Universitatea “Politehnica” din Bucureşti Facultatea de Electronică,
Telecomunicaţii şi Tehnologia Informaţiei

Asigurarea calității transferului în rețele wireless de tip Ad-Hoc, în condiții de congestie

Proiect de diplomă

prezentat ca cerinţă parţială pentru obţinerea titlului de
Inginer în domeniul *Calculatoare şi Tehnologia Informaţiei*
programul de studii de licenţă *Ingineria Informaţiei*

Conducător ştiinţific

Conf. dr. ing. Ştefan STĂNCESCU

Absolvent

Mihaela-Elena MILCA

2014-2015

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul Electronică Aplicată şi Ingineria Informaţiei

Aprobat Director de Departament :
Prof. Dr. Ing. Sever Paşca

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Milca F. Mihaela Elena,441A

1. Titlul temei: Asigurarea calităţii transferului în reţele wireless de tip Ad-Hoc, în condiţii de congestie

2. Contribuţia practică, originală a studentului va consta în (*în afara* părţii de documentare): analiza comparativă cu un simulator Network Simulator (NS) şi scripturi Tcl a unor algoritmi de dirijare pentru reţele Ad-Hoc, în scenarii cu granularitate şi dinamică diferite, punându-se în evidenţă influenţa asupra calităţii transferului (QoS) în aceste variante; modificarea unor algoritmi de dirijare cunoscuţi în vederea adaptării la aceste scenarii şi justificarea teoretică a acestor modificări; implementarea modificărilor prin modificarea si recompilarea sursei NS; justificarea performanţelor algoritmilor modificaţi prin simularea scenariilor cu NS astfel recompilat, reprezentarea grafică a rezultatelor şi formularea unor concluzii.

3. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline: Arhitecturi de Reţea şi Internet, Sisteme de Operare, Reţele de Calculatoare

4. Proprietatea intelectuală asupra proiectului aparţine: UPB/ Facultatea de Electronică, Telecomunicaţii ţi Tehnologia Informaţiei

5. Locul de desfăşurare a activităţii: UPB/ Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

6. Realizarea practică rămâne în proprietatea: UPB/ Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

7. Data eliberării temei: 12.12.2014

CONDUCĂTOR LUCRARE: Conf.Dr.Ing Ştefan Stăncescu

STUDENT: Mihaela-Elena Milca

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul " **Asigurarea calității transferului în rețele wireless de tip Ad-Hoc, în condiții decongestie",** prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de Inginer în domeniul Calculatoare și Tehnologia Informației, programul de studii Ingineria Informației (CTI – INF) este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 25.06.2015

Absolvent Mihaela-Elena Milca

Milca

(semnătura în original)

Cuprins:

Listă figuri

Listă acronime

AP-Access Point

BS-Base Station

RN-Relay Nodes

MANET –mobile ad-hoc network

TCP-Transmision Control Protocol

IP-Internet Protocol

MAC-Media Access Control

LLC-Logical Link Control

FER-Frame Error Ratio

DSDV-Destination-Sequenced Distance-Vector

DSR-Dynamic Source Routing

RREQ-Route REQuest

RREP-Route REPly

RERR-Route ERRor

FIFO-Firt In First Out

ACK- acknowledgement

RTO-retransmision timeout

TCP-F- Transmision Control Protocol-Feedback

RFN- Route Failure Notification

RRN-Route Re-establishment Notification

ELFN- Explicit Link Failure Notification

TCP-BuS-TCP Buffering capability and Sequence information

ECN-explicit congestion notification

BEAD-Best-Effort ACK Delivery

TCP-DOOR-TCp Detection of Out-of-Ordere and Response

TPA-Transport protocol for Ad-hoc Networks

EDAODV- Early Detection Congestion and Control Routing Protocol

CSP- Congestion Satus Packet

AODV-I- Improved Ad-hoc On-demand Distance Vector Routing Protocol
CRP- Congestion Adaptive Routing Protocol

Introducere

Proiectul are ca scop analiza comparativă a unor scenarii diferite în ceea ce privește modul de funcționare al protocolului AODV în condiții de congestie. Rezultatele experimentale obținute vor fi prezentate sub formă de grafice. Structura rețelei imaginate va fi ilistrată cu ajutorul interfeței grafice iar algoritmul de lucru al protocolului va fi explicat prin intermediul unei diagrame.

Principalul motiv pentru care am ales această temă este faptul că rețelele wireless sunt din ce în ce mai populare pentru că oferă posibilitatea de transfer de informații oriunde și oricând. Daca rețelele wireless baze pe infrastructură fixă sunt comune oricărei persoane, rețelele de tip Ad-Hoc nu sunt foarte cunoscute, fiind într-o continuă dezvoltare.

Rețelele de tip Ad-Hoc asigură comunicarea pe câmpurile de luptă, în zonele afectate de dezastre, acolo unde nu exista o infrastructură fixă.Ele se bazează pe faptul că nodurile dintr-o rețea au o mobilitate crescută iar configurarea nodurilor este facuta de ele însăși.

Lucrarea de față este divizată în patru capitole. În primele două capitole voi discuta despre rețelele Ad-hoc dar și despre protocoalele de rutare. În cel de-al treilea capitol voi pune în discuție problema congestiei, cum apare și cum se poate combate. Iar în ultimul capitol voi prezenta partea practică a lucrării: tehnologiile ce m-au ajutat în realizarea simulărilor, rezultatele obținute și explicarea lor.

Capitol 1-Reţele Ad-HOC

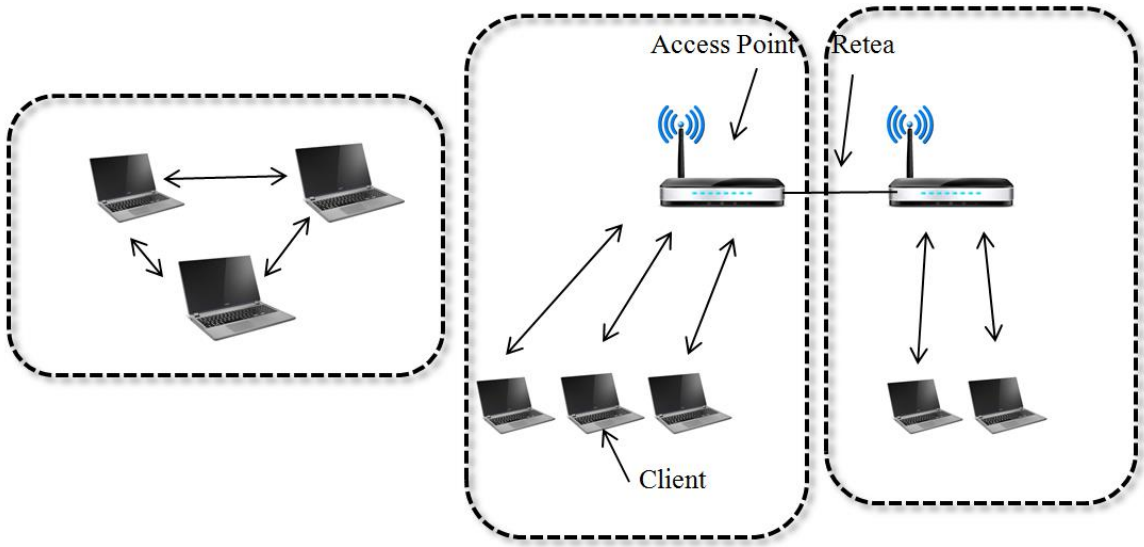
1.1 Informaţii generale

[1]Reţelele şi dispozitivele mobile de tip wireless au devenit din ce în ce mai populare deoarece ele oferă utilizatorilor acces la informaţie şi comunicaţie oricând şi oriunde. Comunicaţiile convenţionale mobile, fără fir, sunt de obicei sprijinite de o infrastructură realizată cu fire fixe. Un dispozitiv mobil ar folosi un singur nod intermediar („hop")de comunicaţie radio pentru a accesa o staţie de bază care îl conectează la întreaga infrastructură. În schimb, reţelele ad-hoc nu folosesc o infrastructură fixă. Nodurile dintr-o reţea mobilă ad-hoc comunică prin intermediul unui singur nod intermediar sau prin mai multe noduri intermediare într-o conexiune cap-la-cap. Nodurile intermediare dintr-o pereche de noduri ce comunică se comportă ca nişte routere. În plus, nodurile se comportă ca nişte gazde, la fel ca şi routerele. Nodurile reţelelor ad-hoc pot fi mobile, deci crearea unor căi de rutare poate fi afectată prin adăugarea sau ştergerea unor noduri. Topologia unei reţele se poate modifica aleator, rapid şi într-un moment neaşteptat.

Standardul care defineşte reţelele de tip wireless este 802.11. Reţelele 802.11 pot fi folosite în două moduri. Cel mai cunoscut mod este de acela în care putem conecta clienţi (ex. laptopuri şi telefoane) la o altă reţea (ex. Internet). Dacă discutăm despre modul bazat pe infrastructură, atunci fiecare client este asociat unui punct de acces(AP-Access Point) care este conectat la rândul lui la o alta reţea. Clientul trimite şi primeşte pachete prin intermediul acestui AP. În schimb, dacă discutăm despre reţea ad-hoc, atunci ne vom gândi la o colecţie de calculatoare asociate care pot să işi trimită direct date unul altuia.

Rețelele de tip ad-hoc au o importantă deosebită în scenarii cum ar fi aplicațiile militare, îngrijire medicală, conferințe sau în spațiile de explorare. În ultimii ani, rețelele de tip ad-hoc au au atras atenția cercetătorilor datorită organizării lor eficente precum și abilității de a-și gestiona topologia dinamică necentralizată. Fiecare nod din rețea joacă dublul rol atât de terminal cât și de ruter sub ipoteza că nu toate nodurile pot avea conectare directă unele cu celelelalte.

Reţelele de tip ad-hoc sunt folosite în multe aplicaţii şi nu necesită sprijinul unei infrastructuri. Asigurarea comunicaţiilor pe câmpurile de luptă sau în zonele afectate de dezastre reprezintă unul din exemplele în care reţelele de tip ad-hoc au o utilizare însemnată.



Reţea de tip Ad-Hoc Reţea bazata pe infrastructura

Figura. 1.1 Reţeaua wireless [2]

[7]Diferența dintre rețeaua bazată pe infrastructură și cea mobilă, de tip ad-hoc, mai poate fi exemplificată și prin următoarele figuri:

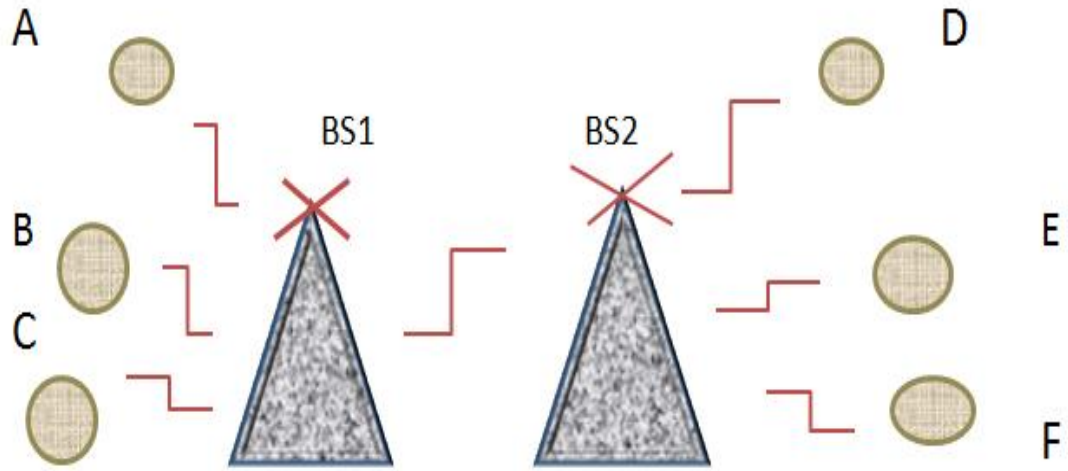


Figura 1.2 Rețeaua bazată pe infrastructură[7]

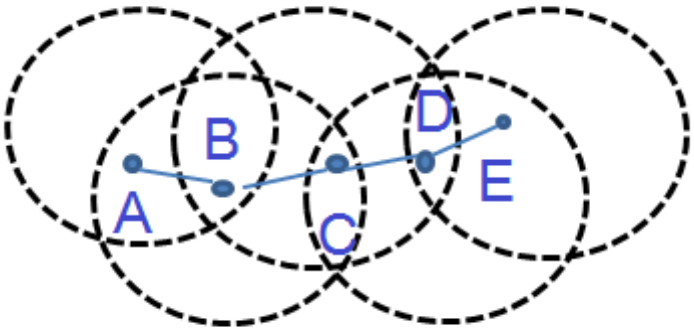


Figura 1.3 Rețeaua de tip ad-hoc[7]

În prima imagine, se poate observa că nodurile A,B, și C comunică între ele prin intermediul Stației de Bază 1(BS1), iar nodul A comunică cu nodul E prin intermediul lui BS1 dar și a lui BS2.

Însă, în imagina a doua, se arată faptul că în cazul rețelelor de tip ad-hoc, nodul A poate comunica cu nodul E prin intermediul nodurilor B,C și D, acestea având denumirea de noduri de încredere(RNs-Relay Nodes). Fiecare nod are posibilitatea de a-și descoperi vecinii.

1.2 Caracteristicile reţelelor ad-hoc

Noţiunea de reţea ad-hoc face referire la o reţea wireless dinamică, fără o infrastructură fixă, formată din noduri mobile, care foloseşte interfeţe wireless pentru a trimite pachete de date.

Una dintre caracteristicile reţelelor ad-hoc este mobilitatea, aceasta fiind reprezentată de rapiditatea prin care nodurile reţelei pot fi repoziţionate sau mutate. Putem avea mobilitate individuală aleatorie, mobilitate de grup, mișcare de-a lungul rutelor pre-planificate, etc. Mobilitatea poate avea un impact major asupra selecției unui sistem de rutare, putând influența performanţa.

O altă caracteristică a reţelelor de tip ad-hoc este aceea că pentru a trimite un pachet de date de la sursă la destinaţie, este nevoie de traversarea mai multor noduri intermediare.

De asemenea, foarte important este faptul că reţelele mobile ad-hoc işi înşuşesc automat propriile configuraţii. Parametrii acestor configuraţii ar fi: adresarea, rutarea, idetificatorul de poziţie, controlul puterii etc. În unele cazuri, unele noduri pot să îşi coordoneze mişcarea şi să fie distribuite dinamic în arii geografice diferite, pentru a putea furniza informaţii în legatură cu porţiunile de reţea deconectate.

Multe dintre nodurile ad-hoc(cum ar fi: laptopuri, senzori) au alimentare limitată şi nu pot să işi genereze singure energia. Pentru a avea o acţiune de lungă durată este esenţială existenţa unui protocol eficient de conservare a energiei .

Dacă ţinem cont de faptul că o reţea ad-hoc poate fi formată din mii de noduri, putem spune că am ajuns la un element care constituie o problema în dezvoltarea reţelei. Însa, pentru o reţea wireless, scalabilitatea reţelei poate fi simplu de manipulat prin implementarea unei structuri ierarhice.

În ceea ce priveste securitatea, reţelele wireless sunt mai puţin sigure faţă de reţelele cablate, din cauza faptului că persoanele neautorizate pot accesa uşor reţeaua în zonele de acoperire.

1.3 Proiectarea nivelelor încrucișate

[7]Modelul de referinţă OSI este format din 7 nivele: fizic, legatură de date, rețea, transport, sesiune, prezentare şi aplicație. Avantajul acestei tehnici de împarțire este de a grupa funcții de comunicare similare în nivele logice. Un nivel trebuie sa păstreze legatura cu nivelul de desupra lui precum și cu cel ce îl precedă. Cu toate acestea, când protocolul de control al transmisiei(TCP), al nivelului transport și protocolul Internet(IP), al nivelului rețea,au fost definite, modelul celor cinci nivele (modelul TCP/IP) a devenit unul de bază. În concluzie, modeulul TCP/IP este alcătuit din nivelul aplicație, trasport, Internet și acces rețea.

Stiva nivelelor Funcții de bază

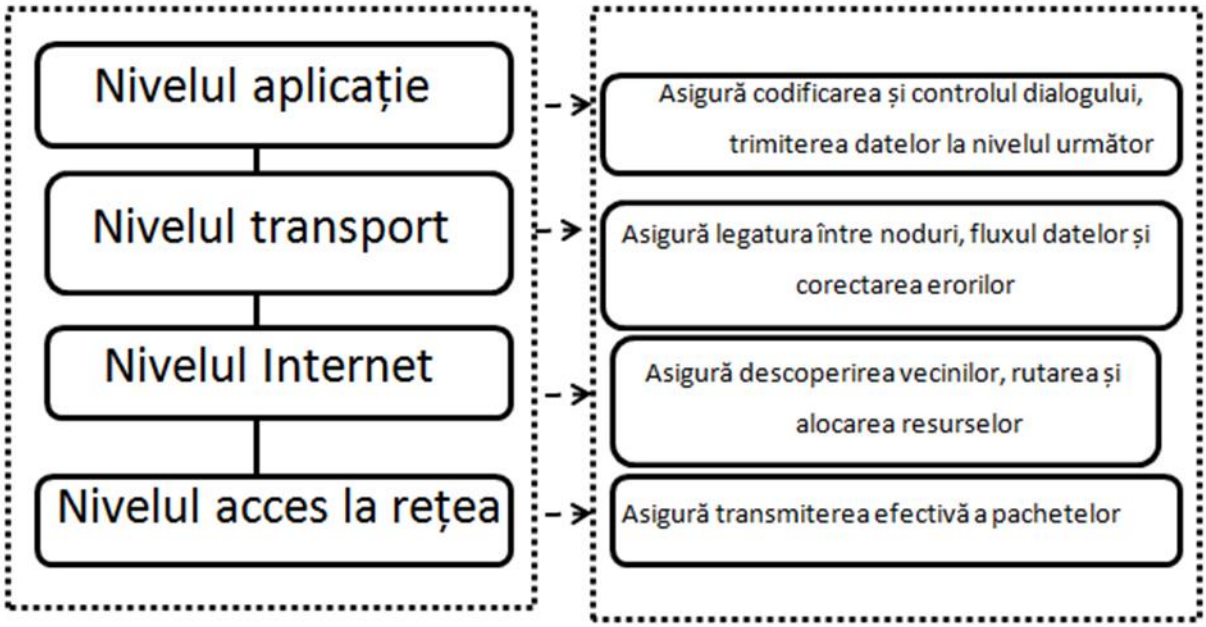


Figura 1.4.Modelul TCP/IP[7]

a) Nivelul acces la rețea se ocupă cu accesul media, restabilirea erorilor, retransmiterea și gestionarea cozior. Acesta este format din două subnivele: Controlul accesului media(MAC-Media Access Control) și Controlul legăturilor logice(LLC-Logical Link Control).

b) Nivelul Internet este responsabil pentru descoperirea vecinilor, rutarea și alocarea resurselor. Rutarea este funcția principală a nivelului rețea, asigurând transmiterea unor pachete de date prin rețea de la sursă la destinație. Numeroase protocoale de rutare au fost proiectate pe baza protocolului IP pentru îndeplinirea cerințelor rețelei wireless de tip Ad-Hoc.

c) Nivelul transport este responsabil de controlul fluxului de date, controlul congestiei, reordonarea pachetelor și organizarea conexiunii capăt-la capăt. Acest nivel transportă pachetele de date către diferite destinații care au fost descoperite la nivelul rețea. De asemenea, acest nivel asigură monitorizarea transmiterii datelor la destinație și evitarea apariției congestiei.

d) Nivelul aplicație constituie interfața cu utilizatorul. Luând în considerare cererile acestuia, nivelul aplicație împarte serviciile în diferite categorii, cum ar fi: servicii în timp real, servicii continue și intermitente.

Chiar dacă arhitectura nivelelor are propriile avantaje și joacă un rol important în rețeaua cablată, din punct de vedere al portabilității, flexibilității și complexității scăzute a proiectării, ea nu este potrivită pentru rețelele wireless, în special pentru rețelele wireless de tip Ad-Hoc. Motivul nepotrivirii este reprezentat de faptul că servicile oferite sunt realizate de protocoale specifice, pentru nivele diferite și astfel este împiedicată comunicarea directă între nivelele neadiacente. Comunicarea dintre nivelele adiacente este limitată la apelarea procedurilor și la răspunsurile

primite. Astfel că, având o proiectare strictă a nivelelor, aceasta nu este destul de flexibilă pentru rețelele mobile, dinamice de tip Ad-Hoc, cauzând performanțe scăzute.

Așadar, conceptul de proiectare încrucișată a nivelelor a fost propus pentru a obinție câștig de performanță prin exploatarea interacțiunii nivelelor diferite. Operația de încrucișare poate fi privită ca o încălcare a arhitecturii acestora, presupunând mai multă interacțiunie între nivele, pe lângă interacțiunea dintre nivelele adjuncte. Proiectarea încrucișată solicită schimb clar de informații între nivele, adaptarea acestor informații la fiecare nivel precum și construirea unui grad fix de diversificare în fiecare nivel pentru a îmbunătăți propunerile robuste.

Proiectarea încrucișată a nivelelor pune la dispoziție două tipuri de informații. Una dintre ele face ca variabilele unui nivel specific să fie vizibile și de celorlalte nivele. Cealălată oferă serviciul de stocare al informațiilor celorlalte nivele.

