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

Algoritmi de optimizare a convergenței informației de rutare în rețele WAN

Proiect de diplomă

prezentat ca cerinţă parţială pentru obţinerea titlului de
Inginer în domeniul *Calculatoare și Tehnologia Informației*
programul de studii de licență *Ingineria Informației*

Conducător ştiinţific
Conf.dr.ing. Ștefan STĂNCESCU

Absolvent
Victor AFLOREI

2014

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
Aflorei V. Victor, grupa 441A

1. Titlul temei: Algoritmi de optimizare a convergenței informației de rutare în rețele WAN

2. Contribuţia practică, originală a studentului va consta în:

Analiza arhitecturilor și algoritmilor de rutare utilizați în rețelele WAN ale furnizorilor de servicii internet. Se va simula o rețea în care se va studia convergența pentru scenarii de dinamică a rețelei folosind algoritmii de rutare: Shortest Path First (SPF), Best Path, algoritmul de expediere cu schimb de etichete.

Testarea, analiza și optimizarea convergenței în rețelele furnizorilor de servicii se va realiza cu ajutorul simulatoarelor de rețea Network Simulator și GNS3.

3. Proiectul se bazează pe cunoştinţe dobândite în principal la următoarele 3 discipline:

- Rețele de Calculatoare;
- Arhitecturi de rețea și Internet;
- Sisteme de comunicații.

4. Realizarea practică/ proiectul rămân în proprietatea: Aflorei Victor şi UPB, Facultatea de Electronică, Telecomunicații și Tehnologia Informației

5. Proprietatea intelectuală asupra proiectului aparţine: Aflorei Victor şi UPB, Facultatea de Electronică, Telecomunicații și Tehnologia Informației

6. Locul de desfăşurare a activităţii: UPB, Facultatea de Electronică, Telecomunicații și Tehnologia Informației

7. Data eliberării temei: 3.12.2013

CONDUCĂTOR LUCRARE: *Conf. Dr. Ing. Ștefan Stăncescu*

STUDENT: *Victor Aflorei*

Declaraţie de onestitate academică

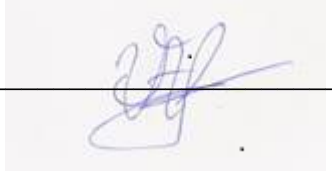
Prin prezenta declar că lucrarea cu titlul "*Algoritmi de optimizare a convergenței informației de rutare în rețele WAN*", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul *Calculatoare și Tehnologia Informației*, programul de studii *Ingineria Informației* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

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

Bucureşti, *03.07.2014*

Absolvent *Victor AFLOREI*



_____

Cuprins

Lista Figurilor

Lista Acronimelor

IP - *Internet Protocol*

RFC - *Request for Comments*

OSI - *Open Systems Interconnection*

ISP - *Internet Service Provider*

MTU - *Maximum Transmission Unit*

ICMP - *Internet Control Message Protocol*

TCP - *Transmission Control Protocol*

UDP - *User Datagram Protocol*

PPP - *Point-to-Point*

RIP - *Rounting Information Protocol*

EIGRP - *Enhanced Interior Gateway Routing Protocol*

BGP - *Border Gateway Protocol*

OSPF - *Open Shortest Path First*

OSPFv2 - *Open Shortest Path First version 2*

RIB - *Routing Information Base*

FIB - *Forwarding Information Base*

AS - *Autononous System*

SPF - *Shortest Path First*

BFD - *Bidirectional Forwarding Detection*

VoIP - *Voice over IP*

TTL - *Time to Live*

Introducere

În acestă lucrare am supus unei analize calitative convergența protocoalelor de rutare. Am realizat o analiză mai detaliată a protocoalelor care folosesc starea legăturilor și în caz particular a protocolului OSPF.

OSFP este un protocol de rutare dinamic care folosește starea legăturilor și care interconectează rutere din interiorul unei rețele prin schimbul de informații de rutare. Dinamica protocolului provine din faptul că își menține tabela de rutare cu toate rutele posibile și o actualizează periodic. El efectuează un calcul în timp real asupra schimbărilor din rețea. Este un protocol de rutare care folosește starea legăturilor deoarece fiecare ruter urmărește starea legăturilor din rețea. Este un Interior Gateway Protocol (IGP) deoarece operează doar în interiorul unui singur AS și este unul din cele mai utilizate IGP din rețelele întreprinderilor mari. Acesta operează folosind stiva TCP/IP și este încapsulat cu numărul de protocol 89. Pentru a schimba informațiile de rutare între ruterelele din rețea folosește IP multicast.

Am început lucrarea printr-o descriere a conceptului general de rutare și a tuturor factorilor care influențează rutarea. Mai departe am descris caracteristicile principale ale protocolului OSPF și după am trecut la descrierea tehnicilor utilizate pentru a îmbunătați convergența protocolului. Am descris tehnici de îmbunătire a convergenței pentru toți cei patru factori implicați în procesul de convergență și anume tenhnici pentru detectarea rapidă a evenimentelor, tehnici pentru îmbunătățirea timpului de propagare a evenimentului, tehnici care influențează calculul SPF și tehnici care îmbunătățesc timpul de actualizare RIB/FIB.

În partea practică a acestei lucrări am simulat o rețea OSPF cu ajutorul căreia am analizat îmbunătățirea convergenței protocolului modificându-i intervalul de trimitere a hello-urilor. Deasemenea am analizat comportamentul protocolului și asupra unei rețele care este alcătuită din mai multe arii. Am simulat o rețea în care am comparat convergența mai multor protocoale (RIP, OSPF, EIGRP).

Motivația pentru alegerea acestei teme a fost înclinația personală pentru domeniul networking-ului împreună cu dorința de a aprofunda modul de lucru al protocoalelor de rutare.

1. Concepte generale despre rutare

1.1 Arhitectura rutării în Internet

Internetul este organizat în regiuni numite Siteme Autonome (*Autonomous Systems AS*). Fiecare sistem autonom constă dintr-o colecție de rutere aflate sub controlul unei singure entități administrative - de exemplu, toate ruterele aparținând unui anunit furnizor de servicii, unei corporații sau unei universități.

Colecția de sisteme autonome este organizată intr-un model ierarhic. Cu cât ne apropiem de vârful ierarhiei - de nucleul Internetului - cu atât mai multe rutere se găsesc în fiecare AS. În același timp prefixele din tabele de rutare ale ruterelor se scurtează. Sistemele autonome din nucleul Internetului transportă întreaga tabela de rutare, aproximativ 500000 de rute și nu folosesc o rută implicită (ele se mai numesc *default-free zone*). Toate celelalte sisteme autonome folosesc o rută implicită, care țintește către sistemele autonome aflate mai sus în ierarhie și care le permite să transporte numai un subset din rutele Internetului.

Sistemele autonome care au rolul de a oferi conectivitate la Internet sunt numite *Furnizori de Servicii de Internet (ISP - Internet Service Providers).* Operatorii de rețea care configurează și întrețin conexiunile dintre ISP folosesc un limbaj propriu. Când doi furnizori de servicii se conectează, aceștia de obicei stabilesc un acord de peering (peering agreement). Dacă cei doi furnizori se găsesc pe același nivel din ierarhie acesta este un simplu acord pentru a schimba informații de rutare. Totuși, când un furnizor este mai jos în ierarhie (downstream) acest AS intră uneori într-o relație de client cu furnizorul upstream. Acest lucru înseamnă că furnizorul upstream anunță adresele furnizorului downstream către restul Internetului și transmite pachetele furnizorului downstream către ceilalți furnizori și către clienții lor. Astfel, furnizorul upstream oferă tranzit pentru furnizorul downstream.

Deasemenea doi furnizori de servicii se pot conecta direct prin intermediul unei conexiuni private cum ar fi o linie închiriată de mare viteză sau cu ajutorul unui circuit ATM (*Asynchronous Transfer Mode*). Acest mod de conectare privată a devenit comună între furnizorii de servicii din vârful ierarhiei care formează nucleul Internetului.

După cum au fost inițial proiectate ruterele din interiorul unui sistem autonom schimbă informații de rutare cu ajutorul unui protocol de rutare comun (numit *Interior Gateway Protocol sau IGP*) în timp ce un protocol de rutare diferit (numit *Exterior Gateway Protocol sau BGP*) a fost folosit pentru a face schimb de informații de rutare între sisteme autonome. Regula care spune că în interiorul unui AS rulează un singur IGP a fost ruptă curând astfel că multiple AS-uri pot rula în același timp mai multe protocoale IGP. O colecție de rutere care funcționează cu același IGP este numită frecvent un domeniu de rutare (*routing domain*), în acest caz, un AS poate consta din mai multe domenii de rutare.

1.2 Rutarea în Internet

Internetul este o rețea cu comutare de pachete, care permite calculatoarelor atașate - PC-ul de pe biroul fiecăruia, stația de lucru care ne permite înscrierea la universitate folosind serviciul Web, mainframe-ul care ne verifică soldul cardului de credit, supercomputerul pe care îl utilizăm pentru simulări în laboratorul de fizică - schimbul de informații. Această informație este codată ca șiruri lungi de biți numite *pachete*. Când aceste pachete călătoresc prin intermediul Internetului pe drumul lor spre calculatoarele destinație, deciziile de rutare sunt realizate în următorul mod: Acest pachet ar trebuie trimis pe această cale sau pe cealaltă cale? Dispozitivele care iau aceste decizii

sunt ele însele calculatoare, numite *rutere*. Algoritmii distribuiți pe care ruterele îi execută între ele cu scopul de a lua deciziile de rutare corecte se numesc *protocoale de rutare*.

În rețelele concrete, de obicei foarte mari, furnizorii de servicii de Internet (*Internet Service Provider - ISP*) se bazează pe protocoale de rutare dinamice pentru a menține tabelele de rutare actualizate. Rețeaua bazată pe protocolul TCP/IP permite rutarea eficientă a pachetelor de date bazându-se pe adresa IP. Ruterele sunt folosite în rețea pentru a controla și pentru a transmite datele.

În comunicarea informației prin pachete, funcțiile rutării sunt de a transporta traficul pe toată întinderea rețelelor și de a face ca ruterele să fie conștiente tot timpul unde trebuie să trimită traficul cu scopul de a ajunge la destinația finală. Pentru ca ruterele să distribuie datele efectiv și eficient alegerea protocolului de rutare devine un factor critic pentru a defini succesul rețelei în timp. Factorii care diferențiază un protocol de rutare de un altul includ viteza cu care se adaptează la schimbările de topologie numită *convergență*, capacitatea de a alege cel mai bun traseu dintre mai multe trasee și cantitatea de trafic de rețea pe care protocolul de rutare îl creează. În plus, a alege între un protocol standard și un protocol proprietar, ușurința de administrare, utilizarea procesorului și a memoriei și problemele legate de securitate, toate contribuie la complexitatea deciziei.

Avantajele principale al rutării dinamice față de rutarea statică sunt scalabilitatea și adaptabilitatea. O rețea cu rutate dinamică poate crește mult mai rapid și mai mare și este capabilă să se adapteze la schimbările din topologia rețelei, schimbări aduse de această creștere sau de eșecul unuia sau mai multor componente.

Folosind un protocol de rutare dinamic, ruterele află topologia rețelei comunicând cu celelalte rutere. Fiecare ruter își anunță prezența și rutele pe care le are la dispoziție celorlalte rutere din rețea. Prin urmare, în cazul în care se adaugă un nou ruter sau un segment suplimentar unui ruter existent celelalte rutere vor învăța despre această modificare și își vor ajusta tabelele de rutare în consecință. Capacitatea de a învăța despre modificările din configurația rețelei are implicații dincolo de adăugarea de noi segmente sau modificarea celor vechi. Deasemenea, acest lucru înseamnă că rețeaua se poate adapta la eșecuri. Dacă o rețea are rute redundante, atunci un eșec parțial al rețelei este observat de rutere ca și în cazul în care unele segmente ar fi fost adăugate sau unele segmente ar fi fost eliminate din rețea. Pe scurt, nu există nicio diferență reală între un eșec al rețelei și o modificare a configurației. Rutarea dinamică permite rețelei să continue să funcționeze, într-un mod degradat, atunci când apare un eșec parțial.

Termenii vectori distanță (*distance vector*) și starea legăturii (*link-state*) sunt utilizați pentru a grupa protocoalele de rutare bazându-se pe modul cum protocolul de rutare selectează cea mai bună cale de rutare fie folosind metrica și o interfață de ieșire fie selectează cea mai bună cale de rutare prin calcularea stării fiecărei legături dintr-o cale și găsește ruta care are cea mai mică metrică totală pentru a ajunge la destinație. Protocoalele vectori distanță folosesc calculul unei distanțe plus o interfață de ieșire pentru a alege cea mai bună cale către o rețea de destinație. Ruterele care folosesc rutarea vectori distanță împart informațiile de rutare sau o hartă a rutelor cu celelalte rutere din rețea. Atunci când se produce o schimbare în rețea, ruter-ul afectat de schimbare propagă noua informație de rutare către toate ruterele vecine. Fiecare beneficiar la care ajunge această informație adaugă un vector distanță tabelei de rutare înainte de a o transmite mai departe vecinilor săi. Protocoalele care folosesc starea legăturii urmăresc starea și tipul conexiunii pentru fiecare legătură și calculează o metrică bazându-se pe acestea dar și pe alti factori, inclusiv unii stabiliți de către administratorul de rețea. În rutarea bazată pe starea legăturii cel mai bun traseu pentru date se calculează pe baza costului și odată ce rețeaua a convers, traficul datorat protocolului este limitat doar la schimbările din anumite legături.

1.3 Suita de protocoale din Internet

În scopul de a realiza transferul de pachete între calculatoarele conectate la Internet și între ruterele care alcătuiesc Internetul în sine, trebuie urmate anumite reguli cu privire la formatul

pachetelor și cu privire la procesare. Aceste reguli sunt denumite protocoale. Suita de protocoale utilizate de Internet este denumită stiva TCP/IP.

Protocoalele din Internet au fost separate pe niveluri într-o încercare de a simplifica modul de proiectare a acestora. Aceste niveluri oferă o referință utilă, dar nu ar trebui considerate ca limite legate de hard și de viteză în proiectarea unui protocol. Protocoale din Internet, uneori, estompează diferențele dintre niveluri sau pur și simplu le încalcă din motive de eficiență. Stratificarea protocoalelor din Internet poate fi vizualizată cu ajutorul celor 7 niveluri ale modelului de referință OSI (*Open Systems Interconnection*).

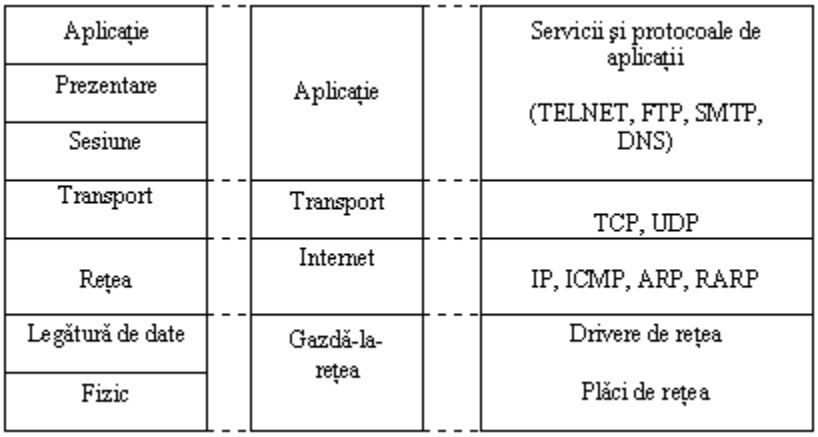


Figura 1.1 Modelul de referință OSI/ Modelul TCP/IP/ Protocoalele și rețeaua TCP/IP

Calculatoarele care comunică folosind Internetul sunt denumite uneori *host-uri* sau *gazde*. Aceste calculatoare și ruterele din Internet sunt interconectate folosind o gamă largă de tehnologii ale legăturilor.

Când vizualizăm nivelurile unui protocol, de obicei vorbim despre o stivă de protocoale. În modelul de referință OSI, cel mai mic nivel al stivei este format din protocoalele nivelului fizic iar cel mai înalt nivel este format din protocoalele nivelului aplicație. Trebuie să ne gândim la datele pachetizate a unei aplicații ca fiind transmise prin nivelurile calculatorului sursă, de la nivelul șapte la nivelul unu, fiecare nivel adăugând header-ul corespunzător. Când pachetele ajung la calculatorul destinație header-ele sunt înlăturate, de la nivelul unu la nivelul șapte, iar datele sunt preluate de aplicația către care erau destinate.

Fiecare nivel al stivei de protocole oferă, de obicei, servicii de multiplexare, astfel încât mai multe instanțe ale nivelului superior pot fi rulate simultan. De exemplu, protocolul unei legături corespunzătoare nivelului legătură de date va permite multiplexarea pachetelor pe baza tipului de pachet, astfel încât mai multe stive de protocole pot fi susținute pe o singură legătură.

Modelul de referință OSI este destul de util în discutarea protocoalelor din Internet de la nivelul fizic până la nivelul transport. Cu toate acestea, nu toate protocoalele din Internet se încadrează în aceste niveluri bine definite. De exemplu, Address Resolution Protocol (ARP) se încadrează atât la nivelul rețea cât și la nivelul legătură de date. De asemenea, o parte din nivelele superioare ale modelului de referință OSI, cum ar fi nivelele sesiune și prezentare, au o relevanță mică pentru suita de protocoale din Internet.

Nivelul fizic

Nivelul fizic specifică modul cum biții care alcătuiesc un pachet sunt fizic codificați atunci când sunt transmiși și recepționați peste o tehnologie a unei anumite legături. Biții sunt de obicei codificați ca impulsuri electronice sau de lumină și organizați astfel încât erorile de transmisie să poată fi detectate. De exemplu, nivelul fizic pentru liniile telefonice este definit de diferite standarde, cum ar fi V.34.

Nivelul legătură de date

Nivelul legătură de date specifică modul în care sunt transmise și recepționate pachetele peste o tehnologie a unei anumite legături. Serviciile oferite la acest nivel includ identificarea stațiilor finale a legăturii, specificarea dimensiunii maxime suportate a pachetelor (*Maximum Transmission Unit - MTU*) și uneori, sistemele pentru prioritatea transmisiunii. De exemplu, protocolul Point-to-Point (PPP) este protocolul nivelului legătură de date utilizat în mod obișnuit în Internet peste liniile telefonice. Adesea, protocoalele nivelului legătură de date sunt specificate împreună cu protocoalele nivelului fizic, ca și în Ethernet. Header-ul nivelului legătură de date pentru 14-byte Ethernet oferă o adresă sursă de 6 octeți (adresa MAC), o adresa destinație de 6 octeți și un tip Ethernet de 2 octeți pentru multiplexarea pachetelor.

Nivelul rețea

Nivelul rețea este responsabil de expedierea pachetelor de-a lungul multiplelor legături (adică, prin unul sau mai multe rutere) atunci când sursa și destinația unui pachet sunt pe legături diferite sau segmente de rețea.

De obicei, este prevăzută o schemă de adresare pentru nivelul rețea, care să permită ruterelor să poată descoperi din ce segment de rețea aparține gazda destinație. Uneori, o capabilitate de fragmentare și reasamblare este prevăzută, permițând nivelului rețea să găzduiască, transparent, legături care suportă diferite dimensiuni maxime a pachetelor. Protocoalele de rutare dinamice, care permit ruterelor să găsească în mod automat căi către diferite destinații ale nivelului rețea, sunt, de asemenea, considerate ca făcând parte din nivelul de rețea, cum sunt protocoalele care controlează interacțiunea dintre gazde și rutere.

Cel mai important protocol al nivelului rețea se numește *Internet Protocol* (IP). Pachetele nivelului rețea sunt numite *pachete IP* sau *datagrame IP*. *Adresele IP* conținute de pachetele IP au dimensiune de 32 de biți. Fiecărui segment de rețea i se atribuie un interval de adrese IP, reprezentate ca prefixul unei adrese. Ruterele din Internet rutează pachetele către aceste prefixe în schimbul gazdelor individuale. Când o gazdă este atașată unui segment de rețea sau are o interfață către segmentul respectiv, îi este atribuită o adresă IP unică din intervalul de adrese a segmentului. Gazdele cu mai multe interfețe au mai multe adrese IP. IP conține mai multe protocoale de rutare, incluzînd OSPF, RIP, BGP. Protocolul ICMP (*Internet Control Message Protocol*) permite gazdelor să identifice ruterele atașate la segmentele lor și oferă anumite capabilități de diagnosticare a gazdelor, atunci când ruterele sunt în imposibilitatea de a livra pachetele către adresele destinație.

Nivelul transport

Nivelul transport asigură o conexiune capăt la capăt între aplicațiile care rulează în calculatoarele sursă și destinație. Serviciile de la acest nivel includ adesea detectarea erorilor, corecția erorilor și secvențierea datelor. Stiva de protocoale TCP/IP conține două protocoale de transport principale. Protocolul de control al transmisiei (TCP) oferă un flux de octeți de încredere între aplicații. Avantajele majore ale TCP cum sunt evitarea și controlarea congestiei au permis Internetului să continuie să prospere în ultimii ani. User Datagram Protocol (UDP) oferă o conexiune fără garanții de fiabilitate și este utilizat în mod normal de aplicațiile care nu necesită acest lucru, cum ar fi Network File System de la Sun (NFS).

Nivelurile superioare

Nivelul sesiune asigură mecanisme pentru pornirea și oprirea conexiunilor de la nivelul transport. Nivelul prezentare oferă modalități standard de codare și tipuri de reprezentare a datelor aplicațiilor. Internetul nu are protocoale explicite pentru nivelurile sesiune și prezentare; în schimb, de obicei, aceste funcții sunt construite direct în aplicațiile din Internet. Există, totuși, unele excepții. Protocolul Internet Remote Procedure Call protocol se încadrează la nivelul sesiune, iar protocoalele ASN.l sau External Data Representation (XDR) se încadrează la nivelul prezentare.

Nivelul aplicație poate fi gândit ca protocoalele care implementează diferite tipuri de servicii de rețea pe care le-am aștepta de la sistemul de operare de pe orice calculator. În Internet, protocoalele nivelului aplicație includ Simple Mail Transfer Protocol (SMTP); protocoale pentru transfer de fișiere, cum ar fi FTP și TFTP și protocolul de autentificare de la distanță, TELNET.

Internet *Domain Name System* (DNS), de asemenea, se încadrează la nivelul aplicație.

1.4 Adresele IP

Un pachet IP este rutat în funcție de adresa IP destinație de 32 de biți găsită în header-ul IP al pachetului. Adresele IP sunt în general reprezentate în notație *dotted zecimal*: valorile zecimale a fiecărui octet din adresă fiind separate prin perioade. De exemplu, adresa IP a cărei valoare hexazecimală este Oxc0090102 este scrisă ca 192.9.1.2.

Uitându-se la adresa IP, un ruter poate determina rapid dacă este vorba de o adresă unicast, broadcast sau multicast așa cum se arată în tabelul 1.1. Spațiul de adrese este împărțit în trei părți: majoritatea utilizate pentru unicast, o serie de adrese multicast și o mică porțiune din partea de sus a spațiului de adrese rezervate pentru aplicații viitoare. Există, de asemenea, unele adrese care au un scop special. O gazdă utilizează adresa 0.0.0.0 ca sursă IP pentru pachetele sale atunci când gazda încearcă să învețe adresa sa IP cu ajutorul protocolelor BOOTP sau DHCP. Pachetele trimise pe un segment de rețea cu IP-ul destinație setat cu adresa de broadcast 255.255.255.255 vor fi primite de către toate celelalte gazde IP și de către toate celelalte rutere de se segmentul respectiv. Această funcție este utilizată de protocoalele de rutare pentru a difuza actualizările de rutare.

Intervalul de adrese	Funcționalitatea adreselor
1.0.0.0 - 233.255.255.255	Adrese IP unicast
224.0.0.0 - 239.255.255.255	Adrese IP multicast
240.0.0.0 - 255.255.255.254	Rezervate pentru utilizări viitoare
0.0.0.0	Specifică o adresă IP necunoscută
255.255.255.255	Segment de broadcast local

Tabelul 1.1 Divizarea spațiului de adrese după funcții

Unui segment de rețea TCP/IP, îi sunt atribuite un set de adrese unicast globale și unice prin atribuirea unei *adrese prefix* unicast segmentului. Gazdelor și interfețelor ruterelor aferente segmentului le sunt apoi alocate adrese unicast din setul respectiv. De exemplu, pentru segmentul de rețea din figura 1.2 a fost atribuit intervalul de adrese 128.186.1.0-128.186.1.255, care este, de asemenea, reprezentat ca adresă prefix 128.186.1.0/24; 24 indică faptul că toate adresele din segmentul respectiv sunt în primii 24 de biți. Acest lucru este, de asemenea, uneori scris ca o pereche adresă - mască [128.186.1.0,255.255.255.0], cu biții 1 din mască reprezentând acei biți din adresă care sunt constanți pentru toate adresele segmentului. Gazda și cele două rutere din figura 1.2 au primit adrese din prefixul respectiv pentru interfețele lor care se conectează la segment (128.186.1.1 - gazda,128.186.1.253 și 128.186.1.254 cele două rutere).

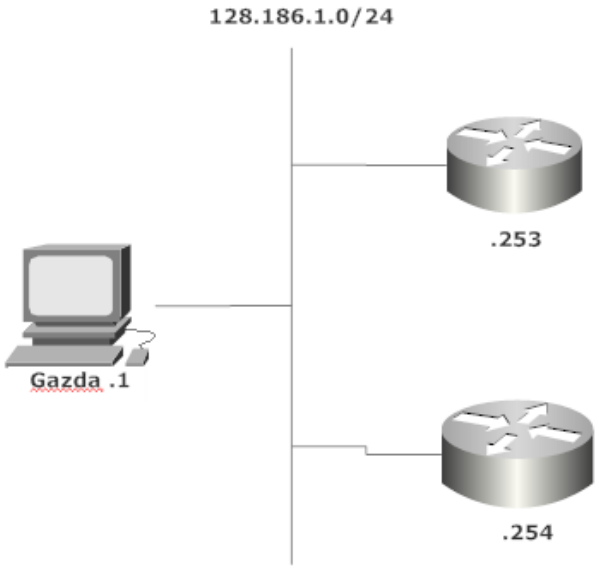


Figura 1.2 Exeplu de atribuire de adrese IP

Cea mai mare adresă dintr-un prefix este atribuită ca adresă de broadcast pentru acest segment. De exemplu, 128.186.1.255 este adresa de broadcast pentru segmentul de rețea din figura 1.2. Un pachet trimis la această adresă va fi livrat la toate gazdele și la toate ruterele atașate la segment.

Clasa	Primul octet în binar	Prima adresă	Ultima adresă
A	0xxxxxxx	0.0.0.1	127.255.255.255
B	10xxxxxx	128.0.0.0	191.255.255.255
C	110xxxxx	192.0.0.0	223.255.255.255
D	1110xxxx	224.0.0.0	239.255.255.255
E	11110xxx	240.0.0.0	255.255.255.255

Tabelul 1.2 Împărțirea spațiului de adrese pe clase

Așa cum am menționat, ruterele TCP/IP rutează către segmente de rețea. Atunci când transmite un pachet, un ruter caută în tabela sa de rutare, pentru a găsi segmentul din care aparține adresa destinație a pachetului și apoi transmite pachetul spre segmentul găsit. Cu toate acestea, este mult mai corect să spunem că ruterele rutează către prefixe. Ruterele nu pot ține urma fiecărui

segment din Internet. În schimb informațiile despre adresarea unui segment sunt agregate în prefixe mai scurte în anumite puncte din Internet. Ruterele trimit pachete către aceste prefixe mai scurte, care e posibil să fie cumulate de trei sau chiar patru ori față de prefixele segmentului original, care sunt păstrate în tabela de rutare a ruterului.

De-a lungul timpului, anumite adrese IP unicast au primit semnificație specială, așa cum se poate observa în tabelul 1.3.

Prefixul	Intervalul de adrese	Scopul rezervat
10/8	10.0.0.0 - 10.255.255.255	Adrese private din Internet
127/7	127.0.0.0 - 127.255.255.255	Adrese de loopback
172.16/12	172.16.0.0 - 172.31.255.255	Adrese private din Internet
192.168/16	192.168.0.0 - 192.168.255.255	Adrese private din Internet

Tabelul 1.3 Scopurile speciale ale adreselor IP unicast

1.5 Tabele de rutare

Toate protocoalele de rutare TCP/IP au diferite moduri de a descoperi prefixele adreselor IP accesibile și pentru fiecare prefix, ruterul next hop pe care îl vor utliza pentru a transmite traficul de date către prefixul respectiv. Când apar modificări în rețea - linii închiriate căzute, linii închiriate nou adăugate, rutere defecte și așa mai departe - protocoalele de rutare reevaluează continuu accesibilitatea prefixelor și next hop-ul utilizat pentru fiecare prefix. Procesul de a găsi un nou next hop după ce au avut loc modificări în rețea se numește *convergență*. Sunt preferate protocoalele de rutare care găsesc noul next hop rapid, ceea ce înseamnă, protocoale cu un *timp de convergență* scurt. Cu toate acestea, pentru orice protocol de rutare, cu cât dimensiunea și complexitatea rețelei crește, același lucru se întâmplă și cu timpul de convergență.

Tabela de rutare a unui ruter instruiește ruterul cum să transmită pachetele. Pentru fiecare pachet, ruterul efectuează o căutare în tabela de rutare, folosind adresa IP destinație a pachetului drep cheie de căutare în tabelă. Această căutare returnează cea mai potrivită înregistrare din tabela de rutare, care îi spune ruterului pe care interfață să transmită pachetul și adresa IP a next hop-ului.

În tabela de rutare există o înregistrare separată pentru fiecare prefix al unei adrese pe care o știe ruterul. Intrările din tabela de rutare, în mod obișnuit, sunt denumite *rute*.

De exemplu, ruterul C din Figura 1.2 are tabela de rutare ilustrată tabelul 1.1.

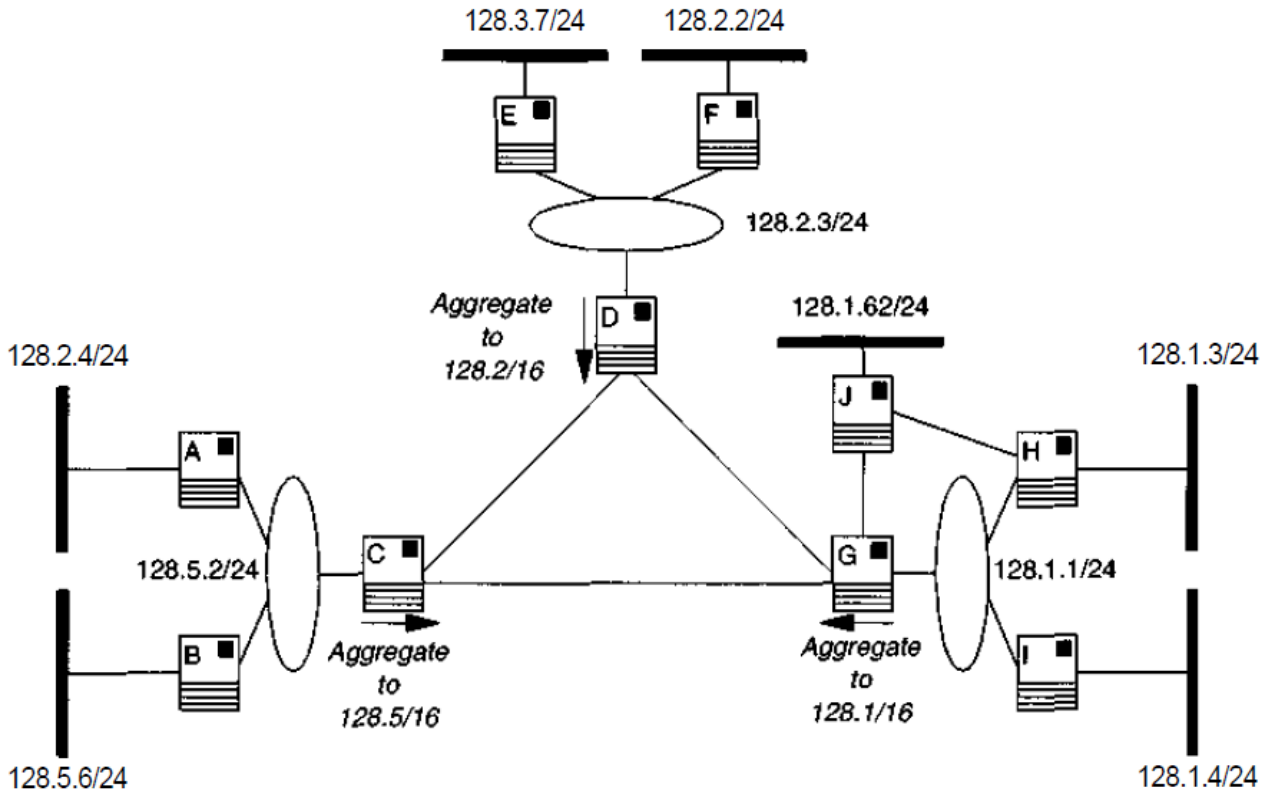


Figura 1.3 Rețea simplă TCP/IP care ilustrează agregarea prefixelor

Prefix	Next Hop
128.1/16	Ruterul G
128.2/16	Ruterul D
128.2.4/24	Ruterul A
128.3.7/24	Ruterul D
128.5.2/24	El însuși
128.5.6/24	Ruterul B

Tabelul 1.4 Tabela de rutare a ruterului C din Figura 2.1.

