla somma delle n particelle, si sposti in continuazione. Esistono sistemi di particelle continui, in cui la distanza tra le particelle è infinitesimale, e discreti, dove le particelle sono molto distanti tra di loro. Nel caso in esame, dal momento che gli host della rete possono essere anche a distanze molto elevate fra di loro, si può immaginare la rete come un grande sistema discreto di n particelle di eguale massa, suddiviso in m sottosistemi discreti di particelle che interagiscono tra loro all'interno di un sistema chiuso bidimensionale. In particolare, in un ambiente caratterizzato da un'epidemia in corso, ogni particella è soggetta a cambiamenti di stato e di comportamento che possono modificare o meno le caratteristiche del sistema stesso: ad esempio, un cambiamento della velocità di movimento a seguito dell'infezione di una particella può accelerare o rallentare lo spostamento del centro di massa o può tagliare il collegamento tra la particella stessa e il proprio sistema di particelle di appartenenza, svincolandosi da esso ed eventualmente entrare a far parte di un altro sistema vicino.

2.2 Struttura ideale di una simulazione epidemica

Una simulazione ideale può svolgersi in un campo bidimensionale, che rappresenta il sistema chiuso che deve contenere l'intero sistema di particelle, che d'ora in poi assumeranno il nome di *nodi*. Il campo di simulazione che ospiterà la rete virtuale viene inizialmente suddiviso in $n + 1$ clusters ($n \in \mathbb{Z}$), uno dei quali è il cluster riservato ai nodi sorgente dell'infezione e gli altri n rappresentano le n regioni. Ogni cluster può avere un fattore distintivo che lo differenzia dagli altri e i nodi contenuti in un determinato cluster avrebbero pertanto delle caratteristiche che si basano su quel fattore distintivo. Un esempio di fattore distintivo potrebbe essere un "timbro" univoco per ogni nodo. Il concetto di timbro può essere definito prendendo in prestito concetti dall'acustica e dall'anatomia umana, secondo la quale la voce di ogni essere umano è differente in quanto è possibile individuare, digitalizzando la registrazione analogica della voce, le frequenze che compongono per la maggior parte il suono emesso dalle corde vocali, rendendo di fatto un essere umano riconoscibile

quasi univocamente soltanto dalla sua voce. Il timbro di voce dell'essere umano rappresenta idealmente l'impronta digitale delle sue corde vocali, l'insieme delle frequenze più intense che esso riesce ad emettere. Quindi, ogni cluster potrebbe avere un timbro identificativo e i nodi che ne fanno parte hanno un timbro identificativo molto simile ad esso, in modo tale da poter essere riconosciuti universalmente come elementi che fanno parte di quel cluster. Per affermare che un nodo x faccia parte di un determinato cluster, si deve controllare la similarità tra il fattore distintivo del cluster e quello del nodo x: se questo valore è inferiore a una certa soglia τ, allora il nodo x farà parte del cluster, altrimenti no.

Se il fattore distintivo fosse un elemento bidimensionale, cioè una coppia di valori, la simulazione potrebbe essere visualizzata graficamente tramite la costruzione di un piano cartesiano (il cui centro di coordinate $(0, 0)$ è il valore di default del fattore distintivo), sul quale vengono rappresentate le n regioni con n cerchi, i cui centri rappresentano le coordinate (x_{R_i}, y_{R_i}) del fattore distintivo della regione. Maggiore è la similarità tra il fattore distintivo della regione e quello del nodo x di coordinate (x_0, y_0), minore sarà la sua distanza sul piano dal punto. Si può affermare che:

$$X \in R_i \;\Leftrightarrow\; (x_0 - x_{R_i})^2 + (y_0 - y_{R_i})^2 \leq \tau^2 \tag{2.1}$$

dove τ_i è il fattore di threshold della regione R_i che indica il raggio del cerchio corrispondente alla regione sul grafico.

Ogni nodo potrebbe avere un indice $\rho \in \mathbb{N}$ di resistenza all'infezione: ρ indica quanti *pacchetti* infetti[1] può ricevere ed elaborare quel nodo prima di soccombere al virus. Questo indice può essere casuale o può essere calcolato secondo un determinato criterio: ad esempio, prima di assegnare l'indice di resistenza a qualsiasi nodo, dopo la creazione dei nodi all'interno di una regione si possono considerare un numero $\gamma \in \mathbb{Z}$ di coppie di nodi interni alla stessa regione, direttamente proporzionale al numero di nodi in essa contenuti (più host saranno nella regione e più coppie si prenderanno in considerazione), e, dopo aver calcolato lo scarto quadratico medio tra i fattori distintivi di entrambi i nodi di una coppia, si può reiterare il processo per ogni coppia scelta per poi trovare la media di tutti questi scarti, ottenendo un valore di similarità medio tra le coppie in esame, sulla base del quale calcolare poi l'indice di resistenza. Utilizzando questo criterio, si potrebbe calcolare l'indice di

[1] Un *pacchetto* infetto è un pacchetto trasmesso da un nodo infetto verso un nodo ancora sano.

resistenza comune a tutti i nodi della regione in modo tale da renderlo direttamente o inversamente proporzionale rispetto alla media delle similarità delle coppie: nel primo caso, se i fattori distintivi dei nodi all'interno di quella regione sono molto simili tra di loro, la resistenza del singolo nodo è alta, altrimenti sarà bassa.

Quanto discusso finora non considera minimamente elementi che possono garantire la sicurezza generale della rete. Per "contrastare" l'eventuale infezione, ogni nodo x generato potrebbe possedere un indice di accettazione α che rappresenta il limite massimo di discrepanza tra il proprio fattore distintivo e quello di un nodo y intenzionato a stabilire un *encounter*; se il nodo x rileva un *encounter* in arrivo da parte del nodo y, l'*encounter* verrà immediatamente tagliato, rifiutando la connessione, se e solo se $D(x, y) < \alpha$, dove D è la funzione che calcola la distanza euclidea tra due punti. La disuguaglianza è verificata quando il fattore distintivo di y in arrivo è troppo distante da quello di x: ciò implica che x reputa y una "minaccia" a causa della diversità del suo fattore distintivo rispetto al proprio. È importante sottolineare la differenza tra l'indice di resistenza e l'indice di accettazione: la resistenza del nodo si riferisce a quanti pacchetti infetti il nodo può ricevere al massimo, mentre l'indice di accettazione si riferisce alla massima distanza accettabile tra il fattore distintivo del nodo sorgente e quello del nodo di destinazione per instaurare una connessione tra i due host.

2.3 Routing tra nodi

Il routing all'interno di un'architettura di rete DTN è un grande problema al quale possono esistere le più disparate soluzioni, che sfruttano approcci diversi. In mancanza di una categorizzazione formale dei possibili protocolli di instradamento di un pacchetto in una DTN, in questa sezione viene proposta una loro possibile classificazione informale, scientificamente inaccurata. Vengono di seguito discusse alcune metodologie di routing che gli algoritmi potrebbero utilizzare in un'architettura di rete delay-tolerant reale.

2.3.1 Epidemic routing

Un esempio di algoritmo epidemico banale, identico a quello che verrà utilizzato per l'implementazione della simulazione presentata in questo documento, potrebbe

prevedere che il nodo sorgente possa avere uno dei seguenti comportamenti: Una possibile classe di algoritmi di routing potrebbe essere quella degli algoritmi epidemici. Questi algoritmi si basano su un semplice concetto: ogni nodo all'interno della rete, non appena è pronto a comunicare, in base alle proprie intenzioni invia in broadcast su tutti i suoi canali di comunicazione attivi in quel momento lo stesso pacchetto a tutti. In un contesto epidemiologico, gli algoritmi di tipo epidemico sono quindi quelli più indiziati per permettere a un elemento infettivo di diffondersi capillarmente per tutta la rete. L'epidemic routing è un approccio di instradamento ampiamente esplorato: a partire da algoritmi epidemici più semplici come *Epidemic Routing Protocol* (ERP) [11]

- il nodo inoltra in broadcast il prossimo pacchetto contenuto nella propria coda (completando il processo di *store-and-forward* iniziato con la ricezione e la conservazione di quel pacchetto) tentando un trasferimento di custodia, del cui esito positivo non può essere certo, data l'assenza di *ACK* come conferma di trasmissione del pacchetto;

- in assenza di pacchetti da inoltrare, il nodo trasmette un pacchetto generato da sé in broadcast a tutti i canali di comunicazione attivi.

In fase di ricezione di un pacchetto, il nodo ricevente controlla se l'indirizzo (logico o fisico) di destinazione finale del pacchetto coincide con il proprio indirizzo: in caso affermativo, elabora il pacchetto, altrimenti lo conserva nella propria coda secondo il meccanismo di *store-and-forward* per poi favorirne un successivo reinoltro verso altre destinazioni. Un occhio attento potrebbe riscontrare già a prima vista diversi limiti dell'algoritmo in questione: ad esempio, esso non ha coscienza di un eventuale *hop limit* del singolo pacchetto, infatti se il pacchetto avesse un limite massimo di salti che può effettuare da un nodo router a un altro, la probabilità che il pacchetto "scada" prima di arrivare a destinazione sarebbe non trascurabile e la trasmissione verso nodi molto distanti sarebbe molto difficoltosa. Inoltre, un continuo invio in broadcast potrebbe rimandare indietro verso il mittente pacchetti già spediti al destinatario del trasferimento di custodia del pacchetto, creando un *flooding* ingiustificato di pacchetti duplicati che rallenterebbero gravemente le prestazioni della rete, intasando anche le poco capienti code di ricezione dei singoli nodi e di

conseguenza mettendo in difficoltà l'hardware dei singoli nodi; tuttavia, questo problema potrebbe risolversi utilizzando un meccanismo "a finestra" per classificare i pacchetti troppo vecchi o già ricevuti e scartarli.

2.3.2 Algoritmi epidemici ottimizzati

Algoritmi epidemici banali che fanno routing inviando in broadcast, come quello appena descritto, consumano enormi quantità di risorse di rete, di fatto sprecando la banda a disposizione e intasando velocemente le code di ridistribuzione dei singoli nodi. Per ovviare ai problemi generati dal *flooding*, esistono degli algoritmi epidemici ottimizzati che, grazie ad alcune accortezze, riescono a limitare anche di molto lo spreco complessivo delle risorse fisiche dei singoli hardware, accelerando anche le prestazioni della rete e aumentando il throughput di ogni nodo. Di seguito, vengono citati due esempi di ottimizzazione di un algoritmo epidemico di base.

Limitare i pacchetti duplicati Il *flooding* è la causa principale della duplicazione dei pacchetti all'interno di una rete di qualsiasi tipo. Alcuni algoritmi cercano di limitare questa pratica, ponendo un *cap* al numero di pacchetti copia generati, così da non inviare troppe volte lo stesso pacchetto a troppe destinazioni. Questa limitazione può ridurre la congestione di rete ma può anche avere implicazioni significative dal punto di vista della sicurezza, in quanto lo stesso dato non viene reinoltrato a chiunque ma soltanto a un numero ristretto di nodi, almeno a livello locale. La limitazione delle copie può avvenire a livello parametrico (cioè stabilendo un semplice limite numerico ai pacchetti uguali che un nodo può inviare ai propri contatti vicini) oppure secondo una policy sulla base della quale le copie vengono inviate soltanto a determinati nodi [12].

La duplicazione di pacchetti può essere limitata anche controllando alcune proprietà del nodo destinazione prima di inviargli un pacchetto. Nell'algoritmo di epidemic routing illustrato in [13], vengono considerati il livello di carica della batteria del nodo intermedio che inoltrerà il pacchetto e la dimensione del suo buffer di ricezione. Se questi valori rilevati sono sotto una certa soglia stabilita a priori, allora il pacchetto non può essere inviato a quel nodo, in quanto non avrebbe abbastanza energia per continuare la catena di custodia e non riuscirebbe a garantire con sicurezza il mantenimento del pacchetto all'interno del proprio buffer di storage. Un

algoritmo basato su simili principi viene presentato in [14]: a differenza del precedente, questo algoritmo si pone come obiettivo quello di massimizzare il risparmio energetico dei nodi senza inficiare sull'efficienza del routing all'interno della rete, aumentando al contempo la probabilità che la trasmissione di un pacchetto verso la destinazione vada a buon fine.

Immunità del pacchetto Gli algoritmi epidemici di tipo *immunity-based* sono fondati sugli stessi principi degli algoritmi epidemici di base, migliorando però di molto le prestazioni globali della rete. Negli algoritmi epidemici di base, ad ogni encounter stabilito tra un nodo A e un nodo B, i due nodi si scambiano un "vettore" composto dai messaggi contenuti nei rispettivi spazi di memoria riservata alla conservazione dei pacchetti ricevuti. A questo punto, A cerca nel vettore di B eventuali messaggi che non ha ricevuto, così da fargli una ulteriore richiesta per ottenerli, e viceversa. Questa logica di trasferimento può essere bidirezionale, come nel caso appena analizzato, oppure unidirezionale, se il nodo A usa una logica *push-pull*[2] [12].

Negli algoritmi epidemici *immunity-based*, invece, A scambia con B ben due vettori: un vettore m che contiene i messaggi descritti poc'anzi e un vettore i che contiene i messaggi "immuni", cioè quei messaggi che hanno A come nodo destinazione finale e che non devono essere più spostati da un nodo all'altro. Dopo lo scambio di questi vettori, entrambi vengono modificati, con il vettore i di ognuno dei nodi che comprenderà anche eventuali nuovi messaggi ricevuti la cui destinazione finale è proprio il nodo che adesso li possiede. In questo modo, ognuno dei due nodi blocca il trasferimento dei pacchetti contenuti nel vettore i, rendendoli immuni a ogni tipo di tentativo di routing verso l'esterno.