Operația de încrucișare a nivelelor ajută la proiectarea rețelelor wireless de tip ad-hoc. De exemplu, un pachet trebuie să fie retransmis la nivelul legătură de date sau transmiterea puterii trebuie să fie ajustată pentru a garanta transmisia integrală, ceea ce poate determina interferență pe celealte noduri sau o dispută agresivă pentru canalele de acces. La nivelul rețea, ruta curentă poate deveni invalidă iar procesul de repararea al rutei trebuie să fie activat sau măcar un nou proces de descoperire al rutelor noi trebuie să fie inițializat. Ca o consecință, este consumată mai multă energie, iar timpul de întârziere al transmiterii unui pachet de la sursă la destinație va crește, în timp ce rata de transfer va crește și ea. Din acest motiv, trebuie să aibă loc o adaptare atentă a stivei de protocoale la fiecare nivel pentru a putea fi compensate variațiile unui nivel.

Adaptarea locală a parametrilor fiecărui nivel cât și adaptarea celorlalte nivele trebuie luată în considerare. De exemplu, puterea transmisă, informația ratei de transmisie, schemele de codare și modulație, rata erorii cadrelor(FER-Frame Error Ratio) și mobilitatea din nivelul fizic constituie parametrii importanți a căror împărțire cu celelalte nivele poate fi benefică. Proiectarea protocolului primelor nivele trebuie să ia în considerare informația adunată de la nivelul fizic în scopul minimizarii cunsumului de energie, alocarea resurselor, controlarea cozilor. Între timp, numărul retransisiilor, la fel ca rutarea și topologia rețelei, asociate informației primite de la nivelele superioare pot fi împărțite în mod benefic.

Aceste nivele încrucișare se pot clasifica în diferite categorii în funcție de diferitele cereri ce pot fi aplicate. Ele sunt proiectate pentru a reduce consumul de energie, întârzierea pachetele când sunt transmise de la sursă la destinație, îmbunătățirea debitului retelei chiar și pentru optimizarea constrângerilor.

Așa cum am spus și mai devreme, proiectarea încrucișată a niveleleor are beneficii substanțiale dar prezintă și dezavantaje. Spre exemplu, interacțiunile nivelelor încrucișate creează dependeță între nivele, acest lucru afectând nu numai nivelul propriu-zis dar și celelalte nivele. În plus, o reproiectare completă a rețelei și a protocoalelor va duce la un cost foarte mare. Proiectarea nivelelor trebuie să fie implementată cu atenție deoarece din momentul când structura OSI este încălcată, beneficiile independenței și proiectarea specifică a protocoalelor va dispărea. Efectele fiecărui protocol ales pentru fiecare nivel trebuie alese cu atenție.

Capitol 2: Protocoale de rutare.

Nivelul rețea joacă un rol important în rețelele de tip ad-hoc, influențând performanța totală a sistemului. Nivelul rețea este responsabil pentru alocarea adreselor IP și pentru alegerea rutei corecte, asigurând comunicarea între sursă și destinație.

[3]Găsirea rutelor în reţelele mobile de tip ad-hoc reprezintă un element important şi dificil. Un nod se poate muta într-o nouă locaţie şi trebuie să gasească o nouă cale către destinaţie de la această nouă locaţie. Protocoalele care sunt folosite pentru a rezolva aceasta problemă au latenţă. Această latenţă poate fi minimizată prin preluarea datelor de rutare de la vecinii nodului curent.

Astfel , protocoalele de rutare folosite se pot împarţi în trei categorii : protocoale de rutare proactive, protocoale de rutare reactive şi protocoale de rutare hibride.

- protocoalele de rutare proactive oferă performanţe bune în găsirea rapidă a rutei şi sunt bazate pe tabela de rutare. Avantajul acestui tip de protocol de rutare este acela că, după stabilirea rutelor, acestea vor fi întotdeauna disponibile. Dezavantajul este reprezentat de supraîncărcărea continuă a reţelei din cauza traficului de control(DSDV)

- Protocoalele de rutare reactive sunt stabilite la cerere, acestea nu îşi actualizează permanent tabela de rutare şi nu au o tabelă de rutare permanentă. Principalul dezavantaj al folosirii unui protocol reactiv constă în inundarea reţelei cu mesaje trimise de către echipamente care nu au nici o posibilitate de a comunica cu ehipamentul solicitat.(DSR,AODV)

- Protocoalele de rutare hibride sunt o combinaţie de protocoale de rutare reactive şi proactive.

Putem împărți toată rețeaua ad-hoc în câteva mici secțiuni și pentru fiecare secțiune în parte, protocolul proactiv se va ocupa cu determinarea legăturilor dintre noduri. În schimb, între secțiuni, protocoalele de rutare reactive sunt folosite pentru a reduce numărul de pachete de control. Protocoalele hibride sunt folosite în rețelele ad-hoc de mari dimensiuni.

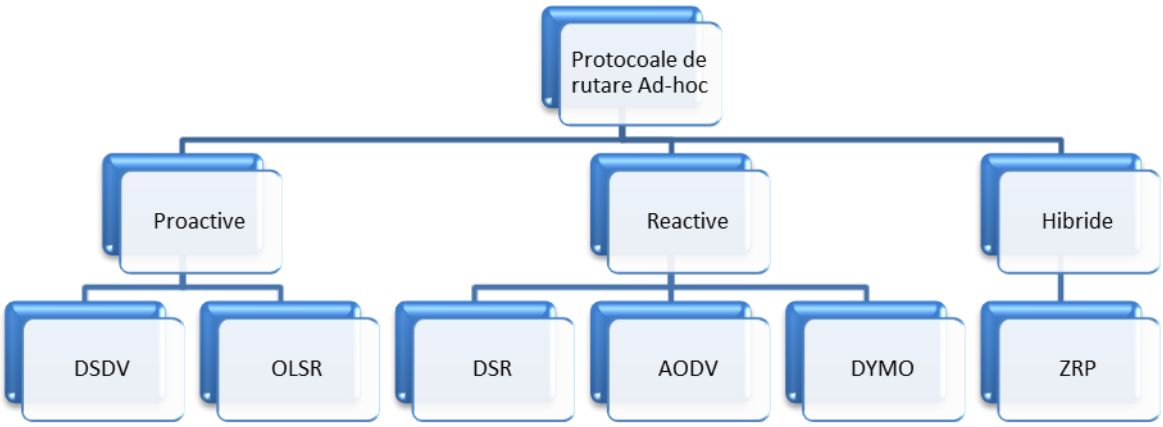


Figura 2.1Schema protocoalelor de rutare[7]

2.1 Protocolul de tip DSDV(Destination-Sequenced Distance-Vector)

[7]Protocolul de rutare DSDV(Destination-Sequenced Distance-Vector-„ Vector Distanţă cu Destinaţie Ordonată") a fost primul protocol dezvoltat pentru reţelele de tip ad-hoc. Fiecare nod deţine o secvenţă de numere crescătoare. De asemenea, mai deţine şi cel mai mare număr de ordine pentru fiecare destinaţie, în tabela de rutare. Informaţiile referitoare la distanţa/metrica fiecărui destinatar, se schimbă prin intermediul actualizărilor rutării dintre vecini cu ajutorul protocoalelor de vectori distanţă. Aceste secvenţe de numere sunt folosite pentru a determina informaţiile noi referitoare la distanţa dintre două noduri pentru aceiaşi destinaţie.

La baza protocolului DSDV se află transferul de mesaje de control între nodurile din rețea. În acest tip de mesaje se găseşte întreaga tabelă de rutare pe care o deţine fiecare nod. Traficul de control este trimis fie folosind o adresare MAC de nivel 2 fie o adresare de nivel 3 (IP). Algoritmii bazaţi pe starea legăturilor selectează rutele optime cu ajutorul metodei celui mai scurt drum (Shortest Path First). Fiecare nod gestionează o "hartă" ce descrie topologia curentă a reţelei. Această hartă este actualizată regulat prin testarea posibilităţii de a accesa diferite părţi ale reţelei şi prin schimbul de informaţii cu alte rutere. Determinarea celei mai bune căi („shortest path") poate fi făcută pe baza unor metrici diferite care indică costul trimiterii unei datagrame pe o anumită rută.

Protocoalele de tip proactive transmit periodic pachete de test pentru a identifica posibilele rute din rețea. Astfel că fiecare nod trebuie să dețină o tabelă de rutare care conține toate rutele de la el către toate nodurile din rețea. Avantajul acestui tip de protocol este că timpul necesar descoperiri rutei este foarte mic. În schimb, este necesar ca nodurilor să dețină câte o tabelă de rutare. Dacă numărul de noduri din rețea crește, atunci și tabela de rutare va deveni mai mare, ocupând o memorie mai mare. Pe de altă parte, transmiterea periodică de pachete de test va duce la creșterea încărcării rețelei.

2.2 Protocolul de rutare DSR(Dynamic Source Routing)

[5]DSR este un protocol de rutare reactiv şi este numit protocol de rutare la cerere(„on demand routing protocol"). Este un protocol de rutare sursă si de accea este simplu şi eficient. Protocolul DSR face ca reţeaua să fie foarte bine organizată şi configurată.

Acest tip de protocol este caracterizat de faptul că sursa cunoaşte topologia rutei şi fiecare nod, până la destinaţie. Pachetele de date conţin în antetul acestora rutele sursei.

Când un nod din reţeaua ad-hoc doreşte să trimită un pachet de date către o destinaţie pentru care nu cunoaşte căile de rutare, el foloseşte processul de descoperire al rutelor(„route descovery process") pentru a determina calea în mod dinamic. Acest proces trimite pachete cu cereri pentru aflarea rutelor. Fiecare nod primeşte câte o cerere şi o retransmite doar în cazul în care nu este nod destinaţie sau în cazul în care are în „memoria" sa o altă cale către destinaţie. Un astfel de nod răspunde cererii cu un pachet care conţine ruta şi pe care îl trimite către sursa originală.

2.2.1 Destoperirea rutelor

Atunci când o sursă dorește să trimită un pachet către un nod destinație, ea plasează în antetul pachetului secvența de noduri intermediare prin care pachetul trebuie să treacă pentru a ajunge la destinația dorită. În mod normal, sursa deține o rută favorabilă în memoria rutelor sale pe care le-a obținut prin căutări anterioare, dar dacă nu găsește nicio rută pentru destinația dorită, ea va genera un proces de descoperire al rutelor.

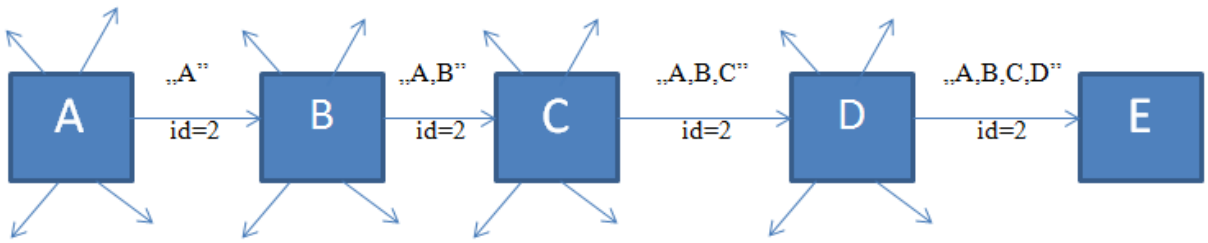


Figura 2.2 Procesul de descoperire a rutelor[5]

Dacă luăm în consderare exemplul din figură, A reprezintă nodul sursă, iar E este nodul destinație. Pentru a ilustra procesul de căutare al rutelor, nodul sursă va genera un mesaj RREQ care va fi primit de toate nodurile care se află în aria de transmisiune a lui A. Fiecare RREQ identifică inițiatorul și targhetul cererii și se identifică printr-un număr unic („request id"). În plus, RREQ mai conține și adresele tutuor nodurile intermediare prin care s-a trecut pentru a ajunge la destinație. Această înregistrare de rute este inițializată într-o listă goală de sursa care a generat RREQ-ul.

Atunci când un alt nod primește mesajul RREQ, dacă este un nod destinație, va trimite un mesaj de răspuns RREP către inițiatorul cererii, transmițându-i o copie a rutei. Când sursa primește răspunsul, va copia mesjul în memoria sa pentru a-l putea folosi la trimiterea pachetelor pe ruta respectivă la destinația dorită.

În schimb, dacă nodul care primește RREQ-ul observă că a mai răspuns unei cereri de la aceeași sursă, cu același număr de indetificare sau dacă observă că adresa sa este trecută pe lista rutelor înregistrate din cerere, el va anula cererea. Altfel, nodul va aduga adresa sa în lista de înregistrare a rutelor și va trimite cererea la următoarele noduri.

Atunci când s-a ajuns la nodul destinație E, se dorește să se transmită răspunsul cu ruta dorită către nodul sursă A. Nodul E va căuta în memoria sa dacă găsește o rută pentru a întorce mesajul, dacă o găsește, mesajul va ajunge la sursă, în caz contrar, nodul E va genera și el un nou proces de descoperire a rutelor către nodul dorit.

Când este inițiată o desoperire a rutelor, nodul sursă va deține o copie a pachetului original într-un „buffer"local. Acest „buffer" conține o copie a fiecărui pachet ce nu a putut fi transmis pentru că nu se știa ruta sa către destinație. Fiecare pachet este marcat cu timpul la care a fost adăugat în „buffer". Pentru a preveni supraîncărcarea „buffer"-ului este folosită tehnica First-In-First-Out(FIFO) sau alte strategii pentru a putea trimite pachetele în rețea.

Cât timp un pachet se află în „buffer", nodul trebuie să inițieze procese de căutare a rutelor pentru a descoperi adresa destinației. În particular, din cauza limitării ariei de transmisie, a mobilității nodurilor, rețeaua poate deveni partiționată, ceea ce însemnă că este posibil să nu existe nicio secvență de noduri prin care să treacă un pachet pentru a ajunge la destinație.

2.2.2 Păstrarea rutelor

[6]Atunci când are loc transmiterea unui pachet în rețea, folosind ruta sursei, fiecare nod care transmite pachetul este responsabil să confirme dacă pachetul a fost primit de următorul nod intermediar.

Dacă luăm în discuție exemplul inițial, nodul A a transmis un pachet către nodul E folosind ruta sursei. Pachetul trebuie să treacă prin nodurile intermediare: B, C și D. În acest caz, nodul A este responsabil ca pacheul să fie primit de nodul B, nodul B este responsabil de primirea pachetului de către nodul C, nodul C va răspunde pentru primirea pachetului de către nodul D și în final, nodul D este responsabil pentru primirea pachetului de către nodul E.

Dacă pachetul este retransmis de câteva noduri intermediare și nu este primit niciun mesaj de confirmare de primire, nodul va trimite mesaje de eroare RERR către nodul care a generat trimiterea pachetului, putându-se indetifica locul în care s-a produs întreruperea.

De exemplu, dacă nodul C nu poate transmite pachetul către nodul D, nodul C va trimit un RERR către nodul A, știindu-se faptul că legătura dintre C și D este întreruptă. Nodul sursă va șterge din memoria sa legătura dintre C și D. Dacă se dorește iar transmiterea unui pachet de la A la E și dacă nodul sursă are o rută disponibilă, pachetul va fi trimis imediat.

Figura 2.3 Apariția unei întreruperi în rețea[6]

2.2.3 Formatul pachetelor DSR

[6]Antentul mesajor de tip cerere(RREQ):

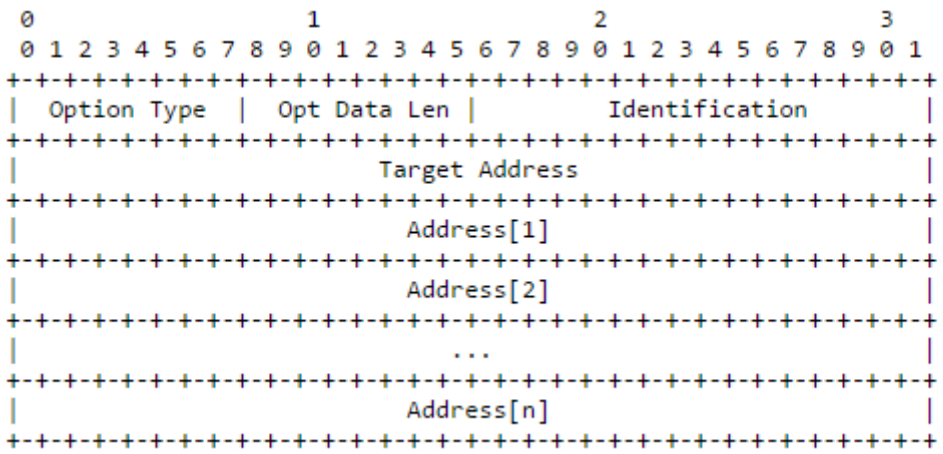


Figura 2.4 Antetul mesajelor de tip RREQ[6]

Câmpurile antetului mesajelor de tip RREQ sunt următoarele:

Tipul opțiunii- implicit are valoarea 1;

Lungimea datelor opționale- măsurată în octeți, trebuie sa aibă valoarea (4*n)+6, unde n reprezintă numarul de adrese din mesajul RREQ;

Identificatorul-reprezintă o valoare unică generată de inițiatorul mesajului RREQ. Nodurile care inițiază o cerere de rute, generează și o nouă valoare a indentificatorului pentru fiecare cerere. Această valoare permite unui nod de recepție să determine dacă el a văzut vreo copie a mesajului RREQ. Dacă valoarea identificatorului este găsită de acest nod în tabela sa de cererei, acesta trebuie să anuleze cererea. Când un RREQ este transmis, acest câmp trebuie să fie copiat de la cererea primită;

Adresa țintă-adresa nodului care reprezintă ținta cereri(adresa nodului destinație);

Adresa[1...n]- adresa[i] este adresa Ipv4 a nodului „i” înregistrată în mesajele RREQ. Adresa ce se află în câmpul adresei sursei a antetului IP este adresa sursei care a inițiat procesul de descoperire a rutelor și nu trebuie să se afle în campul adresa[i]; adresa care se află în adresa[1] reprezintă adresa Ipv4 a primului nod din cale, după sursă;

Antetul mesajelor de răspuns(RREP):

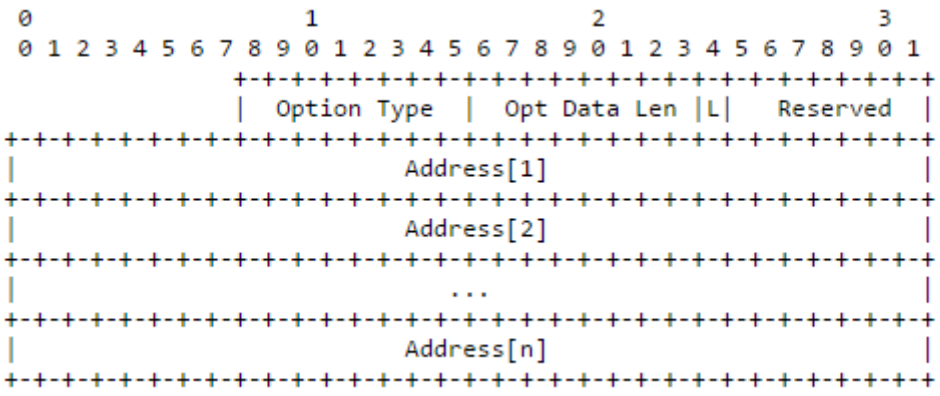


Figura 2.5 Antetul mesajelor de tip RREP[6]

Tipul opțiunii- este implicit 2 pentru cererile de tip RREP;

Lungimea datelor opționale- este de forma unui întreg, fără semn, pe 8 biți. Lungimea opțiunii, măsurată în octeți, trebuie să fie calculată cu relația (4*n)+1, unde n reprezintă numărul de adrese din mesajul RREP;

L-ultimul nod intermediar, extern- este setat pentru a indica faptul că ultimul nod intermediar prezent în mesajul RREP reprezintă defapt o cale arbitrară ce leagă rețeaua externă de rețeaua DSR. Rutele care nu se află în rețeaua DSR nu se află nici în antetul mesajului RREP.

Rezervat- este setat la valoarea 0 și este ignorat la recepție;

Adresa[1..n]- adrsele nodurilor intermediare dintr-o rută de la destinație către sursa care a generat cererea de rutare.

Antetul mesajelor de eroare(RERR):

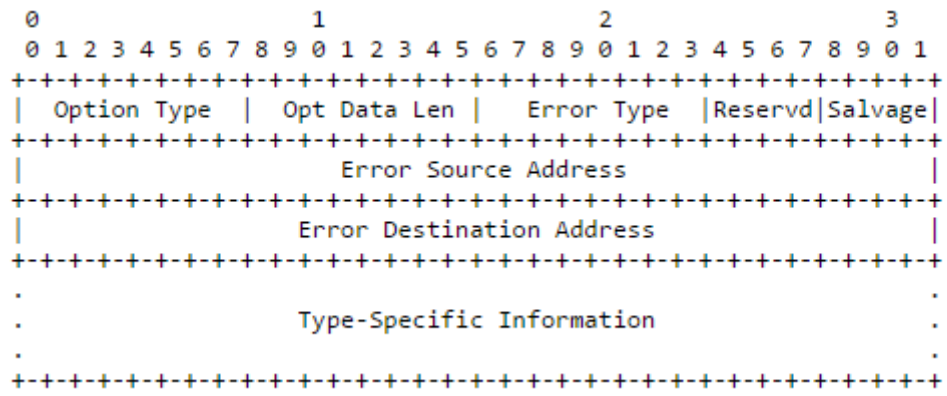


Figura 2.6 Antetul mesajelor de tip RERR [6]

Tipul opțiunii- este setat la valoarea 3;

Lungimea datelor opționale- o valoare întreagă, fară semn, pe 8 biți.În ceea ce privește mesajele de eroare, acest câmp trebuie să aibă valoarea 10, la care se adaugă mărimea oricărui tip specific de informație prezent în RERR;

Tipul erori:

-dacă valoarea câmpului este 1, însemană că nodul este inaccesibil(eroare: „NODE_UNREACHABLE”);
-dacă valoarea câmpului este 2, însemnă că fluxul de date nu este suportat(eroare: FLOW_STATE_NOT_SUPPORTED);
-dacă valoarea câmpului este 3, însemnă că opțiunea nu este suportată(eroare: OPTION_NOT_SUPPORTED).

