Contents

List of Figures

List of Tables

List of Abbreviations

AR	Access Router
BGP	Border Gateway Protocol
CN	Corespondent Node
DNS	Domain Name Service
eBGP	external Border Gateway Protocol
ER	Exit Router
FA	Foreign Agent
fHA	flexible Home Agent
HA	Home Agent
iBGP	Border Gateway Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IPsec	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
MIP	Mobile IP
MIPv4	Mobile Internet Protocol version 4
MIPv6	Mobile Internet Protocol version 6
MN	Mobile Node
NEMO	Network Mobility
RFC	Request for Comments

RR	Return Routability
RTT	Round Trip Time
STCP	Stream Control Transmission Protocol
TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
UoC	University of Coimbra
UPC	Universitat Politecnica de Catalunya
VMCD	Virtual Mobility Control Domain

Chapter 1

Rezumat

Mobile IPv6 (MIPv6), sau, mai general, Mobile IP, e considerata ca fiind una dintre tehnologiile cheie care o să faciliteze mobilitatea pentru noduri conectate la internet. Cu "mobilitate" un utilizator poate să se deplaseze şi sa îşi schimbe punctul de ataşare la internet, pastrandu-şi adresa IP şi chiar şi conexiunile de reţea.

Protocoalele Mobile IP permit rutare de datagrame IP în internet independentă de locaţie, furnizand un mecanism eficient şi scalabil pentru roaming. Scalabilitatea este datorată faptului că mobilitatea nodului e realizată fară nevoia de a propaga rute specifice pentru el prin totată harta de rutare a intrnetului. Cât timp este în afara reţelei, fiecărui nod mobil (MN) îi este asociată cate o adresă Care-of, care îi identifică locaţia curentă, şi adresa de reşedinţă este asociata cu capătul unui tunel catre HAul lui. În alte cuvinte, nodurile îşi pot schimba punctul de ataşament la internet, fără a îşi schimba adresa IP de reşedinţă. Acest lucru le permite să menţină conexiunile de pe straturile mai înalte active în timp ce se deplasează ıin alte reţele.

În MIPv4 , când un MN se înregistreaza în o reţea vizitată, traficul său este tunelat de către agentul de mobilitate din reţeaua vizitata (FA) către agentul de mobilitate din reţeaua reşedinţa (HA), de unde este rutat către destinaţie. Acest lucru se întâmplă pentru toate comunicaţiile MNului. Acest comportament este similar unui VPN, unde un utilizator stabileşte un tunel securizat până în reţeaua proprie în care are încredere, şi îşi rutează tot traficul prin acest tunel. Această cale neoptimă de rutare este una dintre cele mai importante limitări ale MIPv4.

Mobilitatea Retelelor (NEMO) este o extensie a protocoalelor Mobile IP, care intenţionează sa rezolve problema cererii în creştere pentru suport de mobilitate pentru întregi reţele de dispozitive IP. Cu toate că rutarea neoptimizată este o limitare majoră pentru comunicaţii, protocolul de baza pentru suport NEMO nu încearcă să corecteze aceast impediment. Deci, având în vedere că NEMO flososeşte aceeaşi rută neoptimizată ca şi MIPv4, din punctul de vedere al studiului de faţa au aproape acelaşi comportament, şi vor fi tratate împreună.

În MIPv6, MNul nu are nevoie de un agentîn reţeaua vizitată (FA) care să-i tuneleze comunicaţiile, ci poate să comunice direct cu agentul din reţeaua de reşedinţă (HA) lui. Traficul cu excepţia absenţei FAului, urmează în schimb cam aceeaşi rută ca la restul protocoalelor. Un avantaj major în schimb îl consituie faptul că în MIPv6 MNul poate să execute procesul de Remote Routability, pentru a optimiza ruta traficului său, când comunică cu noduri din afara reţelei de reşedinţa. Deşi este o posibilitate teoretică, în aceasta lucrare se consideră că rutarea triangulară nu se foloseşte, şi aceasta din două motive. Primul ar fi faptul că un MN nu poate folosi ca şi adresă sursă adresa de reşedinţă cand comunică direct cu CN, deoarece în majoritatea reţelelor actuale acest fel de trafic e împiedicat de mecanismele de protecţie anti spoofing. În al doilea rănd, acest fel de rutare este suboptim, deci se preferă optimizarea rutei.

În timp ce aceste tehnologii sunt promiţaţoare, ele nu sunt înca mature şi pot fi imbunataţite incă în multe aspecte. Una dintre aceste probleme consta în faptul că traficul între Nodul Mobil şi alte noduri (Nodul Corespondent), e rutat prin HA, care este o entitate speciala care menţine legaturile între cele 2 adrese ale MN, adresa de reşedinţă, care identifică identitatea MNului, şi adresa Care-of, care identifică locaţia curentă a MNului. Astfel MNul e dependent de HA pentru a ramâne conectat. Fiindca un HA poate fi responsabil pentru MNuri multiple, defectarea unui singur HA poate avea ca rezultat pierderea conectiviţatii pentru multe MN-uri. Astfel HAul reprezintă punctul unde un singur defect poate fi fatal, pentru o reţea bazată pe Mobile IP. Mai mult, având în vedere că toate, (sau în cazul MIPv6 doar câteva dintre) comunicaţiile MNului sunt rutate prin HA, poate rezulta în legatura cu HAul sau chiar HAul însuşi sa devina factorul limitant al sistemului.

Pentru a soluţiona câteva dintre aceste probleme, mai specific, pentru a îmbunataţi fiabilitatea şi pentru a asigura echilibrarea încarcării, standardul MIPv6, permite amplasarea mai multor HAuri în aceeasi reţea. La defectarea unui HA, un alt HA îi poate prelua funcţiile celui defect. Astfel, serviciile oferite MNurilor de catre HAul defect sunt menţinute. Totusi, transferul serviciilor este problematic în oarecare măsură, deoarece soluţia este bazată pe actiunile MNurilor, care sunt obligate să descopere singure defecţiunea pe cont propriu, şi sa aleagă un alt HA. Acest lucru poate duce la o decţie târzie a defecţiunii, care imlpica întreruperea serviciilor furnizate de stratul reţea straturilor superioare, o incarcare suplimentara asupra MNurilor, şi necesitatea restabilirea asocierilor de IPsec.

Multe grupuri de cercetători au publicat diferite soluţii pentru aceste probleme. Majoritatea se bazează pe redundanţă sugerând folosirea mai multor HAuri, care sunt în aceeaşi stare de înregistrare, şi pe un set de algoritmi eficienţi de detecţie a defecţiunii şi înlocuire a HAului defect. Aceşti algoritmi sunt de două tipuri: unii dintre ei sunt executaţi de MNuri, alţii sunt perfect transparenţi din punctul de vedere a acestora. Totuşi majoritatea soluţiilor sunt incomplete, pentru că, de exemplu, sunt gândite specific pentru o tehnologie sau alta. De asemenea, nu iau în considerare necesitatea aplicabilitaţii acestor soluţii în reţele mari, cu zeci de subreţele, unde câteva HAuri pentru fiecare subreţea, nu este fezabil economic.

O altă propunere care nu necesita HAuri multiple pentru fiecare subreţea este Virtual Mobility Control Domain protocol. Acest protocol, în schimb, foloseşte external Border Gateway Protocol(eBGP), iar acest lucru îi conferă un efect impredictibil asupra scalabilitaţii sistemului de rutare bazat pe external BGP .

Studiul de faţa trateaza o arhitectură nouă de HA, care asigura, ca şi celălalte soluţii, fiabilitatea şi echilibrarea încărcării sistemului, necesitând doar un set de HAuri pentru o întreagă reţea. Un alt punct forte al acestei arhitecturi constă în faptul că spre deosebire de VMCD, nu interferează cu rutarea eBGP. Desi gândită iniţial pentru MIPv6, această soluţie nu are nici un elemet specific IPv6, şi poate fi implementată foarte uşor şi pentru IPv4. Ideea de bază a acestei arhitecturi este că locaţia MNului aflat în afara reţelei poate fi anunţat la routerele de margine ale retelei, astfel realizându-se o redirectare eficientă a pachetelor, fără implicarea HAului. Pe langă faptul că mareşte considerabil fiabilitatea cu un număr mic de HAuri, şi a faptului că practic elimină intreruperea serviciului către straturile superioare în cazul defectării unui HA, aceastăa abrodare "scuteşte" HAul şi reteaua de reşedinţă de o cantintate substantială de trafic. Lucrarea de faţa va descrie amănunţit incărarea unui astfel de HA, numit "flexible Home Agent" de către inventatorii acestei arhitecturi, precum şi sa o comare cu incărcarea unui HA standard.

Autorii flexible Home Agentului (fHA) au studiat deja din punct de vedere analitic performanţa acestei arhitecturi. Pe lânga acest lucru ei au creat un simulator bazat pe un model de mobilitate "Random Trip". Din pacate acest simulator este foarte imprecis, deoarece se beazeaza pe o serie de aproximari. In primul rând modelul de mobilitate folosit e doar un model. În al doilea rand, traficul care poate ocoli HA prin optimizarea rutei nu este tratat, pur şi simplu nefiind luat în considerare. In al treilea rand cantitatea şi tipul traficului generat de MNuri este aproximată foarte imprecis. Aici aceasată direcţie de cercetare este preluată de catre lucrarea de faţă, în care se prezintă o metodă mult mai precisa de analiză şi simulare a comportamentului a diferite arhitecturi de Mobile IP (MIP) . În abordarea prezentată se folosesc capturi de fluxuri de date din 2 instituţii de învaţamânt superior, ceea ce înseamna că simularea nu se bazează pe modele sau aproximări, ci pe date reale. Astfel aproximările şi inacurateţea simulatorului descris mai sus sunt eliminate. Într-adevar, şi metoda aceasta nu e lipsita de limitările ei, însa e mult mai precisă decat o metoda bazata pe studiu analitic sau chiar pe un model de trafic.

Această metodă de evaluare nu este pe deplin nouă. Ea a mai fost folosită în literatura de specialitate pentru a analiza comportamentul, de exemplu a unui HA MIPv6, supus la trafic real. Spre deosebire de implementările de până acum, aceast studiu nu îşi propune să analizeze sau să modeleze încărcarea unui HA în amanunt, la un nivel înalt de granularitate, ci să creeze un mediu de simulare care să faciliteze compararea din punct de vedere al înc'rcării diferitor arhitecturi de HA. Un accent deosebit s-a pus şi pe accurateţea rezultatelor

Arhitectura fHAului e bazată pe simplitate şi flexibilititate. Astfel sunt posibilie implementări cu unul sau mai multe fHAuri care sa deservească o întreagă reţea. Acestea vor fi identificate de căte o adresă unicast, căreia îi vor fi timise înregistrările de catre MNuri. La recepţionarea unui msesaj de înregistrare, fHA îl valideaza şi trimite un mesaj de rutare anunţând noua rută câtre MN. Această informaţie este trimisa câtre fiecare ER. Pe lânga acest lucru, fHA anunţa ruta şi routerului de acces al subreţelei MNului. În acest moment întreaga reţea cunoaşte noua locaţie a MNului.

Când comunică cu un CN prin HA, MNurile nu trimit pachetele către fHA ci către o adresă anycast deţinută de routerele de margine a retelei. În acest mod, un ER oarecare va primi pachetele de date ale MNului, le va decapsula, şi le va înainta către CN. În mod similar, CNurile trimt pachete către adresa de reşedinţă a MNului. Când un astfel de pachet este recepţionat, routerul de margine (ER), datorită rutei instalate împrealabil de fHA, ştie că MNul nu este acasă şi încapsulează pachetul şi îl înaintează către noua locaţie a MNului. Astfel, această arhitectură face un management eficient al traficului între MN şi nodul corespondent (CN) , deoarece pachetele sunt "respinse" la nivelul ruterelor de margine. De asemenea traficul intern al reţelei este redus considerabil.

Referitor la comunicaţiile de la MN către reteaua de reşedinţă, MNul adresează pachetele sale protejate prin Internet Protocol Security (IPSec) fHAului care la rândul lui decapsulează pachetele şi le înaintează destinaţiei propriuzise. Acesta adresează pachetele sale adresei de reşedinţă a MNului. Dar, deoarece fHA a aunnţat routerului de acces al subreţelei MNului o nouă rută pentru MN, routerul de acces stie ca MNul nu e în reţea şi încapsulează pachetul trimiţându-l la fHA. Pe perioada cât timp MNul nu este în reţea, routerul de acces al subreţelei joaca rolul MNului, interceptând astfel comunicaţiile din reţea destinate MNului, pe care le înaintează printr-un tunel fHAului. O observaţie importantă este aceea că un MN aflat în afara reţelei de reşedinţă, poate sa adreseze traficul său direct fHAului. Acest lucru conferă retrocompatibilitate cu HAuri standard.

Pentru a evita necesitatea unei configurări manuale, fHAurile işi anunţă prezenţa în reţea. Acest lucru permite routerelor de acces sa descopere fHAul automat. De asemenea, acest mecanism ajuta la echilibrarea incarcarii fHAurilor, deoarece MNurile işi aleg singure fHAul pe baza unei "valori de preferinţa" din anunţul acestuia, care e decrementată de feicare router. Astfel MNul isi poate alege fHAul cel mai apropiat.

Unul din beneficiile majore ale fHAului este flexibilitatea acestuia. Pe deoparte, arhitectura fHA poate deservi toate MNurile dintr-o reţea cu doar unul, sau cu mai multe fHAuri. Dacă mai multe fHAuri exista în reţea, MNurile îl vor selecta pe cel mai apropiat pe baza valorii de preferinţă. Astfel incarcarea este distribuită între diferitele fHAuri, şi fiecare fHA nu va proecesa decât mesaje de semnalizare şi comunicaţii de către şi înspre reţeaua de reşedinţă (acţionând astfel ca un gateway Virtual Private Network(VPN)
nomenclatureVPNVirtual Private Network). Comunicaţiile intre MNuri şi CNuri sunt procesate de

catre routerele de margine. Pe de altă parte arhitectura aceasta este transparentă pentru MNuri care folosesc HAuri standard, ambele tehnologii putând coexista în aceeaşi reţea.

Pentru realizarea simulatorului folosit în acest stiudiu au fost folosite urmatoarele tehnologii:

Cisco NetFlow. NetFlow e un protocol dezvoltat de fima Cisco. Routerele Cisco care au modulul NetFlow activat, generează înregistrări NetFlow în care înmagazineaza detalii despre fluxurile de date pe care le rutează. Aceste înregistrări sunt exportate de către router folosind datagrame UDP(User Datagram Protocol) sau SCTP(Control Transmission Protocol) şi colectate folosind un colector specializat. Datele folosite în acest studiu sunt alcătuite din astfel de inregistrari netflow. Formatul este NetFlow V5, care este pe departe cel mai raspândit format în ziua de astăzi. Cele mai relevante informaţii pentru studiul de faţă prezente în formatul V5 sunt următoarele: moment de început, moment de sfarşit. adresă IP sursă, port sursă, adresă IP destinaţie, port destinaţie, protocol strat 4, număr pachete şi număr octeţi.

"Flow-tools". Flow-tools este o colecţie de programe şi utilitare folosite pentru a captura şi prelucra înregistrări NetFlow. Pentru înregistrarea datelor exportate de router s-a folosit utilitarul flow-capture. Pentru concatenarea datelor până la obţinerea unei granularitaţi acceptabile (un fişier pentru fiecare 24 ore de captură) s-a folosit utilitarul flow-cat. Pentru a transforma înregistrările din fomatul brut, greu de prelucrat, în format text, s-a folosit utilitarul flow-print. Cel mai important ca şi funcţionalitate pentru proiect a fost utilitarul flow-nfilter. Acesta a fost folosit pentru filtrarea înregistrărilor, pe diferite criterii definite intr-un fisier de filtre facut special pentru acest proiect, în categorii de trafic, după impactul fiecarei categorii asupra HAului.

Principalii algoritmi folosiţi în simulatorul creeat şi folosit în acest studiu au fost implementaţi în limbajul de programare/scriptare Perl. Acesta este un limbaj interpretat, de nivel înalt, şi a fost ales din următoarele motive: Primul motiv este faptul că Perl este foarte eficient în manipulare de text. Înregistrările NetFlow sunt stocate întru-un format brut comprimat, care este destul de greu procesabil. Folosind utilitarul flow-print, acestea pot fi transformate în format text, care este mult mai usor de procesat şi pentru ochiul uman şi pentru Perl datorită uneltelor lui puternice de procesare a textului. Faptul ca Perl este atât de puternic în prelcrari de text se datoreaza standardului sau bine definit de scriere a expresiilor regulare şi motorului sau deosebit de eficient de potrivire a şirurilor de caractere cu aceste expresii regulare. Al doilea motiv este faptul că Perl este capabil să lucreze cu seturi mari de date. Acest factor a fost important deoarece, de exemplu, unul din seturile de date pe care le-am folosit este o captura de 36 946 620 fluxuri de date fiecare dintre ele avand 7 campuri de diferite feluri. Nu în ultimul rând, Perl are un ciclu rapid de dezvoltare şi punere în lucru a programelor. Acest lucru se datorează faptului că, spre deosebire de alte limbaje de programare, precum C/C++, care "au fost gândite sa utilizeze eficient echipament costisitor", Perl "a fost gândit sa utilizeze eficient programatori costisitori". Din păcate, aceste funcţionalitaţi, precum management automat al memoriei, expresii regulare, liste, sau hashuri, care uşureaza munca programatorului, vin la preţul unei utilizari mai mari de procesor şi memorie RAM.

Estimarea încărcarii HAului, şi generarea a diferite alte statistici este realizată de către simulator prin procesarea succesiva a unui set de date. Paşii acestei procesări sunt de obicei scripturi Perl, sau comenzi de sistem care invocă unul dintre flow-tools-urile menţionate mai sus. Pentru automatizarea procesului de prelucrare, au fost creeate scripturi care sa execute aceşti paşi într-o anumita ordine. Deoarece Bash este shellul standard pentru majoritatea distribuţiilor moderne de UNIX, şi este specializat tocmai în acest fel de operaţii, adică executarea de comenzi de sistem şi scripturi de diferite feluri, a fost preferat pentru această funcţie.

Pentru a evalua performanţa arhitecturii fHA şi, mai important, diferenţa de performanţa intre fHA şi HA stantard, am efectuat o simulare bazată pe înregistrări NetFlow. Mai exact, am colectat înregistrări de la două instituţii, anume Universitat Politecnica de Catalunya şi Universitatea din Coimbra. Ambele reţele furnizează conectivitate atât pentru prsonalul academic cât şi pentru studenţi, şi include servicii obişnuite precum Web sau DNS. Înregistrările NetFlow au fost capturate de catre routerele celor 2 departamente, şi se compun atât din trafic intern, cât şi din trafic spre exteriorul reţelelor.

Aceste înregistrări ne-au permis să simulăm performanţa unui fHA şi a unui HA ipotetic, folosite în cele două instituţii. Analiza a fost efectuată pornind de la ipoteza că toate nodurile din reţea sunt de fapt nodui mobile (MN), şi că acestea se afla în afara reţelei. Deci, plecăm de la presupunerea că profilul de trafic generat de un MN mobil este comparabil, daca nu chiar identic cu cel al uni nod fix. Aceasta e o presupunere rezonabila având în vedere că traficul generat este dependent de utilizatorul şi aplicaţiile instalate pe un calculator, iar acestea vor fi cam aceleaşi pentru ambele feluri de noduri. Mai mult, aceasta e o presupunere necesară, deoarece nu exista înca înregistrări de trafic mobil disponibile public. O observaţie importantă aici este faptul că serverele din ambele reţele au fost considerate puncte fixe, nu MNuri, deoarece serverele nu îşi vor schimba niciodata poziţia fizica sau punctul de ataşament la internet. De asemenea, profilul de trafic al unui server este fundamental diferit faţa de cel al unui MN.

Performanţa fHAului a fost evaluată considerând existenţa acestuia în reţele cu suport Mobile IPv4, Mobile IPv6, şi NEMO. Am prezentat de asemenea performanţa HAului standard, ca şi termen de comparaţie. Mai exact am evaluat beneficiile şi dezavantajele arhitecturii fHA. Beneficiile au fost analizate comparând încarcarea celor două arhitecturi concurente din punct de vedere a urmatoarelor marimi: octeţi pe secundă, pachete pe secundă, fluxuri active pe secundă, şi fluxuri noi pe secundă. Tehnologiile IPv4 şi NEMO vor fi tratate împreuna deoarece au profiluri de trafic practic identice pentru un număr mare de “nodurî.

Dupa o analiză amanunţită a traficului din punctul de vedere a unei reţele Mobile IP, am ajuns la concluzia că, din punctul de vedere a HAului, traficul poate fi împarţit în trei categorii diferite: Trafic care trece în întregime prin HA, trafic care nu trece niciodată prin HA, şi trafic care trece parţial prin HA (această ultimă categorie este cea a cărei rută poate fi optimizată).

Bazându-ne pe această observaţie, traficul este întai împarţit în aceste 3 categorii, folosind unelte specializate. Apoi, fiecare tip de flux este procesat în mod diferit. Această abordare a fost aleasă deoarece împarţirea fluxurilor putea fi facută cu unelte existente deja, anume flow-tools, care sunt optimizate pentru filtrare şi procesare de înregistrări NetFlow în format brut. Urmatorul pas în procesare este estimarea cantităţii de încărcare pe care fiecare dintre aceste tipuri de fluxuri o exercită asupra HAului standard şi a fHAului. Această etapă a fost efectuată folosind scripturi Perl create special pentru această sarcina. Prima categorie este adaugată pur şi simplu ca încărcare în totalitatea ei. A doua categorie e ignorată în întregime, fiind folosita doar pentru a genera statistici care sa ajute la înţelegerea profilului de trafic cu care avem de a face. Partea interesantă este categoria a treia. Deoarece standardul MIPv6 nu precizează exact cand procedura de RR (Return Routability) sa fie efectuată, dar sugerează ca o implementare de MIPv6 sa nu efectueze RR pentru fluxuri scurte de date, şi, deoarece nu exista înca implenetari de MIPv6 decât experimentale, se pot face doar speculaţii asupra impactului acestei categorii de trafic asupra HAului. Deoarece acesta este un subiect discutabil, a fost tratat cu o atenţie deosebita.

O altă categorie importantă de trafic, care nu ar trebui neglijata, este cea generată de semnalizări. Semnalizarea este evaluata ca şi încărcare a HAului atât timp cât doar trece prin el. Când semnalizarea este generată de către sau destinată HAului, atunci aceasta are un efect mult mai complex asupra încarcarii acestuia. Astfel, semnalele UPDATE şi WITHDRAWAL de internal Border Gateway Protocol (iBGP) trimitse catre routerele de margine (ER), nu sunt socotite aici. În aceeaşi situaţie intră şi semnalizarea de handover. Singura semnalizare care doar tranzitează HAul este cea de Remote Routability. Fiindca RR este specifica pentru MIPv6, acest tip de încarcare nu va fi prezenta în cazule MIPv4 şi NEMO. De exemplu, daca analizăm comparativ tipurile de trafic care încarca fHAul în cazurile MIPv6 şi MIPv4/NEMO, observăm că ar trebui să aiba acelaşi nivel de încarcare. Rezultatele experimentale, totuşi, arata o mica diferenţa. Acest lucru se datorează semnalizarilor de RR prezente doar în MIPv6.

Deoarece decizia de a efectua RR este luată la nivelul stratului 3, şi la acest nivel nu se deţin informaţii asupra cantităţii de date care va fi transmisă, deoarece aceasta se stabileste la nivel mai înalt, singura informaţie prezentă pe baza căreia se poate decide când sa se efectueze RR este cantitatea de date care au fost deja transmise prin ruta neoptimizată. Am considerat ca este rezonabil sa plecam de la premisa ca o implementare viitoare a standardului MIPv6 va efectua RR dupa un prag k de pachete a fost transmis şi primite prin ruta neoptimă. Valoarea lui k va fi cel mai probabil aleasă static şi va fi bazată pe multe astepcte. În această lucrare voi denumi un "algoritm de decizie RR optim" pe acela, care are k ales în aşa fel încat numarul total de pachete care trec prin HA per unitate de timp în medie, sa fie minim. Cu alte cuvinte, daca $f(k)$ este numarul mediu de pachete care trec prin HA per unitate de timp, cand RR este efectuat după k pachete au trecut prin ruta neoptimizată, atunci pragul k este considerat optim, pentru acea valoare pentru care $f(k)$ ia valoarea minimă. Rezultate experimentale bazate pe setul de date de la UPC au arătat că valoarea lui k pentru care algoritmul de decizie RR este optim, este de 1. Aceasta pare initial o surpriză,

având în vedere că lungimea fluxurilor de date are o distribuţie de tip “coadă lungă’, şi este evident ca nu e o idee bună sa facem RR, ceea ce implica 2 pachete suplimentare de încărcare, pentru un număr mare de fluxuri foarte scurte. Totuşi, rezultatele experimentale au arătat că penalizarea de a nu avea rută optimizată pentru cazul unor fluxuri lungi, este mai mare decât cea de a efectua RR pentru fluxuri scurte. În ciuda simplităţii sale, acest fel de algoritm permite simularea unui HA care este considerabil mai eficient din punctul de vedere al încărcarii sale, decât orice implementare viitoară. Această abordare a fost aleasă pentru a maximiza corectitudinea comparaţiei pentru HA, pentru ca este o optimizare exact în domeniul în care va fi inferior fHAului. Deci, din punct de vedere a fHAului, aceasta va fi cazul cel mai nefavorabil.

Estimarea încărcarii generate de traficul a cărei rută poate fi optimizată se bazează pe ideea că putem estima când RR va fi efectuat pentru un anume flux, cantitatea de date care va trece prin HA înainte ca RR sa fie efectuat, şi cantitatea de date care va trece prin HA în timpul efectuarii procedurii de RR (procedura RR duraza apoximativ 2 round trip timeuri (RTT) între MN şi CN). Acest lucru e calculat pe baza datelor pe care le deţinem despre acel flux, mai exact pe numărul de pachete şi timpul inceperii şi terminării fluxului, şi pe algoritmul de decizie RR prezentat mai sus. Momentul efectuării RR este calculat pe baza numărului mediu de pachete per unitate de timp, şi pragului de pachete din algoritmul de decizie RR. Numarul de pachete şi octeţi care trec prin HA înainte şi în timp ce se efectuează RR, este calculat pe baza numarului mediu de pachete şi octeţi per unitate de timp al fluxului, şi pe timpul care dureaza de la primul octet pană când procesul de RR s-a finalizat. Pentru a acoperi o gamă mai largă de cazuri posibile, s-au efectuat un număr de 3 simulări, fiecare având o alta valoare pentru durata RR (300ms, 500ms, şi 2000ms). Acest lucru a fost decis deoarece RTTul unei conexiuni poate varia mult în funcţie de calitatea conexiunii la internet, ora la care se face transferul de date, şi tehnologia folosita pt aceasta conecxiune (WIMAX, UMTS, 802.11, etc).

Mai exista incă o problemă în estimarea exactă a încarcarea HAului datorită traficului a cărei rută poate fi optimizată. Problema este moştenită din felul în care routerele Cisco generează înregistrări NetFlow. Când un router cu capacităţi NetFlow primeşte un pachet, modulul său NetFlow scanează antentul pachetului, şi pe baza adresei IP sursă, adresei IP destinaţie, port sursă, port destinaţie, tip protocol, şi a altor câmpuri, hotărăşte daca acest pachet face sau nu parte dintr-un flux pe care îl are deja în memoria tampon. Daca da, fluxul existent este updatat, dacă nu, un nou flux este creat. Intrările expirate din memoria de tampon sunt exportate periodic şi trimise prin UDP către o adresa IP preconfigurată. Pentru a evita supraîncărcarea routerului, înregistrările din memoria de tampon expira relativ rapid. Perioada de exportare e şi ea destul de scurtă. Ca şi rezultat, fluxurile sunt destul de scurte în lungimne, şi adesea, pachete care se deplasează pe exact aceeasi rută şi sunt parte din aceeaşi conexiune, sunt în fluxuri diferite. Ignorând aceasta problema, tot am avea pachete care folosesc aceeasi ruta în fluxuri diferite, pentru că fluxurile NetFlow sunt unidirecţionale, iar ruta e folosită pentru trafic în ambele direcţii. Aceasta problemă a fost rezolvată prin agregarea tuturor fluxurilor care vor parcurge o rută comună în un singur flux. agregat. Lungimea maximă a unui

astfel de flux a fost aleasă ca fiind 420 secunde, care e timpul maxim recomandat de validitate pentru RR în RFC-ul MIPv6. Din nou, acesta e un scenariu nefavorabil pentru fHA, deoarece el minimizează numărul de RR per unitate de timp, şi a fost ales pentru a maximiza corectitudinea comparaţiei faţa de HA. In realitate, acest timp va fi mult mai mic, marind sensibil încarcarea HAului datorita semnalizărilor şi a datelor care trec prin HA înainte şi în timp ce RR este efectuat.

Simulatorul este alcatuit dintr-un număr de scripturi Perl şi Bash, fiecare dintre ele cu propria funcţie. Acestea sunt împarţite în nivele de ierarhie. Nivelul 3 de ierarhie conţine doar un singur script, numit sugestiv *everything.ş*, şi singurul lui scop este de a lansa în execuţie cele 4 scripturi de nivel ierarhie 2 intr-un mod convenabil. Scripturile de nivel 1 au fiecare cate o funcţie simplă şi bine definită. Ele pot fi privite ca elemente într-un lanţ de procesări. Această înla;nţuire ia loc la nivelul 2 ierarhic, unde scripturile de acest nivel chiama scripturile de nivel 1 într-o anumită ordine pentru a obţine rezultatul dorit. Legat de limbajul de scriptare/programare folosit, scripturile de nivel 2 şi 3 sunt făcute în bash, iar cele de nivel 3 în Perl. De asemenea, flow-tools sunt considerate "scripturî de nivel 1. În cele ce urmează vor fi descrise în amanunt scripturile de nivel 2, funcţia şi structura lor, pe când pentru cele de nviel 3 doar funcţia lor va fi amintită.

Primul script de nivel 2 e folosit să calculeze valoarea instantanee a numarului de noduri per minut, pe timpul întregii durate a capturii de fluxuri NetFlow. Această variabilă este importantă pentru estimarea cantităţii de semnalizare generata de fHA cand primeste inregistrări din partea la MNuri care nu se afla în reţeaua de reşedinţă, deoarece pentru fiecare inregistrare, el trebuie sa genereze şi să trimită cate un mesaj de iBGP UPDATE la routerele de margine, pentru a instala noi rute. În unele cazuri vechea rută trebuie de asemenea stearsă cu un mesaj iBGP de tip WITHDRAWAL.

Al doilea script de nivel 2 este cel ce analizează capturile de înregistrări NetFlow din punct de vedere a stratului de transport şi de aplicaţie, şi generează statistici din acest pucnt de vedere. Aceeste statistici nu au importanţă din punct de vedere al încarcarii şi comparaţiei intreh HA şi fHA, dar sunt un bun ajutor în a înţelege mai bine rezultatele experimentale.

Al treilea script de nivel 2 calculează compoziţia procentuală a capturii de înregistrări NetFlow din punctul de vedere al celor 5 categorii de trafic definite şi detaliate mai devereme. Aceste statistici sunt generate în pachete, octeţi, şi număr de fluxuri. Şi aceste statistici au ca prim scop o prezentare amanunţită a capturii de înregistrări, dar, spre deosebire de setul anterior de statistici, acestea ne pot da un indiciu despre încarcarea pe care o sa o aiba HAul şi fHAul.

Al patrulea script de nivel 2 calculeaza încarcarea HAului şi al fHAului în urmatoarele unitaţi de masura: pachete/s, octeţi/s, fluxuri noi/s, şi fluxuri active/s. Acestea sunt calculate ca şi marime instantanee, pentru întreaga durată a capturii de înregistrări NetFlow. Acest script mai calculează de asemenea şi CDFul acestor 4 marimi, şi un tabel cu un sumar al încarcării. Acest pas este cel principal în lanţul de prelucrări şi şi cel mai complicat.

In cele ce urmeaza o sa prezint rezultatele experimentale, punând accentul pe diferenţa între fHA şi HA. În cazul MIPv4 şi NEMO, diferenţa de încarcare e semnificativă, în special în cazul la pachete pe secundă şi octeţi pe secunda. Acest lucru se datorează faptului că, în timp ce fHAul este tranzitat de doar 3.155% din pachete şi 2.93% din octeţi, 90.5% din octeţi şi 87.3% din pachete trec prin HAul standard. Este de remarcat faptul că, în cazul acestor două mărimi, mai mult de 90% din timp traficul prin fHA e mai mic decât valoarea minima instantanee a aceloraşi mărimi pentru HAul standard. În cazul încarcarii în fluxuri active pe secundă, diferenţa este şi mai mare. Încarcărea fHAului e mai mică decat cea a HAului cu 2 unitaţi de marime. Diferenţa în fluxuri noi pe secundă este şi ea notabilă, dar nu este atât de spectaculoasă ca în cazul celălaltor metrici. Diferenţa în proportie între fluxuri active pe secundă şi fluxuri noi pe secundă pentru fHA, care poate fi extrapolată pentru toate fluxurile externe de date, se datorează faptului că fluxurile interne sunt mult mai scurte decât cele către şi dinspre internet. Reţeaua internă are o rată de transfer mult mai mare decât conexiunea către internet, iar latenţa între noduri mult mai mică decât latenţa catre noduri din afara reţelei. Ambele aceste marimi afectează puternic pefromanţa unui transfer de date, mai ales în cazul TCP, care e responsabil pentru 95% din cantitatea de trafic în cazul de faţă. Din acest motiv, fluxurile de date din interiorul reţelei sunt mult mai rapide şi mai scurte decât cele spre şi dinspre internet.