2.3.3 Class-conscious routing

Gli algoritmi di class-conscious routing sono una categoria di algoritmi che, a seconda delle policy di inoltro basate su criteri di potenza e di affidabilità del singolo nodo, scelgono come possibile prossima destinazione di un pacchetto un nodo all'interno di un insieme ristretto, costituito dai nodi più potenti, che accentrano su di sé la maggior parte delle operazioni effettuate nella rete.

[2]Nella logica push-pull, dopo lo scambio dei vettori A richiede a B i messaggi che vuole acquisire (*pull*) e nel frattempo invia a B i messaggi di cui deve liberarsi (*push*).

In un'architettura DTN realistica, gli algoritmi di routing solitamente si basano su tecniche di *hill climbing*. L'hill climbing è una metodologia iterativa di ottimizzazione di una ricerca locale in cui si parte da una soluzione arbitraria al problema in esame e la si modifica in modo incrementale per trovare un'altra soluzione leggermente migliore della precedente; si ripete questo procedimento fin quando la soluzione n non mostra miglioramenti rispetto alla soluzione $n-1$, non garantendo tuttavia che questa soluzione sia la più ottimale, essendo questo un algoritmo euristico. Usando questo genere di algoritmi per il routing dei pacchetti, le soluzioni (i path) che si ottengono coinvolgono sempre un gruppo ristretto di nodi, cioè quelli che riescono in qualche modo a garantire un'affidabilità maggiore, che riescano ad inoltrare correttamente il pacchetto nel tempo minore possibile anche limitando lo store-and-forward: ciò vale principalmente per i nodi che riescono a instaurare connessioni in modo più stabile rispetto a quelli che hanno un'attività saltuaria. Il fatto che solo un gruppo di nodi venga coinvolto negli inoltri e negli *handovers*[3], fa sì che, statisticamente, i nodi più potenti (circa il 10% del totale) si prendano carico di più della metà degli inoltri e degli handovers, spesso sovraccaricandosi e lasciando quasi del tutto inoperativi gli altri nodi [15]. Questo approccio genera una "casta" di nodi inoltratori che, su larga scala, peggiora di molto le prestazioni ottimali della rete. Per risolvere questo problema, esistono altri approcci per affrontare il problema del routing, come illustrato nella sezione successiva.

2.3.4 Interaction-based routing

Per mitigare i problemi di inefficienza e di inoperatività estesa della rete provocati dagli algoritmi di class-conscious routing, si possono utilizzare algoritmi che si basano sia sulla capacità di avere più canali di comunicazione attivi contemporaneamente, sia sulla probabilità di mantenere un *encounter* stabile nel tempo. In questa sezione, questi algoritmi vengono definiti informalmente come algoritmi *interaction-based*[4]. Un esempio di algoritmo che utilizza un approccio interaction-based è l'algoritmo *FairRoute* [15], il quale si basa su due concetti principali:

[3]Nell'ambito delle comunicazioni wireless, per handover si intende la procedura per la quale viene cambiato il canale usato dalla connessione di un terminale mobile alla rete wireless mantenendo attiva la comunicazione.

[4]Un algoritmo *interaction-based* è basato sulla capacità di interazione di un nodo con altri nodi, osservata nel tempo.

- **forza d'interazione percepita**: un indicatore che descrive la probabilità che un contatto rimanga stabilmente nel tempo.
- **assortatività**: in una rete assortativa i nodi di grado più alto (cioè che si collegano con un numero maggiore di nodi) preferiscono connettersi con nodi di grado similmente alto;

Forza d'interazione percepita L'algoritmo definisce due tipi di forza di interazione percepita: σ_{ij} forza d'interazione tra i e j nel breve termine e λ_{ij} forza di interazione tra i e j nel lungo termine. Nel momento in cui si verifica un contatto tra i e j, col passare del tempo σ_{ij} diminuisce di un fattore esponenziale r_σ e λ_{ij} diminuisce di un fattore esponenziale r_λ, molto minore di r_σ, in quanto essendo λ_{ij} un valore riferito a un lasso di tempo molto lungo, la curva esponenziale decresce molto più lentamente per equilibrare la decrescita più veloce di σ_{ij}: al tempo t, i valori delle forze di interazione vengono aggiornati come segue [15]:

$$\sigma_{ik} = \sigma_{ik} e^{-r_\sigma (t - t_i)} \quad \forall k \in N_i \tag{2.2}$$

$$\lambda_{ik} = \lambda_{ik} e^{-r_\lambda (t - t_i)} \quad \forall k \in N_i \tag{2.3}$$

$$(\sigma_{ij},\, \lambda_{ij}) = (\sigma_{ij},\, \lambda_{ij}) + (1, 1), \tag{2.4}$$

dove N_i è la lista dei contatti del nodo i, t_i è il tempo in cui il nodo i ha avuto il suo ultimo contatto e t è il tempo corrente. Le equazioni 2.2 e 2.3 descrivono il decremento esponenziale delle due forze tra il nodo i e tutti i nodi k già incontrati, mentre l'equazione 2.4 descrive l'aumento lineare delle forze d'interazione tra i nodi i e j [15]. Si definisce anche la forza d'interazione complessiva s_{ij} tra i nodi i e j come segue:

$$s_{ij} = \lambda_{ij}(\lambda_{ij} - \sigma_{ij}). \tag{2.5}$$

Questo valore indica quanto sono frequenti le interazioni a lungo termine rispetto a quelle a breve termine ed è un parametro di riferimento utile a capire se i due nodi sono stabilmente connessi o meno. Infatti, se due nodi hanno valori di forza d'interazione simili sia per il breve che per il lungo termine, vuol dire che la loro attività di connessione è molto discontinua, con picchi frequenti di attività seguiti da assenza di comunicazione, segno di poca stabilità. Matematicamente, s_{ij} sarebbe un

valore basso perché la differenza $\lambda_{ij} - \sigma_{ij}$ costituirebbe un fattore di proporzionalità basso per λ_{ij}. Si definisca poi u_{ijk} valore di utilità per il nodo i del passaggio attraverso il nodo j di un messaggio con destinazione k, come segue, normalizzato dall'utilità totale:

$$u_{ijk} = \begin{cases} \frac{\lambda_{jk}(\lambda_{jk}-\sigma_{jk})}{\lambda_{jk}(\lambda_{jk}-\sigma_{jk})-\lambda_{ik}(\lambda_{ik}-\sigma_{ik})} & se\ \lambda_{ik} + \lambda_{jk} > 0 \\ 0 & altrimenti \end{cases} \tag{2.6}$$

La precedente equazione esprime l'utilità di j percepita da i per recapitare un messaggio a k. Si noti che l'utilità ha un valore non nullo se e solo se le forze d'interazione a lungo termine verso k di entrambi i nodi i e j sono entrambe non nulle [15]: ciò è vero a partire dal primo aggiornamento delle forze di interazione a breve e lungo termine di uno dei due nodi, così come definito nell'equazione 2.4. Se $u_{ijk} > \frac{1}{2}$ si desume che il nodo j possa trasmettere il pacchetto verso k in maniera più efficiente rispetto a quanto potrebbe fare i.

Per l'invio di un pacchetto verso una destinazione qualsiasi, invece, l'utilità di j percepita da i è così definita [15]:

$$u_{ij} = \frac{\sum_{k \in N_j} \lambda_{jk}(\lambda_{jk} - \sigma_{jk})}{\sum_{k \in N_j} \lambda_{jk}(\lambda_{jk} - \sigma_{jk}) + \sum_{k \in N_i} \lambda_{ik}(\lambda_{ik} - \sigma_{ik})}. \tag{2.7}$$

Questo valore indica quanto "conviene" a i passare un pacchetto a j per farlo arrivare ad una qualsiasi destinazione k compresa nella lista di contatti N_j del nodo j, ed è l'incidenza in percentuale delle forze d'interazione complessiva s_{jk} sulla somma totale delle forze d'interazione complessiva $s_{jk} + s_{ik}$, cioè l'insieme delle forze d'interazione che sia i che j hanno nei confronti di ogni destinazione k da loro raggiungibile: in poche parole, questo indice esprime quanto è stato "legato" in totale nel tempo, a breve e lungo termine, il nodo j a tutte le destinazioni k da esso raggiungibili rispetto a quanto sono stati legati in totale entrambi i nodi alle destinazioni k contenute nelle rispettive liste di contatti raggiungibili. Di conseguenza, se il rapporto $u_{ij} > \frac{1}{2}$ allora j ha avuto più interazione a lungo termine con le sue destinazioni raggiungibili e quindi i capisce che j può essere in grado di trasmettere il pacchetto verso k con maggiore probabilità di successo rispetto allo scenario in cui sia i stesso a trasmetterlo.

Assortative-based queue control La caratteristica assortativa di questo algoritmo permette di implementare un'euristica basata sulle code, definita **assortative-based queue control**. Essa fa sì che i nodi, avendo risorse limitate, cerchino di utilizzarle sempre massimizzando l'utilità ed evitando di sprecarle interagendo con nodi più deboli. A livello sociologico, si potrebbe fare un parallelismo con la società umana: euristicamente, le persone di un certo status sociale di solito non stanno con persone di status inferiore. Avendo constatato ciò, lo *status sociale* di un nodo i si può rappresentare figurativamente tramite la capienza Q_i della sua coda di inoltro, la quale si può utilizzare come misura dell'utilità percepita di un nodo da parte dagli altri: più la sua coda è capiente, più pacchetti potrà inoltrare e quindi è più probabile che sia percepito come utile da parte degli altri. Di conseguenza, i nodi inoltreranno un pacchetto se e solo se la richiesta di inoltro proviene da un nodo di status uguale o superiore. In questo modo, se un nodo i ha una coda molto capiente ma è già ampiamente occupata, può inviare una richiesta di inoltro ad un nodo j di status inferiore, il quale accetterà il trasferimento di custodia.

Questo procedimento crea l'effetto collaterale di rendere operativi anche nodi che sono lontani da quelli con uno status sociale più alto della media, coinvolgendo nella maniera più opportuna anche nodi poco potenti. Con questa euristica, i nodi con status più alto inoltreranno i pacchetti più velocemente. In generale, il nodo i inoltrerà al nodo j il messaggio di destinazione k se e solo se si verifica una delle seguenti condizioni, come definito in [15]:

$$\begin{aligned} u_{ij} > \frac{1}{2} \quad &\wedge \quad \lambda_{ik} + \lambda_{jk} = 0 \quad \wedge \quad (Q_j \leq Q_i) \\ u_{ijk} > \frac{1}{2} \quad &\wedge \quad \lambda_{ik} + \lambda_{jk} > 0 \quad \wedge \quad (Q_j \leq Q_i) \\ u_{ijk} = 1 \quad &\wedge \quad \lambda_{ik} + \lambda_{jk} > 0 \end{aligned} \tag{2.8}$$

Riassumendo quanto espresso dalle condizioni 2.8, nel caso in cui la capienza della coda della sorgente i è maggiore di quella dell'intermediario j, l'inoltro avverrà con successo in due casi:

1. la somma delle forze di interazione a lungo termine di i e j verso k è positiva ($\lambda_{ik} + \lambda_{jk} > 0$) e i percepisce j come molto utile ($u_{ijk} > \frac{1}{2}$) per inoltrare un pacchetto verso k;

2. entrambe le forze d'interazione a lungo termine considerate sono nulle ($\lambda_{ik} + \lambda_{jk} = 0$) e i percepisce j come molto utile ($u_{ij} > \frac{1}{2}$) per inoltrare un pacchetto verso qualsiasi destinazione.

Nel caso in cui invece la coda dell'intermediario j sia più capiente della coda della sorgente i, vuol dire che i viene considerato come nodo di status inferiore rispetto a j e il passaggio del pacchetto da i a j con destinazione k avverrà solo a patto che sussista una condizione più stringente, cioè se e solo se l'utilità di j percepita da i per inoltrare il pacchetto a quella determinata destinazione k è massima, da cui deriva che la somma delle forze di interazione a lungo termine di entrambi i nodi verso k è ovviamente positiva. Questo può verificarsi, ad esempio, quando j e k coincidono.

Interaction-based routing probabilistico: *PRoPHET* Esistono altri algoritmi che si basano su concetti probabilistici molto simili a quelli illustrati in *FairRoute*. Ad esempio, l'algoritmo *PRoPHET* [16] conserva e utilizza la "cronologia" degli encounters di un nodo a per calcolarne la probabilità di inviare un messaggio alla destinazione b. In particolare, il valore di questa metrica di probabilità $P_{(a,b)} \in [0, 1]$ viene aggiornato ad ogni encounter stabilito tra a e b e, con il passare del tempo, se i due nodi non si incontrano più, questo valore subisce un processo di "invecchiamento" (o *aging*) che lo porta a ridursi sempre più in assenza di nuovi encounters tra i due nodi. Inoltre, $P_{(a,b)}$ ha anche una proprietà *transitiva* che si manifesta nello scenario in cui se a incontra spesso b, il quale a sua volta incontra spesso c, allora c può riuscire a inoltrare i pacchetti ricevuti e destinati ad a con una buona probabilità di successo. Per instradare il pacchetto verso una destinazione k, ogni nodo dovrà quindi valutare tutti i valori di probabilità $P_{(a,x)}$, per ogni nodo x raggiungibile in quel momento, inviando il pacchetto al nodo che garantisce una maggiore probabilità che il pacchetto destinato a k arrivi a destinazione. Tuttavia, non esiste un approccio matematico per la risoluzione di questo problema, in quanto potrebbero verificarsi scenari sempre diversi che non si prestano a una generalizzazione delle scelte di routing da effettuare: ad esempio, potrebbe succedere che un nodo inoltri a più nodi raggiungibili il pacchetto da recapitare, per cercare di aumentare il più possibile la probabilità che esso arrivi a destinazione, oppure potrebbe fare un solo invio al nodo con la metrica di probabilità più alta.