Rezervat- trebuie să fie setat la 0 și este ignorat la recepție;

Adresa sursei erorii- adresa nodului care a generat RERR(de exemplu: nodul care a încercat să trimită un pachet în rețea dar a descoperit o legătură ruptă);

Adresa destinației erorii- adresa nodului către care trebuie trimis RERR. De exemplu, dacă avem o eroare care ne spune că un nod este inaccesibil, acest câmp va fi setat cu adresa nodului care a generat informația spunând că nodul eronat este defapt unul valid;

Tipul specific al informației- reprezintă informația specifică tipului erorii a mesajului RERR.

2.3 Protocolul de rutare AODV(Ad hoc On-demand Distance Vector)

[5]Acest tip de protocol are aceleaşi caracteristici ca şi protocolul DSR, deoarece are proces similar de descoperire a rutelor. Totuşi, AODV adoptă un mod diferit de stocare a informaţiilor de rutare folosind tabelele de rutare tradiţionale cu o intrare pentru fiecare destinaţie. Făcând comparaţie, DSR, poate memora mai multe rute de intrare pentru fiecare destinaţie.

De asemenea, AODV(la fel ca şi DSDV) foloseşte secvenţa de numere destinaţie pentru a preveni buclele de rutare şi pentru a avea mereu informaţii noi despre rute. Aceste secvenţe de numere sunt conţinute de toate pachetele de rutare.

În plus, AODV foloseşte secvenţele de numere pentru a avea mereu rutele actualizate ceea ce face ca rutele vechi să fie înlăturate chiar dacă există posibilitatea ca acestea să fie încă valide.

Spre deosebire de celelalte protocoale prezentate, AODV deţine un „timer" ce reprezintă un mecanism de expirare al rutelor, pentru a putea şterge cât mai repede rutele vechi.

Protocoalele de tip reactive implică trimiterea de pachete care să descopere rutele către destinație. Ca și beneficiu, nu mai este nevoie ca fiecare nod să dețină câte o tabelă de rutare pentru a memora rutele către toate nodurile. Ele doar memorează rutele care au fost găsite în procesul de descoperire. Această tehnică reduce încărcarea rețelei în comparație cu protocolele de rutare proactive. Mutarea nodurilor în rețea, schimbă tolopologia acesteia, ceea ce determină transmiterea mai multor pachete de control pentru a menține sesiunea de comunicare curentă.

2.3.1 Descoperirea rutelor

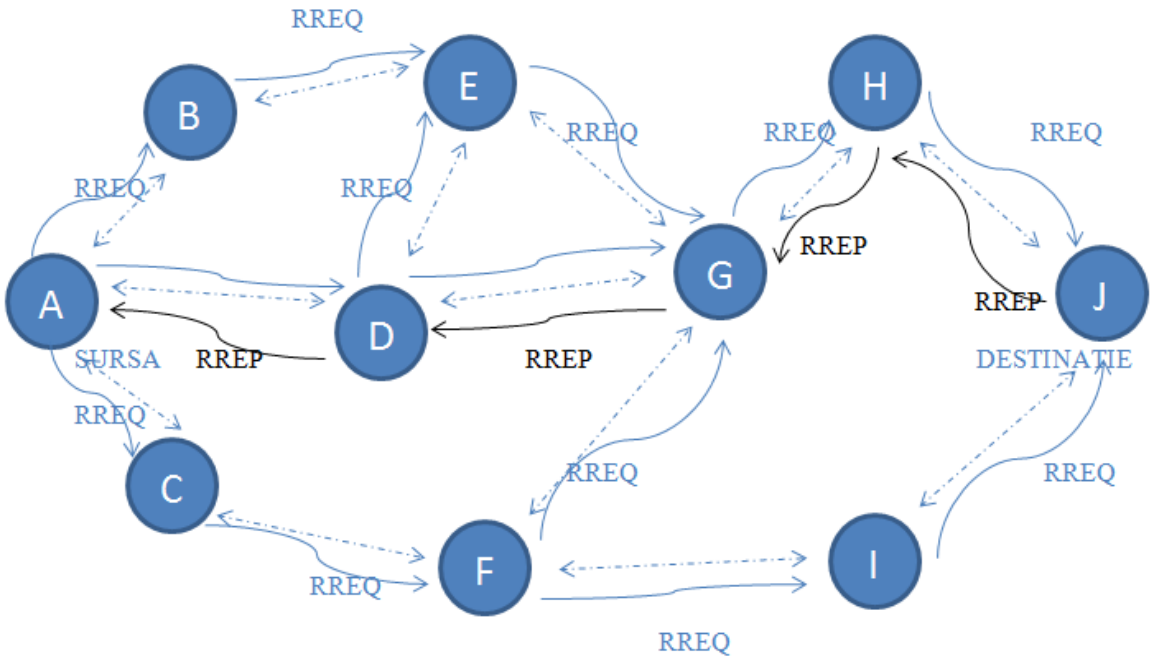


Figura 2.7 Procesul de descoperirea rutelor[AODV]

Fiind un protocol reactiv(la cerere) atunci când un nod dorește să transmită pachete, prima dată el începe un proces de descoperire a rutelor, „inundând" rețeaua cu cereri RREQ(Route Request). Pachetele RREQ sunt transmise către toate nodurile din rețea până când se ajunge la destinație. În drumul către destinație, RREQ informează toate nodurile intermediare despre ruta către sursă.

Atunci când RREQ descoperă destinația, aceasta trimite un pachet de tip răspuns(RREP-Route Reply) ce conține ruta descoperită de RREQ. Acesta anunță nodurile intermediare în legătură cu ruta descoperită pentru trimiterea pachetului de la sursă la destinație. După ce RREQ și RREP sunt transmise către destinațiile lor, fiecare nod intermediar va ști către care nod să trimită pachetele cu scopul de a ajunge la sursă sau la destinație. În plus, pachetele de date nu trebuie să dețină adresele tuturor nodurilor din rețea, ele trebuie să conțină doar adresa nodului destinatar, scăzând astfel încărcarea rețelei.

2.3.2 Păstrarea rutelor

Un al treilea tip de mesaje de rutare ar fi, cel de eroare(RERR-Route Error).Când o rută pentru trimiterea pachetelor este ruptă, soluția este de a trimite un mesaj de eroare(RERR) către nodul sursă. Astfel se va iniția un proces nou pentru descoperirea rutei către detinație.

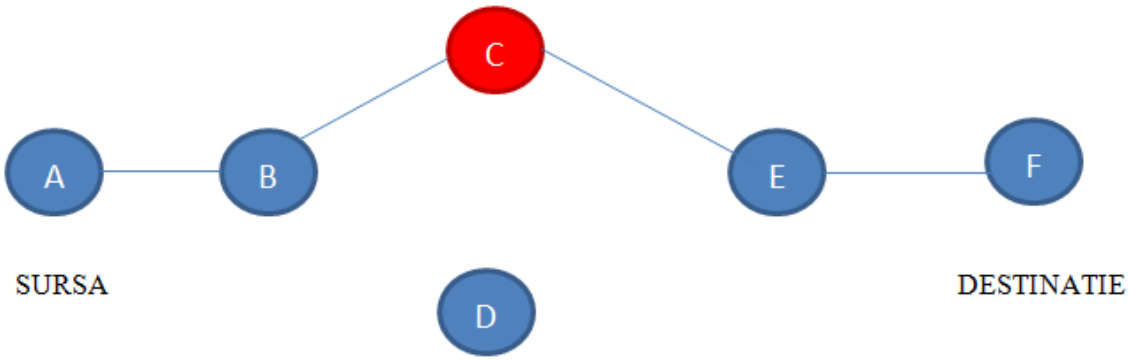


Figura 2.8 Dispariția unui nod din rețea

Să presupunem că aveam ruta de mai sus de la A la F: A→B→C→E→F. Dacă nodul C va dispărea din aria de transmitere a nodului B, ruta va fi ruptă între sursă și destinație. Când nodul B va încerca să transmită un pachet către nodul C, nodul B ar trebui să aibă un canal de comunicare pentru a transmite pchetul. Din momentul ce canalul nu mai există, nodul B va observa că legătura cu nodul C este ruptă, pentru că nu va mai primi niciun răspuns de la C. Astfel, B va trimite un mesaj de eroare către sursă.

Atunci când este primit un mesaj de eroare, nodurile intermediare, care trimiseseră pachete flosind acea rută, vor invalida informația referitoare la ruta care a dispărut. Nodul sursă A, va cere să se realizeze un nou proces de descoperire a rutelor pentru a determian un nou drum de transmitere al pachetelor.Această metodă folosită pentru descoperirea rutelor poate fi ineficientă. Trimiterea unui mesaj de eroare către nodul sursă și invocarea unui nou proces pentru descoperirea rutelor, poate fi costisitoare. Acest proces face ca transmiterea pachetelor să dureze foarte mult.

2.3.3 Formatul pachetelor AODV

[5]Formatul mesajelor pentru cererile de rute(RREQ)

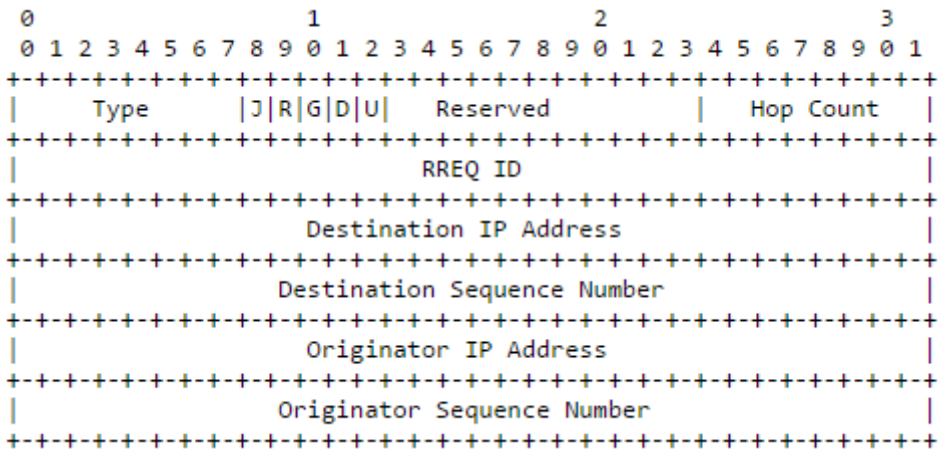


Figura 2.9 Antetul mesajelor de tip RRE[5]

Câmpurile care formează mesajul ce este transmis pentru determinarea rutelor au următoarea semnificație:

Tip-RREQ are predefinit ca valoare pentru Tip, valoarea 1;

J-fanion Join;folosit pentru multicast;

R-fanion Repair; folosit pentru multicast;

G-fanion pentru RREP; indică dacă o cerere RREP poate fi unicast pentru nodul specificat în secțiunea pentru adresă IP a destinației;

D-fanion pentru destinație; indică doar nodul destinație ce a răspuns mesajului RREQ

U-secvența de numere necunoscute; indică faptul că nu se cunoaște secvența de numere a destinației;

Rezervat-are valoarea 0; este ignroat la recepție;

Numărul de noduri intermediare(hop-uri)-indică numărul de noduri intermediare de la adresa IP a nodului sursă până la nodul care a gestionat cererea;

RREQ ID- secvența de numere care identifică, împreună cu adresa IP, mesajul RREQ;

Adresa IP a nodului destinație- adresa IP a destinației, pentru care se dorește aflarea căii;

Secvența de numere a destinației-ultima secvență de numere primită în trecut de sursă, referitoare la orice rută către destinație;

Adresa IP a sursei-Adresa IP a nodului care a generat cererea de căutare a rutei;

Secvența de numere a sursei-Secvența de numere curentă care este folosită de emițătorul cererii RREQ

Formatul mesajului de răspuns (RREP)

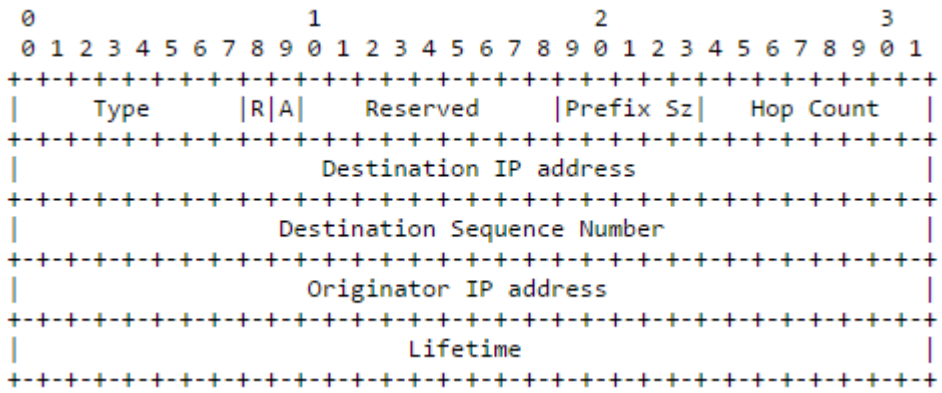


Figura 2.10 Antetul mesajelor de tip RREP[5]

Structura mesajului folosit drept răspuns este următoarea:

Tip-valoarea implicită tipului de mesaje RREP este 2;

R- fanionul Repair; este folosit pentru multicast;

A-fanionul ce indică confirmarea cererii;

Rezervat-este setat la valoarea 0; este ignorat la recepție;

Mărimea prefixului- atunci când este diferită de zero, mărimea prefixului este de 5 biți și specifică faptul că următorul nod intermediar poate fi folosit de orice nod care are același prefix și aceeași destinație.

Numărul hop-urilor- reprezintă numărul nodurilor intermediare de la sursă la destinație.

Adresa IP a destinației- reprezintă adresa IP a destinației pentru care o rută este prevăzută.

Secvența de numere a destinației- secvența de numere a destinației asociată rutei

Adresa IP a sursei- adresa IP a nodului care a trimis mesajul RREQ și pentru care este prevăzută ruta

Timpul de viață- timpul,măsurat în milisecunde, în care se așteaptă primirea mesajului RREP pentru ca o rută să fie considerată validă.

Formatul mesajelor de eroare (RERR)

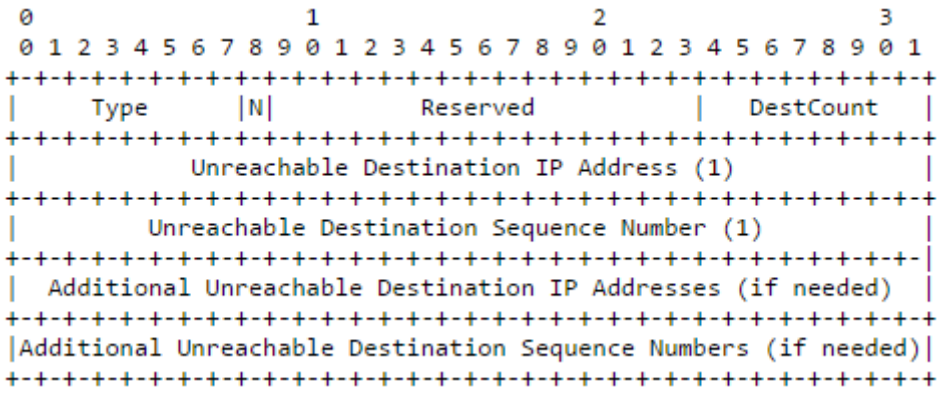


Figura 2.11 Antetul mesajelor de tip RERR[5]

Structura mesajelor de tip eroare este următoarea:

Tip- pentru mesajele de eroare, tipul este implicit 3;

N-reprezintă fanionul de ștergere; este setat atunci când un nod a reparat o legătură, iar nodul sursă nu a șters ruta.

Rezervat-este setat la valoarea 0 și este ingorat la recepție;

Contorul destinaților- numărul destinațiilor inaccesbile ce sunt incluse în mesaj; trebuie să aibă valoarea cel puțin 1;

Adresele IP ale destinațiilor inaccesibile- adresele IP ale destinațiilor care nu au putut fi accesibile din cauza ruperii unei legături;

Secvența de numere a destinațiilor inaccesibile- secvența de numere ce se află în câmpul anterior, din tabela de rutare, a nodului destinație;

Mesajele RERR sunt trimise atunci când are loc o întrerupere de legătură, iar nodurilor destinație nu pot fi accesate de către nodurile intermediare.

Capitolul 3: Controlul congestiei în rețelele mobile Ad-Hoc

[8]În majoritatea rețelelelor wireless, dispozitivele utilizatorilor comunică fie prin intermediul rețelelor cu infrastructură fixă, reprezentate de stațiile de bază, sau direct cu partenerul (folosind 802.11 în modul ad-hoc). O rețea mobile ad-hoc nu deține o infrastructură fixă iar dispozitivele nu trebuie să facă parte din aceeași categorie pentru a realiza comunicarea. Dispozitivele mobile ale utilizatorilor se comportă ca niște rutere și pachetele de date sunt transferate prin noduri intermediare către destinație.

3.1 Problema congestiei

Într-o rețea cu resurse partajate, unde mai multe noduri care trimit pachete sunt în competiție din punct de vedere al lărgimii de bandă, este necesar să se interpreteze rata de trimitere a pachetelor a fiecărui emițător pentru a nu se ajunge la supraîncărcarea rețelei. Pachetele care ajung la un nod și nu pot fi trimise în rețea, sunt aruncate, respinse, în consecință, dacă vin foarte multe pachete în rețea atunci va crește și numărul pachetelor aruncate. Aceste pachete ce au fost respinse au parcurs deja un anumit drum, ceea ce însemna că au consumat resurse. În plus, pachetele pierdute, de obicei, transmit cereri de retrimitere, ceea ce însemnă că vor fi trimise și mai multe pachete de date. Astfel, congestia rețelei va deteriora transferal pachetelor prin rețea.

3.2 Controlul congestiei cu ajutorul protocolului TCP

Controlul congestiei reprezintă responsabilitatea nivelului transport, mai precis, protocolului de control al trasmiterii (TCP-Transmission Control Protocol).TCP combină mecanisme de control ale congestiei și mecanisme de siguranță. Această combinație face să fie permis controlul congestiei fără a avea nevoie de un răspuns explicit legat de starea congestiei rețelei și fără a fi nevoie de participarea nodurilor intermediare. Pentru a detecta congestia în rețea, protocolul TCP observă, pur și simplu, pierderea de pachete. Din moment ce pierderea de pachete este cauzată, în majoritatea cazurilor, de congestie, pierderea unu pachet este interpretată ca un semnal al congestiei în rețea.

TCP se bazează pe primiri de confirmări (acknowledgement): destinația TCP mereu constată atunci când datele sunt primite corect și mai ales complet. Dacă segmentele sunt primite într-o altă ordine față de cum au fost trimise, este trimisă încă o cerere de confirmare(ACK duplicat).

În cazul recepționării unui ACK duplicat, transmițătorul trebuie să se asigure că în rețea are loc o reordonare a pachetelor de date. Dar în cazul în care au loc patru procese de ACK, atunci este clar că în rețea avem parte de congestie.

În plus, TCP folosește un tip de pauză care depinde de timpul în care a fost stabilită conexiunea. Dacă timpul de retransmitere (RTO) se scurge fără a primi niciun mesaj ACK , atunci TCP anunță o situație de congestie în rețea. Pauza până când va avea loc următoarea retransmitere, dacă nu este primit niciun mesaj de ACK, va fi dublă. În plus, această pauză are o creștere exponențială.

3.3 Congestia în rețelele mobile ad-hoc

Mediul vast în care rețelele mobile ad-hoc sunt prezente reprezintă o problemă pentru standardul TCP.

Parametrii specifici proprietăților rețelelor MANET sunt reprezentați de mobilitatea nodurilor și prezența mare a nodurilor intermediare. Putem astfel sublinia faptul că rutele se pot schimba din cauza mobilități nodurilor, iar nesiguranța rețelei duce la instabilitate din punct de vedere al timpilor de întârziere și al pierderilor pachetelor de date.

Folosirea rețelelor cu noduri intermediare multiple permite transmiterea unui singur pachet de test la un moment dat. Acest lucru influeteaza modul în care se manifestă congestia, în rețelele MANET, afectând întreaga arie din cauza mediului partajat. Nu nodurile sunt supraîncărcate ci regiuni din rețea.

Oricum, acest lucru depinde de tipul de rețea, pierderea pachetelor care nu este cauzată de congestia rețelei poate fi mai frecventă în rețelele wireless.

În plus, din cauza lățimii mici de bandă a rețelelor mobile, un singur transmițător este capabil să producă degradarea rețelei din cauza congestiei.

3.4 Moduri de abordare a congestiei

O abordare a problemei în ceea ce privește controlul congestiei o constituie TCP-Feedback(TCP-F). Această abordare propune dezactivarea mecanismului de control al congestiei al TCP-ului atunci când nu apare congestia, pentru a putea fi relatate pierderi sau expirări a unor evenimente cauzate de pierderea rutelor. Transmițătorul TCP este notificat mai ales când apar pierderi de rute(RFN, Route Failure Notification) și atunci când o nouă rută a fost descoperită(RRN-Route Re-establishment Notification). Când o rută către destinație nu este diponibilă, după primirea unui mesaj RFN, transmițătorul TCP-F introduce o stare de alarmare. Prin această stare el oprește valorile TCP-ului referitoare la timer și la mărimea ferestrei.