În cazul MIPv6, situaţia se schimba dramatic. Algoritmul de decizie RR optim folosit în aceasta simulare asigură ca doar o fracţiune din traficul extern sa treacă prin HAul standard, astfel avantajul fHAului este doar marginal. Acest lucru este cel mai vizibil în cazul la pachete pe secundă şi octeţi pe secunda, unde cele 2 arhitecturi au aproape aceeasi valoare instantanee. În ciuda acestui fapt, diferenţa este înca vizibilă în metrica fluxuri active pe secundă, din cauză că fluxurile externe sunt mult mai lungi decă't cele interne. Din acest motiv, cantiteatea relativ mică de trafic adiţional care trece prin HA, fiind în întregime dinspre şi înspre internet, adaugă totuşi o canitate seminificativă de fluxuri active la care HAul trebuie să le păstreze starea. Este de asemenea important să întelegem că, desi dupa ce se efectuează RR, marea majoritate a datelor este transferată prin ruta optimă, toate fluxurile sunt iniţiate prin HA. Durata RR de asemenea nu afectează numarul de fluxuri noi per unitate de timp, asa că toate cele 3 cazuri de HA se suprapun perfect.

Chapter 2

State Of The Art

Mobile IPv6 [1], or, to a more general extent, Mobile IP, is considered one of the key technologies which will facilitate mobility for hosts connected to the Internet. With "mobility" a user can move and change his point of attachment to the internet while maintaining his IP address or even his network connections.

While theses technologies are quite promising, they are not yet mature and there is a lot of room for improvement. One of the problems is the fact that the communication between the Mobile Node (MN) and it's peers (Correspondent Nodes, CN) is routed through the Home Agent (HA), which is a special entity that maintains bindings between the two addresses of a MN, the Home Address, which identifies the MN's identity, and the Care-of Address, which identifies the MN's current location. Thus, the MN relies on the HA for its connectivity. Since one HA may be responsible for multiple MNs on a Home Link, the failure of a single HA may result in the loss of connectivity of many MNs. Thus, the HA represents the possibility of a single point of failure for a Mobile IP based network. Moreover, since all, (or in case of Mobile IPv6, some) of the communications of the MN is routed through the HA, may lead to the Home Link or even the HA itself becoming the bottleneck of the system.

To overcome some of these problems, more specifically to enhance reliability and to provide load balancing, the Mobile IP standard allows the deployment of multiple HA's on the Home Link. Upon failure of a HA, another HA can take over the functions of the failed one. Thus, continuous service is provided to the MNs registered with the failed HA. However, the transfer of service is somewhat problematic [3], since the solution is MN driven. The current specification of Mobile IPv6 does not provide solutions to these problems. The MNs are forced to detect the failure and select a new HA by themselves. This can cause delayed failure detection, which implies service interruption in the upper layer applications, increased workload on the MN, message overhead over the air interface, and the need for IPsec security associations reestablishments.

Many researchers have published papers in which they address these problems. For example, the solutions presented in [4] [5] [6] [7] [8] [12] increase HA reliability and load balancing by

deploying several redundant HAs at the Home Link. In these solutions, all the HAs share the registration state and they define efficient mechanisms for HA recovery. These solutions reduce the service disruption time in front of Mobile IPv6. In addition, the MNs traffic is balanced among the different HAs. The main difference among them is that some [4] [5] [6] are MN-driven solutions while others [7] [8] [12] are transparent to the MN. A problem with existing solutions, is that many of them focus on one technology only (Mobile IPv4, or Mobile IPv6), having limited or no applicability for the other. For example, [13] [14] [15] [16] [17] [18] are Mobile IPv4 specific, and could not be applied to solve the reliability problems of Mobile IPv6. This is because the main problem in Mobile IPv4 is single point of failure of the HA, which does not exist in Mobile IPv6 because of the multiple HAs provision on the Home Link.

Most of the above mentioned solutions require deploying redundant HAs on each sub-network. Although these solutions effectively mitigate the reliability problems mentioned above, they do not take into account the requirements of large networks with dozens of sub-networks. Deploying several HAs on each sub-network may be too expensive to deploy and to manage.

A different proposal, which does not require deploying redundant HAs on each Home Link, is the Virtual Mobility Control Domain protocol (VMCD) [9] [10]. The VMCD protocol allows multiple HAs to be placed at different domains. A MN may use multiple HAs simultaneously. The basic idea behind this proposal is that each HA advertises, through eBGP, the same home network prefix from multiple routing domains. Each MN then picks the best HA according to its topological position. The main drawback for this proposal is that the impact on the exterior BGP routing system scalability is unpredictable.

In [11], the authors present a novel HA architecture that only requires a set of HAs for the whole network, while providing reliability and load balancing like the other existing solutions. Another advantage of this architecture is the fact that, although designed initially for Mobile IPv6, it can be also implemented with virtually no modification for Mobile IPv4 [26], the only problem, the fact that IPv4 has no reserved address space for multicast, being only a conceptual one. The basic idea of this architecture is that the Mobile Nodes location can be announced to exit routers, this way effectively redirecting packets without involving the HA. Besides increasing reliability with a small number of HAs, and practically eliminating service interruption in case of HA failure, this approach also relieves the HA and the Home Link of a substantial amount of traffic. This paper aims to accurately describe the load of such a distributed HA, simply called fHA by the authors, and compare it to the load of a standard HA.

In [11], the authors try to validate their proposal by analytically evaluating it in comparison with other solutions to the same problem. Besides this analytical evaluation, they also present the results of a simulation. In an effort to provide realistic values, they have configured a highly mobile environment by using a Random Trip mobility model [21]. Specifically, they have used the Random Waypoint on Generalized Domain model with a set of 8 domains in their simulation. Despite these efforts, these results are limited in accuracy by the fact that a simulator is used that is

based on a mobility model. Another approximations is made regarding route optimized traffic (or, to be precise, traffic that can be route optimized) in Mobile IPv6, which is not accurately modeled, but simply not taken into consideration. Also, crude approximations and assumptions are made regarding the type and amount of traffic generated by the Mobile Nodes. This leaves a lot of room for improvement, and this is where this paper comes in. Since in this approach NetFlow Traces generated by the department routers of 2 institutions, which means real traffic, is used, all the above mentioned inaccuracies and approximations are eliminated. While this approach is by far not perfect itself, it is considerably more accurate than an analytical evaluation even and a simulation based on a model. To my knowledge, this is the method through which the highest accuracy can be achieved, without actually implementing and measuring the values in a real life application.

This evaluation method is not entirely new, being based on the one used in [23]. Ih that paper, the authors present the first steps towards characterizing the load of a Mobile IPv6 Home Agent. First, the internal traffic of a medium-size department from UPC (Universitat Politecnica de Catalunya) was analyzed. It has been assumed that all hosts were Mobile IPv6 nodes and that they were away from their Home Network. In this scenario the type and amount of traffic that had to be processed by a hypothetical Mobile IPv6's HA has been analyzed. Secondly existing models of traffic that applied to this particular scenario were reviewed. Specifically the focused was set on the models that characterized the load of Wireless LAN networks [24] [25], evaluating their goodness-of-fit for this particular case. In addition, it has been analyzed at which level of granularity these models could be applied. Finally, the authors have modeled the following variables characterizing the load of the HA: average flow sizes using a Pareto distribution and flow inter-arrival times using a Lognormal distribution. In contrast to [23], this paper does not aim to accurately describe or model the load of neither the standard IPv6 Home Agent, nor the "flexible Home Agent" proposed in [11]. The aim of this paper is to estimate as accurately as possible and compare the load of a hypothetical standard Home Agent to the load of a hypothetical flexible Home Agent deployed in the same network and environment. The aim is simply to compare the two architectures in terms of traffic volume at the Home Agent. For this purpose the method used by [23] to accurately estimate the traffic has been employed.

Chapter 3

Theoretical Fundamentals

3.1 Mobile IP

Mobile IP (or IP mobility) is an Internet Engineering Task Force (IETF) standard communications protocol that is designed to allow mobile devices to move from one layer 3 network to another while maintaining a permanent IP address, and without loosing it's active connections.

The Mobile IP protocols allow location-independent routing of IP datagrams on the Internet, providing an efficient, scalable mechanism for roaming. The scalability is achieved because node mobility is realized without the need to propagate host-specific routes throughout the Internet routing fabric. Each mobile node is identified by its home address disregarding its current location in the Internet. While away from its home network, a mobile node is associated with a care-of address which identifies its current location and its home address is associated with the local endpoint of a tunnel to its home agent. In other words, nodes may change their point-of-attachment to the Internet without changing their home IP address. This allows them to maintain transport and higher-layer connections while roaming.

Since performing of handovers and registering to a foreign network by the MN are beyond the scope of this study, they will not be presented here. The most important aspects for this paper are the ones concerning the routing of the flows from and to the MN to its peers when it is roaming. These will be analyzed in detail in this section.

3.1.1 Mobile IPv4/NEMO

In MIPv4 [2], when a MN registers to a foreign network, it's traffic is tunneled by the foreign agent to the home agent, from where it is routed to the destination. This happens for all of the MN communications, as you can see in the figure 3.1. This is similar to a VPN, where a host establishes a secure tunnel to the trusted home network and routes all it's traffic through that tunnel. This unoptimal routing path is one of the most important limitations of IPv4.

Network Mobility (NEMO) is an extension to the Mobile IP protocols, which is intended to solve the problem of the growing demand for mobility support for entire networks of IP devices. Although unoptimized routing is a major problem for this protocol, the NEMO Basic Support

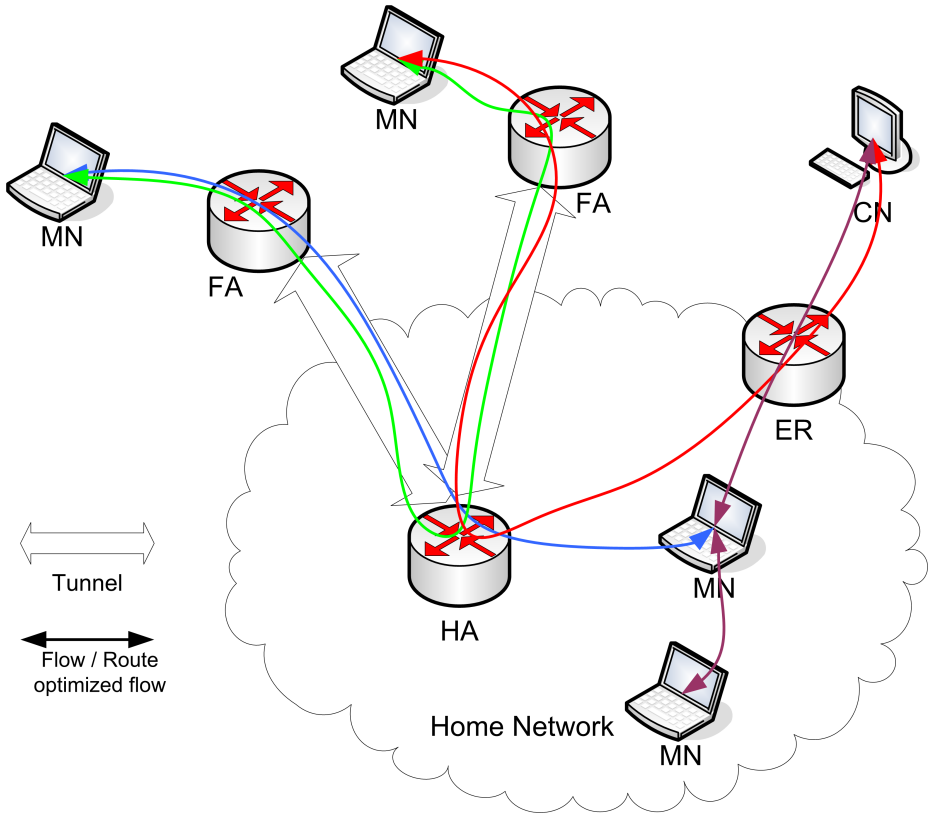


Figure 3.1: Routing in MIPv4

Protocol does not address this issue and the NEMO Working Group is not currently chartered to define a standard for Route Optimization [22]. Since NEMO uses the same unoptimized route as MIPv4, from the point of view of this study they have virtually the same behavior, so they will be treated in common.

3.1.2 Mobile IPv6

In MIPv6 [1], the MN does not need a foreign agent to tunnel its communications, but communicates directly with it's HA. The traffic, except for the absence of the FA, follows basically the same route. One of the main advantages of MIPv6 over MIPv4 is the fact that MIPv6 can perform Return Routability to optimize the route of it's flows, when it is communicating with hosts that are outside of the home network. Please refer to the figure 3.2 for a complete overview of the routes the different types of traffic follow, both for the case of an optimized and an unoptimized route. Please note that, although it is a possibility, the suboptimal triangle routing was not considered in this study for two main reasons: Firstly, in most modern networks the MN cant use the home address as source address for outgoing flows, since anti spoofing ACLs at the foreign network usually prevent this. Secondly, it is suboptimal routing, and the option of optimizing the route for both directions of flows is preferred.

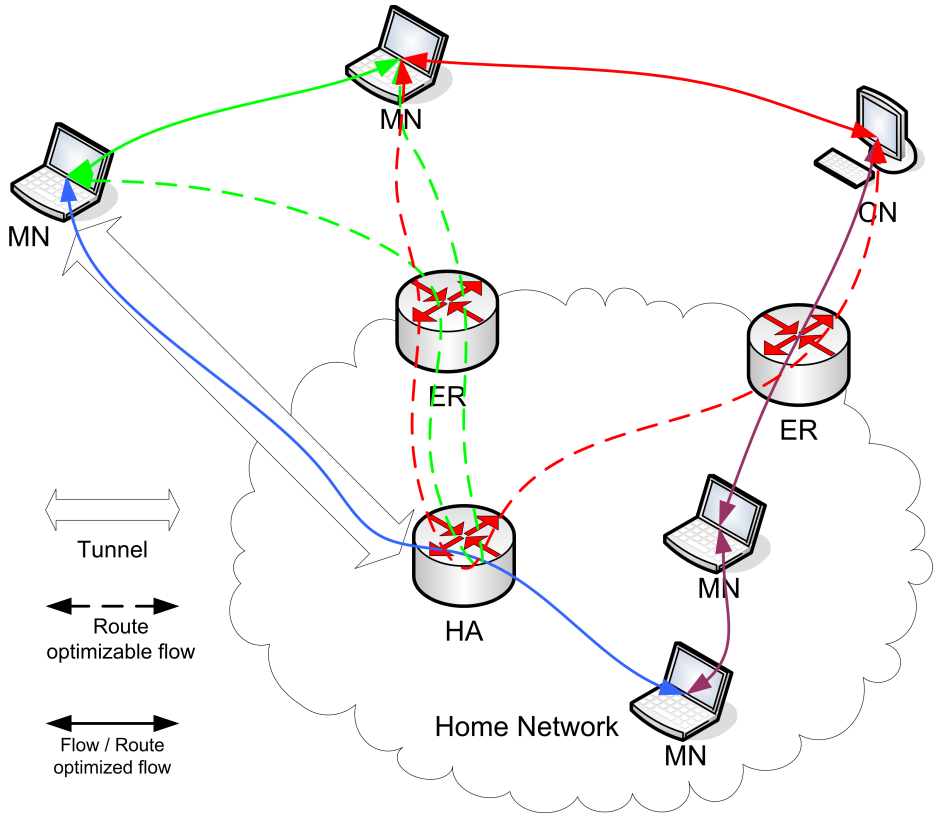


Figure 3.2: Routing in MIPv6

3.2 The flexible Home Agent Architecture

3.2.1 Design Rationale

In this section, the different operations of a Home Agent (HA) and the way they can be distributed from a network's point of view has been analyzed. In the rest of the paper, the following terminology will be used: the Home Network is defined as the set of Home Links managed by our HA. The Exit Routers (ER) are defined as the routers that connect the Home Network with the rest of the Internet. These ERs may or may not be the AS's border routers and an AS may have several Home Networks. Home Agents are responsible for maintaining bindings between the MN's identity and its location. The HAs forward the MN's signaling and the MN's data packets as well. MNs send data packets through their HA when communicating with their Home Network or with CNs. Since MNs can communicate directly with its CNs it is expected that communications through the HA are mainly used for short-term connections.

The Mobile IPv6 RFC states that packets sent through the HA may be secured through IPSec [27]. It should be taken into account that the MN can use IPSec with its peers regardless of the IPSec connection with their HA. We believe that it is not useful to secure MN to CN communications because the packets are only secured on half of the path (MN $\rightarrow$ HA) while the rest of the path (HA $\rightarrow$ CN) is not secured. Regarding the MN's communications with the Home Network,

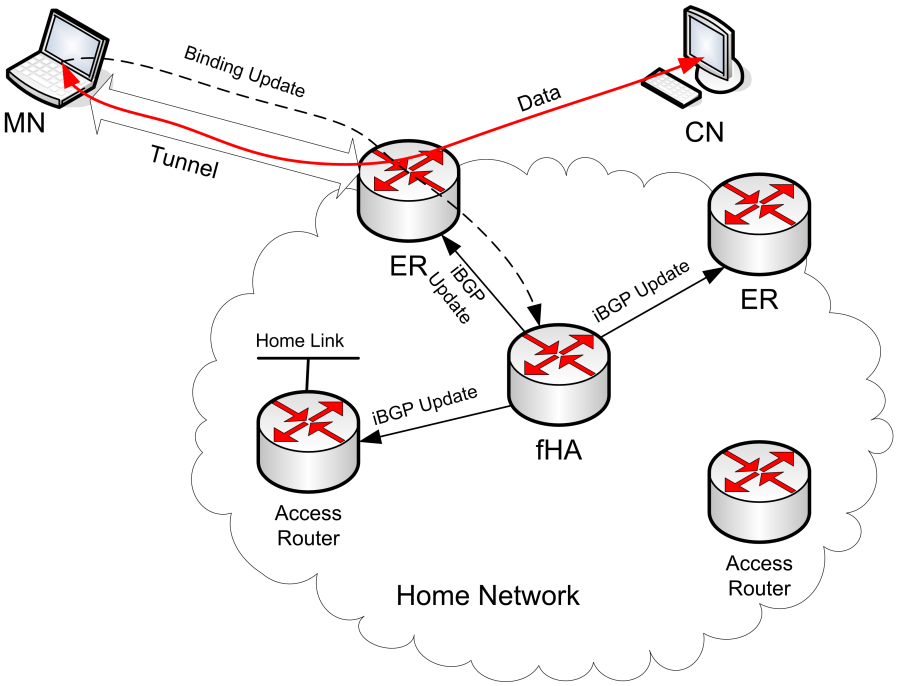


Figure 3.3: Overview of the fHA architecture

protecting the path is useful. In this case the HA is acting like a Virtual Private Network (VPN) gateway.

Under these assumptions and following the basic idea that a registration from a MN into a HA can be viewed as an internal route the HA's operations can be distributed throughout the network. In the proposed architecture, a single HA is required for the whole network; the authors of the [11] call it flexible Home Agent (fHA). This fHA will process (using IPSec) the MN's signaling messages and will maintain registration information. It will also distribute this information throughout the network as internal routes. The network will directly process the MN's communications with the CNs while the fHA will process the MN's communications with the Home Network (using IPSec) in the same way as a VPN gateway.

3.2.2 Overview

Figure 3.3 presents an overview of the fHA architecture. It has only one HA (called fHA) that will serve all of the MNs of the network. Take into account that the fHA architecture allows more than one fHA to be deployed to distribute the load. This fHA will be identified by an unicast address and the MNs will address its registration messages to it. Upon reception of a registration message, the fHA validates it and sends a routing message announcing the new route towards the MN. This information is then sent to each ER. In addition, the fHA advertises the route to the Home Link's Access Router (AR). At this point, the network knows the location of the MN.

When communicating with a CN through the HA, MNs do not address packets to the fHA but to an anycast [28] address owned by the ERs. In this way, a given ER receives the MN's data packets and de-capsulate, lookup and forward packets to the CN. Similarly, CNs send packets to the MN's Home Address. Upon reception, the ER lookups the packet's destination address (the MN's Home Address). Since the fHA has previously installed a route at the ERs they know that the MN it is not at home. Therefore, the ERs encapsulate and forward the packet to the MN's location. Our architecture manages efficiently MN to CN communications because some packets "bounce" at the ERs. This way the network's internal traffic is reduced considerably.

Regarding the communications from the MN to the Home Network, the MNs addresses its IPSec protected packets to the fHA that, in turn, de-capsulate and forward them to the MN's peer. The MN's peers address its data packets to the MN's Home Address. Since the fHA has announced to the Home Link's AR a route for the MN, the AR knows that the MN is away and it encapsulate the packet towards the fHA.

The Home Link's AR also multicasts Neighbor Advertisement messages on behalf the MN. This enables the AR to intercept communications from the Home Link to the MN and forwards them through the tunnel with the fHA.

In the following subsections the detailed operations of the fHA architecture are presented.

3.2.3 Dynamic fHA Address Discovery

This subsection specifies how the fHA announces their presence. In standard Mobile IPv6 HAs announce their presence through Router Advertisement messages. In this way, the MN's can automatically select a HA. Our architecture implements this functionality in exactly the same way that Mobility Anchor Points (MAP) announce their presence in the Hierarchical Mobile IPv6 (HMIPv6) [29] protocol. Our mechanism is also compatible with legacy MNs.

Each fHA sends Router Advertisement messages announcing its presence to the routers operating in the network. These messages include a preference value. In turn, the routers propagate the fHA's announcements to ARs that then forward them to the Home Link. Each router will decrement the preference value. This way MNs can automatically discover their fHA's address and select the best one according to the preference value.

This mechanism has many benefits. On the one hand, it enables ARs to automatically discover the fHA thus avoiding manual configuration. On the other hand, it allows us to deploy more than one fHA on the network and distribute the load among them. The fHA's Router Advertisement messages include the prefix(es) of the Home Network that it is serving and the anycast address owned by the ERs. Including the Home Network's prefix enables the MNs to know if its peers

are on the Home Network or not. Depending if the peer is on the Home Network or not MNs will address the data packets to the fHA or to the ERs.

Finally, in order to provide compatibility with legacy Mobile IPv6 nodes, MNs may send its traffic to the fHA.

3.2.4 Signal Processing

Each MN selects a given fHA through the above-mentioned mechanism. All the fHAs have pre-configured keys with the MNs as the Mobile IPv6 RFC states. Please note that ARs and ERs do not share any keys with the MNs. The fHAs receive registration messages from the MNs as stated by the Mobile IPv6 RFC.

Upon reception of a successful registration message, the fHA has to announce this information (route) to the ERs, to the Home Link's AR and to the rest of the fHAs. To distribute this type of information we use a routing protocol. Instead of designing a new routing protocol we use an already existing and deployed one. The routing protocol that best fulfills our requirements is the interior Border Gateway Protocol (iBGP) [30]. In the proposed architecture, the fHAs, the ERs and Home Link's ARs create an IBGP domain. It is very important to remark that this IBGP domain may be an already existing IBGP domain or a separate one. The routes announced through this IBGP domain always have the longest prefix (/128) and never affect regular BGP routes. It should be noted that the routes announced by the fHAs will never be distributed outside the network. Finally, the entities participating in the IBGP domain have pre-configured keys to provide confidentiality, integrity and authentication to the communications.

For each successful received registration message, the fHAs send an iBGP UPDATE message to the ERs and to the AR responsible of the MN's Home Link. The fHAs are able to determine the appropriate AR by inspecting the MN's Home Address.

We introduce new options in the iBGP UPDATE message. The UPDATE message sent to ERs includes the following information: <Home Address, Care-of Address, Lifetime>. Upon reception of this message, the ERs setup a tunnel endpoint with the MN. The tunnel source address is the anycast address while the destination address is the Care-of Address. In addition, each ER adds the following route to its routing table: HomeAddress
128 → Tunnel. The tunnel and the route are automatically deleted after "Lifetime" seconds.

The UPDATE message sent to the AR includes the following information: <Home Address, Lifetime>. Upon reception of this message, the AR knows that the MN is away from home (note that the AR does not know the location of the MN). Next, the AR setups a tunnel endpoint towards the fHA that announced the route and adds the following route to its routing table: HomeAddress

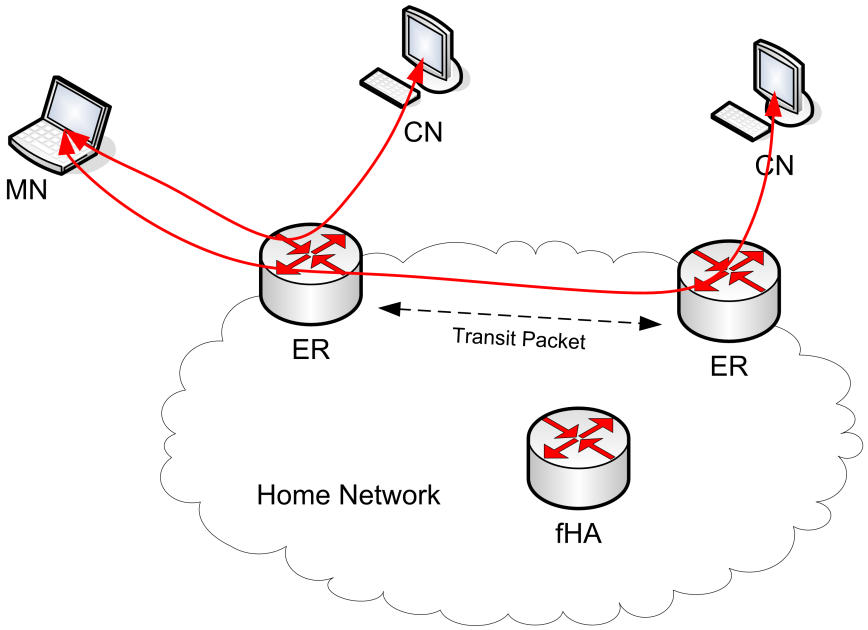


Figure 3.4: MNs to CNs communications

128 -¿ Tunnel. The AR also starts sending multicast Neighbor Advertisement messages on behalf of the MN at the Home Link. If a node of the Home Network (or Home Link) sends a packet to the MN, the AR intercepts it and encapsulates it towards the fHA. Once again, the tunnel and the route are automatically deleted after "Lifetime" seconds.

Once the MN returns home it sends a registration message to the fHA. Upon reception, the fHA sends an IBGP WITHDRAWAL message to the ERs and to the corresponding AR to immediately remove all the routes and tunnels related to the MN's Home Address.

3.2.5 Data Packet Processing

This subsection presents how packets are routed from/to the MNs. MNs communicating with CNs encapsulate their data packets to the anycast address owned by the ERs (figure 3.4). The packets are received by the "nearest" ER that will de-capsulate and forward them towards the packet's destination address (the CN's address). If the exit point of the CN's address is another ER then the packet traverses the network as a transit packet. It is important to remark that our solution does not require anycast routing. Packets addressed to the anycast address are routed normally (like unicast) and delivered to a given ER. We use anycast addresses because it is the standard procedure to assign the same address to different network interfaces.

MNs communicating with nodes located into their Home Network (figure 3.5) encapsulate their packets towards the fHA. However, packets sent by MN's peers are addressed to the MN's Home Address. The MN's AR intercepts those packets. Since the AR knows that the MN is away from home, it encapsulates the packet towards the fHA. Since the Mobile IPv6 RFC states that the

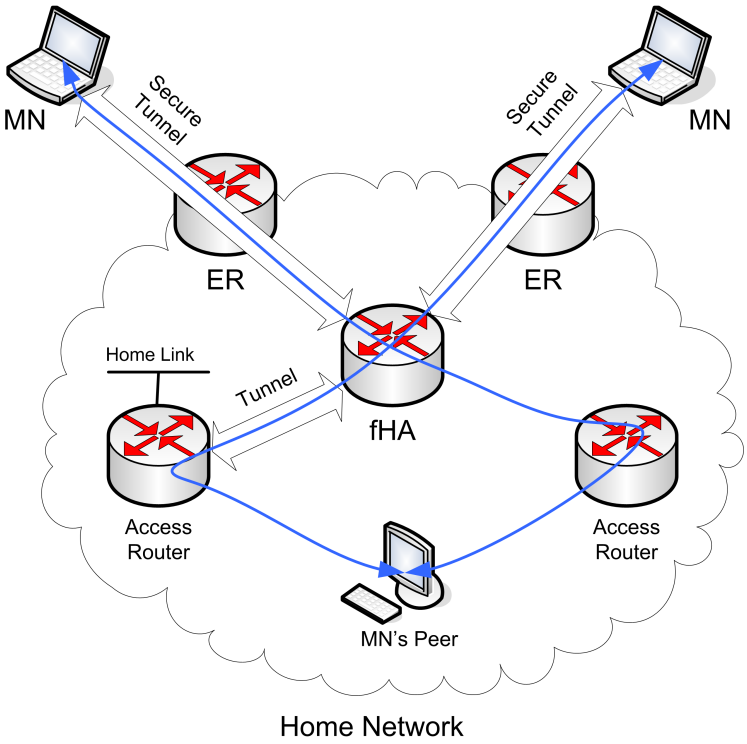


Figure 3.5: MNs to Home Network communications

packets are tunneled through the HA encapsulating the packet from the AR to the fHA does not affect the path's MTU [1]. As has already been mentioned, the MN's communications with the Home Networks are protected with IPSec.It is very important to remark that multihoming is only possible when BGP is used [31]. This means that a Home Network that is not running BGP will have just one exit router. Our proposal is flexible and works in both cases.

3.2.6 Flexible Home Agent Location

This subsection discusses the possible locations of the fHAs. Each fHA can be placed anywhere in the network, as a separate server, co-located with an ER/border router or even with a BGP Route Reflector.

One of the major benefits of the fHA is its flexibility. On the one hand, the fHA architecture can serve all the MNs of a network with one or more fHAs. If more than one fHA is deployed MNs will select the nearest one based on the preference value. This way the load is distributed among them. Each fHA thus only process signaling messages and communications from/to the Home Network (like a VPN gateway). MNs to CNs communications are then processed by ERs. On the other hand, our architecture is transparent to MNs running with legacy Home Agents and both technologies may co-exist on the same network.

Figure 3.6 shows an example of the flexibility of the fHA architecture. This AS has three networks and each one can independently select which approach it deploys. For instance, the "A"

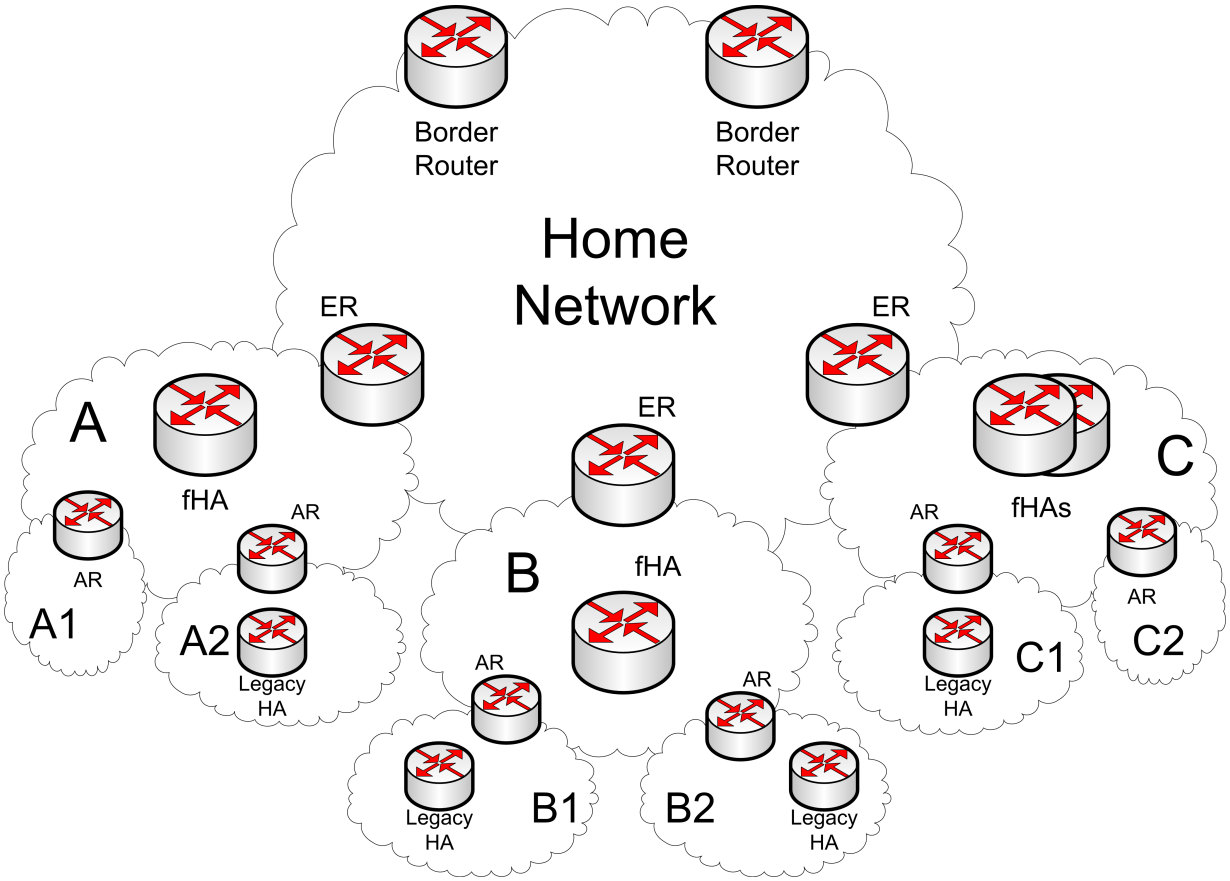


Figure 3.6: fHA location example

network can deploy both technologies. The fHA could serve MNs belonging to the "A.1" sub-network while MNs belonging to the "A.2" sub-network could be served by a legacy HA. The "B" network can deploy only legacy Home Agents on each sub-network. Finally, the "C" network can deploy two fHAs and all the MNs from "C.1" and "C.2" could be served by them.