2.3.5 Hop-based routing

Gli algoritmi di routing hop-based si basano sul calcolare il prossimo hop che risulta ottimale in termini di distanza e di lunghezza della catena di salti che porta il pacchetto dal mittente al destinatario finale. Questo genere di metodologie si prefigge come obiettivo quello di minimizzare il più possibile il numero di salti da far compiere a un pacchetto prima che giunga a destinazione; l'algoritmo non segue un approccio greedy, in quanto il nodo localmente più ottimale in termini di distanza potrebbe rivelarsi invece molto più distante rispetto a tutti gli altri nodi successivi e quindi la trasmissione potrebbe impiegare più tempo se passa da quel nodo, mentre avrebbe potuto impiegarne di meno se il pacchetto fosse passato direttamente a un nodo leggermente più lontano ma con una posizione globalmente più ottimale rispetto alla destinazione finale. Questo tipo di algoritmi presentano diversi aspetti positivi sul fronte dell'efficienza, dal momento che il salto di un pacchetto da un router a un altro impiega un determinato intervallo temporale e scegliendo bene il percorso da fare si potrebbe abbassare il tempo di trasmissione anche a fronte di un numero di hop maggiore, ma anche sul lato della sicurezza, in quanto si riduce la platea di nodi che ricevono il pacchetto e che, potenzialmente, possono leggerne i dati. In questo modo, viene garantito un livello di privacy del singolo pacchetto maggiore rispetto a un semplice algoritmo che invia in broadcast.

Tuttavia, per conoscere il nodo che rappresenta l'hop più vicino, il nodo deve verificare il numero di hop necessari per arrivare a ognuno dei nodi in quel momento da esso raggiungibili, facendo quindi una sorta di *polling* per classificare i nodi in base a questa metrica. L'esecuzione di questo "check" farebbe capire ai nodi raggiungibili che il nodo sorgente vuole effettivamente comunicare e questa informazione potrebbe rappresentare un buco di privacy: una soluzione potrebbe essere quella di incaricare un intermediario, come un gateway, di raccogliere questa metrica per conto del nodo sorgente, in modo tale che i nodi destinazione non conoscano la sorgente di questa richiesta; tuttavia, ciò rappresenterebbe un single point of failure a livello di privacy, dal momento che se il gateway non riuscisse a soddisfare questa richiesta perché offline, in base a come l'algoritmo di routing affronta questa evenienza il routing potrebbe smettere di funzionare oppure il nodo sorgente potrebbe essere obbligato a quel punto ad effettuare la richiesta da sé, esponendo quindi al pubblico il desiderio di voler trasmettere al nodo più vicino.

2.3.6 Privacy-preserving routing

Quasi tutte le categorie di algoritmi considerati fino ad ora non applicano alcuna pratica di sicurezza, delegando la responsabilità di mettere in sicurezza la trasmissione e i dati trasmessi a protocolli esterni. Gli algoritmi di privacy-preserving routing, invece, si concentrano su entrambi gli aspetti, cercando di fornire una soluzione all'instradamento ma inglobando anche protocolli di sicurezza utili ad evitare la massiccia diffusione del pacchetto e garantendo autenticazione (o anonimato, in base all'approccio utilizzato), integrità e segretezza dei pacchetti inviati. Sono stati studiati diversi modi per rendere più sicura una trasmissione all'interno di una DTN: alcuni algoritmi utilizzano metodologie crittografiche per criptare le connessioni o per rendere il tutto anonimo, altri ancora hanno un approccio "protettivo" come l'algoritmo *spray and wait* [17], che essenzialmente si basa sul trasmettere un pacchetto soltanto a un certo numero di interlocutori distinti vicini (fase di *spray*), che una volta ricevuto il pacchetto aspettano fin quando non incontrano direttamente la destinazione per completare il recapito (fase di *wait*).

Più in generale, come discusso anche in [18], la sicurezza in una DTN può essere intesa come protezione di diversi aspetti della comunicazione. Infatti, gli algoritmi di questo tipo vengono solitamente classificati, in base all'elemento che proteggono, nelle seguenti categorie:

- ID privacy-preserving: mettono in sicurezza indirizzi fisici o elementi identificativi univoci di ciascun nodo [19] grazie a processi crittografici;
- location privacy-preserving: nascondono la posizione degli interlocutori, garantendo anonimato nella comunicazione, ad esempio frammentando un pacchetto in piccoli segmenti, ciascuno dei quali viene mandato a un diverso intermediario per poi venire recapitati tutti allo stesso destinatario, il quale rimetterà insieme i pezzi per elaborare il pacchetto, come avviene nell'algoritmo ALAR [20];
- message content privacy-preserving: mettono in campo delle strategie e delle procedure, basate su attributi di affidabilità propri di ogni nodo, che fanno sì che il contenuto del messaggio, criptato, possa essere decifrato e letto soltanto dal nodo o dai nodi ai quali era inizialmente destinato; un esempio potrebbe essere un algoritmo che rivela il contenuto di un pacchetto soltanto dopo aver

controllato che lo "status sociale" del destinatario risulti simile a quello del mittente oppure che il mittente e il destinatario abbiano una sorta di relazione tra loro. Questo procedimento si basa sul concetto empirico che una persona è più propensa a rivelare un segreto ad un amico con cui ha un rapporto consolidato, piuttosto che a uno sconosciuto;

- routing utility privacy-preserving: evitano la diffusione di informazioni circa l'utilità del nodo come intermediario, cioè sulla sua capacità di reinoltrare in modo efficiente verso una generica destinazione. Proteggere questa proprietà dagli occhi di terzi può ridurre le probabilità che vengano costruiti degli attacchi denial-of-service verso un nodo più "potente" che fa da router con un alto traffico: in base al modo in cui i pacchetti vengono trasmessi all'interno della rete, ciò potrebbe rappresentare un problema, dal momento che alla caduta di quel determinato nodo, una grossa fetta delle comunicazioni che avvengono all'interno della rete si fermerebbe drasticamente, causando un danno calcolato sulla base della sensibilità dei dati che passano per quel nodo e sulla base del periodo di downtime, che potrebbe essere breve ma anche lungo.

Chapter 3

netdemic: la simulazione epidemica

L'architettura DTN si presta ad applicazioni pratiche che coinvolgono grandi sistemi di dispositivi non permanentemente connessi tra di loro e nei quali il transito di dati o l'accesso di un nodo ad una regione deve essere reso affidabile e sicuro grazie all'introduzione di protocolli di sicurezza.
In questo progetto, invece, viene illustrata un'applicazione pratica a scopo simulativo di un'architettura di delay-tolerant network semplificata, che trascura diverse problematiche fondamentali in una reale architettura di questo tipo, come quelle sulla sicurezza della rete.

3.1 Introduzione a *netdemic*

netdemic è una simulazione epidemica che sfrutta un'implementazione virtuale dell'architettura di rete DTN, nella quale uno o più nodi malevoli, vettori di contagio di un virus letale, cercano di contagiare tutta la rete entrando in contatto con quanti più nodi possibili, i quali nel frattempo comunicano con il resto della rete e, se contagiati, possono partecipare all'infezione.

Struttura di base La simulazione viene illustrata all'interno del game engine Unity (ver. 2021.3.8f1), sul quale viene costruita una visualizzazione grafica di uno spazio bidimensionale suddiviso in cluster circolari separati tra loro e residenti in uno spazio vuoto, dove ogni cluster rappresenta una regione. All'interno di una stessa regione, coesistono diversi nodi che possono essere connessi o meno tra di loro

ad un certo istante e ogni nodo ha un fattore distintivo univoco rappresentato da un vettore di 22000 valori casuali nell'intervallo $[0, 1]$. L'insieme di valori ottenuto costituisce una sorta di "timbro" caratteristico del nodo, facendo un parallelismo con la voce umana, unica per ogni persona. Tuttavia, esistono gruppi di nodi con timbri simili, pur mantenendo la loro unicità quando presi singolarmente: questi gruppi formano proprio le regioni.

Obiettivo della simulazione L'obiettivo della simulazione è quello di mostrare cosa potrebbe avvenire all'interno di una rete delay-tolerant posta in una condizione di "emergenza" epidemiologica e in presenza di misure di sicurezza limitate o addirittura assenti: attraverso l'ausilio dell'algoritmo di routing epidemico elementare implementato, proprio come in un sistema di particelle portatrici di un virus, i nodi facenti parte della rete vengono tutti contagiati man mano, in maniera più o meno rapida in base alla concentrazione di nodi presente nelle singole regioni. Reti tra loro molto vicine e molto affollate hanno una più alta probabilità di soccombere molto rapidamente al virus rispetto a reti tra loro molto lontane o con una presenza rarefatta di hosts al loro interno.

3.2 Costruzione e avvio della simulazione

La simulazione viene avviata dall'utente, che tramite il menù di configurazione interno ad essa può tararne il comportamento in base alle proprie esigenze. *netdemic* permette una personalizzazione abbastanza ampia di ciò che avviene all'interno della simulazione attraverso alcuni parametri con cui si possono gestire diversi aspetti della simulazione: la dimensione della stessa, la potenza e la viralità dell'infezione, ma anche l'efficienza e la frequenza dei collegamenti tra i nodi. La simulazione viene costruita sulla base di questi parametri, seguendo le fasi preliminari di seguito discusse.

3.2.1 Configurazione iniziale dei parametri di avvio

Il comportamento della simulazione viene regolato a seconda di alcuni parametri, la maggior parte dei quali sono inseriti dall'utente in fase di avvio della simulazione.

Vengono di seguito elencati alcuni parametri fondamentali, la cui definizione tornerà utile nelle prossime sezioni.

Table 3.1: Tabella di riferimento dei parametri principali di netdemic

Descrizione del parametro	Simbolo	Valore/i possibili	Intero	Random
Frequenza di invio in broadcast (in secondi)	F_B	[0.5, 5]	No	No
Probabilità globale di fare un encounter	$P[E]$	[0.01, 0.1]	No	No
Probabilità globale che una connessione cada	$P[D]$	[0.01, 0.2]	No	No
Probabilità globale di trasmettere un pacchetto	$P[T]$	[0.05, 1]	No	No
Numero di regioni	n	[2, 64]	Sì	No
Massimo numero di nodi per regione	$\hat{n}$	[2, 16]	Sì	No
Carico virale iniziale	ν	[1, 64]	Sì	No
Varianza della distribuzione gaussiana di un timbro	σ^2	[200, 600]	No	Sì
Estensione intervallo di generazione rumore di un timbro	h	[0.1, 2]	No	Sì
Numero di nodi generati	η	$[n, n\hat{n}]$	Sì	Sì
Capacità di store di un nodo (in pacchetti)	`size(x)`	[5, 100]	Sì	Sì
Potenza emissiva di un nodo	W_x	[2, 5]	No	Sì
Velocità di movimento di un nodo	v_x	[1, 3]	No	Sì
Limite di resistenza di un nodo sano	Γ_x	[8, 16]	Sì	Sì
Velocità di contagio di un nodo infetto	γ_{ν_i}	[1, 10]	No	Sì

3.2.2 Generazione della rete virtuale

Sul campo di simulazione che ospiterà la rete virtuale, viene inizialmente rappresentata una meshgrid[1] di otto righe e sedici colonne. Vengono poi scelte casualmente n celle della meshgrid (con $2 \leq n \leq 64$, $n \in \mathbb{Z}$), ognuna delle quali ospiterà una delle n regioni. Per ogni regione viene poi assegnata un'onda archetipica, la cui frequenza fondamentale varia in base al numero n scelto e in modo tale che tutte le frequenze fondamentali generate siano equidistanti fra loro.

Dopo la costruzione degli archetipi regionali, ognuna delle n regioni viene riempita con m nodi ($1 \leq m \leq 16$, $m \in \mathbb{Z}$), i cui timbri caratteristici vengono creati da una funzione generatrice f a partire dall'archetipo della regione d'appartenenza, creando forme d'onda simili all'archetipo regionale ma univoche all'interno della regione

[1] Una meshgrid è il prodotto cartesiano di due vettori n e m, che determinano rispettivamente il numero delle righe e delle colonne. A ogni cella della meshgrid viene associato un singolo punto nel piano.

e in generale dell'intera rete. La diversità delle varie forme d'onda è però limitata: a ogni nodo viene assegnato un valore $h \in \mathbb{R}$ nell'intervallo [0.1, 2] che rappresenta la lunghezza di un intorno I di centro $h = 0$ dal quale estrarre un valore casuale $\rho \in \mathbb{R}$, che influenzerà la similarità tra l'onda della regione e l'onda del nodo in questione come descritto nel paragrafo successivo.