Următorul hop "El însuși" pentru prefixul 128.5.2/24 indică faptul că acest segment este direct conectat ruterlui C; C va livra pachetele destinate acestui prefix direct către destinația lor. Trebuie reținut că, deși există un segment de rețea cu prefixul 128.2.2/24, acest prefix este ascuns față de C datorită prefixului agregat 128.2/16 anunțat de ruterul D. În mod similar ruterul G din partea dreaptă a figurii 1.2 ascunde o serie de prefixe specifice ruterului C anunțând un singur prefix agregat 128.1/16.

Dacă IP-ul destinație a unui pachet se încadrează în intervalul de adrese descrise de un prefix dintr-o anumită înregistrare din tabela de rutare, spunem că înregistrarea este o *potrivire (match)*. În exemplul nostru, înregistrarea 128.3.7/24 din tabelul 1.4. cuprinde toate destinațiile de forma 128.3.7.x. În exemplul nostru, multe destinații nu au nicio înregistrare care să se potrivească (28.4.56.77, 192.9.1.3, 11.11.11.11 reprezintă câteva exemple). În cazul în care ruterul C primește un pachet adresat uneia dintre aceste destinații, ruterul pur și simplu aruncă pachetul și încearcă să trimită un mesaj ICMP Destinație Inaccesibilă (ICMP Destination Unreachable) înapoi la sursa pachetului pentru a o informa despre eroarea apărută.

În tabelul 1.4, destinația 128.2.4.5 se potrivește cu două intrări din tabela de rutare: 128.2/16 și 128.2.4/24. Acest lucru nu este deloc surprinzător. În aceste cazuri, înregistrarea cu prefixul cel mai lung, 128.2.4/24, este selectată ca cea mai bună potrivire - *best match* (uneori, menționată ca cea mai lungă potrivire - *longest match*); spunem, de asemenea, că prefixul 128.2.4/24 este mai specific decât prefixul 128.2/16. O altă modalitate când apar multiple potriviri este atunci când se ultilizează modelul de adresare al furnizorului de servicii. O organizație ia o gamă de adrese de la furnizorul său de servicii Internet iar mai târziu schimbă furnizorul dar nu vrea să schimbe și gama de adrese. Apoi, noul ISP, trebuie să anunțe o rută mai specifică pentru o parte din spatiul de adrese al vechiului furnizor de servicii.

Multe rutere au o rută implicită - *default route* în tabela lor de rutare. Aceasta este o înregistrare în tabela de rutare pentru prefixul de lungime zero 0/0. Ruta implicită potrivește toate destinațiile, deși aceasta este suprascrisă de către toate prefixele mai specifice. De exemplu, să presupunem că rețeaua locală a unei organizații are un ruter atașat la rețeaua publică Internet. Toate celelalte rutere ale organizației pot folosi mai curând o rută implicită care indică către conexiunea la Internet decât să știe toate rutele disponibile în Internet.

Fiecare ruter are o tabelă de rutare diferită, care reflectă poziția sa unică în cadrul rețelei Internet. De la un ruter la altul, nu numai next hop-urile pentru intrările din tabela de rutare se schimbă dar chiar și prefixele pe care fiecare ruter le cunoște pot fi diferite, deoarece adresele sunt agregate în prefixe mari. În figura 1.2, ruterele D, E, F cunosc prefixul 128.2.2/24 dar toate celelalte rutere văd doar prefixul agregat 128.2/16. Ruterele de la granițele Internetului pot să se bazeze foarte mult pe înregistrarea implicită din tabela de rutare, cu doar câteva sute de intrări mai specifice în tabela lor de rutare. Cu toate acestea, ruterele de bază din Internet conțin în tabela de rutare peste 500000 de înregistrări. Aceste rutere, care nu folosesc o rută implicită, dețin tabela de rutare completă a Internetului. Trendul dimensiunii complete a tabelei de rutare din Internet este prezentat în figura 1.3.

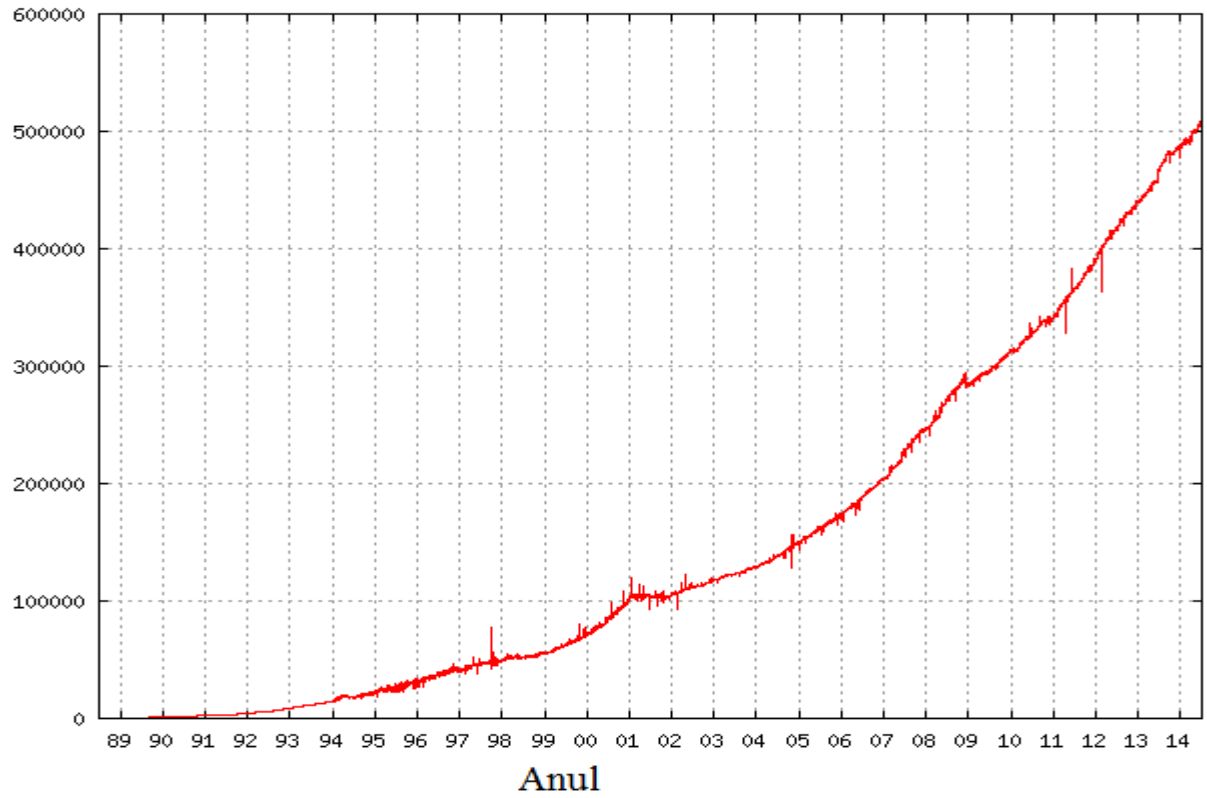


Figura 1.4 Ritmul de creștere a tabelelor de rutare din Internet (număr rute)

În exemplul nostru, presupunem că ruterul C își construiește tabela de rutare prin intermediul protocoale de rutare dinamice - prefixele agregate 128.1/16 și 128.2/16 sunt anunțate ruterului C (de către ruterele G și D), folosind același protocol așa cum sunt anunțate și rutele spre alte segmente îndepărtate. Protocoalele de rutare dinamice sunt cu siguranță etalonul, deși ele nu trebuie să fie utilizate obligatoriu. În schimb, înregistrările din tabela de rutare pot fi configurate explicit de către administratorul rețelei, acest lucru numindu-se *rutare statică*.

A avea o rețea funcțională care folosește doar rutare statică este foarte dificil de administrat, necesitând modificarea rutelor statice când apar modificări în rețea sau chiar în timpul unei defecțiuni a rețelei. În exemplul nostru, dacă legătura dintre ruterul C și ruterul D cade, administratorul trebuie să îi spună ruterului C să trimită pechetele destinate 128.3.7/24 prin ruterul G în schimb același lucru un protocol de rutare dinamic l-ar face automat. Totusi, rutarea statică este încă folosită astăzi în Internet pentru a spori rutarea dinamică. De exemplu, este posibil ca două organizații să nu aibă încredere una în cealaltă pentru a folosi rutare dinamică la granița dintre ele dar în schimb se bazează pe un set de rute statice.

1.6 Implementarea tabelelor de rutare

Performanța de transmitere, de obicei exprimată în pachete/secundă, este una dintre cela mai importante caracteristici a unui ruter. Prin urmare producătorii de rutere încearcă să organizeze tabelele de rutare astfel încât cea mai bună operație de potrivire să fie efectută cât de repede posibil.

O modalitate comună de aranjare a tabelei de rutare este arborele Patricia (*Patricia tree*) - o categorie specială a arborelui radix (*radix tree*) care minimizează comparațiile între biți iar când este folosit ca tabelă de rutare necesită o singură operație mască și potrivire (mask-and-match). Acești arbori sunt arbori binari, prin urmare traversarea arborelui depinde de o secventă de comparații cu un singur bit din cheie, cheia fiind adresa IP destinație.

Un arbore Patricia care poate fi folosit pentru a implementa căutarea pentru tabela de rutare din tabelul 1.4 este ilustrat în figura 1.4. Să presupunem că ruterul încearcă să trimită un pachet la destinația 128.2.5.6. Ruterul începe din partea de sus a arborelui Patricia testând bitul 13 din adresa destinație (biții sunt numerotați începând cu 0 pentru cel mai semnificativ bit din adresă). Din moment ce acest bit este 0, ruterul face ramuri spre stânga, testând succesiv biții 14 (acesta fiind 1), 15 (acesta fiind 0), și 21 (acesta fiind 0). Când testarea ajunge la apariția care specifică că nu se mai fac comparații pe bit traversarea se oprește. Deasemnea, traversarea se oprește dacă noul bit de testat este mai mic sau egal decât ultimul bit testat. Când traversarea se oprește, destinația este comparată cu prefixul din înregistrare pentru a vedea dacă este o potrivire între acetea. În exemplul nostru, 128.2.5.6 nu se potrivește cu 128.2.4/24. Ruterul trebuie să verifice potriviri cu prefixe mai scurte decât 128.2.4/24, urmărind pointer-ul prefixului înapoi la 126.2/16 care produce cea mai bună potrivire cu o înregistrare din tabela de rutare.

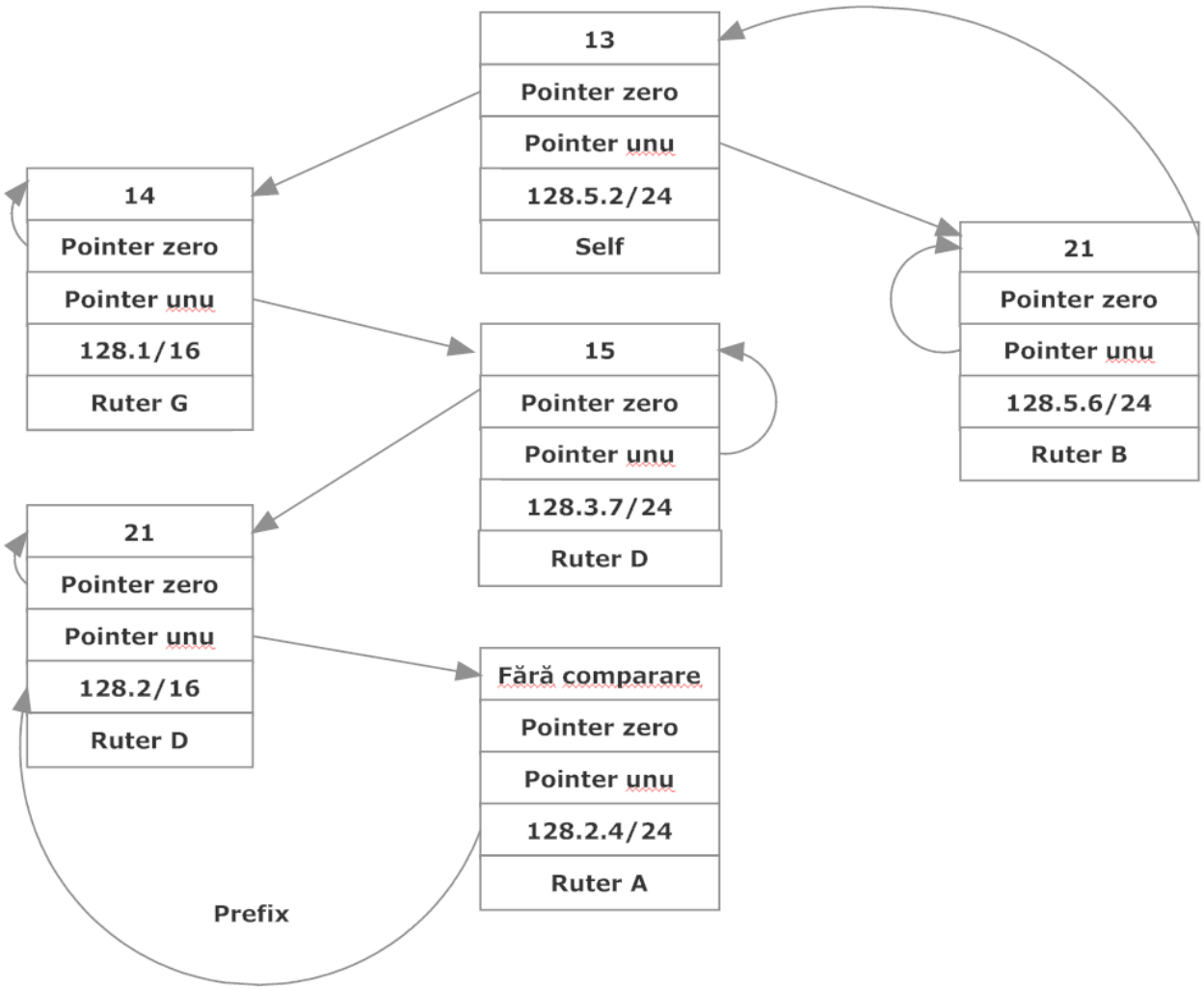


Figura 1.5 Implementarea arborelui Patricia pentru căutarea în tabela de rutare

Cu o tabelă de rutare care conține peste 500000 de înregistrări ruterele vor efectua în medie 16 comparații de biți pentru fiecare căutare. Totuși, performanța arborelui Patricia este oarecum dependentă de date. Cu o colecție particulară de prefixe supărătoare în tabela de rutare căutarea unei anumite adrese poate conduce până la 32 de comparații de biți, câte o comparație pentru fiecare bit din adresa IP destinație.

Chiar dacă variantele de arbori radix, cum este arborele Patricia, sunt mult mai comune și alți algoritmi de căutare în tabela de rutare sunt implementați în ruterele TCP/IP.

O altă modalitate de a mări viteza de căutare în tabela de rutare este reprezentată de încercarea de a evita căutarea în totalitate. Căutarea în tabela de rutare furnizează next hop-ul pentru o adresă IP dată. Unele rutere depozitează această asociere adresă IP destinație - next hop într-o bază de date separată care este consultată înainte de a se efectua căutara în tabela de rutare (aceasta poate fi asociată ca *the front end* pentru tabela de rutare). Găsirea unei anumite adrese în această bază de date este mai ușoară deoarece are loc o singură potrivire exactă în shimbul unei operații mai complicate de căutare a celei mai bune potriviri în tabela de rutare. Ca un exemplu concret această bază de date front-end poate fi organizată ca o *tabelă hash*. Dacă funcția hash ar fi suma biților trei și patru din IP-ul destinație o pereche de buckets din tabela hash ar arăta ca în figura 1.5 după ce ruterul a trimis câteva pachete. Acum dacă ruterul vede iarăși oricare dintre aceste destinații (un cache hit) căutarea lor va fi foarte rapidă. Totuși, pachetele trimise către destinații noi vor fi mai încete deoarece costul unei căutări eșuate în tabela hash va fi adăugat costului căutării normale în tabela de rutare.

Tabelele cache front-end ale tabelelor de rutare pot lucra bine la granița Internetului sau în interiorul organizațiilor. Totuși, se pare că tabelele cash nu funcționează bine la ruterele din nucleul

Internetului. Numărul mare de destinații a pachetelor văzute de ruterele din nucleu pot face ca căutarea în tabela cache să fie mai înceată decât căutarea în tabela de rutare.

Chiar dacă adresare IP datează de pe 20 de ani noi algoritmi de căutare în tabela de rutare sunt încă dezvoltați în încercarea de a construi rutere mai rapide.

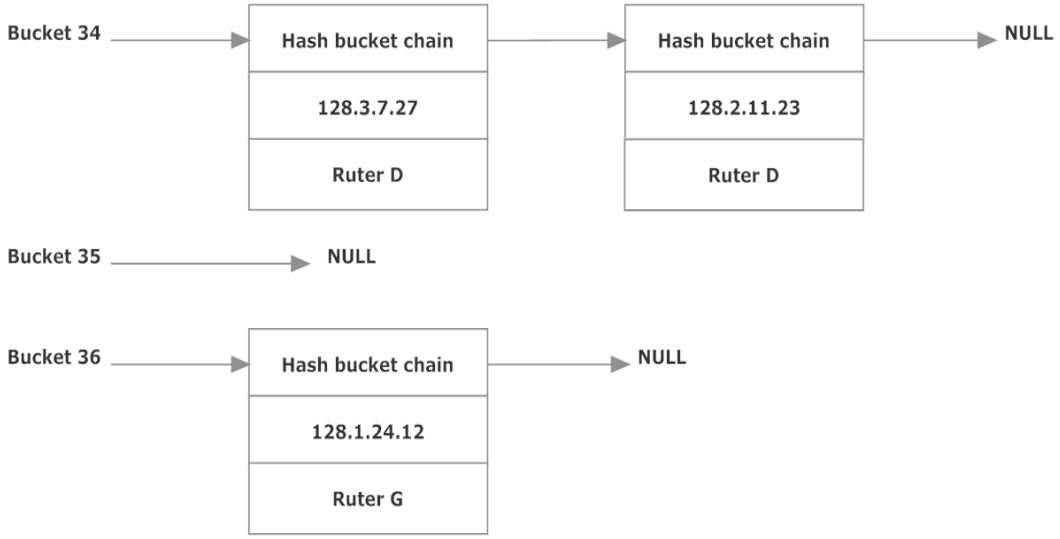


Figura 1.6 Tabela hash front-end a tabelei de rutare

2. Protocoale de rutare în Internet

2.1 Algoritmi vectori distanță (Distance Vector Algorithms)

În cadrul unui protocol vector distanță ruterele cooperează pentru efectuarea unui calcul distribuit. Algoritmii vectori distanță calculează separat cea mai bună cale pentru fiecare prefix destinație, de obicei încearcă să găsească rutele care minimizează o metrică simplă, cum ar fi numărul de hop-uri până la destinație. La fiecare pas intermediar din algoritm, fiecare ruter are ruta sa actuală cea mai bună către prefixul destinație. În acest moment ruterul anunță toți vecinii săi despre ruta sa actuală; totodată și vecinii anunță ruterul despre rutele pe care le-au ales ei. Ruterul, cunoscând rutele folosite de către toți vecinii săi, poate găsi o cale mai bună (cu un cost mai mic) printre rutele vecinilor. Dacă se întâmplă acest lucru, ruterul își actualizează next hop-ul și costul pentru această destinație și își anunță vecinii de noua rută aleasă și astfel procedura iterează. După un număr de iterații, alegerea acestei rute se va stabiliza, fiecare ruter găsind astfel cea mai bună cale către destinație.

Principalul avantaj al algoritmilor vectori distanță este simplitatea. Deasemenea, acești algoritmi pot fi supuși agregării rutelor și pot fi utilizați ușor alături de anumite politici simple de rutare (de exemplu un ruter poate ruta o categorie de prefixe printr-un anumit ruter dar altele nu).

Cel mai important exemplu de protocol vector distanță este *Routing Information Protocol (RIP).*

2.1.1 Convergența algoritmilor vectori distanță

Ruterele RIP adaugă fiecare înregistrare destinație în tabela de rutare cu un câmp asociat numit *metrică*. Această metrică indică numărul minim de hop-uri necesare pentru a ajunge la destinație. Prefixele pentru segmentele de rețea direct conectate au întotdeauna metrica setată 1. La fiecare 30 de secunde, fiecare ruter RIP trimite actualizări broadcast către toate ruterele RIP vecine. Aceste actualizări conțin prefixele din tabela de rutare a ruterului alături de metricile asociate. Când un ruter RIP primește o actualizare RIP de la vecinul X, ruterul examinează toate prefixele din actualizare. Dacă numărul de hop-uri necesar pentru a ajunge la un anumit prefix trecând prin X (obținut adăugând 1 la metrica anunțată de X pentru prefixul respectiv) este mai bun decât metrica pe care o are ruterul pentru acel prefix atunci ruterul își actualizează tabela de rutare. Inregistrarea actualizată din tabela de rutare are next hop-ul setat către X și metrica cu unu mai mare decât metrica anunțată de X.

Exemplul ilustrat în tabelul 2.1 presupune că la momentul T1 interfața ruterului I dinspre 192.1.4/24 din figura 2.1 tocmai a devenit operațională și că la momentele T1-T4 fiecare dintre ruterele A-I trimite simultan actualizări de rutare către vecinii lor. Coloana de sub A reprezintă intrarea din tabela de rutare pentru 192.1.4/24 a ruterului A odată cu trecerea timpului. După momentul T4 intrările din tabela de rutare rămân stabile iar protocolul RIP a convers.

Tabelul 2.1 ilustrează ce se întâmplă când o nouă cale mai scurtă către un prefix destinație este descoperită. Totuși, uneori, ruta actuală cea mai scurtă nu mai este disponibilă datorită unor defectări a legăturii sau a ruterului iar rutarea trebuie să întoarcă pe o cale mai lungă. Un ruter descoperă acestă defectare în două moduri. Uneori cel mai bun hop actual al ruterului va trimite doar o actualizare prin care spune că cea mai bună cale a devenit mai lungă. În alte cazuri, din lipsa de actualizări primite de la actualul next hop ruterul își dă seama că next hop-ul nu mai este operațional și astfel ruterul alege un nou next hop anunțând o cale egală sau chiar mai lungă.

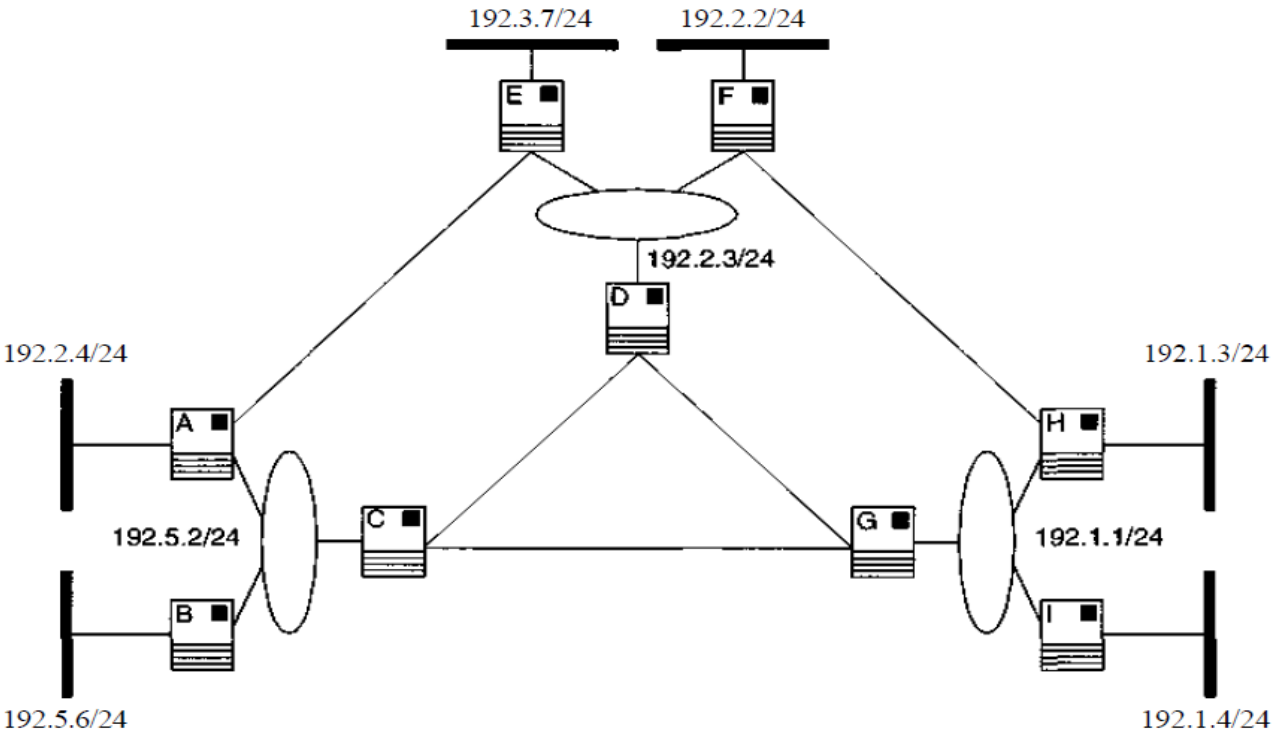


Figura 2.1 Topologie de rețea simplă care ilustrează comportamentul protocolului vector distanță

Momentul de timp	A	B	C	D	E	F	G	H	I
T1	∞	∞	∞	∞	∞			∞	1
T2	∞	∞	∞	∞			2, 1	2, 1	1
T3			3, G	3, G		3, H	2, 1	2, 1	1
T4	4, C	4, C	3, G	3, G	4, D	3, H	2, 1	2, 1	1

Tabelul 2.1 Convergența protocolului RIP când subrețeaua 192.1.4/24 este adăugată figurii 2.1

Numele protocoalelor de rutare vectori distanță provine din forma pe care o iau actualizările lor de rutare: o *listă* (sau *vector*) de *metrici* (sau *distanțe*) pentru fiecare prefix anunțat. Dacă ne uităm la calculul distribuit pe care un algoritm vector distanță îl efectuează peste rețea, cum este ilustrat în tabelul 2.2, vom observa că reprezintă o aplicație distribuită a algoritmului Bellman-Ford pentru găsirea celor mai scurte rute. Din acest motiv, algoritmii vectori distanță sunt adesea numiți algoritmi Bellman-Ford (chiar dacă acest lucru este puțin confuz deoarece și algoritmii care folosesc starea legăturilor pot utiliza algoritmul Bellman-Ford pentru a efectua calculele necesare rutării).

2.1.2 Probleme de convergență ale protocoalelor vectori distanță

Calculul distribuit a unui protocol vector distanță este destul de robust dar converge după anumite schimbări în rețea chiar dacă actualizările de la diferite rutere nu sunt sincronizate sau dacă ruterele folosesc intervale variabile pentru actualizări. Totuși, protocoalele vectori distanță când întâmpină anumite defectări ale rețelei pot lua o perioadă lungă de timp pentru a converge. Să presupunem, de exemplu, că interfața ruterului I către 192.1.4/24 din figura 2.1 devine neoperațională. Tabelul 2.2 ilustrează comportamentul protocolului RIP în acest caz, facând din nou presupunerea că fiecare dintre ruterele A-I trimit actualizări sincron la momentele de timp T1-T15.

Momentul de timp	A	B	C	D	E	F	G	H	I
T1	4, C	4, C	3, G	3, G	4, D	3, H	2, 1	2, 1	∞
T2	4, C	4, C	3, G	3, G	4, D	3, H	3, H	3,G	3,G
T3	4, C	4, C	4, G	4, G	4, D	4, H	4, H	4, G	4, G
T4	5, C	5, C	5, G	5, G	5, D	5, H	5, H	5, G	5, G
T5	6, C	6, C	6, G	6, G	6, D	6, H	6,H	6, G	6, G
T6	7, C	7, C	7, G	7, G	7,D	7, H	7, H	7, G	7, G
T7.....T13									
T14	15, C	15,C	15, G	15, G	15, D	15, H	15, H	15, G	15, G
T15	∞	∞	∞	∞	∞	∞	∞	∞	∞

Tabelul 2.2 Convergența vector distanță când subrețeaua 192.1.4/24 este ștearsă din figura 2.1

După cum se poate observa, este nevoie de puțin timp până când ruterele se decid că nu pot ajunge la prefixul 192.1.4/24. În acest timp, buclele de rutare se pot forma foarte ușor; în exemplul nostru, se formează o buclă de rutare între ruterele G și H din momentul T2 până la momentul T14. Trebuie notat că pentru fiecare ruter costul prefixului 192.1.4/24 crește continuu până când atinge limita 16 care reprezintă limita infinit a protocolului RIP. Acest comportament este denumit *numărare la inifinit (couting to infinity)* și din acest motiv costul maxim al unei rute pentru protocoalele vectori distanță este setat la o valoare mică. Cu cât este mai mare costul maxim pentru o cale cu atât poate dura mai mult numărarea la infinit astfel consumând lățime de bandă și procese ale procesorului. Protocoalele vectori distanță, cum este RIP, prezintă aceleași proprietăți atunci când o defectare a rețelei prelungește ruta către o anumită destinație. În acest caz, comportamentul general al convergenței poate fi caracterizat în următorul mod: pe drumul spre noua valoare a metricii, înregistrarea din tabela de rutare a ruterului ia în calcul toate lungimile, ale tuturor rutelor posibile care existau în domeniul de rutare înainte de defectare.

2.1.3 Îmbunătăţirea convergenţei protocoalelor vectori distanţă

Pentru a îmbunătăți convergența protocoalelor vectori distanță, pe parcursul anilor au fost implementate multe modificări. Principalele modificări aduse protocoalelor vectori distanță sunt:

- Pentru a reduce timpul de convergență, multe rutere care rulează protocoale vectori distanță trimit actualizări imediat ce tabelele lor de rutare se modifică și nu așteaptă următorul interval de 30 de secunde pentru a trimite actualizările. Aceste actualizări se numesc *actualizări declanșate (triggered updates)*.
- Pentru a preveni buclele de rutare în timpul convergenței, protocoalele vectori distanță de obicei folosesc o procedură numită *split horizon.* În cadrul procedurii split horizon, când un ruter trimite o actualizare de rutare pe interfața X, ruterul omite toate rutele din actualizăre a căror interfață de ieșire este egală cu X. De exemplu, la momentul T1 din tabelul 2.2, ruterul G nu va trimite o actualizare broadcast pentru 192.1.4/24 către H și I. Într-o modificare a procedurii split horizon, numită *infinite split horizon* sau *split horizon with poison reverse*, actualizările trimise pe interfața X anunță rutele cu interfața de ieșire X fiind inaccesibile (aceasta înseamnă cu un cost 16). Infinite split horizon este mai util decât split horizon în reducerea buclelor de rutare, în fapt reduce complet buclele formate numai din două rutere. Totuși, deoarece infinite split horizon mărește dimensiunea actualizărilor split horizon este mai folosit.
- Într-o altă modificare, numită *hold down*, un ruter refuză să accepte actualizări pentru o rută pentru o anumită perioadă de timp după ce inițial ruta a fost declarată inaccesibilă. De exemplu, dacă pe ruterul I din tabelul 2.2 funcționează hold down, el

va refuza să accepte actualizări de rutare de la vecinii săi pentru câteva perioade de timp după momentul T1. Hold down pot bloca formarea buclelor de rutare în anumite situații, chiar dacă astfel prelungește timpul de convergență. Din acest motiv nu este foarte des implementat în protocoale precum RIP dar este folosit în alte protocoale vectori distanță precum EIGRP (*Enhanced Interior Gateway Routing Protocol*) și DVMRP (*Distance Vector Multicast Routing Protocol*).