There will be a separate IBGP domain for each Home Network. MNs served by an fHA send its data packets to an anycast address owned by the ERs. Since the prefix of the anycast address belongs to the Home Network's prefix, the AS's border routers knows how to forward the packets and do not need to be aware of the protocol, or be in the iBGP domain.

3.2.7 Overview of fHA routing

Since this study aims to analyze he load of a fHA and compare it to the one of a standard HA (both hypothetical) deployed in the same network it is important to have an overview of the routes that different types of flows take from and to a roaming MN. The figure 3.7 presents all possible routes, both suboptimal and route optimized, for all possible cases, and is valid both for MIPv4 and MIPv6. The only difference is that for IPv4, the routes will always be the ones called "route optimizable" in the figure, and never the route optimized ones, because MIPv4 has no support for route optimization.

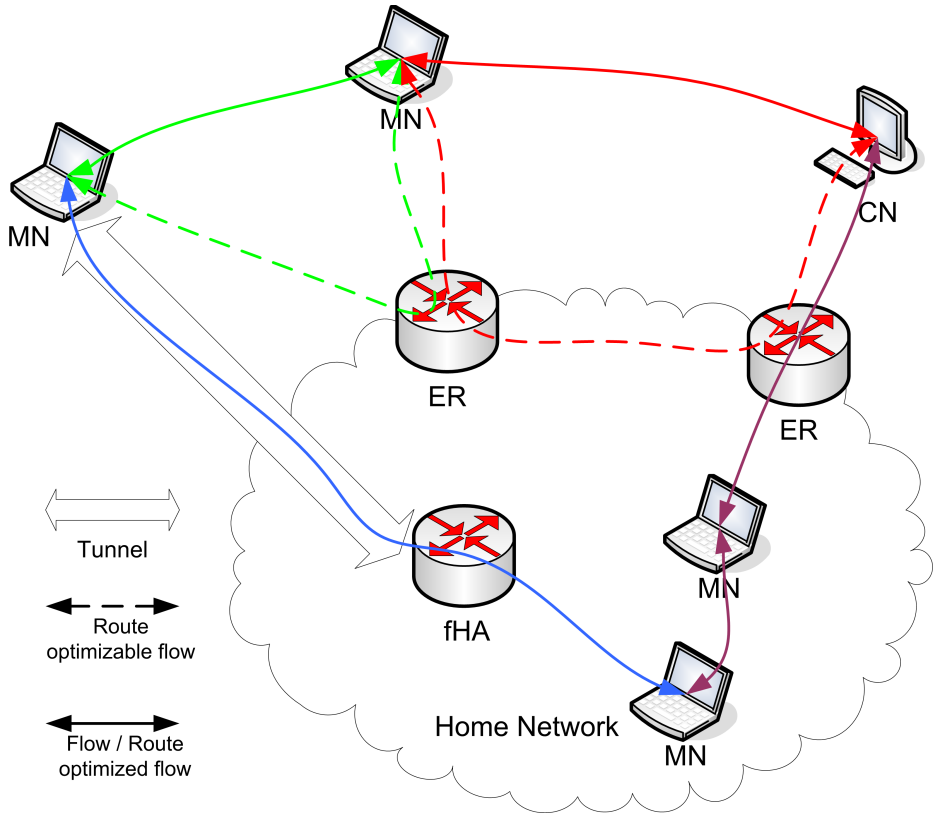


Figure 3.7: Routing in MIPv4 and MIPv6 when using fHA architecture

3.3 Technologies Used in Evaluation

3.3.1 Cisco NetFlow Trace Technology

NetFlow is an open but proprietary network protocol developed by Cisco Systems, one of the leading switching and telecommunications equipment producers, to run on Cisco IOS-enabled equipment for collecting IP traffic information [36]. Being open, it is also supported by platforms other than IOS, such as Juniper routers or FreeBSD and OpenBSD. The NetFlow technology was designed to provide a key set of services for IP applications, including network traffic accounting, usage-based network billing, network planning, security, Denial of Service monitoring capabilities, and network monitoring. Cisco routers that have the Netflow feature enabled generate netflow records; these are exported from the router in User Datagram Protocol (UDP) or Stream Control Transmission Protocol (SCTP) packets and collected using a netflow collector.

When a NetFlow enabled router receives a packet, its NetFlow module scans the header of the packet for source IP address, destination IP address, source port, destination port, protocol type, TOS (Type of Service) field, the login input (or output) port of the network equipment of the IP packets. Judging by this data, it updates the existing flow record that this packet is part of, or creates a new flow record in it's cache, if there are no records that mach the data. When these flows "expire", that is after a certain timeout, they are, like previously mentioned, exported through UDP

(or SCTP) to a preconfigured IP address. The data used for this study comprises of such NetFlow records. The data we have used is of the NetFlow V5 format, which is the most common format nowadays [37]. The most relevant (for this study) data fields that V5 format records consist of are: start time, end time, source IP address, source port, destination IP address, destination port, layer 4 protocol, number of packets, and number of octets.

3.3.2 Flow-Tools

Flow-tools is a library and a collection of programs used to collect, send, process, and generate reports from NetFlow data. The tools can be used together on a single server or distributed to multiple servers for large deployments. Flow data is collected and stored by default in host byte order, yet the files are portable across big and little endian architectures.

The following flow-tools have been used in this project:

- *flow-capture* is a tool used to collect, compress, store, and manage disk space for exported flows from a router [38]. It has been indirectly used in this project, meaning that, although has no application in processing the data towards obtaining scientific results, it has been used in gathering the trace records used as input data.

- *flow-cat* is a tool to concatenate flow files. Typically flow files will contain a small window of 5 or 15 minutes of exports. Flow-cat can be used to append files for generating reports that span longer time periods [38]. This tool has been used to preprocess the trace records and merge them into reports that span 1 day. This is a period of time that is an acceptable compromise between convenience of having a small number of files, and the hardware requirements necessary for large files.

- *flow-nfilter* is an utility used to filter flows based on user selectable criteria. Filters are defined in a configuration file and are composed of primitives and a definition. Definitions contain match lines grouped to form logical AND and OR operations on the flow using the selected primitives. A definition may contain the invert command which will invert the result of the evaluation [39]. This tool has been extensively used throughout the project. As explained further on, the flows can be divided into different categories (eg. MN to MN, MN to CN, etc), each of these categories impacting the HA in a different way. This is why separating the flows by filtering the was the first operation in processing them. All the filtering has been done using flow-nfilter and custom designed filters.

3.3.3 Perl Programming and Scripting Language

Perl is a high-level, general-purpose, interpreted, dynamic programming language originally developed by Larry Wall, a linguist working as a systems administrator for NASA in the late 1980s, as a general purpose Unix scripting language to make report processing easier [32]. Since

it's birth, when it's main design rationale was to create an efficient text manipulation tool, it has undergone many changes and revisions and become widely popular among programmers. Now it is used for a wide range of tasks including system administration, web development, network programming, games, and GUI development. Although it has far surpassed his original creation, Larry Wall still oversees development of the core language, and the newest version, Perl 6.

The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). Its major features include support for multiple programming paradigms (procedural, object-oriented, and functional styles), reference counting memory management (without a cycle-detecting garbage collector), built-in support for text processing, and a large collection of third-party modules. [33]

Perl borrows features from other programming languages including C, shell scripting (sh), AWK, and sed. For example, the overall structure of Perl derives broadly from C. Perl is procedural in nature, with variables, expressions, assignment statements, brace-delimited code blocks, control structures, and subroutines. Perl takes lists from Lisp, associative arrays (hashes) from AWK, and regular expressions from sed. These simplify and facilitate many parsing, text-handling, and data-management tasks. All variables are marked with leading sigils, which unambiguously identify the data type (for example, scalar, array, hash) of the variable in context. This is a feature inspired by Unix shell programming/scripting. Importantly, sigils allow variables to be interpolated directly into strings. Perl has many built-in functions that provide tools often used in shell programming (although many of these tools are implemented by programs external to the shell) such as sorting, and calling on system facilities. An important set of features that support complex data structures, first-class functions (that is, closures as values), and an object-oriented programming model, were introduced only in the 5th version of Perl. These include references, packages, class-based method dispatch, and lexically scoped variables, along with compiler directives (for example, the strict pragma). A major additional feature introduced with Perl 5 was the ability to package code as reusable modules. [34]

Perl was chosen for this project for 3 main reasons:

- It is very efficient in text manipulation. The NetFlow traces that had to be processed are available in 2 formats. A binary format, the "raw" format, which is especially difficult to process, and a humanly readable text format. The latter is by far the more user-friendly format, and due to Perl's powerful text processing tools, it is quite easily processable. Perl owes much of it's text processing power due o the specialized syntax for writing regular expressions (RE, or regexes) it includes, and the engine for matching strings to these regular expressions of its interpreter. The regular-expression engine uses a backtracking algorithm, extending its capabilities from simple pattern matching to string capture and substitution. The regular-expression engine is derived from regex written by Henry Spencer.

- Perl is capable to handle large data sets. This has been again a decisive factor in choosing the programming language, since, for example, one of the data sets used in this paper is a capture of 36'946'620 traces, each with 7 fields of different types of data.

- Perl has rapid application development and deployment cycle. This is because, In contrast, to many computer languages, such as C, which were designed to make "efficient use of expensive computer hardware". Perl is designed to "make efficient use of expensive computer programmers" [34]. These features, such as automatic memory management; dynamic typing; strings, lists, and hashes; regular expressions; and introspection; which are designed to ease the programmer's task, come at the expense of greater CPU and memory requirements.

3.3.4 Bash Scripting Language

Bash is a shell, or command language interpreter, that will appear in many GNU operating systems. It currently is the default shell on most systems built on top of the Linux kernel as well as on Mac OS X and it can run on most Unix-like operating systems. It has also been ported to Microsoft Windows. Bash is an sh-compatible shell that incorporates useful features from the Korn shell (ksh) and C shell (csh). It offers functional improvements over sh (its predecessor) for both programming and interactive use. In addition, most sh scripts can be run by Bash without modification. [35] Bash, like other UNIX shell programs, interprets user commands, which are either directly entered by the user, or which can be read from a file called the shell script or shell program. Shell scripts are interpreted, not compiled. The shell reads commands from the script line per line and searches for those commands on the system. Apart from passing commands to the kernel, the main task of a shell is providing a user environment.

Estimating the load of the home agent, and generating various other statistics, is done in this study by progressively processing a set of data. The steps of processing are usually Perl scripts or system commands that invoke the above mentioned net-flow tools. To automate the processing process, scripts had to be created that call these processing steps in a certain order. Since Bash is specialized in exactly this, calling system commands and starting executable scripts, it was favored for this task. Since the usage of Perl already implies a UNIX machine (Perl comes as a default component of most modern UNIX based operating systems), Bash was not considered a supplementary problem, since it is included by default in most UNIX OSes too.

Chapter 4

Design and Experimental Results

4.1 Design

4.1.1 Methodology of Evaluation

In order to evaluate the performance of the fHA architecture, and more important, the performance difference between the fHA and the standard HA, we have carried out a trace-driven simulation. In particular, we have collected NetFlow traces from different institutions, namely Universitat Politecnica de Catunya (UPC) and University of Coimbra (UoC). Both networks provide connectivity to both the academic staff and to the students, and include common services such as Web or DNS. The NetFlow traces have been captured by the department routers of the two institutions and comprise of host to internet traffic, and of host to host traffic as well.

These traces have allowed us to simulate the performance of a theoretical fHA deployed at these two institutions. The analysis has been carried out based on the assumption that all the end-hosts attached are, in fact, mobile nodes, and that these nodes are away and roaming. Hence, we are assuming that the traffic profile of a mobile node is at least comparable, if not identical, to that of a fixed node. This is a reasonable assumption since the type of traffic generated by a host is dependent of the user of the host and the applications running on it and both users and applications will be basically the same for a roaming MN as for a fixed MN. Further, this is a necessary assumption since, at the best of our knowledge, no mobile public data traces are available[1]. Please, note that the servers deployed at both networks were considered as fixed nodes, not mobile nodes, because servers will never change their point of attachment to the internet. Furthermore the traffic generated by a server is fundamentally different from the one generated by a end-host.

The performance of the fHA has been evaluated considering deployments in a Mobile IPv4, Mobile IPv6, and NEMO enabled network. We have also considered the performance of a standard HA for comparison. In particular, we have evaluated the benefits and the costs of the flexible Home Agent architecture. The benefits have been analyzed comparing the load of the standard HA with

[1] It is important to remark that recently a new workshop has appeared, that aims to fill this gap, and provide to the research community large-scale mobility datasets [40]

that of a fHA in terms of bandwidth, packets per second, active flows/s and flow arrival/s. Please note that from this point on, IPv4 and NEMO will be treated as one, since from the point of view of the load on the HA, they have virtually the same behavior, with the exception that a MN in IPv4 will have a slightly different traffic pattern than a NEMO "Mobile Node", but for large numbers of MNs, the traffic pattern difference disappears.

4.1.2 Design Approach

For the simulation, the aim was to create a simulator that can be used, (with as few as possible modifications) with any set of input data. The input data that the simulator is designed to handle consists of NetFlow traces. These netflow traces have to be preprocessed, or at least have been preprocessed in our case in terms of file granularity. A compromise between file size and file number, which means a compromise between hardware requirements and user-friendliness has been found. If the user possesses a sufficient amount of processing power, or a sufficient amount of patience, these preprocessing steps can be skipped. We have found that one file per day of traffic is decent in terms of both user convenience and hardware requirements. Regarding the export format of the NetFlow traces, the simulator supports only V5 version, which is only a minor drawback, since V5 is by far the most used export version nowadays.

After a detailed analysis of the traffic types involved in a Mobile IP enabled network, we have found that, from the point of view of the HA, this traffic can be divided into three different categories.

- Traffic that always passes through the HA.
- Traffic that never passes through the HA.
- Traffic that partially passes through the HA. (This is the traffic that is subject to route optimisation. This category of traffic does not exist in the case of the fHA, or in case of MIPv4/NEMO)

Based on this observation, the traffic is first split into these 3 categories, using specialized tools. Then, each type of flow is subjected to a different type of processing. This approach was preferred because separating the flows could be done with a readily available tool, namely flow-tools, which is optimized for filtering and processing NetFlow traces in raw format. The next step in processing is estimating which amount of load each of these categories of traffic inflicts upon the HA and the fHA. This part has been done using customly created Perl scripts. The first category, is simply added as load in its entirety. The second category was simply ignored when considering the load of the HA and the fHA, being mentioned only in statistics aimed to better describe and understand the traffic. The interesting part is the third category. Since the MIPv6 standard does not state when exactly the Return Routability (RR) procedure should be performed, but suggests that an implementation of MIPv6 should not perform RR for small flows of data, and since to the best of

our knowledge only experimental MIPv6 implementations exist, one can only make assumptions about how much of this kind of traffic will actually pass through the HA. Since this can be subject to debate, much effort has been invested into finding an acceptable solution to this problem.

As stated above, the IPv6 RFC [1] does not specify when exactly RR should be performed. The only statement regarding this matter gives us the main criterion that should be taken into consideration when taking this decision: the amount of data of a flow. Thus, in an attempt to be as fair as possible towards the standard HA, we have taken into consideration, when assessing its load, a RR decision algorithm which is ideal from the point of view of minimizing the amount of data that passes through the Home Agent. A detailed description about what we consider to be an optimal RR algorithm is presented below.

Since the decision to perform RR is taken at layer 3, and, at layer 3 the information about the amount of data that will be sent is not accessible, being decided at higher layers, the only available information for deciding when to perform RR is the amount of data that has already been transmitted and received through the unoptimised route. We have considered that it is reasonable to believe a future implementation of the HA will perform RR after a certain threshold k of packets sent and received through the HA. The value of k will most probably be static and chosen based on many aspects. We call "an optimal RR decision scheme" the one, where this threshold is chosen such that the overall number of packets that pass through the HA is the smallest. In other words, if $f(k)$ is the mean number per unit time of packets that pass through the HA when performing RR after a total of k packets have been sent and received through the HA, then the optimal threshold is considered the value of k for which $f(k)$ is at it's minimum. Experimental results have shown that, for our set of input data (the NetFlow traces from UPC and UoC), the value for the threshold k of the optimal RR decision scheme is 1. This might seem a bit of a surprise at first, because flow sizes follow a long tail distribution, and it is obvious that it is not efficient to perform RR, which implies sending 2 packets through the HA, for the vast amount of short flows. However, experimental results have proven that the penalty of not having an optimized route for longer flows is greater than the one of performing RR for short flows. Despite it's simplicity, this kind of decision scheme allows us to simulate a HA that is by far more efficient from the point of view of the load that will pass through it, than any future implementation. This approach has been chosen in order to maximize fairness towards the standard HA, because it is an optimization in exactly the field in which it is outperformed by the fHA architecture. From the point of view of the fHA, this is a worst case scenario comparison.

4.1.2.1 Categories of Traffic

As stated earlier, the first step in estimating the load of the different architectures of HA is dividing the traffic into different categories based on their contribution to the load of each HA type. For this purpose we have conducted the following analysis:

First, let us investigate the traffic in the MIPv4/NEMO technologies. From the point of view of the standard HA architecture we have:

- Traffic that never passes through the HA. This traffic is generated by MNs that are inside the Home Network, and communicate either with CNs, or with other MNs from inside the Home Network. In our case, in this category fall the inter server traffic and the Server to internet traffic. This type of traffic is marked with "cat.0" in the table 4.1
- Traffic that always passes through the HA. This traffic is generated by roaming MNs. If a MN from the outside of the Home Network communicates with any entity, be it a CN from the outside the Home Network, another roaming MN, or a MN from inside the Home Network (in our case a server), all it's traffic has to be tunneled through the HA.This type of traffic is marked with "***cat.1***" in the table 4.1

Table 4.1: Types of traffic, and their impact on the standard HA in IPv4/NEMO technology

	dst:Server	**dst:MN**	**dst:CN**
src:Server	cat.0	***cat.1***	cat.0
src:MN	***cat.1***	***cat.1***	***cat.1***
src:CN	cat.0	***cat.1***	n/a[1]

And from the point of view of the fHA we have:

- Traffic that never passes through the HA. This traffic is generated by MNs that are inside the Home Network, and communicate either with CNs, or with other MNs from inside the Home Network. This is the regular type of traffic, and is perfectly identical with that of fixed hosts inside a network with or without support for mobility. Additionally, some of the traffic of roaming MNs, namely traffic between roaming MNs and hosts from outside the Home network, which now "bounces" off at the Exit Routers of the network, now falls into this category. In our case, in this category we find the inter server traffic and the server to internet traffic, roaming MN to roaming MN traffic and roaming MN to CN traffic. This type of traffic is marked with "cat.0" in the table 4.2
- Traffic that always passes through the HA. This traffic is generated exclusively by roaming MNs. If a MN from the outside of the Home Network communicates with any host from inside the Home Network (in our case a server), all it's traffic has to be tunneled through the HA. This type of traffic is marked with "***cat.1***" in the table 4.2

Table 4.2: Types of traffic, and their impact on the fHA in IPv4/NEMO technology

	dst:Server	**dst:MN**	**dst:CN**
src:Server	cat.0	***cat.1***	cat.0
src:MN	***cat.1***	cat.0	cat.0
src:CN	cat.0	cat.0	n/a

Secondly, this is the detailed analysis of the traffic in a IPv6 network. From the point of view of the standard HA architecture we have:

- Traffic that never passes through the HA. This traffic is generated by MNs that are inside the Home Network, and communicate either with CNs, or with other MNs from inside the Home Network. In our case, in this category fall the inter server traffic and the server to internet traffic. This type of traffic is marked with "cat.0" in the table 4.3
- Traffic that always passes through the HA. This traffic is generated by roaming MNs. If a MN from the outside of the Home Network communicates with a MN from inside the Home Network (in our case a server), all this traffic has to be tunneled through the HA. This type of traffic is marked with "***cat.1***" in the table 4.3
- Traffic that partially passes through the HA. This is the traffic that is subject to Route Optimization, and is generated by roaming MNs that communicate with other hosts outside the Home Network, be it other roaming MNs or CNs. This type of traffic is marked with "*cat.2*" in the table 4.3

Table 4.3: Types of traffic, and their impact on the standard HA in IPv6 technology

	dst:Server	**dst:MN**	**dst:CN**
src:Server	cat.0	***cat.1***	cat.0
src:MN	***cat.1***	*cat.2*	*cat.2*
src:CN	cat.0	*cat.2*	n/a

And from the point of view of the fHA we have:

- Traffic that never passes through the HA. This traffic is generated by MNs that are inside the Home Network, and communicate either with CNs, or with other MNs from inside the Home Network. Additionally, the traffic that is subject to Route Optimization, namely traffic between roaming MNs and hosts from outside the Home network, which now "bounces" off at the Exit Routers of the network, now falls into this category. In our case, in this category we find the inter server traffic and the server to internet traffic, roaming MN to roaming MN traffic and roaming MN to CN traffic. This type of traffic is marked with "cat.0" in the table 4.4

- Traffic that always passes through the HA. This traffic is generated exclusively by roaming MNs. If a MN from the outside of the Home Network communicates with any host from inside the Home Network (in our case a server), all it's traffic has to be tunneled through the HA. This type of traffic is marked with "***cat.1***" in the table 4.4

Table 4.4: Types of traffic, and their impact on the fHA in IPv6/NEMO technology

	dst:Server	**dst:MN**	**dst:CN**
src:Server	cat.0	***cat.1***	cat.0
src:MN	***cat.1***	cat.0	cat.0
src:CN	cat.0	cat.0	n/a

Another important category of traffic, that should not be neglected, is the signaling. The signaling that is evaluated as traffic load to the different architectures of HA, is the one that only passes through it. The signaling generated and intended for the HA load it in a much more complex manner than simple packet forwarding, iBGP UPDATE and WITHDRAWAL messages which add and delete routes in the Exit Routers of the network are therefore ignored. The same will happen for handover signaling. Thus, here we discuss only the signaling generated by roaming MNs when performing Return Routability. Since RR is IPv6 specific, this type of traffic will affect only the HAs deployed in IPv6 networks. For example, when analyzing and comparing the types of load for the fHA in IPv4/NEMO and IPv6, one can notice that they should have the exact same load. The experimental results however show a small difference in load between the fHA IPv4 and the fHA IPv6. This difference is due to exactly the above mentioned signaling for RR.

It is also important to notice that in the tables 4.1, 4.2, 4.3, 4.4, the flow types MN $\rightarrow$ CN, CN $\rightarrow$ MN and MN $\rightarrow$ MN always fall into the same category. Although this is true from the point of view of the above classification criteria, MN to MN communications and MN cu CN communications, as seen later on, will be treated differently for the following reason: since there are 2 MNs involved in a MN to MN connection and only one in a CN to MN, the MN to MN connection generates twice as many RR per unit time, which implies twice as much signaling as the MN co CN connection. Secondly, since the Route Optimization expires twice as fast (more accurately: we have twice the amount of Route Optimization expiries per unit time), we will also have a larger amount of flows passing through the HA before and during the performing of RR. Thus, the MN to MN and MN to CN connections have to be processed separately, although they fall into the same category from the point of view of the above classification.

Please also note that all the above tables are symmetrical. This means, that a flow from host A to host B follows the same path as a flow from host B to host A, from the point of view of the HA.

4.1.2.2 Filter Structure

As mentioned above, for separating the flows in the different categories, one of the flow-tools was used, namely flow-nfilter. The flow-nfilter tool is used to filter flows based on user selectable criteria. Filters are defined in a configuration files, so they can then be easily used again. Unluckily, a custom set of filters had to be created for each set of input data. The filtering criteria are the IP addresses of the source and destination of the flow. Since the IP addresses of the servers, the mobile nodes, and of other types of hosts from the network will vary from network to network, It is impossible to create a general filter, valid for any set of input data. If a user wants to run the simulation for a different set of data, he will have to manually create a new set of custom filters.

A set of 6 filters was devised in order to be able to separate the flows in the categories presented in the previous section, more exactly 3 filters and their inverse filters:

- The *servers* filter. This filter outputs the flows that have one or both of the IP addresses (source and destination IP address) equal to one of the IP addresses defined as belonging to a fixed node. A fixed node is considered to be a server, a networking backbone equipment, or other kinds of hosts that will never leave the Home Network for one reason or the other. The filter is detailed in the table 4.5

Table 4.5: The *servers* filter.

	dst:Server	**dst:MN**	**dst:CN**
src:Server	allow	allow	allow
src:MN	allow	deny	deny
src:CN	allow	deny	deny

- The *nonservers* filter, the inverse of the *servers* filter, will output the subset of flows that have neither as source, nor as destination IP address a address belonging to a fixed node. The filter is detailed in the table 4.6

Table 4.6: The *nonservers* filter.

	dst:Server	**dst:MN**	**dst:CN**
src:Server	deny	deny	deny
src:MN	deny	allow	allow
src:CN	deny	allow	allow

- The *internal* filter. This filter will output exactly those flows, which have as source and destination IP address, IP addresses defined as belonging to the internal Home Network. This includes the above mentioned servers, as well as normal hosts, or MN, that will be able to (and in this study assumed to) roam. The filter is detailed in the table 4.7

Table 4.7: The *internal* filter.

	dst:Server	**dst:MN**	**dst:CN**
src:Server	allow	allow	deny
src:MN	allow	allow	deny
src:CN	deny	deny	deny

- The *noninternal* filter, the inverse of the *internal* filter, outputs the flows which are between an internal IP address, that means a fixed or mobile host of the Home Network (server or MN), and an outside host, or CN. The filter is detailed in the table 4.8

Table 4.8: The *noninternal* filter.

	dst:Server	**dst:MN**	**dst:CN**
src:Server	deny	deny	allow
src:MN	deny	deny	allow
src:CN	allow	allow	allow

- The *only_servers* filter, as it's name suggests, will output only the flows that have both as source and destination IP address, an IP address defined as belonging to fixed hosts. This filter is used only for statistics purposes, so it will not be detailed here.

- The *not_only_servers* filter, the inverse of the *only_servers* filter, will output the flows that has at least one of the IP addresses, the source or destination, belonging to a host that is not fixed, that means, to a MN or CN. The filter is detailed in the table 4.9

Table 4.9: The *not_only_servers* filter.

	dst:Server	**dst:MN**	**dst:CN**
src:Server	deny	allow	allow
src:MN	allow	allow	allow
src:CN	allow	allow	allow

The above described filters are then used to split the traffic into the three significant categories (which are related to, but are not the same with the categories mentioned in the previous section), that will be processed further, each in it's own customized way. These are:

1) MN → Server and Server → MN flows (from now on referred to as *type 112 traffic*, see figure 4.1)

2) MN → MN flows (from now on referred to as *type 21 traffic*, see figure 4.1)

3) MN → CN and CN → MN fows (from now on referred to as *type 22 traffic*, see figure 4.1)

Please note that these are the categories of traffic that are relevant for the load of the HA and fHA. The other two categories, namely:

4) CN → Server and Server → CN flows (from now on referred to as *type 12 traffic*, see figure 4.1)

5) Server → Server (from now on referred to as *type 111 traffic*, see figure 4.1)

which are not relevant to the load of the HA, and are used only to generate statistics, in order to better understand the makeup of the traffic. Please refer to the table 4.10 for understanding the link between these 5 types of traffic and the classification from the previous section.

Table 4.10: Relation between the 2 classifications of traffic

	dst:Server	**dst:MN**	**dst:CN**
src:Server	*type 111 traffic*	*type 112 traffic*	*type 12 traffic*
src:MN	*type 112 traffic*	*type 21 traffic*	*type 22 traffic*
src:CN	*type 12 traffic*	*type 22 traffic*	n/a

To efficiently filter the traffic, a chain of filters is applied for each category. For example, to obtain type 112 traffic, the following filter chain is applied:

- The *servers* filter. The obtained traffic is called type 1 traffic.
- The *internal* filter. This filter is applied to the type 1 traffic. The obtained traffic is called type 11 traffic.
- The *not_only_servers* filter. This filter is applied to the type 11 traffic. The obtained traffic is called 112 traffic.

Please refer to the figure 4.1 for a complete overview of the filter structures. Please note that the intermediate output of a filter chain (for example, in the case of the filter chain presented above, type 1 and 11 traffic), is reused, in order to increase the speed of the simulator.

4.1.2.3 Generating Flow Statistics

When run, the simulator, besides it's main task (to accurately estimate the load of the flexible and standard HAs), also computes various statistics about the traffic. These are computed in order to give the researchers analyzing the results of the simulation a better understanding of the makeup of the traffic.

Firstly the traffic is split into the 5 types of traffic mentioned above, and the percentage of packets, octets, and flows belonging to each category is computed. This, when understanding the classification from section **Categories of Traffic**, gives us a hint of the benefits of the fHA compared to the HA. We can already approximate the difference in load, knowing that the fHA

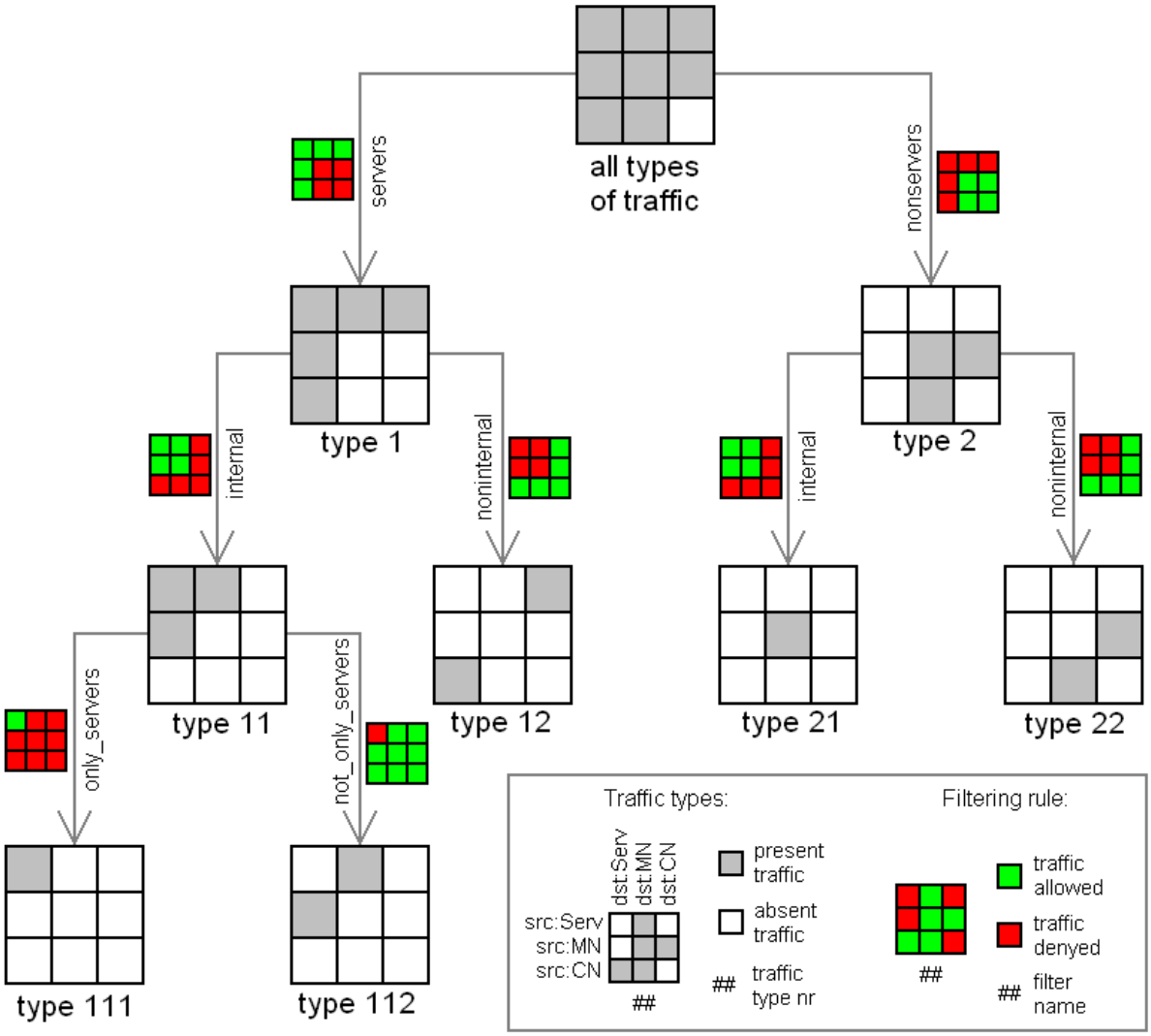


Figure 4.1: Complete filtering tree structure

processes only a subset of the internal traffic, compared to the standard HA, which has to additionally process a subset of the external traffic. The amount of external traffic the standard HA has to process strongly depends on the mobility protocol and on the route optimization policy in case of MIPv6.

Secondly the traffic is analyzed from the point of view of the transport and application layer. The analysis is conducted like in [41]. From the point of view of the transport layer the categories are UDP and TCP. ICMP is also considered in this classification, because no higher layer protocol can exist untop of it. From the point of view of the application layer, the classification in table 4.11 has been used, which is the same as in [41]. For application layer, the flows have been identified by their destination port, which is not the most reliable solution, but is considered a good compromise between low complexity of implementation and acceptable accuracy.

Table 4.11: Applications and protocols aggregated into categories

Category	Protocols
Bulk	ftp,https,tftp,rtip
Email	smtp,imap,pop,brutus,pop3s
Interactive	telnet,ssh
Name	Dns,netbios-ns
Net-manage	dhcp,ntp,epmap,snmp,timed
Web	http,https
Windows	netbios-ssn,netbios-dgm
Authentication	ident
Printing	Ipp
Streaming	hp-pdl-datastr

4.1.2.4 Evaluating the Load of Home Agent

The main goal of this study is to comparatively analyze the load of the standard HA and the flexible HA architecture when deployed in the same conditions. For this reason, great effort was invested in the accuracy of this part of the analysis. We modeled the load of both a hypothetical standard HA and a hypothetical fHA deployed in two networks of different sizes. We then present the data in two formats: The instantaneous value, averaged per minute, and a CDF of the following four variables:

- Packets per second. This metric gives us a hint on the processing power needed by the Home Agent. The HA, like almost any other network equipment has to process each header of a packet, and not it's contents. This is why, the required processing power is dependent on the number of headers per second (thus packets per second)
- Octets per second. This metric is also important in estimating the bandwidth required at the HA link to the network.