3.2.3 Algoritmo di generazione dei timbri

Qualsiasi timbro generato all'interno della simulazione viene calcolato utilizzando una coppia di parametri $(\mu, \sigma) \in \mathbb{R}^2$ che rappresentano rispettivamente la media e la deviazione standard di una distribuzione gaussiana dalla quale verranno campionati esattamente 22000 valori in modo casuale. Questo numero non è casuale, in quanto ogni timbro viene rappresentato come se fosse lo spettro delle frequenze udibili di un file audio: infatti, è composto da una lista di campioni Σ dove al generico elemento s_i (che corrisponde alla frequenza i Hz) corrisponde un valore di intensità nell'intervallo $[0, 100]$, che rappresenta il rapporto tra il numero di volte in cui è stato estratto casualmente il valore di frequenza corrispondente all'interno delle 22000 estrazioni, diviso per il numero delle estrazioni.

Quando viene creato un nodo oppure una regione, viene generato un timbro per quell'entità. Ad una regione viene sempre assegnato un timbro generato da zero, senza l'influenza di alcun parametro, mentre ad ogni nodo di cui è composta viene assegnato un timbro generato a partire da quello regionale. Siano dati i seguenti elementi:

- un nuovo timbro t tale che $\Sigma_t = \varnothing$;
- una lista l di 22000 valori estratti casualmente da una distribuzione gaussiana d con media μ e varianza σ^2;
- una funzione `count(l, i)` che conta il numero di k_i occorrenze della frequenza i in l.

Si definisce Σ_t la lista dei campioni generati dalla funzione `count(l, i)` (iterata su tutta la lista l), se t fosse il timbro di una generica regione R_j (con $j \in \mathbb{Z}$, numero

identificativo progressivo della regione), come segue:

$$\Sigma_t = \left\{ s_i : 0 \leq i < 22000, s_i = \frac{100 k_i}{22000} \right\} \tag{3.1}$$

Si definiscono inoltre la funzione $c(s, \rho) : \mathbb{R} \times [-\frac{h}{2}, \frac{h}{2}] \rightarrow [0, 100]$ e Σ'_t lista dei campioni di t se fosse il timbro di un nodo $x \in R_j$, come segue:

$$c(s, \rho) = \begin{cases} 0 & \text{se } s + \rho < 0 \\ 100 & \text{se } s + \rho > 100 \\ s + \rho & \text{altrimenti} \end{cases} \tag{3.2}$$

$$\Sigma'_t = \left\{ s'_i : 0 \leq i < 22000, s'_i = c(s_i, \rho) \right\} \tag{3.3}$$

dove ρ rappresenta lo scarto (o rumore) da applicare ad ogni singolo campione del timbro t, in modo tale che il massimo errore quadratico medio (MSE) possibile tra l'archetipo T della regione R_j e il nuovo timbro t generato per il nodo x sia limitato superiormente da un valore $E_{R_j} \in \mathbb{R}$. Vale quindi la seguente affermazione:

$$x \in R_j \Leftrightarrow MSE(T, t) \leq E_{R_j} \tag{3.4}$$

Dopo questo processo, visualizzando graficamente la regione R_j e i nodi in essa contenuti su un piano cartesiano avente come origine degli assi l'archetipo della regione, si otterrà che i suoi nodi saranno distribuiti all'interno di un'area circolare di raggio E_{R_j}. Ciò implica che maggiore sarà l'errore quadratico medio tra l'archetipo regionale e la forma d'onda del nodo, maggiore sarà la distanza del nodo dall'origine degli assi.

Generazione del timbro dei nodi malevoli Un caso particolare di applicazione del precedente algoritmo si verifica quando, al momento della creazione dei ν nodi virali, viene assegnato loro lo stesso timbro t_ν. Questi nodi appaiono colorati di rosso (al contrario dei nodi sani, che vengono colorati di bianco) all'interno della visualizzazione grafica. Per via dell'implementazione del metodo di contagio utilizzato nella simulazione, è necessario che tutti i nodi virali abbiano lo stesso timbro, perché

Algorithm 1: Algoritmo di generazione di un timbro

```
Data: μ ≥ 0, σ² ≥ 0, 0.1 ≤ h ≤ 2
Result: t
e ← get_entity();
t ← create_empty_waveform();
i ← 1;
if e is Region then
    d ← gaussian_distribution(μ, σ);
    l ← sample_from_distribution(d);
    while i ≤ 22000 do
        k ← count(l, i);
        t[i] ← 100 * k/22000;
        i ← i + 1
    end
else
    T ← get_region_waveform(e);
    ρ ← random_noise(h);
    while i ≤ 22000 do
        if 0 ≤ T[i] + ρ ≤ 100 then
            t[i] ← T[i] + ρ;
        else
            if T[i] + ρ < 0 then
                t[i] ← 0;
            else
                t[i] ← 100;
            end
        end
    end
end
```

l'algoritmo di contagio non contempla che un nodo sano possa avere più infezioni di diverso tipo. L'algoritmo illustrato nel paragrafo precedente si può riassumere con lo pseudocodice che segue.

3.3 Propagazione del virus

Dopo la configurazione iniziale, con successiva generazione dei timbri di tutte le regioni e di tutti i nodi, la simulazione viene avviata e l'infezione comincia fin dal primo frame generato dal programma.

3.3.1 Instradamento di un pacchetto nella rete virtuale

Fin dal primo fotogramma visualizzato a schermo, la rete è pienamente operativa. I nodi trasmettono pacchetti solo agli host della loro regione d'appartenenza tramite un algoritmo di routing epidemico elementare, già discusso nella Sezione 2.3.1. Nella simulazione, per fini pratici, l'algoritmo di routing invia un pacchetto soltanto a destinatari sani, in quanto sarebbe inutile inviare un pacchetto a un nodo già infetto, dal momento che un nodo infetto smette di utilizzare la propria coda di storage dei pacchetti. Riprendendo quanto già detto, di seguito si fornisce lo pseudocodice dell'algoritmo di routing epidemico di *netdemic*.

3.3.2 Inizio dell'infezione

Dopo aver generato la rete virtuale e aver generato i ν nodi sorgente dell'infezione, a ognuno di essi viene assegnata una posizione casuale sul campo. Il generico nodo infetto z si muove all'interno della rete calcolando dapprima un punto P_0 all'interno di un cerchio di raggio r e centro uguale alla posizione corrente del nodo, per poi spostarsi verso di esso gradualmente, ripetendo il processo non appena la sua posizione coinciderà con P_0.

Il nodo z, non appena sufficientemente vicino, stabilirà un collegamento con il nodo più facilmente raggiungibile, cioè quelli più vicino nel piano. Ogni nodo della simulazione può tenere attivi contemporaneamente un massimo di tre collegamenti per fotogramma. In generale, il collegamento verrà stabilito solo se il nodo destinazione rientra nel raggio d'azione W_z del nodo infetto, cioè se e solo se

Algorithm 2: Algoritmo di routing di *netdemic*

```
Data: x ≠ NIL, size(x) ≠ 0
L ← get_active_encounters_list();
l ← length of list L;
p ← packet_dequeue();
if p is NIL then
    p ← create_packet();
end
if final destination of p is x then
    process_packet(p);
else
    if l > 0 then
        i ← 1;
        while i ≤ l do
            if L[i] is not infected then
                sendto(p, L[i]) ;      // sendto(p, x) invia p al nodo x
            end
            i ← i + 1;
        end
    end
end
```

$D(z, x) \leq W_z$, dove D è la funzione distanza euclidea tra due punti del piano. In realtà, il collegamento non avviene a prescindere, ma solo con una certa probabilità $P[E]$: ad ogni frame disegnato dal motore di gioco, viene campionato un numero reale casuale tra 0 e 1. Se questo numero è minore di $P[E]$ allora il collegamento viene stabilito, altrimenti il nodo sano rimarrà nel raggio d'azione del nodo infetto senza che esso vi si connetta. Tuttavia, questa procedura si ripete a ogni frame e dal momento che, in media, il motore di gioco visualizza più di 30 fotogrammi al secondo, la continua estrazione di valori casuali aumenta comunque la probabilità nel tempo di creare un encounter tra i due nodi.

Inoltre, dati due nodi x e y con rispettive potenze emissive W_x e W_y, i due nodi hanno distanza reciproca $D(x, y)$ e se vale che $W_x \geq D(x, y) \ \wedge \ W_y \geq D(x, y)$ allora verrà stabilito un collegamento bidirezionale, mentre se una delle due disuguaglianze non è vera, soltanto il nodo la cui potenza emissiva verifica la disugualianza vera stabilirà con l'altro nodo un collegamento esclusivamente unidirezionale. Questo penalizza i nodi con bassa potenza emissiva, in quanto devono avvicinarsi molto di più agli altri per comunicare.

3.3.3 Modalità di contagio: l'algoritmo di infezione

L'infezione non si diffonde tramite la distribuzione di pacchetti infetti per tutta la rete, ma tramite il contatto tra nodi sani e nodi infetti: infatti, un nodo sano molto lontano da tutti i nodi infetti non può essere facilmente contagiato, in quanto ha bisogno di essere contattato da un nodo già infetto. Di conseguenza, il ruolo dell'algoritmo di routing epidemico di *netdemic* nell'infezione è marginale.

L'obiettivo del nodo infetto z, una volta collegatosi ad un altro nodo x, è quello di inviare tramite un pacchetto il proprio timbro per "contagiare" quello di destinazione, aumentando di un'unità il numero k di pacchetti infetti[2] ricevuti da x dopo che il pacchetto è giunto a destinazione. Il contagio inizia quando il nodo infetto viene a contatto con un nodo sano e gli trasmette un pacchetto: l'infezione avviene applicando ai due timbri t_z e t_x una funzione di interpolazione lineare $L(a, b, f)$, dove f è il fattore di interpolazione nell'intervallo di numeri reali $[0, 1]$ costituito dal rapporto $\frac{k}{\Gamma_x}$. La lista di campioni risultante dall'iterazione della funzione di interpolazione su tutto il timbro t_x è definita come segue:

$$\Sigma_{x \to z} = \left\{ s_i : 0 \leq i < 22000, s_i = L(s_{x_i}, s_{z_i}, \frac{k}{\Gamma_x}),\ s_{x_i} \in \Sigma_x,\ s_{z_i} \in \Sigma_z \right\} \tag{3.5}$$

dove Σ_x è la lista dei campioni del timbro t_x al tempo t_0 (cioè l'istante iniziale di avvio della simulazione) e Σ_z è la lista dei campioni del timbro t_z al tempo t_0, mentre s_{x_i} e s_{z_i} costituiscono l'i-esimo campione dei rispettivi timbri. Per scelta implementativa, quando un nodo diventa infetto, esso rimane legato a livello logico alla propria regione, ma perde il suo legame spaziale, iniziando quindi a muoversi per tutta la rete anziché soltanto nell'area della regione di appartenenza, ad una velocità pari a $3v_x$, dove v_x è la velocità di movimento del nodo x (vedi Tabella 3.1).

Questo processo verrà ripetuto ad ogni ciclo di trasmissione *broadcast* in cui il collegamento è attivo, con una certa probabilità $P[T]$ (vedi Tabella 3.1). In altre parole, ogni F_B secondi tutti i nodi della simulazione trasmetteranno un pacchetto, inclusi gli infetti, e se in quel momento il collegamento da z a x è attivo, verrà eseguito questo algoritmo, di cui viene fornito lo pseudocodice a seguire.

Sotto il punto di vista degli encounter necessari a contagiare un nodo sano, si noti che l'algoritmo di infezione non è deterministico, ma dipende dal parametro di probabilità di trasmissione $P[T]$, utilizzato dal nodo per scegliere se trasmettere in quel fotogramma o meno: dal momento che esso può essere minore di 1, la trasmissione di un nuovo pacchetto infetto lungo un canale di comunicazione già attivo non è per nulla un evento certo. La natura pseudocasuale della frequenza di trasmissione non permette di determinare il numero di collegamenti con nodi malevoli che un nodo sano deve instaurare per essere completamente contagiato.

[2] Un pacchetto infetto è un normale pacchetto, il cui mittente è però un nodo infetto.

Ad esempio, si supponga che un nodo abbia un limite di resistenza $\Gamma = 10$: esso potrebbe divenire infetto grazie a un solo encounter a lungo termine con un nodo malevolo, che riesce a mantenere il contatto attivo quanto basta per inviare dieci pacchetti infetti nella stessa sessione, oppure potrebbe contagiarsi grazie a dieci contatti diversi, anche a brevissimo termine, con dieci nodi malevoli, ognuno dei quali riesce ad inviare il proprio pacchetto infetto alla vittima nel breve lasso di tempo in cui la connessione è attiva. Gli scenari possibili sono così numerosi da rendere quindi impossibile l'esistenza di una formulazione che esprima il numero minimo o massimo di collegamenti malevoli necessari per contagiare un nodo vittima.

Algorithm 3: Algoritmo di infezione tramite interpolazione

Data: $\Sigma_x \neq \varnothing,\ \Sigma_z \neq \varnothing,\ f \geq 0,\ node\_status \neq NIL$
Data: $\Gamma_x \neq 0$; `// Γ_x è il limite di resistenza di x`
$i \leftarrow 1$;
$k \leftarrow k + 1$;
while $i \leq 22000$ **do**
  $\Sigma_x[i] \leftarrow L(\Sigma_x[i], \Sigma_z[i], f)$;
end
if $k = \Gamma_x$ **then**
  $node\_status \leftarrow infected$;
end

La procedura non sarà eseguita soltanto nei seguenti casi:

1. quando si è fuori dal ciclo periodico di broadcast;
2. quando il nodo infetto vuole trasmettere ma non ha collegamenti attivi con alcun nodo sano;
3. il numero di pacchetti infetti ricevuti da x sfora il limite imposto dal suo indice di resistenza ($k \geq \Gamma_x$) e x viene definitivamente infettato dal virus, abilitandolo a contagiare i nodi a lui circostanti e colorandolo di rosso per segnalare il proprio stato;
4. il collegamento tra z e x viene tagliato a causa dell'allontanamento di uno dei due nodi, costringendo z (o un altro nodo infetto z_1 suo pari) a continuare il processo da dove si era interrotto, non appena avrà stabilito un nuovo *encounter* con x;
5. la connessione intermittente tra z e x cade per un errore o per un motivo diverso da quanto precedentemente detto, con una certa probabilità $P[D]$ (vedi Tabella 3.1).