Un nod intermediar generează un mesaj RFN când detectează o legătură întreruptă pe rută. Odată ce un nod a generat și a trimis RFN-ul,el învață o nouă cale către nodul destinație, generează un mesaj RRN și îl trimite către nodul sursă. Când nodul sursă primește un RRN, el reîncepe sesiunea TCP cu stările valorilor "înghețate" anterioare. Pentru a preveni ca o sesiune TCP-F să rămână în starea de alarmă pe termen nelimitat, în cazul în care un mesaj RRN se pierde, este folosit un contor pentru întoarcere.

Notificările explicite ale căderii legăturilor (Explicit Link Failure Notification-ELFN) sunt folosite pentru a primi un răspuns de la nivelele mai joase pentru a notifica în mod explicit TCP-ul despre legăturile și rutele picate. În cazul unui astfel de eșec , transmițătorul intră în modul „standby", care este echivalent modului "alarmă" al TCP-F-ului.

În contrast cu propunerea TCP-F, nu mai este necesară nicio notificare explicită în cazul în care o rută restabilită este folosită. În schimb, transmițătorul TCP-ELFN-ului trimite pachete de test în interval regulate atunci când se află în starea de asteptare. Aceste pachete de test nu sunt pachete de control special. Primul pachet din coadă transmițătorului este folosit pentru a realiza acest lucru. Starea de așteptare este părăsită odată ce un pachet de test este acceptat de destinatar.

De asemenea, pentru notificările legate de rutele picate, nu sunt trimise pachete de test în ELFN.

Astfel s-a demonstrate că ELFN îmbunătățește performanța TCP-ului, atunci când apar situații în care au loc degradări majore. De asemenea este folosit și atunci când în rețea sunt multe conexiuni active. Motivul este că ELFN este un mecasim care face ca TCP să devină mai agresiv, de exemplu trimite un număr mare de pachete în rețea.

În ceea ce privește TCP-ul cu capacitate de stocare și segmentare a informației (TCP-Bus- „Buffering capability and Sequence information"), idea de notificări explicite de la nivelele de rutare este extinsă.

În TCP-Bus, nodurile intermediare stochează pachete în cazul în care apare pierderea rutelor, în loc să le respingă; intenția este de a nu mai trebui să se retransmită toate aceste pachete. Pentru a preveni pauzele sursei, când pachetele ce au fost stocate sunt trimise la reconstrucția rutei, pauzele pentru aceste pachete sunt mărite.

Luând în discuție ATCP-TCP pentru rețele mobile ad-hoc, putem să subliniem faptul că acest tip de protocol nu doar gestionează foate bine rutele căzute dar are în vedere și gestionarea corectă a perioadelor de discontinuitate și distingerea congestiei de alte tipuri de pierderi. Astfel, în cazul unei congestii sunt trimise notificări de congestie(ECN-explicit congestion notification) .

În loc să invoce controale de congestie standard TCP, în cazurile în care au loc pierderi, dar nu din cauza congestiei, protocolul ATCP floseste mecanismul de "îngheț" al TCP-ului pentru a introduce în rețea pachete de test. Orice fel de pierdere nu este privită ca fiind urmare a unei congesti atât timp cât nu sunt primite mesaje de tip ECN.

În plus, ATCP este privit ca un nivel adițional, sub nivelul transport, reducând astfel interacțiunea cu protocolul TCP.

Posibilitatea de a pierde pachete de date și ACK-uri înainte de modul "îngheț" poate avea efecte negative după ce starea s-a restabilit. Acest lucru poate fi reprezentat ori de apariția pauzelor sau de confirmări(ACK) duplicate. Pentru evitarea acestor problem au fost introduse două mecanisme: notificarea timpurie a pierderilor de pachete(EPLN-Early Packet Loss Notification) și transmiterea corectă a ACK-urilor(BEAD-Best-Effort ACK Delivery).

EPLN notifică transmițătorul în legătură cu secvența de numere a pachetelor pierdute care pot fi recuperate. Astfel, transmițătorul poate să dezactiveze timpii și să retransmită pachetele imediat după ce ruta și-a revenit. BEAD generează notificări referitoare la pierderile de ACK, la nodurile intermediare și le trimite către destinatar. Acest lucru previne pierderea continuă de ACK.

În ceea ce privește TCP-DOOR, pachetele de date și ACK-urile transmise într-o ordine aleatoare sunt utilizate ca indicatori de schimbare de rute fără nevoia unui răspuns explicit. Cât timp nodul destinatar poate să notifice transmițătorul în legătură cu apariția aleatoare a pachetelor de date, transmițătorul va putea, la rândul său, să notifice apariția aleatoare a ACK-urilor.

Ca o reacție la aceste evenimente sunt prezentate două tipuri de mecanisme. Când pachetele neordonate sunt detectate, transmițătorul va dezactiva,temporal, mecanismul de control al congestiei, ținându-l într-o valoare constantă. În plus, acesta se poate întoarce la o anumită stare mai veche. Acest mecanism poartă denumirea de recuperare instantantanee (Instant Recovery). Efectul așteptat este similar cu cel al metodei de "îngheț": după ce parametrii TCP-ului se schimbă, conexiunea continuă ca și cum nu s-ar fi întâmplat nicio schimbare a rutei.

Cea mai bună performanță, se obține când cele două acțiuni se combină: deazctivarea temporală a controlului congestiei și începerea recuperări.

3.5 Construirea unor protocoale alternative

O serie de particularități ale rețelei wireless sunt nefolositoare protocolului TCP în încercarea de a controla congestia rețelelor mobile ad-hoc. Astfel, nu se încearcă modificarea protocolului TCP pentru a avea un anumit comportament ci se dezvoltă protocoale de transmisiune care sunt specifice caracteristicilor rețelei MANET.

Un astfel de protocol este protocolul EXACT. Acest tip de protocol este suportat de rețeaua însăși, de exemplu de nodurile intermediare. Nodurile intermediare dețin variabile de stare pentru toate fluxurile de pachete care se transmit prin ele. Toate nodurile determină lățimea de bandă către vecinii lor.

Rata de informație explcită este introdusă în toate pachetele, de către nodurile intermediare, pentru a putea transmite lățimea de bandă minimă către destinație. Fiecare nod verifică dacă rata de transmise a unui nou pachet este mai mică decât rata curentă specificată în antetul pachetului. Dacă este mai mică, atunci rata va fi scrisă în antetul pachetului înainte ca pachetul să fie trimis în rețea.

Putem spune că acest mecanism este folosit de două ori. Un câmp conține rata curentă a transmițătorului iar un altul rata cerută de transmițător. Pe de o parte, cu această nouă procedură, este posibil ca nodurile intermediare să nu mai ofere o lățime de bandă decât cât este necesar fluxului de pachete iar pe de altă parte, transmițătorul este notificat atunci când are voie să crească rata de transfer peste limita curentă.

O fereastră de siguranță previne transmițătorul să supraîncarce rețeaua în cazul căderii unei rute. Astfel că transmițătorului nu îi va mai fi permis să aibă pachete fără răspuns, mai puține decât sunt prevăzute în fereastra de siguranță.

Un alt protocol care are propietăți asemănătoare cu protocolul EXACT este protocolul de transport Ad-hoc(ATP). El nu folosește timpi pentru retransmiterea pacheteleor ci doar separă strict controlul congestiei de mecanismele de fiabilitate și cere răspuns limitat de la destinatar. În comparație cu EXACT, ATP nu deține variabile de stare referitoare la fluxul de pachete. Toate nodurile intermediare vor calcula o medie exponențială a întârzierii pachetelor ce vor trece prin aceste noduri. Această întârziere reprezintă timpul în care un pachet are de așteptat în coada unui nod până când acesta poate fi transmis în rețea. Aceste valori sunt independete de fluxul de pachete căruia îi aparține.

La fel ca la rata de informație de la EXACT, timpul de întârziere curent este înlocuit în antetul pachetelor, dacă valoarea lui este mai dezavantajoasă față de cea nou calculată. În acest fel, timpul de întârziere maxim poate fi comunicat destinatarului. Destinatarul primește această informație și o retransmite înapoi transmițătorului. În funcție de acest lucru, expeditorul poate să-și determine calea de trasmitere a pachetului.

Pentru a găsi o rată cât mai bună, la începutul unei noi conexiuni se va trimite în rețea un pachet de test ce va colecta informații de la nodurile intermediare referitoare la starea curentă a rețelei.

Un alt protocol de transport pentru rețelele mobile ad-hoc este TPA(Transport protocol for Ad-hoc Networks). Mecanismul său de control al congestiei este inspirat de TCP dar creat pentru a minimiza numărul cererilor de retransmitere al pachetelor.

Pachetele sunt transmise în blocuri. Un număr fix de pachete este grupat într-un bloc și transmis către destinație înainte ca oricare alt pachet din blocul următor să fie transmis. Retransmiterea pachetelor nu este permisă înainte ca fiecare pachet din blocul curent să nu se fi transmis măcar odată. În concluzie un bloc de pachete este transmis în mai multe etape: prima dată fiecare pachet este transmis odată, apoi pachetele care nu au fost primite la destinație, mai sunt trimise încăodată până când fiecare pachet din bloc va fi transmis și va avea confirmare de primire de la destinatar.

În cazul în care mecanismul ELFN este disponibil, TPA îl poate folosi și poate introduce timpii "morți" în cazul în care apare o rută căzută și se va descreşte lărgimea ferestrei la unu. Dacă ELFN nu este disponibil, atunci TPA va detecta rutele căzute prin numărul de timpi pauză consecutivi.

Pentru controlul congestiei, TPA foloseşte un mecanism fereastră cu o limită maximă a mărimii acesteia bine stabilită. De fapt, fereastra poate să aibă doar două valori: o fereastră "largă" de 2 sau 3 segmente -în timpul unei operații normale sau poate avea valoarea minimă, 1, când este detectată congestia.

În concluzie, TPA demonstrează faptul că și un simplu protocol care nu conține informații suplimentare de la nodurile intermediare poate mări transferal de date, în comparație cu TCP.

3.5.1 Protocoale alternative pentru AODV

EDAODV-Protocolul de detecțe timpurie a congestiei și de control al rutării:

EDAODV este un protocol de rutare pentru retelele MANET, în care ruta alternativă este gasită atât de nodul predecesor cât și de nodurile succesoare pe calea principală. În găsirea unei rute alternative, nodul anterior folosește o nouă rută și trimite pachetele congestionate către primul nod necongestionat.

EDAODV cuprinde 3 componente:

-Descoperirea rutelor: include descoperirea rutelor către destinație de către sursă prin trimiterea mesajelor RREQ. Destinația va trimite sursei un mesaj RREP. Fiecare nod are două tabele de rutare, o tabela de rutare primară și o tabelă de rutare alternativă.

-Detecția timpurie a congestiei: congestia poate aparea oricând în rețea. Aceasta apare din cauza trimiterii multor pachete către un nod, depășindu-se capacitatea acestuia. Acest lucru duce la congestia nodului și pierderea de pachete. Pentru a detecta congestia în avans, poate fi introdusă o metrică de congestie nodului afectat.

-Descoperirea bidirecționaală a căilor. Un nod congestionat poate să trimită în rețea pachete care să conțină starea de congestie a nodului. Pachetele CSP(congestion satus packet) vor conține starea congestiei și parametrii precum: sursa, destinația, nodul anterior și predecesor celui congestionat. Când pachetul va fi trimis în rețea se va cunoaște starea de congestie a nodului. Acest tip de informație este util pentru a găsi o cale alternativă, necongestionată pentru trimiterea pachetelor de date.

AODV-I este protocolul care îmbunătățeste protocolul de rutare AODV în scopul eliminării congestiei. Acesta se bazează pe trimiterea de mesaje RREQ care vor evita rutele congestionate automat în procesul de descoperire de rute noi. În AODV, daca sursa cere căutarea unei rute care are secvența de numere a destinației mai mare sau numărul de

noduri intermediare mai mic, noua rută o va înlocui pe prima și încărcarea primei rute va fi transferate celei noi. Și dacă noua rută este congestionată, traficul transmis pe ruta anterioară va face ca noua rută să fie și mai încărcată. Acest lucru duce la creșterea numărului de pachete pierdute

și a timpului de transmitere al pachetelor, implicit la reducerea performațelor rețelei. Dar, AODV-I îmbunătățește traditionalul AODV prin repararea rutei congestionate.

Protocolul de rutare adaptiv al congestiei(CRP) încearcă să prevină congestia din primul moment în care a apărut. Fiecare nod va preveni nodul anterior lui când poate să fie congestionat. CRP foloseste rutele alternative, numite rute ocolitoare, pentru a redirecționa traficul congestionat. Astfel, este redusă întârzierea în trimiterea pachetelor. Dar, în același timp, CRP încearcă să minimizeze rutele ocolitoare pentru a reduce supraîncărcarea rețelei.

Capitol 4. Analiză experimentală şi implementare

4.1 Platforma de dezvolatre-NS3

[10]Ns-3 a fost dezvoltat pentru a oferi o platformă extensibilă de simulare a rețelelor. Pe scurt, NS3 oferă posibilitatea de a observa cum pachetele de date sunt transmise într-o rețea și oferă un mecanism de simulare pentru a ajuta utilizatorii să își simuleze experimentele. Unul dintre motivele pentru care este folosit NS3 ar fi posibilitatea de a studia scenarii dificile sau imposibile de realizat pe un sistem real, de a observa comportamentul unui scenariu controlat și reproductibil și pentru a studia modul în care lucrează o rețea.

Caracteristicile NS3-ului, ce îl desebesc de alte medii de simulare ar fi:

-NS3 este proiectat ca un set de librări care pot fi combinate între ele, dar și cu alte librării externe. Ns3 poate folosi atât animații externe cât și modele pentru analiza datelor și pentru vizualizare. Moldul de lucru este în linie de comandă, folosind un mecanism de dezolvare reprezentat de C++ sau Pyton.

-ns3 rulează pe un sistem Linux, dar poate fi suportat și de Windows Visual Studio care este folosit pe Windows.

4.2 Instalarea simulatorului NS3

[9]Atât instalarea cât și rularea simulatorului se realizează din linia de comandă. Pentru acest lucru trebuie să intrăm în terminal(Alt+Ctrl+T-deschiderea terminalului).

Pașii ce trebuiesc urmați pentru instalare sunt:

a)Instalarea pachetelor folosite de NS3:

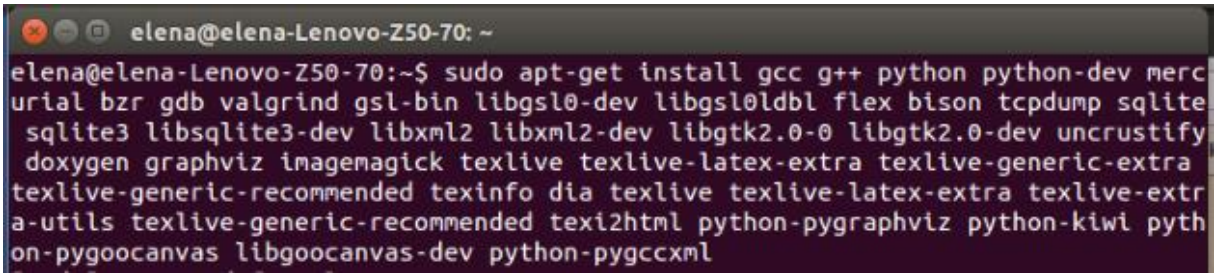


Figura 4.1 Instalarea pachetelor pentru NS3

b) Vom crea un director ns3v3.19 în care vom descărca simulatorul. Se va crea implicit, după descărcare, directorul ns3-allinone-3.19. Dacă îl vom lista(ls) vom găsi scripturile ce ne ajută la compilarea programului. Scripturile sunt scrise cu ajutorul limbajului Python:

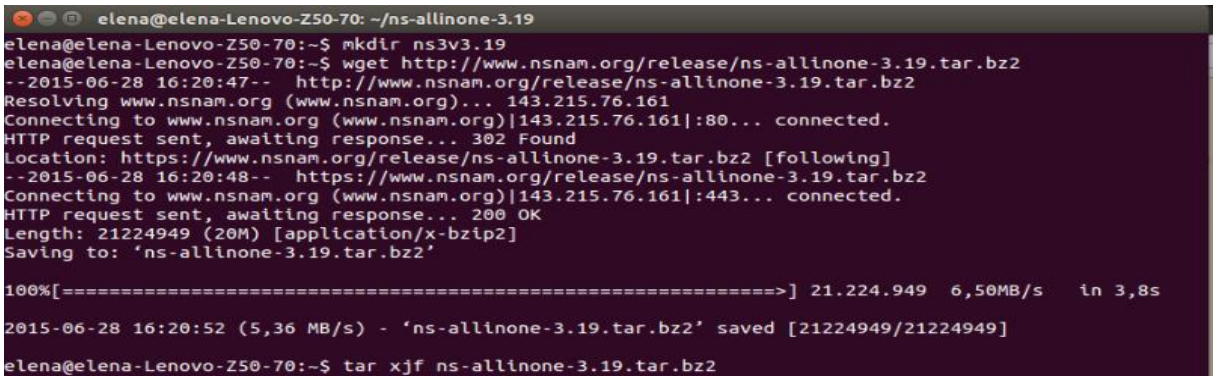


Figura 4.2 Descărcarea NS3-ului

```
elena@elena-Lenovo-Z50-70:~$ cd ns-allinone-3.19
elena@elena-Lenovo-Z50-70:~/ns-allinone-3.19$ ls
bake  build.py  constants.py  netanim-3.104  ns-3.19  pybindgen-0.16.0.834  README  util.py
```

Figura 4.3 Scripturile NS3-ului

c) Pentru a putea construirea exemplele oferite de NS3 va trebui să rulăm comanda:

```
elena@elena-Lenovo-Z50-70:~/ns-allinone-3.19$ ./build.py --enable-examples --enable-tests
# Build NetAnim
Entering directory `netanim-3.104'
 =>  qmake -v
QMake version 3.0
Using Qt version 5.2.1 in /usr/lib/x86_64-linux-gnu
qmake found
 =>  qmake NetAnim.pro
Project ERROR: Unknown module(s) in QT: svg
Error building NetAnim.
Skipping NetAnim ....
Leaving directory `netanim-3.104'
# Building examples (by user request)
# Building tests (by user request)
# Build NS-3
Entering directory `./ns-3.19'
 =>  /usr/bin/python waf configure --enable-examples --enable-tests --with-pybindgen ../pybindgen-0.16.
0.834
Setting top to                           : /home/elena/ns-allinone-3.19/ns-3.19
Setting out to                           : /home/elena/ns-allinone-3.19/ns-3.19/build
Checking for 'gcc' (c compiler)          : /usr/bin/gcc
Checking for cc version                  : 4.8.4
Checking for 'g++' (c++ compiler)        : /usr/bin/g++
Checking for compilation flag -Wl,--soname=foo... support : ok
Checking for program python                                : /usr/bin/python
Checking for python version                                : (2, 7, 6, 'final', 0)
Checking for library python2.7 in LIBDIR                   : yes
Checking for program /usr/bin/python-config,python2.7-config,python-config-2.7,python2.7m-config : /usr
/bin/python-config
Checking for header Python.h                                                                       : yes
Checking for compilation flag -fvisibility=hidden... support                                       : ok
```

Figura 4.4 Construirea exemplelor/testelor

Dacă rularea a fost efectuată cu succes vom primi mesajul:

```
[2232/2232] cxxprogram: build/src/tap-bridge/model/tap-creator.cc.4.o build/src/tap-bridge/model/tap-en
code-decode.cc.4.o -> build/src/tap-bridge/ns3.19-tap-creator-debug
Waf: Leaving directory `/home/elena/ns-allinone-3.19/ns-3.19/build'
'build' finished successfully (7m35.352s)

Modules built:
antenna                  aodv                     applications
bridge                   buildings                config-store
core                     csma                     csma-layout
dsdv                     dsr                      emu
energy                   fd-net-device            flow-monitor
internet                 lte                      mesh
mobility                 mpi                      netanim (no Python)
network                  nix-vector-routing       olsr
point-to-point           point-to-point-layout    propagation
sixlowpan                spectrum                 stats
tap-bridge               test (no Python)         topology-read
uan                      virtual-net-device       visualizer
wave                     wifi                     wimax
```

Figura 4.5 Mesajul rulării cu succes

d) Configurarea scriptul folosit de noi (waf) pentru configurarea exemplelor/testelor:

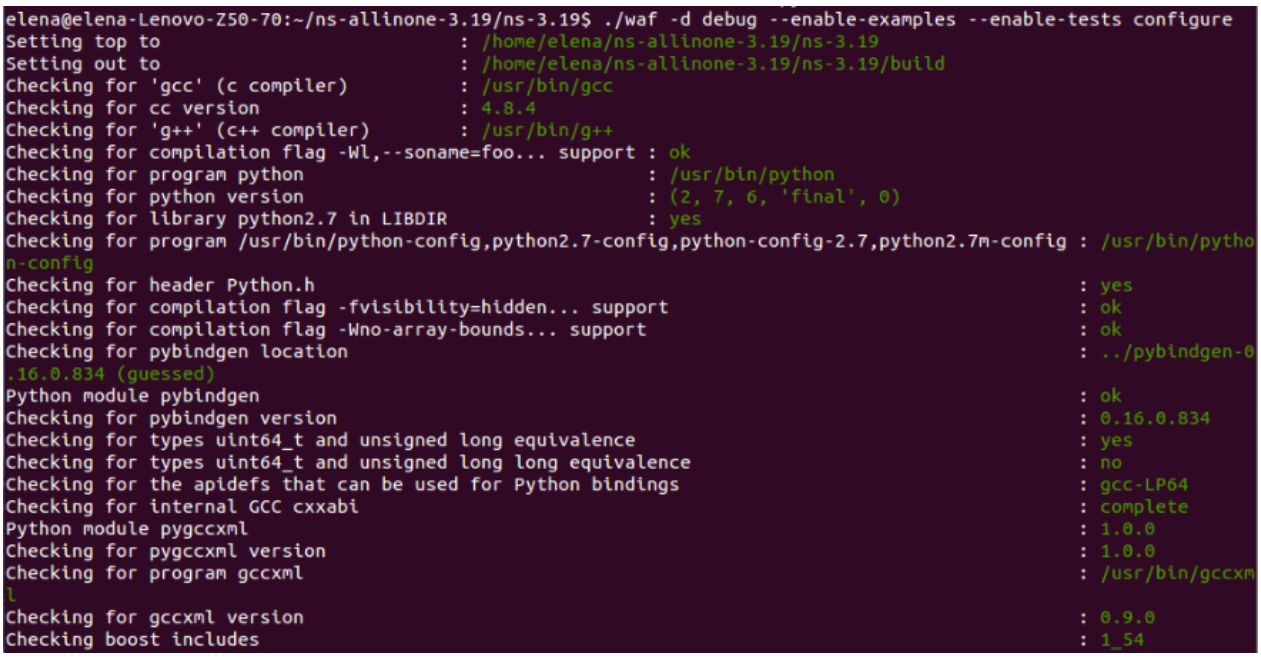


Figura 4.6 Configurarea exemplelor/testelor

Figura 4.7 Configurare cu cu succes

e) Pentru a testa dacă simulatorul a fost instalat cu succes vom rula scriptul ./test.py:

Figura 4.8 Testarea simulatorului

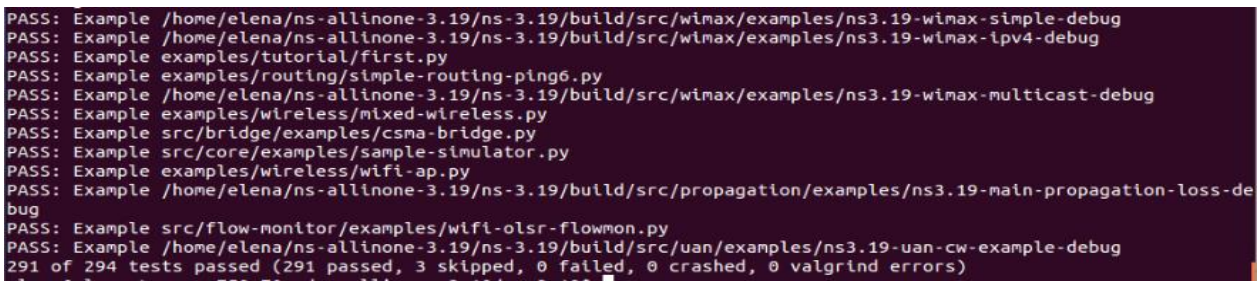


Figura 4.9 Testare cu succes

4.3 Scenariul simulării

Prima data voi simula o rețea formată din 10 noduri. Nodurile sunt legate între ele și sunt plasate aleator în rețea. Pentru fiecare simulare în parte voi varia numărul de pachete trimise in rețea de la 100 până la 900.