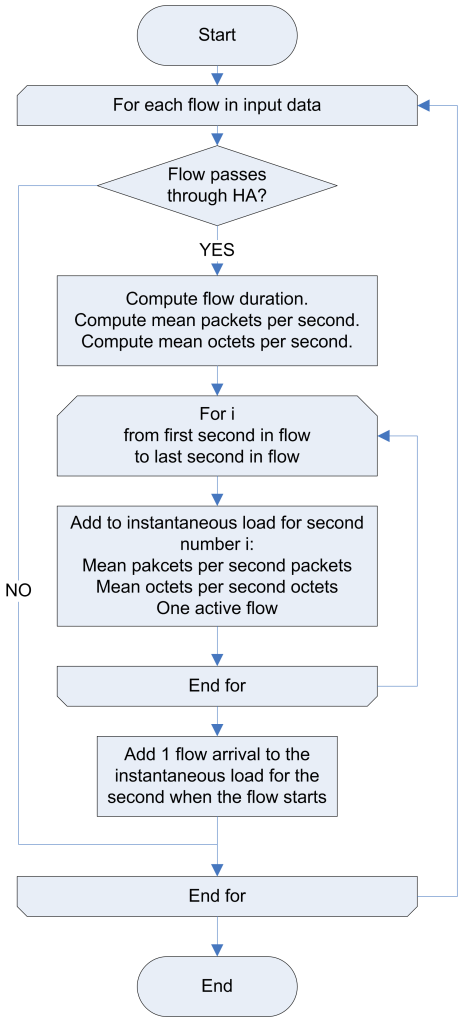


Figure 4.2: Flowchart: HA load computation algorithm for MIPv4/NEMO

- Flow arrivals per minute. This metric is also important because for each new flow, the HA has to do some security checks. These security checks are especially CPU intensive. Also, the HA has to store the state of the new connection.
- Active flows per second. This metric is important because the HA needs to keep state of an active connection.

In the case of MIPv4/NEMO, from the point of view of the standard HA we have only 2 categories of traffic. Traffic that passes through the HA in its entirety, and traffic that does not pass through the HA. Please refer to the section "*Categories of Traffic*" for a detailed analysis. In table 4.12 we present an overview. For accurately generating the instantaneous load, the algorithm presented in 4.2 has been employed. The basic idea of this algorithm is to compute the mean packets per second and octets per second for each flow, and add to the instantaneous load this value for each second of the duration of the flow. A value of 1 is also added to the instantaneous value of active flows for this period of time.

Table 4.12: Flows that pass hrough IPv4 HA and fHA

	MIPv4		MIPv6	
Type	through HA	through fHA	through HA	through fHA
Internal flows				
Server to Server (type 111)	no	no	no	no
Server to MN (type 112)	**yes**	**yes**	**yes**	**yes**
MN to MN (type 21)	**yes**	no	***partially***	no
External flows				
Server to INET (type 12)	no	no	no	no
MN to INET (type 22)	**yes**	no	***partially***	no

In the case of MIPv6, from the point of view of the standard HA, we have a third category of traffic, compared to MIPv4/NEMO. This is comprised of the flows that can be route optimized. Please refer to the section "*Categories of Traffic*" for a detailed analysis and to the table 4.12 for an overview. The basic idea of the algorithm is that based on the data we have about the flows, (more specific: number of packets, number of octets, start time and end time), and on the specific route optimization policy (detailed in section **Design Approach**), we can estimate when RR will be performed, the amount of data that will flow through the HA before RR is performed, the amount of data that will flow through the HA during the performing of RR (RR takes a little bit more than 2 round trip times between the MN and the CN). The moment of time when RR is performed is computed taking into account the mean packets per unit time of a flow, and the packet threshold. The amount of octets and packets that pass through the HA before and while RR is performed is computed based on the mean packets and octets per unit time of the flow, and on the time the flow passes through the HA (this is the time from the start of the flow up until RR has finished). The process is detailed in figure 4.3. Please note that three cases have been considered for the RR duration: 300ms, 500ms, and 2000ms. This is an attempt to cover the different technologies that ensure wireless connectivity, which being of a large variety (UMTS, WIMAX, 802.11 family) vary substantially in additional delay introduced.

The basic idea of aggregating the flows is similar to the algorithm which a Net-Flow enabled Cisco router generates flows. There are only two differences. The first is the fact that while a router aggregates packets into a flow, we aggregate flows into an "aggregated flow". The second difference consists in the way flows in the cache expire. In our case, an "aggregated flow" is considered expired, when it has reached the duration of the maximum RR validity, or when this amount of time has passed since the beginning of the flow. As stated above, all traffic between these two IP addresses would share the same path, so all traffic between two addresses is aggregated in the same flow, disregarding differences in direction, layer 4 protocol, source and destination port, etc. For a more accurate description of the aggregation mechanism, please refer to figure 4.4. A problem with this mechanism is the fact that it looses the details of exactly when different packets

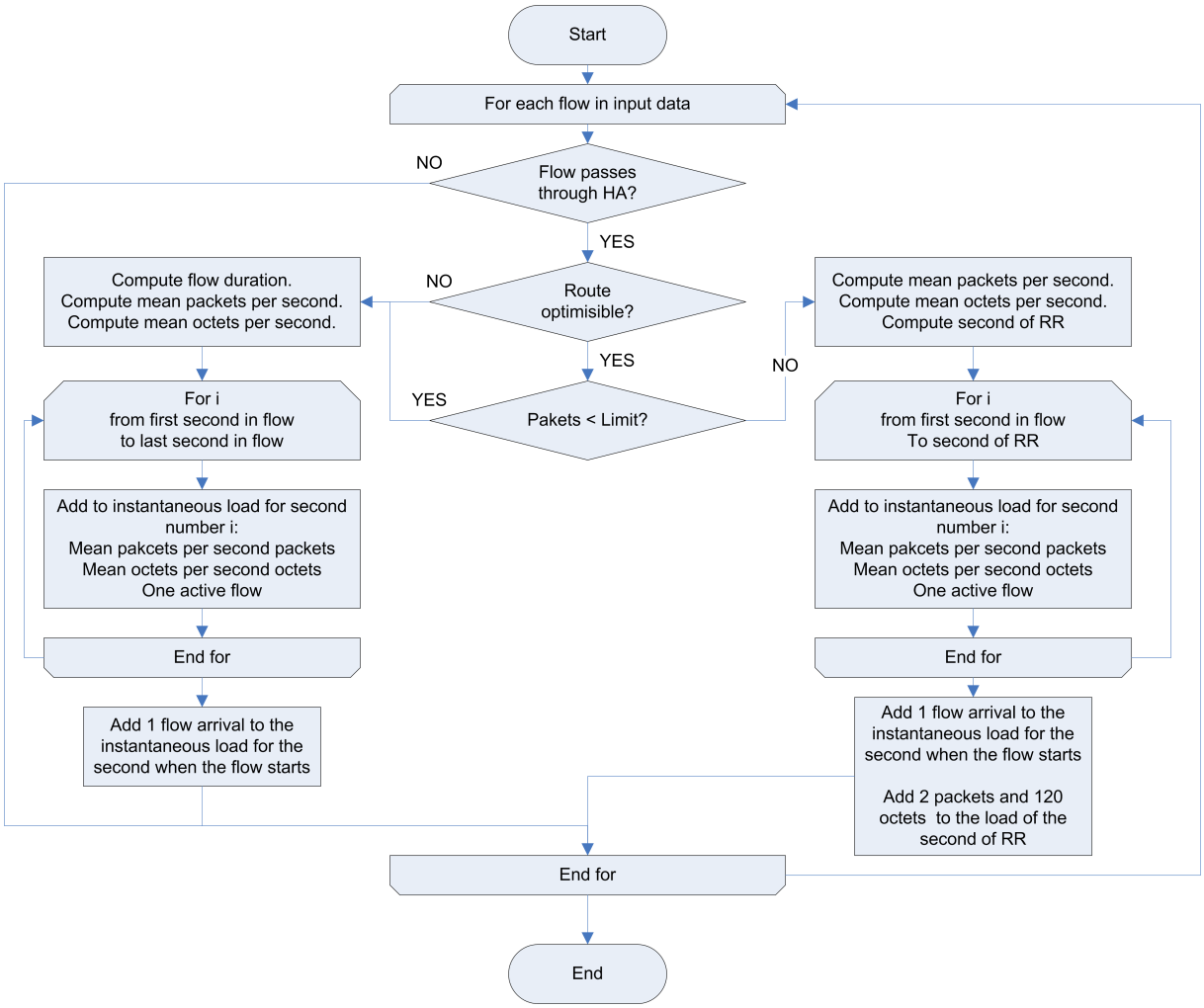


Figure 4.3: Flowchart: HA load computation algorithm for MIPv6

(or packet groups) pass through a hypothetical HA, maintaining only the global statistics for the entire aggregated flow. A later estimation of the amount of traffic that passes through the HA during the duration of RR looses it's accuracy. For this reason the amount of traffic that would pass through the HA is recorded, but only for the case of the RR decision scheme studied.

There is another issue with accurately assessing the load of the HA due to the route optimisable traffic. The problem is inherited form the way Cisco routers generate NetFlow traces. When a NetFlow enabled router receives a packet, its NetFlow module scans the source IP address, the destination IP address, the source port, the destination port, the protocol type, and other flags and fields of the header, judges whether it belongs to a flow that already exists. If so, update the flow record; otherwise, a new flow record is created in cache. The expired flow records in cache are exported periodically to a destination IP address using UDP. To avoid overloading the router, the flow records in the cache expire in a relatively short time. The period of exporting the expired cache records is also rather small. As a result, packets that are part of the same flow, and use the same optimized route, are now in two or more separate flows. But even if these times are configured to be large enough, we would still have more flows that would use the same optimized route, since flows are unidirectional, but traffic in both directions is routed through the optimized route once RR has been performed. This problem has been solved by aggregating all flows that will use the same (optimized) route in a single flow. The maximum length of such a flow has been chosen to be 420s, equal to the maximum recommended validity time for the RR in the MIPv6 RFC [1]. This, again, is a worst case scenario form the point of view of the fHA, since it produces the least RR signaling per unit time, and has been chosen to maximize fairness towards the standard HA architecture. In real application, the RR validity time will be substantially smaller, increasing the load on the HA to a certain degree.

4.1.2.5 Evaluating the Load of the "flexible Home Agent"

In the case of the flexible HA, for IPv4, the traffic can be split into the same 2 categories, like the standard Home Agent for IPv4 (see table 4.12). This is why the same basic algorithm, like the one presented in 4.2, has been employed. The only difference in processing is a different definition of which flows pass and do not pass through the HA, which has been done according to table 4.12.

Again, like in the case of the standard HA, for IPv6, the algorithm of computing the amount of data that passes through the fHA is more complex. Although from the point of view of the traffic the fHA has the same behavior as for IPv4 (see table 4.12), we also have a certain amount of signaling that passes through it. This signaling is generated by the MNs when performing RR. The MNs perform RR only for type 21 and 22 traffic, so, although this traffic does not pass through the fHA (see table 4.12), this being one of the main advantages of this architecture over the standard one, this traffic needs to be taken into consideration when assessing the load of the fHA. The algorithm, detailed in the figure 4.5, basically, estimates the moments when the roaming MNs will

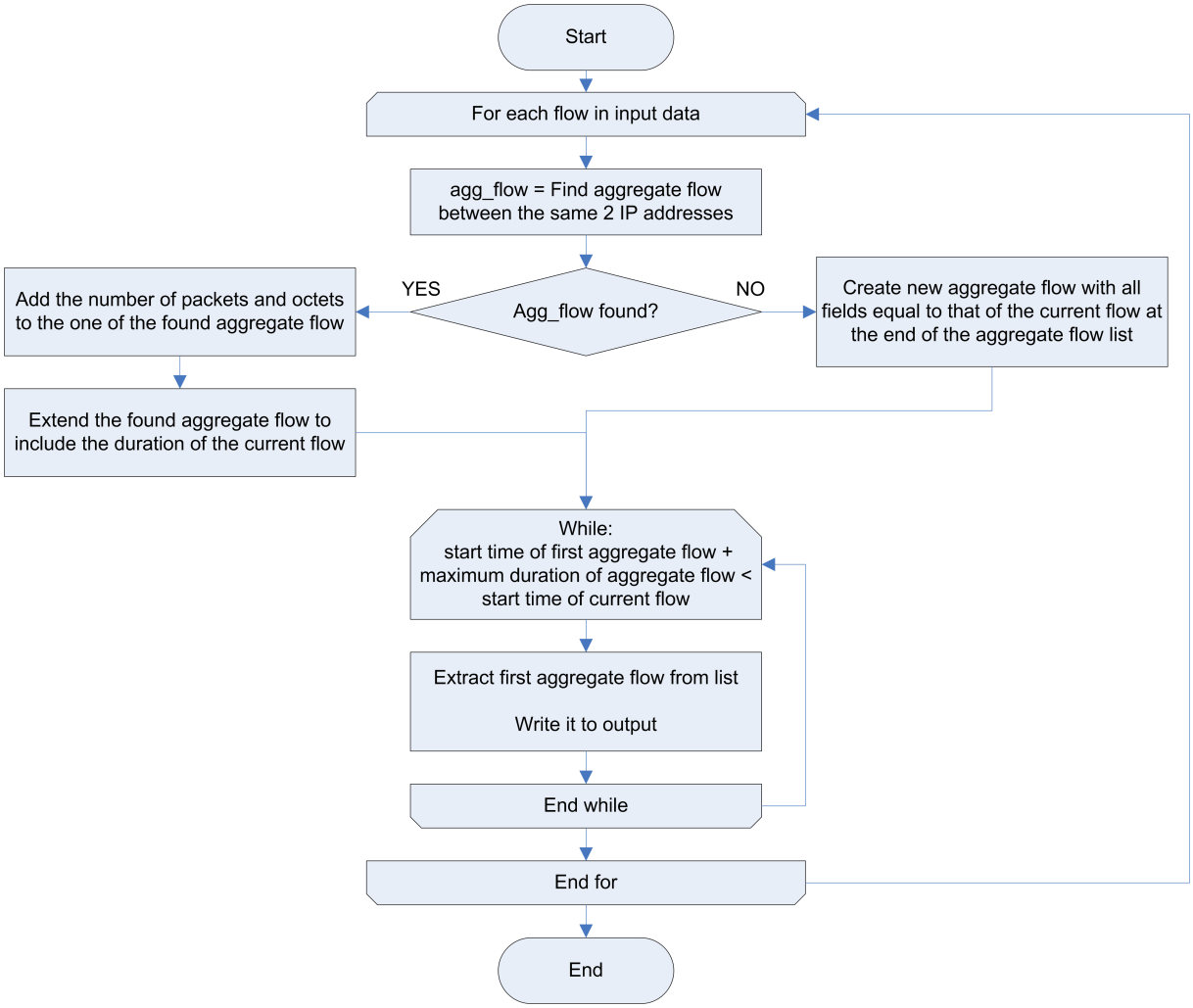


Figure 4.4: Flowchart: flow aggregation mechanism.

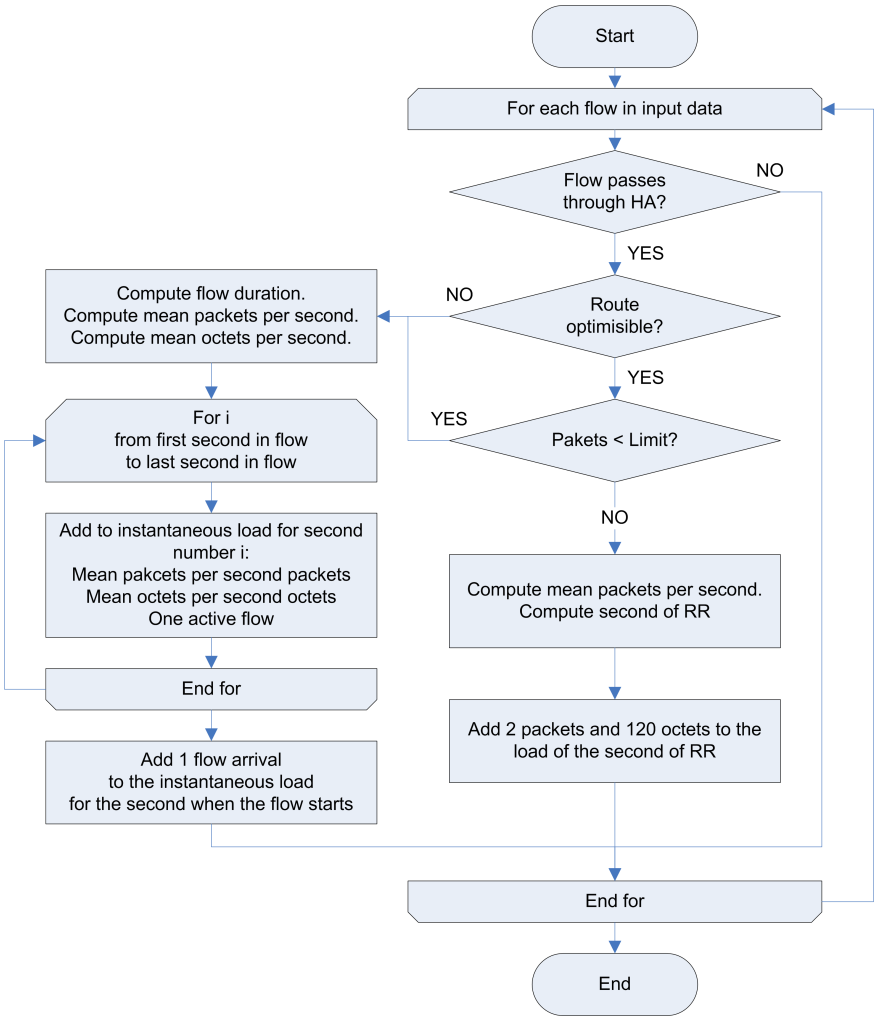


Figure 4.5: Flowchart: HA load computation algorithm for MIPv6/NEMO

perform RR, and adds the load of the 2 signaling packets of a RR procedure to the instantaneous load data, besides adding the load generated by the traffic that actually passes through the fHA.

The above presented algorithms generate the instantaneous load in packets/second, octets/second, flow arrivals/second and active flows/second. For enhancing the user friendliness of the output when plotted as graph, the data is postprocessedin the following manner: first it is averaged per minute, and then further smoothened, to eliminate the vast amount of spikes in the graphs. Based on these instantaneous data, the CDF (Cumulative distribution function) is then computed.

4.1.3 Structure of the simulator

The whole simulator consists of a number of Perl and Bash scripts, each with it's own function. These are divided into 3 levels of hierarchy. There is only one level 3 hierarchy script, suggestively called *everything.sh*, and its single function is to call the four level 2 hierarchy scripts in a convenient manner. Layer 1 scripts each have a simple and well defined function. They can be associated to elements in a chain of processing. The processing chain takes place in the level 2

scripts, which call many level 1 scripts in a certain order, to step-by-step process the input data (NetFlow records), in order to produce the desired results. Regarding scripting language involved, level 2 and 3 scripts are Bash scripts, while level 1 scripts are Perl scripts. Please note that, for a better overview of the structure of the simulator, flow-tools are also considered level 1 scripts. This section will detail the structure of the level 2 hierarchy scripts, while merely mentioning the functions of the level 1 hierarchy scripts. The rest of this section accurately describes the four layer 2 hierarchy scripts, their function, and their inner structure.

"active_nodes_computer.sh", the first level 2 script, is used to compute the instantaneous value of the number of active nodes per minute, during the whole duration of the flow. This measure is important for estimating the signaling generated by the fHA when MNs handover. For each handover, the fHA has to generate and send iBGP UPDATE signaling messages to the exit routers of the network to install the new route. In some cases, the old route has also to be deleted, with an iBGP WITHDRAWAL message. This level 2 script performs the following operations (in the here presented order):

- *Performed operation*: It filters the flows dividing them into two categories: outgoing flows (flows that have as source IP address an internal IP address of the Home Network), and incoming flows (flows that have as destination IP address an internal IP address of the Home Network).

  Input data: The flow files in raw format.

  Output data: outgoing flows and incoming flows in raw format.

  Level 3 script used: flow-nfilter with f_incoming and f_outgoing filter definitions.

- *Performed operation*: Transforming the flows into text. This is done to avoid the complicated format of raw NetFlow traces, and make efficient use of Perls text processing power. Please not that the text format used is f5, since all Perl scripts are designed to use this format as input.

  Input data: outgoing flows and incoming flows in raw format.

  Output data: outgoing flows and incoming flows in text format.

  Level 3 script used: flow-print

- *Performed operation*: Aggregating the flows. It is important to remark that altough a host does not send or receive data, it might be active. For this reason, we aggregate the flows. This way, we fill the "silence" periods from in between the rather short flows that Cisco routers generate.

  Input data: outgoing flows and incoming flows in text format.

  Output data: outgoing and incoming "aggregated flows" in text format.

  Level 3 script used: flow_aggregator.pl

- *Performed operation*: Computing the instantaneous value of the number of active nodes per minute. This is done using both incoming and outgoing flows at once (the above operations were performed independently for incoming / outgoing flows).

  Input data: outgoing and incoming "aggregated flows" in text format.

  Output data: active nodes per minute for the duration of each of the NetFlow record files (usually 24 hours)

  Level 3 script used: active_nodes.pl

- *Performed operation*: Adding the active nodes per minute data of each independent day to form a continuous set of data for the entire period of time of the NetFlow traces.

  Input data: active nodes per minute for the duration of each of the NetFlow record files (usually 24 hours)

  Output data: active nodes per minute for the duration the entire NetFlow record

  Level 3 script used: sumator.pl

- *Performed operation*: Postprocessing the data. (smoothening the graphic, for a more user-friendly appearance.)

  Input data: active nodes per minute for the duration the entire NetFlow record

  Output data: active nodes per minute for the duration the entire NetFlow record, smooth

  Level 3 script used: smoothener.pl

statistics_computer.sh, the second level 2 script, is the one that analyses the traces from the point of view of transport and application layers, and generates statistics from this point of view. These statistics are of no importance from the point of view of the comparison between the standard and flexible HA architectures, but, as stated earlier, are a good aid in better understanding the results of the comparison. This level 2 script performs the following operations (in the here presented order):

- *Performed operation*: Transforming the flows into text. This is done to avoid the complicated format of raw NetFlow traces, and make efficient use of Perls text processing power. Please not that the text format used is f5, since all Perl scripts are designed to use this format as input.

  Input data: flows in raw format.

  Output data: flows in text format.

  Level 3 script used: flow-print

- *Performed operation*: Aggregating the flows for application detection. Please note that this aggregation is fundamentally different from the one used in the other level 2 scripts. This aggregation mechanism is aimed to facilitate the identification of application layer protocols such as FTP, that use random destination ports for most of the traffic. Here is a short description of how it works: when a FTP connection is established, first a "signaling channel" with destination port 21 is established. Then, through their channel, a series of other data transfer channels are established, which have destination ports randomly chosen from a certain interval. For this reason most of the data will be impossible to identify by destination port. This aggregation mechanism, aggregates the entire traffic between the 2 hosts, FTP server and client, into a single "aggregated flow" with destination port 21. This way, all the data can be then identified using only a destination port based analysis. Please note that this Aggregation mechanism does not mix UDP and TCP traffic, contrary to the aggregation mechanism employed in the rest of the cases. Also, this step is not mandatory; in certain cases it has not been used because it actually damaged the application layer detection success rate.

  Input data: flows in text format.

  Output data: "aggregated flows" in text format.

  Level 3 script used: flow_aggregator_ad.pl

- *Performed operation*: Generating transport and application layer statistics. The percentage of packets, octets, and flows belonging to different transport and application layer protocols are computed. The application layer protocol is decided using the destination port of each flow.

  Input data: flows or "aggregated flows" in text format. (depends on whether aggregation has been made or not)

  Output data: transport and application layer trace statistics

  Level 3 script used: trace_statistics.pl

traffic_breakdown.sh, the third level 2 script, computes the make-up of the flow in terms of percentual composition of the 5 categories of traffic detailed in the previous sections. The statistics are generated from the point of view of packets, octets and flows. These statistics are of little importance from the point of view of the comparison between the standard and flexible HA architectures, since they only provide us with a hint of the difference in load between the two HAs but, as the transport and application layer analysis, are a good aid in better understanding the results of the comparison. This level 2 script performs the following operations (in the here presented order):

- *Performed operation*: Filtering the flows for obtaining the 5 independent flow types: 111, 112, 12, 21, 22. This step represents the filtering chain (or filtering tree) presented in the section "Filter Structure". (se figure 4.1)

  Input data: flows in raw format

Output data: 111, 112, 12, 21, 22 type flows in raw format

Level 3 script used: flow-nfilter with various filter definitions

- *Performed operation*: Transforming the flows into text. This is done to avoid the complicated format of raw NetFlow traces, and make efficient use of Perls text processing power. Please not that the text format used is f5, since all Perl scripts are designed to use this format as input.

  Input data: 111, 112, 12, 21, 22 type flows in raw format.

  Output data: 111, 112, 12, 21, 22 type flows in text format.

  Level 3 script used: flow-print

- *Performed operation*: Computing the percentual distribution of the traffic for the 5 types of flows. Please note that for technical reasons (the processing power requirement for processing large files) type 22 and 12 flows have been not concatenated and are processed 1 file per day. For this reason the output needs a little bit of postprocessing.

  Input data: 111, 112, 12, 21, 22 type flows in text format.

  Output data: Percent of each of the 5 types of flows (in terms of packets, octets and flow number) in composition of the input data set (NetFlow recprds)

  Level 3 script used: procentual2.pl

ha_load_computer.sh, the fourth level 2 script, computes the load of the HA and of the fHA in terms of packets/s, octets/s, flow arrivals/second, and active flows/second. These are computed as instantaneous load, for the whole duration of the NetFlow records used as input data. The script also generates the CDF of the 4 metrics, and a table summarizing the load. This step is the main, and by far the most elaborate one. This level 2 script performs the following operations (in the here presented order):

- *Performed operation*: Filtering the flows for obtaining the 5 independent flow types: 111, 112, 12, 21, 22. This step represents the filtering chain (or filtering tree) presented in the section “Filter Structure”. (see fig. 4.1)

  Input data: flows in raw format

  Output data: 111, 112, 12, 21, 22 type flows in raw format

  Level 3 script used: flow-nfilter with various filter definitions

- *Performed operation*: Transforming the flows into text. This is done to avoid the complicated format of raw NetFlow traces, and make efficient use of Perls text processing power. Please not that the text format used is f5, since all Perl scripts are designed to use this format as input.

  Input data: 111, 112, 12, 21, 22 type flows in raw format.

Output data: 111, 112, 12, 21, 22 type flows in text format.

Level 3 script used: flow-print

- *Performed operation*: The flows are aggregated. The aggregation mechanism is detailed in section **Evaluating the Load of Home Agent**, (see figure 4.4). Its main purpose is to aid in accurately describeing the load generated by RR signaling and by traffic that follows the unoptimised path before RR is performed. Here we use the aggregation mechanism that does not differentiate between UDP and TCP traffic, since we are concerned only about the data link layer.

  Input data: 111, 112, 12, 21, 22 type flows in text format.

  Output data: 111, 112, 12, 21, 22 type "aggregated flows" in text format.

  Level 3 script used: flow_aggregator.pl

- *Performed operation*: HA load estimation. The instantaneous Home Agent load is computed using the algorithms described in section **Evaluating the Load of Home Agent**, for each type flow that generates load for the standard HA separately. Please see figures 4.2, 4.3, 4.4, 4.5 for a detailed description of the used algorithms. The results are than added to obtain the combined load. This process is performed for each day individually. Also, for the case of the MIPv6 HA, three different results are generated: one for each of the following RR durations: 300ms, 500ms and 2000ms

  Input data: 112, 21, 22 type "aggregated flows" in text format.

  Output data: HA load due to 112, 21 and 22 type flows

  Level 3 script used: ha_load.pl with parameters (1, 300), (1, 500), (1, 2000) and (-1, 1); sumator.pl

- *Performed operation*: fHA load estimation. This step computes the load of the fHA due to type 21 and 22 flows, and also due to the signaling of performing RR due to the 112 flows. The results are than added to obtain the combined load. This process is performed for each day individually.

  Input data: 112, 21, 22 type "aggregated flows" in text format.

  Output data: fHA load due to 21 and 22 type flows and RR signaling

  Level 3 script used: ha_load.pl with parameters (-1, 1); fha_load.pl with parameter 1; sumator.pl

- *Performed operation*: Adding the load per day to obtain the instantaneous load for the entire duration of the NetFlow records.

  Input data: fHA load due to 21 and 22 type flows and RR signaling (one file per day)

  Output data: fHA load due to 21 and 22 type flows and RR signaling (continuous data)

  Level 3 script used: sumator.pl

- *Performed operation*: Postprocessing the data. The data contains the instantaneous load per second. This is now averaged per minute, for easier manipulation and plotting, and smoothened to eliminate useless spikes and improve user perception.

  Input data: fHA load due to 21 and 22 type flows and RR signaling (continuous data, time unit:second)

  Output data: fHA load due to 21 and 22 type flows and RR signaling (continuous data, time unit:minute, smoothened)

  Level 3 script used: averager.pl, smoothener.pl

- *Performed operation*: Generating CDFs. Based on the instantaneous value of the load, the CDF of the load in packets/second, octets/second, flow arrivals/second and active flows/second are computed for both HA and fHA and for both mobility protocols.

  Input data: HA and fHA instantaneous load (continuous data, time unit:second).

  Output data: CDFs for all the input data.

  Level 3 script used: cdf_pps.pl, cdf_ops.pl, cdf_faps.pl, cdf_afps.pl

- *Performed operation*: Computing the load summary. This script computes the minimum, mean, and maximum of the 4 measures for a quick overview of the load for (or load difference between) the standard HA and the flexible HA, for MIPv4 and MIPv6 as well.

  Input data: fHA load due to 21 and 22 type flows and RR signaling (continuous data, time unit:minute)

  Output data: tables containing the summary of the load

  Level 3 script used: trace_summary.pl

The table 4.13 summarizes the level 2 scripts, level 3 scripts they use, and output files they produce. The here generated results can be plotted (or further processed) with tools such as Gnuplot, or Matlab. The graphs in this paper have all been plotted using Gnuplot. The format of the output data is usually simple, consisting of float values organized in columns, for the instantaneous data, or of data formatted in tables with humanly readable labels.

4.1.4 Usage of the simulator

This paragraph describes how the simulator can be used for other sets of input data. The problem is, that since this simulator has been designed especially for these 2 sets of input data (the NetFlow traces from UPC and from UoC), some values and constants have been hardcoded in the scripts. For getting the scripts to work with other input data, one has to modify these values and constants to another value suited for the new set of data. For example many scripts have a constant named "firstday". This is a value for calibrating the 0 point of the Ox axis. It should be set to the number of the day (the day of the month) before the first day of captures. Another thing that

Table 4.13: Level 2 scripts, level 3 scripts they call, and output they generate.

Level 2 script	Used level 3 scripts	Output
active_flows_computer.sh	flow-nfilter flow-print flow_aggregator.pl active_nodes.pl sumator.pl smoothener.pl	anpm anpm_smooth
statistics_computer.sh	flow-print flow_aggregator.pl trace_statistics.pl	statistics
traffic_breakdown.sh	flow-nfilter flow-print procentual2.pl	traffic_makeup
ha_load_computer.sh	flow-nfilter flow-print flow_aggregator.pl ha_load.pl fha_load.pl sumator.pl averager.pl cdf_*.pl (ex: cdf_faps.pl) trace_summary	112_21_22_load_* (ex: 112_21_22_load_ha_ipv4_min) load_* (ex: load_ha_ipv4_cdf_faps) summary

should be changed are the filter definitions. These have been custom made for each set of flows and no automation process has been created. This, although violating the "good practice" rules, has been done because the difficulty of creating an automated filter generator based on a simple user configurable file. If there is interest for the simulator, this can and will be done (see *Future Work*). After solving these two issues, the scripts can be used as for the existing data, and should not present any other problems (unless the new data triggers a bug that was not triggered by the input data which the scripts have already tested/used with).

4.2 Experiments

4.2.1 Traces from UPC

4.2.1.1 Trace Statistics and Analysis

In this section an overlook of the dataset of the traces captured from the Universitad Politecnica de Catalunya is presented. The traffic is described similarly to [41], where the authors analyzed the load of an enterprise network. Specifically, we describe the main components of the traces at the transport and application layers, and how the traffic was divided in terms of Internal and External flows.