Il processo di infezione termina quando tutti i nodi sono stati contagiati e risultano completamente infetti. L'infezione completa blocca ogni comunicazione tra i vari nodi, in quanto la comunicazione verso un nodo infetto non è prevista nella simulazione. Lo scenario di infezione completa prevede che tutti i nodi abbiano la stessa forma d'onda assegnata inizialmente a z.

Chapter 4

netdemic su Unity: implementazione in C#

netdemic è un software interamente sviluppato in linguaggio C# e integrato successivamente con Unity. In questo ultimo capitolo, verrà discussa la produzione del software, dal diagramma UML di progettazione iniziale all'implementazione di alcune meccaniche peculiari della simulazione.

4.1 Informazioni preliminari

Il codice di *netdemic* consta di 18 classi scritte in C#, di cui 4 interfacce o classi astratte e 2 inutilizzate, previste soltanto per eventuali funzionalità aggiuntive da implementare per scopi futuri. Nello sviluppo sono state utilizzate le librerie native del linguaggio e del motore grafico Unity, ma anche la libreria esterna *Math.NET Numerics* [21], che fornisce gli algoritmi fondamentali per la costruzione delle distribuzioni gaussiane utilizzate per la creazione dei timbri. *netdemic* gira su Unity in modalità multi-threading, grazie all'utilizzo della *Task Parallel Library (TPL)* di .NET Framework 4.x [22] per accelerare le prestazioni. Di base, Unity esegue il codice in modalità single-thread: la maggior parte del lavoro viene preso in carico dal thread principale (Main) sul quale Unity è in esecuzione. Come si vedrà in seguito, l'esecuzione di alcune funzionalità della simulazione richiede a run-time calcoli matematici semplici effettuati su grandi insiemi di numeri, per molte volte al secondo, il cui risultato deve essere poi disegnato a schermo nel minor tempo possibile. Se queste procedure venissero fatte eseguire al thread Main, esse lo bloccherebbero,

rallentando anche il rendering dei fotogrammi e di fatto causando continui e prolungati *freeze* del software.

A causa delle peculiarità di C# che non è possibile rappresentare con un classico diagramma UML, persistono delle difformità tra il codice effettivo e il diagramma costruito. Una su tutte è la presenza di molte variabili che nell'UML sono contrassegnate con accesso `public`, ma in realtà nel codice hanno accesso `private` e vengono accedute grazie alla definizione di una *property*: essa consiste nell'implementare in modo facile e veloce i metodi *getter* e *setter* per una determinata proprietà, stabilendo il livello di protezione per entrambi i metodi. Per rendere la rappresentazione dell'UML più semplice e più generica possibile, cioè applicabile anche a linguaggi diversi da C#, le variabili su cui è stata definita una *property* sono contrassegnate esclusivamente come `public` nell'UML.

Inoltre, nell'UML sono omesse molte proprietà e funzioni la cui implementazione è dovuta all'integrazione con Unity, come ad esempio le variabili di tipo `Vector3` e `GameObject`, le funzioni facenti parte del workflow di Unity stesso come le funzioni `Awake()` o `OnEnable()` di una classe di tipo `MonoBehaviour`, che a volte vengono usate come alternativa a un metodo costruttore per oggetti di quella specifica classe, ma anche le funzioni che lavorano con oggetti di tipo `UnityEvent` e anche alcune proprietà primitive che però vengono utilizzate da oggetti esclusivamente legati all'ambiente di Unity.

4.2 Progettazione del software: diagramma UML

Dopo aver definito nel dettaglio la maggior parte delle funzionalità principali da implementare all'interno del software, si è provveduto a progettare il diagramma UML delle classi utilizzate all'interno di *netdemic*, come rappresentato nella Figura 4.1. Per semplicità di descrizione, le classi di *netdemic* vengono suddivise in quattro categorie:

- classi "entità": in questa categoria rientrano le classi che si riferiscono a un oggetto fisicamente presente nella simulazione, come ad esempio un nodo o una regione;
- classi "di controllo": questa categoria racchiude le classi i cui script controllano la generazione e la visualizzazione della simulazione;

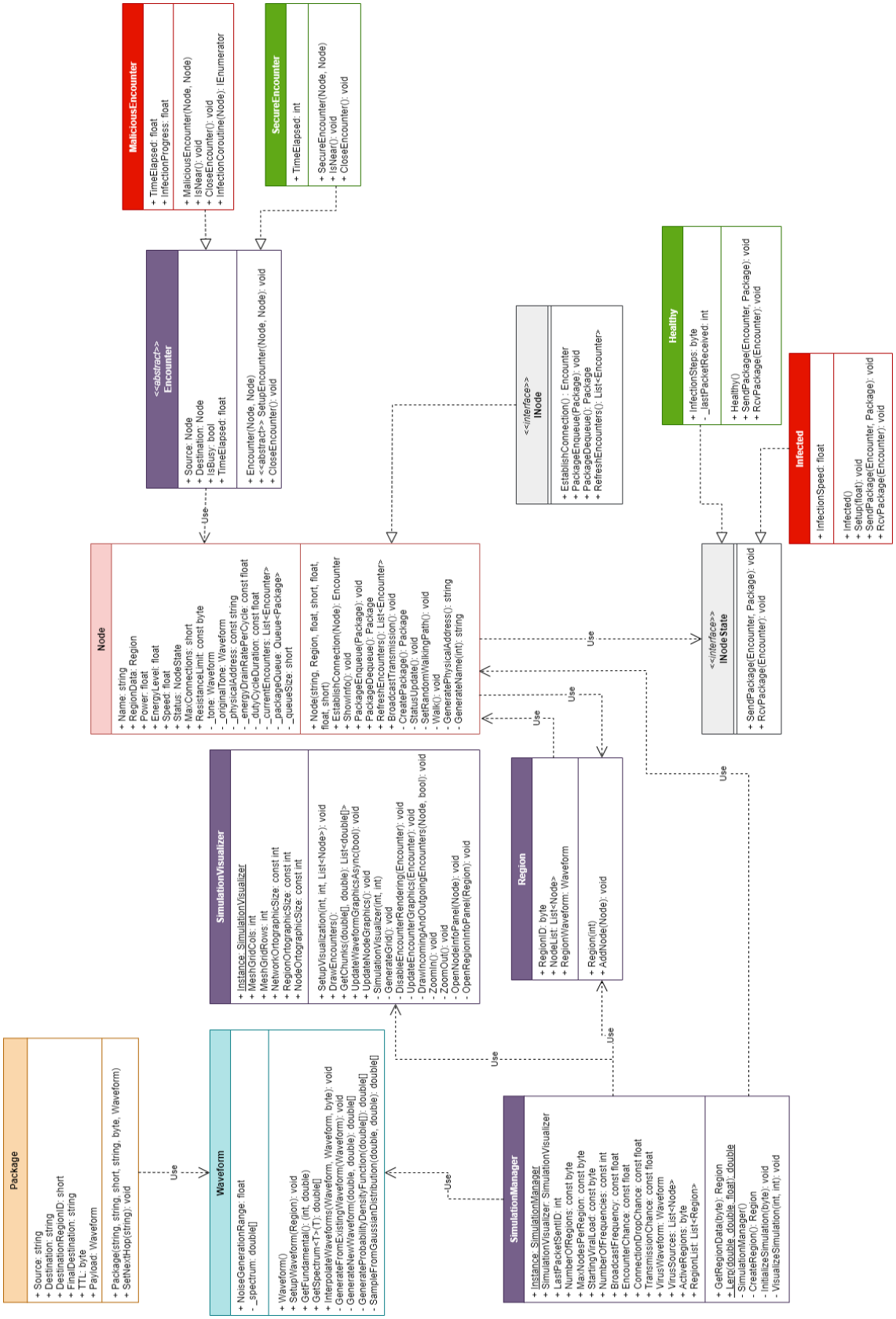

Figure 4.1: *Diagramma UML completo di netdemic*

- classi "di stato": sono quelle classi che descrivono il comportamento degli oggetti a cui vengono "attaccate" tramite l'editor di Unity, come ad esempio le classi che regolano il comportamento di nodi sani e infetti;
- classi "di sistema": gestiscono il menù principale e le procedure preliminari all'inizio della simulazione, come l'applicazione della configurazione iniziale impostata dall'utente nelle impostazioni di gioco.

In Unity, ogni classe che eredita dalla classe `MonoBehaviour` interna a Unity può essere attaccata come script ad un oggetto di gioco per far eseguire a quell'oggetto determinate funzioni. Questo script viene quindi definito *componente*. All'interno di *netdemic*, quasi tutte le classi sono componenti. Nelle sezioni successive, vengono elencate per categorie tutte le classi che fanno parte del codice di *netdemic*.

4.3 Classi entità

Le classi entità sono i componenti che permettono a oggetti di tipo `Node`, `Region`, `Encounter`, `Waveform` e `Package` di eseguire i propri compiti.

4.3.1 Region

Ogni regione viene visualizzata come un GameObject formato da uno sprite 2D bianco semitrasparente che esegue le proprie funzioni tramite il componente `Region`. La regione è un'entità fissa non soggetta a movimento né forzato né spontaneo e, il suo unico scopo è quello di contenere, nell'area circolare ristretta individuata dallo sprite 2D, tutti i nodi sani che ha precedentemente generato sulla base del proprio timbro.

4.3.2 Node

Ogni nodo è costituito da un GameObject visualizzato come uno sprite 2D bianco, al quale è attaccato il componente `Node`, implementazione dell'interfaccia `INode`. Esso determina le proprietà del singolo nodo a cui è attaccato: nome (una stringa di 8 caratteri alfanumerici case-sensitive generata randomicamente), regione di appartenenza, potenza, velocità, stato corrente (infetto o sano), numero massimo di

connessioni (per tutti i nodi uguale a tre), timbro caratteristico originale e corrente, grandezza della coda di storage.

Indirizzo fisico del nodo Esiste anche un campo che contiene l'indirizzo fisico del nodo: questa proprietà è una stringa di 32 caratteri alfanumerici (case-insensitive) risultante dal calcolo della funzione di hash crittografico `SHA-256` il cui input è la lista dei byte che costituiscono ogni singolo campione del timbro caratteristico. Dal momento che un singolo campione è espresso come dato `double` e quindi con 8 bytes, l'input dell'algoritmo è costituito da $22.000 * 8 = 176.000$ bytes $= 176$ kB. Questa proprietà è stata introdotta per implementare eventuali funzionalità di sicurezza in un futuro aggiornamento, in quanto la sua conservazione potrebbe essere utile per un DTN gateway che, mantenendo una dizionario di coppie che associano il nome del nodo al suo indirizzo fisico, diventerebbe una sorta di firewall: se un nodo volesse comunicare con un nodo di una regione diversa dalla propria, conoscerebbe soltanto il suo nome e non il suo indirizzo fisico, che sarebbe nascosto al pubblico per ragioni di sicurezza; pertanto, il pacchetto dovrebbe prima passare dal gateway della regione di destinazione che, una volta accertatosi dell'attendibilità del pacchetto e dopo aver risolto l'associazione hostname-indirizzo, come fosse un vero e proprio server DNS, inoltrerà a sua volta il pacchetto verso la giusta destinazione all'interno della propria regione di competenza.

Movimento del nodo In base allo stato del nodo, il suo comportamento differisce. Se il nodo è sano, esso si sposterà soltanto nei confini della propria regione, scegliendo un punto al suo interno verso cui muoversi e ripetendo questo processo non appena giunto a destinazione: ciò genera un movimento scorrevole del nodo a ogni fotogramma disegnato grazie alla funzione di Unity `MoveTowards(current, target, maxDistance)` che sposta un punto verso una destinazione lungo il tempo, effettuando uno spostamento massimo per ogni fotogramma. L'effetto che si ottiene è simile a quello del movimento di una persona che passeggia all'interno di un'area circoscritta. Se il nodo è infetto o lo diventa in seguito, allora non sarà più vincolato alla propria regione, avrà molta più libertà di movimento e la sua velocità triplicherà, come già discusso all'inizio della Sezione 3.3.2.

Funzionalità principali Dopo la creazione, ogni nodo avvia una `Coroutine` che esegue ogni F_B secondi la procedura `BroadcastTransmission()`, la quale invia un pacchetto in broadcast a tutti i collegamenti attivi; inoltre, per ogni fotogramma la funzione `RefreshEncounters(`*reachable_nodes*`)` cattura tutti gli oggetti `Node` all'interno del suo raggio di comunicazione (la cui lunghezza è determinata dalla proprietà *Power*) e controlla se, nella lista degli encounters correnti, ci sono encounters da disattivare, da riattivare o da generare, per poi visualizzarli. Il concetto di disattivazione e generazione di un encounter verrà chiarito meglio nel paragrafo in cui si discute delle entità di tipo `Encounter`.