În cel de-al doilea scenariu voi varia numărul de noduri si voi trimise un număr constant de pachete:100.

La un moment dat, un nod va ceda. Pachetele ce cunoșteau ruta către destinație prin intermediul acelui nod vor fi rerutate. Pachetele vor fi trimise în rețea de la fiecare nod în parte, la celelalte.

4.4 Topologia rețelei.

Scopul acestor topologii este de a pune în evidență modul în care protocolul AODV funcționează în condiții de congestie.

Parametrii generali:

Numărul de noduri mobile	10,20,30,40,50,60,70,80,90,100
Protocolul de rutare	AODV
Dimensiunea retelei(X)	200
Dimensiunea retelei(Y)	200
Dimensiunea retelei(Z)	200
Durata simulării	100s
Viteza nodurilor	aleatoare
Număr pachete	1p/s(100 pachete)

Tabelul 4.1 Topologia rețelei1

Parametrii generali:

Numărul de noduri mobile	10
Protocolul de rutare	AODV
Dimensiunea retelei(X)	200
Dimensiunea retelei(Y)	200
Dimensiunea retelei(Z)	200
Durata simulării	100s
Viteza nodurilor	aleatoare
Număr pachete	100,200,300,400,500,600,700,800,900

Tabel 4.2 Topologia rețelei 2

Pentru a putea explica mai bine mecanismul de funcționare al protocolului voi discuta numai pentru cele 10 noduri și 100 de pachete trimise. Celelalte le-am folosit pentru simularea protocolului și extragerea rezultatelor.

Vom considera o rețea formata din 10 noduri. Nodurile sunt poziționate aleator în interiorul unei „cutii". Inițial nodurile se vor mișca pe direcții aleatoare, în interirorul cutiei, cu viteze diferite. Pentru a putea realiza simularea, nodurile se vor opri. Toate nodurile au legături între ele.

Algoritmul presupune folosirea corectă a mesajelor de tip RREQ, RREP precum și RERR pentru trimiterea pachetelor la destinațiile dorite.

Pentru a putea simula și mai bine congestia, la un moment nodul 5 va ceda, astfel toate pachetele ce trebuiau trimise către el și prin intermediul lui sunt rerutate și trimise pe altă cale în rețea. De asemenea, toate rutele care aveau legătură cu nodul 5 vor fi șterse din tabela de rutare.

Pentru început pentru a avea o viziune mai bună a rețelei, de asamblu, vom afișa tabela de rutare:

```
Node: 0 Time: 8s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.2        10.0.0.2        10.0.0.1        UP      2.01            1
10.0.0.3        10.0.0.3        10.0.0.1        UP      1.01            1
10.0.0.5        10.0.0.5        10.0.0.1        UP      0.01            1
10.0.0.6        10.0.0.6        10.0.0.1        DOWN    23.01           1
10.0.0.7        10.0.0.7        10.0.0.1        UP      0.02            1
10.0.0.8        10.0.0.8        10.0.0.1        UP      2.01            1
10.0.0.9        10.0.0.9        10.0.0.1        UP      2.01            1
10.0.0.10       10.0.0.10       10.0.0.1        UP      3.22            1


Node: 1 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.2        UP      2.08            1
10.0.0.3        10.0.0.3        10.0.0.2        UP      2.01            1
10.0.0.4        10.0.0.4        10.0.0.2        UP      2.01            1


Node: 2 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.3        UP      2.08            1
10.0.0.2        10.0.0.2        10.0.0.3        UP      2.01            1
10.0.0.4        10.0.0.4        10.0.0.3        UP      2.01            1
10.0.0.5        10.0.0.5        10.0.0.3        UP      2.01            1


Node: 3 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.5        10.0.0.4        DOWN    23.01           2
10.0.0.2        10.0.0.2        10.0.0.4        UP      2.01            1
10.0.0.3        10.0.0.3        10.0.0.4        UP      2.01            1
10.0.0.5        10.0.0.5        10.0.0.4        UP      2.01            1


Node: 4 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.5        UP      2.08            1
10.0.0.3        10.0.0.3        10.0.0.5        UP      2.01            1
10.0.0.4        10.0.0.4        10.0.0.5        UP      2.01            1
```

Figura 4.10 Tabela de rutare(noduri 0-4)

```
Node: 5 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.6        DOWN    23.02           1
10.0.0.7        10.0.0.7        10.0.0.6        DOWN    23.02           1
10.0.0.8        10.0.0.8        10.0.0.6        DOWN    23.02           1
10.0.0.9        10.0.0.9        10.0.0.6        DOWN    23.02           1
10.0.0.10       10.0.0.10       10.0.0.6        DOWN    23.02           1
10.255.255.255  10.255.255.255  10.0.0.6        UP      9223372028.85   1
127.0.0.1       127.0.0.1       127.0.0.1       UP      9223372028.85   1

Node: 6 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.7        UP      2.08            1
10.0.0.6        10.0.0.6        10.0.0.7        DOWN    23.02           1
10.0.0.8        10.0.0.8        10.0.0.7        UP      2.01            1
10.0.0.9        10.0.0.9        10.0.0.7        UP      2.01            1
10.0.0.10       10.0.0.10       10.0.0.7        UP      2.09            1


Node: 7 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.8        UP      2.08            1
10.0.0.6        10.0.0.6        10.0.0.8        DOWN    23.01           1
10.0.0.7        10.0.0.7        10.0.0.8        UP      2.01            1
10.0.0.9        10.0.0.9        10.0.0.8        UP      2.01            1
10.0.0.10       10.0.0.10       10.0.0.8        UP      2.09            1


Node: 8 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.9        UP      2.08            1
10.0.0.6        10.0.0.6        10.0.0.9        DOWN    23.02           1
10.0.0.7        10.0.0.7        10.0.0.9        UP      2.01            1
10.0.0.8        10.0.0.8        10.0.0.9        UP      2.01            1
10.0.0.10       10.0.0.10       10.0.0.9        UP      2.09            1


Node: 9 Time: 8.00s
AODV Routing table
Destination     Gateway         Interface       Flag    Expire          Hops
10.0.0.1        10.0.0.1        10.0.0.10       UP      2.08            1
10.0.0.6        10.0.0.6        10.0.0.10       DOWN    23.02           1
10.0.0.7        10.0.0.7        10.0.0.10       UP      2.01            1
10.0.0.8        10.0.0.8        10.0.0.10       UP      2.01            1
10.0.0.9        10.0.0.9        10.0.0.10       UP      2.01            1
```

Figura 4.11 Tabela de rutare(noduri 5-9)

Din tabela de rutare putem să extragem următoarele informații: nodurile din rețea împreună cu vecinii lor, numărul de noduri intermediare dintre sursă și destinație,timpul de expirare al rutelor, statusul nodurilor precum și adresa interfeței nodurilor.

Tabela de rutare ne prezintă evoluția nodurilor dupa ce nodul 5 nu mai este funcțional.

De exemplu: „Node 0” are ip-ul interfeței 10.0.0.1 și drept vecini toate celelalte 9 noduri. De asemenea, putem să adăugăm și faptul că toate legăturile cu celelalte noduri sunt bune pentru trimiterea de pachete, cu expceția nodului 5. Nodul 5, ce are adresa 10.0.0.6, am spus, mai devreme că îl vom considera nefuncțional. Statusul fanioanelor este influențat direct de către timpul de expirare. Dacă legătura dintre noduri s-a stabilit în mai puțin de 8 secunde, însemnă că ruta este „UP”, dacă timpul este mai mare de 8 secunde, vom avea o legătură „DOWN”.

Dupa ce am stabilit legătura dintre noduri, voi trimite un mesaj de tip RREQ. Acesta va strabate rețeaua de la sursă la destinație.
În cazul în care mesajul a ajuns la un nod destinație , nodul v-a trimite un mesaj de tip RREP înapoi către sursă.
În cazul în care, nodul 5 a disparut din rețea iar mesajul de tip RREQ a fost trimis pe ruta ce îl conținea,se va trimite un mesaj de tip RERR către sursă pentru a anunța ca ruta nu mai este actuală. Odată primit acest mesajul, ruta va fi ștearsă din tabela de rutare.

După ce tabela de rutare a fost actualizată se va începe un nou proces de căutare a rutelor către destinație pentru a putea trimite pachetul la nodul dorit.

4.5 Implementarea protocolului

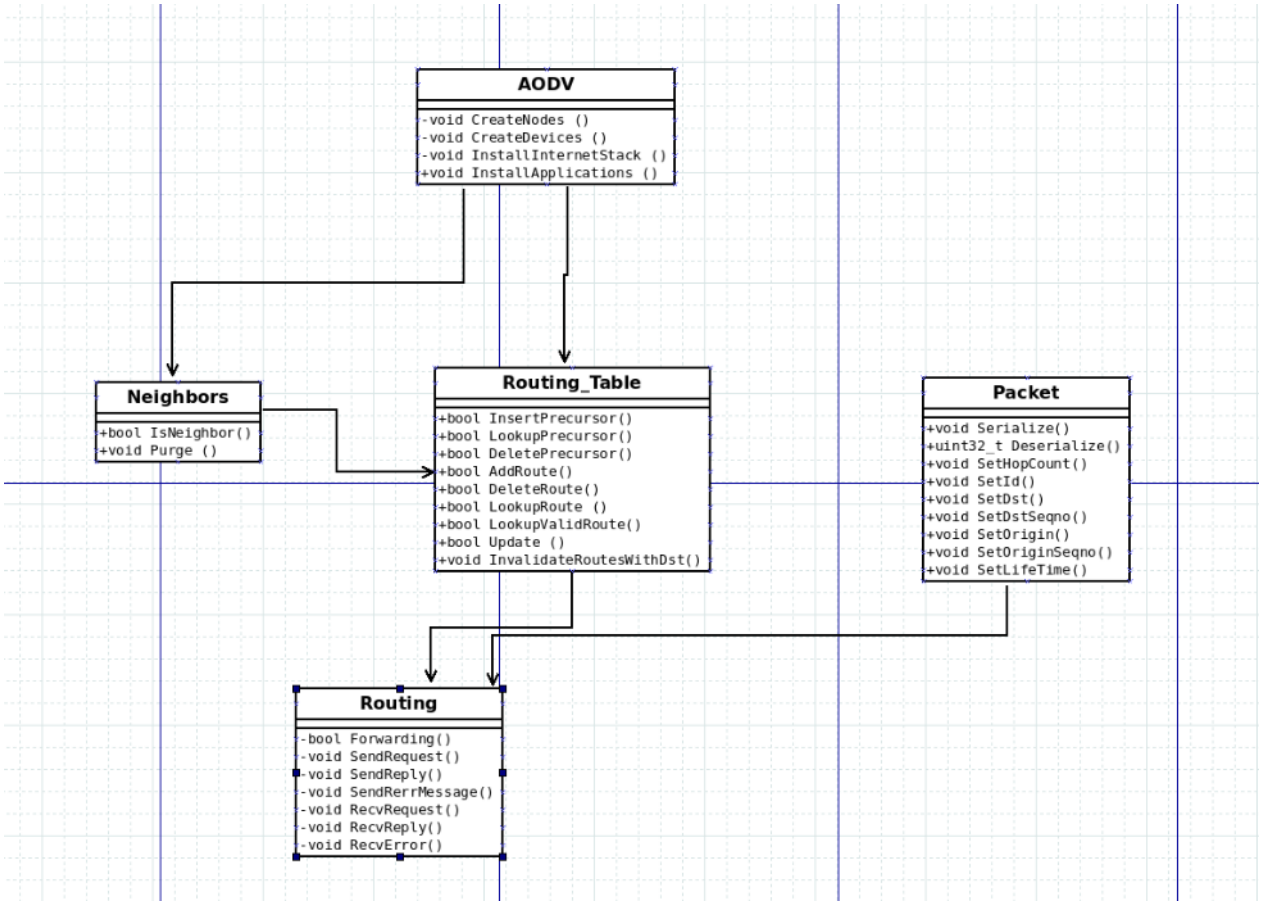


Figura 4.12 Diagrama claselor

Explicarea claselor:[10]

AODV

Programul principal este AODV, acesta realizează, cu ajutorul funcțiilor aferente: `void CreateNodes(); void CreateDevices();` `void InstallInternetStack(); void InstallApplications()`, crearea nodurilor și realizarea legăturilor dintre ele.

Neighbors

Clasa `Neighbors` ajută la stabilirea vecinilor unui nod.

`bool IsNeighbor (Ipv4Address addr)` verifică dacă nodul următor celui curent îi este vecin. Luând în considerare că știm toate adresele nodurilor, comparăm adresa predecesorului nodului cu adresa „`addr`" . Dacă este returnată valoarea de adevăr `true` însemnă că nodul îi este vecin, altfel nu sunt noduri vecine.

În cazul în care nu au fost găsite noduri cu adresele "`addr`", funcția `purge()` va elimina aceste intrări invalide.

Routing_Table

Clasa `Routing_Table` oferă posibilitatea de a creea și actualiza tabela de rutare. Acest lucru se relizează atât la nivel de nod și implicit la nivel de rețea.

Căutarea și inserarea se realizează cu urmatoarele două funcții:

```
bool RoutingTableEntry::LookupPrecursor (Ipv4Address id);

bool RoutingTableEntry::InsertPrecursor (Ipv4Address id);
```

Iar ștergerea unui nod predecesor celui curent se execută cu funcția:

```
bool RoutingTableEntry::DeletePrecursor (Ipv4Address id);
```

`bool AddRoute (RoutingTableEntry & r)` va adauga o rută în tabela de rutare, daca nu este adaugată încă, iar `bool DeleteRoute (Ipv4Address dst)` va șterge ruta către destinația "`dst`", daca ea există.

`bool LookupRoute (Ipv4Address dst, RoutingTableEntry & rt)` va căuta o rută către destinație în schimb, `bool LookupValidRoute (Ipv4Address dst, RoutingTableEntry & rt)` realizează căutarea unei rute valide.

Funcția `bool Update (RoutingTableEntry & rt)` realizează actualizarea tabelei de rutare privind starea rutelor, în schimb `void InvalidateRoutesWithDst (std::map<Ipv4Address, uint32_t> const & unreachable)` schimbă în tabela de rutare doar rutele invalide.

Packet

Clasa `Packet` este formată din funcțiile `void RreqHeader::Serialize ()` si `uint32_t RreqHeader::Deserialize()` ce se ocupă cu scrierea și citirea mesajelor de tip RREQ,RREP și RRER. De asemenea, celelalte funcții aferente clasei au ca scop actulizarea antetelor mesajelor.

Routing

Clasa `Routing` folosește tabela de rutare și mesajele pentru a putea transfera date în rețea.
Funcția `void SendRequest (Ipv4Address dst)` trimite mesajul de tip RREQ către destinație, iar `void RecvRequest (Ptr<Packet> p, Ipv4Address receiver, Ipv4Address src)` sugerează primirea mesajului.

`void SendReply (RreqHeader const & rreqHeader, RoutingTableEntry const & toOrigin)` va trimite mesajul de tip RREP, iar `void RecvReply (Ptr<Packet> p, Ipv4Address my,Ipv4Address src)` va specifica primirea lui.

`void SendRerrMessage (Ptr<Packet> packet, std::vector<Ipv4Address> precursors)` trimite mesajul de tip RERR, iar void `RecvError (Ptr<Packet> p, Ipv4Address src)` confirmă primirea lui de către nodul sursă.

Atunci când este gasită o rută validă, pachetul este trimis cu ajutorul funcției `bool Forwarding (Ptr<const Packet> p, const Ipv4Header & header, UnicastForwardCallback ucb, ErrorCallback ecb);.`

4.6 Implementarea grafică

Implementarea grafică am realizat-o prin adaugarea modulului `#include "ns3/netanim-module.h".`

Mai mult decât atât, am impus crearea unui fisier de tip .xml pentru a putea fi rulat de programul NetAnim.
Pentru a avea nodurile pozitionate în rețeaua noastră grafică, am prestabilit pentru acestea coordonate aleatoare.

```
std::string animFile = "aodvanim.xml";
AnimationInterface anim(animFile);
anim.SetConstantPosition(nodes.Get(0), 35, 55, 50);
anim.SetConstantPosition(nodes.Get(1), 50, 150, 90);
anim.SetConstantPosition(nodes.Get(2), 100, 100, 50);
anim.SetConstantPosition(nodes.Get(3), 150, 15, 70);
anim.SetConstantPosition(nodes.Get(4), 100, 100, 105);
anim.SetConstantPosition(nodes.Get(5), 130, 15, 10);
anim.SetConstantPosition(nodes.Get(6), 50, 170, 50);
anim.SetConstantPosition(nodes.Get(7), 120, 70, 90);
anim.SetConstantPosition(nodes.Get(8), 130, 90, 50);
anim.SetConstantPosition(nodes.Get(9), 113, 162, 100);
```

Pentru pornirea programului ce ajută la realizarea părții grafice, se rulează scriptul ./NetAnim:

```
elena@elena-Lenovo-Z50-70:~/ns3/ns-allinone-3.19$ cd netanim-3.104
elena@elena-Lenovo-Z50-70:~/ns3/ns-allinone-3.19/netanim-3.104$ ./NetAnim
```

Figura 4.13 Pornirea NetAnim-ului

În urma simulării am putut oberva cum se trimit mesajele între noduri:

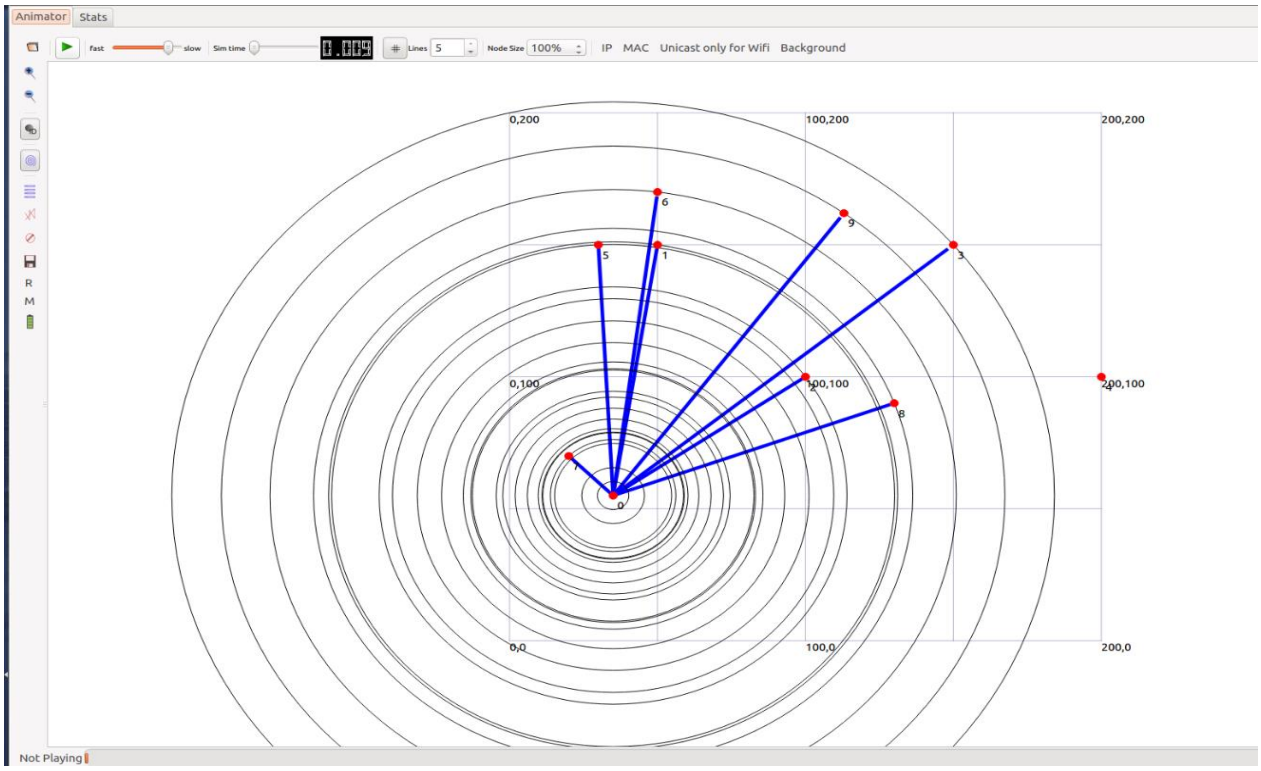


Figura 4.14 Trimiterea de mesaje de la nodul 0

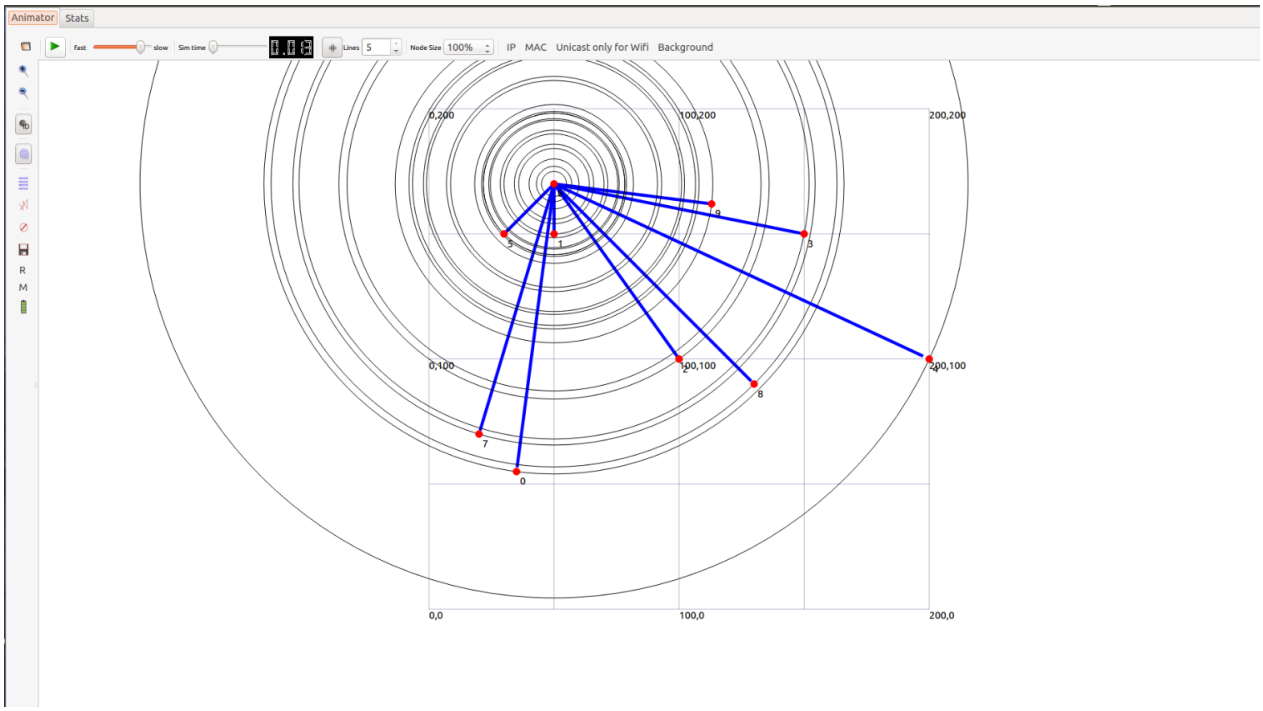


Figura 4.15 Trimiterea mesajelor de la nodul 6

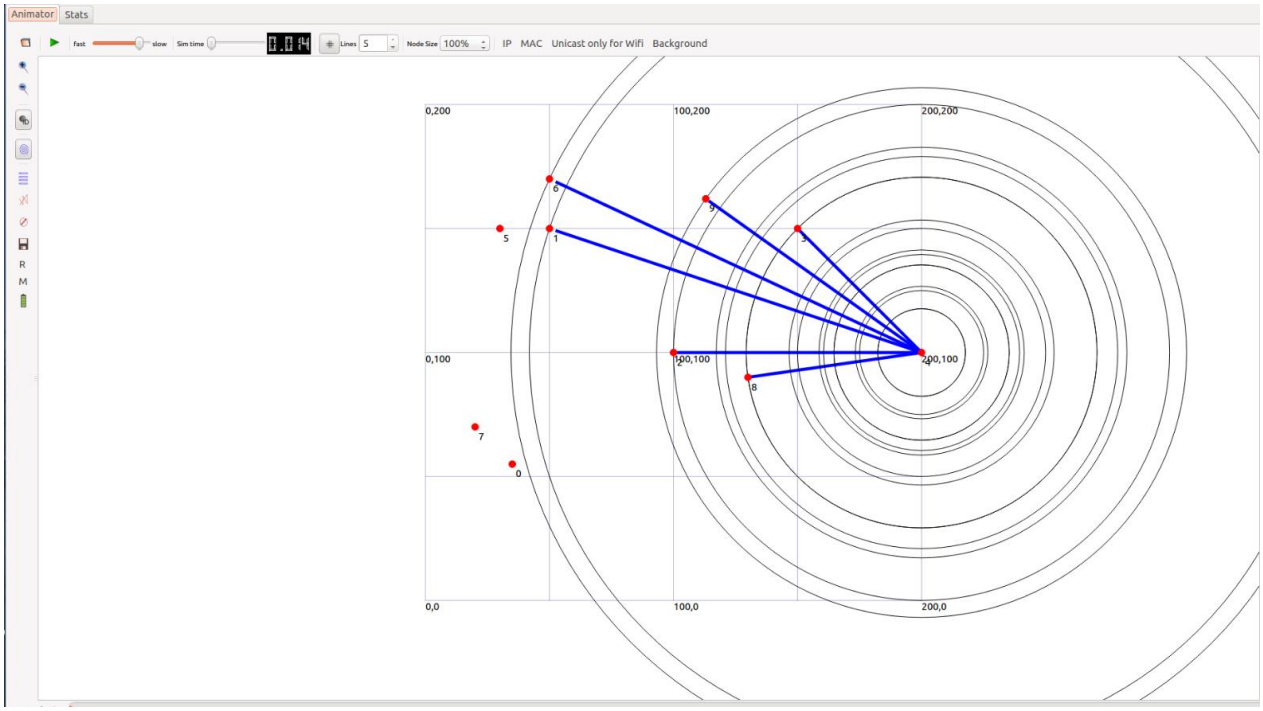


Figura 4.16 Trimiterea mesajelor de la nodul 4

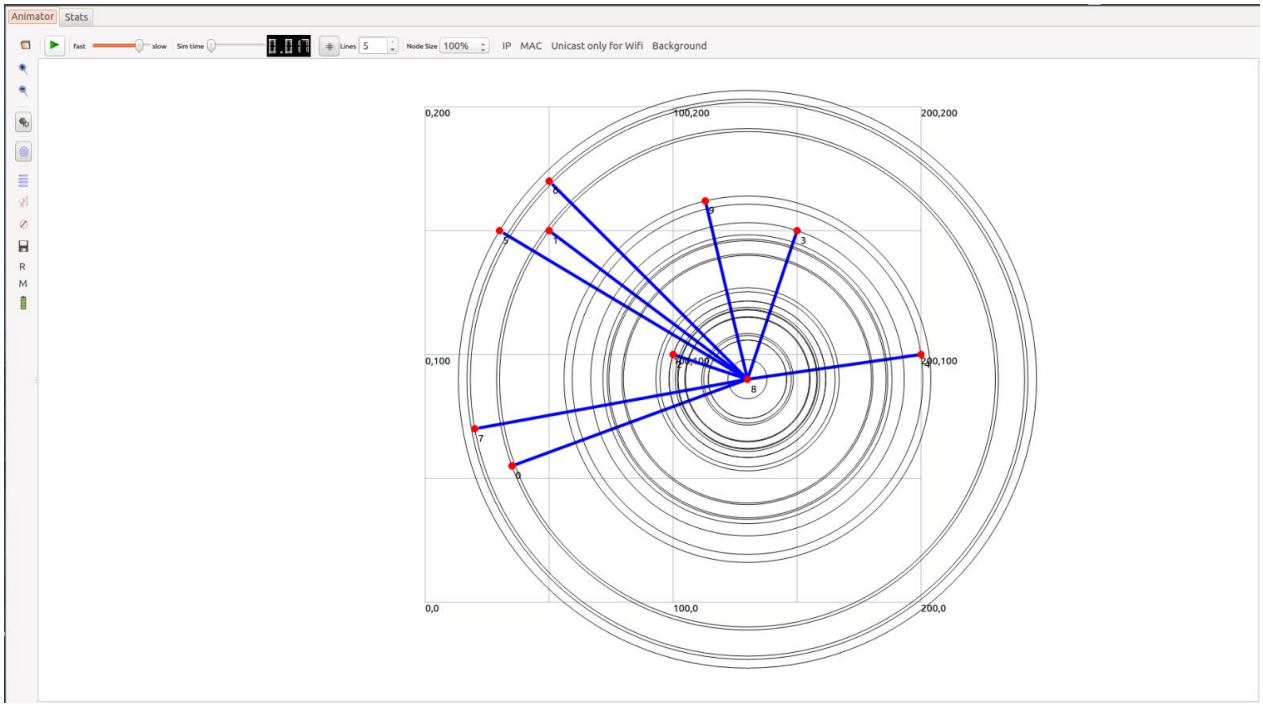


Figura 4.17 Trimiterea mesajelor de la nodul 8

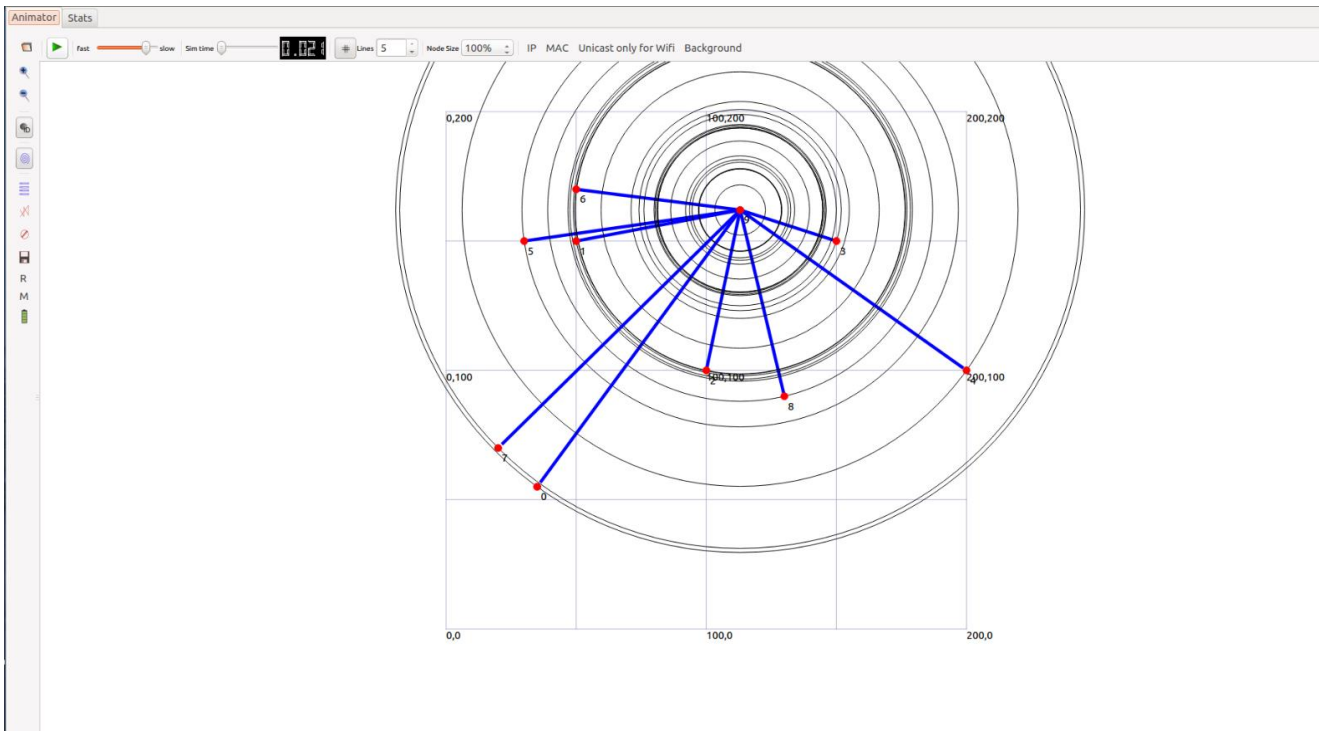


Figura 4.18 Trimiterea mesajelor de la nodul 9

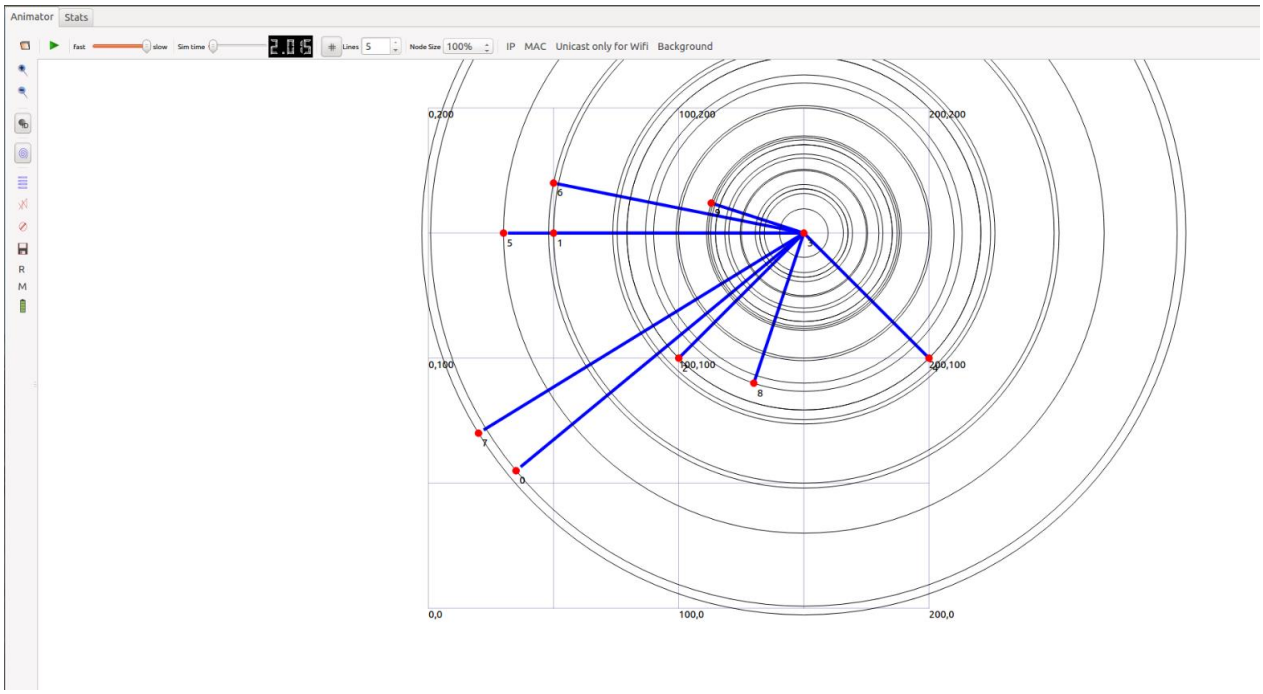


Figura 4.19 Trimiterea mesajelor de la nodul 3

4.7 Rezultate experimentale

Topologia 1

După cum știm congestia apare odata cu creșterea numărului de pachete din rețea. Reprezentarea grafică ne oferă o viziune asupra numărului de pachet pierdute în urma creșterii numărului de pachete de date trimise în rețea.

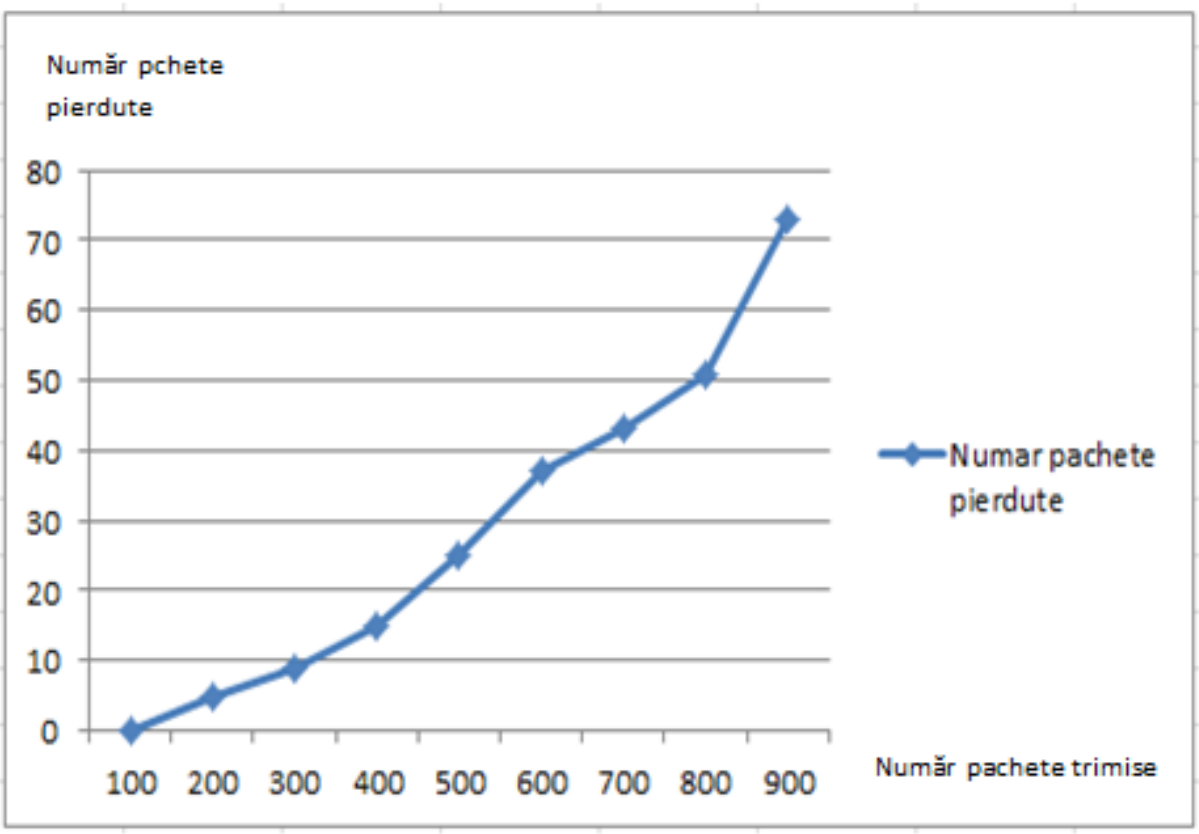


Figura 4.20 Pierderea de pachete în rețea

Apariția congestiei și a pierderilor în rețea se explică prin creșterea numărului de cereri pentru descoperirea rutelor către destinație. Un număr de pachete mare determină trimiterea mai multe mesaje de tip RREQ, RREP și RERR.