Aceste modificări pot îmbunătăți proprietățile de convergență a protocoalelor vectori distantă dar nu pot elimina în totalitate formarea buclelor de rutare în timpul convergenței sau comportamentul de numărare la infinit. Există două modalități cunoscute de a rezolva aceste probleme din cadrul unui protocol vector distanță. Prima metodă, constă în anuțarea rutei complete (aceasta înseamnă întreaga secvență de rutere) pentru fiecare prefix destinație și nu doar metrica prefixului. A două metodă, implementată de către EIGRP, constă din controlarea strictă a ordinii actualizărilor de rutare dintre noduri. A doua metodă poate garanta rutarea fără bucle dar în general complică protocolul semnificativ și poate duce la creșterea timpului de convergență.

2.2 Algoritmi care folosesc starea legăturilor (Link-state algorithms)

În schimbul calculului gradual și distribuit folosit de algoritmii vectori distanță, algoritmii de rutare cu starea legăturilor implică o abordare replicată a bazelor de date distribuite (*replicated distributed database*). Fiecare ruter dintr-un algoritm care folosește starea legăturilor contribuie cu piese la această bază de date prin descrierea mediului său local: setul de legături active către segmente locale de rețea IP și către ruterele vecine, fiecărei legături fiindu-i atribuite un cost. De aici provine și numele algoritmilor care folosesc starea legăturilor: în schimbul anunțării unui set de distanțe către fiecare destinație cunoscută un ruter care rulează un algoritm cu starea legăturilor anunță starea legăturilor sale de rețea locale. Aceste anunțuri care conțin starea legăturilor sunt distribuite către toate ruterele. Rezultatul final este acela că toate ruterele obțin aceeași bază de date a colecției de anunțuri care împreună descriu harta actuală a rețelei. Costul unei rute din rețea constă din suma costurilor legăturilor care formează ruta. Din harta rețelei, fiecare ruter calculează cea mai scurtă rută pentru fiecare prefix destinație folosind algoritmul lui Dijkstra, chiar dacă și alți algoritmi precum Bellman-Ford sunt folosiți uneori.

Algoritmii cu starea legăturii sunt în general considerați a avea proprietăți de convergență mai bune: când rețeaua se modifică noile rute sunt găsite ușor cu un minim de suprasarcini pentru protocolul de rutare. Din moment ce bazele de date a protocoalelor care folosesc starea legăturilor au mai multe date la dispoziție (și anume o întreagă descriere a rețelei) decât protocoalele vectori distanță ele pot calcula mai ușor rute cu caracteristici mai complexe și nu rute simple care au cel mai mic cost. De exemplu, protocoalele care folosesc starea legăturilor au fost proiectate pentru Internet să calculeze rute separate pentru fiecare tip de serviciu IP, rute care urmează o gamă largă de politici de constrângeri sau rute care pot oferi anumite garanții de calitate a serviciilor.

Protocoalele care folosesc starea legăturilor sunt cu siguranță mai complicate decât protocoale vectori distanță după cum se poate observa și din compararea dimensiunilor specificațiilor protocoalelor OSPF și RIP. Totuși, putem modifica un algoritm vector distanță pentru a obține câteva dintre proprietățile standard ale algoritmilor cu starea legăturilor (rezistență la bucle de rutare, trasmiterea doar a modificărilor de rutare în schimbul actualizărilor repetate) rezultând astfel un protocol care este la fel de greu de implementat ca și un protocol cu starea legăturilor.

Ramificarea în EGP/IGP permite Internetului să utilizeze cele mai bune caracteristici ale ambelor tipuri de protocoale. De exemplu, folosind OSPF ca IGP permite o convergență locală mai rapidă în timp ce folosirea BGP între sisteme autonome facilitează agregarea rutelor și rutarea bazată pe politici de rutare. Desigur, aceste două tehnologii pot fi amestecate într-un singur protocol - ierarhiea pe două nivele din OSPF, un protocol cu starea legăturilor care folosește mecanisme vectori distanță.

3. Protocolul OSPF - Open Shortest Path First

3.1 Prezentare generală a protocolului OSPF

Internet Engineering Task Force (IETF) a început în 1987 cercetarea la un protocol de rutare numit OSPF pentru a înlocui faimosul, pentru perioada respectivă, protocol de rutare vector distanță numit RIP. RIP consuma multă lărgime de bandă în timpul trimiterii actualizărilor iar timpul său de convergență nu mai funcționa bine odată cu creșterea rețelor IP din acea perioadă. Dezvoltarea protocolului OSPF este determinată în pricipal de eșecul protocolului RIP în rezolvarea problemelor legate de creșterea rețelelor IP. Dezvoltarea protocolului OSPF a început cu scopul de a obține un timp de convergență mai rapid care consumă mai puțină lărgime de bandă în timpul actualizărilor.

OSPF este un protocol de rutare dinamic care folosește starea legăturilor și care interconectează rutere din interiorul unei rețele prin schimbul de informații. Dinamica protocolului provine din faptul că își menține tabela de rutare cu toate rutele posibile și o actualizează periodic. El efectuează un calcul în timp real asupra schimbărilor din rețea. Este un protocol de rutare care folosește starea legăturilor deoarece fiecare ruter urmărește starea legăturilor din rețea. Este un Interior Gateway Protocol (IGP) deoarece operează doar în interiorul unui singur AS și este unul din cele mai utilizate IGP din rețelele întreprinderilor mari. Acesta operează folosind stiva TCP/IP și este încapsulat cu numărul de protocol 89. Pentru a schimba informațiile de rutare între ruterele din rețea folosește IP multicast.

Ruterele configurate cu OSPF într-o rețea sunt capabile să învețe dinamic rute către alte rutere din rețea și astfel vor împărți informația învățată între ele. Principalul scop al împărțirii informației despre starea legăturilor între ruterele dintr-o rețea OSPF este de a construi și de a menține aceeși bază de date cu starea legăturilor (*LSDB - link state database*) a întregii topologii pe fiecare ruter. Acest lucru reprezintă o necesitate într-o rețea OSPF. LSDB conține o colecție de rute și interfețele active corespunzătoare cât și costul asociat pentru a folosi fiecare legătură activă de pe un ruter. Transformându-se într-un nod rădăcină fiecare ruter își construiește arborele cu cele mai scurte rute (shortest path tree - SPT) către fiecare nod disponibil din rețea. SPT este construit folosind algoritmul lui Dijkstra. Algoritmul lui Dijkstra folosește cel mai mic cost ca fiind cea mai bună metrică. Costul asociat fiecărei legături este folosit pentru a oferi o anumită prioritate între diferitele rute din rețea. În final, fiecare ruter își construiește tabela de rutare folosind acest arbore.

După ce și-a construit tabela de rutare un ruter OSPF menține o tabelă de rutare sincronizată cu toți vecinii săi. Într-o rețea OSPF protocolul *hello* este folosit pentru a descoperi vecinii și pentru a menține conectivitatea între vecini. Parametrul *HelloInterval* trebuie să fie același în toată rețeaua și este folosit în combinație cu parametrul *HelloDeadInterval.* Intervalul implicit folosit pentru a trimite pachetele *hello* este de 10 secunde pentru o legătură Ethernet și de 30 de secunde pentru o legătură non broadcast. Dacă pachetele *hello* nu sunt primite de un ruter în intervalul *HelloDeadInterval,* care de obicei este 4**HelloInterval,* ruterul consideră că acel vecin este căzut. Această schimbare se reflectă prin inundarea întregii rețele cu o actualiazare a stării legăturilor (*Link State Update*). Pachetul cu actualizarea stării legăturilor transportă doar infomații despre starea legăturii care s-a schimbat și nu transpotă întreagă LSDB. Inundarea pemite ca toate ruterele să se sincronizeze la această modificare.

3.2 Procesul de convergenţă al protocolului OSPF

3.2.1 Rutarea folosind planul de control şi planul de date a ruterelor

Un ruter este un dispozitiv responsabil de expedierea pachetelor între dispozitivele interconectate. Un protocol de rutare este o regulă care guvernează şi reglementează modul în care ar trebui să fie făcută comunicarea între dispozitivele de comunicare. Ruterele leagă header-ul IP al unui pachet cu înregistrările din tabela de rutare pentru a determina cel mai bun next hop către care să expedieze pachetul. Ruterele moderne au o arhitectură distribuită, care funcţionează în mod independent, mecanismul de rutare (*control plane - planul de control*) şi mecanismul de transmitere a pachetelor (*data plane - planul de date*). Mecanismul de expediere, care este un comutator, este folosit pentru a transmite între interfaţa de intrare şi cea de ieşire a unui card. Prin urmare, planul de control se ocupă de protocolul de rutare iar planul de date se ocupă de funcţiile de expediere.

Planul de control este responsabil de construirea şi menţinerea RIB (RIB - *Routing Information Base*) din care este construită FIB (FIB - *Forward Information Base*); RIB constă din cele mai bune rute pentru a ajunge la diferite noduri dintr-o reţea întreagă. Planul de date efectuează comutarea de pachete, căutările de rute şi transmiterea de pachete. Practic trimite pachetele de la un port la altul, bazându-se pe potrivirea dintre prefixul cel mai lung şi o listă de control a accesului, care filtrează pachetele. FIB conţine cel puţin informaţiile necesare pentru transmiterea pachetelor IP adică next hop şi interfaţa de ieşire asociată.

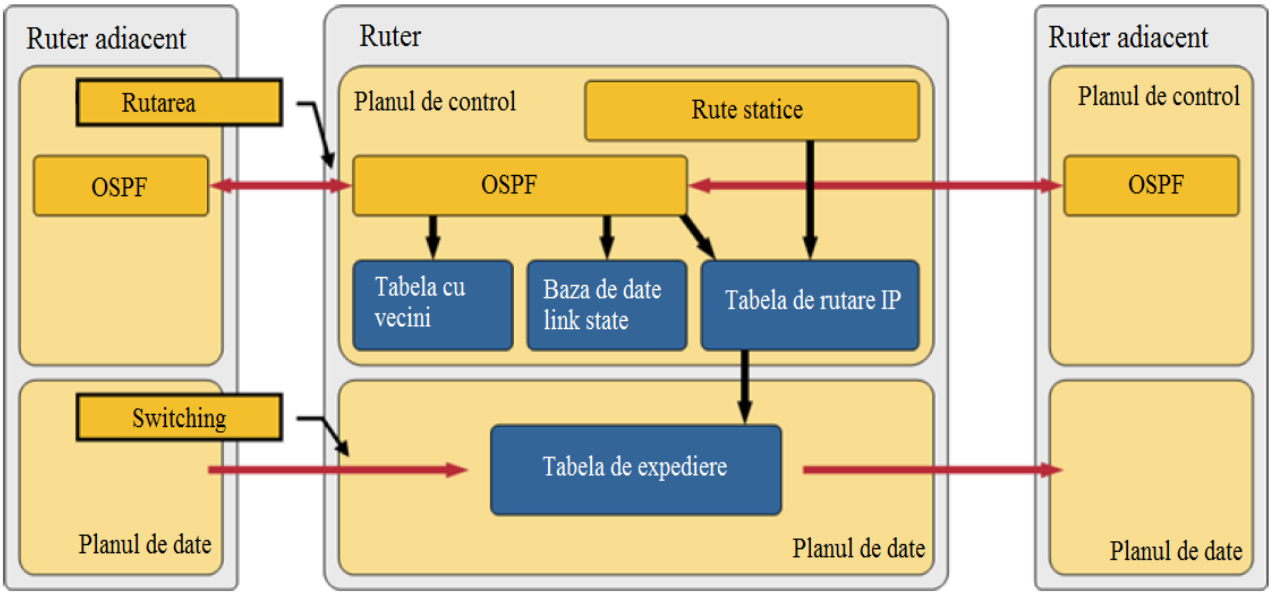


Figura 3.1 Planul de control şi planul de date

Procesul de transmitere include verificarea header-ului IP şi a TTL (*Time to Live*) a pachetului de intrare, diminuarea TTL şi actualizarea câmpului checksum din header-ul IP precum şi căutarea în FIB a adresei IP destinaţie pentru a hotărî adresa next hop-ului. Ruterele efectuază recalcularea SPF pe baza actualizărilor LSA primite iar modificarea se reflectă în tabela de expediere prin intermediul RIB. Natura distribuită a arhitecturii ruterului reduce sarcina pe CPU. De asemenea, poate permite planului de date să continuie expedierea pachetelor spre vecini, chiar şi atunci când mecanismul de control reporneşte pentru unele activităţi, cum ar fi actualizarea software-ului său.

3.2.2 Procesarea pachetelor de intrare într-un ruter

Un pachet de intrare IP, în cele mai multe implementări, este mai întâi procesat de planul de date care după această procesare poate fi transmis spre planul de control sau spre interfeţele de ieşire în funcţie de natura pachetului. După cum se poate vedea în figura 3.2, pachetele destinate altor rutere sunt transmise direct fără a fi nevoie de transmiterea către planul de control. Unele pachete speciale, cum sunt pachetele *hello* şi actualizările de rutare, trebuie să fie transmise la planul de control.

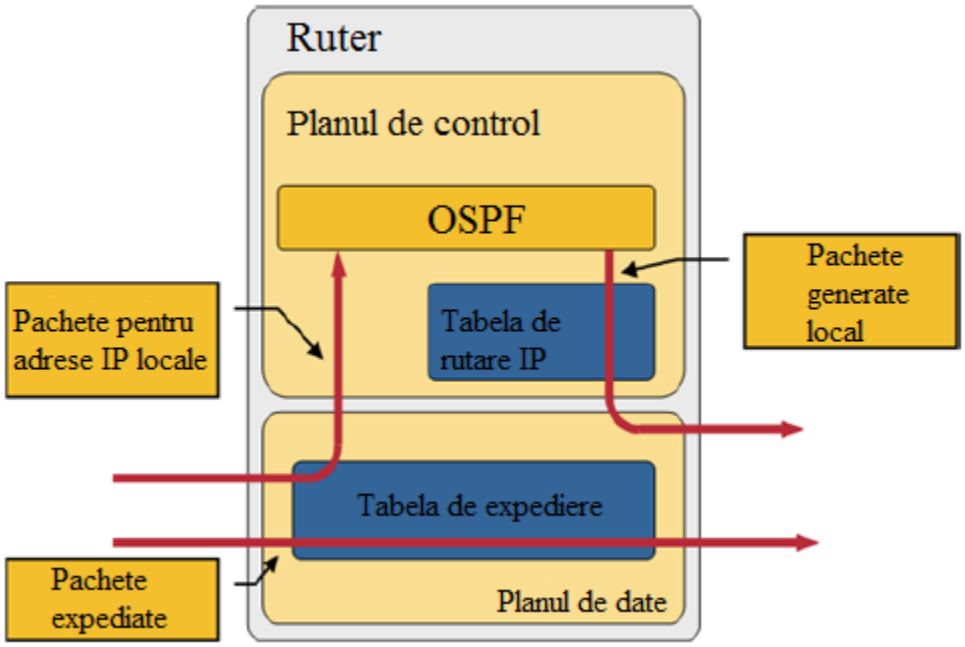


Figura 3.2 Procesarea pachetelor de intrare

3.2.3 Schimbări topologice într-o reţea OSPF

O schimbare topologică într-o reţea IP poate rezulta din erori ale componentele de reţea sau din adăugarea de noi echipamente în reţea. Erorile pot apărea din cauza unor lucrări planificate de întreţinere, probleme software şi/sau hardware sau pot fi datorate unor erori umane. Impactul unei erori poate varia în funcţie de natura sa. Modificările topologice sunt reflectate prin intermediul actualizărilor LSA trimise tuturor celorlalte noduri din reţea. Atunci când apare o eroare a unei legături, aceasta este detectată prin intermediul mecanismelor de bază de detectare a erorilor iar modificarea este reflectată prin actualizarea LSA, care este inundată în întreaga reţea. Calculul SPF este programat până la sosirea actualizărilor într-un nod, ceea ce conduce la scopul final de a instala o nouă rută alternativă în reţea. Un proces de convergenţă OSPF al ruterului porneşte de la momentul în care o eroare este detectată până când o nouă rută alternativă este instalată în ruter. Durata de adaptare la schimbare a ruterului este determinată de timpul de convergenţă al acestuia.

3.2.4 Probleme asociate cu convergenţa OSPF

Modificările topologice într-o reţea sunt de obicei urmate de unul sau mai multe dintre următoarele aspecte, aspecte care durează până la adaptarea la noua schimbare a reţelei. Pierderile de pachete de date sau livrări neordonate, utilizarea excesivă a CPU/memorie din cauza procesării

suplimentare şi calculele tabelei de rutare pot fi menţionate ca exemple. O astfel de utilizare excesivă a resurselor, în cel mai rău caz, ar putea conduce la prăbuşirea întregii reţele.

3.2.5 Procesul de convergenţă OSPF

OSPF este un protocol interesant, deoarece implică un timp redus pentru instalarea unei noi rute şi pentru redirecţionarea traficului la apariţia unui eşec. Procesul de convergenţă OSPF este compus din următoarele procedee.

- Detectarea schimbărilor în starea reţelei;
- Generarea unui nou LSA pentru a reflecta schimbarea;
- Inundarea LSA-ului în reţeaua OSPF;
- Efectuarea calculelor SPF de către fiecare ruter care primeşte informaţiile de actualizare;
- Actualizarea RIB/FIB a fiecărui ruter.

Este o cerinţă pentru toate ruterele afectate de schimbări topologice de a trece prin procesul de convergenţă iar timpul implicat depinde de mărimea şi de complexitatea reţelei. Efectul dorit al procesului de convergență este de a obține un timp de convergență mai scurt. Cum ruterele converg rapid, este relativ uşor de a evita problemele menţionate la punctul 3.2.4.

3.2.6 Adăugarea unui nod într-o reţea convergentă

Presupunând o reţea convergentă, în primul rând, (Figura 3.3), adăugând un nou ruter numit *New_node* (noul nod) rezută o schimbare topologică a reţelei. Ca urmare, reţeaua trebuie să treacă prin paşii de mai jos, astfel încât să menţină o informaţie sincronizată despre topologia reţelei.

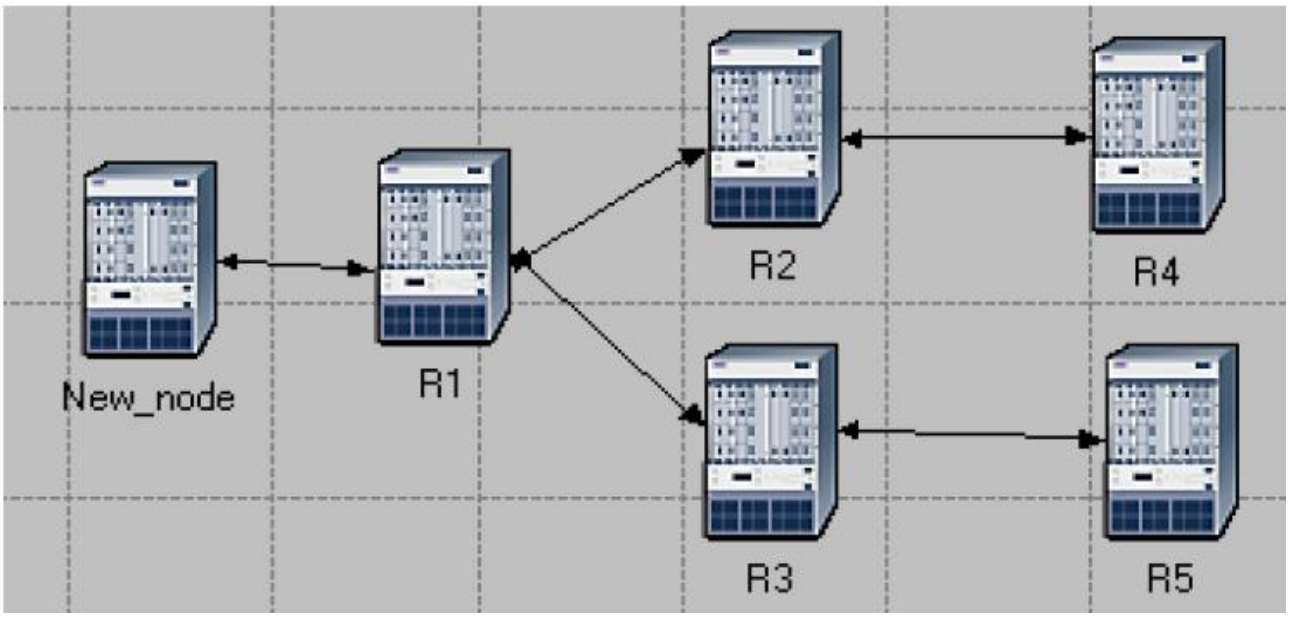


Figura 3.3 Adăugarea unui nod într-o reţea convergentă

În procesul de convergenţă sunt implicaţi următorii paşi:

1. Nodul nou (New_node) trimite un pachet *hello* către R1. R1 răspunde folosind un pachet *hello* pentru a confirma descoperirea. Cele două noduri se decid apoi să formeze adiacenţa.
2. Nodul nou şi R1 schimbă descrierea bazei de date a pachetelor. R1 avea cel mai recent LSA pentru toate ruterele din reţea cu excepţia LSA-ului nodului New_node. New_node are informaţii doar despre sine.
3. Noul nod trimite o *cerere a stării legăturii (Link State Request)* către R1, solicitând LSA-ul tuturor ruterelor şi îl primeşte în forma de *actualizare a stării legăturii (Link State Update)* de la R1.
4. R1 trimite un pachet cu o cerere a stării legăturii către New_node iar New_node trimite LSA-ul său către R1 ca un pachet de actualizare a stării legăturii. Cele două noduri au sincronizat acum LSDB-urile lor, urmat de calculul arborelui SPF şi ambele instalează noile lor tabele de rutare.
5. După sincronizarea cu New_node R1 trimite un pachet de actualizare a stării legăturii către ruterele adiacente R2 și R3. Pachetul de actualizare conţine LSA-ul aflat de la New_node. R2 și R3 efectuează calculul SPF după ce au primit actualizarea de la R1.
6. R2 și R3 inundă actualizarea la R4 şi respectiv R5, apoi ambele rutere care primesc efectuează propriile calcule SPF şi instalează propriile tabele de rutare.

3.2.7 Factorii care afectează procesul de convergenţă OSPF

Un proces de convergenţă al unei reţele constă din următoarele operaţiuni majore:

1. Timpul necesar pentru a detecta schimbările care au avut loc în reţea.
2. Timpul necesar pentru a propaga evenimentul în reţea, acesta incluzând atât generarea LSA cât şi inundarea.
3. Timpul necesar pentru a efectua calculele arborelui SPF.
4. Timpul necesar pentru a construi tabela de rutare a fiecărui router.

Prin urmare, timpul total de convergenţă pentru o reţea OSPF este dat de formula:

Timpul de convergenţă = Timpul de detectare a unui evenimet + Timpul de propagare a evenimentului + Timpul de rulare a calculului SPF + Timpul de actualizare RIB și FIB

Viteza de convergenţă poate fi afectată de un număr de factori, incluzând metoda folosită de detectare a tipului de eroare, congestia reţelei, diametrul reţelei, încărcarea procesoarelor datorită protocoalelor de rutare şi parametrului SPF utilizat în procesul de convergenţă. Figura 2.4 ilustrează parametrii implicaţi în procesul de convergenţă.

Figura de mai jos arată diferiții parametri implicați în procesul de convergenţă OSPF. Uneii dintre parametri sunt specificați în RFC 2328, iar alții sunt specifici producătorului. Protocolul *hello* funcţionează împreună cu asociatul său *HelloDeadInterval* pentru a detecta eşecurile. Parametrii *MinLSInterval* şi *MinLSArrival* sunt utilizați pentru a determina rata cu care LSA-urile sunt generate, respectiv, primite. Deoarece inundarea LSA este un proces sigur, acesta presupune trimiterea de confirmări până la recepţionarea LSA-urilor. LSA-urile nerecunoscute sunt retrimise până la expirarea parametrului *Rxmtinterval*. Calculul SPF include *SPFholdTimer (timpul de aşteptare SPF)* şi *SPFdelayTimer (timpul de întârziere SPF)* utilizați în ruterele comerciale pentru a preveni calculul frecvent al SPF în momentele în care sunt generate furtuni de LSA-uri.

PacingTimer este specific producătorului şi este folosit pentru a nu copleşi vreun vecin în timpul procesului de inundare. Actualizarea RIB/FIB reprezintă scopul final al procesului de convergenţă, după un calcul SPF de succes. Acest proces are cota maximă în procesul de convergenţă şi depinde în principal de numărul de prefixe actualizate şi de mărimea reţelei.

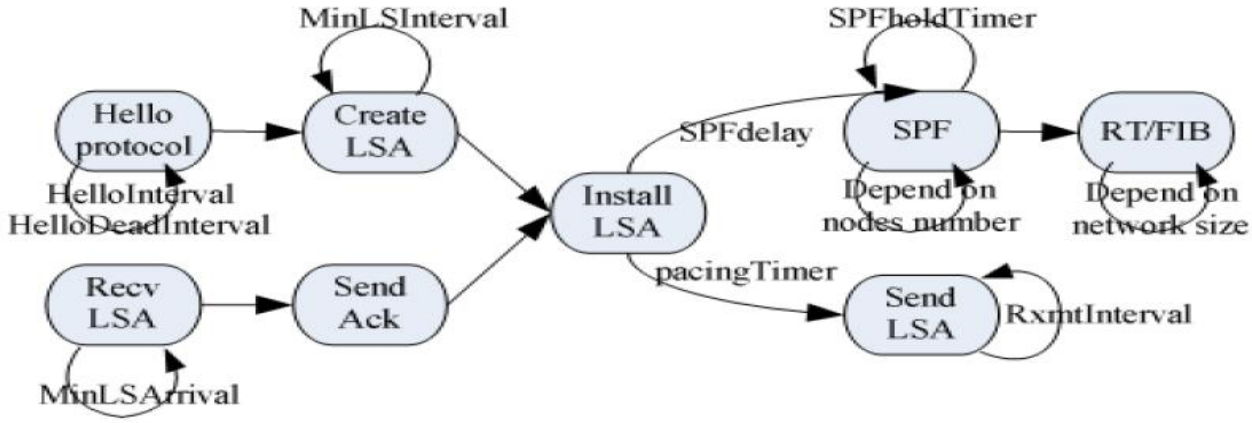


Figura 3.4 Parametrii din cadrul procesului de convergenţă OSPF

3.2.7.1 Metode de detectare rapidă a defecţiunilor

Pentru a atinge o viteză de convergenţă mai rapidă într-o reţea, mecanismul de detectare a defecţiunilor într-o reţea OSPF joacă un rol important; cu cât o schimbare este detectată mai rapid într-o topologie de reţea, cu atât se atinge o mai bună viteză de convergenţă a reţelei. Eşecurile într-o reţea pot fi detectate cu ajutorul protocolului *hello*, BFD (capitolul 3) sau folosind mecanisme de detectare a eşecului pe bază hardware, cum sunt pachetele trimise peste reţelele SONET care pot oferi un mecanism de detectare a erorilor în zeci de milisecunde, deşi punerea în aplicare a acestuia ar putea fi incomodă.

Pachetele *hello* sunt schimbate periodic între rutere vecine. Un ruter menţine un contor numit timp de inactivitate pentru fiecare vecin cu care este adiacent şi această valoare este resetată la primirea unui pachet *hello* de la vecinul său. În cazul în care apare o defecţiune la un vecin după primirea ultimului pachet *hello*, un ruter poate detecta eşecul folosind timpul său *HelloDeadInterval* care în acest caz este de 40 secunde (contorul de timp de inactivitate). Folosirea unui timp *HelloDeadInterval* de 40 secunde este unul dintre dezavantajele de punere în aplicare a protocolului *hello* în ariile în care VoIP (Voice Over IP) şi PWS (sistemul public de avertizare) sunt implementate. Expirarea contorului de timp de inactivitate determină distrugerea adiacenţei între vecini şi astfel rezultă generarea unui nou LSA. Prin urmare, parametrul *HelloDeadInterval* joacă unul din rolurile majore în timpul de detectare al defecţiunilor OSPF, atribuit vitezei de convergenţă. Prin reducerea acestui parametru la o valoare optimă mică, timpul de detectare a defecţiunilor poate fi îmbunătăţit.

3.2.7.2 Propagarea evenimentului

Într-o reţea OSPF, un LSA poate fi utilizat pentru a reflecta starea reţelei iar un LSA de tip ruter LSA descrie stările legăturilor ruterului respectiv. Odată ce un eşec este detectat într-o reţea, propagarea schimbării tururor ruterele din reţea este un alt factor care afectează viteza de convergenţă OSPF. Pentru a controla frecvenţa LSA-urilor generate până la modificările dintr-o reţea, RFC 2328 recomandă ca fiecare două LSA să fie trimise la un interval de 5 secunde; aşa numitul parametru *MinLSInterval*.

Propagarea evenimentului poate fi afectat de congestia reţelei care poate fi cauzată de furtunile de LSA sau de alți parametri asociați cu generarea LSA-urilor şi cu inundarea. Un ruter afectat de schimbare pentru a genera un LSA, se bazează pe timpul mecanismului de detectare a erorilor şi pe valoarea contorului *MinLSInterval*. Procesul de inundare include și întârzierea care se produce la ruterele de recepţie reprezentată de valoarea parametrului *MinLSArrival*, plus *pacingTimer* aplicat de ruterele de trimitere plus întârzierea care poate apărea în momentul propagării. În funcţie de natura reţelei, o furtună de LSA poate rezulta din cauza oricăror dintre următorii factori:

- un Link/ruter care flapează;
- tăieturi ale fibrei rezultând una sau mai multe legături defectate;
- schimbarea versiunii software rezultând în reîmprospătarea tuturor LSA-urilor din sistem;
- datorită LSA-urilor peridice de reîmprospătare a rețelei (1800 secunde).

Domeniul de aplicare a inundării a unei schimbări topologice poate varia în funcţie de tipul de schimbare topologică care a avut loc. LSA-urile de tip ruter, rețea sau summary LSA sunt inundate de-a lungul unei singure zone în schimb un LSA de tip ruter AS External ar trebui inundat tuturor ariilor din acel AS. Faptul că un ruter are o valoare mică pentru *MinLSInterval* poate ajuta la atingerea convergenţei rapid, dar nu este recomandată menţinerea la zero, deoarece afectează stabilitatea rutării şi consumul de resurse. Pentru a echilibra nevoia atât de convergenţă rapidă cât şi de stabilitate a unei reţele, diferiţi furnizori, cum ar fi Cisco şi Junipers, au implementat un algoritm exponenţial de rezervă pentru generarea LSA, denumit reglarea LSA (LSA throttling).

3.2.7.3 Calculul SPF

LSA-urile care reflectă schimbarea topologică sunt urmate de calculul tabelei de rutare și actualizarea FIB de pe cardurile de linie. Într-o rețea OSPF fiecare nod transformându-se într-un nod rădăcină, construieşte arborele SPF folosind algoritmul lui Dijkstra (*shortest path first*). Cea mai scurtă rută rezultată din calculul SPF este distanţa optimă, cea mai scurtă, de la o sursă la o destinaţie. La sfârşitul calculului SPF rutele optime sunt introduse în tabela de rutare şi acest rezultat se reflectă în FIB, în funcţie de arhitectura ruterului. Calculul SPF se face în funcţie de numărul de noduri din rețea şi de numărul de prefixe anunțate.

Unele implementări urmează abordarea tradiţională în care SPF este calculat pentru fiecare nou LSA sosit care astfel necesită un nivel ridicat de utilizare a procesorului. Pentru a evita un calcul continuu al rutelor şi pentru a minimiza consumul procesorului ruterului, ruterele comerciale introduc parametrii *SPFholdTimer* şi *SPFdelayTimer*. *SPFholdTimer* specifică intervalul de timp dintre două calcule consecutive ale SPF. *SPFdelayTimer* specifică timpul pe care trebuie să îl aștepte un ruter pentru a efectua calculul SPF după primirea primului LSA într-o încercare de a include mai multe LSA-uri în calcul.

Într-o rețea stabilă, este de dorit ca *SPFholdTimer* să fie mic pentru a se realiza o convergenţă rapidă, deoarece nu există multe schimbări topologice care poate apărea în rețea. Într-o reţea instabilă, unde există schimbări frecvente în topologie, o valoare mică a *SPFholdTimer* ar putea duce la un calcul SPF frecvent. Ca urmare, într-o astfel de reţea cu intervale mari în calculul SPF, este de preferat să se acumuleze mai multe modificări şi să se execute simultan.

Atât *SPFholdTimer* cât şi *SPFdelayTimer* contribuie la realizarea stabilităţii într-o reţea, dar reduce viteza în care rețeaua converge spre o nouă topologie stabilă. Ruterele Cisco, după versiunea 12,2, implementează un parametru de timp exponenţial de rezervă pentru a echilibra nevoia de stabilitate și convergență rapidă a unei rețele în orice circumstanțe. Pentru a optimiza calculul tabelei de rutare au fost sugerate metode alternative de programare a calculului tabelei de rutare, cum ar fi corelarea LSA-urilor şi metodele de calcul iSPF.

3.2.7.4 Actualizarea tabelelor Routing Information Base (RIB)/Forwarding Information Base (FIB)

RIB, numită şi tabelă de rutare, conţine informaţii destre destinaţiile accesibile şi costurile asociate pentru fiecare rută. Rutele sunt instalate fie în mod static, fie dinamic. Pentru a reflecta modificările dintr-o topologie, OSPF efectuează actualizările RIB în urma calculelor SPF. În funcţie de tipul de arhitectură a ruterului, RIB poate fi propagat mai departe către tabela de transmisie.

RIB conţine ID-ul reţelei de destinaţie, costurile asociate interfeţei şi următoarea adresă next hop, care va fi folosită ca gateway pentru a transmite pachetul de intrare. FIB este o copie a tabelei de rutare, dar ea conţine cel puţin informaţiile necesare pentru a transmite un pachet IP. FIB poate să nu fie necesar să urmărească rutele întregi şi metricile corespunzătoare pentru a ajunge la o reţea. FIB este optimizat pentru transmiterea eficientă sau pentru o căutare mai rapidă a adreselor destinaţie. Când pachetul destinat altor noduri ajunge la un ruter, FIB încearcă să transmită rapid pachetul către next hop.

Scopul final al unui calcul SPF de succes este urmat de actualizarea RIB/FIB. Timpul de actualizare este liniar dependent de numărul de prefixe modificate care pot afecta semnificativ timpul de convergenţă a unei reţele. Impactul poate lua o durată considerabilă de timp în reţelele mari. Pentru a îmbunătăţi timpul de actualizare, unele rutere folosesc o metodă proprietară numită iFIB care reflectă numai partea modificată pentru a construi apoi FIB de la zero.

3.3 Tehnici de convergenţă rapidă a protocolului OSPF

Aşa cum s-a discutat în capitolele anterioare, în general, este posibil să se realizeze convergenţa rapid prin setarea contoarelor la o valoare mai mică. Cu toate acestea, e la fel de important să se analizeze impactul asociat asupra resurselor reţelei (CPU, memorie, lărgime de bandă etc), în special pentru reţele cu instabilităţi frecvente. Cunoaşterea setărilor optime ale acestor parametri a fost întotdeauna o provocare dintr-un motiv care depinde în mare măsură de o serie de factori generici, incluzând aici nivelul aşteptat de trafic/congestionare a reţelei, numărul de noduri/legături şi alte constrângeri fizice ale legăturilor.

Această provocare a dus la evoluţia a ceea ce noi numim contoare dinamice, contoare ce se pot adapta în funcţie de comportamentul reţelei. În prezent, cele mai multe optimizări ale protocolului OSPF se bazează pe implementarea contoarelor dinamice care pot fi introduse, de preferinţă, la diferite niveluri operaţionale, pornind de la pachetul *hello* originar din generarea LSA-urilor, calcularea rutei celei mai scurte (SPF), inundarea şi actualizarea FIB.

3.3.1 Contorul Hello dinamic

Contoarele *hello* sunt parte componentă integrală din multe soluţii de rutare. Ele ajută la descoperirea vecinilor şi la detectarea defecţiunilor. Implementarea OSPFv2 bazată pe RFC 2328 prevede un interval pentru *hello* fix de 10 secunde, care este destul de mare conform studiilor moderne şi a experienţei [2][4]. Odată cu creşterea rapidă a puterii de calcul a ruterelor moderne, a devenit de bun simţ să se reducă intervalul *hello* la nivelul milisecundei. În general, este sigur să se facă acest lucru pentru reţele stabile dar ar putea apărea unele efecte nedorite în reţelele instabile, cum sunt variaţiile frecvente ale legăturilor/nodurilor şi aparaţia congestiei. În aceste situaţii şi în altele similare, în care procesorul este ţinut ocupat, contoarele *hello* nu pot fi procesate în timp util şi se pot pierde (omiterea hello-urilor). Omiterea frecventă a *hello-urilor* face ca relaţia cu vecinii să

varieze iar ruterele vecine să genereze LSA-uri care reflectă acest lucru. Acest lucru, la rândul său, forţează toate ruterele din reţea să adauge şi să şteargă o anumită rută, din când în când, provocând o întrerupere în reţea. Contoarele dinamice *hello* sunt proiectate bazându-se pe un lucru: pentru a suprima flaparea rutei cauzate de flaparea legăturii în reţelele OSPF punct la punct.

Tehnicile *hello* dinamice folosesc contorul *hello_tune,* alocat pentru fiecare interfaţă activă. Ruter-ul monitorizează prezenţa/absenţa instabilităţilor (fluctuațiile rutei, în acest caz), în intervalul *hello_tune*. În cazul în care, în perioada *hello_tune*, un ruter are parte de un eveniment *neighborDown* (ca urmare a expirării contorului de inactivitate), ruter-ul dublează intervalul său de *hello*. Făcând acest lucru, ruter-ul presupune prezenţa unei rute care flapează datorită unei legături care flapează şi dublarea intervalului *hello* cauzează o nepotrivire a parametrului *hello* cu vecinul său. Deorarece doi vecini OSPF trebuie să aibă acelaşi interval pentru *hello*, această variaţie determină ca relaţia cu vecinul să fie suspendată temporar până când celălalt vecin dublează, deasemenea, intervalul *hello*, ceea ce este mult mai probabil să se întâmple înainte ca primul ruter să-şi dubleze intervalul *hello* pentru a doua oară, deoarece cel anterior are un *routerDeadInterval* mai mic. Odată ce celălalt vecin şi-a dublat intervalul *hello*, adiacenţa se va recupera încă o dată, cu un *routerDeadInterval,* crescut de data asta. În cazul în care fluctuaţia continuă, pentru următorul interval *hello_tune,* ruterul dublează din nou intervalul *hello* (creştere exponenţială) şi întregul proces continuă încă o dată. Cu absenţa unor astfel de fluctuaţii în perioada *hello_tune,* parametrul *hello* este resetat la valoarea sa implicită. Obiectivul principal din spatele acestei tehnici este de evitare a rutelor care flapează prin mascarea legăturilor care flapează în parametrul *routerDeadInterval* crescut. Pragul maxim pentru *hello-uri* este setat la 10 secunde.

O ilustrare posibilă a acestei tehnici este în figura următoare.

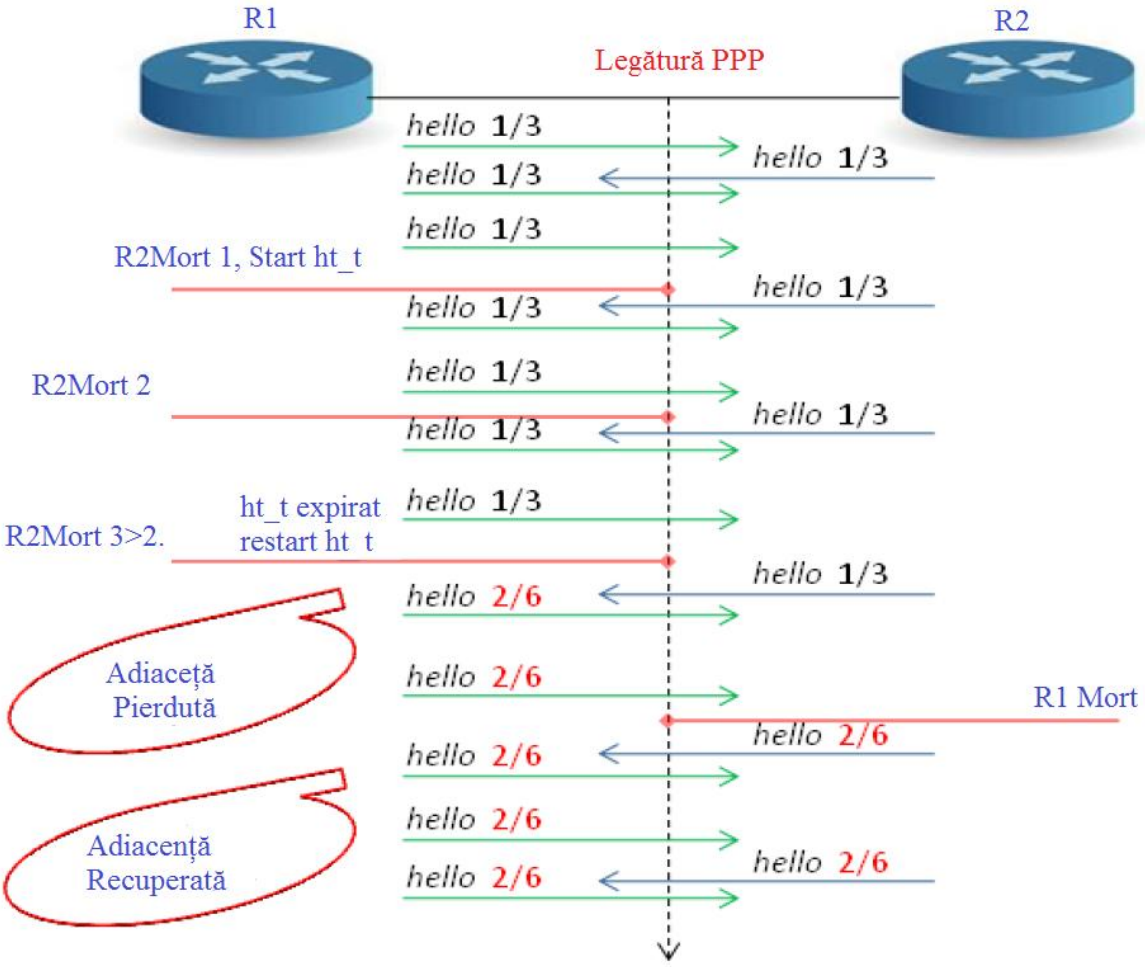


Figura 3.5 Schema Hello-urilor dinamice

3.3.2 Bidirectional Forward Detection (BFD)/Detectarea Expedierii Bidirecţionale

După cum s-a discutat în capitolul de mai sus timpul de detectare a defecţiunilor este unul dintre factorii care întârzie procesul de convergenţă. În implementările tradiţionale OSPF, timpul este dictat de parametrul *routerDeadInterval* (4**hello*). Realizarea unui timp de convergenţă sub o secundă, folosind *hello-uri* native sau *fastHello* este o provocare, dacă nu o imposibilitate, pentru că *fastHello* au nevoie de minim o secundă pentru a detecta o eroare. [6] Mai mult, folosirea *fastHello* poate fi o alegere greşită pentru funcţionarea CPU. Astăzi, cea mai bună alternativă pentru majoritatea implementarilor IGP este BFD (RFC5880), care se bazează pe contoare de tip menţinere în viaţă (keep-alive) mai mici de o secundă. BFD este un protocol de detectare a erorilor, proiectat pentru a detecta eroarea repede pe calea bidirecţională dintre două rutere, incluzând interfeţele, legăturile de date şi tabelele de rutare. Acesta este conceput pentru a fi pus în aplicare pe planul de date a arhitecturii ruterului distribuit, independent de mediu, date și de protocoalele de rutare.

Spre deosebire de legăturile punct la punct, care au un mecanism hardware de detectare a erorii, tehnologiile de nivel 2, cum este Ethernet-ul, care se bazează în principal pe protocolul *hello* pentru a detecta defecţiunile, necesită mecanisme mai rapide de detectare a erorilor, cum ar fi BFD. În cel mai bun scenariu, BFD asigură timp de detectare a erorilor comparabil cu tehnologii mai scumpe, de nivel 1/2, cum ar fi pachetele SONET/SDH, care în mod normal necesită câteva zeci de milisecunde. [2][6]

3.3.2.1 Formarea şi distrugerea relaţiei BFD între vecinii

BFD se bazează pe protocolul de rutare configurat pentru a descoperi egalii săi. Ca un proces OSPF să descopere un vecin, trimite o cerere către BFD care rulează la nivel local, pentru a forma o relaţie BFD cu vecinul. Vecinii BFD creează sesiunea BFD şi negocieză timpul în care pachetele de control sunt trimise şi primite pentru a detecta problemele de conexiune.

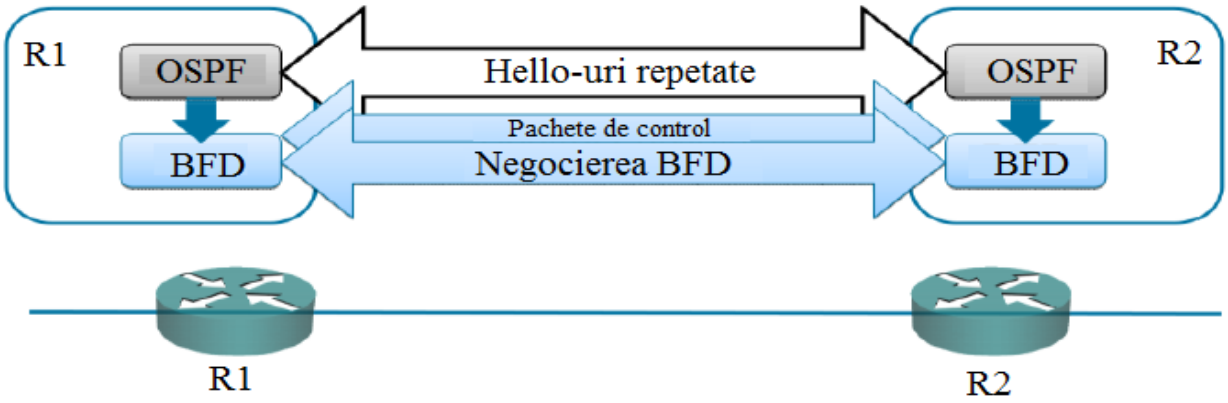


Figura 3.6 Formarea unei relații BFD între vecini (Mod asincron)

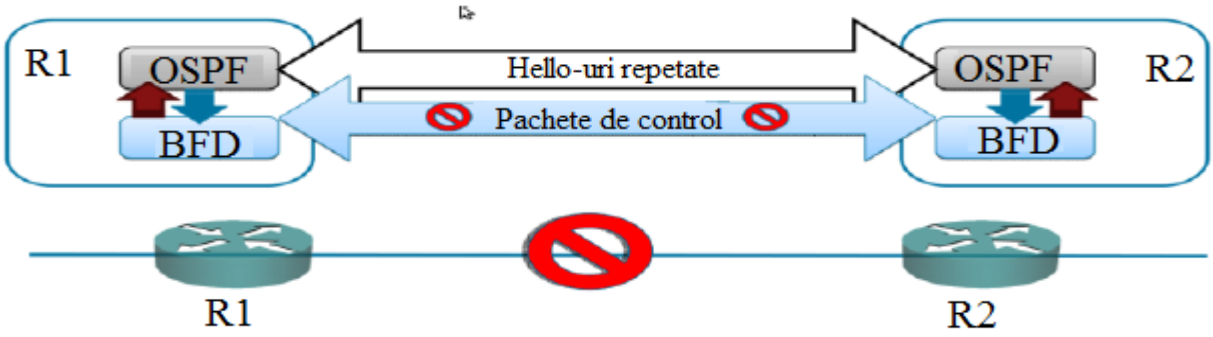


Figura 3.7 Mecanismul BFD de detecție a erorilor (Mod asincron)

Când apar defecţiuni într-o reţea, aşa cum se arată în figura 3.7, pachetele de control nu sunt primite în ambele părţi. Ca urmare, sesiunea BFD este ruptă şi BFD raportează la OSPF că vecinul este inaccesibil. Acest lucru ajută OSPF să detecteze erorile rapid şi să găsească o cale alternativă, dacă este disponibilă.

Două moduri principale de operare sunt sprijinite de BFD, modurile asincrone şi cerere. Modul asincron este principalul mod de funcţionare care implică un schimb periodic de pachete de control (la fel ca *hello-urile*). Cele mai multe dintre sistemele IOS ale Cisco sprijină modul asincron BFD. Modul cerere, care funcţionează pe circuitul de cerere, cum ar fi ISDN, nu implică un schimb periodic de pachete de control BFD după ce sesiunea BFD este stabilită. În schimb, o secvenţă scurtă a pachetului BFD este schimbată, atunci când un dispozitiv trebuie să verifice conectivitatea explicit şi poate funcţiona independent, în fiecare sau în ambele direcţii. De asemenea, BFD sprijină funcţia ecou, care funcţionează atât cu modul asincron cât şi cu modul cerere. Funcţia ecou permite unui dispozitiv să trimită pachete ecou la vecinul său prin setarea propriei adrese ca adresă de destinaţie (pachete auto destinate); vecinul transmite imediat pachetul ecou fără procesarea acestuia şi acest lucru reduce timpul bruiaj dus-întors. Pachetele ecou BFD sunt încapsulate în pachete UDP folosind portul destinatie 3785 şi detectează defecţiunile între vecinii direct conectaţi. Dacă pachetele ecou trimise nu sunt primite în intervalul de timp de detectare BFD, dispozitivul poate declara că sesiunea este încheiată. Deoarece funcţia ecou poate furniza în mod independent un mecanism de detectare mai rapid, rata cu care pachetele de control BFD sunt schimbate între dispozitive poate fi redusă. Dezavantajul BFD este că nu poate detecta defecţiuni în planul de control aşa cum este implementat în planul datelor. Totuşi, poate fi utilizat cu *fasthello* pentru a realiza acest lucru. Detalii despre protocolul BFD pot fi găsite pe RFC 5880.

3.3.3 Întârzieri cu reglaj fin în generarea LSA, calculul SPF şi inundarea

3.3.3.1 Reglarea LSA (minLSAInterval dinamic)

Modificările topologice dintr-o reţea OSPF sunt comunicate folosind LSA-uri. RFC 2328 menţionează următoarele modificări topologice, ca posibile cauze ale generării de LSA-uri.

- Schimbarea stării unei interfeţe (sus/jos);
- Schimbarea ruterelor desemnate (Designated Router - DR);
- Schimbarea ruterelor vecine de la/la starea FULL.

În general, o modificare a conţinutului unui LSA existent apare într-un nou LSA care reflectă noua schimbare. Din motive de siguranţă, iniţierea oricăror două LSA-uri consecutive trebuie să fie cel puţin separată de *minLSAInterval*. *MinLSAInterval* (implicit 5s) este un mecanism aparţinând ruterului care protejează CPU în momentele de modificari de topologie la scară largă sau de fluctuaţii persistente ale legăturilor/nodurilor. Cu toate acestea, pentru procesoarele moderne, această întârziere pare prea conservatoare şi intolerabilă. În schimb, acesta poate fi setat la o valoare iniţială mai mică pentru o convergenţă rapidă atunci când apar schimbări şi poate fi setat să crească adaptat cu încărcarea reţelei. Reglarea LSA este o astfel de abordare pentru a mări/micşora adaptiv intervalul de iniţiere a LSA-urilor folosind algoritmul exponenţial de revenire.

Reglarea LSA face uz de intervalul *iniţial* şi de contoarele *aşteptare* (*hold)* şi *aşteptare maximă* (*max_wait*). În cazul în care un LSA urmează să fie generat pentru oricare dintre motivele de mai sus, acesta va fi mai întâi întârziat pentru o perioadă *iniţială* de timp iar când timpul de întârziere iniţială expiră, parametrul *hold* începe cu un interval *hold* dat. Evenimentele ulterioare intervalului *hold* nu vor genera LSA; în schimb, acestea sunt acumulate până la expirarea timpului *hold* şi, ca rezultat, un singur LSA este generat. Atât timp cât fluctuaţia continuă, procesul deasemenea continuă, crescând intervalul hold exponenţial (2^t*$hold$) până când se atinge o

anumită valoare *max_wait* prestabilită. Parametrul *hold* nu va mai creşte dincolo de *max-wait*, dar îşi menţine acea valoare, chiar dacă fluctuaţia continuă. Când nici o fluctuaţie nu mai este detectată, în durata de *2*max-wait* a întârzierii, parametrul *hold* este resetat la valoarea sa iniţială. Este important de reţinut că intervalul *hold* ar echivala aproximativ cu timpul total de convergenţă explicat în subcapitolul 2, astfel încât toate ruterele să reflecte deja noua schimbare, înainte de următoarea schimbare. Configurarea implicită folosită de Cisco foloseşte intervalele 10 100 5000 milisecunde pentru aceşti trei parametri şi recomandă ca intervalul de aşteptare să fie stabilit mai mare sau egal cu *minLSArrival*, astfel încât alte rutere care primesc actualizarea să nu piardă LSA-urile valabile.

3.3.3.2 Reglarea SPF/Aşteptarea SPF (SPF Throttling/SPF hold-down)

Spre deosebire de ruterele moderne, de mare viteză, care iau comanda într-un timp variind de la milisecunde până la câteva secunde, pentru a finaliza calculul SPF, sistemele mai vechi, din anii 1980 şi 1990, necesitau zeci de secunde pentru a finaliza operaţiunea. Un factor pentru acest lucru, în afară de puterea procesorului, atunci mai mică, este ineficienţa algoritmului Dijkstra original. Algoritmul Dijkstra original, cu o complexitate medie de *n*logn* (chiar 2*n pentru reţeaua plasă completă - full mesh), este considerat a fi ineficient şi mai puţin accesibil, comparativ cu variantele sale moderne, care au o complexitate medie de *logn*. Cu toate acestea, pentru reţelele ISP moderne care menţin sute de mii de rute, calculul SPF a rămas una dintre operaţiunile intensive ale CPU care trebuie să fie optimizate.

RFC 2328 precizează calculul SPF imediat după primirea/iniţierea unui nou LSA. În cazul în care un ruter programează calculul SPF pentru fiecare LSA pe care-l primeşte, acest lucru ar putea în cele din urmă bloca CPU-ul doar pentru calculul SPF şi toate celelalte sarcini utile ar fi irecuperabile. O abordare oarecum conservatoare foloseşte un *fixedHoldTime* între calculele SPF succesive, dar nici aceasta nu scalează bine şi întârzie convergenţa. Reglarea SPF este un sistem modern de planificare SPF care utilizează aceeaşi tehnică şi procedură ca şi reglarea LSA, dar funcţionează pe întârzierea calculului SPF, după ce un LSA a fost deja generat/primit prin inundare.

Trei parametri şi anume intervalul *spf-start*, contorul *spf-hold* şi contorul *spf-max-wait*, sunt utilizaţi în cadrul aceastei tehnici. *Spf-hold* controlează numărul de repetări SPF şi ar trebui să fie setat optim pentru convergenţa reţelei. De asemenea, *spf-start* ar trebui să fie cât mai mic posibil pentru a permite o reacţie instantanee la schimbări, cum ar fi defectele tranzitorii şi modificările de topologie ocazionale, dar suficient de mare pentru a permite inundarea cu succes a unui LSA generat/primit înainte de începerea calculului SPF.

Pentru Juniper punerea în aplicare a acestei tehnici diferă puţin şi anume prin faptul că are are două moduri de funcţionare *slow mode (modul lent)* şi *fast mode (modul rapid)* şi spre deosebire de creşterea exponenţială Cisco a perioadei *hold,* foloseşte o creştere liniară. În cazul în care numărul de rulări SPF depăşeşte o limită prestabilită, numită *rapid-runs (rulări rapide)*, într-o anumită perioadă de verificare, calculul SPF se schimbă pe *slow mode*, pornind contorul *hold-down*. Toate evenimentele SPF ulterioare programării, nu vor declanşa calculul SPF până la expirarea contorului *hold-down*. Figura următoare ilustrează implementarea Cisco a reglării SPF, care recomandă 10 100 500 ms. Un model similar se poate deduce pentru reglarea LSA.

După cum se poate observa în figura de mai jos, la a treia iteraţie, *spf-hold* este programat la 4*increment iar dacă aceasta depăşeşte pargul *spf-max-wait,* va fi trunchiat la *spf-max-wait*. În absenţa fluctuaţiei pe durata a două sau mai multe astfel de iteraţii, *spf-max-wait* va fi resetat spre start.

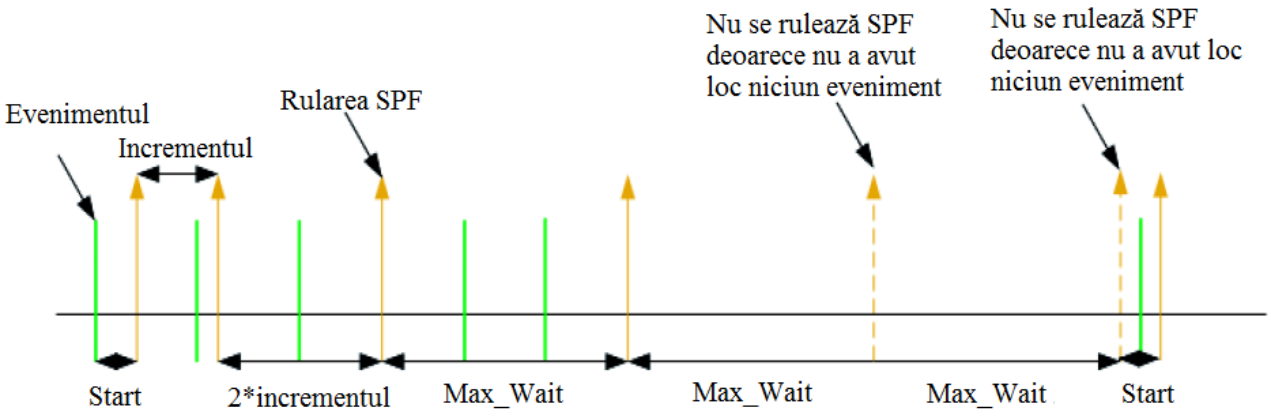


Figura 3.8 Parametrii folosiți în reglarea SPF

3.3.4 Întârzierile ritmului pachetelor (pachet-pacing) OSPF

Versiunea 12.2 a Cisco IOS a introdus, de asemenea, trei tipuri de întârzieri *packet-pacing* OSPF, pentru a optimiza procedura de inundare. Implementări similare există şi în junOS. Întârzierile *packet-pacing* sunt utilizate pentru a evalua limita inundării/retransmiterii pachetelor de actualizare LSA, cu intenţia de a reduce CPU sau de a-i limita utilizarea. Ajutând în acelaşi timp la realizarea convergenţei rapid, în momente de modificări topologice ocazionale, aceste tehnici urmează în schimb o procedură controlată de inundare, pentru reţelele care conţin o mare bază de date a legăturilor (LSDB).

3.3.4.1 Parametrul de inundare packet-pacing

Dacă un ruter are un număr mare de LSA care trebuie inundate pe o interfață, acest parametru dictează viteza cu care se întâmplă acest lucru. Acest tip de parametru este unul setat pe interfaţă şi are efect numai în cazul în care un ruter are mai mult de un LSA.

3.3.4.2 Parametrul de retransmisie packet-pacing

Ca parte a transmisiei fiabile OSPF, LSA-urile trimise către un vecin sunt păstrate într-o listă de retransmisie pentru retransmisia ulterioară, în absenţa confirmării. Parametrul de retransmitere pachet-pacing funcţionează în acelaşi mod ca parametrul de inundare packet-pacing, prin controlarea vitezei cu care LSA-urile sunt trimise din lista de retransmitere. O punere în aplicare mai avansată a parametrilor de retransmisie, aşa cum se sugerează în RFC 4222, este de a avea *RxmtInterval* dinamic la fel ca tehnica *minLSAInterval* dinamic, discutată în secţiunea 3.3.3.1, astfel încât *RxmtInterval* să crească exponenţial atunci când numărul de pachete nerecunoscute din lista de retransmitere creşte peste un anumit prag (de exemplu, în momente de congestie). Intervalul este din nou resetat la valorile iniţiale, atunci când nu este detectat vreun semn de congestie.

3.3.4.3 Parametrul de grup packet-pacing

Înainte de această facilitate, implementarea OSPF Cisco avea un interval fix de 30 minute de reîmprospătare, după care toate LSA-urile auto-generate de un ruter erau reîmprospătate. Reîmprospătarea se aplică tuturor acestor LSA-uri la un moment dat, în ciuda vârstei lor actuale. Un astfel de program creează o perioadă de vârf a LSA-urilor inundate în reţea şi necesită o cantitate considerabilă de procesare CPU şi utilizarea unui buffer.

Specificațiile OSPFv2, în conformitate cu RFC 2328, aplică, în schimb, un timp individual de reîmprospătare pentru fiecare LSA auto-generat, astfel încât un LSA este inundat când va atinge limita de 30 minute (care este jumătate din vârsta lui). Pe de altă parte, acest lucru face inundarea mai frecventă şi mai incomodă. O punere în aplicare mai recentă a Cisco adaugă o întârziere numită întârziere de grup pachet-pacing (240ms implicit) pentru fiecare reîmprospătare, într-o încercare de a conţine mai multe LSA-uri într-un singur pachet de actualizare LSA. Acest lucru ar face inundarea mai eficientă şi ar reduce sarcina pe expeditor, precum şi pe vecinii care primesc, dar acest lucru întârzie într-un fel procesul de convergenţă.

3.3.5 Amortizarea evenimentului IP (IP Event Dampening)

Amortizarea evenimentului IP sau amortizarea BGP (pentru BGP, RFC2439) este, de asemenea, o tehnică utilizată la scară largă pentru a asigura stabilitatea de rutare la nivel mondial, prin reducerea efectului fluctuațiilor rutelor, care sunt cauzate de fluctuațiile de înaltă frecvenţă ale interfeţei/legăturii. Aproape toate soluţiile de rutare ale Cisco începând cu versiunea de IOS 12.0 S şi implementările BGP pentru junOS includ această caracteristică în distribuţiile lor.

Implementarea Cisco IGP pentru acest lucru este folosită pentru a pedepsi o interfaţă care are fluctuaţii persistente, prin ascunderea stărilor tranziţiilor sale (sus/jos) de protocoalele de nivel superior şi, prin urmare, de reţea. Acest lucru ajută la închiderea temporară a legăturii şi îngheaţă toate rutele care trec prin ea până când se stabilizează. Efectul aceastei tehnici este foarte similară cu efectul reglării LSA, dar este un pic conservatoare şi nu poate fi activată în cazul în care un ruter foloseşte reglarea LSA sau invers. De fiecare dată când o interfaţă are fluctuaţii (cade), i se atribuie o valoare de pedeapsă conform formulei Pn=Pn-1*2^(-t/H)+Pn-1. Dacă această pedeapsă cumulată depăşeşte valoarea maximă a unui prag, numit prag de suprimare, interfaţa este suprimată. Interfața își va reveni când pedeapsa scade sub valoarea de reutilizare.

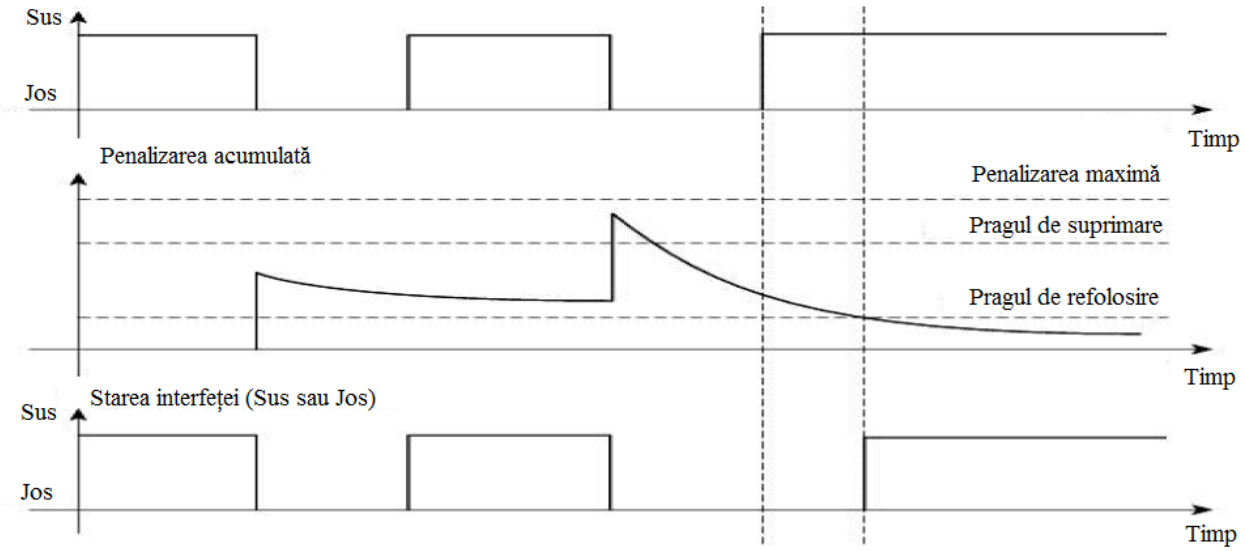


Figura 3.9 Amortizarea evenimentului IP

Algoritmul foloseşte cinci parametri; *penalizare (penalty)* (*cifra de merit* aşa cum o numeşte junOS), *pragul de suprimare/de tăiere (suppress/cut-off), perioada de înjumătățire (half-life), pragul de reutilizare (reuses)* şi *timpul maxim de suprimare (max-suppress).*

Pragul de suprimare (implicit 2000) - Este valoarea de penalizare maximă acumulată după care legătura este suprimată. În cazul în care contorul de penalizare este >1, în perioada de înjumătățire, pedeapsa creşte continuu cu P*2^(-t/H)+P.

Cu absenţa fluctuației în fiecare perioadă *de înjumătățire*, pedeapsa scade exponenţial conform formulei P(t)=P(0)*2^(-t/H); dacă contorul fluctuației este >1 în perioada de *înjumătățire*, pedeapsa creşte conform P*2^(-t/H)+P.

P(t) reprezintă *penalizarea* (iniţial 1000), la timpul t, iar H reprezintă perioada *de înjumătățire,* definită mai jos.

Perioada de înjumătățire (5s) **–** este durata de timp după care pedeapsa scade exponenţial corespunzător formulei de mai sus (determină cât de repede pedeapsa scade exponenţial). Acest timp trebuie ales cu grijă, bazându-ne pe frecvenţa fluctuației. În general, derivarea formulei de mai sus păstrează H >= T/log2(P/(S-P).

Pragul de reutilizare (implicit 1000) **–** Este limita de jos a *penalizării*. Dacă valoarea *penalizării* scade sub această valoare, interfaţa este reutilizată.

Parametrul *max-suppress-time* (stabilit 4**half-life*) **–** Se referă la timpul maxim în care o rută poate fi ascunsă. Se recomandă a se avea configurată această caracteristică, în special în reţelele cu legături redundante, care sunt specifice pentru furnizorii mari de servicii, datorită faptului că ruterele încep să folosească o legătură alternativ în mod stabil, până când legătura care are fluctuații se stabilizează.

3.3.6 Îmbunătăţiri ale SPF

3.3.6.1 Incremental SPF (iSPF)

Calculul SPF folosind vechiul algoritm Dijkstra, numit *static dijkstra* este acum considerat a fi ineficient din mai multe aspecte. Una dintre aceste limitări este că pentru un calcul complet SPF pentru fiecare nou LSA primit sunt afectate toate LSA-urile din LSDB. Cu toate acestea, ar putea fi cazul în care un astfel de LSA nu va afecta/schimba arborele celor mai scurte rute existente (SPT), sau poate schimba doar o parte din el. Acest lucru se întâmplă, de exemplu, atunci când LSA-ul descrie eşecul unei legături care nu este parte a SPT-ului salvat (fiecare nou SPT trebuie să fie salvat pentru iSPF). Astfel de modificări topologice ar trebui să fie pur şi simplu ignorate, dacă sunt corect identificate.

iSPF este o abordare mai sofisticată, care evită astfel de calcule inutile prin compararea atentă a noilor LSA-uri cu SPT salvat, înainte de declanşarea unui nou calcul SPF. iSPF permite, deasemenea, actualizări RIB rapide, prin evitarea actualizării RIB redundante şi acest lucru contribuie foarte mult la viteza de convergenţă.

Caracteristica este disponibilă în versiunea de IOS Cisco 12.0(24) S şi în cele ulterioare.

Proprietăți ale iSPF:

- Dacă schimbarea topologică este adăugarea unui nou nod frunză, cum ar fi nodul R8 din figura de mai jos, pur şi simplu se extinde SPT de la R6 şi costul său ar fi costul pentru R6 plus costul de ieşire al interfeţei pentru R8. Această proprietate este similară modului de calcul al unei rute parţiale, discutat în secţiunea 3.3.6.2 și 3.3.6.3.

- Dacă o legătură, care nu a fost salvată în SPT anterior, de exemplu R4-R5, cade acest lucru nu ar declanşa un nou calcul SPF.
- În cazul în care orice legătură din arborele SPT actual cade, de exemplu legătura dintre R1 şi R5, acest lucru ar declanşa un calcul SPF de la rădăcină până la R5, R6, R7 şi R8; acest lucru se întâmplă la toate nodurile din sub-arborele legăturii eronate.

iSPF se dovedeşte mai eficient pentru topologii cu un număr mai mic de interconexiuni (mai puţin dens) şi cu cât eroarea este mai departe de rădăcină, cu atât mai simplu ar fi calculul iar acest lucru compensează întârzierea mai mare de propagare, rezultată din astfel de eşecuri îndepărtate.

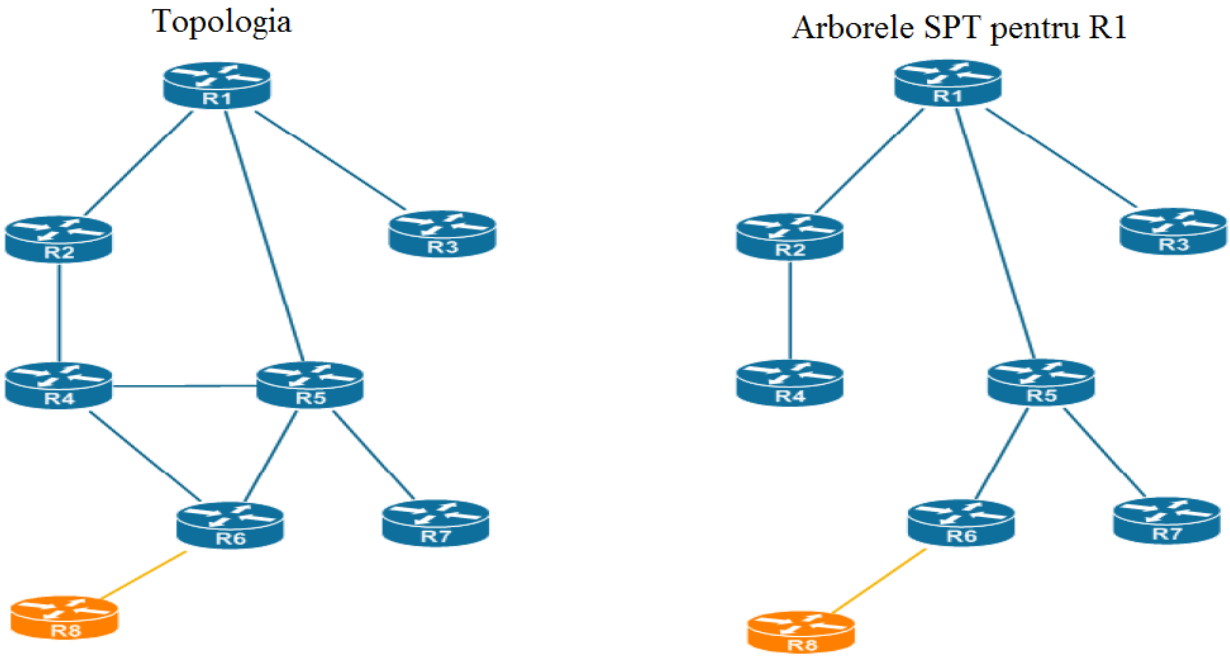


Figura 3.10 Topologie simplă (stânga) şi arborele de acoperire (dreapta)

3.3.6.2 Calculul rutei parţiale (PRC) în IS-IS

Aşa cum s-a arătat în secţiunea anterioară, modul inerent OSPFv2 de gestionare a informaţiilor din LSA-urile sale are unele inconveniente atunci când vine vorba de PRC. Informaţia privind prefixul IP (informaţia despre subreţeaua conectată) este parte integrată dintr-un LSA OSPF de tip 1 sau 2 care poartă, de asemenea, informaţia topologică. Deci, ori de câte ori un prefix IP sau statusul unei legături ciot din aceeaşi arie se schimbă, sunt generate aceleaşi LSA-uri ca pentru orice altă modificare topologică care trebuie să declanşeze calculul SPF. OSPF nu are nici o modalitate de a identifica faptul că astfel de modificări sunt doar modificări ale prefixului IP şi nu modificări de topologie; ca urmare, programează un calcul SPF complet şi normal. Dacă ar detecta astfel de modificări, doar PRC-ul prefixelor modificate va fi calculat, ceea ce este o sarcină simplă de îndeplinit. Dar, OSPF face acest lucru numai pentru LSA-urile de tip 3 şi 5, ambele transportând informaţii ale prefixului IP pentru rute externe.

PRC este parte integrală a IS-IS datorită faptului că manipularea de informaţii IS-IS din LSP-ul său este mai adecvată pentru astfel de optimizări. IS-IS are un TLV separat (*Tip-Lungime-Valoare*) pentru a transporta informaţii de accesibilitate IP şi un altul pentru informaţia IS a vecinilor (IS - sistem intermediar) (de exemplu, informaţii topologice) din care SPF se calculează independent de informaţia prefixului IP. Fiecare reţea IP, în IS-IS este considerată ca fiind externă şi sfârşeşte prin a fi o frunză; aşa că ori de câte ori se adaugă/elimină o legătură ciot sau un nod

frunză, IS-IS consideră PRC la fel ca un vector distanţă de adăugare/îndepărtare şi doar erorile tranzitorii ale legăturii, care potenţial ar putea afecta întreaga topologie, declanşează un calcul SPT complet. Acestea fiind spuse, marii furnizori, cum ar fi Cisco, au reuşit să găsească o cale de ieşire pentru această problemă tot în OSPFv2, astfel: atunci când nodurile frunză sunt adăugate/eliminate din reţea, traseele sunt redistribuite la fel ca LSA-urile de tip 3 şi 5 spre deosebire de cazul normal în care acestea sunt promovate ca LSA tip 1 sau 2. Acest truc a permis OSPF să facă PRC pe legături ciot/frunză, cu puțină încărcare asociată CPU. Oricum, cu introducerea iSPF atât în OSPF cît şi în IS-IS, acest lucru nu mai este o problemă.

3.3.6.3 Calculul rutei parţiale (PRC) în OSPFv3

OSPFv3 face acest lucru pentru IPv6 într-un mod mai inteligent prin introducerea unui nou tip de LSA (LSA de tip 9) numit prefix LSA intra-arie, care transportă o informaţie separată, intra-arie, a prefixului de reţea (fără o semantică a adresării IP). LSA-urile ruterele/reţelelor sunt transportate în LSA-uri separaţe, care transportă doar informaţii topologice.

În OSPFv3, identificatorii LSA, cum ar fi ruter ID, ID-ul ariei şi ID-ul stării legăturii, sunt numere de 32 de biţi, scrise în reprezentare zecimală, dar ele nu sunt de fapt adrese IPv4, ci doar numere. Aceste numere identifică în mod unic fiecare router într-o anumită arie, spre deosebire de OSPFv2, care identifică vecinii după adresele IP ale interfeţelor sau un ruter ID IPv4.

Tipurile 1 sau 2 de LSA-uri nu transportă informaţii de semantică ale adreselor IP şi sunt transmise doar atunci când există informaţii pertinente pentru calcularea SPF. În caz contrar, un simplu prefix IP sau o legătură ciot se modifică doar pentru a genera un prefix intra-arie LSA pentru care SPF nu ar rula. Acest lucru a făcut uşoară modificarea informaţiei despre prefixul IP, fără a afecta arborele SPT.

3.3.7 Corelarea LSA-urilor

Corelarea LSA este o altă tehnică propusă pentru optimizarea SPF[7]. Calculul SPF se bazează pe timpul de aşteptare, care utilizează algoritmul fix/exponenţial de revenire care are întârzieri inutile şi este ineficient în anumite circumstanţe. LSA-urile individuale sunt doar simptome ale unei schimbări topologice şi nu ar trebui să declanşeze întotdeauna calculul SPF. În schimb, fiecare stare a unei legături primită în pachetele de actualizare trebuie să fie colectată pentru o anumită perioadă de timp şi corelate pentru eventualele modificări de topologie. Procesul de corelare identifică posibilele similitudini între stările legăturilor pentru adiacenţe prin inspectarea, în principal, a câmpurilor LSA - *ID-ul legăturii, datele legăturii* precum şi *tipul legăturii*. Dacă orice schimbare topologică se observă în timpul procesului de corelaţie, acest lucru va declanşa imediat calculul SPF. Ideea pare inteligentă, datorită faptului că LSA-urile care descriu aceeaşi schimbare, ar putea fi generate de mai multe rutere adiacente, fiecare dintre acestea necesitând rularea SPF când este primit de către o destinaţie.

O parte a procesării din corelarea LSA-urilor este, desigur, deja specificată în RFC 2328, iar autorii au observat acest lucru. RFC afirmă clar că fiecare LSA primit ar trebui să fie verificat dacă are modificări față de LSA-urile stocate anterior, dacă nu este primit pentru prima dată iar calculul SPF este programat, dacă şi numai dacă există o diferenţă în topologie. Cu toate acestea, în ceea ce ne privește, specificaţiile inițiale OSPFv2 nu diferenţiază similitudini pentru adiacenţe.

În cel mai simplu caz, atunci când o legătură dintre două rutere adiacente, dintr-o reţea punct la punct, cade, ambele rutere generează un LSA care descrie schimbarea. Un ruter care primeşte aceste LSA-uri este posibil să programameze, peste o anumită perioadă de timp, calculul SPF pentru fiecare, în ciuda faptului că ele vorbesc despre aceeași modificare. Pentru a da un alt exemplu, să presupunem că un nod central, care transportă multe adiacenţe dintr-o reţea punct la punct cade; în urma acestei căderi, fiecare dintre ruterele adiacente generează un LSA descriind

acest lucru. Să presupunem că un ruter aflat la o anumită distanţă primeşte toate aceste LSA-uri. Implementarea tradiţională ar putea programa calculul SPF pentru fiecare LSA primit. Întrebarea este putem face acest lucru mai inteligent? Acest lucru se poate face mai inteligent, în cazul în care ruterul care primeşte ar putea analiza într-un fel natura stării legăturii din aceste LSA-uri. Pentru primul LSA primit, programează imediat un calcul SPF (pentru a reflecta imediat modificările, considerându-le ca o schimbare topologică ocazională), dar, dacă alt LSA este primit, de la un alt ruter anunţând pierderea adiacenţei cu acelaşi nod central, atunci acest lucru ar putea fi un semn că ruterul central cade, aşa că aşteaptă pentru un timp înainte de a rula SPF, în speranţa că va mai vedea mai multe astfel de anunţuri de la ruterele adiacente rămase (presupune că ruterul care calculează ţine evidenţa tuturor informaţiilor adiacenţei unui ruter, prin intermediul LSA-ului ruterului) şi odată ce se asigură că are toate LSA-urile de la toate ruterele adiacente, programează un nou calcul SPF (al doilea calcul SPF). Acest lucru este în contradicţie cu LSA-ul pur cu timp de aşteptare condus/fix sau cu schema exponenţială de revenire, care necesită mai multe rulări SPF; în cel mai rău caz, poate fi una pentru fiecare astfel de exemplu.

Cei trei paşi importanţi din corelarea LSA-urilor sunt:

Pasul 1: Identifică o schimbare în sus, în jos sau de cost a unui sub-eveniment prin examinarea cu atenţie a conţinutului noului LSA şi a versiunii salvate;

Pasul 2: Corelează sub-evenimentele pentru a identifica o schimbare topologică;

Pasul 3: Post procesare ca urmare a modificării topologice.

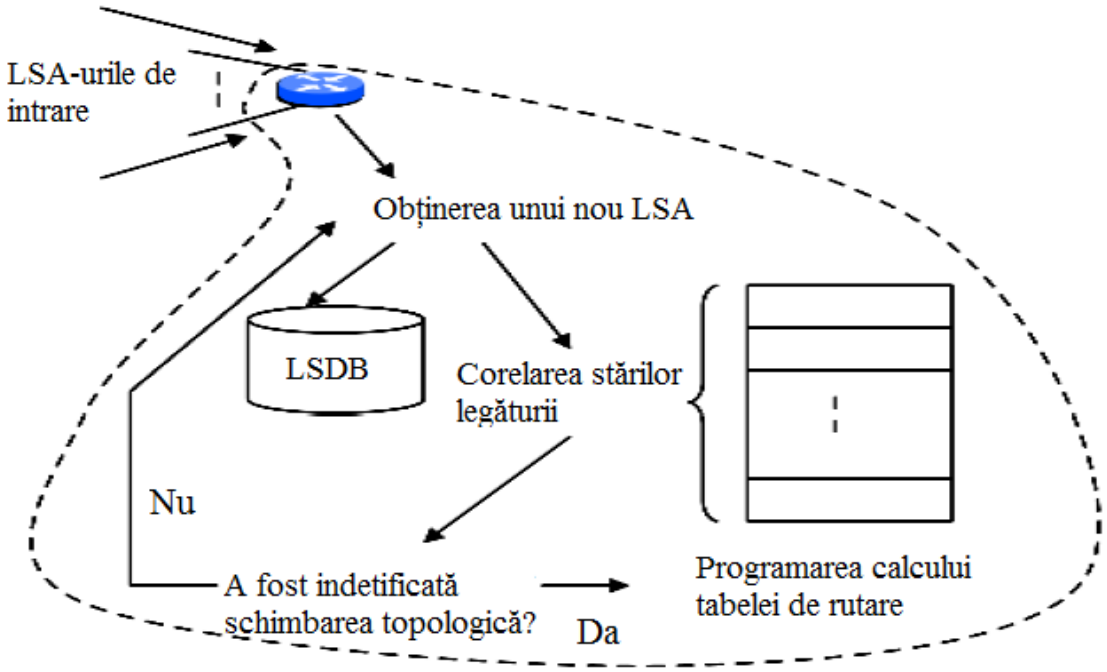


Figura 3.11 Procedura corelării LSA-urilor

3.3.8 Restartul graţios (Graceful restart)

Din când în când, este necesar ca un ruter să fie oprit temporar, ca parte a unei activităţi de actualizare planificată/a unei activități de întreţinere sau din cauza unor evenimente neprevăzute, cum ar fi căderi de tensiune sau accidente. Cu toate acestea, între timp, un operator poate solicita să nu fie întreruptă comunicarea în reţeaua existentă. Această nevoie poate fi abordată printr-o procedură numită repornire graţioasă sau expediere non-stop (*graceful restart* sau *non-stop forwarding*), aşa cum este documentat în RFC 3623. Repornirea graţioasă face uz de arhitectura

separată a planurilor de control și de date ale ruterelor moderne. Această arhitectură distribuită a dat o creştere a posibilităţii ca planul de control, care se ocupă de toate operaţiunile OSPF, să fie repornit în condiţii de siguranţă lăsând operaţiunea de expediere de date planului de expediere a datelor.

Procedura decurge după cum urmează:

Ruterul care repornește, numit *router inițiator*, trimite un mesaj special, numit *LSA grațios*, care este un LSA opac de tip legătură locală, către toţi vecinii săi adiacenţi, numiţi *ajutoare*, înainte de a reporni (face acest lucru după ce a repornit, înainte de a trimite *saluturi*, pentru reporniri neplanificate). LSA-ul grațios nu este inundat de ajutoare, este doar un semnal care anunţă că inițiatorul repornește, astfel încât ajutoarele nu trebuie să se rupă de adiacenţa lor, dar, în schimb pretind că ruterul este încă în stare funcţională şi continuă să publice acest lucru în LSA-urile lor. Inițiatorul trebuie să transmită rapid un LSA, după o repornire de succes. În acelaşi timp, ajutoarele intră pe *mod ajutor*.

În scopul evitării unei posibile asimetrii, ruterul inițiator salvează numărul său de ordine într-o memorie non-volatilă, înainte de repornire şi atunci când repornește, se reintroduce prin reoriginarea LSA-ului ruterului/rețelei, executând din nou calculul SPF şi actualizând tabela de rutare (trebuie să trimită LSA-ul grațios înainte de acest lucru). Cu toate acestea, dacă ar apărea orice modificare topologică în timpul perioadei de repornire, inițiatorul nu ar putea reflecta această schimbare în tabela de rutare şi s-ar putea crea o buclă de rutare. În astfel de cazuri, ajutoarele sale nu ar mai ascunde repornirea şi ar rupe adiacenţa lor (ar ieşi din modul de ajutor), prin omiterea legăturii adiacente din LSA-urile lor. De data aceasta, atât inițiatorul cât şi ajutoarele sale presupun o repornire normală şi acest lucru este, desigur, nedorit. Pentru repornirile planificate, administratorul de rețea trebuie să emită comanda corespunzătoare, împreună cu o perioadă estimată de graţie care nu ar trebui să fie în mod normal mai mică de 1800 s (în scopul de a evita îmbătrânirea LSA-urilor în timpul perioadei de repornire). Repornirea graţioasă nu se recomandă în momentele de repornire neplanificată, din cauza faptului că ruterul nu va avea suficient timp să se pregătească pentru ea şi această opţiune ar trebui să fie în mod normal dezactivată implicit.[8]

3.3.9 Actualizări incrementale ale FIB

Timpii de actualizare FIB variază între rutere, în funcţie de natura erorii şi în funcţie de numărul de prefixe de rutare, afectate de eroare. Timpul de actualizare FIB contribuie cel mai mult la întârzierea convergenţei. Îmbunătăţirile recente sugerează o tehnică numită actualizarea FIB incrementală (iFIB). Această tehnică, în loc să descarce întreaga tabelă de rutare de pe cardurile de linie, actualizează destul de selectiv doar rutele sau prefixele afectate de modificare, ori de câte ori se modifică topologia. Această abordare funcţionează cel mai bine cu tehnica iSPF, discutată în secţiunea 3.3.6.1, astfel se reduce semnificativ timpul de convergenţă şi se economiseşte lăţimea de bandă.

3.4 Durata de convergenţă sub o secundă şi provocările asociate

OSPF are un timp de convergenţă mai bun decât un protocol de rutare vector distanţă cum ar fi RIP. După cum s-a menţionat în capitolul 3.3, viteză de convergenţă poate fi afectată de diferiţi factori. Astăzi, într-o rețea, există o cerere mare pentru o viteză de convergenţă sub o secundă dar provocările asociate cu utilizarea CPU și cu instabilitatea rețelei, trebuie, de asemenea, abordate. Este un lucru obişnuit să vedem legături de 10 gigabiți între două dispozitive din nucleul rețelelor

furnizorilor de servicii. Căderea unor astfel de legături, pentru câteva secunde, poate duce la pierderea de informaţii, înainte de încheierea completă a procesului de convergenţă a reţelei şi poate influenţa serviciile clienţilor.

Unele dintre principalele motive pentru care convegenţa sub o secundă este foarte importantă:

- Importanţa deosebită a serviciilor uptime;
- Implementarea pe scară largă a aplicaţiilor în timp real, cum ar fi VoIP (Voice over IP);
- Implementarea sistemelor de avertizare cum ar fi ETWS - sistemul de avertizare pentru cutremur şi tsunami;
- Misiuni în timp real şi aplicaţii critice, de la controlerele care sunt utilizate în industrie până la diferite sisteme sofisticate de comunicare în câmp de luptă.

După cum s-a menţionat în capitolul 3.2, timpul de convergenţă al unei reţele este suma duratelor necesare pentru a detecta erorile, propaga evenimentul, efectua calcule SPF şi actualiza RIB/FIB pentru fiecare ruter. Pentru a realiza convergenţa sub o secundă, fiecare pas implicat în procesul de convergenţă trebuie să fie finalizat în milisecunde. Reacţia rapidă la un mediu de reţea perturbat poate conduce la o reţea care nu va converge pentru o anumită perioadă de timp. Pe de altă parte, în timp ce se încearcă să se realizeze o convergenţă rapidă, poate apărea instabilitatea reţelei. În funcţie de frecvenţa fluctuațiilor stării componentelor, o reacţie excesivă a protocolului de reţea, într-un mediu cu probleme, poate consuma CPU şi lărgime de bandă şi poate chiar afecta întreaga reţea. Ca urmare, stabilitatea este o proprietate cheie a unui mecanism aflat în recuperare, care trebuie să fie rezolvat cu o convergenţă rapidă.

În mediile deranjate, stabilitatea unei reţele poate fi menţinută prin controlul modului în care un mecanism de recuperare a reţelei reacţionează la schimbări. Tehnicile de amortizare care pot fi puse în aplicare la diferite niveluri într-o reţea sunt metodele de bază pentru a menţine stabilitatea în condiţii de reţea perturbată. Interfaţa de amortizare care foloseşte un algoritm de descompunere exponenţială şi un algoritm exponenţial de revenire, atât în generarea LSA-urilor cât şi în calculele SPF reprezintă unele dintre cele mai importante tehnici disponibile. Timpul de convergenţă mare, încărcarea excesivă de rutare pe rutere şi fluctuațiile numeroase ale rutelor în perioade scurte pot fi consideraţi ca indicatori de instabilitate într-o reţea.

3.4.1 Metode recomandate pentru realizarea mecanismelor de detectare rapidă a defecţiunilor

După cum s-a menţionat în capitolele anterioare, convergenţa rapidă se realizează printr-o descoperire rapidă a erorilor din reţea. Mecanismul de detectare a erorilor sub o secundă este una dintre cerinţele pentru a obţine convergenţa sub o secundă. O descoperire rapidă a erorilor din reţea se poate realiza cu ajutorul unui interval pentru parametrul *hello* sau mai preferabil, folosind tehnici bazate pe hardware şi BFD.

A) O valoare redusă pentru contorul *hello*

La o valoare redusă pentru contorul *hello*, se îmbunătăţeşte semnificativ timpul de detectare a erorilor dintre noduri, dar va creşte, deasemenea, sarcina pe fiecare ruter, când li se cere să proceseze pachetele *hello* rapide. Este posibilă reducerea intervalului *hello* la ordinul milisecundei, dar trebuie luate în considerare impactul asupra CPU şi nevoia de stabilitate a reţelei.

Utilizarea CPU în rutere poate creşte, din următoarele motive:

- deoarece ruterele procesează sute de pachete *fastHello* ale vecinilor;
- deoarece apar multe erori şi/sau recuperări concurente;
- deoarece într-o reţea pot apărea furtuni de LSA-uri.

Cum OSPF reacţionează rapid la fluctuațiile repetate care sunt apropiate în timp, mai multe rute dintr-o tabelă de rutare pot fi eliminate şi adăugate din nou într-o perioadă scurtă de timp, aceastea fiind caracterizate ca fluctuații ale rutelor. Fluctuațiile rutelor cauzate de fluctuațiile legăturilor, dintr-o reţea OSPF perturbată, conduc întotdeauna la instabilitate. Folosind un model real de reţea ISP (292 noduri şi 765 legături) Anindya Basu şi Jon G. Riecke [4] au observat o creştere de şase ori a numărului de fluctuații ale rutelor, atunci când se reduce contorul *hello* de la 500 ms la 250 ms. Ei au sugerat valoarea de 275ms ca fiind cea optimă. Cu toate acestea, se sugerează valoarea de 500 ms ca valoare optimă a contorului *hello,* cu capacitatea sa de detectare a defecţiunilor în aproximativ 2 secunde.[2]

În [5] se afirmă că cronometrul *hello* depinde de constrângerea fizică a legăturilor, cum ar fi zgomotul. Autorii din [9] au încercat să determine intervalul optim de *hello* pentru o reţea. Ei au sugerat că cronometrul optim *hello* depinde de nivelul aşteptat de congestie a reţelei şi de numărul de legături din topologia reţelei [9].

Autorii din [3] au arătat în reţeaua lor de simulare că detectarea rapidă este realizabilă într-o reţea stabilă, folosind un contor optim *hello*. Cu toate acestea, în condiții instabile, contorul dinamic *hello* creşte exponenţial, afectând timpul de detectare şi viteza de convergenţă, dar relativ reduce consumul de CPU şi lărgimea de bandă. Acesta oferă un mecanism mai tolerant la schimbările frecvente din reţea.

B) Tehnici bazate pe hardware

Tehnicile bazate pe hardware pot descoperi erori în zeci de milisecunde dar nu sunt întotdeauna disponibile, fie din cauza tipului de reţea, fie a costului implicat. O legătură gigabit punct la punct poate detecta erori între două noduri aproape instantaneu, folosind pulsul reţelei.

C) BFD

În cel mai bun scenariu, BFD poate ajuta cu 50ms mecanismul de detectare a erorilor [6]. Pentru a detecta erori în planul de expediere, este de preferat ca BFD să fie peste *hello-urile* sub o secundă, atât în ceea ce priveşte detectarea rapidă cât şi încărcarea de rutare. Acesta poate fi utilizat în asociere cu *fastHello* pentru a minimiza timpul de detectare în planul de control.

Găsirea unui interval optim pentru *hello* care evită alarmele false şi prelucrarea suplimentară într-o reţea nu este o sarcină uşoară. Mecanismele rapide de detectare a eşecurilor în medii de reţea perturbate pot conduce la instabilitate de exemplu, fluctuații ale legăturilor care provoacă fluctuații ale rutelor şi pot fi prevenite folosind tehnici de contor *hello* dinamic. Tehinicile de amortizare a evenimentului IP pot fi de asemenea folosite pentru a amortiza unele interfețe ale ruterele care oscilează, pentru a realiza un sistem mai stabil. Prin urmare, BFD poate fi folosit cu tehnicile de amortizare a evenimentul IP pentru a monitoriza orice interfață care fluctuează.

3.4.2 Îmbunătăţiri OSPF pentru optimizarea propagării evenimentului

Fiind unul dintre factorii care determină procesul de convergenţă, tehnica de propagare a evenimentului necesită o optimizare, pentru a realiza convergenţa rapid şi pentru a obţine o reţea mai stabilă, în acelaşi timp. Propagarea evenimentului poate fi afectată de mai mulți factori. Printre factori, furtunile de LSA-uri provoacă utilizarea intensă a CPU şi a memoriei la un ruter. Acest lucru face ca pachetele de intrare să fie întârziate sau anulate, în special în cazul în care se utilizează *fastHello*. Pierderea sau întârzierea pachetelor poate duce la ruperea adiacenţelor sau la o creştere a ratei de retransmisie a LSA-urilor, care în continuare pot pune reţeaua într-o stare instabilă.