4.3.3 Waveform

La classe `Waveform` contiene gli script necessari a costruire, da zero o sulla base di un timbro esistente, un nuovo timbro da assegnare a una nuova regione o un nuovo nodo. Non costituisce una vera e propria entità, in quanto un oggetto di questo tipo non può esistere da solo, ma è fondamentale per comporre le entità già viste. Come già detto precedentemente, sui campioni del timbro si basa il calcolo dell'indirizzo fisico del nodo a cui il timbro è assegnato.

Costruzione di un nuovo timbro La costruzione del nuovo timbro inizia con l'esecuzione della funzione `SetupWaveform(`*region*`)` che funge da costruttore per il nuovo oggetto. La funzione stabilisce in che modo creare il nuovo timbro: se riceve in input un oggetto `Region`, allora deve creare un nuovo timbro sulla base di un timbro regionale, applicando quindi un rumore casuale e uguale per tutti i campioni del timbro di partenza, altrimenti lo crea da zero. Ci sono due diverse possibilità di creare da zero un nuovo timbro: se il timbro in questione è destinato ad essere assegnato ai nodi virus iniziali, allora la distribuzione gaussiana sulla quale verranno campionati i valori avrà una media e una varianza casuali, altrimenti se deve essere assegnato a una regione il valore di media verrà scelto in modo tale da rendere equidistanti tutte le medie di tutte le distribuzioni che è necessario creare per generare le regioni della simulazione. In generale, la costruzione di un timbro da zero si divide nei seguenti passi:

1. si genera la distribuzione gaussiana di media μ e varianza σ^2;

2. si campionano 22000 valori usando la funzione della libreria *MathNet.Numerics* `Samples(array_destinazione, `μ`, `σ`)`;

3. con la funzione `GenerateProbabilityDensityFunction(`*samples*`)` si genera la funzione di densità di probabilità su `array_destinazione`, il quale conteneva i 22000 valori campionati, calcolando quindi la frequenza all'interno dell'array di ognuno dei valori campionati;

4. l'array restituito dalla precedente funzione diventerà lo spettro del nuovo timbro.

Interpolazione tra timbri La funzione `InterpolateWaveforms(`$t_1, t_2, factor$`)` viene richiamata nel momento in cui si verifica un contagio e si occupa non solo di interpolare linearmente i singoli campioni del timbro vittima in base al numero di pacchetti infetti ricevuti con la funzione `Lerp(`a, b, f`)`, ma anche di aggiornare la visualizzazione dell'onda nel pannello delle informazioni del nodo (il cui utilizzo si approfondirà in seguito).

4.3.4 SecureEncounter e MaliciousEncounter

Gli encounter vengono rappresentati all'interno della simulazione come degli oggetti che contengono un'istanza delle classi `SecureEncounter` o `MaliciousEncounter`. Ogni oggetto di questo tipo racchiude in sé determinate informazioni: il riferimento ai nodi sorgente e destinazione e lo stato della trasmissione, cioè se quel collegamento sta trasferendo dati, ma anche un riferimento a un oggetto `LineRenderer`: esso è un componente interno di Unity che serve allo script di visualizzazione per disegnare, quando necessario, una linea (o corda) colorata i cui estremi sono collegati alle posizioni dei nodi sorgente e destinazione. La corda è più stretta in corrispondenza dell'estremo appeso alla sorgente e più larga all'estremo opposto e cambia colore in base al tipo di encounter: se l'oggetto in esame è un `SecureEncounter` allora la linea sarà color ciano per poi sfumare sul magenta e diventerà completamente verde quando su quel canale di comunicazione viene trasmesso un pacchetto, se invece è un `MaliciousEncounter` allora la sfumatura sarà dall'arancio al giallo e durante la trasmissione del pacchetto diventerà temporaneamente tutta rossa.

Ottimizzazione delle prestazioni: object pooling In base alla configurazione iniziale della simulazione, in un singolo frame possono essere attivi anche più di mille encounters contemporaneamente ma, data la natura intermittente di questi collegamenti, molti di essi potrebbero cadere per poi essere ricostituiti un attimo dopo. Inizialmente, il processo di creazione di un encounter a schermo prevedeva di creare un nuovo `GameObject` a cui appendere uno script di tipo `SecureEncounter` o `MaliciousEncounter`, per poi configurarlo: questa procedura è molto costosa in termini di allocazione per il motore di gioco ed era quella utilizzata fin dalle prime fasi di sviluppo. Ciò rallentava drasticamente le prestazioni del programma, scendendo a circa 5 FPS in una configurazione massima con 64 regioni con circa 16 nodi ciascuna, in quanto Unity doveva allocare e deallocare troppi oggetti e troppe volte al secondo.

La soluzione a questo problema è usare il metodo dell'*object pooling*: in generale, in presenza di un grande numero di oggetti identici o molto simili tra di loro che necessitano di essere creati, distrutti e ricreati in massa e in tempi brevissimi, questa tecnica si rivela molto utile in quanto non distrugge gli oggetti ma li disattiva, mantenendoli in memoria per un eventuale uso futuro. Nel caso in esame, questo metodo viene applicato nella seguente maniera: ogni volta che il nodo x si collega a una destinazione con la quale non ha mai comunicato, viene creato un nuovo oggetto encounter e viene inserito come figlio del nodo x, cosicché x possa tenere traccia di tutte le destinazioni con cui ha stabilito un contatto almeno una volta. Quando uno degli encounter attivi cade, l'oggetto di riferimento non viene distrutto ma disattivato, così da non costringere il motore di gioco ad allocare lo stesso oggetto potenzialmente migliaia di volte. Seguendo questo procedimento, il nodo x potrà alla fine avere un massimo di $n - 1$ figli, dove n è il numero totale di nodi generati (compresi quelli inizialmente virali). In questo modo vengono diminuite le allocazioni ma soprattutto vengono completamente annullate le deallocazioni, le quali vengono solitamente svolte dal *garbage collector* di Unity una volta che gli oggetti vengono distrutti. Dal momento che il garbage collector effettua un lavoro generale su tutta la memoria e viene attivato molto spesso, se fosse costretto a fermarsi molte volte per cancellare ogni traccia in memoria degli oggetti già distrutti, il software fermerebbe frequentemente e bruscamente l'esecuzione per permettergli di fare il suo lavoro.

4.3.5 Package

Le istanze della classe `Package` rappresentano i singoli pacchetti che vengono scambiati tra due nodi connessi da un encounter. Un singolo pacchetto contiene una serie di metadati come il numero identificativo del pacchetto, che viene posto uguale a un numero progressivo che viene incrementato ogni volta che ne viene creato uno nuovo all'interno della simulazione, sorgente e destinazione finale, prossimo salto e il suo TTL[1], nonché il payload, composto dal timbro da inserire nel pacchetto, sia che esso sia inviato da un nodo sano sia che il mittente sia malevolo. Un pacchetto viene creato esclusivamente dai nodi per trasmetterlo attraverso un loro encounter attivo ed è un oggetto atomico e immodificabile una volta creato, eccetto per un solo campo: la destinazione del prossimo hop, diversa dalla destinazione finale che è invece il nodo a cui il pacchetto è effettivamente destinato, cioè l'ultimo destinatario del trasferimento di custodia supponendo che il pacchetto arrivi correttamente alla destinazione designata in origine dal mittente.

4.4 Classi di controllo

Le classi di controllo gestiscono la generazione delle entità, l'interazione tra esse e la loro visualizzazione. Le uniche due classi di controllo `SimulationManager` e `SimulationVisualizer` costituiscono un'applicazione del **pattern architetturale MVC** (*Model-View-Controller*), in quanto permettono di separare la generazione dei dati della simulazione, operata da `SimulationManager`, dalla loro visualizzazione che invece è a carico di `SimulationVisualizer`, costituendo rispettivamente il componente *model* e il componente *view* del pattern. `SimulationVisualizer` rappresenta anche il componente *controller* del pattern MVC: grazie all'utilizzo del nuovo sistema di input di Unity (*Input Manager*) e delle `UnityAction`, lo script permette di catturare il click sinistro e destro dell'utente, utili a fare lo zoom in/out su una determinata entità per visualizzarne le proprietà e le statistiche a schermo. Entrambe queste classi sono dei *singleton*: ciò implica che sia presente una e una sola istanza

[1] Il TTL (*Time To Live*) è il numero massimo di salti che il pacchetto può fare prima di essere buttato, cioè il numero massimo di nodi della rete che può attraversare prima di non essere più ritrasmesso.

di ognuno di questi script all'interno della simulazione, in quanto non avrebbe senso avere più di un gestore o più di un visualizzatore.

4.4.1 SimulationManager

L'oggetto di questa classe singleton viene istanziato poco prima della generazione della simulazione. Gli script di `SimulationManager` permettono al `GameObject` al quale lo script è attaccato di creare le regioni e i nodi virali iniziali, nella misura stabilita dai parametri immessi dall'utente nelle impostazioni nel menù principale, per poi delegare la creazione dei nodi alle regioni stesse. `SimulationManager` viene chiamato in causa solo nella fase di generazione della simulazione e il suo lavoro si esaurisce dopo l'esecuzione di due funzioni:

- `InitializeSimulation()`: inizializza la simulazione creando le n regioni e i ν nodi virali iniziali, conservandone i singoli riferimenti in due liste adibite a raccoglierli;
- `VisualizeSimulation(`$rows, cols$`)`: trasferisce il compito di visualizzare tutto ciò che è stato generato al componente visualizzatore, richiamando la sua funzione `SetupVisualization(`$rows, cols, virusList$`)`.

L'oggetto `SimulationManager` è fondamentale in quanto incapsula dati fondamentali all'esecuzione della simulazione: tutti i parametri impostati dall'utente nella configurazione iniziale, il numero dell'ultimo pacchetto inviato nella rete, la lista delle regioni attive e dei nodi sorgente dell'infezione, ma anche alcune `LayerMask` relative alle regioni, ai nodi e ai virus, utili a distinguere all'interno di Unity l'entità su cui l'utente ha cliccato. In breve, una `LayerMask` individua un sottoinsieme di `GameObject` che fanno parte dello stesso *layer*, come se gli oggetti dello stesso tipo fossero su un piano diverso dagli altri. L'uso di queste maschere verrà illustrato in seguito.

4.4.2 SimulationVisualizer

`SimulationVisualizer` è il componente *view* del pattern MVC in esame ed è la classe che si occupa di disegnare sullo schermo tutto quello che succede all'interno

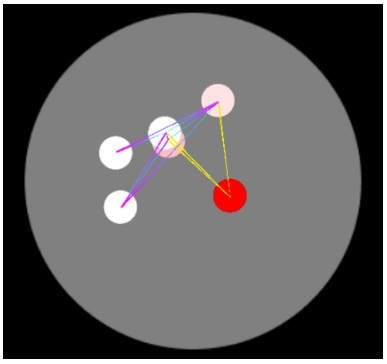

Figure 4.2: Visualizzazione dei `SecureEncounter` (in ciano-magenta) e dei `MaliciousEncounter` (in giallo) a cui i nodi di una regione partecipano

della simulazione: spostamento dei nodi, visualizzazione degli encounter e delle statistiche di ogni nodo e regione.

Visualizzazione della simulazione Inizialmente, il componente dispone graficamente le regioni in corrispondenza dei punti che formano la meshgrid, già menzionata nella Sezione 3.2.2, e visualizza il campo di simulazione per intero con una telecamera ortografica, disegnando soltanto le regioni e il movimento dei vari nodi. Facendo click sinistro su una regione, il campo visivo della telecamera si restringerà in modo tale da accogliere la regione selezionata al centro della visuale, mentre facendo click sinistro su un nodo, la telecamera si restringerà fino a visualizzare al centro della schermata il nodo e lo seguirà durante il suo cammino. In qualsiasi momento, si può tornare alla visualizzazione principale facendo click destro su qualsiasi punto dello schermo. Per rilevare se la destinazione del click sia una regione o un nodo, lo script lancia un raggio in profondità partendo dalla posizione del puntatore: se al momento del click il raggio collide con un'entità, viene controllato qual è la `LayerMask` che contiene quel tipo di oggetto e, in base al risultato, lo script farà uno zoom in della telecamera sul primo oggetto attraversato dal raggio (che, banalmente, rappresenta l'oggetto cliccato dall'utente), eseguendo poi delle operazioni secondarie a seconda dell'entità selezionata.

Ottimizzazione del rendering Se tutte le entità attive venissero visualizzate a schermo allo stesso tempo, bisognerebbe disegnare, nel caso peggiore, più di un migliaio di oggetti in movimento e ciò appesantirebbe tantissimo il carico di lavoro

del rendering thread di Unity, rallentando la visualizzazione dei singoli fotogrammi e facendo crollare quindi il framerate. Per evitare cali di prestazioni, in base a quale entità viene visualizzata tra l'intero campo di simulazione, una regione o un nodo, la funzione `DrawIncomingAndOutgoingEncounters(`*node*`)` limiterà il rendering degli encounters, le entità più dinamiche all'interno della simulazione. La funzione si comporta in modo diverso in base a cosa viene visualizzato nel momento in cui viene chiamata:

- se viene visualizzata l'intera simulazione, non viene disegnato a schermo nessun encounter;
- se viene visualizzata una regione, vengono disegnati solamente gli encounter ai quali partecipano i nodi di quella regione, escludendo tutti gli altri (è il caso mostrato nella Figura 4.2);
- se viene visualizzato un nodo qualsiasi, vengono visualizzati a schermo solo gli encounter che hanno come sorgente o destinazione proprio quel nodo.