Acest lucru influețează şi timpul în care pachetele ajung la destinație, realizându-se întarzieri în rețea:

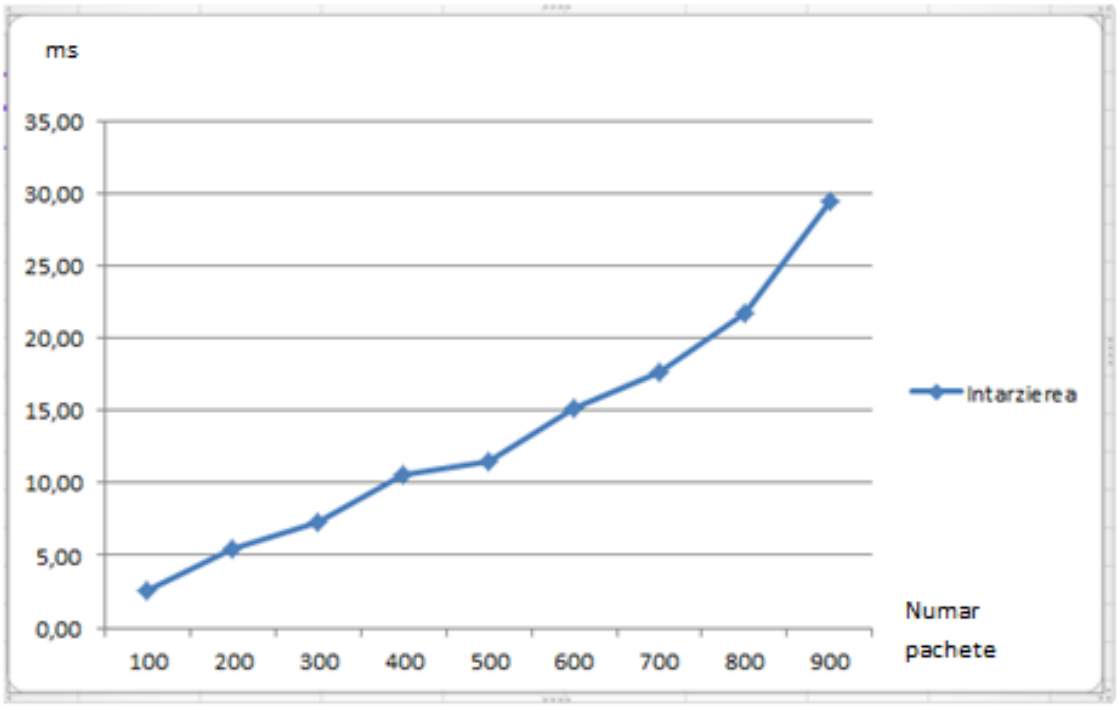


Figura 4.21 Întârzierea pachetelor

Tolopologia 2

În urma celei de-a doua simulari, am observat că și variația numărului de noduri duce la pierderea de pachete în rețea. Acest lucru ar putea fi explicat prin faptul că de fiecare dată când trebuie trimis un pachet în rețea trebuie trimis și un mesaj RREQ pentru determinarea rutelor. Cu cât numărul de noduri crește, cu atât crește și numărul de mesaje, apărând rute mai multe către destinație. Acest lucru duce la încărcarea rețelei.

Dacă, inițial, la 10 noduri, toate pachetele au fost trimise cu succes, când avem în rețea 100 de nodurile apar și pierderi de pachete.

Acest lucru este arătat în diagrama:

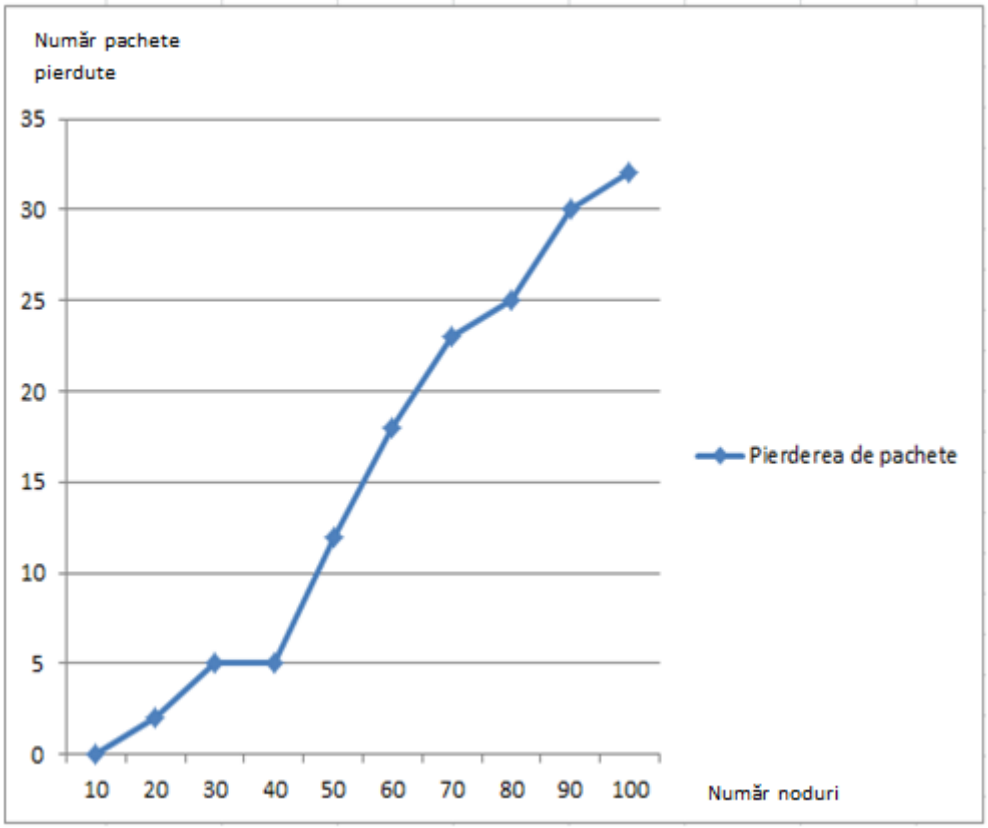


Figura 4.22 Pierderea de pachete in rețea

4.8 Concluzii

După cum știm, rețele de tip Ad-Hoc sunt folosite cu precădere pentru comunicațiile pe câpurile de luptă sau în urma unui dezastru. În aceste situații critice rețeaua trebuie să funcționeze la standele ridicate.

Prin intermediul simulărilor mi-am dorit să pun în evindență modul în care protocolul AODV transmite mesaje și mai ales cum se comportă când apare congestia în rețea.

O caracteristică importantă a protocolului este reprezentată de trimiterea de mesaje pentru verificarea rețelei de fiecare dată când este voie să se trimită un pachet în rețea.

În urma simulărilor am observat că acest lucru va constitui un dezavantaj pentru ca odată ce numărul de pachete trimise crește, va crește și numărul de mesaje trimise ducând la congestionarea rețelei.

Se pare că acest protocol este bun pentru rețelele de mici dimensiuni, problemele apărând și atunci cand numărul de noduri din rețea crește.

Un viitor subiect de studiu ar putea fi constituit din încercările de îmbunătățire a acestui protocol de rutare în așa fel încât absolut toate pachetele din rețea să fie rerutate cu succes către destinația dorită.

Bibliografie

[1]http://download-v2.springer.com/static/pdf/635/bok%253A978-0-387-22690-3.pdf?token2=exp=1431642423~acl=%2Fstatic%2Fpdf%2F635%2Fbok%25253A978-0-387-22690-3.pdf*~hmac=9242ea82a51841eb86c455da5ecaaa7229060cd4216b929aaecc685955edf4ae

Accesat la data de : 10.05.2015

[2]http://stst.elia.pub.ro/news/RC/Computer%20Networks%20-%20A%20Tanenbaum%20-%205th%20edition.pdf

Accesat la data de: 10.05.2015

[3] http://pdos.csail.mit.edu/decouto/papers/zhu02.pdf

Accesat la data de : 06.06.2015

[4]http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1020231&tag=1

Accesat la data: 12.06.2015

[5] https://www.ietf.org/rfc/rfc3561.txt

Acesat la data de: 13.06.2015

[6] https://www.ietf.org/rfc/rfc4728.txt

Acesat la data de: 13.06.2015

[7] Cross-Layer Aided Energy-Efficient Routing Design for Ad Hoc Networks

Autori:Jing Zuo, Chen Dong, Soon Xin Ng, Lie-Liang Yang, Lajos Hanzo

An publicație:2015

[8] A Survey on Congestion Control for Mobile Ad-Hoc Networks

[9] http://ns3help.blogspot.ro/2014/05/installing-ns3-in-ubuntu-1210.html

Accesat la data de 23.04.2015

[10]https://www.nsnam.org

Accesat la data de: 02.05.2015

Anexa [10]

Aodv.cc

```
#include "ns3/aodv-module.h"
#include "ns3/core-module.h"

#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/mobility-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/wifi-module.h"
#include "ns3/v4ping-helper.h"
#include "ns3/netanim-module.h"
#include <iostream>
#include <cmath>

using namespace ns3;
class AodvExample
{
public:
  AodvExample ();
bool Configure (int argc, char **argv);



  void Run ();
  void Report (std::ostream & os);


private:
  uint32_t size;
  double step;
  double totalTime;
  bool printRoutes;
  NodeContainer nodes;
  NetDeviceContainer devices;

 Ipv4InterfaceContainer interfaces;



private:
  void CreateNodes ();
  void CreateDevices ();
  void InstallInternetStack ();
  void InstallApplications ();
};

int main (int argc, char **argv)
{
  AodvExample test;
  if (!test.Configure (argc, argv))
```

Pentru a avea cât mai puține fșsiere declarare s-a preferat împărțirea acestora în module.

```
aodv-module.h :
#include "aodv-dpd.h"
#include "aodv-helper.h"
#include "aodv-id-cache.h"
#include "aodv-neighbor.h"
#include "aodv-packet.h"
#include "aodv-routing-
protocol.h"
#include "aodv-rqueue.h"
#include "aodv-rtable.h"
```

→declararea functiilor publice

→configurarea parametrilor, returneaza“true”daca configurarea este buna

→ rularea simularii
→ afiseaza reultatelor

→declararea variabilelor private
→ numărul de noduri
→ distanța dintre noduri
→ timpul simulării, în secunde
→ folosit pentru printarea rutelor
→ nodurile sunt de tipul NodeContainer
→ dispozitivele sunt de tip NetDevicesContainer
→ interfetele sunt de tip Ipv4InterfaceContainer

→declararea funcțiilor private

```
    NS_FATAL_ERROR ("Configuration failed. Aborted.");

  test.Run ();
  test.Report (std::cout);
 return 0;
}
AodvExample::AodvExample () :
  size (100),
  step (100),
  totalTime (100),
  printRoutes (true)
{
```

bool AodvExample::Configure ne ajută să introducem parametrii necesari programului din linie de comandă: numărul de noduri, timpul simulării, distanța dintre noduri.

```
}
bool
AodvExample::Configure (int argc, char **argv)
{

  CommandLine cmd;
  cmd.AddValue ("printRoutes",
"Print routing table dumps.",printRoutes);
  cmd.AddValue ("size",
 "Number of nodes.", size);
  cmd.AddValue ("time",
 "Simulation time, s.", totalTime);
  cmd.AddValue ("step",
"Grid step, m", step);

  cmd.Parse (argc, argv);
  return true;
}
void
      AodvExample::Run ()
```

void AodvExample::Run () este apelată simularea dar și eliminarea ei odata ce s-a terminat. Tot în aceeastă funcție am realizat partea grafică a programului prin AnimationInterface anim(animFile); și am introdus coordonatele nodurilor în rețea.

```
{
  CreateNodes ();
  CreateDevices ();
  InstallInternetStack ();
  InstallApplications ();

std::cout << "Starting simulation for " << totalTime << " s ...\n";
std::string animFile = "aodvanim.xml";

Simulator::Stop (Seconds (totalTime));

AnimationInterface anim(animFile);
anim.SetConstantPosition(nodes.Get(0), 35, 55, 50);
anim.SetConstantPosition(nodes.Get(1), 50, 150, 90);
```

```
anim.SetConstantPosition(nodes.Get(2), 100, 100, 50);
anim.SetConstantPosition(nodes.Get(3), 150, 15, 70);
anim.SetConstantPosition(nodes.Get(4), 100, 100, 105);
anim.SetConstantPosition(nodes.Get(5), 130, 15, 10);
anim.SetConstantPosition(nodes.Get(6), 50, 170, 50);
anim.SetConstantPosition(nodes.Get(7), 120, 70, 90);
anim.SetConstantPosition(nodes.Get(8), 130, 90, 50);
anim.SetConstantPosition(nodes.Get(9), 113, 162, 100);

  Simulator::Run ();
  Simulator::Destroy ();
}
```

```
void
AodvExample::CreateNodes ()
```

void AodvExample::CreateNodes ()

creează nodurile și stabilește poziționarea acestora. Eu am ales ca nodurile să fie poziționate haotic ca într-o cutie 3D. După aranjarea nodurilor am hotărât ca ele să se miște aleator , cu o viteze diferite în interorul cutiei. Pentru a realiza simularea ne dorim ca nodurile să fie într-o poziție fixă, asa că le vom opri.

```
{
std::cout << "Creating "
<< (unsigned)size << " nodes "
<< step << " m apart.\n";
nodes.Create (size);
for (uint32_t i = 0; i < size; ++i)
{
std::ostringstream os;
os << "node-" << i;
Names::Add (os.str (), nodes.Get (i));
    }
MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::RandomBoxPositionAllocator",
                                 "X",                          StringValue
("ns3::UniformRandomVariable[Min=0.0|Max=1.0]"),
                                 "Y",                          StringValue
("ns3::UniformRandomVariable[Min=0.0|Max=1.0]"),
                                 "Z",                          StringValue
("ns3::UniformRandomVariable[Min=0.0|Max=1.0]"));
  mobility.SetMobilityModel   ("ns3::RandomWalk2dMobilityModel","Bounds",
RectangleValue (Rectangle (-50, 50, -50, 50)));
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
 mobility.Install (nodes);
}
void
AodvExample::CreateDevices ()
{
  NqosWifiMacHelper wifiMac = NqosWifiMacHelper::Default ();
 wifiMac.SetType ("ns3::AdhocWifiMac");
  YansWifiPhyHelper wifiPhy = YansWifiPhyHelper::Default ();
  YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
  wifiPhy.SetChannel (wifiChannel.Create ());
  WifiHelper wifi = WifiHelper::Default ();
```

void AodvExample::CreateDevices ne ajută la configurarea legăturilor dintre noduri.
În cod vom folosi configurarea nivelului fizic "default" și modelele canalelor de legatură care se gasesc în clasele YansWifiPhyHelper și YansWifiChannelHelper. Odată ce obiectele au fost create, vom avea un canal de legatură și îl vom asocia nivelului nostru fizic. Astfel ne asigurăm că legaturile dintre noduri sunt la fel și că pot comunica.
În ceea ce privește configurarea nivelului MAC, atributul "ns3::AdhocWifiMac" ne specifică faptul că va fi folosită o rețea wifi de tip ad-hoc.
În final, vor fi instalate caracteristicile pentru nodurile noastre.

```
 wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
 "DataMode", StringValue ("OfdmRate6Mbps"),
"RtsCtsThreshold", UintegerValue (0));
  devices = wifi.Install (wifiPhy, wifiMac, nodes);
}
void
 AodvExample::InstallInternetStack ()
```

Până acum putem spune că am creat nodurile, mobilitatea nodurilor și legăturile dintre ele. Acum vom realiza stiva de protocoale. Functia InternetStackHelper va implementa stiva iar Ipv4AddressHelper va asigna IP-uri interfetelor dispozitivelor noastre. Ip-urile vor începe de la 10.0.0.0 pentru primul nod și vor ajunge la 10.0.0.10 pentru al 10-lea nod.

```
{
  AodvHelper aodv;
  InternetStackHelper stack;
  stack.SetRoutingHelper (aodv);
  stack.Install (nodes);
  Ipv4AddressHelper address;
  address.SetBase ("10.0.0.0", "255.0.0.0");
  interfaces = address.Assign (devices);

  if (printRoutes)
    {
      Ptr<OutputStreamWrapper>routingStream = Create<OutputStreamWrapper>
("aodv.routes2", std::ios::out);
      aodv.PrintRoutingTableAllAt (Seconds (8), routingStream);
    }
}

void
AodvExample::InstallApplications ()
{
  V4PingHelper ping (interfaces.GetAddress (size-1));
  ping.SetAttribute ("Verbose", BooleanValue (true));
  ApplicationContainer p = ping.Install (nodes.Get (0));
```

```
  p.Start (Seconds (0));
  p.Stop (Seconds (totalTime) - Seconds (0.001));

Ptr<Node> node = nodes.Get (size/2);
  Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
  Simulator::Schedule              (Seconds               (totalTime/30),
&MobilityModel::SetPosition, mob, Vector (200,200,200));

}
```

Pentru a realiza eliminarea unui nod am început prin alegerea nodului țintă: Ptr<Node> node = nodes.Get (size/2); Funcția GetObjectm-a ajutat să identific nodul în rețea. Functia Schedule realizează eliminarea nodului din rețea, în timpul simularii la momentul prestabilit de mine.

Aodv_Neighbor.cc

```
#include "aodv-neighbor.h"
#include "ns3/log.h"
#include <algorithm>

NS_LOG_COMPONENT_DEFINE ("AodvNeighbors");

namespace ns3
{
namespace aodv
{
     bool
Neighbors::IsNeighbor (Ipv4Address addr)
{
  Purge ();
  for (std::vector<Neighbor>::const_iterator i = m_nb.begin ();
       i != m_nb.end (); ++i)
    {
      if (i->m_neighborAddress == addr)
        return true;
    }
  return false;
}

Time
Neighbors::GetExpireTime (Ipv4Address addr)
{
  Purge ();
  for (std::vector<Neighbor>::const_iterator i = m_nb.begin ();
 i!= m_nb.end (); ++i)
    {
      if (i->m_neighborAddress == addr)
        return (i->m_expireTime - Simulator::Now ());
    }
  return Seconds (0);
}

void
Neighbors::Update (Ipv4Address addr, Time expire)
{
```

```
  for (std::vector<Neighbor>::iterator i = m_nb.begin (); i!= m_nb.end();
++i)
    if (i->m_neighborAddress == addr)
      {
        i->m_expireTime
          = std::max (expire + Simulator::Now (), i->m_expireTime);
        if (i->m_hardwareAddress == Mac48Address ())
          i->m_hardwareAddress = LookupMacAddress (i->m_neighborAddress);
        return;
      }

void
Neighbors::Purge ()
{
  if (m_nb.empty ())
    return;
  CloseNeighbor pred;
  if (!m_handleLinkFailure.IsNull ())
    {
      for (std::vector<Neighbor>::iterator j = m_nb.begin ();
 j!= m_nb.end (); ++j)
        {
          if (pred (*j))
            {
              NS_LOG_LOGIC ("Close link to " << j->m_neighborAddress);
              m_handleLinkFailure (j->m_neighborAddress);
            }
        }
    }
  m_nb.erase (std::remove_if (m_nb.begin (), m_nb.end (), pred),
 m_nb.end ());
  m_ntimer.Cancel ();
  m_ntimer.Schedule ();
}
```

Aodv_Rtable.cc

```
#include "aodv-rtable.h"
#include <algorithm>
#include <iomanip>
#include "ns3/simulator.h"
#include "ns3/log.h"

NS_LOG_COMPONENT_DEFINE ("AodvRoutingTable");

namespace ns3
{
namespace aodv
{
RoutingTableEntry::RoutingTableEntry  (Ptr<NetDevice>  dev,  Ipv4Address
dst,          bool          vSeqNo,          uint32_t          seqNo,
Ipv4InterfaceAddress  iface,  uint16_t  hops,  Ipv4Address  nextHop,  Time
lifetime) :
  m_ackTimer (Timer::CANCEL_ON_DESTROY),
  m_validSeqNo (vSeqNo), m_seqNo (seqNo), m_hops (hops),
```

```
  m_lifeTime (lifetime + Simulator::Now ()), m_iface (iface), m_flag
(VALID),
  m_reqCount    (0),    m_blackListState    (false),    m_blackListTimeout
(Simulator::Now ())
{
  m_ipv4Route = Create<Ipv4Route> ();
  m_ipv4Route->SetDestination (dst);
  m_ipv4Route->SetGateway (nextHop);
  m_ipv4Route->SetSource (m_iface.GetLocal ());
  m_ipv4Route->SetOutputDevice (dev);
}

RoutingTableEntry::~RoutingTableEntry ()
{
}

bool
RoutingTableEntry::InsertPrecursor (Ipv4Address id)
{
  NS_LOG_FUNCTION (this << id);
  if (!LookupPrecursor (id))
    {
      m_precursorList.push_back (id);
      return true;
    }
  else
    return false;
}

bool
RoutingTableEntry::LookupPrecursor (Ipv4Address id)
{
  NS_LOG_FUNCTION (this << id);
  for (std::vector<Ipv4Address>::const_iterator i = m_precursorList.begin
(); i
       != m_precursorList.end (); ++i)
    {
      if (*i == id)
        {
          NS_LOG_LOGIC ("Precursor " << id << " found");
          return true;
        }
    }
  NS_LOG_LOGIC ("Precursor " << id << " not found");
  return false;
}

bool
RoutingTableEntry::DeletePrecursor (Ipv4Address id)
{
  NS_LOG_FUNCTION (this << id);
  std::vector<Ipv4Address>::iterator i=std::remove (m_precursorList.begin
(),                                             m_precursorList.end (), id);
  if (i == m_precursorList.end ())
    {
      NS_LOG_LOGIC ("Precursor " << id << " not found");
      return false;
    }
```

```
  else
    {
      NS_LOG_LOGIC ("Precursor " << id << " found");
      m_precursorList.erase (i, m_precursorList.end ());
    }
  return true;
}
bool
RoutingTable::LookupRoute (Ipv4Address id, RoutingTableEntry & rt)
{
  NS_LOG_FUNCTION (this << id);
  Purge ();
  if (m_ipv4AddressEntry.empty ())
    {
      NS_LOG_LOGIC ("Route to " << id << " not found; m_ipv4AddressEntry
is empty");
      return false;
    }
  std::map<Ipv4Address, RoutingTableEntry>::const_iterator i =
    m_ipv4AddressEntry.find (id);
  if (i == m_ipv4AddressEntry.end ())
    {
      NS_LOG_LOGIC ("Route to " << id << " not found");
      return false;
    }
  rt = i->second;
  NS_LOG_LOGIC ("Route to " << id << " found");
  return true;
}

bool
RoutingTable::LookupValidRoute (Ipv4Address id, RoutingTableEntry & rt)
{
  NS_LOG_FUNCTION (this << id);
  if (!LookupRoute (id, rt))
    {
      NS_LOG_LOGIC ("Route to " << id << " not found");
      return false;
    }
  NS_LOG_LOGIC ("Route to " << id << " flag is " << ((rt.GetFlag () ==
VALID) ? "valid" : "not valid"));
  return (rt.GetFlag () == VALID);
}

bool
RoutingTable::DeleteRoute (Ipv4Address dst)
{
  NS_LOG_FUNCTION (this << dst);
  Purge ();
  if (m_ipv4AddressEntry.erase (dst) != 0)

 {
      NS_LOG_LOGIC ("Route deletion to " << dst << " successful");
```

```
      return true;
    }
  NS_LOG_LOGIC ("Route deletion to " << dst << " not successful");
  return false;
}

bool
RoutingTable::AddRoute (RoutingTableEntry & rt)
{
  NS_LOG_FUNCTION (this);
  Purge ();
  if (rt.GetFlag () != IN_SEARCH)
    rt.SetRreqCnt (0);
  std::pair<std::map<Ipv4Address,   RoutingTableEntry>::iterator,   bool>
result = m_ipv4AddressEntry.insert (std::make_pair (rt.GetDestination (),
rt));
  return result.second;
}

bool
RoutingTable::Update (RoutingTableEntry & rt)
{
  NS_LOG_FUNCTION (this);
  std::map<Ipv4Address, RoutingTableEntry>::iterator i =
    m_ipv4AddressEntry.find (rt.GetDestination ());
  if (i == m_ipv4AddressEntry.end ())
    {
      NS_LOG_LOGIC  ("Route  update  to  "  <<  rt.GetDestination  ()  <<  "
fails; not found");
      return false;
    }
  i->second = rt;
  if (i->second.GetFlag () != IN_SEARCH)
    {
      NS_LOG_LOGIC ("Route update to " << rt.GetDestination () << " set
RreqCnt to 0");
      i->second.SetRreqCnt (0);
    }
  return true;
}
```

Aodv_packet.cc

```
#include "aodv-packet.h"
#include "ns3/address-utils.h"
#include "ns3/packet.h"

namespace ns3
{
namespace aodv
{
```

```
NS_OBJECT_ENSURE_REGISTERED (TypeHeader) ;

TypeHeader::TypeHeader (MessageType t) :
  m_type (t), m_valid (true)
{
}

TypeId
TypeHeader::GetTypeId ()
{
  static TypeId tid = TypeId ("ns3::aodv::TypeHeader")
    .SetParent<Header> ()
    .AddConstructor<TypeHeader> () ;
  return tid;
}

TypeHeader::GetSerializedSize () const
{
  return 1;
}

void
TypeHeader::Serialize (Buffer::Iterator i) const
{
  i.WriteU8 ((uint8_t) m_type);
}

uint32_t
TypeHeader::Deserialize (Buffer::Iterator start)
{
  Buffer::Iterator i = start;
  uint8_t type = i.ReadU8 ();
  m_valid = true;
  switch (type)
    {
    case AODVTYPE_RREQ:
    case AODVTYPE_RREP:
    case AODVTYPE_RERR:
    case AODVTYPE_RREP_ACK:
      {
        m_type = (MessageType) type;
        break;
      }
    default:
      m_valid = false;
    }
  uint32_t dist = i.GetDistanceFrom (start);
  NS_ASSERT (dist == GetSerializedSize ());
  return dist;
}

void
TypeHeader::Print (std::ostream &os) const
{
  switch (m_type)
    {
    case AODVTYPE_RREQ:
```

```
      {
        os << "RREQ";
        break;
      }
    case AODVTYPE_RREP:
      {
        os << "RREP";
        break;
      }
    case AODVTYPE_RERR:
      {
        os << "RERR";
        break;
      }
    case AODVTYPE_RREP_ACK:
      {
        os << "RREP_ACK";
        break;
      }
    default:
      os << "UNKNOWN_TYPE";
    }
}
```

RREQ:

```
RreqHeader::RreqHeader (uint8_t flags, uint8_t reserved, uint8_t
hopCount, uint32_t requestID, Ipv4Address dst,
                        uint32_t dstSeqNo, Ipv4Address origin, uint32_t
originSeqNo) :
  m_flags (flags), m_reserved (reserved), m_hopCount (hopCount),
m_requestID (requestID), m_dst (dst),
  m_dstSeqNo (dstSeqNo), m_origin (origin), m_originSeqNo (originSeqNo)
{
}

NS_OBJECT_ENSURE_REGISTERED (RreqHeader) ;

TypeId
RreqHeader::GetTypeId ()
{
  static TypeId tid = TypeId ("ns3::aodv::RreqHeader")
    .SetParent<Header> ()
    .AddConstructor<RreqHeader> ();
  return tid;
}

TypeId
RreqHeader::GetInstanceTypeId () const
{
  return GetTypeId ();
}

uint32_t
RreqHeader::GetSerializedSize () const
{
```

```
  return 23;
}

void
RreqHeader::Serialize (Buffer::Iterator i) const
{
  i.WriteU8 (m_flags);
  i.WriteU8 (m_reserved);
  i.WriteU8 (m_hopCount);
  i.WriteHtonU32 (m_requestID);
  WriteTo (i, m_dst);
  i.WriteHtonU32 (m_dstSeqNo);
  WriteTo (i, m_origin);
  i.WriteHtonU32 (m_originSeqNo);
}

uint32_t
RreqHeader::Deserialize (Buffer::Iterator start)
{
  Buffer::Iterator i = start;
  m_flags = i.ReadU8 ();
  m_reserved = i.ReadU8 ();
  m_hopCount = i.ReadU8 ();
  m_requestID = i.ReadNtohU32 ();
  ReadFrom (i, m_dst);
  m_dstSeqNo = i.ReadNtohU32 ();
  ReadFrom (i, m_origin);
  m_originSeqNo = i.ReadNtohU32 ();

  uint32_t dist = i.GetDistanceFrom (start);
  NS_ASSERT (dist == GetSerializedSize ());
  return dist;
}

void
RreqHeader::Print (std::ostream &os) const
{
  os << "RREQ ID " << m_requestID << " destination: ipv4 " << m_dst
     << " sequence number " << m_dstSeqNo << " source: ipv4 "
     << m_origin << " sequence number " << m_originSeqNo
     << " flags:" << " Gratuitous RREP " << (*this).GetGratiousRrep ()
     << " Destination only " << (*this).GetDestinationOnly ()
     << " Unknown sequence number " << (*this).GetUnknownSeqno ();
}
```

RREP:

```
RrepHeader::RrepHeader (uint8_t prefixSize, uint8_t hopCount, Ipv4Address
dst,
                        uint32_t  dstSeqNo,  Ipv4Address  origin,  Time
lifeTime) :
  m_flags (0), m_prefixSize (prefixSize), m_hopCount (hopCount),
  m_dst (dst), m_dstSeqNo (dstSeqNo), m_origin (origin)
{
  m_lifeTime = uint32_t (lifeTime.GetMilliSeconds ());
}
```

```
NS_OBJECT_ENSURE_REGISTERED (RrepHeader) ;


TypeId
RrepHeader::GetTypeId ()
{
  static TypeId tid = TypeId ("ns3::aodv::RrepHeader")
    .SetParent<Header> ()
    .AddConstructor<RrepHeader> ()
  ;
  return tid;
}

TypeId
RrepHeader::GetInstanceTypeId () const
{
  return GetTypeId ();
}

uint32_t
RrepHeader::GetSerializedSize () const
{
  return 19;
}

void
RrepHeader::Serialize (Buffer::Iterator i) const
{
  i.WriteU8 (m_flags);
  i.WriteU8 (m_prefixSize);
  i.WriteU8 (m_hopCount);
  WriteTo (i, m_dst);
  i.WriteHtonU32 (m_dstSeqNo);
  WriteTo (i, m_origin);
  i.WriteHtonU32 (m_lifeTime);
}

uint32_t
RrepHeader::Deserialize (Buffer::Iterator start)
{
  Buffer::Iterator i = start;

  m_flags = i.ReadU8 ();
  m_prefixSize = i.ReadU8 ();
  m_hopCount = i.ReadU8 ();
  ReadFrom (i, m_dst);
  m_dstSeqNo = i.ReadNtohU32 ();
  ReadFrom (i, m_origin);
  m_lifeTime = i.ReadNtohU32 ();

  uint32_t dist = i.GetDistanceFrom (start);
  NS_ASSERT (dist == GetSerializedSize ());
  return dist;
}

void
RrepHeader::Print (std::ostream &os) const
{
```

```
  os << "destination: ipv4 " << m_dst << " sequence number " <<
m_dstSeqNo;
  if (m_prefixSize != 0)
    {
      os << " prefix size " << m_prefixSize;
    }
  os << " source ipv4 " << m_origin << " lifetime " << m_lifeTime
     << " acknowledgment required flag " << (*this).GetAckRequired ();
}

void
RrepHeader::SetLifeTime (Time t)
{
  m_lifeTime = t.GetMilliSeconds ();
}

Time
RrepHeader::GetLifeTime () const
{
  Time t (MilliSeconds (m_lifeTime));
  return t;
}
```

RERR:

```
RerrHeader::RerrHeader () :
  m_flag (0), m_reserved (0)
{
}

NS_OBJECT_ENSURE_REGISTERED (RerrHeader) ;

TypeId
RerrHeader::GetTypeId ()
{
  static TypeId tid = TypeId ("ns3::aodv::RerrHeader")
    .SetParent<Header> ()
    .AddConstructor<RerrHeader> () ;
  return tid;
}

TypeId
RerrHeader::GetInstanceTypeId () const
{
  return GetTypeId ();
}

uint32_t
RerrHeader::GetSerializedSize () const
{
  return (3 + 8 * GetDestCount ());
}

void
RerrHeader::Serialize (Buffer::Iterator i ) const
{
  i.WriteU8 (m_flag);
  i.WriteU8 (m_reserved);
```

```
  i.WriteU8 (GetDestCount ());
  std::map<Ipv4Address, uint32_t>::const_iterator j;
  for (j = m_unreachableDstSeqNo.begin();j!= m_unreachableDstSeqNo.end();
++j)
    {
      WriteTo (i, (*j).first);
      i.WriteHtonU32 ((*j).second);
    }
}

uint32_t
RerrHeader::Deserialize (Buffer::Iterator start )
{
  Buffer::Iterator i = start;
  m_flag = i.ReadU8 ();
  m_reserved = i.ReadU8 ();
  uint8_t dest = i.ReadU8 ();
  m_unreachableDstSeqNo.clear ();
  Ipv4Address address;
  uint32_t seqNo;
  for (uint8_t k = 0; k < dest; ++k)
    {
      ReadFrom (i, address);
      seqNo = i.ReadNtohU32 ();
      m_unreachableDstSeqNo.insert (std::make_pair (address, seqNo));
    }

  uint32_t dist = i.GetDistanceFrom (start);
  NS_ASSERT (dist == GetSerializedSize ());
  return dist;
}

void
RerrHeader::Print (std::ostream &os ) const
{
  os << "Unreachable destination (ipv4 address, seq. number):";
  std::map<Ipv4Address, uint32_t>::const_iterator j;
  for (j = m_unreachableDstSeqNo.begin (); j != m_unreachableDstSeqNo.end
(); ++j)
    {
      os << (*j).first << ", " << (*j).second;
    }
  os << "No delete flag " << (*this).GetNoDelete ();
}
```

Aodv_routing_protocol.cc

```
#include "aodv-routing-protocol.h"
#include "ns3/log.h"
#include "ns3/boolean.h"
#include "ns3/random-variable-stream.h"
#include "ns3/inet-socket-address.h"
#include "ns3/trace-source-accessor.h"
#include "ns3/udp-socket-factory.h"
#include "ns3/wifi-net-device.h"
#include "ns3/adhoc-wifi-mac.h"
#include "ns3/string.h"
```

```
#include "ns3/pointer.h"
#include <algorithm>
#include <limits>

NS_LOG_COMPONENT_DEFINE ("AodvRoutingProtocol");

namespace ns3
{
namespace aodv
{
……
bool
RoutingProtocol::Forwarding (Ptr<const Packet> p, const Ipv4Header &
header,
                             UnicastForwardCallback ucb, ErrorCallback
ecb)
{
  NS_LOG_FUNCTION (this);
  Ipv4Address dst = header.GetDestination ();
  Ipv4Address origin = header.GetSource ();
  m_routingTable.Purge ();
  RoutingTableEntry toDst;
  if (m_routingTable.LookupRoute (dst, toDst))
    {
      if (toDst.GetFlag () == VALID)
        {
          Ptr<Ipv4Route> route = toDst.GetRoute ();
          NS_LOG_LOGIC (route->GetSource ()<<" forwarding to " << dst <<
" from " << origin << " packet " << p->GetUid ());

NS_OBJECT_ENSURE_REGISTERED (RoutingProtocol)
  ;
void
RoutingProtocol::PrintRoutingTable (Ptr<OutputStreamWrapper> stream)
const
{
  *stream->GetStream () << "Node: " << m_ipv4->GetObject<Node> ()->GetId
() << " Time: " << Simulator::Now ().GetSeconds () << "s ";
  m_routingTable.Print (stream);
}

          RoutingTableEntry toOrigin;
          m_routingTable.LookupRoute (origin, toOrigin);
          UpdateRouteLifeTime (toOrigin.GetNextHop (),
ActiveRouteTimeout);
          m_nb.Update (route->GetGateway (), ActiveRouteTimeout);
          m_nb.Update (toOrigin.GetNextHop (), ActiveRouteTimeout);
          ucb (route, p, header);
          return true;
        }
      else
        {
          if (toDst.GetValidSeqNo ())
            {
              SendRerrWhenNoRouteToForward (dst, toDst.GetSeqNo (),
origin);
              NS_LOG_DEBUG ("Drop packet " << p->GetUid () << " because
no route to forward it.");
```

```
              return false;
            }
        }
    }
  NS_LOG_LOGIC ("route not found to "<< dst << ". Send RERR message.");
  NS_LOG_DEBUG ("Drop packet " << p->GetUid () << " because no route to
forward it.");
  SendRerrWhenNoRouteToForward (dst, 0, origin);
  return false;
}

void
RoutingProtocol::SetIpv4 (Ptr<Ipv4> ipv4)
{
  NS_ASSERT (ipv4 != 0);
  NS_ASSERT (m_ipv4 == 0);

  if (EnableHello)
    {
      m_htimer.SetFunction (&RoutingProtocol::HelloTimerExpire, this);
      m_htimer.Schedule (MilliSeconds
(m_uniformRandomVariable->GetInteger (0, 100)));
    }
  m_ipv4 = ipv4;
  RoutingTableEntry       rt       (/*device=*/       m_lo,       /*dst=*/
Ipv4Address::GetLoopback (), /*know seqno=*/ true, /*seqno=*/ 0,
                                    /*iface=*/       Ipv4InterfaceAddress
(Ipv4Address::GetLoopback (), Ipv4Mask ("255.0.0.0")),
                                    /*hops=*/     1,     /*next     hop=*/
Ipv4Address::GetLoopback (),
                                    /*lifetime=*/
Simulator::GetMaximumSimulationTime ());
  m_routingTable.AddRoute (rt);

void
RoutingProtocol::SendRequest (Ipv4Address dst)
{
  NS_LOG_FUNCTION ( this << dst);

  if (m_rreqCount == RreqRateLimit)
    {
      Simulator::Schedule    (m_rreqRateLimitTimer.GetDelayLeft    ()    +
MicroSeconds (100),
                              &RoutingProtocol::SendRequest, this, dst);
      return;
    }
  else
    m_rreqCount++;
RreqHeader rreqHeader;
  rreqHeader.SetDst (dst);

  RoutingTableEntry rt;
  if (m_routingTable.LookupRoute (dst, rt))
    {
      rreqHeader.SetHopCount (rt.GetHop ());
      if (rt.GetValidSeqNo ())
        rreqHeader.SetDstSeqno (rt.GetSeqNo ());
      else
```

```
        rreqHeader.SetUnknownSeqno (true);
       rt.SetFlag (IN_SEARCH);
       m_routingTable.Update (rt);
     }
   else
     {
       rreqHeader.SetUnknownSeqno (true);
       Ptr<NetDevice> dev = 0;

       RoutingTableEntry   newEntry   (/*device=*/   dev,   /*dst=*/   dst,
/*validSeqNo=*/ false, /*seqno=*/ 0,
                                                    /*iface=*/
Ipv4InterfaceAddress (),/*hop=*/ 0,
                                                    /*nextHop=*/    Ipv4Address
(), /*lifeTime=*/ Seconds (0));
       newEntry.SetFlag (IN_SEARCH);
       m_routingTable.AddRoute (newEntry);
     }

   if (GratuitousReply)
     rreqHeader.SetGratiousRrep (true);
   if (DestinationOnly)
     rreqHeader.SetDestinationOnly (true);

   m_seqNo++;
   rreqHeader.SetOriginSeqno (m_seqNo);
   m_requestId++;
   rreqHeader.SetId (m_requestId);
   rreqHeader.SetHopCount (0);
void
RoutingProtocol::RecvRequest   (Ptr<Packet>   p,   Ipv4Address   receiver,
Ipv4Address src)
{
   NS_LOG_FUNCTION (this);
   RreqHeader rreqHeader;
   p->RemoveHeader (rreqHeader);

   // A node ignores all RREQs received from any node in its blacklist
   RoutingTableEntry toPrev;
   if (m_routingTable.LookupRoute (src, toPrev))
     {
       if (toPrev.IsUnidirectional ())
         {
           NS_LOG_DEBUG ("Ignoring RREQ from node in blacklist");
           return;
         }
     }

   uint32_t id = rreqHeader.GetId ();
   Ipv4Address origin = rreqHeader.GetOrigin ();
   if (m_rreqIdCache.IsDuplicate (origin, id))
     {
       NS_LOG_DEBUG ("Ignoring RREQ due to duplicate");
       return;
     }
   if (m_rreqIdCache.IsDuplicate (origin, id))
     {
       NS_LOG_DEBUG ("Ignoring RREQ due to duplicate");
```

```
      return;
    }
RoutingTableEntry toOrigin;
  if (!m_routingTable.LookupRoute (origin, toOrigin))
    {
      Ptr<NetDevice>     dev     =     m_ipv4->GetNetDevice     (m_ipv4-
>GetInterfaceForAddress (receiver));
      RoutingTableEntry  newEntry  (/*device=*/  dev,  /*dst=*/  origin,
/*validSeno=*/ true, /*seqNo=*/ rreqHeader.GetOriginSeqno (),
                                                /*iface=*/          m_ipv4-
>GetAddress  (m_ipv4->GetInterfaceForAddress  (receiver),  0),  /*hops=*/
hop,
                                                /*nextHop*/            src,
/*timeLife=*/   Time   ((2   *   NetTraversalTime   -   2   *   hop   *
NodeTraversalTime)));
      m_routingTable.AddRoute (newEntry);
    }
  else
    {
      if (toOrigin.GetValidSeqNo ())
        {
          if   (int32_t   (rreqHeader.GetOriginSeqno   ())   -   int32_t
(toOrigin.GetSeqNo ()) > 0)
            toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
        }
      else
      toOrigin.SetSeqNo (rreqHeader.GetOriginSeqno ());
      toOrigin.SetValidSeqNo (true);
      toOrigin.SetNextHop (src);
      toOrigin.SetOutputDevice       (m_ipv4->GetNetDevice       (m_ipv4-
>GetInterfaceForAddress (receiver)));
      toOrigin.SetInterface          (m_ipv4->GetAddress         (m_ipv4-
>GetInterfaceForAddress (receiver), 0));
      toOrigin.SetHop (hop);
      toOrigin.SetLifeTime (std::max (Time (2 * NetTraversalTime - 2 *
hop                         *                          NodeTraversalTime),
toOrigin.GetLifeTime ()));
      m_routingTable.Update (toOrigin);

    }

  RoutingTableEntry toNeighbor;
  if (!m_routingTable.LookupRoute (src, toNeighbor))
    {
      NS_LOG_DEBUG ("Neighbor:" << src << " not found in routing table.
Creating an entry");
      Ptr<NetDevice>     dev     =     m_ipv4->GetNetDevice     (m_ipv4-
>GetInterfaceForAddress (receiver));
      RoutingTableEntry      newEntry      (dev,      src,      false,
rreqHeader.GetOriginSeqno                                          (),
m_ipv4->GetAddress (m_ipv4->GetInterfaceForAddress (receiver), 0),1, src,
ActiveRouteTimeout);
      m_routingTable.AddRoute (newEntry);
    }
  else
    {
      toNeighbor.SetLifeTime (ActiveRouteTimeout);
      toNeighbor.SetValidSeqNo (false);
```

```
      toNeighbor.SetSeqNo (rreqHeader.GetOriginSeqno ());
      toNeighbor.SetFlag (VALID);
      toNeighbor.SetOutputDevice (m_ipv4->GetNetDevice
(m_ipv4->GetInterfaceForAddress (receiver)));
      toNeighbor.SetInterface (m_ipv4->GetAddress
(m_ipv4->GetInterfaceForAddress (receiver), 0));
      m_routingTable.Update (toNeighbor);
    }
  m_nb.Update (src, Time (AllowedHelloLoss * HelloInterval));
  NS_LOG_LOGIC (receiver << " receive RREQ with hop count " <<
static_cast<uint32_t>(rreqHeader.GetHopCount ())
                        << " ID " << rreqHeader.GetId ()
                        << " to destination " << rreqHeader.GetDst ());

     }

     ……

     }
```