Următoarele metode au fost declarate ca practici bune pentru a îmbunătăţi scalabilitatea şi stabilitatea reţelelor OSPF mari:

1. Alocarea unei priorități mari pachetelor *hello* şi pachetelor de confirmare a stării legăturii şi alocarea unei priorități reduse celorlate pachete.

2. Resetarea contoarelor de inactivitate pentru o adiacenţă la primirea oricăror pachete OSPF unicast sau pachete trimise către *AllSPFRouters* peste legătura punct-la-punct în loc de aşteptarea doar a pachetului *hello*. Adiacenţa este declarată defectă, numai la absenţa acestor pachete în perioada *RouterDeadInterval*.

3. Folosirea un algoritm exponenţial de revenire pentru a determina rata cu care LSA-urile sunt retransmise (*RxmtInterval*). Acest lucru ajută la reducerea ratei de retransmisie a LSA-urilor, în timp ce reţeaua este congestionată.

4. Detectarea congestiei implicit (Implicit Congestion Detection) la ruterele vecine şi luarea de măsuri cu ajutorul mecanismelor exponenţiale de revenire. Detectarea se face cu ajutorul nivelului de pachete LSA nerecunoscute. În cazul în care nivelul trece un anumit nivel numit "high-water mark", rata cu care LSA-urile sunt trimise la ruterul vecin este redusă, folosind mecanismul exponenţial de revenire. Rata creşte din nou exponenţial atunci când numărul LSA-urilor nerecunoscute de ruter ajunge la nivelul numit "low-water mark". Atât 3 cât şi 4 ajută la evitarea congestionării excesive la un vecin, diferenţa fiind următoarea: 3 se foloseşte doar pentru retransmisie iar 4 lucrează atât pentru formarea de LSA-uri noi cât şi pentru retransmisia de LSA-uri.

5. Reducerea numărului de adiacențe care se realizează simultan, deoarece trimiterea unui număr mare de adiacențe către vecini poate provoca o congestionare severă, datorită sincronizării bazei de date şi a activităţii de inundare a LSA-urilor. Aici, doar „n” număr de adiacențe se pot forma în același moment. Valoarea „n”, care este configurabilă, se bazează pe capacitatea de procesare a ruterelor, lărgimea de bandă totală disponibilă pentru traficul planului de control şi întârzierea propagării.

În plus faţă de diferitele metode menţionate mai sus, următoarele sunt unele dintre cele mai importante îmbunătăţiri disponibile, care optimizează procesul de propagare a evenimentului într-o reţea OSPF.

6. Tehnica de reglare LSA care ia în considerare convergenţa şi stabilitatea reţelei.

7. Tehnica group pacing delay care este disponibilă pe ruterele comerciale. LSA-urile sunt grupate pentru a actualiza împreună deci pentru a reduce numărul de pachete de actualizare LSA şi pentru a preveni furtunile LSA.

8. Setarea bitului *DoNotAge* din LSA pentru a evita reîmprospătările periodice, aceasta reducând semnificativ procesarea excesivă LSA a ruterelor. Ruterele inundă LSA-urile generate de ele cu bitul *DoNotAge* setat; acest lucru reduce supraîncărcarea traficului de protocol în reţeaua stabilă. Nu este nevoie de reinundarea LSA-urilor generate la fiecare 30 de minute, intervalul de reinundare fiind extins la un interval configurat de inundare-forţată.

3.4.3 Optimizări în calculele tabelei de rutare

Fiind unul dintre factorii care determină procesul de convergenţă, calculul tabelei de rutare necesită o optimizare. Mai jos sunt descrise unele dintre cele mai importante tehnici care pot fi utilizate pentru a îmbunătăţi procesul de calcul al tabelei de rutare.

Mecanisme	**Descriere**	**Avantaje/Dezavantaje**
Timp fix de aşteptare	Este o întârziere fixă care este pusă în aplicare între calcule SPF succesive.	Împiedică prea multe calcule ale tabelei de rutare după schimbarea topologică. Afectează durata de

		convergenţă, dar este bun pentru stabilitatea de rutare.
Reglarea SPF, (Cisco)	Timp de aşteptare iniţial mic, dar primirea unuia sau mai multor LSA-uri în perioada timpului de aşteptare dublează următorul timp de aşteptare până la o anumită valoare maximă. Timpul de aşteptare este resetat la o valoare mică în cazul în care niciun LSA nu este primit după *2* Max_hold*, (capitolul 3.3.3.1)	Ajută la realizarea convergenţei rapide. Încearcă să păstreze stabilitatea reţelei în perioade de perturbare a acesteia, care ar putea duce la mai multe calcule SPF.
Schema Juniper	Primele câteva calcule SPF se fac cu valori mici (fast operation mode). În cazul în care există prea multe calcule SPF la un anumit prag, trece la timp de aşteptare mărit (slow mode). Dacă nu este primit niciun LSA în timpul de aşteptare mărit, se resetează din nou la o valoare mică.	Modificările cu câteva calcule ale tabelei de rutare au ca rezultat convergenţa rapidă, dar afectează frecvenţa calculului SPF pentru modificări topologice de mare amploare.
Corelarea LSA-urilor	Încearcă să identifice schimbarea topologcă de bază prin corelarea LSA-urilor. (capitolul 3.3.7)	Convergenţă rapidă cu număr redus de calcule SPF.
iSPF	Evită calculelele SPF inutile prin examinarea LSA-urilor nou sosite împotriva SPT-urilor salvate. (capitolul 3.3.6.1)	Contribuie la viteza convergenţei, deoarece permite actualizări RIB rapide.
PRC	Presupune calcularea rutei parţiale peste prefixele schimbate. (capitolul 3.3.6.2 şi 3.3.6.3)	Îmbunătăţeşte viteza de convergenţă.

Tabelul 3.1 Tehnici de optimizare a calculului tabelei de rutare

3.4.4 Optimizarea actualizărilor FIB

iFIB este folosit pentru a optimiza timpul de actualizare FIB şi funcţionează cel mai bine cu iSPF (capitolul 3.3.9).

3.4.5 Obţinerea convergenţei de sub o secundă într-o reţea IP mare

În [10] autorii au arătat că este posibil să se realizeze convergenţa sub o secundă într-o reţea de scară largă (modelul de reţea ISP Tier-1, cu 200 de rutere în Europa, America şi Asia), fără a afecta stabilitatea reţelei. Simulările lor au fost realizate folosind protocolul IS-IS, care se aplică în acelaşi mod ca OSPF.

Experimentul de simulare conţine următoarele ipoteze şi abordări conservatoare.

- Legaturi pachet peste SDH (Synchronous Digital Hierarchy/SONET sau POS) în SP backbone sau BFD ceea ce înseamnă mecanisme de detectare a erorii sub o secundă;
- Contoare dinamice în generarea LSA-urilor şi calculelor SPF;
- Legătură de viteză 40 G, fără contor de pacing;
- iSPF şi iFIB;
- Prioritizarea prefixelor.

Folosind experienţa din acest studiu şi experimentul de simulare din acest proiect, cred că folosirea unei combinaţii dinte următoarele tehnici în diferite etape ale procesului de convergenţă permite realizarea unei convergenţe rapide, fără a afecta stabilitatea reţelei.

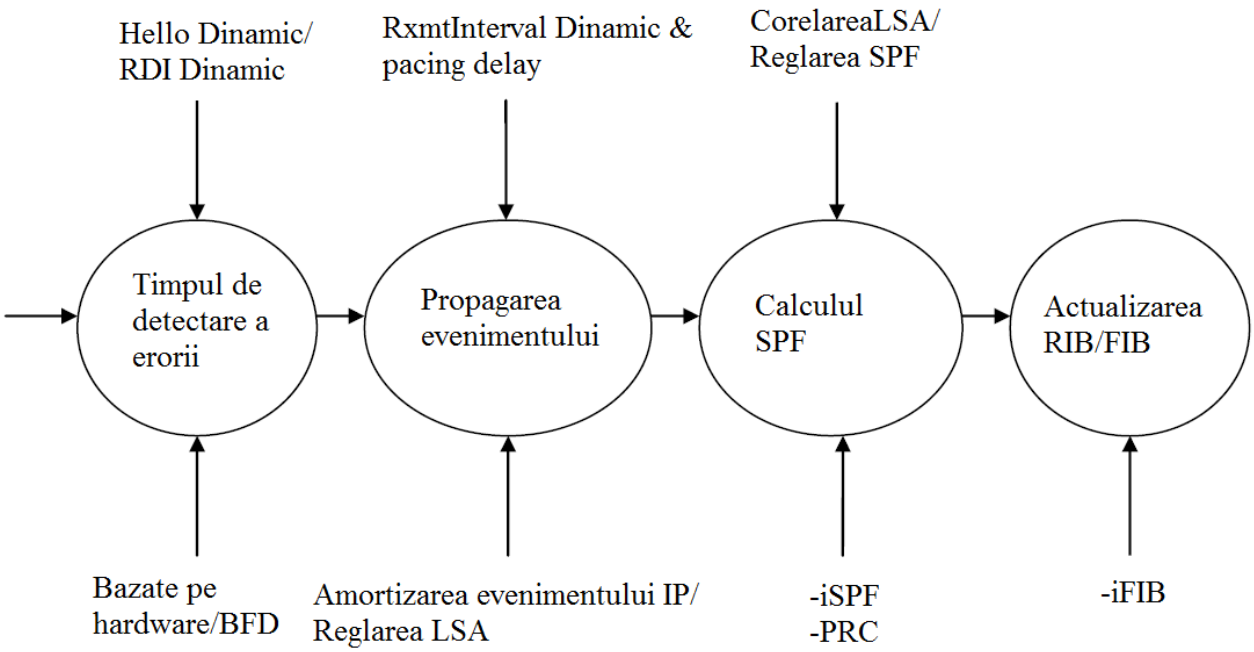


Figura 3.12 Tehnici pentru realizarea unei convergenţe rapide într-o rețea OSPF

4. Simulări

4.1. Simulatorul OPNET Modeler

Opnet Modeler este un puternic instrument de simulare folosit pentru a verifica diferite protocoale și diferite dispozitive. Interfața sa grafică oferă posibilitatea de a construi într-o perioadă scurtă de timp modele de rețele de dimensiuni mari formate din numeroase echipamente (rutere, switch-uri, calculatoare). Opnet modelază rularea diferitelor tipuri de protocoale de rutare în rețele astfel construite. Versiunea de Opnet IT Guru Academic Edition folosită în scopuri educaționale are următoarele caracteristici:

- Echipamnetele menționate în Opnet Modeler constau din obiecte, fiecare cu seturi mai reduse de atribute configurabile;
- Modele sunt introduse cu ajutorul editoarelor grafice;
- Simulările se generează automat;
- Statisticile specifice aplicațiilor pot fi colectate automat în timpul simulărilor;
- Mediu de simulare este scalabil și include suport pentru simulări paralele și distribuite.

Versiunea de Opnet IT Guru Academic Edition poate fi descărcată de la adresa: https://enterprise37.opnet.com/4dcgi/SIGNUP_NewUserOther

4.2 Simulări folosind Opnet Modeler

4.2.1 Simularea convergenţei protocolului OSPF prin evidenţierea intervalelor de trimitere a Hello-urilor

În această simulare, folosim termenii *fastHello* și *normalHello.* Termenul *fastHello* reprezintă un interval de trimitere a hello-urilor de o secundă și un interval de patru secunde pentru *routerDeadInterval.* Termenul *normalHello* reprezintă tradiționalul interval de trimitere a hello-urilor de zece secunde.

Topologia rețelei simulate în Opnet Modeler este reprezentată în figura 5.1. Rețeaua constă din 24 de noduri cu 49 de legături punct la punct și 98 de rute pentru fiecare.

Nu toți parametrii caracteristici protocolului OSPF pot fi configurați în Modeler, de exemplu parametrii *minLSAInterval/Arrival* sunt fixați cum sunt specificați în RFC 2328. Așadar, inafară de parametrii de retransmisie singurii parametri de interes pe care putem să îi folosim sunt intervalele de trimitere a hello-urilor și parametrii SPF întârziere/așteptare (SPF delay/hold - cu o valoare fixată pentru hold). Vom discuta mai jos ce valori vom alege, de ce alegem aceste valori și cum influențează ele convergența și stabilitatea rețelei.

Componentele rețelei folosite în simulare sunt rutere slip8_gtwy_adv, legături punct la punct de tip PPP_SONET_OC48 cu o rată de transfer de 2488.32 Mbps și un nod controller pentru a modela scenariile de defectare/restabilire (failure/recovery). Toate simulările din Opnet presupun o rețea OSPF cu o sigură arie cu legături punct la punct. Legăturile punct la punct au fost alese pentru a analiza problemele legate de convergență și stabilitate a unei rețele OSPF într-o formă simplă. În pricipal, această simulare încearcă să răspundă la următoarele întrebări legate de convergență rapidă și stabilitatea protocolului OSPF.

1. Ce efect are reducerea intervalului de trimitere a hello-urilor de la 10 la 1 secundă asupra vitezei de convergență și asupra utilizării procesorului?

2. Cum variază utilizarea procesorului și perioada de convergență între *fastHello* și *normalHello* în prezența unui nod care variază continuu (defectare/restabilire)?

3. Ce impact are asupra utilizării procesorului defectarea concurentă a nodurilor?

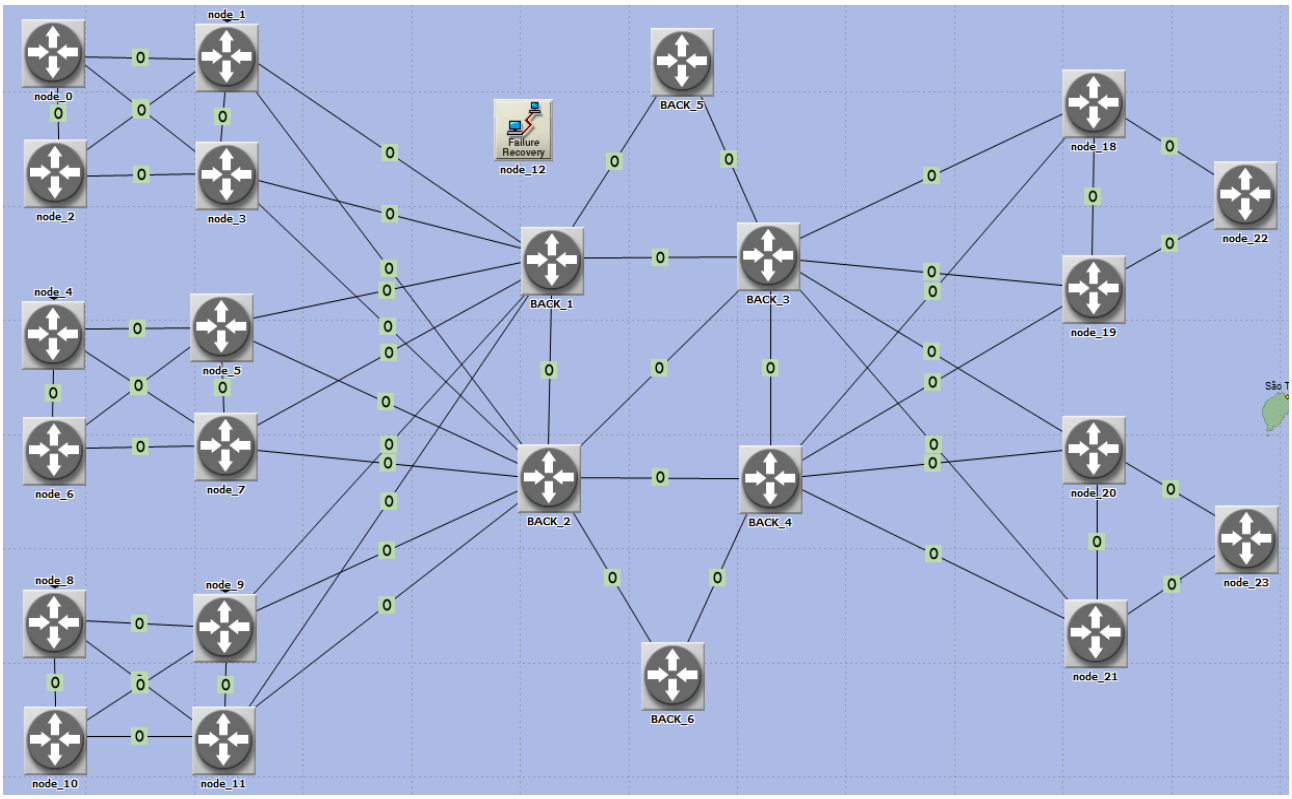


Figura 4.1. Topologia rețelei simulate în Opnet Modeler

În Opnet Modeler viteza de convergență a unui ruter poate fi măsurată folosind durata de convergență. Durata de convergență reprezintă timpul de când apare primul semn de activitate de convergență (inițial sau cu fiecare variație a rețelei) până la ultimul semn de convergență după care urmează un anumit interval, fără activitate de convergență. În acest caz, un semn de activitate de convergență reprezintă o recalculare a tabelei de rutare a protocolului OSPF. Această durată este reprezentată de valuarea axei Y din graficul duratei de convergență.

1. Prima parte a acestei simulări reprezintă comparația dintre utilizarea CPU și durata de convergență dintre *fastHello* și *normalHello*. Acest lucru este observat pe un nod de referință **node_1** care este supus a trei scenarii diferite ilustrate in figura 5.2 și în figura 5.3. Primul scenariu presupune defectarea nodului selectat **BACK_3** în timpul simulării, la momentul 100 secunde iar al doilea presupune restabilirea întârziată a nodului defectat la momentul 200. Acest lucru ne ajută să izolăm efectele unei defectări și a unei restabiliri. În al treilea scenariu, nodul selectat este defectat la momentul 300 și restabilit la momentul 320; în limita unei durate de mai puțin de 40 de secunde care reprezintă routerDeadInterval pentru hello-urile trimise normal. Acest lucru are loc deoarece presupune o scurtă repornire între 4 și 40 de secunde pentru care *fastHello* și *normalHello*

reacționează diferit. Stilul SPF utilizat este periodic cu valoarea 1 iar acest lucru a fost ales pentru a ține procesorul ocupat cât mai mult posibil sub prezența continuă a LSA arrival.

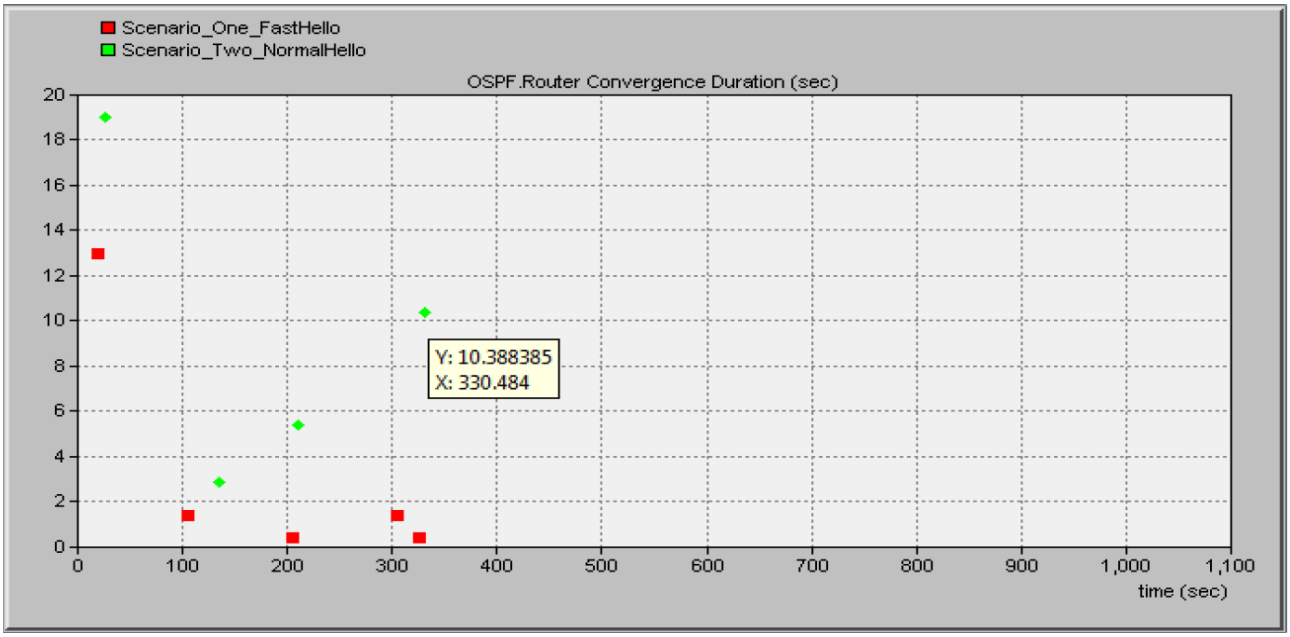


Figura 4.2 Durata de convergență între hello-uri fast și normal

	Convergența Inițială	BACK_3 Defectare (100 sec)	BACK_3 Restabilire (200 sec)	BACK_3 Defectare (300 sec)	BACK_3 Restabilire (320 sec)
fastHello	Y: 13, X: 19.48	Y: 1.43, X: 104.48	Y: 0.44, X: 205.48	Y: 1.40, X: 304,48	Y: 0.44, X: 325.48
normalHello	Y: 19, X: 25.48	Y: 2.91, X:134.48	Y: 5.42, X: 210.48	Y:10.39, X: 330.48	

Tabelul 4.1 Valorile duratei de convergență (valorie axei y) corespunzătoare timpului simulării (valorile axei x)

După cum se poate observa din tabelul 5.1, durata de convergență (valorile axei y) este în general mai mică pentru cazul *fastHello* datorită detectării și restabilirii rapide a defectărilor. Un alt punct interesant este acelă că în cazul *fastHello* pentru al treilea scenariu tabela de rutare este actualizată de două ori. Acest lucru se datorează expirării timpului de inactivitate în timpul repornirii scurte și astfel are loc o nouă calculare a tabelei de rutare presupunând că vecinul este mort.

Pe de altă parte în cazul *normalHello* are loc o singură calculare a tabelei de rutare și astfel conține doar o durată de convergență. Acest lucru se datorează faptului că defectarea nu a fost detectată într-un timp mai mic decât *routerDeadInterval*.

Utilizarea CPU corespunzătoare acestor puncte este ilustrată în figura 5.3. Utilizarea CPU crește de două ori în cazul *fastHello* pentru numerele corespunzatoare duratei de convergență.

Din acestea puntem trage concluzia că în timp ce cazul *fastHello* conduce la o convergentă mai rapidă acesta nu poate fi la fel de eficient în termenii utilizării CPU dacă într-o rețea au loc continuu reporniri scurte.

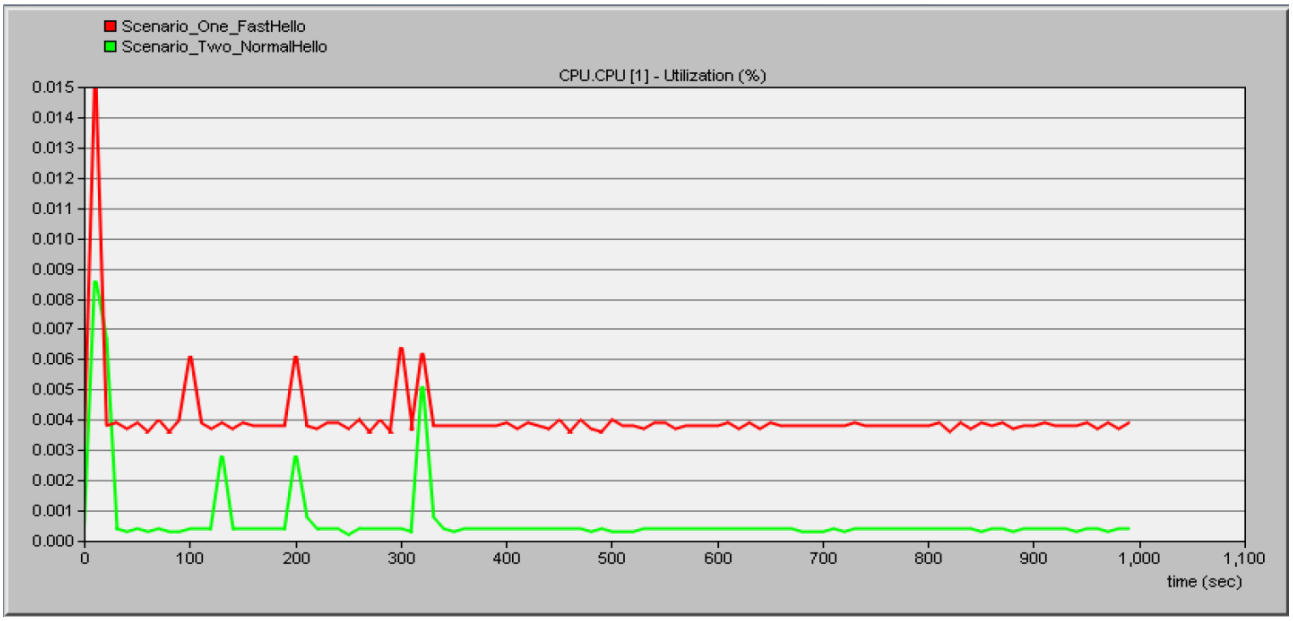


Figura 4.3 Utilizarea CPU între hello-uri trimise *fast* și *normal*

	Convergența inițială	BACK_3 Defectare (100 sec)	BACK_3 Restabilire (200 sec)	BACK_3 Defectare (300 sec)	BACK_3 Restabilire (320 sec)
fastHello CPU	Y: 0.01500	Y: 0.00610	Y: 0.00610	Y: 0.00640	Y: 0.00620
normalHello CPU	Y: 0.00860	Y: 0.00280	Y: 0.00280	Y: 0.00510	

Tabelul 4.2 Utilizarea CPU între hello-uri trimise *fast* și *normal*

2. În partea a doua a acestei simulări nodul selectat BACK_3 cade și se ridică (up and down) continuu la intervale de 20 de secunde. Acest lucru are loc între timpii simulării 100 și 300 de secunde. Stilul SPF este periodic cu valoarea 1. Scopul este de a arăta cum afectează procesorul reacția rapidă la asemenea defectări în cazul *fastHello* față de reacția relativ întârziată în cazul *normalHello*. Rezultatele simulării din figurile 4.4 și 4.5 ilustrează faptul că utilizarea procesorului crește în zonele corespunzătoarele duratei de convergență pentru ambele cazuri.

3. Cea de-a treia parte a simulării ilustrează în figura 4.6 influența asupra procesorului defectarea și restabilirea concurentă a nodurilor în cazul *fastHello*. Inflența asupra procesorului este analizată în trei scenarii diferite. Pentru aceasta folosim cele 12 noduri din partea stângă a topologiei din figura 4.1. În primul scenariu, cele 4 noduri din partea de sus a topologiei sunt căzute în timpul simulării la momentul 100 secunde și restabilite la momentul 200 secunde. În al doilea scenariu, 8 noduri sunt căzute în timpul simulării la momentul 100 secunde și restabilite la momentul 200 secunde. În cel de-al treilea scenariu, toate cele 12 noduri sunt căzute în timpul simulării la momentul 100 secunde și restabilite la momentul 200 secunde. Stilul SPF utilizat este periodic cu valoarea 1 pentru a crea presiune maximă asupra procesorului.

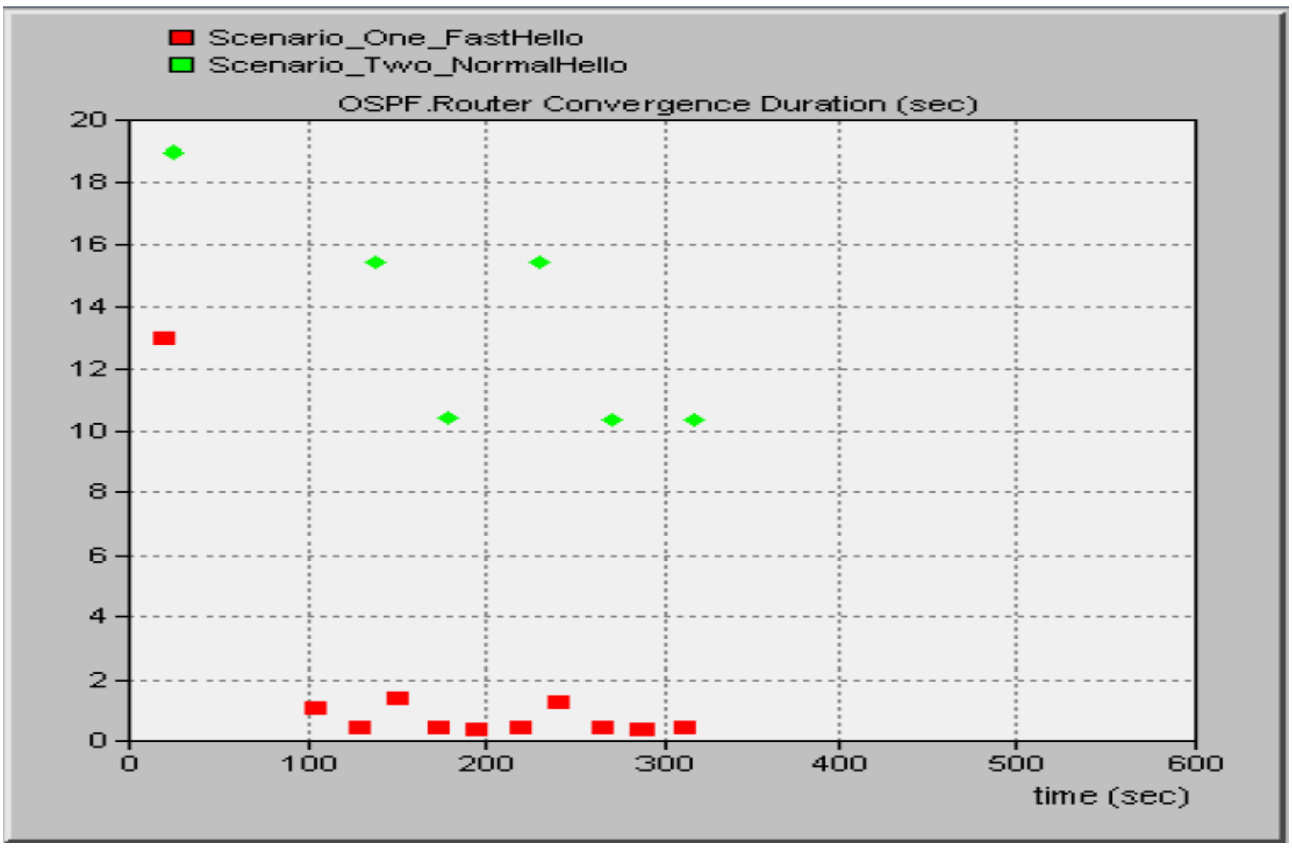


Figura 4.4 Durata de convergență în cazul unei modificări continuie up and down

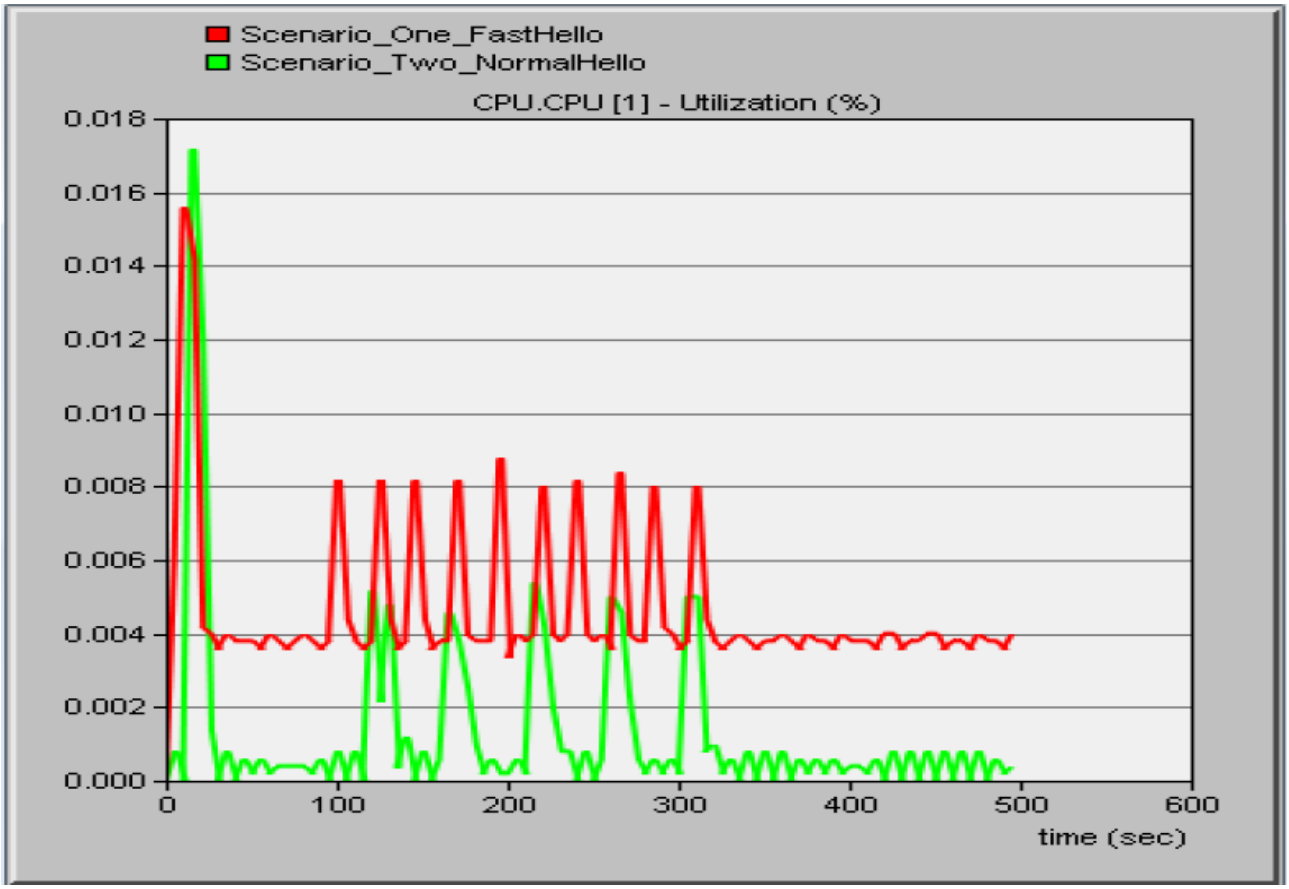


Figura 4.5 Utilizarea procesorului în cazul unei modificări continuie up and down

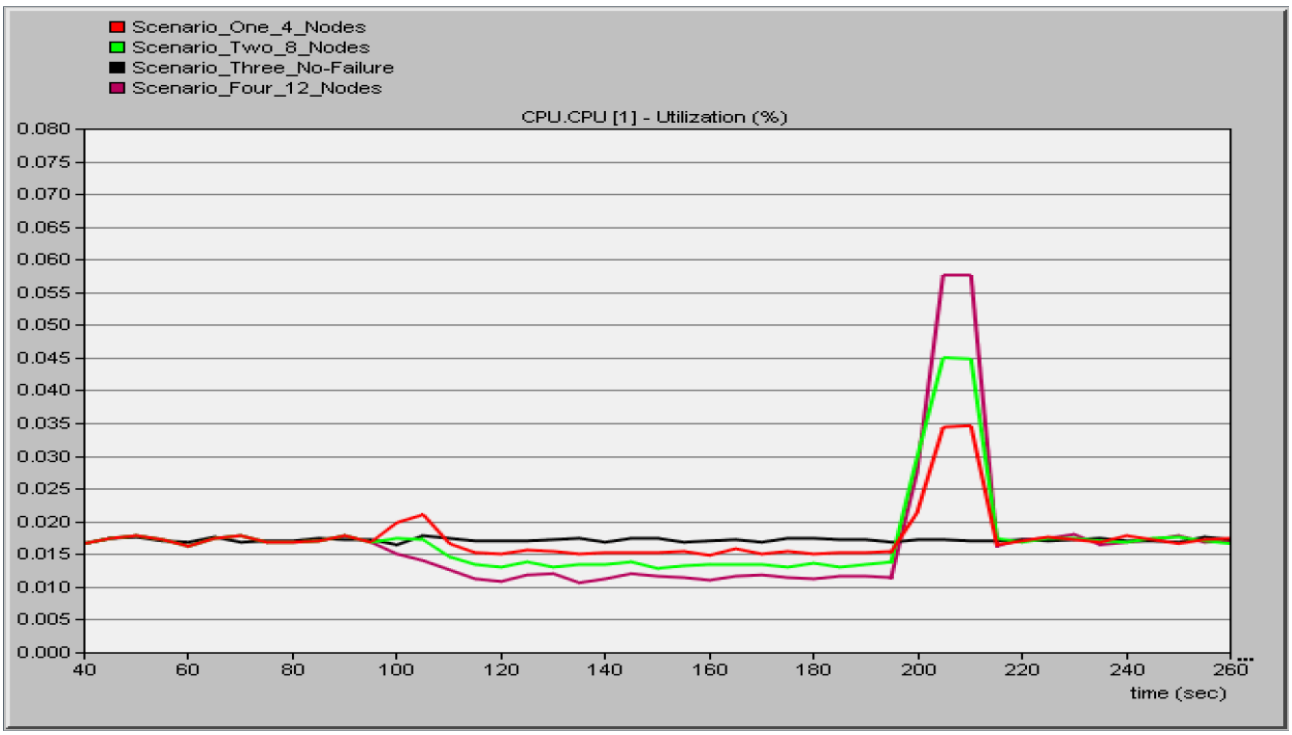


Figura 4.6 Utilizarea procesorului pentru *fastHello* în cazul eșecurilor concurente

Din acest rezultat putem să concluzionăm următoarele:

- Utilizarea procesorului este foarte dependentă de numărul de adiacențe pe care le are un ruter. După cum se poate observa din figura 4.6 utilizarea procesorului este invers proporțională cu numărul de noduri defecte în schimb este direct proporțională cu numărul de noduri aflate în restabilire. Acest lucru se datorează următorului fapt: cu cât numărul de adiacențe este mai mare cu atât crește și numărul corespunzător de LSA-uri pe care un ruter trebuie să le proceseze iar operațiile OSPF care includ și calculul SPF sunt strâns dependente de mărimea bazei de date a stării legăturilor (link state database) care conține toate aceste LSA-uri.
- O altă analiză interesantă este reprezentată de faptul că tendința consumului procesorului este în general descrescătoare pentru cazul defectărilor și crescătoare pentru cazul restabilirilor, schimbările relative nu sunt la fel. În exemplu se observă o creștere puternică pentru restabilire și o schimbare relativ mică pentru defectare. O mulțime de mesaje OSPF trebuie schimbate și procesate când relațiile dintre vecini trebuie reconstruite de la zero

4.2.2 Simularea unei rețele OSPF alcătuită din mai multe arii

Vom analiza comportamentul protocolului OSPF asupra unei rețele compusă din patru arii. Topologia rețelei este prezentată în figura 4.9.

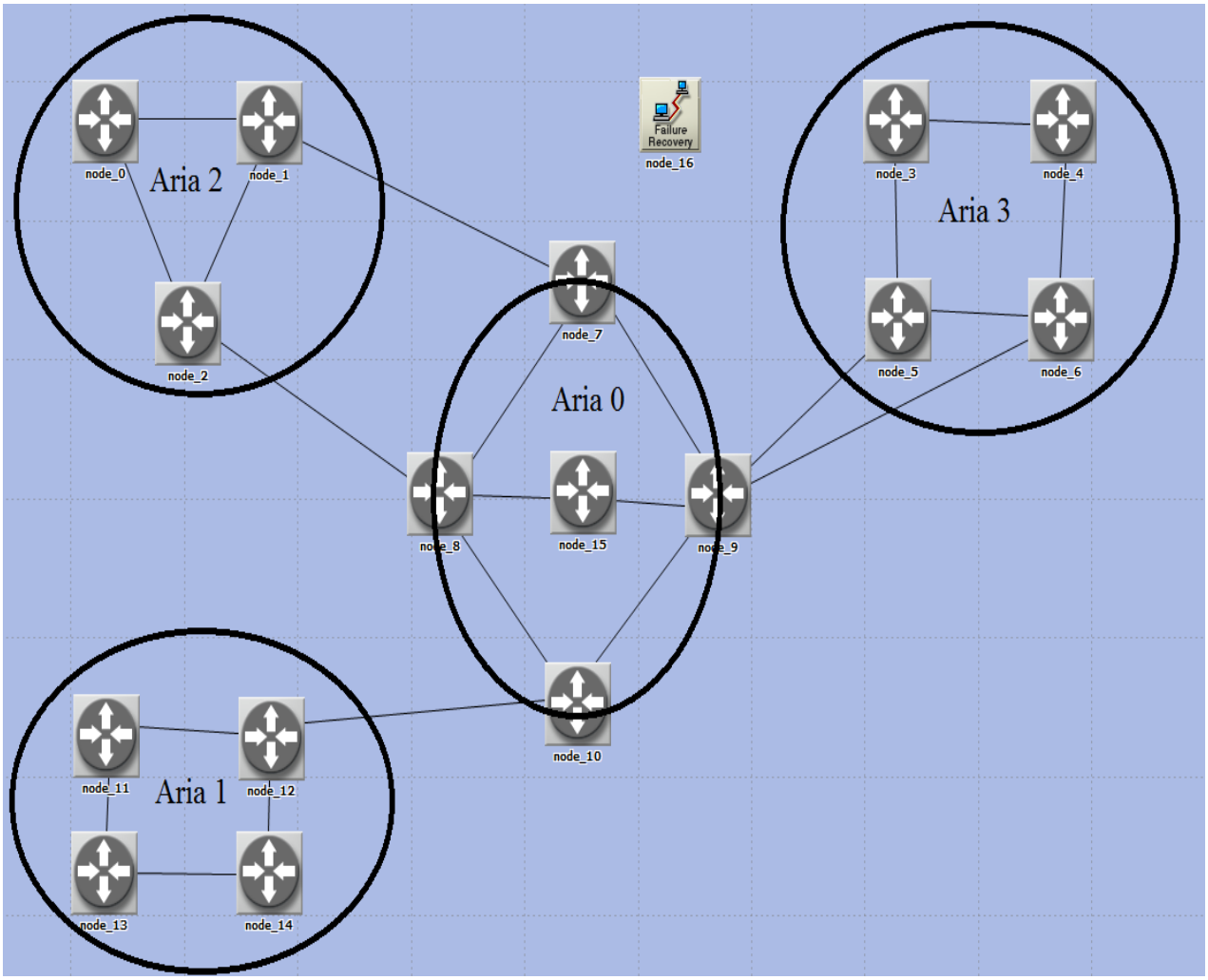


Figura 4.7 Arhitectura rețelei OSPF compusă din mai multe arii

Cele 4 arii ale rețelei sunt:

- aria 0 (aria de backbone) compusă din nodurile: node_7, node_8, node_9, node_10, node_15;
- aria 1: compusă din nodurile: node_11, node_12, node_13, node_14;
- aria 2: compusă din nodurile: node_0, node_1, node_2;
- aria 3: compusă din nodurile: node_3, node_4, node_5, node_6.

Vom analiza convergența protocolului OSPF în două situații simulare cu cele de la prima simulare: atunci când intervalul de trimitere a hello-urilor nu este modificat (*normalHello* la 10 secunde) și cu intervalul de trimitere a hello-urilor modificat la o secundă (*fastHello*).

Pe durata simulării de 1000 de secunde vor avea loc 4 evenimente:

- la momentul 300 cad două legături - legătura dintre node_1 și node_7 legătură care asigură conexiunea dintre Aria 0 și Aria 2 și legătura dintre node_5 și node_9 care asigură conexiunea dintre Aria 0 și Aria 3;
- la momentul 500 are loc revenirea legăturii dintre node_1 și node_7;
- la momentul 700 are loc revenirea legăturii dintre node_5 și node_9.

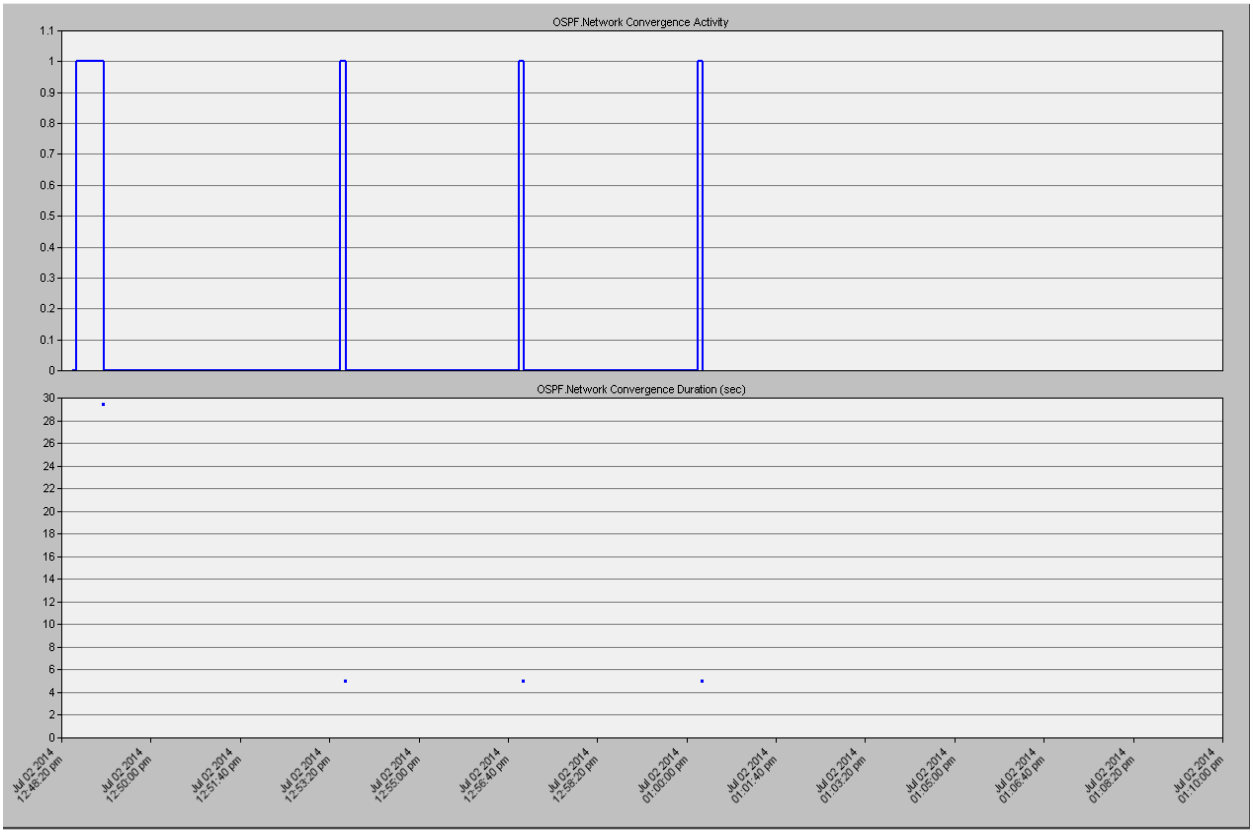


Figura 4.8 Activitatea de convergeță OSPF în cazul *normalHello*

Start Time	Start Node	End Time	End Node	Duration	Details
0.000000	Enterprise Network.node_0	0.000000	Enterprise Network.node_0	0.000000	Added Local route to destination 192.0.1.0/24
20.574339	Enterprise Network.node_6	34.988892	Enterprise Network.node_13	14.414553	Added OSPF 1 route to destination 192.0.5.0/24
300.000000	Enterprise Network.node_1	300.003184	Enterprise Network.node_13	0.003184	Deleted Local route to destination 192.0.8.0/24
500.000000	Enterprise Network.node_1	500.003832	Enterprise Network.node_13	0.003832	Added Local route to destination 192.0.8.0/24
700.000000	Enterprise Network.node_5	700.003184	Enterprise Network.node_13	0.003184	Added Local route to destination 192.0.12.0/24

Figura 4.9 Detalierea activitații de convergență în cazul *normalHello*

Start Time	Start Node	End Time	End Node	Duration	Details
0.000000	Enterprise Network.node_0	0.000000	Enterprise Network.node_0	0.000000	Added Local route to destination 192.0.1.0/24
20.574339	Enterprise Network.node_6	24.988892	Enterprise Network.node_13	4.414553	Added OSPF 1 route to destination 0.0.0.5/32
300.000000	Enterprise Network.node_1	300.003184	Enterprise Network.node_13	0.003184	Deleted Local route to destination 192.0.8.0/24
500.000000	Enterprise Network.node_1	500.003832	Enterprise Network.node_13	0.003832	Added Local route to destination 192.0.8.0/24
700.000000	Enterprise Network.node_5	700.003184	Enterprise Network.node_13	0.003184	Added Local route to destination 192.0.12.0/24

Fgura 4.10 Detalierea activității de convergență în cazul *fastHello*

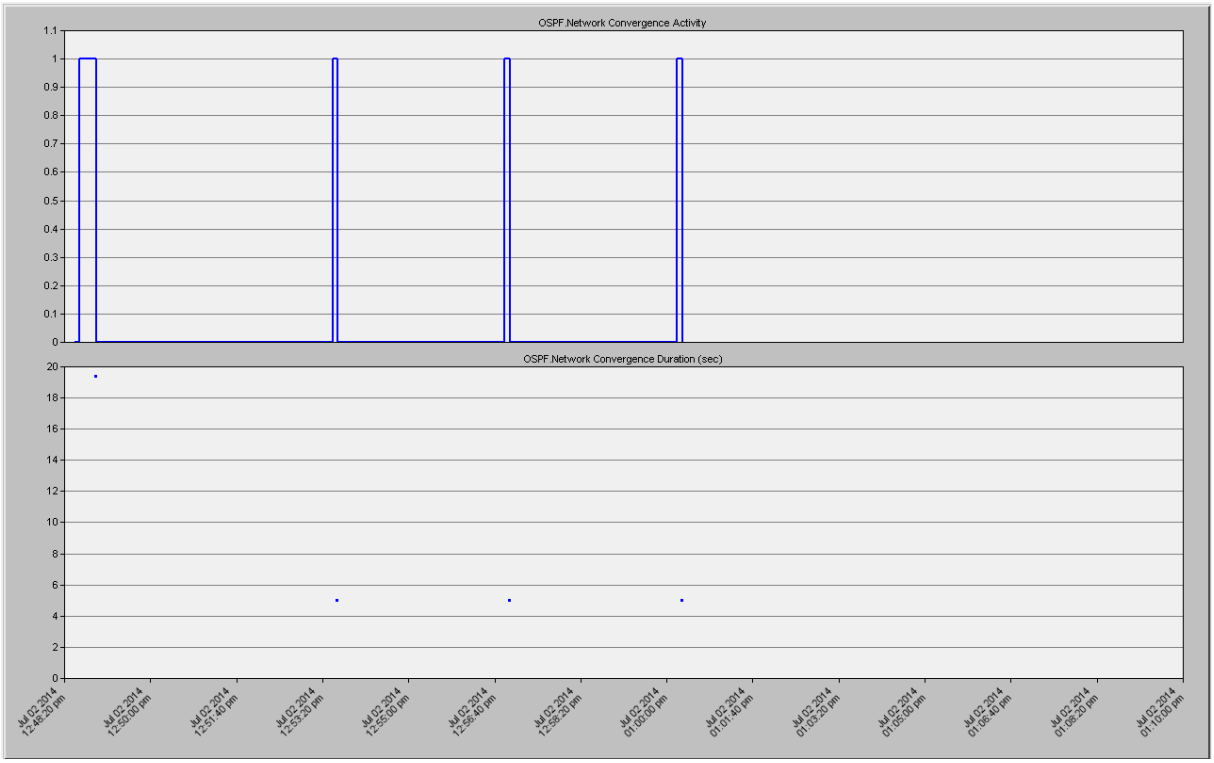


Figura 4.11 Activitatea de convergență OSPF în cazul *fastHello*

Îmbunătirea duratei de convergență în cazul *fastHello* se observă cel mai bine pentru convergența inițială a rețelei. Pentru cazul *normalHello* convergența durează 14.41 secunde iar pentru cazul fastHello convergența durează 4.41 secunde. Acestă imbunătire provine din faptul că procesul de determinare a vecinilor și de formare a adiacențelor se realizează mult mai rapid în cazul *fastHello.*

Node Name	[Total]
[Total]	3836
Enterprise Network.node_0	47
Enterprise Network.node_10	70
Enterprise Network.node_11	48
Enterprise Network.node_12	71
Enterprise Network.node_13	48
Enterprise Network.node_14	48
Enterprise Network.node_15	47
Enterprise Network.node_16	0
Enterprise Network.node_2	69
Enterprise Network.node_3	48
Enterprise Network.node_4	47
Enterprise Network.node_5	72
Enterprise Network.node_6	72
Enterprise Network.node_7	72
Enterprise Network.node_8	94
Enterprise Network.node_9	116

Figura 4.12 Numărul de pachete *Hello* create de fiecare nod în cazul *NormalHello*

Node Name	[Total]
[Total]	22556
Enterprise Network.node_0	477
Enterprise Network.node_10	720
Enterprise Network.node_11	483
Enterprise Network.node_12	718
Enterprise Network.node_13	484
Enterprise Network.node_14	484
Enterprise Network.node_15	477
Enterprise Network.node_16	0
Enterprise Network.node_2	713
Enterprise Network.node_3	482
Enterprise Network.node_4	478
Enterprise Network.node_5	716
Enterprise Network.node_6	727
Enterprise Network.node_7	722
Enterprise Network.node_8	955
Enterprise Network.node_9	1188

Figura 4.13 Numărul de pachete Hello create de fiecare nod în cazul *fastHello*

Se poate observa că numărul de pachete Hello trimise în cazul *fastHello* este aproape de 10 ori mai mare. Deasemenea putem observa că cele mai multe pachete Hello sunt create de nodurile node_7, node_8, node_9 și node_10 adică nodurile de la granița ariei 0 prin care notifică celelalte arii de modificările topologice apărute.

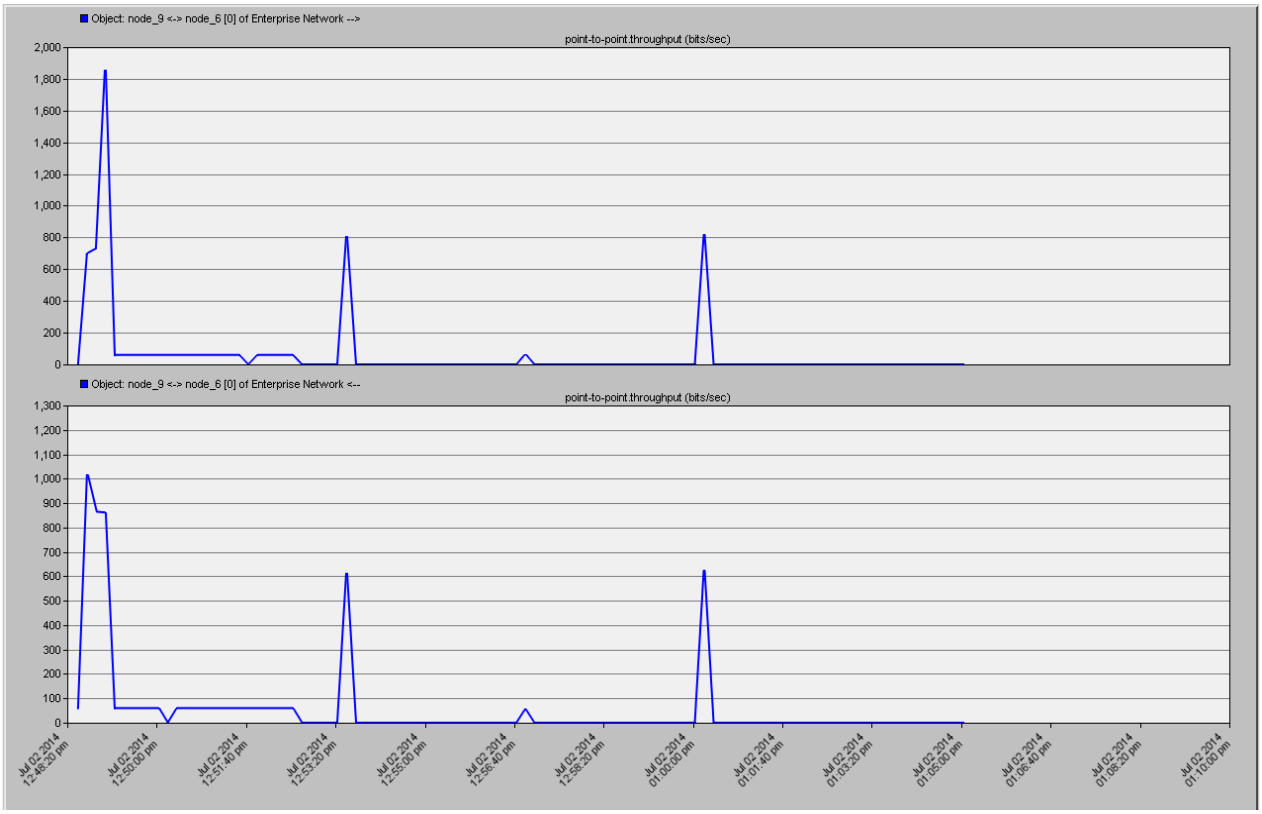


Figura 4.14 Throughput-ul legăturii node_6 - node_9 în cazul *normalHello*

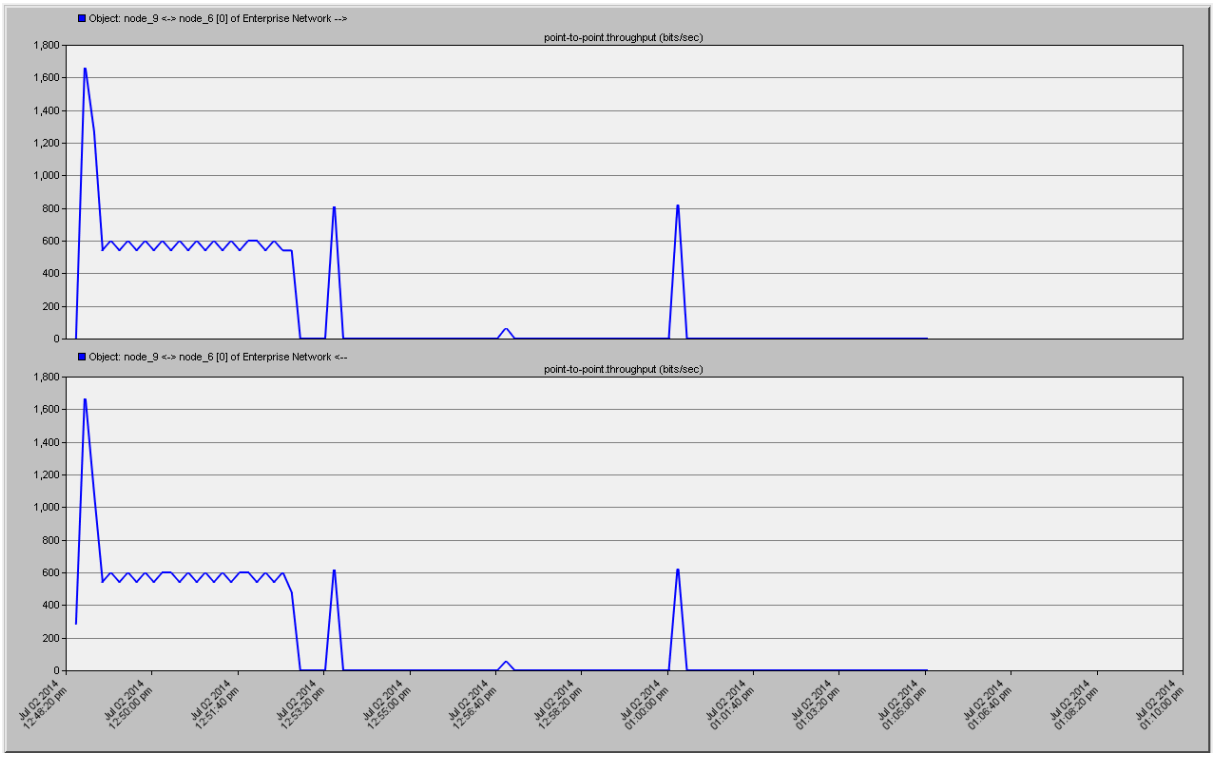


Figura 4.15 Throughtput-ul legăturii node_6 - node_9 în cazul *fastHello*

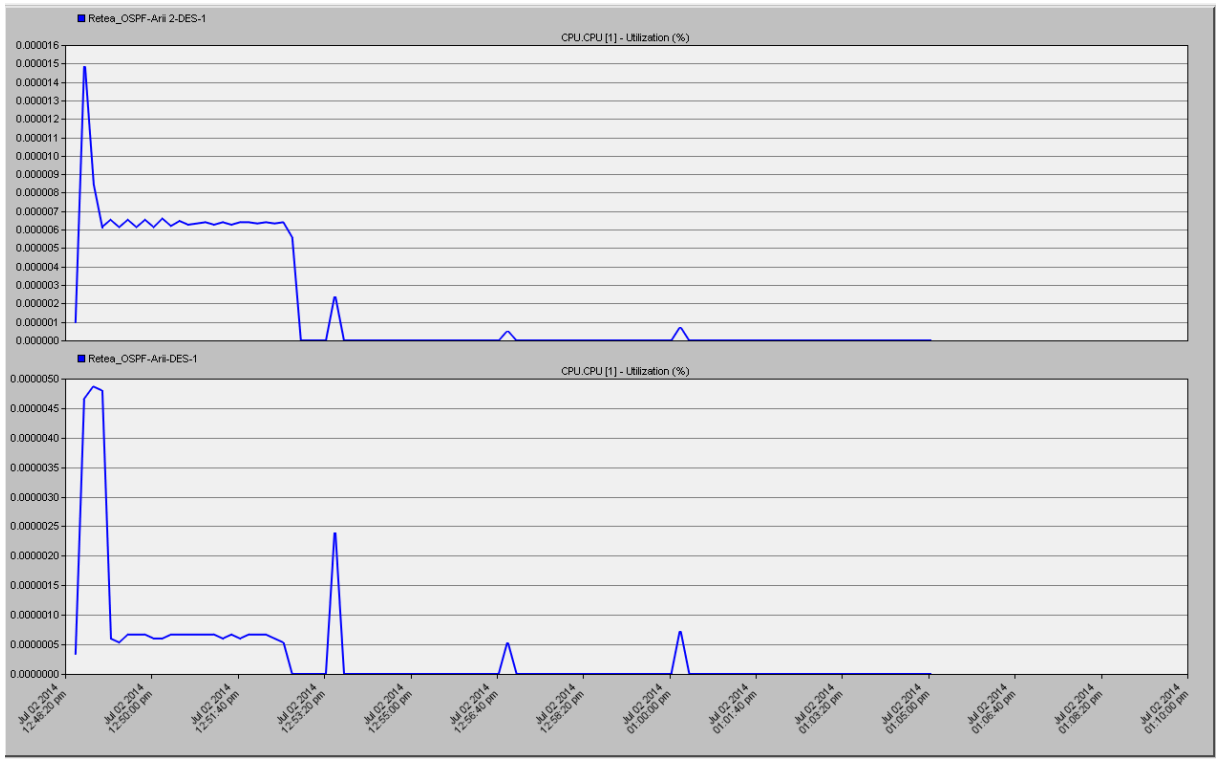


Figura 4.16 Activitatea procesorului nodului 9 în ambele cazuri

File Edit View Help

	Destination	Source Protocol	Route Preference	Metric	Next Hop Address	Next Hop Node	Outgoing Interface	Outgoing LSP	
1	0.0.0.1/32	OSPF 1	110	9	192.0.13.2	Enterprise Network.node_7	IF05	N/A	700.000
2		OSPF 1	110	9	192.0.14.1	Enterprise Network.node_10	IF04	N/A	700.000
3		OSPF 1	110	9	192.0.21.1	Enterprise Network.node_15	IF08	N/A	700.000
4	0.0.0.2/32	OSPF 1	110	9	192.0.13.2	Enterprise Network.node_7	IF05	N/A	700.000
5		OSPF 1	110	9	192.0.14.1	Enterprise Network.node_10	IF04	N/A	700.000
6		OSPF 1	110	9	192.0.21.1	Enterprise Network.node_15	IF08	N/A	700.000
7	0.0.0.3/32	OSPF 1	110	7	192.0.13.2	Enterprise Network.node_7	IF05	N/A	33.646
8		OSPF 1	110	7	192.0.14.1	Enterprise Network.node_10	IF04	N/A	33.646
9		OSPF 1	110	7	192.0.21.1	Enterprise Network.node_15	IF08	N/A	34.986
10	0.0.0.4/32	OSPF 1	110	7	192.0.11.2	Enterprise Network.node_6	IF07	N/A	300.000
11	0.0.0.5/32	OSPF 1	110	5	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
12	0.0.0.6/32	OSPF 1	110	5	192.0.11.2	Enterprise Network.node_6	IF07	N/A	300.000
13	0.0.0.7/32	OSPF 1	110	3	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
14	0.0.0.8/32	OSPF 1	110	3	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
15	0.0.0.9/32	OSPF 1	110	5	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
16		OSPF 1	110	5	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
17		OSPF 1	110	5	192.0.21.1	Enterprise Network.node_15	IF08	N/A	34.986
18	0.0.0.10/32	Direct	0	0	0.0.0.10	Enterprise Network.node_9	LB0	N/A	0.000
19	0.0.0.11/32	OSPF 1	110	3	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
20	0.0.0.12/32	OSPF 1	110	7	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
21	0.0.0.13/32	OSPF 1	110	5	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
22	0.0.0.14/32	OSPF 1	110	9	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
23	0.0.0.15/32	OSPF 1	110	7	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
24	0.0.0.16/32	OSPF 1	110	3	192.0.21.1	Enterprise Network.node_15	IF08	N/A	34.986
25	192.0.1.0/24	OSPF 1	110	10	192.0.13.2	Enterprise Network.node_7	IF05	N/A	700.000
26		OSPF 1	110	10	192.0.14.1	Enterprise Network.node_10	IF04	N/A	700.000
27		OSPF 1	110	10	192.0.21.1	Enterprise Network.node_15	IF08	N/A	700.000
28	192.0.2.0/24	OSPF 1	110	8	192.0.13.2	Enterprise Network.node_7	IF05	N/A	33.646
29		OSPF 1	110	8	192.0.14.1	Enterprise Network.node_10	IF04	N/A	33.646
30		OSPF 1	110	8	192.0.21.1	Enterprise Network.node_15	IF08	N/A	34.986
31	192.0.3.0/24	OSPF 1	110	8	192.0.13.2	Enterprise Network.node_7	IF05	N/A	700.000
32		OSPF 1	110	8	192.0.14.1	Enterprise Network.node_10	IF04	N/A	700.000
33		OSPF 1	110	8	192.0.21.1	Enterprise Network.node_15	IF08	N/A	700.000
34	192.0.4.0/24	OSPF 1	110	6	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
35	192.0.5.0/24	OSPF 1	110	6	192.0.11.2	Enterprise Network.node_6	IF07	N/A	300.000
36	192.0.6.0/24	OSPF 1	110	4	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
37	192.0.7.0/24	OSPF 1	110	4	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
38	192.0.8.0/24	OSPF 1	110	4	192.0.13.2	Enterprise Network.node_7	IF05	N/A	500.001
39	192.0.9.0/24	OSPF 1	110	6	192.0.13.2	Enterprise Network.node_7	IF05	N/A	33.646
40		OSPF 1	110	6	192.0.14.1	Enterprise Network.node_10	IF04	N/A	33.646
41		OSPF 1	110	6	192.0.21.1	Enterprise Network.node_15	IF08	N/A	34.986
42	192.0.10.0/24	OSPF 1	110	4	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
43	192.0.11.0/24	Direct	0	0	192.0.11.1	Enterprise Network.node_9	IF07	N/A	0.000
44	192.0.12.0/24	Direct	0	0	192.0.12.1	Enterprise Network.node_9	IF06	N/A	700.000
45	192.0.13.0/24	Direct	0	0	192.0.13.1	Enterprise Network.node_9	IF05	N/A	0.000
46	192.0.14.0/24	Direct	0	0	192.0.14.2	Enterprise Network.node_9	IF04	N/A	0.000
47	192.0.15.0/24	OSPF 1	110	4	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
48	192.0.16.0/24	OSPF 1	110	4	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
49	192.0.17.0/24	OSPF 1	110	6	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
50	192.0.18.0/24	OSPF 1	110	8	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
51	192.0.19.0/24	OSPF 1	110	8	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
52	192.0.20.0/24	OSPF 1	110	6	192.0.14.1	Enterprise Network.node_10	IF04	N/A	32.532
53	192.0.21.0/24	Direct	0	0	192.0.21.2	Enterprise Network.node_9	IF08	N/A	0.000
54	192.0.22.0/24	OSPF 1	110	4	192.0.21.1	Enterprise Network.node_15	IF08	N/A	34.986

Figura 4.17 Tabela de rutare a nodului node_9 în cazul *normalHello*

Din tabela de rutare se poate observa cum la momentele de timp 300, 500 respectiv 700 sunt introdu-se anumite înregistrări ceea ce reflectă modificările topologice care au avut loc. Deasemenea putem observa că toate celelalte înregistrări au fost introduse pe parcursul perioadei inițiale de conergență a rețelei.

Același lucru putem concluziona și pentru cazul *fastHello.*

	Destination	Source Protocol	Route Preference	Metric	Next Hop Address	Next Hop Node	Outgoing Interface	Outgoing LSP	
1	0.0.0.1/32	OSPF 1	110	9	192.0.13.2	Enterprise Network.node_7	IF05	N/A	700.000
2		OSPF 1	110	9	192.0.21.1	Enterprise Network.node_15	IF08	N/A	700.000
3		OSPF 1	110	9	192.0.14.1	Enterprise Network.node_10	IF04	N/A	700.000
4	0.0.0.2/32	OSPF 1	110	9	192.0.13.2	Enterprise Network.node_7	IF05	N/A	700.000
5		OSPF 1	110	9	192.0.21.1	Enterprise Network.node_15	IF08	N/A	700.000
6		OSPF 1	110	9	192.0.14.1	Enterprise Network.node_10	IF04	N/A	700.000
7	0.0.0.3/32	OSPF 1	110	7	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
8		OSPF 1	110	7	192.0.21.1	Enterprise Network.node_15	IF08	N/A	24.986
9		OSPF 1	110	7	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
10	0.0.0.4/32	OSPF 1	110	7	192.0.11.2	Enterprise Network.node_6	IF07	N/A	300.000
11	0.0.0.5/32	OSPF 1	110	5	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
12	0.0.0.6/32	OSPF 1	110	5	192.0.11.2	Enterprise Network.node_6	IF07	N/A	300.000
13	0.0.0.7/32	OSPF 1	110	3	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
14	0.0.0.8/32	OSPF 1	110	3	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
15	0.0.0.9/32	OSPF 1	110	5	192.0.21.1	Enterprise Network.node_15	IF08	N/A	24.986
16		OSPF 1	110	5	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
17		OSPF 1	110	5	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
18	0.0.0.10/32	Direct	0	0	0.0.0.10	Enterprise Network.node_9	LB0	N/A	0.000
19	0.0.0.11/32	OSPF 1	110	3	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
20	0.0.0.12/32	OSPF 1	110	7	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
21	0.0.0.13/32	OSPF 1	110	5	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
22	0.0.0.14/32	OSPF 1	110	9	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
23	0.0.0.15/32	OSPF 1	110	7	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
24	0.0.0.16/32	OSPF 1	110	3	192.0.21.1	Enterprise Network.node_15	IF08	N/A	24.986
25	192.0.1.0/24	OSPF 1	110	10	192.0.13.2	Enterprise Network.node_7	IF05	N/A	700.000
26		OSPF 1	110	10	192.0.21.1	Enterprise Network.node_15	IF08	N/A	700.000
27		OSPF 1	110	10	192.0.14.1	Enterprise Network.node_10	IF04	N/A	700.000
28	192.0.2.0/24	OSPF 1	110	8	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
29		OSPF 1	110	8	192.0.21.1	Enterprise Network.node_15	IF08	N/A	24.986
30		OSPF 1	110	8	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
31	192.0.3.0/24	OSPF 1	110	8	192.0.13.2	Enterprise Network.node_7	IF05	N/A	700.000
32		OSPF 1	110	8	192.0.21.1	Enterprise Network.node_15	IF08	N/A	700.000
33		OSPF 1	110	8	192.0.14.1	Enterprise Network.node_10	IF04	N/A	700.000
34	192.0.4.0/24	OSPF 1	110	6	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
35	192.0.5.0/24	OSPF 1	110	6	192.0.11.2	Enterprise Network.node_6	IF07	N/A	300.000
36	192.0.6.0/24	OSPF 1	110	4	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
37	192.0.7.0/24	OSPF 1	110	4	192.0.11.2	Enterprise Network.node_6	IF07	N/A	24.986
38	192.0.8.0/24	OSPF 1	110	4	192.0.13.2	Enterprise Network.node_7	IF05	N/A	500.001
39	192.0.9.0/24	OSPF 1	110	6	192.0.21.1	Enterprise Network.node_15	IF08	N/A	24.986
40		OSPF 1	110	6	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
41		OSPF 1	110	6	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
42	192.0.10.0/24	OSPF 1	110	4	192.0.13.2	Enterprise Network.node_7	IF05	N/A	24.986
43	192.0.11.0/24	Direct	0	0	192.0.11.1	Enterprise Network.node_9	IF07	N/A	0.000
44	192.0.12.0/24	Direct	0	0	192.0.12.1	Enterprise Network.node_9	IF06	N/A	700.000
45	192.0.13.0/24	Direct	0	0	192.0.13.1	Enterprise Network.node_9	IF05	N/A	0.000
46	192.0.14.0/24	Direct	0	0	192.0.14.2	Enterprise Network.node_9	IF04	N/A	0.000
47	192.0.15.0/24	OSPF 1	110	4	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
48	192.0.16.0/24	OSPF 1	110	4	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
49	192.0.17.0/24	OSPF 1	110	6	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
50	192.0.18.0/24	OSPF 1	110	8	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
51	192.0.19.0/24	OSPF 1	110	8	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
52	192.0.20.0/24	OSPF 1	110	6	192.0.14.1	Enterprise Network.node_10	IF04	N/A	24.986
53	192.0.21.0/24	Direct	0	0	192.0.21.2	Enterprise Network.node_9	IF08	N/A	0.000
54	192.0.22.0/24	OSPF 1	110	4	192.0.21.1	Enterprise Network.node_15	IF08	N/A	24.986

Figura 4.18 Tabela de rutare a nodului node_9 în cazul *fastHello*

4.2.3 Analiză comparativă a convergeței protocoalelor RIP, OSPF și EIGRP

Tabelul de mai jos prezintă o comparație a caractesticilor protocoalelor RIP, OSPF și EIGRP.

Caracteristică	RIP	OSPF	EIGRP
Algoritmul utilizat	Vector distață	Starea legăturilor	Ambele. Vector distanță și starea legăturilor. Protocol hibrid.
Metrica	Numărul de hop-uri	Depinde de lărgimea de bandă, întârziere, throughput și RTT (round trip delay)	Depinde de lărgimea de bandă, sarcină, întârziere, numărul de hop-uri și realibility
Numărul maxim de hop-uri	15. 16 hop-uri sunt considerate ca fiind infinit	Depinde de dimensiunea tabelelor de rutare	Maxim 255
Segmentarea sub-sistemului	Sistemul autonom este tratat ca un singur subsistem	Divide sistemul autonon în arii	Sistemul nu este divizat în arii
Integritate	Fără autentificare RIPv1. Autentificarea este adăugată în RIPv2.	Suportă autentificare	Suportă autentificare
Protocol/Port	UDP 520	IP 89	IP 88

Tabelul 4.1 Comparație între protocoalele RIP, OSPF și EIGRP

În cadrul acestei simulări am utilizat următoarele componente ale simulatorului Opnet:

- Application Config - acesta este un nod folosit pentru a seta aplcația din rețea
- Profile Config - reprezintă un nod folosit pentru a defini aplicații și pentru a le gestiona. Profilele utilizatorilor create pe aceste noduri sunt folosite pe diferite noduri din rețea pentru a genera trafic corespunzător nivelului aplicație. Nodul profile config este deasemenea folosit pentru a defini modele de trafic urmate de aplicații.
- CS_7000_6s_a_e6_fe2_fr4_slr_4_tr4 - reprezintă o configurație specifică a unui model de ruter IP gateway.
- Ethernet_wkstn - este un model de nod care reprezintă o stație de lucru cu aplicații client-server care ruleaza peste TCP/IP și peste UDP/IP. Stația de lucru suportă una dintre următoarele conexiuni Ethernet de 10mbps, 100mbps și 1000mbps.
- PPP_DS3 - legătură full duplex de 45Mb care conectează două noduri IP.
- 100BaseT - legăturile full duplex 100BaseT sunt folosite pentru a reprezenta conexiunile Ethernet.
- Failure Recovery - acest nod controller este folosit pentru a modela scenarii de tip eșec-revenire.

Pentru a observa convergența fiecărui protocol am folosit 3 scenarii diferite, un scenariu pentru RIP, un scenariu pentru OSPF și un scenariu pentru EIGRP. Pentru a studia rezultatele acestor scenarii am construit o rețea care conține cinci rutere CS_7000 interconectate cu legături de tip PPP_DS3 și 2 stații de lucru Ethernet care se conectează la ruterele vecine prin legături de tip 100BaseT. Cele două stații de lucru din rețea sunt stații care folosesc aplicația de videoconferință definită cu ajutorul nodului Application Config.

Modelul de rețea construit este ilustrat în figura de mai jos.

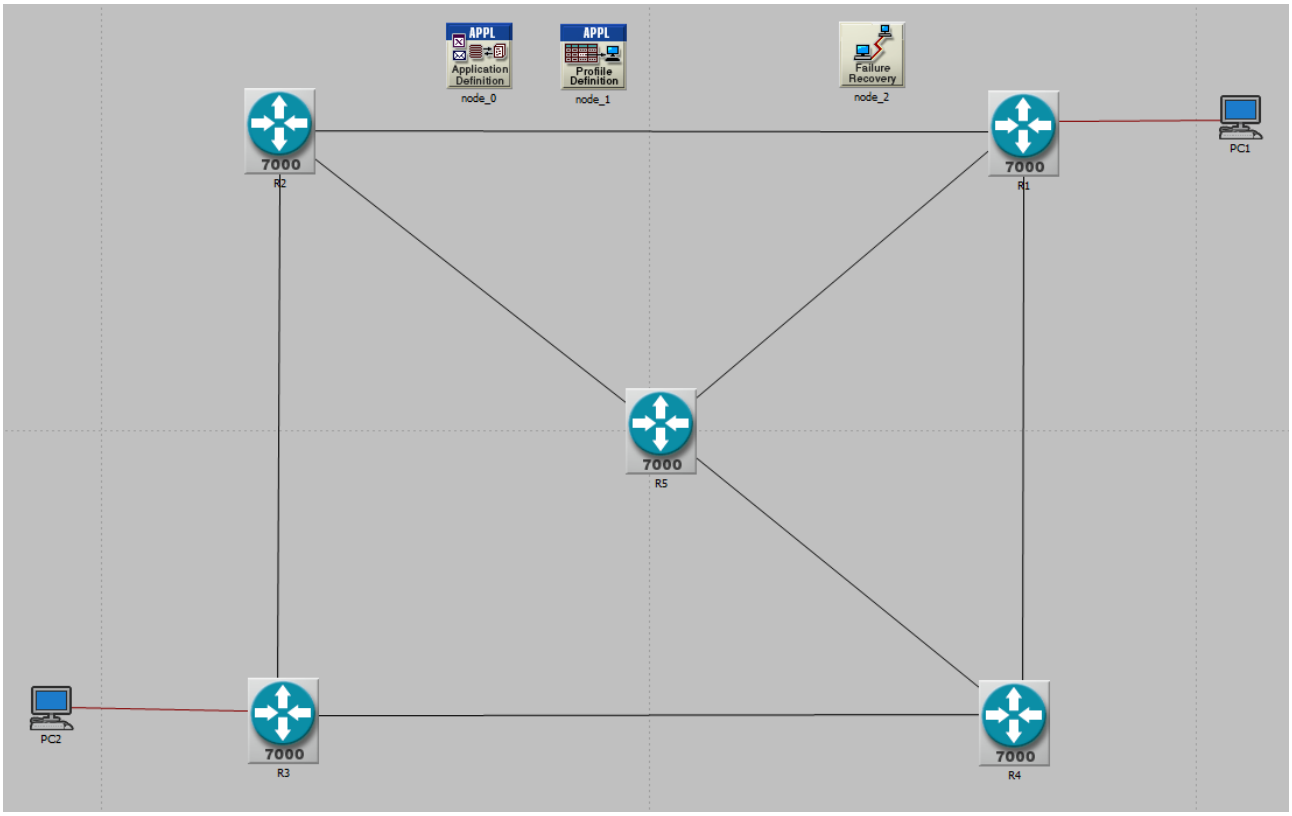


Figura 4.19 Topologia rețelei folosite pentru analiza convergenței protocoalelor RIP, OSPF, EIGRP

Pentru a verifica convergența celor trei protocoale această rețea a fost analizată în două situații. Aceste situații presupun eșecul și revenirea legăturii dintre ruterele R1 și R2 la momemtele de timp ilustrate în tabelele 4.2 și 4.3.

Stare	Momentul de timp (secunde)
Eșec	240
Revenire	420
Eșec	520
Revenire	580
Eșec	610
Revenire	620
Eșec	625
Revenire	626
Eșec	726
Revenire	826

Tabelul 4.2 Situația 1 pentru eșecurile și revenirile neregulate a legăturii dintre ruterele R1 și R2

Stare	Momentul de timp (secunde)
Eșec	30
Revenire	60
Eșec	90
Revenire	120
Eșec	150
Revenire	180
Eșec	210
Revenire	240
Eșec	270
Revenire	300
Eșec	330
Revenire	360

Tabelul 4.3 Situația 2 pentru eșecurile și revenirile regulate a legăturii dintre ruterele R1 și R2

Am analizat performanță protocoalelor de rutare RIP, OSPF și EIGRP în termeni de convergență pentru topologia din figura 4.7, în cele trei scenarii diferite, timp de 15 minute pentru situația 1 și timp de 6 minute pentru situația 2.

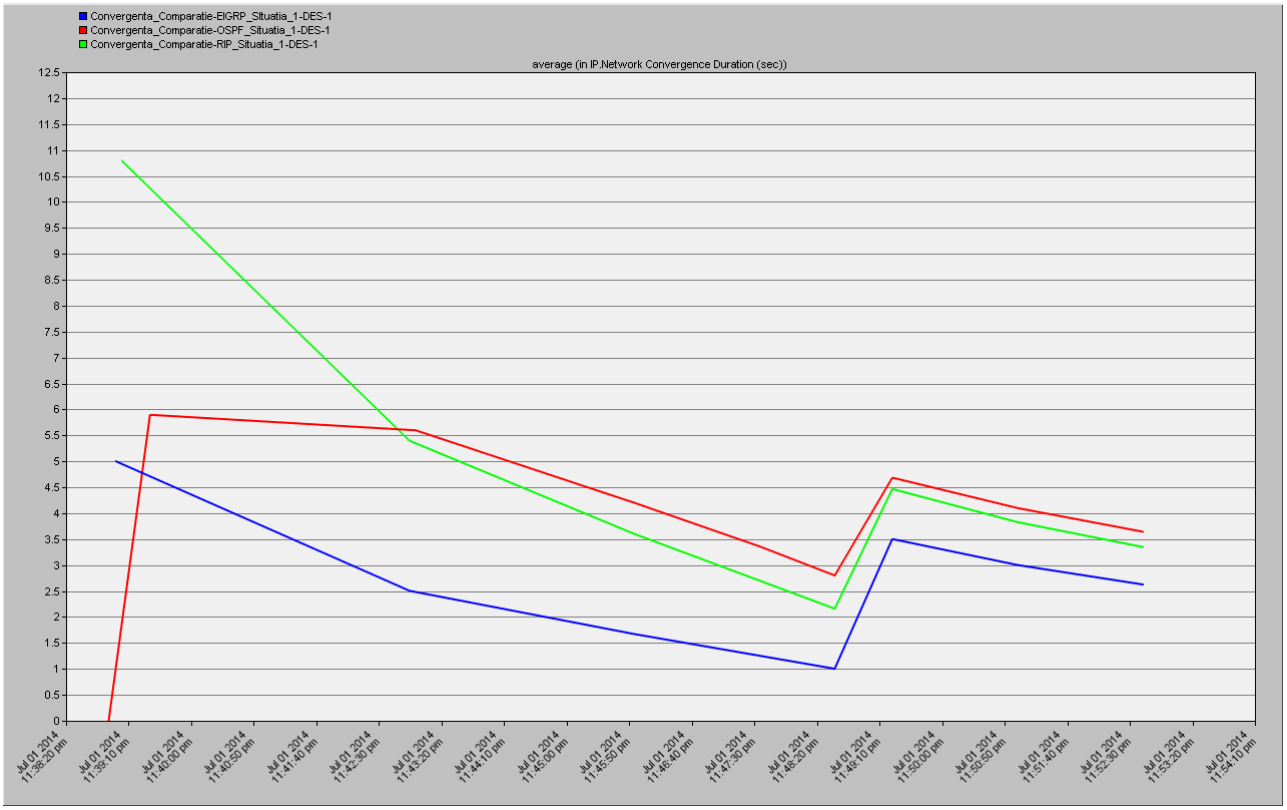


Figura 4.20 Convergență în cazul eșecurilor/revenilor neregulate

Protocoalele vectori distanță, cum este RIP, sunt mai încete în ceea ce privește convergență și adaptarea la schimbările topologice a rețelei. După o schimbare topologică și înainte ca toate ruterele să conveargă există probabilitatea să apară erori de rutare și pierderi de date. Protocoalele care folosesc starea legăturilor, cum este OSPF, au un timp de convergență mult mai mic. Convergența protocolului EIGRP este mai rapidă deoarece el folosește un algoritm numit DUAL,

care este rulat atunci când un ruter detectează că o anumită rută nu este disponibilă. Deoarece fiecare ruter OSPF are o copie a bazei de date a topologiei (topology database) și a tabelei de rutare pentru aria din care face parte, orice schimbări ale rutelor sunt detectate mai rapid decât în cazul protocoalelor vector distanță și astfel rutele care alternează sunt determinate.

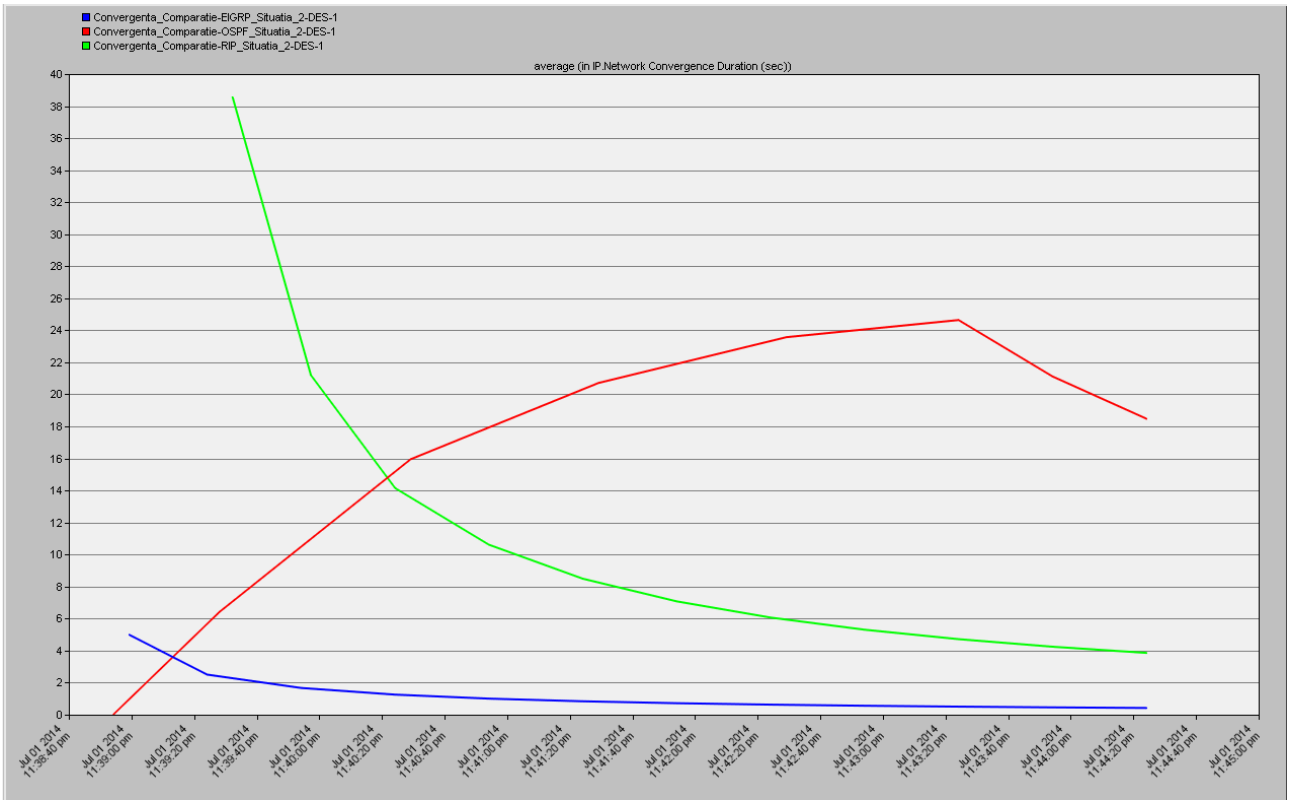


Figura 4.21 Convergeța în cazul situației eșecurilor/revenilor regulate

Concluzii

OSPF este unul dintre cele mai importante și cele mai intens studiate protocoale de rutare din rețelele furnizorilor de servicii (ISP) și ale multor companii. De la ultimul RFC lansat (RFC2328) au existat numeroase schimbări atât în configurația tradițională a parametrilor cum ar fi valorile contoarelor cât și schimbări datorate modalităților de lucru optimizate a operațiunilor sale. Cele mai importante dintre operațiunile OSPF care contribuie la convergența sa rapidă cum ar fi detectarea defecțiunilor, calculul SPF, generarea LSA și inundarea sunt controlate de contoare. Unele dintre aceste contoare au fost considerate în mod tradițional ca fiind constante arhitecturale în timp ce restul au fost setate o singură dată ca fiind valori fixe. Experiențele moderne și arhitectura rețelelor actuale sugerează o implementare dinamică a acestor contoare pentru o operare a rețelei sigură și o convergență rapidă atunci când apar eșecuri. Contoarele dinamice sunt în general setate la o valoare inițială mică bazată pe nivelul de stabilitate așteptat al rețelei iar aceste valori variază odată cu nivelul de încărcare al rețelei. Acest lucru conferă robustețe operării OSPF în momentele de instabilitate și îmbunătățește viteza de convergență în momentele de stabilitate a rețelei.

Această lucrare analizează modalitățile care ajută la realizarea unui timp de convergență mai scurt folosind contoarele protocolului OSPF. Rezultatele simulării arată că în timp ce *fasthello* trimise la o secundă reprezintă o alegere bună pentru descoperirea rapidă a vecinilor și pentru detectarea rapidă a eșecurilor aceasta nu poate fi la fel de eficientă pentru căderile repetate dar scurte ale legăturilor/ruterelor care sunt comune în implementările rețelelor reale.

Una din proprietățile dorite ale OSPF si ale altor protocoale de rutare este convergența rapidă atunci când apar schimbări în rețea. Experiențele indică o combinație de diferite tehnici care sunt utilizate în fiecare din fazele operaționale ale OSPF pentru a obține o convergență de sub o secundă. Rezultatele simulărilor care modelează rețelele reale ale furnizorilor de servicii, formate din sute de noduri arată că contorul *hello* poate fi redus în condiții de siguranță la câteva sute de milisecunde pentru o detectare mai rapidă a eșecurilor. Deși utilizarea de contoare IGP de menținere în viață (keep alive) rapide pare a fi ineficientă în ceea ce privește sarcina de procesare, o implementare dinamică a acesteia împreună cu o tehnică BFD independentă de protocolul de rutare este recomandată pentru detectarea rapidă a eșecurilor în rețelele cu difuzare (broadcast). BFD împreună cu tehnica amortizării evenimentului IP reprezintă combinația recomandată pentru detectarea rapidă a eșecurilor precum și pentru suprimarea alarmelor false. Reglarea LSA este folosită pentru a furniza generarea LSA-urilor dinamic și controlat pentru legăturile care fluctuează. Reglarea SPF este cel mai bine folosită împreună cu reglarea LSA pentru a furniza calcule SPF dinamice și controlate pentru LSA-urile auto-generate sau inundate. În timp ce folosim aceste tehnici de reglare este foarte important să adaptăm întârzierile inițiale cu mare atenție deoarece impactează durata de convergență pentru eșecuri ocazionale și tranzitorii ale rețelei.

Variantele noi ale algoritmului SPF, cum ar fi iSPF, cu un sistem dinamic de planificare sunt cele mai bune pentru a minimiza puterea în plus de procesare a CPU în rețelele mari. Contoarele adaptate dinamic pentru întârzierea ritmului pachetelor (packet pacing delay) sunt recomandate pentru a optimiza inundarea și retransmisia. Tehnicile incrementale de actualizare FIB oferă o performanță optimizată actualizând doar prefixele IP ale rutelor modificate și nu întreaga tabela de rutare.

Bibliografie

[1] J.Moy, OSPF Version 2, URL:http://www.ietf.org/rfc/rfc2328.txt, April 1998

[2] Goyal M, Soperi M, Bacce lli E, Choudhury G, Shaikh A, Hosseini H, and Trivedi K, "*Improving Convergence Speed and Scalability in OSPF: A Survey*", IEEE Communications Surveys & Tutorials ,Accepted For Publication , 11 December 2010.

[3] Fang W, Shanzhi C, Xin L, Yuhong L, "*A Route Flap Suppression Mechanism Based on Dynamic Timers in OSPF Network*", The 9th International Conference for Young Computer Scientists, IEEE,2008.

[4] Anindya B and Riecke Jon G, "*Stability Issues in OSPF Routing* ", SIGCOMM'01, August 27-31, 2001.

[5] Cengiz A, Van J, Haobo Y, "*Towards Millisecond IGP Convergence*", Internet Draft, Packet Design, November 2000.

[6] "*Bidirectional Forwarding Detection for OSPF*", Cisco, 2005.

[7] Goyal M, Xie W, Soperi M, Hosseini SH, Vairavan K, "*Scheduling Routing Table Calculations to Achieve Fast Convergence in OSPF Protocol*", Internet Draft November 29, 2006.

[8] Moy J, Pillay-Esnault P, Lindem A, "*Graceful OSPF Restart*", Network Working Group, RFC: 3623, November 2003.

[9] Mukul G, Ramakrishnan K. K. and Wu-chi F ,"*Achieving Faster Failure Detection in OSPF Networks*", in Proc. IEEE International Conference on Communications (ICC 2003), 2003.

[10] Pierre F, Clarence F, John E, Olivier B, "*Achieving sub-second IGP convergence in large IP networks*", IEEE, 2009.

[11] D. Katz, D Ward, Bidirectional Forwarding Detection (BFD), URL:http://tools.ietf.org/html/rfc5880, June 2010.

[12] J. Moy, Sycamore Networks, P.Pillay-Esnault, Juniper Networks, A, Lindem, Redback Networks, Graceful OSPF Restart, URL: http://tools.ietf.org/html/rfc3623, November 2003.

[13] J. Moy, Proteon, Inc. OSPF Version 2, URL: http://tools.ietf.org/html/rfc1583, March 1994.