Con questa procedura, nel peggiore dei casi vengono rappresentate un numero di linee di collegamento di molto inferiore al numero totale degli encounter attivi globalmente.

Gestione dell'interfaccia grafica Il componente `SimulationVisualizer` si occupa anche di gestire la visualizzazione della GUI di *netdemic*. L'interfaccia grafica, come si vede nella Figura 4.3, è divisa principalmente in due pannelli che si attivano e si aggiornano in base a quale entità è stata cliccata.

Il pannello superiore mostra informazioni e statistiche sulla regione cliccata o sulla regione del nodo cliccato: il nome, l'ID della regione e i nodi che contiene, con lo stato di salute di ciascuno di essi, nonché la visualizzazione tramite `LineRenderer` della forma d'onda archetipica del timbro della regione. Esso viene visualizzato sempre quando si clicca su un nodo che non è uno dei nodi virali di partenza, i quali non hanno una regione assegnata. Al momento del click sulla regione, questo pannello viene dapprima aggiornato in background e poi visualizzato con i valori riferiti a quella regione.

Il pannello inferiore fornisce una panoramica sulle proprietà di un nodo cliccato, come il nome, lo stato di salute e le sue proprietà di potenza, velocità e resistenza

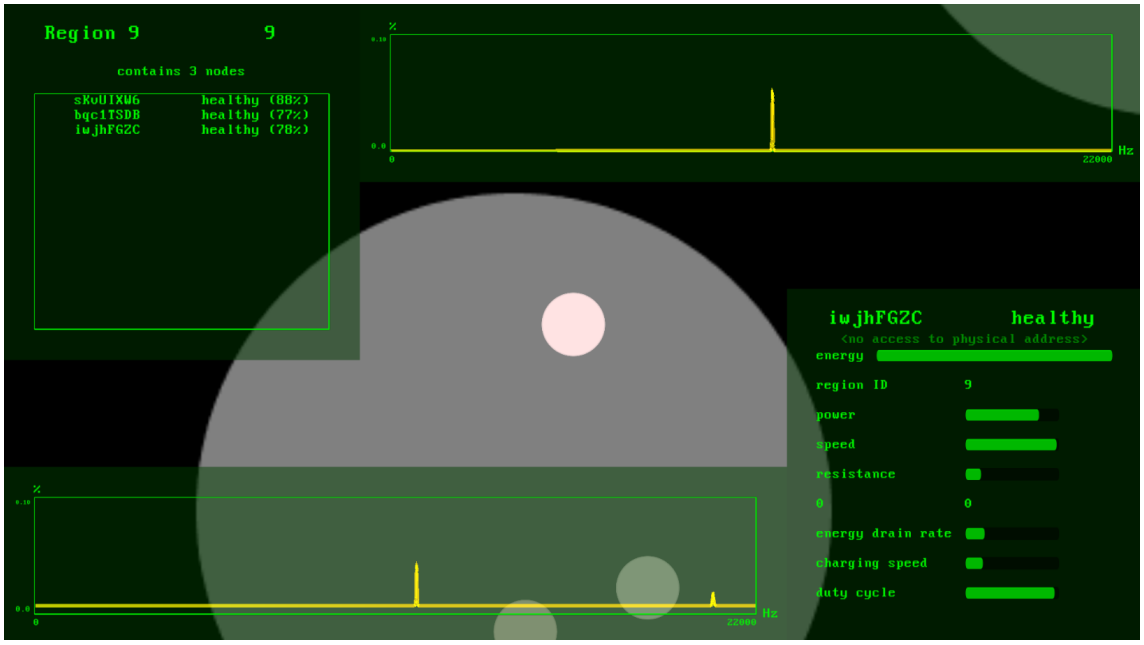

Figure 4.3: Interfaccia grafica di *netdemic* mentre si visualizza un nodo

all'infezione. Anche in questo caso, viene visualizzato il timbro del nodo tramite una linea all'interno dell'istogramma in basso a sinistra.

Visualizzazione dell'infezione in corso Nella Figura 4.3 si può notare come il timbro del nodo iwjhFGZC abbia due "punte": la punta più bassa viene generata per effetto del processo di interpolazione lineare tra il timbro sano e il timbro malevolo, mentre la punta più alta in realtà è la frequenza fondamentale del timbro originale la cui intensità, lentamente, andrà ad abbassarsi fino a diventare nulla, favorendo l'incremento dell'intensità della frequenza fondamentale del timbro virus. Come già spiegato precedentemente, a ogni passo infettivo il nodo cambierà colore, sfumando dal bianco puro di partenza (nodo sano al 100%) verso un rosso acceso (nodo completamente infetto), aggiornando di conseguenza anche la percentuale di salute rimanente all'interno della lista di nodi contenuta nel pannello superiore della regione.

Ottimizzazione dell'aggiornamento real-time dei timbri Un generico timbro, all'interno degli istogrammi presenti nella GUI, viene disegnato come un oggetto `LineRenderer` che disegna una linea con n punti di controllo. Se ad ogni elemento

dell'array dello spettro del timbro fosse associato un punto di controllo, questa linea avrebbe sempre tanti punti di controllo quanti sono i bin dell'istogramma, cioè 22000 punti. Tuttavia, disegnare e aggiornare 22000 punti di controllo è particolarmente oneroso a livello computazionale e dunque questo aggiornamento massiccio fa crollare pesantemente le performance del software.

Partendo dall'assunto che un qualsiasi timbro della simulazione è formato perlopiù da valori uguali al valore minimo globale o comunque vicini ad esso, per risolvere questo problema, *netdemic* usa un algoritmo che trova, preso l'array dello spettro delle frequenze del timbro in questione, le zone dove sono presenti dei picchi di intensità, cioè tutte quelle zone nelle quali i valori sono maggiori del valore minimo. Questo algoritmo viene eseguito dalla funzione `GetChunks(array)`: essa restituisce, a partire dall'array dato come parametro, una lista formata dai chunks[2] dell'array che rispettano la condizione descritta precedentemente.

L'algoritmo esegue il suo lavoro in modo molto semplice, in quanto non fa altro che scorrere un indice lungo tutto l'array e marca due "segnalini" in corrispondenza dell'inizio e della fine di un chunk, per poi estrarre l'intervallo di valori racchiuso tra i due segnalini e aggiungerlo alla lista. Il procedimento appena descritto evita che, nel disegnare la linea dello spettro a schermo, non vi siano più di 2 punti consecutivi che abbiano lo stesso valore di intensità, riducendo la quantità dei punti di controllo della linea a un numero poco più grande del numero di frequenze che non hanno il valore di intensità minimo, alleggerendo di molto il peso computazionale dovuto all'aggiornamento in tempo reale del grafico dello spettro delle frequenze.

4.5 Classi di stato

Gli script di stato configurano il comportamento del singolo nodo a cui vengono attaccate. In fase di progettazione, la gestione degli stati è stata realizzata applicando il **design pattern State**: in questo modo, i comportamenti previsti per ogni stato non vengono implementati dall'entità ma da classi separate. Di conseguenza, per

[2]Nell'algoritmo in esame, un chunk viene definito come una porzione di array di n elementi compresa tra due indici interi a e b ($a \leq b$), tale che gli elementi agli indici $a-1$ e $b+1$ sono uguali al valore minimo dell'array, mentre gli elementi con indice i compreso tra a e b, estremi inclusi, sono tutti maggiori del valore minimo.

cambiare lo stato di un nodo, si dovrà semplicemente modificare il valore della proprietà stato del nodo con un'istanza dello script corrispondente al comportamento che si vuole dare all'entità. Le due classi di stato utilizzate in *netdemic* sono `Healthy` e `Infected` e il loro utilizzo riguarda esclusivamente le entità di tipo `Node`. Esse sono implementazioni della classe astratta `NodeState`, che a sua volta implementa l'interfaccia `INodeState` che definisce le operazioni di base che un'implementazione deve fornire, cioè invio e ricezione di un pacchetto.

4.5.1 Healthy

Un nodo che ha uno stato di tipo `Healthy` è un nodo sano. Un nodo con stato `Healthy` comunica eseguendo una `Coroutine` di trasmissione che, con un certo ritardo (impostabile via codice, ma di default posto a zero millisecondi), invia un semplice pacchetto in broadcast. In fase di ricezione, invece, il nodo controlla la destinazione finale del pacchetto: se essa è uguale al nome del nodo ricevente, allora esso assorbe il pacchetto togliendolo dalla rete, altrimenti se ciò non è vero il pacchetto viene inserito nella coda di storage di quel nodo. In entrambi i casi, il nodo si segna l'ID di questo pacchetto come ultimo pacchetto ricevuto, per evitare di ricevere indietro pacchetti già trasmessi.

4.5.2 Infected

Lo stato `Infected` configura il nodo come un'entità virale. In fase di invio, un nodo con questo stato trasmetterà il pacchetto regolarmente come se fosse un nodo sano, ma controllerà anche se la destinazione vittima sia un nodo sano: in caso affermativo eseguirà la routine di infezione tramite interpolazione, altrimenti il nodo destinazione sarà infetto e pertanto butterà il pacchetto ricevuto. Alla ricezione di un pacchetto, esso verrà tolto dalla rete e buttato via, bloccando di fatto la comunicazione di rete attraverso quel nodo, creando una sorta di "buco nero" per i pacchetti che lo attraversano.

4.6 Classi di sistema

Le classi di sistema sono degli script ausiliari che aiutano nella costruzione della simulazione e nella gestione del menù principale del software. Tra gli script di *netdemic* ci sono due classi di sistema: `MainMenu` e `SettingsMenu`.

4.6.1 MainMenu

Lo script `MainMenu` gestisce interamente il menù principale, effettuando lo switch tra il pannello principale e il pannello delle impostazioni. All'avvio del software, il menù di *netdemic* è composto da tre pulsanti principali: il tasto *quit* termina il programma ritornando al desktop, *settings* apre il pannello delle impostazioni dove l'utente può configurare i parametri della simulazione muovendo gli slider dei valori che intende modificare, e il tasto `start` avvia la procedura di generazione caricando una nuova scena (quella della simulazione in sé) e trasferendo i parametri salvati nelle impostazioni alla nuova scena.

4.6.2 SettingsMenu

Questo script gestisce le modifiche ai parametri iniziali della simulazione effettuate nel pannello delle impostazioni, nonché il loro salvataggio interno. Se l'utente non modifica in nessun modo questi parametri all'inizio dell'esecuzione del programma, la simulazione sarà avviata con dei parametri di default impostati nel costruttore di questa classe. Per applicare le modifiche, lo script comunica con `SimulationManager`, la cui prima operazione è proprio quella di copiare i valori dei parametri conservati da `SettingsMenu` dentro le proprie variabili interne per mantenere i valori visibili anche durante l'esecuzione della simulazione.

4.7 Funzionalità escluse dal prodotto finale

All'interno del codice di *netdemic* sono presenti alcune classi inutilizzate, ma che sono state scritte in previsione di una futura implementazione di funzionalità aggiuntive. Le funzionalità escluse dal prodotto finale riguardano principalmente il controllo dell'energia di un nodo, del suo duty cycle e la capacità di un nodo di fare da gateway per una regione.

Un nodo all'interno di una DTN potrebbe non poter rimanere sempre attivo e potrebbe non poter comunicare molto spesso. A tal proposito, si potrebbe imporre un limite temporale di lavoro al nodo, tale che esso possa elaborare o trasmettere pacchetti solamente in un intervallo di tempo ristretto per ogni secondo e, dopo un determinato numero di elaborazioni o dopo un certo periodo di tempo, il nodo potrebbe spegnersi per poi riaccendersi quando la propria batteria di autoalimentazione viene ricaricata. Uno scenario di applicazione di questo concetto è stato già visto nella Sezione 1.1. La classe di stato `Sleeping`, inutilizzata nel prodotto finale, indicherebbe al nodo come comportarsi in una situazione di shutdown dovuto all'esaurimento dell'energia o al semplice arresto delle operazioni fuori dal proprio dutycycle. La classe di stato inutilizzata `Gateway`, invece, potrebbe essere utile per implementare delle funzionalità di sicurezza legate ai DTN gateway, come già discusso nella Sezione 4.3.2 di questo capitolo.

Chapter 5

Conclusioni e risultati raggiunti

In questo documento si è analizzata un'architettura di rete fondata su principi diversi rispetto a quelli alla base delle reti Internet convenzionali, basate sullo stack TCP/IP. Questa architettura di rete non convenzionale riesce a ovviare a diversi problemi che in una normale rete Internet non esistono, quali l'instabilità dei collegamenti o le scarsissime capacità energetiche e computazionali del singolo dispositivo connesso alla rete, che ne limitano il tempo di attività peggiorando globalmente l'efficienza del trasferimento di dati all'interno della rete.

L'analisi teorica dell'architettura di rete discussa nel primo capitolo di questo documento è stata supportata da uno studio delle possibili applicazioni pratiche di questo modello di rete e da una panoramica sugli aspetti teorici più importanti da considerare in una rete di dispositivi nella quale viene iniettato un fattore virale, riconducendo il tutto ideologicamente ad alcuni scenari reali nei quali viene studiata l'inferenza di particelle virali in un sistema di particelle nel mondo reale.

Il risultato della costruzione della simulazione di delay-tolerant network proposta ricorda infatti un sistema di particelle reagenti tra loro poste all'interno di un sistema chiuso, che vengono alterate da un agente esterno convivente nello stesso spazio su cui il sistema è distribuito. L'esecuzione della simulazione realizzata dimostra come, in assenza di misure di sicurezza di qualsiasi tipo, in base alla dimensione della rete e alla pericolosità dell'agente esterno (misurata tramite il numero di particelle dalle quali l'agente è composto e dalla pericolosità di ogni particella virale), è molto semplice bloccare la comunicazione all'interno di una rete delay-tolerant, contaminandola con un elemento infettivo.