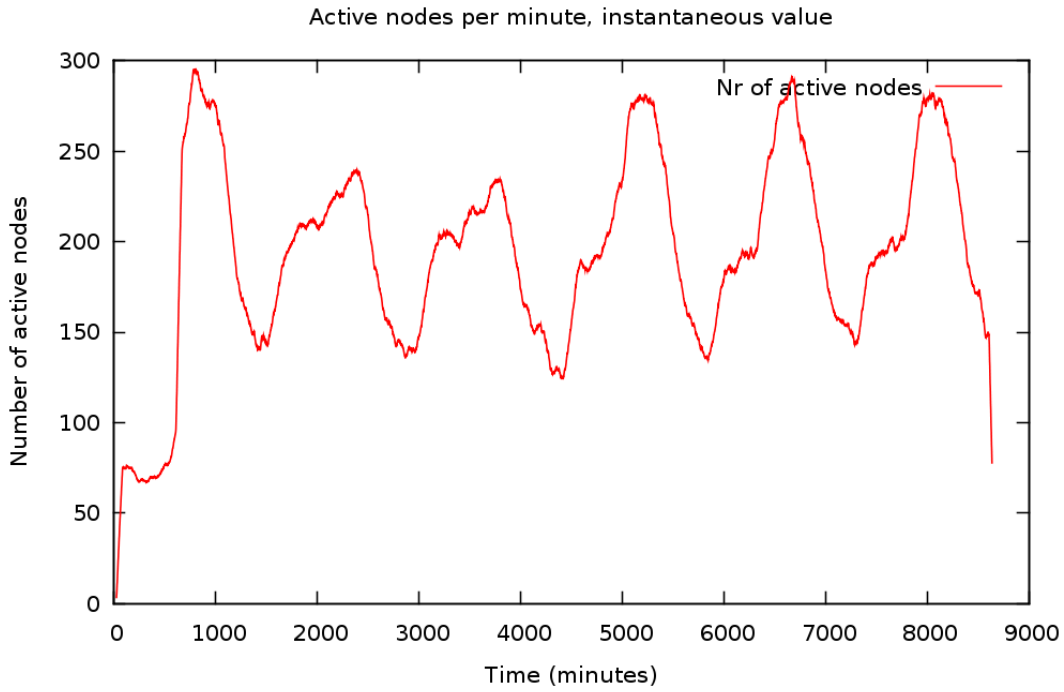


Figure 4.6: Active nodes per minute for the entire duration of the capture from UPC

The data analyzed comprises of NetFlow records from the department router of the Arhitectura de Computadores of UPC, for six days of traffic (from March 9 to March 14 2007). For the 144 hours of monitoring the traffic, a total of 970.356 MB, 1287 million packets, and 37 million flows were captured. The network includes a set of 7 sub-networks connected through the department router, the one that also provides Internet access. The networks includes servers as well as end-hosts (19 servers and an estimated value of 1500 end-hosts) and provides connectivity to a large number of users (an exact number is difficult to approximate. Please refer to figure 4.6 for understanding the magnitude of the network traffic.). Since the whole network is served only by the department router, the traces contain all internal and external ows. Unfortunately, with this setup ows transmitted within a subnet could not be captured. Nevertheless, this type of traffic is expected to be quite low since most of the servers are located in separated subnets.

Table 4.14 shows the break down of traffic by transport layer protocol (and data link layer Internet Control Message Protocol(ICMP)) in terms of packets, octets and flows. We can notice that the vast majority of traffic is carried by TCP flows, which is consistent with the findings of [41]. Furthermore we notice that UDP holds more than 50% of the flows, while a very small number of octets and packets, which indicates that UDP flows are very short. ICMP holds a small fraction of the flows and is and has also short flows compared to TCP. For application layer, the obtained statistics have not been shown here, since much of the traffic does not fall into any of the categories defined by [41]. This traffic would most probably be of “peer to peer” type generated by applications such Skype, IM software; or non-standard port services, chosen for security reasons.

Table 4.14: Fraction of Flows, Octets and Packets for transport layer protocols and ICMP from UPC traces

Protocol	Octets	Packets	Flows
UDP	4.700	1.785	50.235
TCP	94.953	98.130	46.094
ICMP	0.266	0.025	3.539
others	0.081	0.060	0.132

Now lets focus on table 4.15 which plots the ratio of internal vs. external ows. As the table shows roughly 95% of the the ows, octets and packets belong to traffic to and from the Internet, and only 5% represents traffic between hosts inside the institutions. The table provides us a hint about the advantage of the fHA over a standard HA. This is because the fHA only process internal ows, while the standard HA has to forward a subset of the external one. This set depends on the mobility protocol and the route optimization policy.

Table 4.15: Fraction of Flows, Octets and Packets for different types of flows from UPC traces

Type	Flows	Octets	Packets
Internal flows			
Server to Server (type 111)	2.3254%	0.3912%	1.3042%
Server to MN (type 112)	2.0210%	3.1275%	2.7008%
MN to MN (type 21)	0.4545%	0.1278%	0.2300%
External flows			
Server to INET (type 12)	34.2202%	9.7616%	11.3927%
MN to INET (type 22)	60.9787%	86.592%	84.3724%

An interesting observation we can make about the traffic at the UPC, is that while the internal traffic follows the day/night and business day/weekend traffic pattern one would expect, having peaks during the middle of business days, and being close to 0 at night, the external traffic does not. We have a great amount of traffic to and from the internet during the weekend, that actually surpasses the one from during business days. (This can be seen in the figures 4.7 and 4.8. Although they do not plot the internal versus external traffic, in case of MIPv4/NEMO the load of the fHA is an acceptable approximation of the internal traffic, while the load of the standard HA is a good approximation of the total traffic.) Since this anomaly is present only in the amount of octets per second and packets per second, the rest of the metrics showing a clean day/night and businessday/weekend pattern (see figures 4.10 4.9), we can assume that it is due to large file transfers that have been active during the weekend. These are the types of traffic that comprise of large quantities of data (packets and octets), while being a low number of flows.

4.2.1.2 Home Agent Load, MIPv4/NEMO

This section presents the experimental results for the mobility protocols MIPv4 and NEMO, emphasizing the difference in load between the fHA and the HA. In this case, the difference in load is significant, especially when regarding packets per second (fig. 4.7) or octets per second(fig. 4.8). This is because, as can be seen in the table 4.15, the standard HA has to process 90.5% of traffic when regarding octets, and 87.3% of traffic when regarding packets, while the fHA only 3.155% of octets and 2.93% of packets. It is interesting to remark, that in case of the load in packets per second and octets per second, more than 90% of the traffic through the fHA is lower in instantaneous value than the minimum load of the standard HA. In case of the load in active flows per second (fig. 4.10), the difference is even greater: the load of the fHA is smaller than the one of the standard HA by a factor of roughly 100. The difference in flow arrivals per second (fig. 4.9) is also noticeable, but not as spectacular as in the case of the other three metrics. The difference in proportion between active flows per second and flow arrivals per second for the fHA, which can be extrapolated to the internal flows, is due to the fact that the internal flows are much shorter than flows to and from the internet. The internal network has a much higher bandwidth than the link to the internet, and a much lower latency between hosts than the latency to hosts outside the network. Both these variables strongly affect the performance of data transfer, especially in the case of TCP which is responsible for more than 95% of the data transferred. For this reason flows inside the network are much faster and much shorter than the ones to and from the internet.

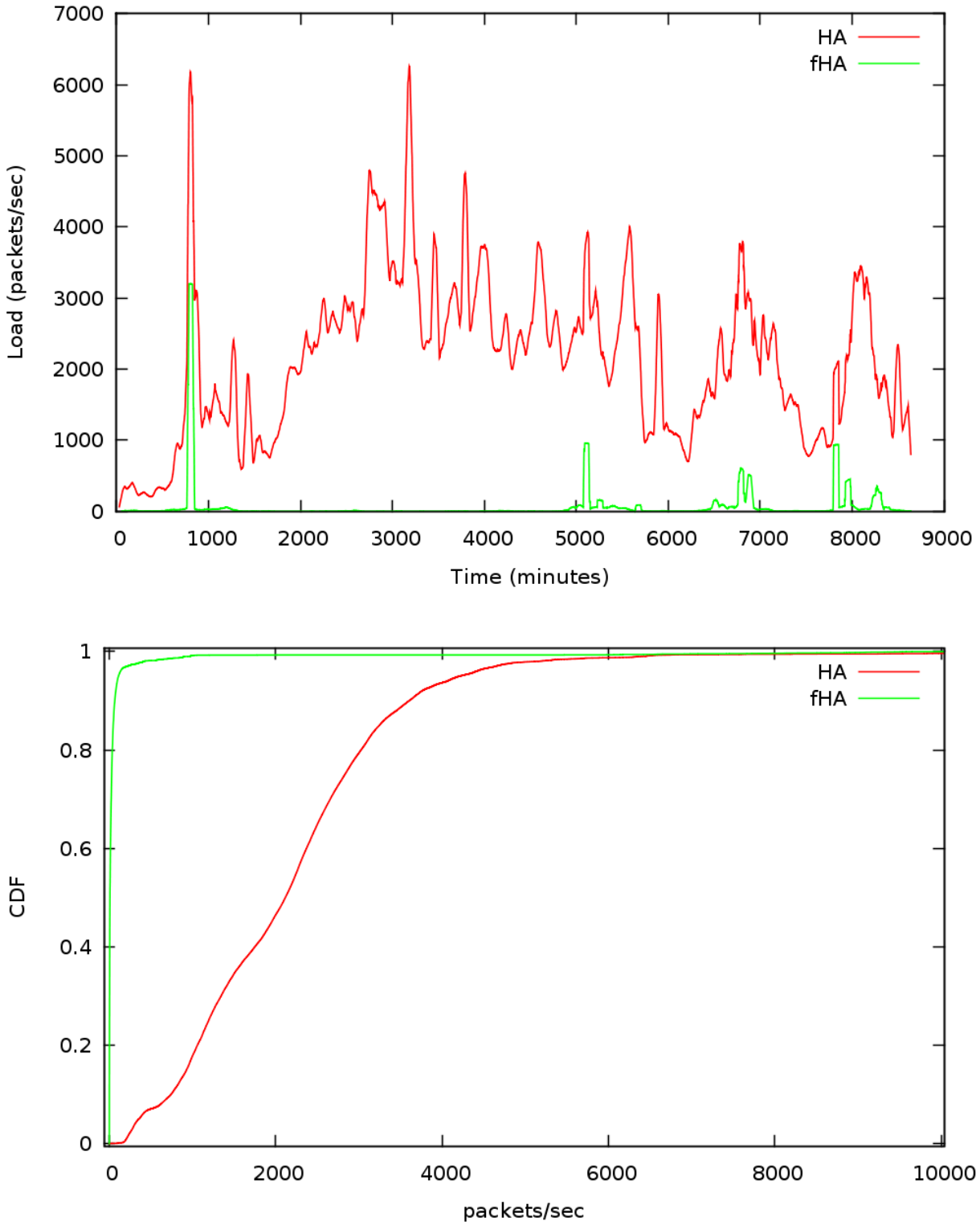


Figure 4.7: MIPv4/NEMO: HA versus fHA load in packets per second.

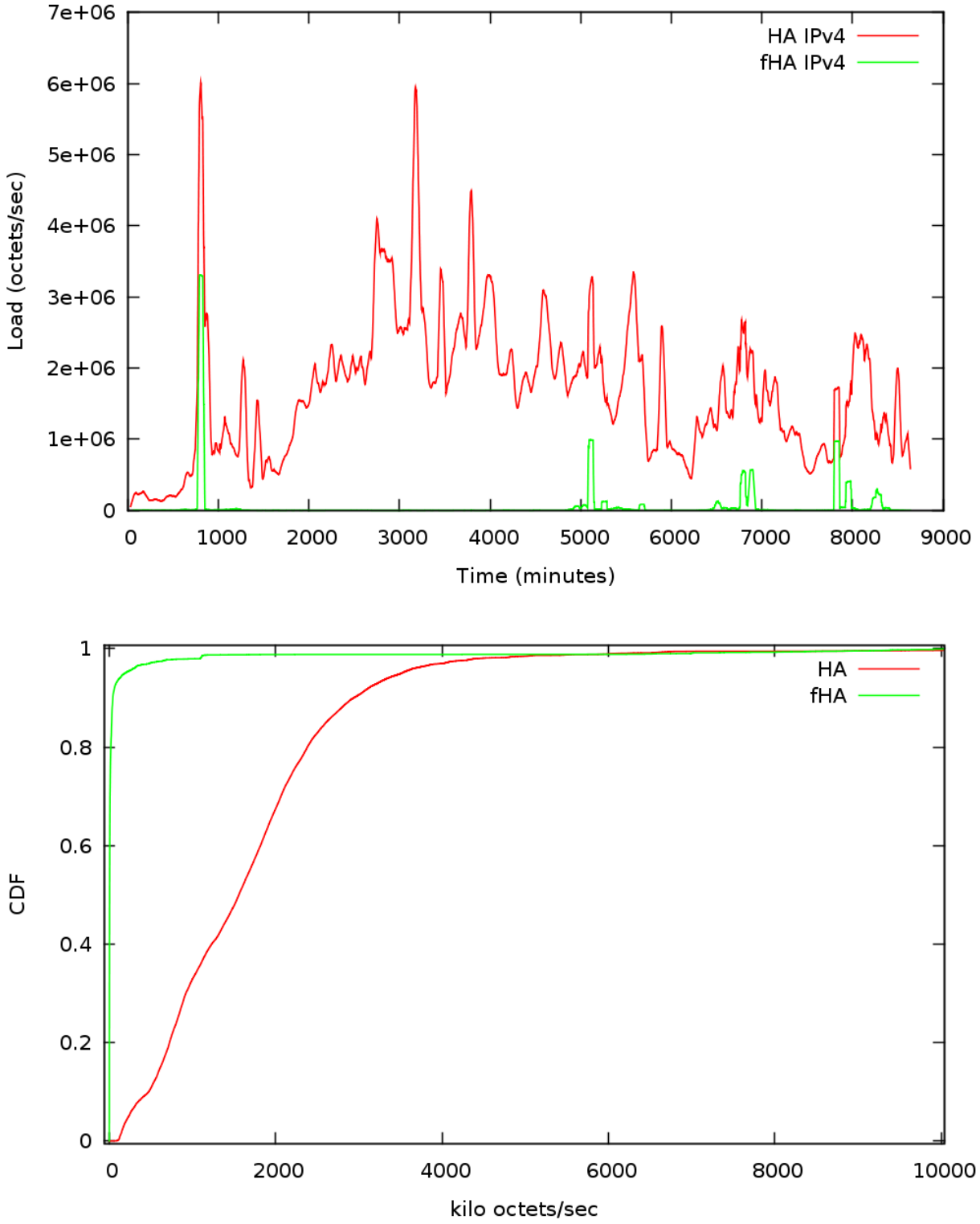


Figure 4.8: MIPv4/NEMO: HA versus fHA load in octets per second.

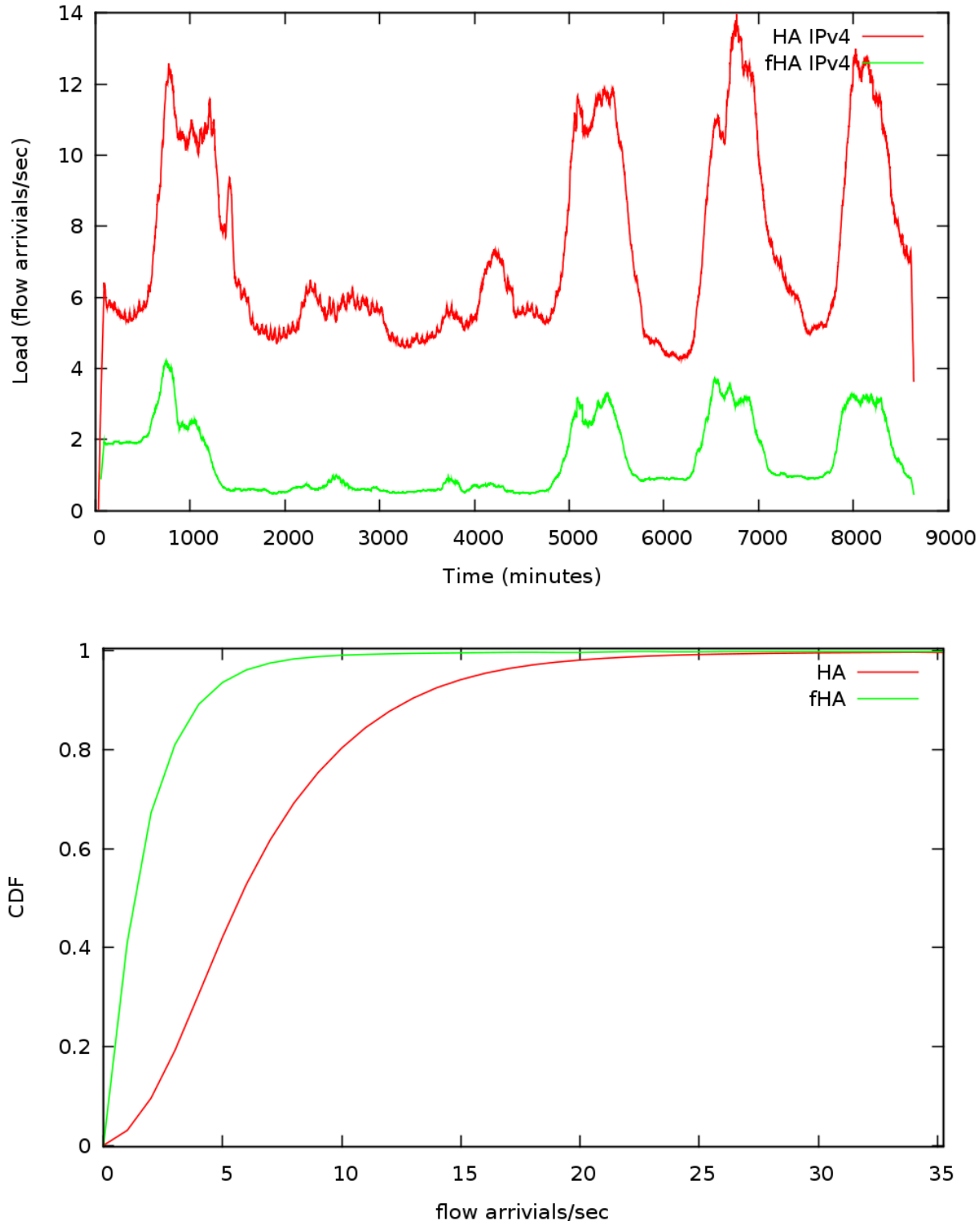


Figure 4.9: MIPv4/NEMO: HA versus fHA load in flow arrivals per second.

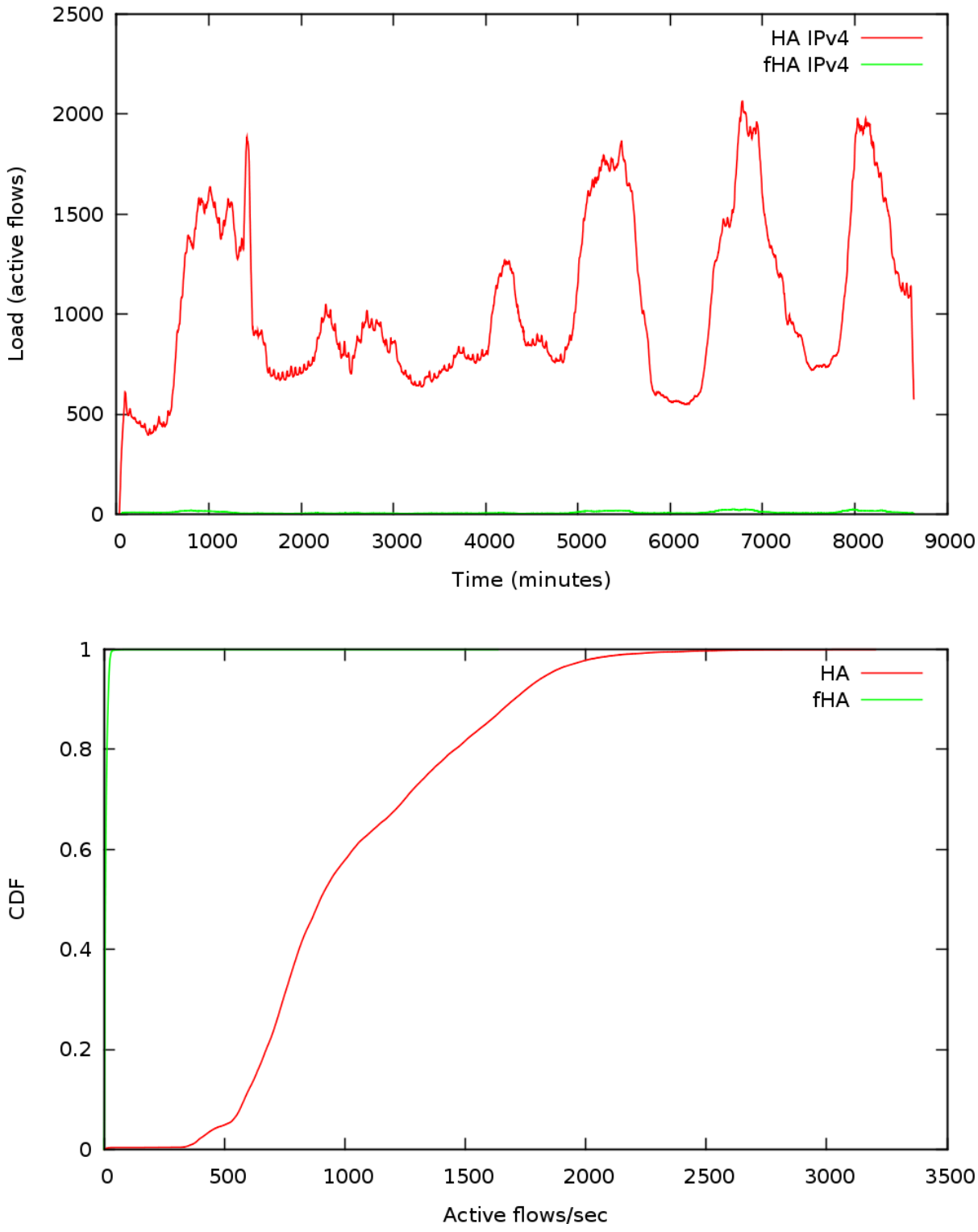


Figure 4.10: MIPv4/NEMO: HA versus fHA load in active flows per second.

4.2.1.3 Home Agent Load, MIPv6

This section presents the experimental results for the mobility protocol MIPv6, emphasizing the difference in load between the fHA and the HA. In this case, the difference in load is almost inexistent, especially when regarding packets per second or octets per second, the exact same metrics where the difference was extremely visible in case of MIPv4/NEMO. For this reason, the figures representing these 2 measures (fig. 4.11 and 4.12), both of instantaneous value and CDF, are not shown entirely, but have been zoomed in to a level considered convenient. Please note that when viewed at default zoom level, the lines for these two metrics overlap, the values being practically the same. The explanation for this is the fact that the vast majority of the traffic through the standard HA is the internal one, that also passes through the flexible HA. Because the fraction of the external traffic that passes through the standard HA also depends on the time it takes to complete RR (which is roughly 2*RTT between the MN and the CN), we have computed the load of the standard HA for three different cases of RR duration: 300ms, 500ms, and 2000ms. These values have been randomly but reasonably chosen, to reflect the variations in latency brought by different qualities of connection of the Home Network to the internet, and by different layer 1/2 technologies that facilitate mobility. Please note that, as stated earlier, for the sake of the fairness of the comparison, the RR decision mechanism has been optimally configured for minimizing the load on the standard HA. In reality, all implementations of MIPv6 will be optimized from all points of view, which will greatly increase the fraction of the external flows that pass through the HA, thus, significantly increase it's total load.

In case of MIPv6, the figures have changed dramatically. The optimal RR decision scheme employed in this simulation ensures that almost none of the external traffic passes through the standard HA, thus the advantage of the fHA is only marginal. This is most visible in the case of packets/second (fig. 4.11) and especially octets/second (fig. 4.12), which now have almost the exact same instantaneous value in the case of the two architectures. Despite of this, the difference is still visible in the active flows per second metric (fig. 4.14), because of the fact that internal flows are much shorter than external flows. For this reason, the small amount of extra traffic through the standard HA, all being to and from the internet, still adds a significant difference in terms of active flows the HA has to keep state of. It is also important to remark that after performing RR the vast amount of data is sent through the optimal route, all flows are initiated through the HA. RR duration time does also not affect flow arrivals, only flow durations, so the figure for flow arrivals per second is the same as the one incase of MIPv4/NEMO, with the 3 cases of RR duration time overlapping perfectly (fig. 4.11).

4.2.1.4 Overview of results

In this sections, a summary of the results is presented. First, for MIPv4/NEMO, the results show the considerable difference in load between the fHA and the standard architecture. The difference in load is handled by the ERs. This is not a supplementary load for the ERs, since all the traffic

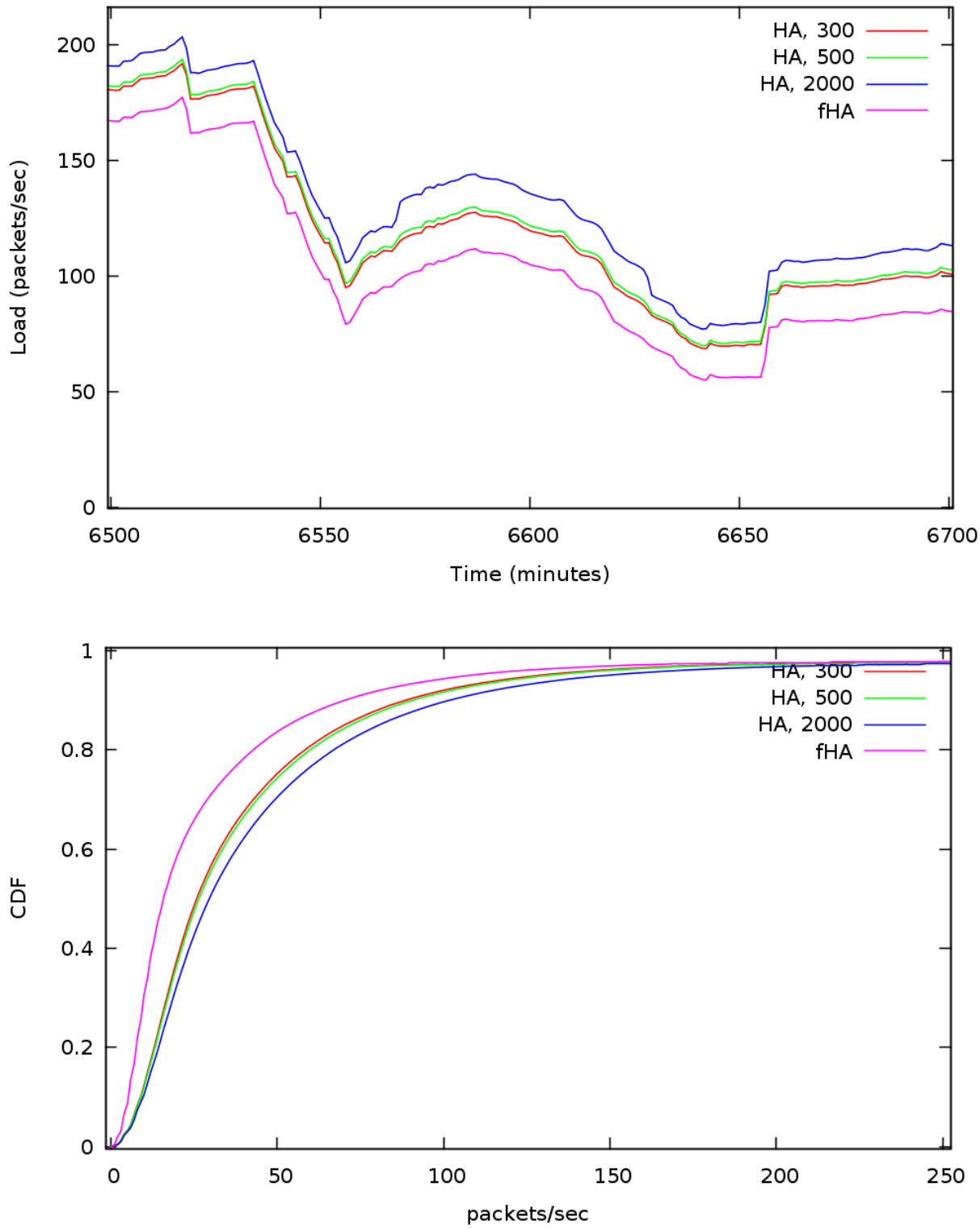


Figure 4.11: MIPv6: HA versus fHA load in packets per second.

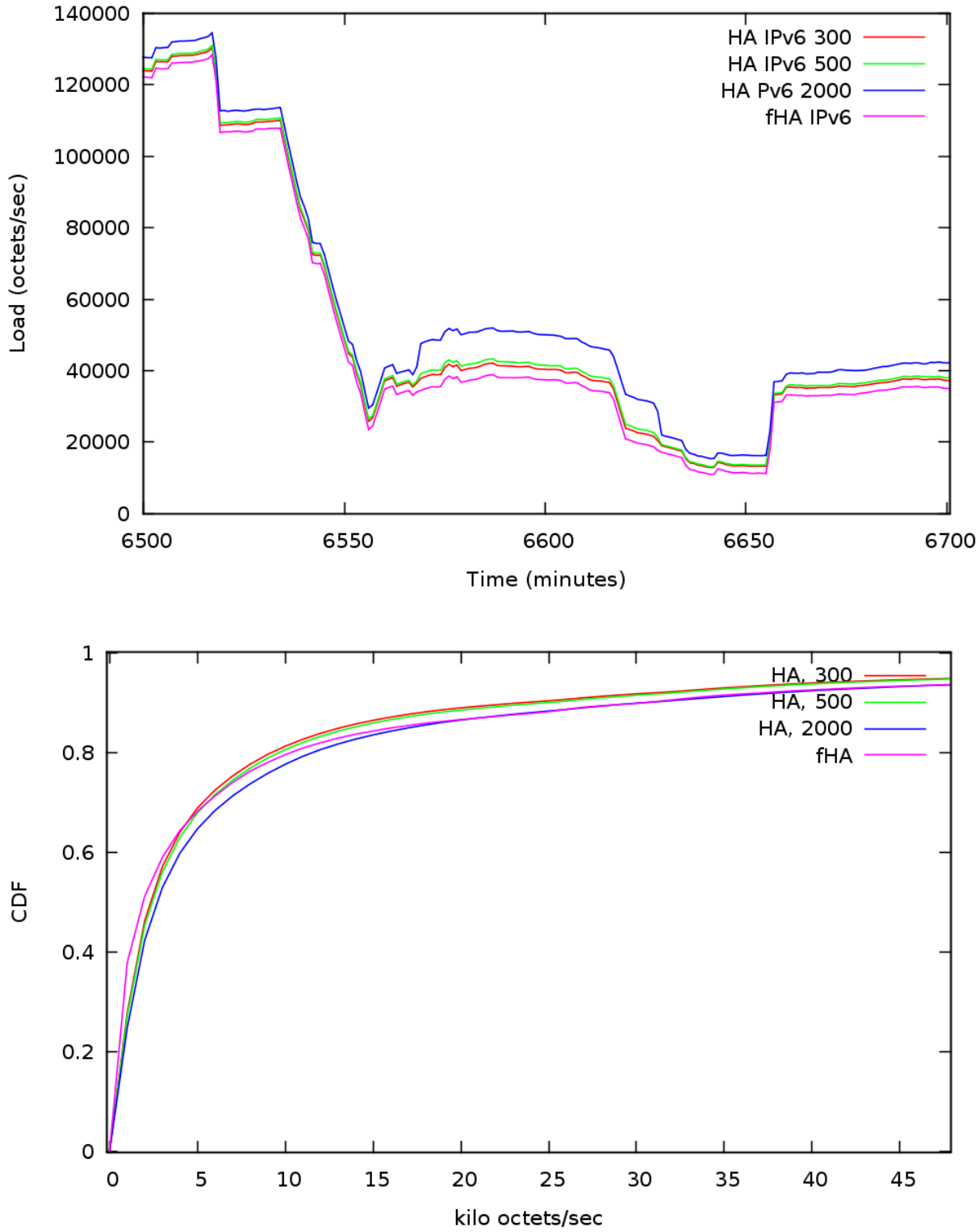


Figure 4.12: MIPv6: HA versus fHA load in octets per second.

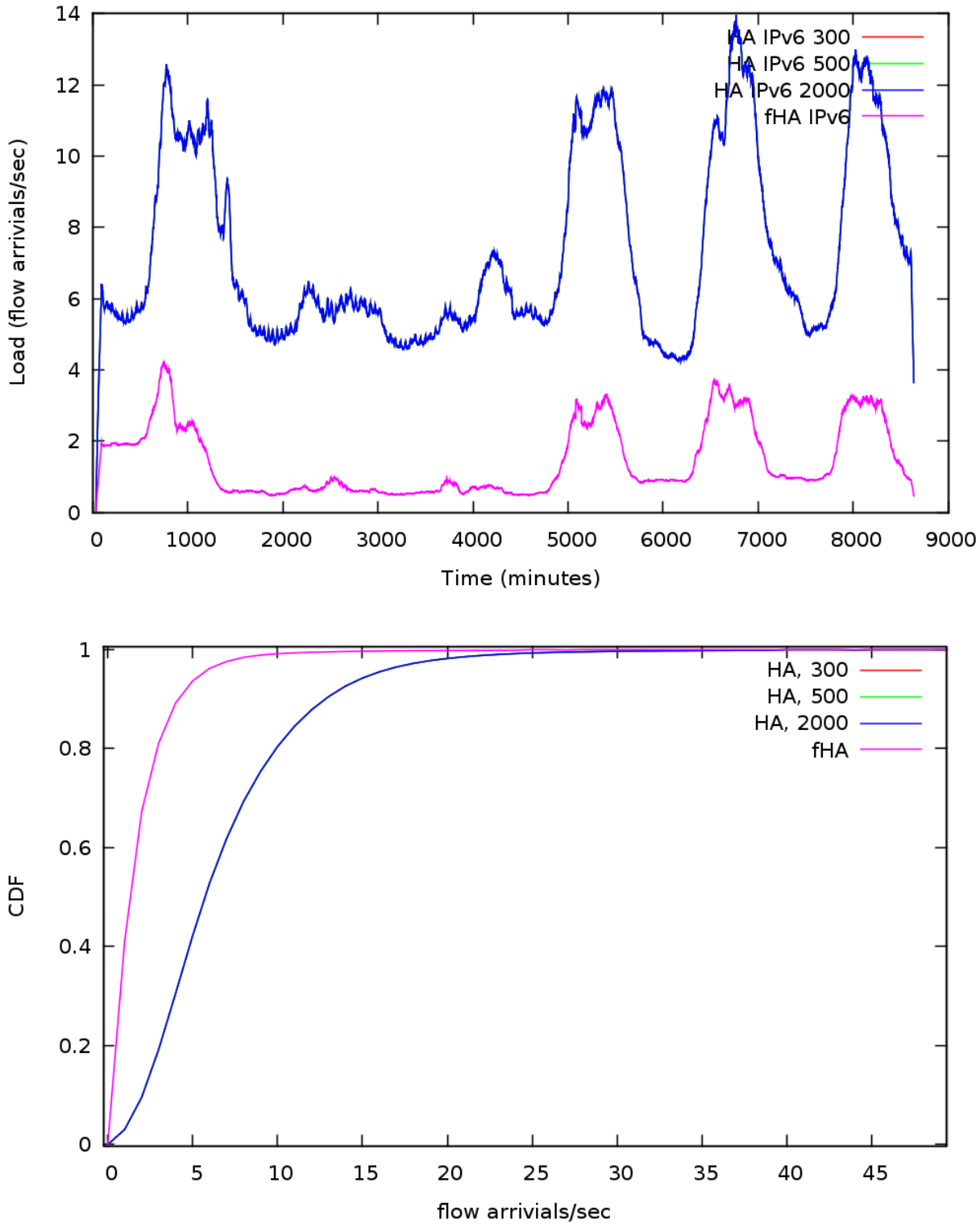


Figure 4.13: MIPv6: HA versus fHA load in flow arrivals per second.

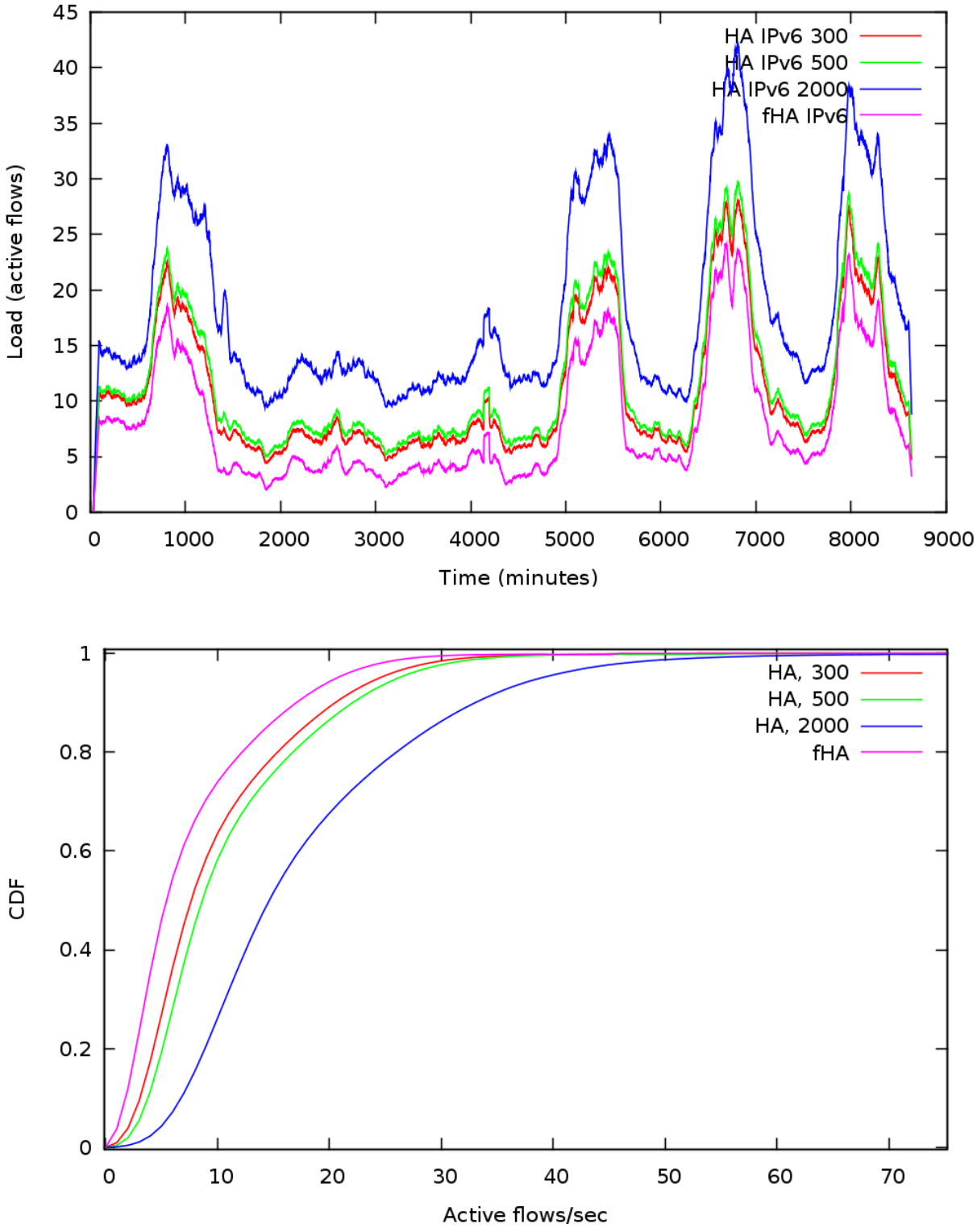


Figure 4.14: MIPv6: HA versus fHA load in active flows per second.

passes through them when entering and exiting the network in the case of the HA too. Secondly, the routers are network equipment optimized especially for this kind of operation. The advantage of the fHA is clear in this case from the point of view of the load. Please refer to the table 4.16 for a experimentally obtained values, and a comparison of the mean load of the standard HA with respect to the fHA.

Table 4.16: Overview of the load of the two architectures of HAs for MIPv4/NEMO, UPC dataset

Units	Architecture	Minimum	Maximum	Mean	Comparison
Packets / second	HA	418.05	11236.86	1965.55	4707%
	fHA	0.82	9803.57	41.75	100%
Octets / second	HA	233355.43	11196535.4	1526326.59	4430%
	fHA	138.53	10321994.99	34450.83	100%
Active flows / second	HA	209.80	3004.283	868.91	13600%
	fHA	0.46	58.683	6.38	100%
Flow arrivals / second	HA	10.48	41.050	5.90	540%
	fHA	0.41	31.017	1.09	100%

For MIPv6, the results are totally different. While the HA has to process more data in this case too, the additional amount is reduced considerably by the route optimization that MIPv6 employees. Please note that while this mechanism results in a strong decrease in packets per second, octets per second and active flows per second at the home agent, the number of flow arrivals per second is the same with the one in case of MIPv4. This result could change in certain cases being also depended on the route optimization policy. Table 4.17 presents an overview of the obtained results. Please note that the HA has a much smaller increase in load with respect to the standard HA. Again it is important to remember that this is a best case scenario form the point of view of the load on the classic HA, since, for the sake of the fairness of the comparison, whenever a value was up to our choice (for example the maximum return routability binding lifetime, or the threshold of packets for remote routability decision scheme), the value that the classic HA would mostly benefit of was chosen. A real world implementation, being optimized for overall performance, would have sensibly different values for these variables, which would result in a considerable increase in load.

Table 4.17: Overview of the load of the two architectures of HAs for MIPv6, UPC dataset

Units	Architecture	Minimum	Maximum	Mean	Comparison
Packets per second	HA 300	13.13	9820.00	60.18	117.15%
	HA 500	13.39	9820.94	61.03	118.80%
	HA 2000	14.95	9827.08	65.70	127.89%
	fHA	7.72	9812.34	51.37	100.00%
Octets per second	HA 300	1255.46	10323781.65	36407.41	103.94%
	HA 500	1294.29	10324256.70	36690.58	104.75%
	HA 2000	1422.53	10327365.60	38680.82	110.43%
	fHA	593.82	10322520.99	35027.84	100.00%
Active flows per second	HA 300	2.56	60.72	8.73	136.77%
	HA 500	3.05	61.48	9.44	147.77%
	HA 2000	6.50	78.01	14.76	231.16%
	fHA	0.46	58.68	6.38	100%
Flow arrivals per second	HA 300	4.20	41.05	5.89	540.70%
	HA 500	4.11	41.05	5.89	540.70%
	HA 2000	5.08	41.05	5.89	540.70%
	fHA	0.41	31.01	1.09	100.00%

4.2.2 Traces from UoC

This section is dedicated to the experimental results when using as input data the NetFlow records from the University of Coimbra. Nevertheless, since the results are very close to those for UPC, we chose not to present the entire dataset, despite the fact that they would prove the consistency of our experimental results for different input data sets. The reader can find the complete set of plots of the results in the Appendix section while below, tables 4.18 and 4.19 present the overview.

Table 4.18: Overview of the load of the two architectures of HAs for MIPv4/NEMO, UoC dataset

Units	Architecture	Minimum	Maximum	Mean	Comparison
Packets / second	HA	0	2564.12	285.28	947.25%
	fHA	0	1512.33	30.12	100.00%
Octets / second	HA	0.01	2572812.73	211148.19	1092.44%
	fHA	0.19	1980661.60	19328.09	100.00%
Active flows / second	HA	0.05	859.78	127.49	5259.65%
	fHA	0.02	30.73	2.42	100.00%
Flow arrivals / second	HA	0	21.82	0.94	461.76%
	fHA	0.02	10.98	0.20	100.00%

Table 4.19: Overview of the load of the two architectures of HAs for MIPv6, UoC dataset

Units	Architecture	Minimum	Maximum	Mean	Comparison
Packets per second	HA 300	0.03	1554.37	33.68	106.59 %
	HA 500	0.03	1558.07	34.12	107.98 %
	HA 2000	0.03	1577.64	36.43	115.29 %
	fHA	0.03	1527.10	31.59	100.00%
Octets per second	HA 300	1.81	1994811.91	20178.80	103.92 %
	HA 500	1.81	1997048.95	20432.34	105.23 %
	HA 2000	1.81	2009797.18	21858.13	112.57%
	fHA	1.81	1981547.60	19416.71	100.00%
Active flows per second	HA 300	0.01	38.07	2.82	116.50%
	HA 500	0.02	38.90	2.92	120.50%
	HA 2000	0.07	45.22	3.66	150.83%
	fHA	0	30.73	2.42	100.00%
Flow arrivals per second	HA 300	0.05	21.82	0.94	461.76%
	HA 500	0.05	21.82	0.94	461.76%
	HA 2000	0.07	21.82	0.94	461.76%
	fHA	0	10.98	0.20	100.00%

4.3 Costs of fHA over HA

In this subsection the costs of the fHA in terms of signaling overhead and stored state at the routers is evaluated. As described earlyer, for each mobile node, and for each handover, the fHA has to inform to the ERs. This is done using an IBGP message, either IBGP UPDATE or IBGP WITHDRAW. Therefore, we can state that the cost of the fHA in terms of signaling is O(H), where H is the amount of handovers per unit of time and per mobile node. Unfortunately it is complex to provide realistic values for H since it depends on the specific deployment of the Mobile IP technology, and as stated before, there are not public data traces. Nevertheless, in this section we aim to discuss a realistic worst-case scenario. The H parameter depends on two variables:

- The amount of active mobile nodes served by a single Home Agent.
- The movements of these mobile nodes, specifically the amount of han-

Regarding the first one, consider as an example figure 4.6, which plots the amount of active mobile nodes per second at UPC. The maximum value is of about 300 avtive nodes at the same time, respectively approximately 20% of the nodes. This provides a hint of the amount of the nodes that are active among the total nodes served by a single Home Agent. This value can be reduced by deploying more than one fHA, or by dividing the mobile nodes across several subnetworks.

The second metric is the amount of handovers per second, and per mobile node. Again, this depends on the specific deployment of the Mobile IP technology. Theoretically, a mobile node

performs a handover each time it changes from one subnetwork to another one. A single subnetworks may provide connectivity to a large geographical area, and this depends on the wireless technology used. All things considered, a handover rate of 1 handover per node each 10 seconds is an unrealisticly high value. In our case this would result in a worst case scenario of 30 handovers per second. Therefore, in this case, the fHA should send 30 IBGP UPDATE messages, within the same second.

We have investigated if an ER can support this load, that is tens of IBGP UPDATE messages per second. In [42] the authors describe and analyse in details the various factors that inuence the convergence time of intradomain link state routing protocols. In order to carry out their analysis they measure the update time of a new router on a Cisco 12000 router. Their results show that, on average, each update takes 100s per new route. It is safe to assume that for a less powerfull router it would take 10 times more, that means 1ms per new route. Recall that our proposed approach fHA requires, in the worst case, tens of updates/s, hence this shows that it is feasible since in the worst case processing the signaling would demand 3% of the routers processing power.

Additionally, regarding the stored state at the router, each mobile node that it is away requires 1 tunnel and 1 route towards the mobile node. The tunnel requires 1 address (32 bytes for IPv4 and 128 bytes for IPv6), while the route requires 2 address (Home Address and Care-of Address) plus a lifetime. In total, this should not exceed, including the memory required for indexing 100 bytes of memory per node, thus a total of 30 kilobytes, which is a small fraction of the memory available on any modern router.

Chapter 5

Conclusions

In this paper "the flexible Home Agent", a novell Home Agent architectre, has been analyzed, and a simulator has been created to estimate it's load as precise as it is possible. Also, the load of the standard Home Agent has been simulated as a comparison. The conclusion of the simulation was that, for the MIPv4 and MEMO mobility protocols, the improvement in load at the HA is spectacular. In case of MIPv6, the improvement dropps to more modest values. Please remember that the here simulated standard MIPv6 is optimized for lowering the HA's load. Any real implementation will most probably greatly exceed the load of the here simulated one, scenario in which the fHA will be again in a considerable advantage from the point of view of the load.

As future work, the simulator will be platform independent and fully standalone, and easily work with other sets of input data, so any researcher can be able to estimate the load of a hypothetical HA deployed in any network he posesses NetFlow traces of. The simulator will also offer the posibility to adjust the RR decision scheme, together with other key variables of the simulation.

References

[1] C. Perkins, D. Johnson, and J. Arkko, “Mobility Support in IPv6”, RFC 3775, 2004

[2] C. Perkins, “IP Mobility Support for IPv4”, RFC 3220, 2002

[3] Jahanzeb Faizan et al. “Problem Statement: Home Agent Reliability” IETF Draft (Work in Progress), 2004

[4] F.Heissenhuber, W. Fritsche and A. Riedl “Home Agent Redundancy and Load Balancing in Mobile IPv6” in Proc. 5th Int. Conf. Broadband Communications, 1999

[5] H.Deng et al, “Load Balance for Distributed Home Agents in Mobile IPv6” in Proc. 14th IEEE PIMRIC 2003.

[6] H.Deng et al, “Load Balance for Distributed Home Agents in Mobile IPv6” IETF Draft (Work in Progress), 2003

[7] J. Faizan et al. “Efficient Dynamic Load Balancing for Multiple Home Agents in Mobile IPv6 based Networks” in Proc. 5th Int. Conf. Pervasive Services, 2005.

[8] R. Wakikawa et al. “Home Agent Reliability Protocol” IETF Draft (Work in Progress), 2006.

[9] R. Wakikawa et al. “Inter Home Agents Protocol Specifications” IETF Draft (Work in Progress), 2006

[10] R. Wakikawa et al. “Virtual mobility control domain for enhancements of mobility protocols”, INFOCOM 2005

[11] Albert Cabellos-Aparicio, Jordi Domingo-Pascual, “A Flexible and Distributed Home Agent Architecture for Mobile IPv6-based Networks” in Proceedings of IFIP Networking 2007

[12] J. Faizan , M. Khalil and H. El-Rewini, “VHARP: Virtual Home Agent Reliability Protocol for Mobile IPv6 based Networks”

[13] B. Chambless, and J. Binkley, “Home Agent Redundancy Protocol,” IETF Draft, draft-chambless-mobileip-harp-00.txt (work in progress), October 1997.

[14] R. Ghosh, and G. Varghese, "Fault Tolerant Mobile IP," Washington University, Technical Report (WUCS-98-11), 1998.

[15] J. Ahn, and C. S. Hwang, "Efficient Fault-Tolerant Protocol for Mobility Agents in Mobile IP," in Proc. 15th Int. Parallel and Distributed Processing Symp., California, 2001.

[16] K. Leung, and M. Subbarao, "Home Agent Redundancy in Mobile IP," IETF Draft, draft-subbarao-mobileip-redundancy-00.txt (work in progress), June 2001.

[17] M. Khalil, "Virtual Distributed Home Agent Protocol (VDHAP)," U.S.Patent 6 430 698, August 6, 2002.

[18] J. Lin, and J. Arul, "An Efficient Fault-Tolerant Approach for Mobile IP in Wireless Systems," IEEE Trans. Mobile Computing, vol. 2, no. 3, pp. 207-220, July-Sept. 2003.

[19] R. Wakikawa, V. Devarapalli, and P.Thubert, "Inter Home Agents Protocol (HAHA)," IETF Draft, draft-wakikawa-mip6-nemo-haha-00.txt (work in progress), October 2003.

[20] F. Heissenhuber, W. Fritsche, and A. Riedl, "Home Agent Redundancy and Load Balancing in Mobile IPv6," in Proc. 5th International Conf. Broadband Communications, Hong Kong, 1999.

[21] PalChaudhuri, S. et al "Perfect Simulations for Random Trip Mobility Models" 38th Simulation Symposium, 2005

[22] Paul Moceri "Enabling Network Mobility: A Survey of NEMO" `http://www.cse.wustl.edu/~jain/cse574-06/ftp/network_mobility.pdf`

[23] M.Cristian, A.Cabellos-Aparicio, R.Cosma, J.Domingo-Pascual, P.Vale Pinheiro, F.Boavida & V.Dobrota, "Estimation of a Mobile IPv6s Home Agent Load", 8th Edition of the International Symposium on Electronics and Telecommunications ETC 2008, Timisoara, September 25-26, 2008, Scientific Bulletin of the "POLITEHNICA" University of Timisoara, Romania, Transactions on ELECTRONICS and COMMUNICATIONS, Tom 53(67), Fascicola 2, 2008, pp.129-134, ISSN: 1583-3380

[24] F.Hernandez-Campos, M.Karaliopolus, M.Papadopouli, H.Shen Spatio-Temporal Modelling of Traffic Workload in a Campus WLAN, WICON 2006

[25] M. Karaliopolus, M. Papadopouli, E.Raftopolous, H.Shen. On scalable measurement-driven modelling of traffic demand in large WLANs, LANMAN 2007

[26] Ruben Cuevas, Albert Cabellos-Aparicio, Angel Cuevas, Jordi Domingo-Pascual, Arturo Azcorra "fP2P-HN: A P2P-based Route Optimization Architecture for Mobile IP-based Community Networks"

[27] S. Kent et al Security Architecture for the Internet Protocol RFC 2401, 1998

[28] S. Deering et al Internet Protocol, Version 6 Specification, RFC 2460, 1998

[29] H. Soliman et al Hierarchichal Mobile IPv6 Mobility Management (HMIPv6), RFC 4140, 2005

[30] Y. Rekhter et al A Border Gateway Protocol 4 (BGP-4) RFC 1771, 1995

[31] J.Abley et al Goals for IPv6 Site-Multihoming Architectures RFC 3582, 2003

[32] Beginner's Introduction to Perl, http://www.perl.com/pub/a/2000/10/begperl1.html

[33] Perlintro unix manual page, http://perldoc.perl.org/perlintro.html

[34] Perl article of Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Perl

[35] Bash Guide for Beginners, http://www.tldp.org/LDP/Bash-Beginners-Guide/html/index.html

[36] Basic Netflow information on the Cisco Site, http://www.cisco.com/go/netflow

[37] NetFlow article of Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Netflow

[38] Flow-tools unix manual page, http://linux.die.net/man/1/flow-tools

[39] Flow-nfilter unix manual page, http://linux.die.net/man/1/flow-nfilter

[40] HotPlanet 2009: The 1st ACM International Workshop on Hot Topics of Planet-scale Mobility Measurements (colocated with ACM Mobisys) http://www.hotplanetconf.net/2009

[41] R.Pang, M.Allman, M.Bennet "A first look at modern enterprise traffic", In Proc. of ACM Internet Measurement Conference, 2005

[42] Pierre Francois and Clarence Filsfils and John Evans and Olivier Bonaventure, "Achieving sub-second IGP convergence in large IP networks" ACM SIGCOMM Computer Communication Review, July 2005

Appendix A

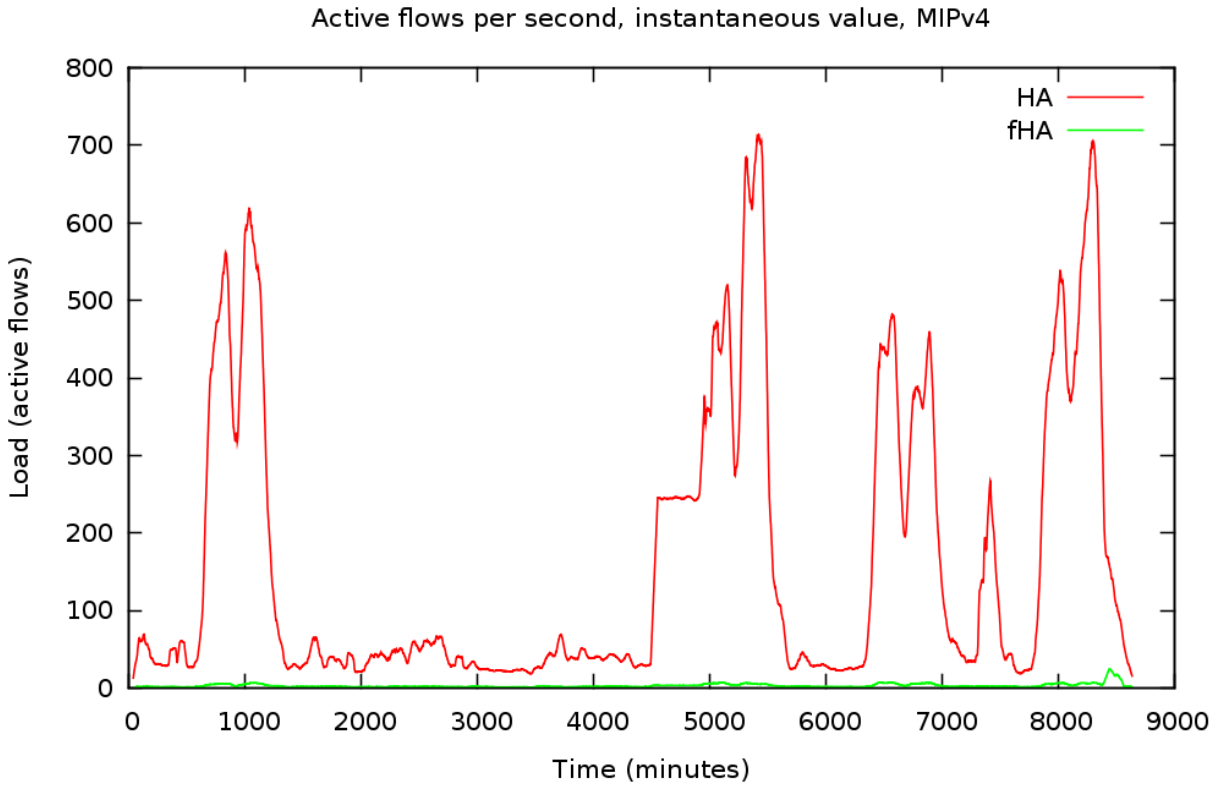
Appdx A: Experimental results for UoC dataset

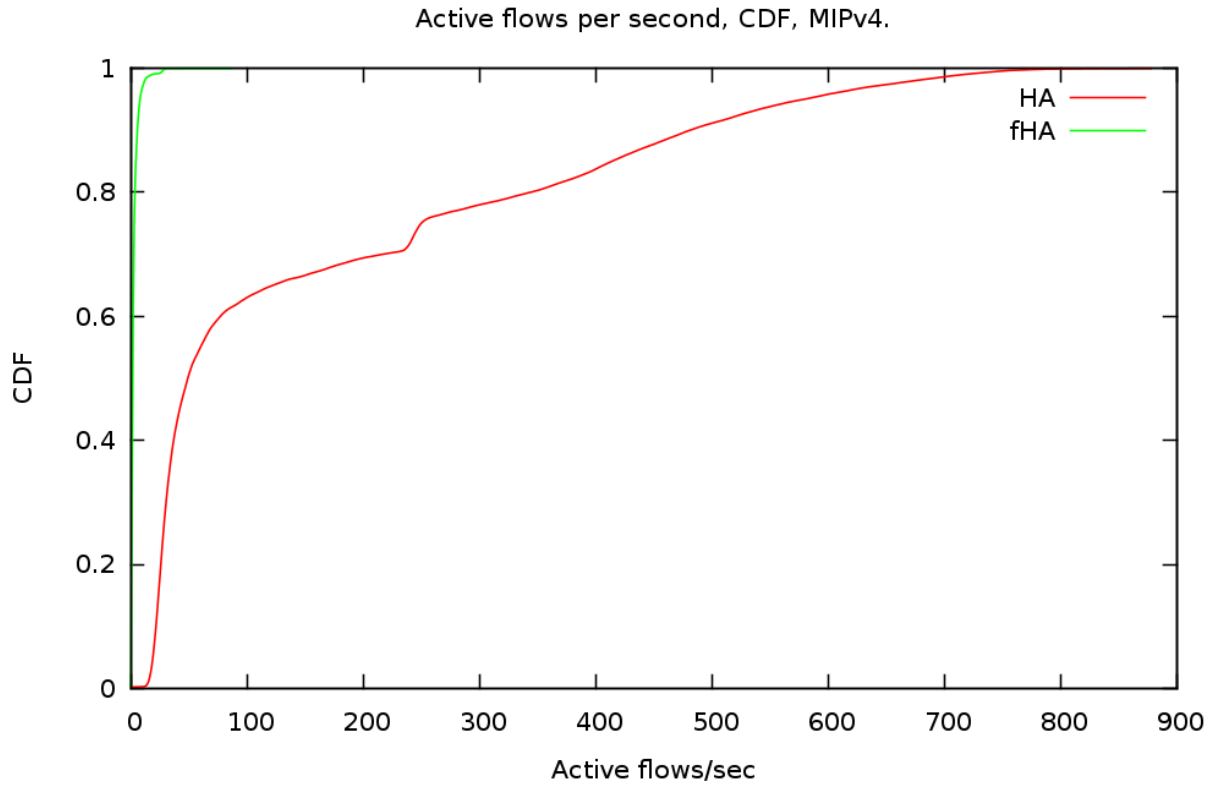
Total statistics	
total packets:	214489165
total Moctets:	159256
total flows:	8197248

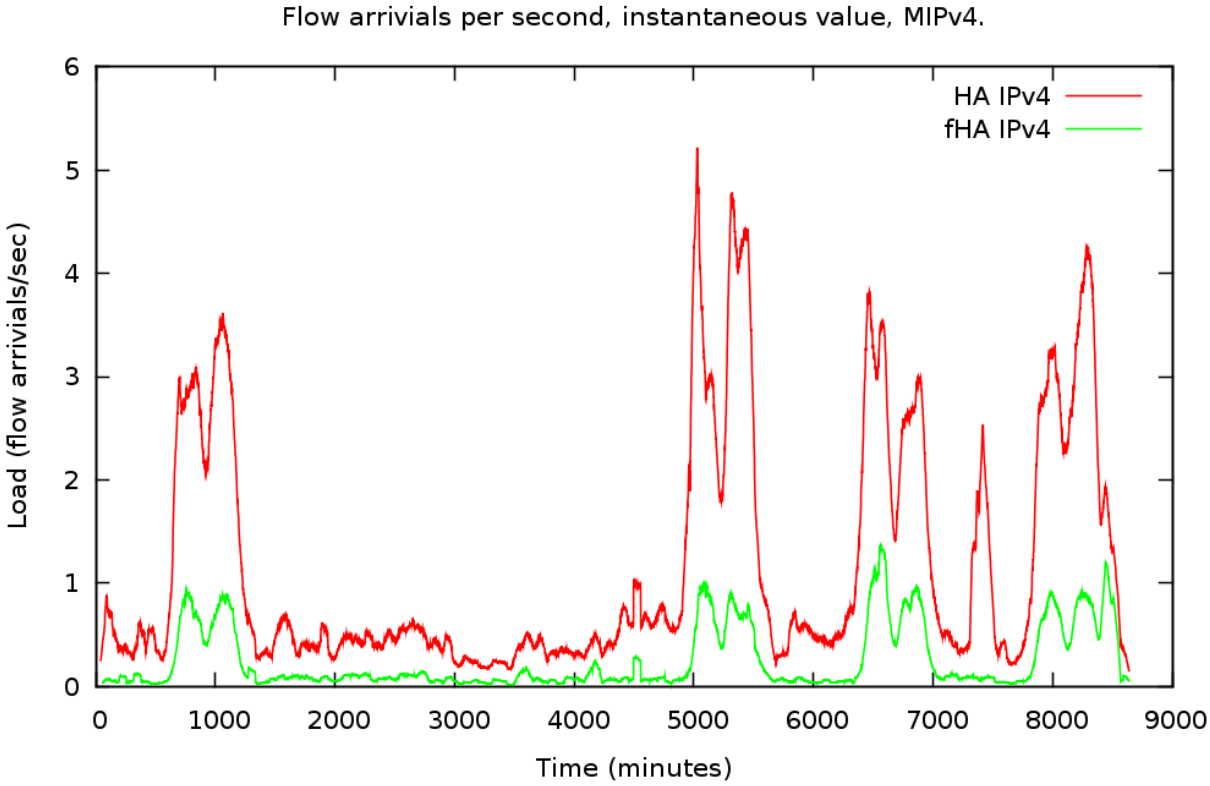
Transport layer and ICMP			
protocol	octets %	packets %	flows %
udp	6.958	3.282	25.401
tcp	92.197	96.592	70.382
icmp	0.846	0.126	4.218
others_t	0.000	0.000	0.000

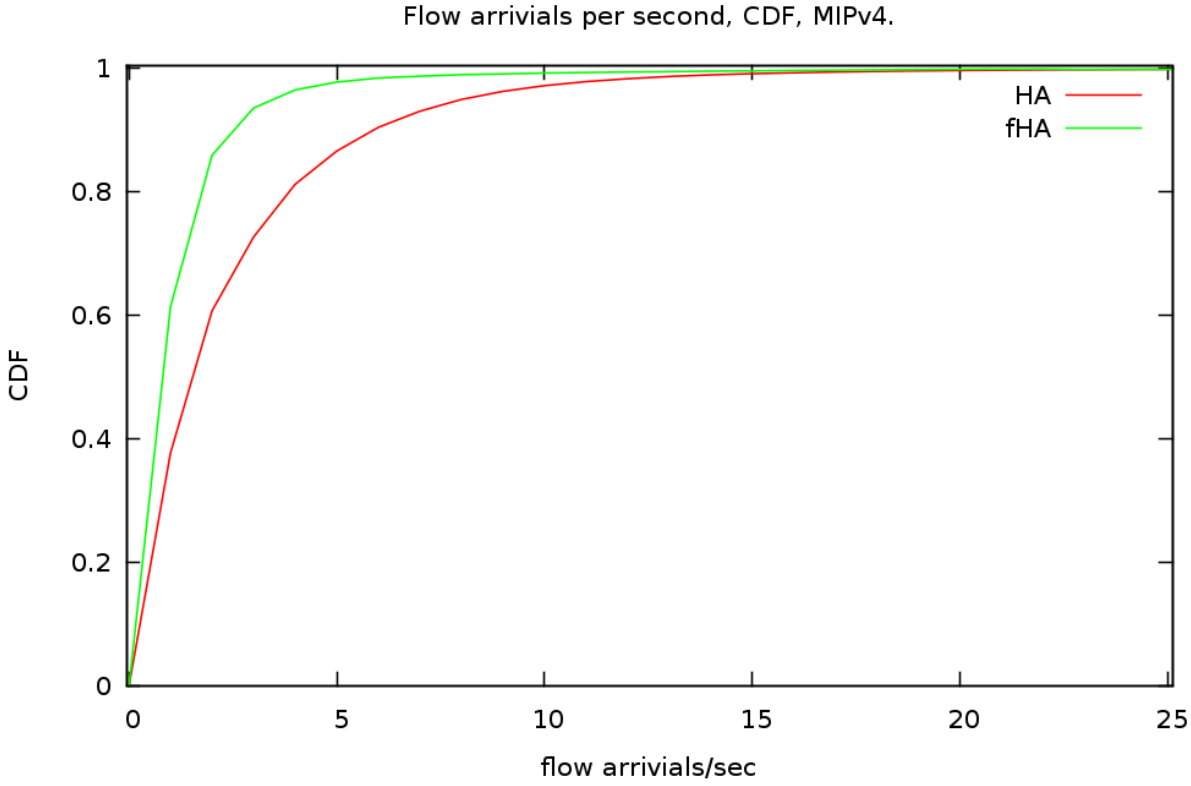
Application layer			
cathegory	octets %	packets %	flows %
bulk	0.870	0.254	3.230
email	11.240	12.346	2.317
interactive	0.060	0.020	0.112
name	0.813	0.176	20.428
net_manage	0.438	0.100	3.105
web	55.179	57.818	57.047
win	0.001	0.000	0.018
auth	0.006	0.007	0.000
printing	0.000	0.000	0.000
streaming	2.676	2.684	0.082
others_a	28.716	26.594	13.661

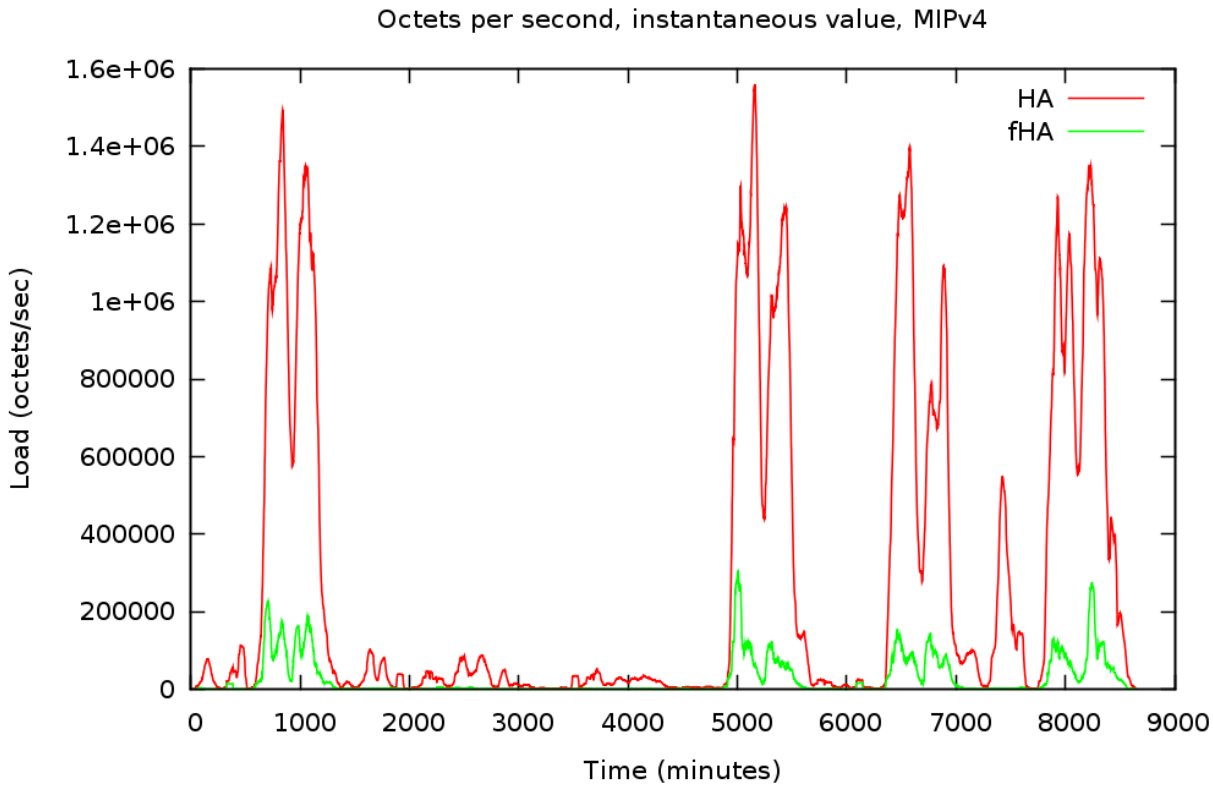
traffic type:	packets,	octets,	flows
type 111 flows:	0.0023%	0.0002%	0.0536%
type 112 flows:	9.3719%	8.7760%	3.5820%
type 21 flows:	0.0003%	0.0000%	0.0003%
type 12 flows:	9.41%	9.25%	3.2%
type 22 flows:	81.21%	81.97%	93.16%

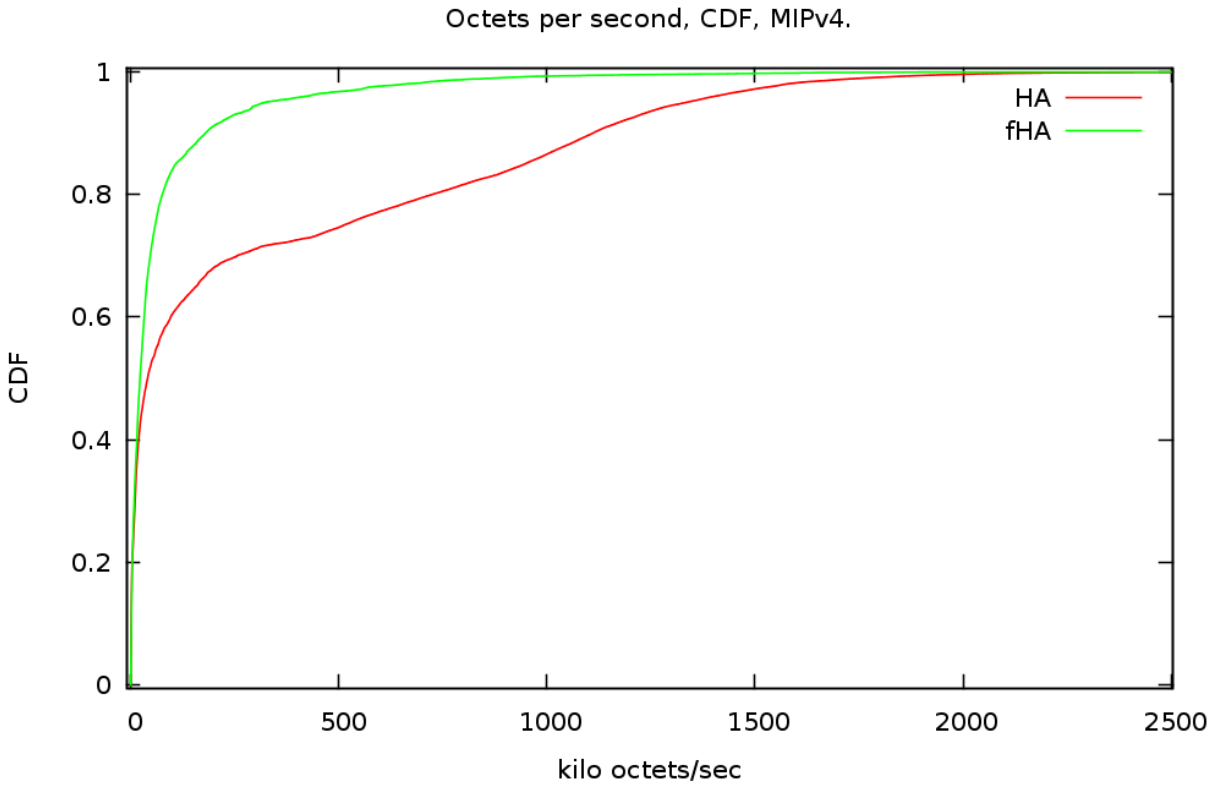

Active flows per second, instantaneous value, MIPv4
Load (active flows)
Time (minutes)
HA
fHA
0
100
200
300
400
500
600
700
800
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

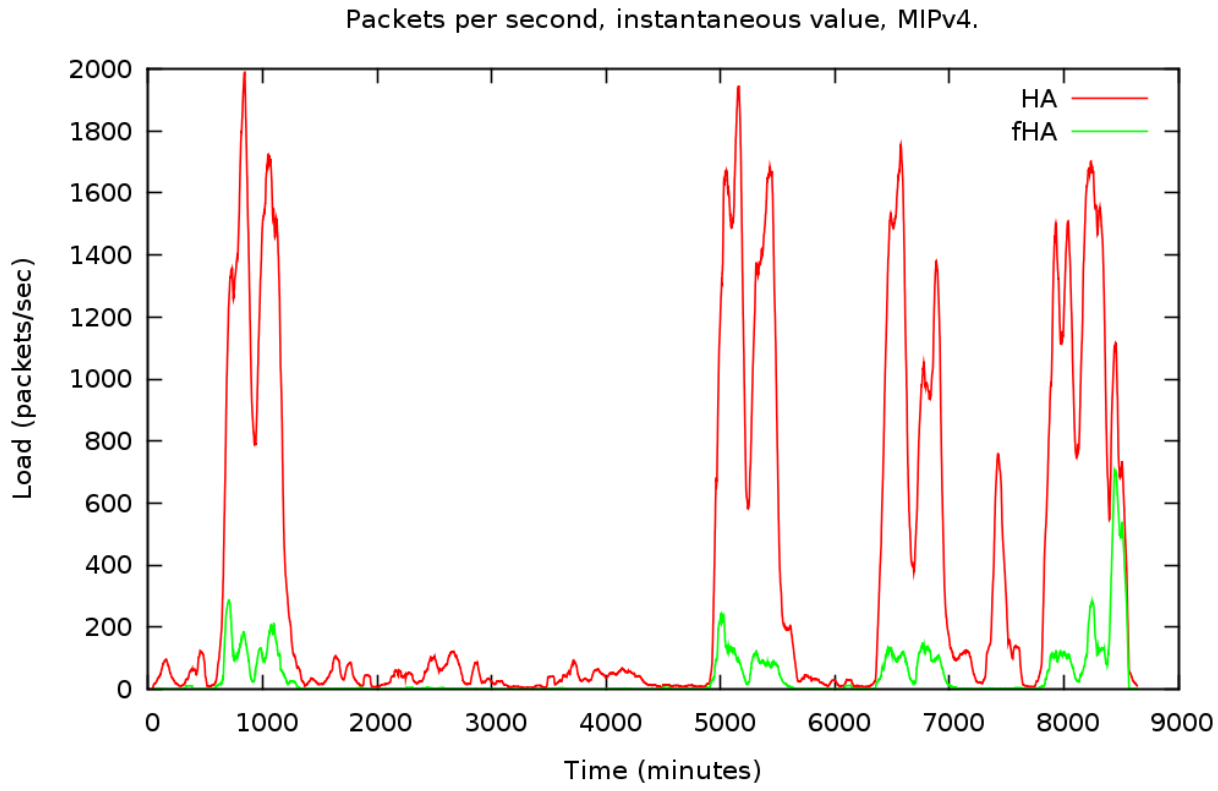

Active flows per second, CDF, MIPv4.
CDF
Active flows/sec
HA
fHA
0
0.2
0.4
0.6
0.8
1
0
100
200
300
400
500
600
700
800
900

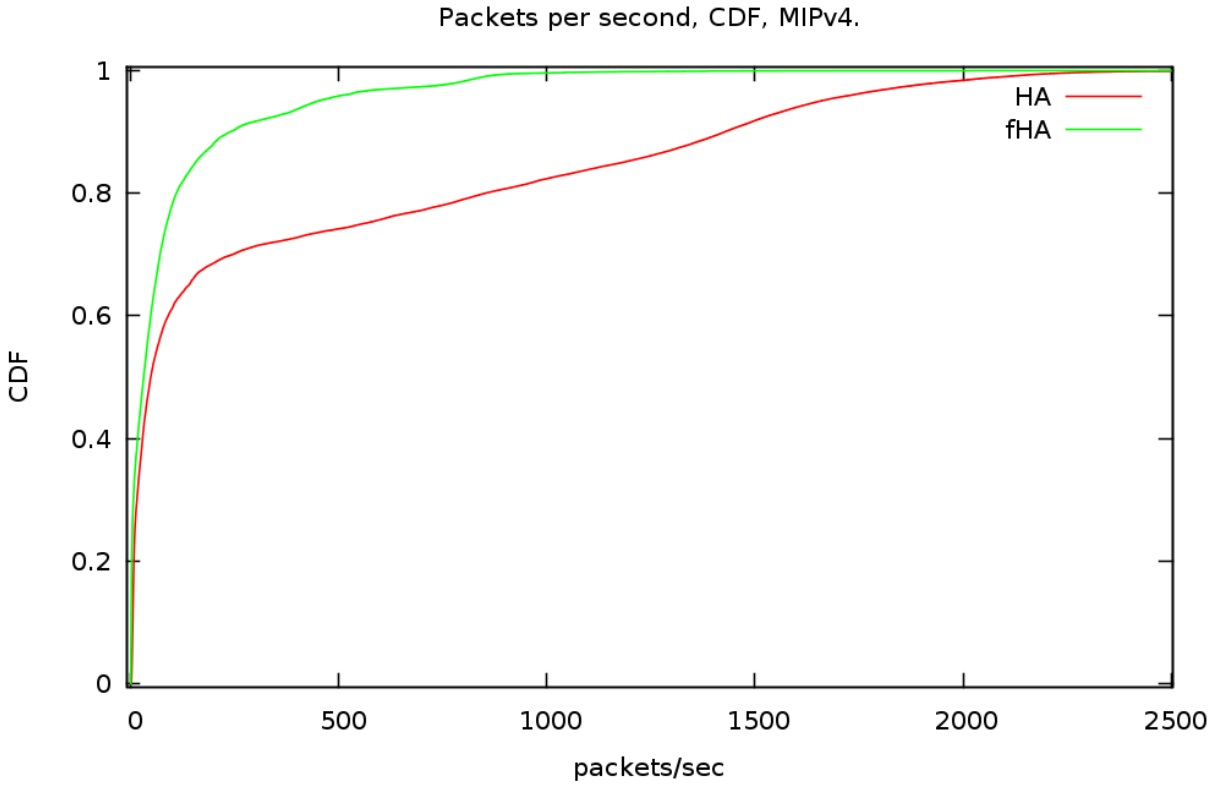

Flow arrivials per second, instantaneous value, MIPv4.
HA IPv4
fHA IPv4
Load (flow arrivials/sec)
Time (minutes)
0
1
2
3
4
5
6
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

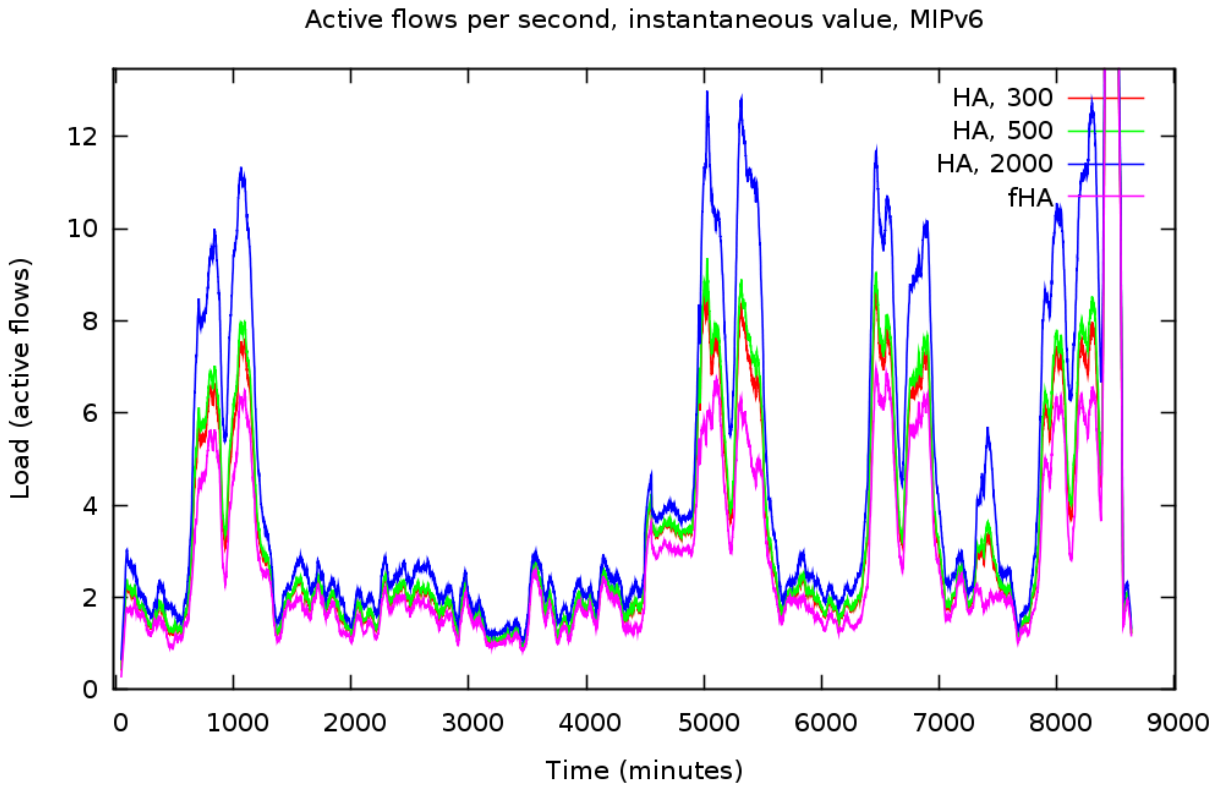

Flow arrivials per second, CDF, MIPv4.
HA
fHA
CDF
flow arrivials/sec
0
0.2
0.4
0.6
0.8
1
0
5
10
15
20
25

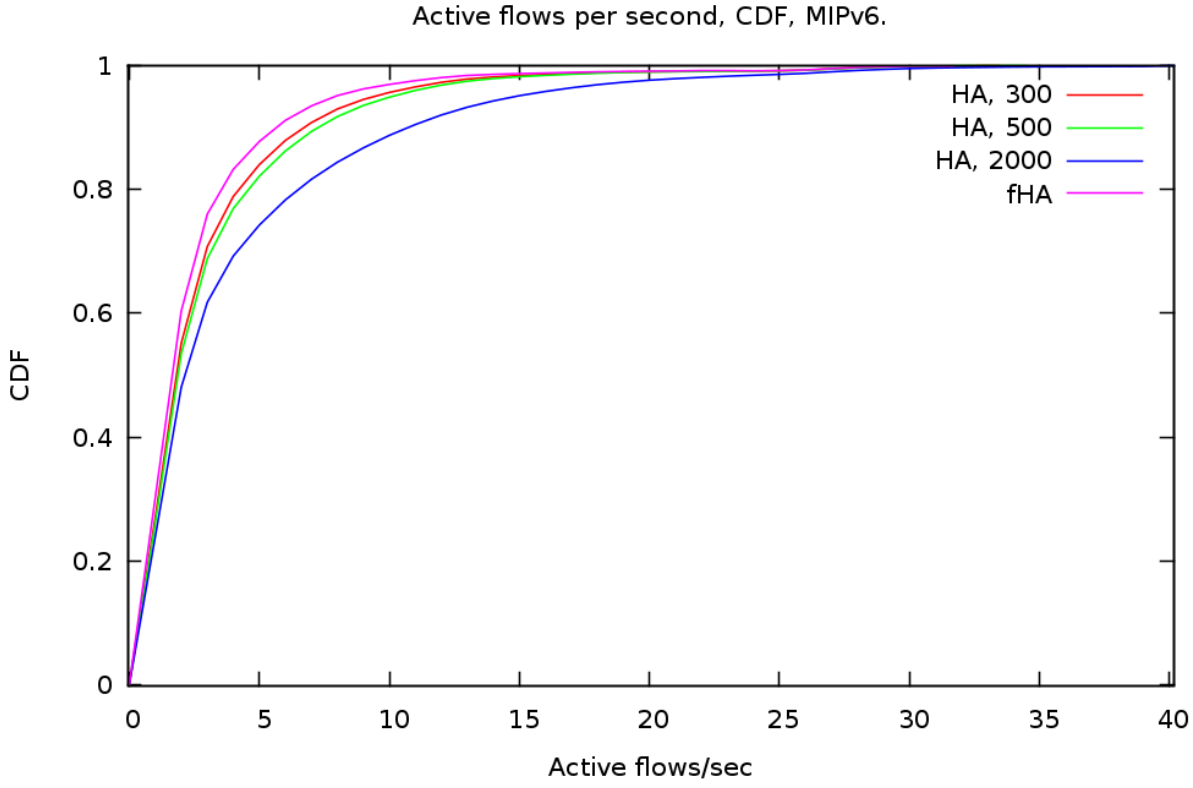

Octets per second, instantaneous value, MIPv4
Load (octets/sec)
Time (minutes)
HA
fHA
1.6e+06
1.4e+06
1.2e+06
1e+06
800000
600000
400000
200000
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

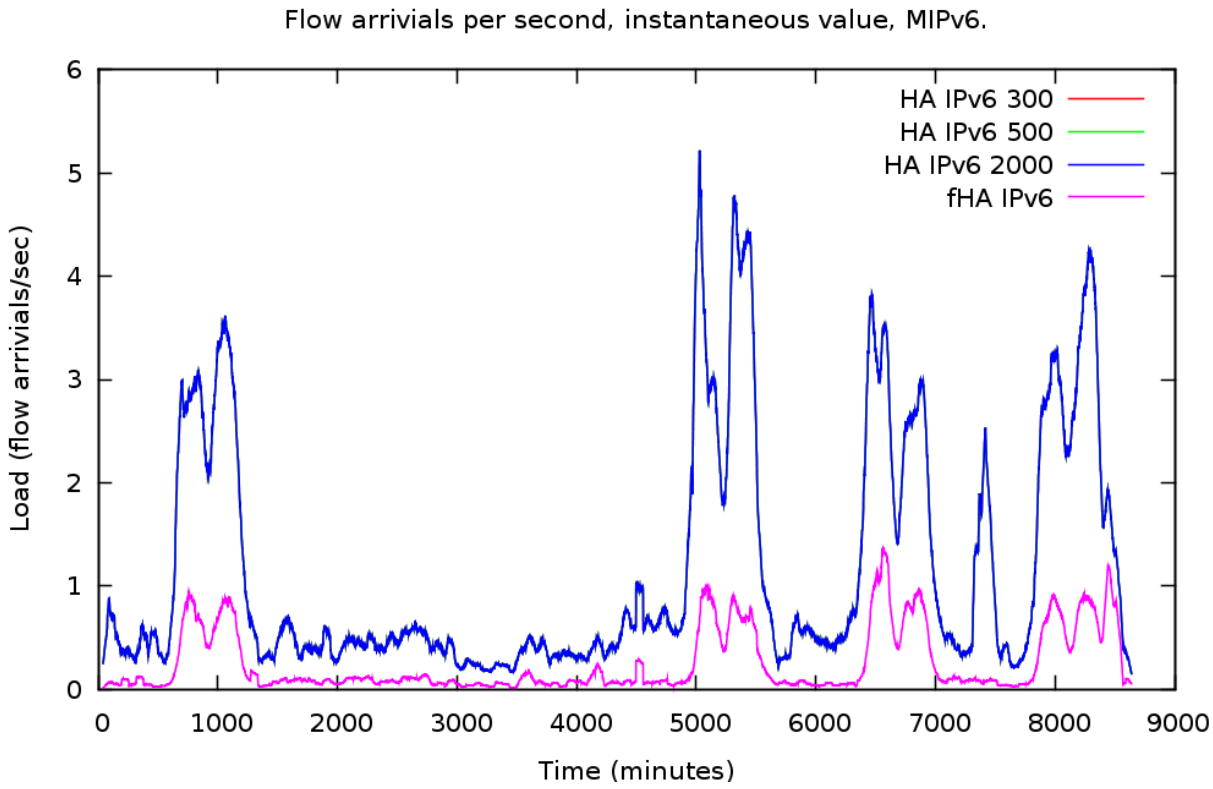

Octets per second, CDF, MIPv4.
CDF
kilo octets/sec
HA
fHA
1
0.8
0.6
0.4
0.2
0
0
500
1000
1500
2000
2500

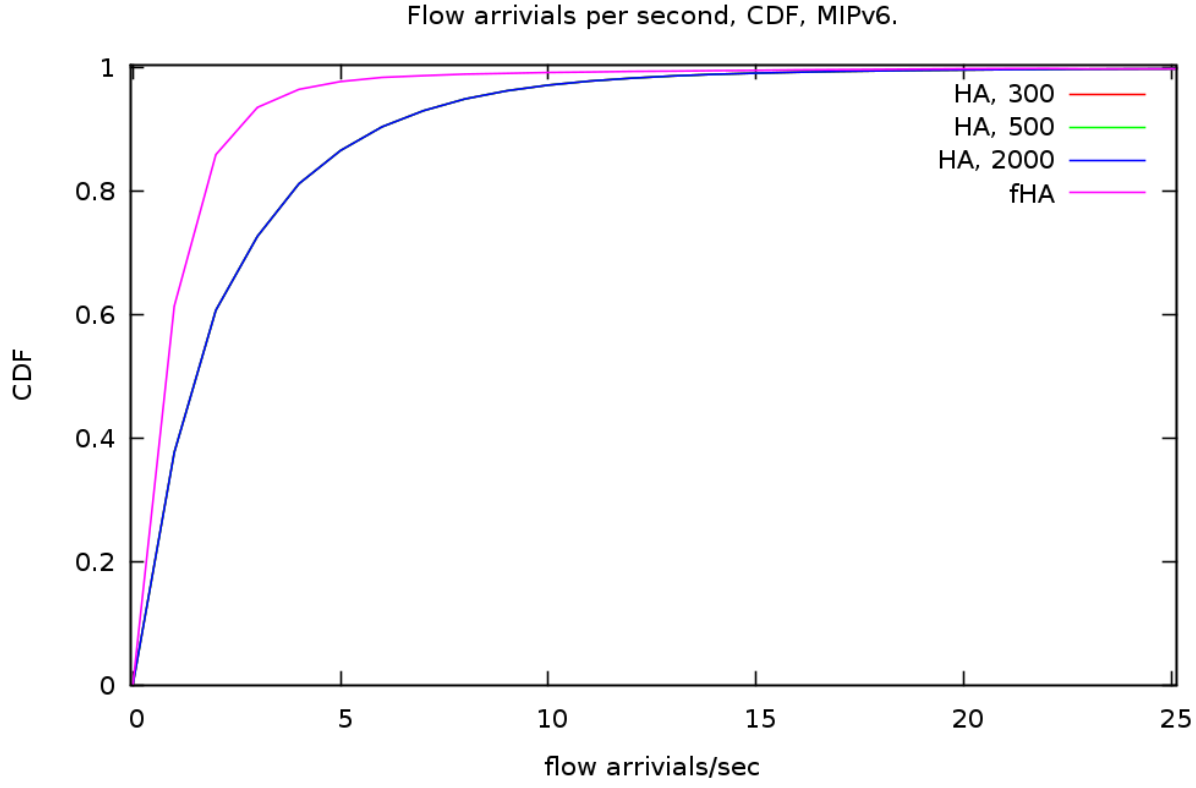

Packets per second, instantaneous value, MIPv4.
Load (packets/sec)
Time (minutes)
HA
fHA
0
200
400
600
800
1000
1200
1400
1600
1800
2000
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

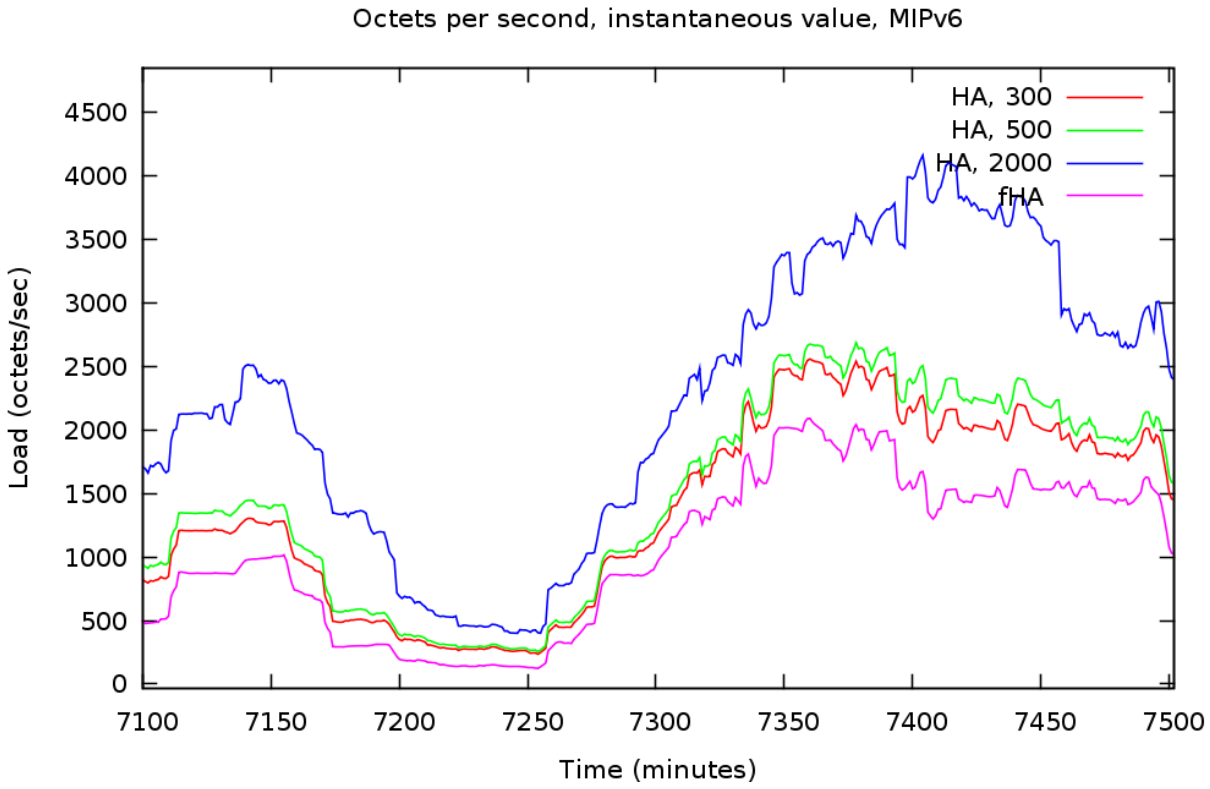

Packets per second, CDF, MIPv4.
CDF
packets/sec
HA
fHA
0
0.2
0.4
0.6
0.8
1
0
500
1000
1500
2000
2500

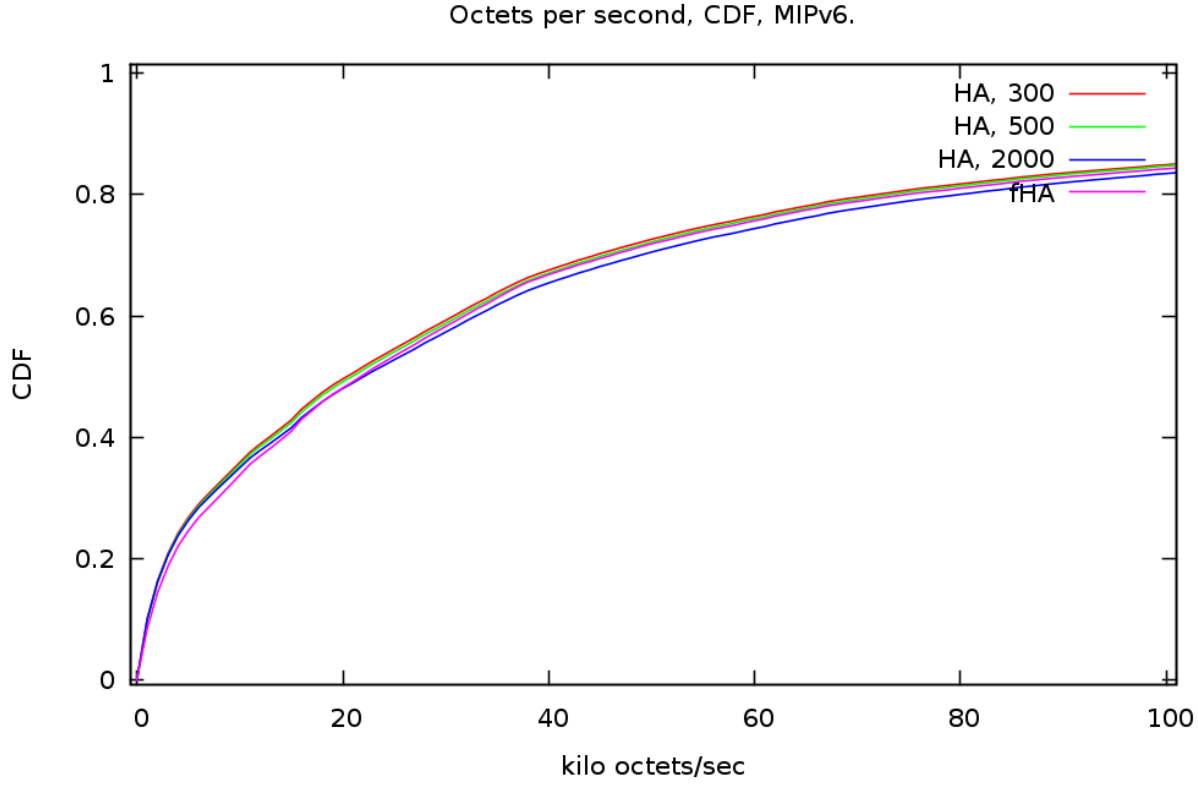

Active flows per second, instantaneous value, MIPv6
Load (active flows)
Time (minutes)
HA, 300
HA, 500
HA, 2000
fHA
0
2
4
6
8
10
12
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

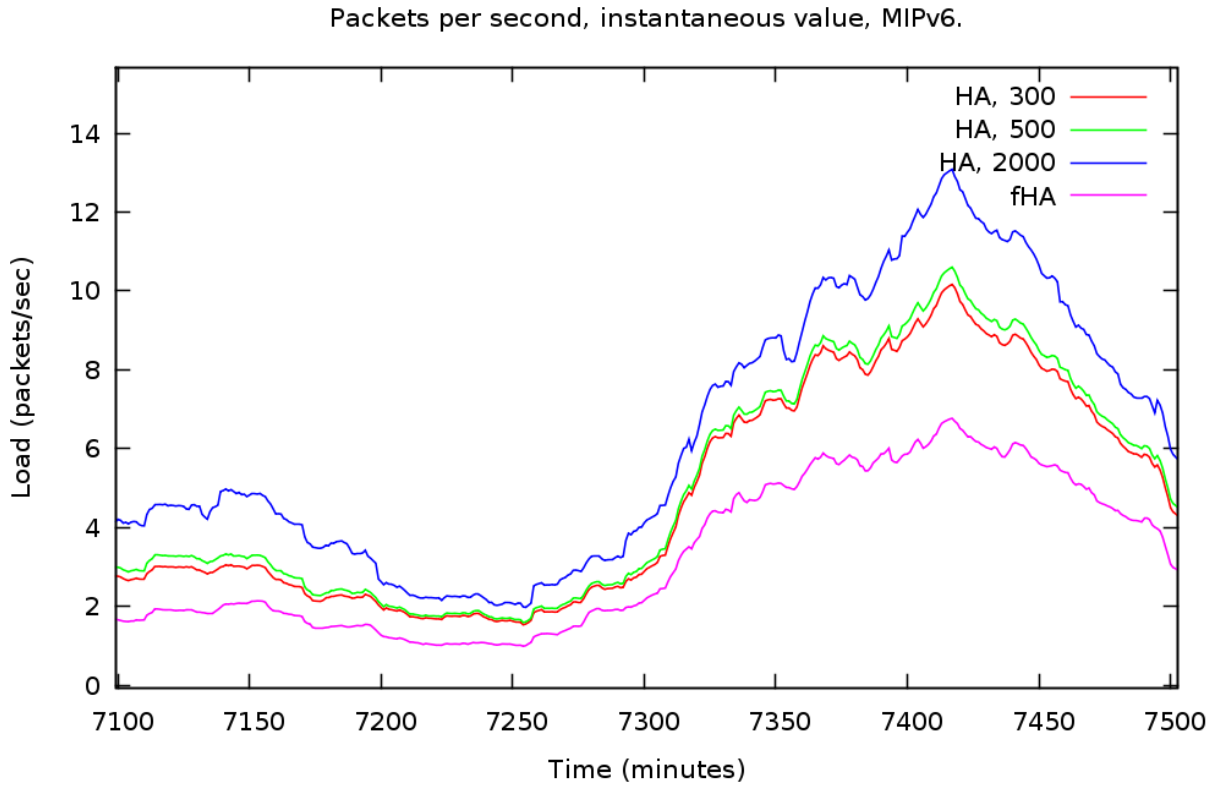

Active flows per second, CDF, MIPv6.
CDF
Active flows/sec
HA, 300
HA, 500
HA, 2000
fHA
0
0.2
0.4
0.6
0.8
1
0
5
10
15
20
25
30
35
40

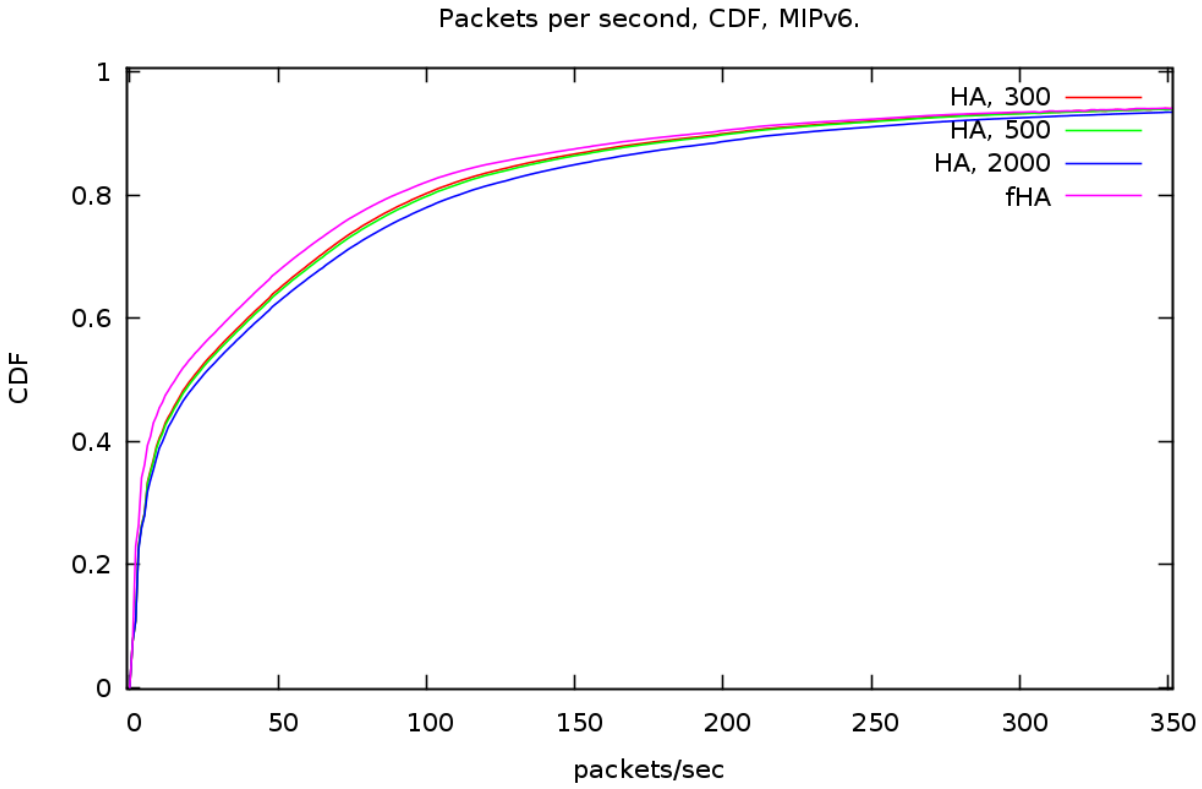

Flow arrivials per second, instantaneous value, MIPv6.
Load (flow arrivials/sec)
Time (minutes)
HA IPv6 300
HA IPv6 500
HA IPv6 2000
fHA IPv6
0
1
2
3
4
5
6
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

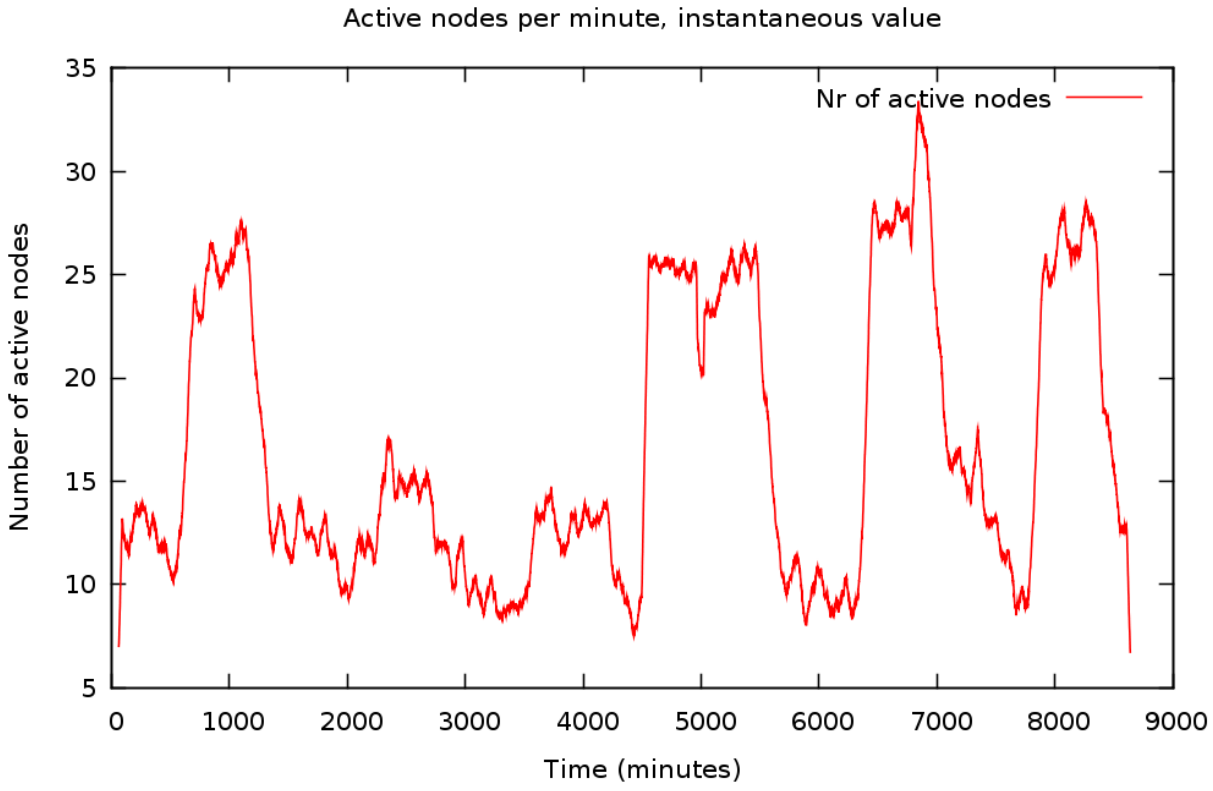

Flow arrivials per second, CDF, MIPv6.
CDF
flow arrivials/sec
HA, 300
HA, 500
HA, 2000
fHA
0
0.2
0.4
0.6
0.8
1
0
5
10
15
20
25


Octets per second, instantaneous value, MIPv6
Load (octets/sec)
Time (minutes)
HA, 300
HA, 500
HA, 2000
fHA
0
500
1000
1500
2000
2500
3000
3500
4000
4500
7100
7150
7200
7250
7300
7350
7400
7450
7500


Octets per second, CDF, MIPv6.
CDF
kilo octets/sec
HA, 300
HA, 500
HA, 2000
fHA
0
0.2
0.4
0.6
0.8
1
0
20
40
60
80
100


Packets per second, instantaneous value, MIPv6.
HA, 300
HA, 500
HA, 2000
fHA
Load (packets/sec)
0
2
4
6
8
10
12
14
7100
7150
7200
7250
7300
7350
7400
7450
7500
Time (minutes)


Packets per second, CDF, MIPv6.
HA, 300
HA, 500
HA, 2000
fHA
CDF
0
0.2
0.4
0.6
0.8
1
0
50
100
150
200
250
300
350
packets/sec


Active nodes per minute, instantaneous value
Nr of active nodes
Number of active nodes
Time (minutes)
35
30
25
20
15
10
5
0
1000
2000
3000
4000
5000
6000
7000
8000
9000

Appendix B

Appdx B: Code snippets

Here you cand find the Perl and Bash code that is the most important for the simulator. Only the main algorithms are presented.

Implementation of filtering tree, part of ha_load_computer.sh. Please note that in order to increase eficiency a large scale storage - processing power compromise is made. If an operation has already been performed the result will be stored and this operation will not be executed again.

```
1  echo filtering the flows twards obtaining 2
2  if [ -f $i"load"/2 ]
3  then
4      echo skipping, 2 already exists!
5  else
6      cat $i | flow-nfilter -f filters.acl -F f_noservers_upc > $i"load"/2
7  fi
8  echo filtering the flows twards obtaining 21
9  if [ -f $i"load"/21 ]
10 then
11     echo skipping, 21 already exists!
12 else
13     cat $i"load"/2 | flow-nfilter -f filters.acl -F f_internal_traffic_upc > $i"load"/21
14 fi
15 echo filtering the flows twards obtaining 22
16 if [ -f $i"load"/22 ]
17 then
18     echo skipping, 22 already exists!
19 else
20     cat $i"load"/2 | flow-nfilter -f filters.acl -F f_non_internal_traffic_upc > $i"load"/22
21 fi
22 echo filtering the flows twards obtaining 1
```

```
    if [ -f $i"load"/1 ]
then
    echo skipping, 1 already exists!
else
    cat $i | flow-nfilter -f filters.acl -F f_servers_upc > $i"load"/1
fi
echo filtering the flows twards obtaining 11
if [ -f $i"load"/11 ]
then
    echo skipping, 11 already exists!
else
    cat $i"load"/1 | flow-nfilter -f filters.acl -F f_internal_traffic_upc > $i"load"/11
fi
echo filtering the flows twards obtaining 112
if [ -f $i"load"/112 ]
then
    echo skipping, 112 already exists!
else
    cat $i"load"/11 | flow-nfilter -f filters.acl -F f_not_only_servers_upc > $i"load"/112
fi
```

flow_aggregator.pl , the script that aggregates flows that follow the same route.

```
#!/usr/bin/perl

##############################################################################################
# synthax:
# perl flow_aggregator.pl inputfile > outputfile
##############################################################################################

# constants
$firstday=8; # the number (day of month) of the day before the first day of the capture

$max_lifetime_s=420; # maximum lifetime of a binding update in seconds. value taken from rfc.

# global variables

my $agg_flows = 0; # total number of aggregate flows in queue.
my $agg_flows_written = 0; #total number of aggregate flows written down.

my @start_a; # altough not all fileds are needed (or even make sense) the format is mentained
my @end_a;   # for compatibility with the rest of the scripts
my @sif_a;
my @sip_a;
```

```
22 my @srcp_a;
23 my @dif_a;
24 my @dip_a;
25 my @dstp_a;
26 my @p_a;
27 my @Fl_a;
28 my @pkts_a;
29 my @octets_a;
30 my @pkts_1;
31 my @octets_1;
32 my @pkts_2;
33 my @octets_2;
34 my @pkts_3;
35 my @octets_3;
36 my $rotime1 = 300; # 3 possible RR durations
37 my $rotime2 = 500;
38 my $rotime3 = 2000;
39
40 $file = $ARGV[0];
41 open (INFO,$file);
42
43 if (scalar(@ARGV) > 1){ # parameters override defaults
44     $max_lifetime_s=$ARGV[1];
45     my rotime1 = $ARGV[2];
46     my rotime2 = $ARGV[3];
47     my rotime3 = $ARGV[4];
48 };
49
50
51 $line = <INFO>;# the first two lines in the file are copyed unmodyfied, they are the title ↩
       lines of the file
52 print "$line";
53 $line = <INFO>;
54 print "$line";
55
56 #foreach(@line=<INFO>)
57 foreach(<INFO>)
58 {
59     @current_line=(split/\s+/);
60     ($start,$end,$sif,$sip,$srcp,$dif,$dip,$dstp,$p,$Fl,$pkts,$octets)=@current_line;
61
62     $existing_flow = &search_agg_flows($sip, $dip); # searches for an aggregate flow that ↩
         maches the source and destination IP address of the current flow
63
64     if ($existing_flow == undef){ # if no flow matching is found, one is created
```

```
65          &create_new_agg_flow($start,$end,$sif,$sip,$srcp,$dif,$dip,$dstp,$p,$Fl,$pkts,$octets);
66      }
67      else{ # if one is found, then it is updated
68          &add_to_agg_flow($existing_flow,$start,$end,$sif,$sip,$srcp,$dif,$dip,$dstp,$p,$Fl,↩
                $pkts,$octets);
69      }
70      # the aggregate flow list is searched for expired flows and the found ones are exported.
71      &write_valid_agg_flows($start);
72
73  }
74  # the input file is closed
75  close INFO;
76
77  #the last aggregated flows are exported also
78  &write_agg_flows;
79
80  sub milliseconds # computes the number of milliseconds passed from the beginning of the day ↩
        defined in $firstday
81  {
82      my ($time)=@_;
83      my $day=substr $time,2,2;
84      my $hour=substr $time,5,2;
85      my $minute=substr $time,8,2;
86      my $sec=substr $time,11,2;
87      my $msec=substr $time,14,3;
88      my $ms=(($day-$firstday)*86400000) + ($hour*3600000) + ($minute*60000) + ($sec*1000) +$msec↩
            ;
89
90      $ms
91  }
92
93  # returns the number of the first (and probably only) aggregate flow that is between the same ↩
        IP adresses
94  sub search_agg_flows # the origin and destination ip address as parameters. since the flow is ↩
        bidirectional the order is irrelevant
95  {
96      my $i;
97      for ($i=0; $i<$agg_flows; $i++)
98      {
99          if( (($_[0] == $sip_a[$i]) && ($_[1] == $dip_a[$i])) || (($_[1] == $sip_a[$i]) && ($_↩
                [0] == $dip_a[$i])) ){
100             return $i;
101         }
102     }
103     return undef;
```

```
104 }
105
106 # creates an aggregate flow with the same data as the parameters.
107 sub create_new_agg_flow # all the flow data in the default order as parameters.
108 {
109
110     #this is the line that actually creates the aggregate flow
111     ($start_a[$agg_flows],$end_a[$agg_flows],$sif_a[$agg_flows],$sip_a[$agg_flows],$srcp_a[↩
            $agg_flows],$dif_a[$agg_flows],$dip_a[$agg_flows],$dstp_a[$agg_flows],$p_a[$agg_flows],↩
            $Fl_a[$agg_flows],$pkts_a[$agg_flows],$octets_a[$agg_flows]) = @_;
112
113     # the following code will aproximte the amount of data that passes through the HA in the ↩
            first 300, 500 and 2000 ms.
114     # 300 ms here
115     if ( milliseconds($start_a[$agg_flows]) + $rotime1 > milliseconds($end_a[$agg_flows]) )
116     {
117         $pkts_1[$agg_flows] = $pkts_a[$agg_flows];
118         $octets_1[$agg_flows] = $octets_a[$agg_flows];
119     }
120     else
121     {
122         $duration_flow = milliseconds($end_a[$agg_flows]) - milliseconds($start_a[$agg_flows]);
123
124         if(($rotime1/$duration_flow) * $pkts_a[$agg_flows] > 1){
125             $pachete = ($rotime1/$duration_flow) * $pkts_a[$agg_flows];
126             $octeti = ($rotime1/$duration_flow) * $octets_a[$agg_flows];
127         }
128         else
129         {
130             $pachete = 1;
131             $octeti = $octets_a[$agg_flows]/$pkts_a[$agg_flows];
132         }
133         $pkts_1[$agg_flows] = $pachete;
134         $octets_1[$agg_flows] = $octeti;
135     }
136
137     #500 ms here
138     if ( milliseconds($start_a[$agg_flows]) + $rotime2 > milliseconds($end_a[$agg_flows]) )
139     {
140         $pkts_2[$agg_flows] = $pkts_a[$agg_flows];
141         $octets_2[$agg_flows] = $octets_a[$agg_flows];
142     }
143     else
144     {
145         $duration_flow = milliseconds($end_a[$agg_flows]) - milliseconds($start_a[$agg_flows]);
```

```
146
147         if(($rotime2/$duration_flow) * $pkts_a[$agg_flows] > 1){
148             $pachete = ($rotime2/$duration_flow) * $pkts_a[$agg_flows];
149             $octeti = ($rotime2/$duration_flow) * $octets_a[$agg_flows];
150         }
151         else
152         {
153             $pachete = 1;
154             $octeti = $octets_a[$agg_flows]/$pkts_a[$agg_flows];
155         }
156         $pkts_2[$agg_flows] = $pachete;
157         $octets_2[$agg_flows] = $octeti;
158     }
159
160     #2000 ms here
161     if ( milliseconds($start_a[$agg_flows]) + $rotime3 > milliseconds($end_a[$agg_flows]) )
162     {
163         $pkts_3[$agg_flows] = $pkts_a[$agg_flows];
164         $octets_3[$agg_flows] = $octets_a[$agg_flows];
165     }
166     else
167     {
168         $duration_flow = milliseconds($end_a[$agg_flows]) - milliseconds($start_a[$agg_flows]);
169
170         if(($rotime3/$duration_flow) * $pkts_a[$agg_flows] > 1){
171             $pachete = ($rotime3/$duration_flow) * $pkts_a[$agg_flows];
172             $octeti = ($rotime3/$duration_flow) * $octets_a[$agg_flows];
173         }
174         else
175         {
176             $pachete = 1;
177             $octeti = $octets_a[$agg_flows]/$pkts_a[$agg_flows];
178         }
179         $pkts_3[$agg_flows] = $pachete;
180         $octets_3[$agg_flows] = $octeti;
181     }
182
183     $agg_flows++;
184
185
186 }
187
188 # updates an already existing aggregate flow.
189 sub add_to_agg_flow # data of the current flow as parameters
190 {
```

```
191     my $agg_flow_nr;
192     my $start;
193     my $end;
194     my $sif;
195     my $sip;
196     my $srcp;
197     my $dif;
198     my $dip;
199     my $dstp;
200     my $p;
201     my $Fl;
202     my $pkts;
203     my $octets;
204
205     ($agg_flow_nr,$start,$end,$sif,$sip,$srcp,$dif,$dip,$dstp,$p,$Fl,$pkts,$octets)=@_;
206
207     #adds the number of octets and packets transmitted to the total number
208     $octets_a[$agg_flow_nr]+=$octets;
209     $pkts_a[$agg_flow_nr]+=$pkts;
210
211     # extends the duration of the aggregate flow to include the current one
212     if( milliseconds($start) < milliseconds($start_a[$agg_flow_nr]) ) {
213         $start_a[$agg_flow_nr] = $start;
214     }
215
216     if( milliseconds($end) > milliseconds($end_a[$agg_flow_nr]) ) {
217         $end_a[$agg_flow_nr] = $end;
218     }
219
220     #computes the volume of data that is transmitted in the first $rotime1 (300 ms)
221     if ( milliseconds($start) < milliseconds($start_a[$agg_flow_nr]) + $rotime1 )
222     {
223         if ( milliseconds($start_a[$agg_flow_nr]) + $rotime1 > milliseconds($end) )
224         {
225             $pkts_1[$agg_flow_nr] += $pkts;
226             $octets_1[$agg_flow_nr] += $octets;
227         }
228         else
229         {
230             my $duration_flow = milliseconds($end) - milliseconds($start);
231             my $useful_time = $start_a[$agg_flow_nr] + $rotmie1 - $start;
232
233             if (($useful_time/$duration_flow) * $pkts > 1){
234                 $pachete = ($useful_time/$duration_flow) * $pkts;
235                 $octeti = ($useful_time/$duration_flow) * $octets;
```

```
            }
            else{
                $pachete = 1;
                $octeti = $octets/$pkts;
            }

            $pkts_1[$agg_flow_nr] += $pachete;
            $octets_1[$agg_flow_nr] += $octeti;
        }
    }

    #computes the volume of data that is transmitted in the first $rotime2 (500 ms)
    if ( milliseconds($start) < milliseconds($start_a[$agg_flow_nr]) + $rotime2 )
    {
        if ( milliseconds($start_a[$agg_flow_nr]) + $rotime2 > milliseconds($end) )
        {
            $pkts_2[$agg_flow_nr] += $pkts;
            $octets_2[$agg_flow_nr] += $octets;
        }
        else
        {
            my $duration_flow = milliseconds($end) - milliseconds($start);
            my $useful_time = $start_a[$agg_flow_nr] + $rotmie2 - $start;

            if (($useful_time/$duration_flow) * $pkts > 1){
                $pachete = ($useful_time/$duration_flow) * $pkts;
                $octeti = ($useful_time/$duration_flow) * $octets;
            }
            else{
                $pachete = 1;
                $octeti = $octets/$pkts;
            }

            $pkts_2[$agg_flow_nr] += $pachete;
            $octets_2[$agg_flow_nr] += $octeti;
        }
    }

    #computes the volume of data that is transmitted in the first $rotime3 (2000 ms)
    if ( milliseconds($start) < milliseconds($start_a[$agg_flow_nr]) + $rotime3 )
    {
        if ( milliseconds($start_a[$agg_flow_nr]) + $rotime3 > milliseconds($end) )
        {
            $pkts_3[$agg_flow_nr] += $pkts;
            $octets_3[$agg_flow_nr] += $octets;
```

```
281         }
282         else
283         {
284             my $duration_flow = milliseconds($end) - milliseconds($start);
285             my $useful_time = $start_a[$agg_flow_nr] + $rotmie3 - $start;
286 
287             if (($useful_time/$duration_flow) * $pkts > 1){
288                 $pachete = ($useful_time/$duration_flow) * $pkts;
289                 $octeti = ($useful_time/$duration_flow) * $octets;
290             }
291             else{
292                 $pachete = 1;
293                 $octeti = $octets/$pkts;
294             }
295 
296             $pkts_3[$agg_flow_nr] += $pachete;
297             $octets_3[$agg_flow_nr] += $octeti;
298         }
299     }
300 
301 }
302 
303 #writes all aggregated flows to output, regardless if expired or not.
304 sub write_agg_flows
305 {
306     my $i;
307     for ($i=0; $i<$agg_flows; $i++){
308         printf("$start_a[$i] $end_a[$i] $sif_a[$i] $sip_a[$i] $srcp_a[$i] $dif_a[$i] $dip_a[$i]↩
              $dstp_a[$i] $p_a[$i] $Fl_a[$i] $pkts_a[$i] $octets_a[$i] $pkts_1[$i] $octets_1[$i]↩
              $pkts_2[$i] $octets_2[$i] $pkts_3[$i] $octets_3[$i] \n");
309     }
310 }
311 
312 # exports the first aggregated flows, until it meets one that is not expired yet.
313 # this method is not very accurate, but accuracy is a necessary sacrifice here because it ↩
      brings a huge performance improvement over the accurate algorithm
314 sub write_valid_agg_flows
315 {
316 
317     my $current_start_time = &milliseconds($_);
318     while( (&milliseconds($start_a[0]) + $max_lifetime_s*1000 < $current_start_time) && (↩
          $agg_flows > 0) )
319     {
320         my $start=shift @start_a;
321         my $end=shift @end_a;
```

```
322         my $sif=shift @sif_a;
323         my $sip=shift @sip_a;
324         my $srcp=shift @srcp_a;
325         my $dif=shift @dif_a;
326         my $dip=shift @dip_a;
327         my $dstp=shift @dstp_a;
328         my $p=shift @p_a;
329         my $Fl=shift @Fl_a;
330         my $pkts=shift @pkts_a;
331         my $octets=shift @octets_a;
332         my $pkts_1=shift @pkts_1;
333         my $octets_1=shift @octets_1;
334         my $pkts_2=shift @pkts_2;
335         my $octets_2=shift @octets_2;
336         my $pkts_3=shift @pkts_3;
337         my $octets_3=shift @octets_3;
338
339         printf("$start $end $sif $sip $srcp $dif $dip $dstp $p $Fl $pkts $octets $pkts_1 ↩
                $octets_1 $pkts_2 $octets_2 $pkts_3 $octets_3\n");
340         $agg_flows_written++;
341
342         $agg_flows--;
343     }
344 }
```

ha_load.pl , the script that can compute the load of the standard HA for IPv4 or IPv6, or the fHA for IPv6.

```
1  #!/usr/bin/perl
2
3  ##############################################################################################
4  # synthax:
5  # perl ha_load.pl inputfile limit > output
6  ##############################################################################################
7
8  # the day (number of day of month) before the first day of captures
9  $firstday=8;
10
11 sub milliseconds
12 {
13     my ($time)=@_;
14     my $day=substr $time,2,2;
15     my $hour=substr $time,5,2;
16     my $minute=substr $time,8,2;
```

```
17      my $sec=substr $time,11,2;
18      my $msec=substr $time,14,3;
19      my $ms=(($day-$firstday)*86400000) + ($hour*3600000) + ($minute*60000) + ($sec*1000) +$msec;
20
21      $ms
22  }
23
24  my @pkts_per_sec;
25  my @octets_per_sec;
26  my @a_flows_per_sec;
27  my @flow_a_per_sec;
28  my @packet_load;
29  my $mean_pkts_per_sec;
30  my $mean_octets_per_sec;
31  my $limit;
32  my $rotime;
33
34  $file = $ARGV[0];
35  open (INFO,$file);
36  $limit = $ARGV[1];  # treshold for RR algorithm
37  $rotime = $ARGV[2]; # time it takes for RR to finish
38
39  $line = <INFO>; # first two title lines ignored
40  $line = <INFO>;
41
42  #foreach(@line=<INFO>)
43  foreach(<INFO>) #all flows in file are read one at a time
44  {
45      @current_line=(split/\s+/); # they are split in fields
46      if (scalar(@current_line) <= 12 )
47      {
48          ($start,$end,$sif,$sip,$srcp,$dif,$dip,$dstp,$p,$Fl,$pkts,$octets)=@current_line;
49      }
50      else
51      {
52          ($start,$end,$sif,$sip,$srcp,$dif,$dip,$dstp,$p,$Fl,$pkts,$octets,$pkts_1,$octets_1,
                $pkts_2,$octets_2,$pkts_3,$octets_3)=@current_line;
53      }
54
55      $start_ms=int(milliseconds($start));
56      $end_ms=int(milliseconds($end));
57
58      $start_s=int(milliseconds($start)/1000);
59      $end_s=int(milliseconds($end)/1000);
```

```
60
61      $flow_duration = $end_s-$start_s+1; # self explainatory naming of variables
62
63      $mean_pkts_per_sec = $pkts/$flow_duration;
64      $mean_octets_per_sec = $octets/$flow_duration;
65      $mean_octets_per_pkt = $octets/$pkts;
66
67
68      if( ($pkts < $limit) || ($limit == -1) ) # if flow is smaller than limit, or limit is ↩
            considered inifinity (for IPv4)
69      {
70          # the load of the entire flow is added to the HA
71          for($i=$start_s; $i<=$end_s; $i++)
72          {
73              $pkts_per_sec[$i]+=$mean_pkts_per_sec;
74              $octets_per_sec[$i]+=$mean_octets_per_sec;
75              $a_flows_per_sec[$i]+=1;
76          }
77
78          $flow_a_per_sec[$start_s]+=1;
79      }
80      else
81      {
82
83          $octets_per_sec[$start_s + 1] += 120; # signaling load of one RR
84          $pkts_per_sec[$start_s + 1] += 2;
85
86          # an accurate computation of the load that passes before and during the performing of ↩
                RR
87          # computes and adds the packets/sec octets/sec, and active flows/sec
88          if(((($rotime/1000)+1)<$flow_duration)
89          {
90              $min_time = $rotime/1000;
91              $a_flows_per_sec[$start_s] += $min_time; #flow; pun tot pe o secunda ca $min_time ↩
                    nu o sa fie valoare mai mare de vreo 2 niciodata si dupa aia sasa fac media pe ↩
                    60 secunde cu averager.pl
92
93              if ($rotime == 300)
94              {
95                  $octets_per_sec[$start_s + 1] += $octets_1; #flow
96                  $pkts_per_sec[$start_s + 1] += $pkts_1; #flow
97              }
98              if ($rotime == 500)
99              {
100                 $octets_per_sec[$start_s + 1] += $octets_2; #flow
```

```
101                 $pkts_per_sec[$start_s + 1] += $pkts_2; #flow
102             }
103             if ($rotime == 2000)
104             {
105                 $octets_per_sec[$start_s + 1] += $octets_3; #flow
106                 $pkts_per_sec[$start_s + 1] += $pkts_3; #flow
107             }
108         }
109         else
110         {
111             $min_octets = $octets;
112             $min_pkts = $pkts;
113             $min_time = $flow_duration;
114
115             $a_flows_per_sec[$start_s] += $min_time; #flow; pun tot pe o secunda ca $min_time ↩
                    nu o sa fie valoare mai mare de vreo 2 niciodata si dupa aia sasa fac media pe ↩
                    60 secunde cu averager.pl
116             $octets_per_sec[$start_s + 1] += $min_octets; #flow
117             $pkts_per_sec[$start_s + 1] += $min_pkts; #flow
118         }
119
120         # finally one flow arrival is added
121         $flow_a_per_sec[$start_s] += 1; #flow
122
123
124     }
125 }
126
127 # the computed load is sent to the output
128 my $total_seconds = scalar(@pkts_per_sec);
129 for($i=1; $i<$total_seconds; $i++)
130 {
131     if($pkts_per_sec[$i] || $octets_per_sec[$i] || $a_flows_per_sec[$i] || $flow_a_per_sec[$i] ↩
            )
132     {
133             printf("%d %f %f %f %d\n", $i, $pkts_per_sec[$i], $octets_per_sec[$i], ↩
                    $a_flows_per_sec[$i], $flow_a_per_sec[$i]);
134     }
135 }
136
137 close INFO;
```

fha_load.pl , the script that computes the load datorated to RR signaling on the fHA IPv6

```
#!/usr/bin/perl

##############################################################################################
# synthax:
# perl fha_load.pl inputfile limit
##############################################################################################

# the day (number of day of month) before the first day of captures
$firstday=8;

sub milliseconds
{
    my ($time)=@_;
    my $day=substr $time,2,2;
    my $hour=substr $time,5,2;
    my $minute=substr $time,8,2;
    my $sec=substr $time,11,2;
    my $msec=substr $time,14,3;
    my $ms=(($day-$firstday)*86400000) + ($hour*3600000) + ($minute*60000) + ($sec*1000) +$msec;

    $ms
}

my @pkts_per_sec;
my @octets_per_sec;
my @a_flows_per_sec;
my @flow_a_per_sec;
my @packet_load;
my $mean_pkts_per_sec;
my $mean_octets_per_sec;
my $limit;

$file = $ARGV[0];
open (INFO,$file);
$limit = $ARGV[1];

$line = <INFO>;# first two title lines ignored
$line = <INFO>;

foreach(<INFO>)#all flows in file are read one at a time
{
    @current_line=(split/\s+/);# they are split in fields
    if (scalar(@current_line) <= 12 )
    {
```

```
45          ($start,$end,$sif,$sip,$srcp,$dif,$dip,$dstp,$p,$Fl,$pkts,$octets)=@current_line;
46      }
47      else
48      {
49          ($start,$end,$sif,$sip,$srcp,$dif,$dip,$dstp,$p,$Fl,$pkts,$octets,$pkts_1,$octets_1,↩
                $pkts_2,$octets_2,$pkts_3,$octets_3)=@current_line;
50      }
51
52      if( $pkts >= $limit ) # if packets greater than limit then RR is performed. the only ↩
            traffic that affects the fHA is the signaling.
53      {
54          $octets_per_sec[$start_s + 1] += 120; # signaling octets
55          $pkts_per_sec[$start_s + 1] += 2; #signaling packets
56      }
57
58  }
59
60  # the computed load is sent to the output
61  my $total_seconds = scalar(@pkts_per_sec);
62  for($i=1; $i<$total_seconds; $i++)
63  {
64      if($pkts_per_sec[$i] || $octets_per_sec[$i] || $a_flows_per_sec[$i] || $flow_a_per_sec[$i] ↩
            )
65      {
66              printf("%d %f %f %d %d\n", $i, $pkts_per_sec[$i], $octets_per_sec[$i], ↩
                    $a_flows_per_sec[$i], $flow_a_per_sec[$i]);
67      }
68  }
69
70  close INFO;
```