Table 5.1: Classificazione dei parametri di configurazione per il calcolo delle metriche di valutazione dei risultati

Livello	Dimensione della rete		Stabilità dei collegamenti	Attività di trasmissione			Viralità
	n	$\hat{n}$	$P[D]$	F_B	$P[E]$	$P[T]$	ν
Basso	16	4	0.15	1500ms	0.01	0.05	8
Medio	32	8	0.06	1000ms	0.05	0.25	32
Alto	64	16	0.01	500ms	0.10	0.50	64

5.1 Metriche di valutazione delle prestazioni

Per mostrare i risultati interessanti che la simulazione produce, vengono calcolate ad ogni istanza della simulazione alcune metriche di valutazione dei risultati che si basano su alcune statistiche raccolte dal programma stesso: il numero di nodi generati dalla simulazione, il tempo impiegato dal virus a contagiare l'intera rete, il numero totale di encounters stabiliti e il numero totale di pacchetti che hanno transitato nella rete dall'inizio alla fine del processo di contagio. Le metriche di valutazione considerate, calcolate sulla base delle statistiche ottenute da ciascuna iterazione della simulazione, sono le seguenti:

- il tempo totale di contagio;
- il numero medio di pacchetti trasmessi al secondo (pkt/s): si ottiene dividendo il traffico di rete totale per il tempo totale di contagio e misura la capacità della rete di far circolare dati. Si noti bene che questa metrica non misura l'ottimalità del routing, dal momento che non è una metrica di efficienza, ma indica solo quanto traffico la rete riesce a generare in un'unità di tempo;
- il numero medio di pacchetti per encounter (pkt/E): è il risultato del rapporto tra il traffico totale e il numero di collegamenti stabiliti e misura l'efficienza dei collegamenti tra i vari nodi. Può essere utilizzata per avere un'idea generale su quanti collegamenti infetti in media servono per infettare un nodo;
- il numero medio di pacchetti trasmessi per nodo (pkt/nodo): è il rapporto tra il numero di pacchetti inviati in totale e il numero totale di nodi generati all'inizio della simulazione, nodi infetti inclusi. Indica la capacità di trasmissione media di un qualsiasi nodo.

Table 5.2: Tabella di riferimento delle configurazioni di test per il calcolo delle metriche di valutazione dei risultati

Configurazione	Dimensione della rete	Stabilità dei collegamenti	Attività di trasmissione	Viralità
I	Media	Media	Media	Alta
II	Media	Alta	Alta	Alta
III	Media	Alta	Media	Alta
IV	Media	Media	Alta	Alta
V	Alta	Media	Media	Media
VI	Alta	Alta	Alta	Media
VII	Alta	Alta	Media	Media
VIII	Alta	Media	Alta	Media
IX	Media	Bassa	Bassa	Alta
X	Alta	Bassa	Bassa	Media

5.2 Benchmarking delle prestazioni di rete

Per effettuare i benchmark, sono state elaborate dieci configurazioni diverse dei sette parametri decidibili dall'utente nelle impostazioni di gioco, le quali sono riassunte nella tabella 5.2. Per semplicità, per ogni parametro vengono usati tre valori fissati, che restringe la scelta del parametro a tre "livelli": uno basso, uno medio e uno alto (vedi tabella 5.1). Le dieci configurazioni sono quindi un insieme ristretto di combinazioni dei livelli di ognuno dei sette parametri. Successivamente, sono state effettuate tre diverse iterazioni della simulazione per ognuna delle dieci configurazioni, per un totale di trenta osservazioni. Nella tabella 5.3, i valori riportati per ogni configurazione sono la media delle tre osservazioni effettuate per ciascuna di esse.

Si nota dai dati osservati che, in presenza di reti molto ampie e di un fattore virale preponderante, l'infezione si diffonde rapidamente, proprio come lo sarebbe un reale agente patogeno che si diffonde all'interno di una comunità di esseri umani caratterizzata da una forte densità abitativa, ma soltanto a patto che i nodi cerchino di comunicare tra di loro con frequenza. Facendo riferimento alla tabella 5.3, nelle configurazioni *IX* e *X* l'infezione fatica a prendere piede, a causa della bassa stabilità dei collegamenti e della poca interazione reciproca tra i nodi, impiegando più di due ore per completare il contagio dell'intera rete. La configurazione *V* rappresenta

Table 5.3: Tabella dei risultati medi delle metriche di valutazione delle prestazioni di rete per ogni configurazione di test (le statistiche raccolte sono arrotondate all'intero successivo)

Configurazione	Nodi generati (sani + infetti)	Encounter totali	Traffico totale (in pacchetti)	Tempo di contagio	pkt/s	pkt/E	pkt/nodo
I	135 + 64	45635	4620	4m 52s	15.82	0.10	23.21
II	139 + 64	6040	7925	1m 41s	78.47	1.31	39.04
III	155 + 64	17175	6257	2m 49s	25.15	0.36	28.57
IV	145 + 64	27248	9322	1m 52s	83.21	0.34	44.60
V	521 + 32	385325	32289	3m 43s	144.71	0.08	58.39
VI	596 + 32	36278	40748	54s	755.29	1.12	64.88
VII	576 + 32	106247	33706	2m 30s	224.74	0.32	55.43
VIII	565 + 32	169026	49533	1m 22s	601.57	0.29	82.97
IX	150 + 64	840912	5003	2h 19m 34s	0.60	0.006	23.38
X	518 + 32	9341825	44348	2h 28m 40s	4.97	0.005	80.63

un ulteriore caso meritevole di approfondimento. In questa configurazione, i nodi comunicano fra di loro non troppo di frequente e la stabilità dei collegamenti è altalenante: si crea dunque uno scenario nel quale i nodi instaurano connessioni con frequenza più o meno regolare ma esse sono poco stabili, quindi i nodi sono costretti a ricollegarsi più e più volte in un intervallo temporale ristretto per poter comunicare e ciò fa aumentare di molto il numero di encounter stabiliti, causando un aumento molto lento del numero di pacchetti circolanti. Dal benchmarking delle altre configurazioni si desume invece che se i collegamenti sono più o meno stabili e la trasmissione è frequente, la rete viene completamente invasa dal fattore virale nel giro di pochi minuti.

Dal momento che una rete di dispositivi, in base allo scenario considerato, può raggiungere dimensioni anche di gran lunga più estese rispetto a quanto considerato all'interno della simulazione, ciò sottolinea l'importanza della ricerca di approcci che possano mettere in sicurezza una trasmissione a lunga distanza, come quelle che vengono instaurate tra i nodi della simulazione, in un contesto non virtuale. Nel mondo reale, infatti, la rete potrebbe essere dislocata e distribuita all'interno di aree immensamente grandi o ad alto rischio di intromissione da parte di soggetti terzi che, in base alla sensibilità e alla riservatezza dei dati che circolano nella rete, possono causare danni, disservizi o violazioni della privacy. In un eventuale scenario futuro al limite della fantascienza, nel quale l'essere umano riesca a creare delle reti interplanetarie dopo aver colonizzato altri corpi celesti vicini, interruzioni molto prolungate di connessione di rete a distanza siderale, soprattutto in ambienti militari e di servizi di prima necessità, potrebbero avere un gravissimo impatto sulla popolazione terrestre

ma soprattutto su quella extraplanetaria, in base a quanto importanti e sensibili siano i dati interscambiati tra i pianeti. Il codice della simulazione, nonché il progetto Unity di *netdemic* con diagramma UML incluso, è completamente consultabile all'indirizzo di seguito, che rimanda alla repository del progetto hostata su *GitHub*.
https://github.com/w8floosh/netdemic

Bibliography

[1] K. Fall. “A Delay-Tolerant Network Architecture for Challenged Internets”. In: *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM ’03. Karlsruhe, Germany: Association for Computing Machinery, 2003, 27–34. ISBN: 1581137354. DOI: 10.1145/863955.863960. URL: https://doi.org/10.1145/863955.863960.

[2] V. Cerf, S. Burleigh, A. Hooke, J. Torgerson, R. Durst, K. Scott, E. Travis, and H. Weiss. “Interplanetary internet (IPN): architectural definition”. In: (Jan. 2001).

[3] Z. Lu and J. Fan. “Delay/Disruption Tolerant Network and its application in military communications”. In: *2010 International Conference On Computer Design and Applications*. Vol. 5. 2010, pp. V5–231–V5–234. DOI: 10.1109/ICCDA.2010.5541302.

[4] M. Małowidzki, T. Dalecki, P. Bereziński, M. Mazur, and P. Skarżyński. “Adapting standard tactical applications for a military disruption-tolerant network”. In: *2016 International Conference on Military Communications and Information Systems (ICMCIS)*. 2016, pp. 1–5. DOI: 10.1109/ICMCIS.2016.7496564.

[5] M. Densmore and K. Fall. “Poster: Delay Tolerant Networking for Sensor Networks”. In: (Jan. 2004).

[6] Z. Haas and T. Small. “A new networking model for biological applications of ad hoc sensor networks”. In: *IEEE/ACM Transactions on Networking* 14.1 (2006), pp. 27–40. DOI: 10.1109/TNET.2005.863461.

[7] P. Juang, H. Oki, Y. Wang, M. tonosi, L.-S. Peh, and D. Rubenstein. “Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet”. In: vol. 37. Jan. 2002. DOI: 10.13140/RG.2.1.2314.6326.

[8] J. Griner, J. Border, M. Kojo, Z. D. Shelby, and G. Montenegro. *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations*. RFC 3135. June 2001. DOI: 10.17487/RFC3135. URL: https://www.rfc-editor.org/info/rfc3135.

[9] D. Feldmeier, A. McAuley, J. Smith, D. Bakin, W. Marcus, and T. Raleigh. "Protocol boosters". In: *Selected Areas in Communications, IEEE Journal on* 16 (May 1998), pp. 437 –444. DOI: 10.1109/49.669053.

[10] M. Seligman. "Storage Usage of Custody Transfer in Delay Tolerant Networks with Intermittent Connectivity". In: *Proceedings of the 2006 International Conference on Wireless Networks, ICWN 2006*. Las Vegas, Nevada, USA, Jan. 2006, pp. 386–392.

[11] A. Vahdat and D. Becker. "Epidemic Routing for Partially-Connected Ad Hoc Networks". In: *Technical Report* (June 2000).

[12] P. Mundur, M. Seligman, and G. Lee. "Epidemic routing with immunity in Delay Tolerant Networks". In: *MILCOM 2008 - 2008 IEEE Military Communications Conference*. 2008, pp. 1–7. DOI: 10.1109/MILCOM.2008.4753334.

[13] P. Garg, H. Kumar, R. Johari, P. Gupta, and R. Bhatia. "Enhanced Epidemic Routing Protocol in Delay Tolerant Networks". In: *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*. 2018, pp. 396–401. DOI: 10.1109/SPIN.2018.8474132.

[14] F. De Rango, S. Amelio, and P. Fazio. "Epidemic Strategies in Delay Tolerant Networks from an Energetic Point of View: Main Issues and Performance Evaluation". In: *Journal of Networks* 10 (Feb. 2015). DOI: 10.4304/jnw.10.01.4-14.

[15] J. M. Pujol, A. Lopez Toledo, and P. Rodriguez. "Fair Routing in Delay Tolerant Networks". In: *IEEE INFOCOM 2009*. 2009, pp. 837–845. DOI: 10.1109/INFCOM.2009.5061993.

[16] A. Lindgren, A. Doria, and O. Schelén. "Probabilistic Routing in Intermittently Connected Networks". In: *Service Assurance with Partial and Intermittent Resources*. Ed. by P. Dini, P. Lorenz, and J. N. de Souza. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 239–254. ISBN: 978-3-540-27767-5.

[17] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. "Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks". In: *Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking*. WDTN '05. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 2005, 252–259. ISBN: 1595930264. DOI: 10.1145/1080139.1080143. URL: https://doi.org/10.1145/1080139.1080143.

[18] Q. Jiang, K. Deng, L. Zhang, and C. Liu. "A Privacy-Preserving Protocol for Utility-Based Routing in DTNs". In: *Information* 10.4 (2019). ISSN: 2078-2489. DOI: 10.3390/info10040128. URL: https://www.mdpi.com/2078-2489/10/4/128.

[19] D. Meli, F. L. M. Milotta, C. Santoro, F. F. Santoro, and S. Riccobene. "Privacy Preserving on Delay-Tolerant Networks". In: *International Conference on Innovative Computing and Communications*. Ed. by D. Gupta, A. Khanna, S. Bhattacharyya, A. E. Hassanien, S. Anand, and A. Jaiswal. Singapore: Springer Nature Singapore, 2023, pp. 163–171. ISBN: 978-981-19-2535-1.

[20] X. Lu, P. Hui, D. Towsley, J. Pu, and Z. Xiong. "Anti-localization anonymous routing for Delay Tolerant Network". In: *Computer Networks* 54.11 (2010), pp. 1899–1910. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2010.03.002. URL: https://www.sciencedirect.com/science/article/pii/S1389128610000721.

[21] Math.NET Initiative. *Math.NET Numerics*. URL: https://numerics.mathdotnet.com/.

[22] Microsoft. *Task Parallel Library (TPL)*. URL: https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl.