Examensarbete

HÖGSKOLAN I SKÖVDE

EFFEKTERNA AV BRANDVÄGGSREGLER FÖR FREEBSD PF & IPTABLES.

THE IMPACT OF FIREWALL RULESETS FOR FREEBSD PF & IPTABLES.

Examensarbete i informationsteknologi med inriktning mot nätverks- och systemadministration

Grundnivå 22,5 Högskolepoäng

Vårtermin år 2018

Andreas Polnäs

Handledare: Joakim Kävrestad

Examinator: Marcus Nohlberg

Innehållsförteckning

Sammanfattning

Paketfiltrering är en av nyckelfunktionerna i de flesta av dagens brandväggar, vilket gör paketfiltrering till en viktig del av det dagliga arbetet för många systemadministratörer. Sedan uppkomsten av paketfiltrering har nätverkskomplexiteten ökat drastiskt, Många av dagens tjänster har behov av olika protokoll för att kommunicera. I kombination med detta måste brandväggen bearbeta en större mängd data än tidigare för att tillgodose dagens nätverkstopologier.

Denna studie syftar till att undersöka om det finns någon skillnad i prestanda mellan två moderna iterationer av de populära UNIX-brandväggarna IPtables och FreeBSD PF. Detta sker genom att de två brandväggarna utsätts för olika antal regler, samtidigt som de genomströmmas av olika stora paketflöden.

De båda brandväggarna kommer att jämföras baserat på tre attribut, CPU, genomströmning och latens. tre olika bandbredder testas. 100, 500 och 1000Mbit/s. Testet omfattar längre tester som upprepas flera gånger för att öka studiens giltighet. Testerna som utförs görs på ursprungliga operativsystemet för varje brandvägg. Linux Ubuntu 16 för IPtables och FreeBSD 11 för FreeBSD PF.

Studien kom fram till att brandväggarnas prestanda är likvärdiga i genomströmning och latens vid lägre regelmängder. Vid högre regelmängder skiljer sig prestandan och PF är bättre anpassad för stora regeluppsättningar. IPtables anses vara den bättre brandväggen för låga regeluppsättningar på grund av dess låga CPU-användning.

Nyckelord: Brandvägg, FreeBSD PF, IPtables, Ubuntu 16, FreeBSD 11, Dual-homed network, CPU, latens, genomströmning, packetfiltreringsregler.

Abstract

Packetfiltering is one of the key features in most of today's firewalls. With many packetfilters being used daily in a system administrator's work. Over the years since founding of the packetfilter technology the complexity of the network has increased drastically, where many of today's services relies on different protocols to communicate, combined with a much larger amount of data that the firewall must process to satisfy todays network topologies.

This study aims to explore if there is any difference in performance between two modern iterations of popular UNIX firewalls, IPtables and FreeBSD PF. By submitting them to different number of rulesets while at the same testing them under a series of different packet flows through the firewall.

Both firewalls will be compared based on three attributes, CPU, throughput and latency, and three different bandwidths will be tested. 100, 500 and 1000Mbits/s. The test include longer tests that is repeated multiple times to increase the validity of the study. The tests were performed on the native operating system of each firewall. Linux Ubuntu 16 for IPtables and FreeBSD 11 for FreeBSD PF.

The study concluded that the performance of the firewalls is equal in throughput and latency at lower volumes. At higher amounts of rulesets, performance is different between the firewalls and PF is considered better for large rules, while IPtables are considered to be a better firewall for low rulesets due to its low CPU usage.

Keywords: Firewall, FreeBSD PF, IPtables, Ubuntu 16, FreeBSD 11, Dual-homed network, CPU, Latency, Throughput, packetfilter rules.

1 Introduktion

Brandväggar används oftast för att försvara ett internt nätverk från obehörig extern åtkomst. För att kunna säkra det interna nätverket effektivt brukar brandväggen placeras vid ingången till nätverket. Detta innebär att brandväggen får ett stort ansvar då majoriteten av all trafik som ska in och ut kommer passera genom den. Om en brandvägg inte kan prestera tillräckligt i genomströmning och i hantering av data kan det innebära kraftiga negativa inverkningar på nätverket och en risk för att tjänster i nätverket störs.

Dagens nätverkstopologier är mer avancerade än tidigare och har många separata tjänster som behöver olika protokoll tillgängliga för att fungera korrekt. Detta medför utmaningar gällande de antal regler som behöver tillsättas i dagens brandväggar för att kunna bibehålla nätverkets funktionalitet. Denna studies mål är att undersöka skillnaderna mellan två existerande brandväggslösningar genom att utföra en serie tester med olika mängder regeluppsättningar och bandbredd.

2 Bakgrund

Denna del innehåller en överblick av vad som har gjorts inom området och som kan kopplas till denna studie. I bakgrunden definieras och förklaras koncept och begrepp som återkommer senare i studien.

2.1 Vad är en brandvägg?

En brandvägg är ett säkerhetssystem för nätverk, som övervakar och kontrollerar inkommande och utgående trafik baserat på brandväggens fördefinierade säkerhetsregler (Oppliger, R. 1997). En traditionell brandvägg brukar vara en dedikerad maskin som fungerar som en barriär mellan ett internt nätverk och ett externt nätverk (t.ex. internet). En brandvägg som använder sig av paketfiltrering undersöker headern på paket som vill ta sig mellan det interna nätverket och det externa nätverket. Värdena i paketets header är det som avgör om den får komma igenom brandväggen eller inte. Det finns fem fält i headern som paketfiltret kontrollerar, ursprungs IP-address, destinations IP-adress, portnummer för ursprung, destination, samt vilken typ av protokoll paketet använder (Comerford, P., Davies, J. N. & Grout, V. 2016; Liu, A. X. & Gouda, M. G. 2009). Om värdena stämmer med de fördefinierade reglerna i brandväggen så accepteras paketet och får komma igenom till andra sidan. Om värdena inte stämmer med de fördefinierade reglerna i brandväggen kommer paketet inte att komma igenom (Acharya, H. B. & Gouda, M. G. 2009). Alla regler som lagts till i brandväggen kontrolleras i den ordning de ligger i paketfiltret för varje paket som når brandväggen. (Errin W. Fulp 2005).

Det finns olika sorters brandväggar. Två av de vanligaste typerna av nätverksbaserade brandväggar är tillståndsbaserade brandväggar *(eng stateful firewall)* och tillståndslösa brandväggar *(eng stateless firewall)*. Tillståndsbaserade brandväggar har möjligheten att utläsa TCP-sessioner. De kan utföra detta genom att undersöka tidigare paket som har kommit ifrån samma klient. En extern klient kan till exempel nekas tillgång till det interna nätverket av en tillståndsbaserad brandvägg, Om den inte fått TCP-trafik från den interna klienten innan dess (Gouda, M. G. and Liu, A. X. 2005). Denna metod går inte att använda för protokoll som inte har en session, till exempel UDP-baserad trafik. Tillståndslösa brandväggar kan inte läsa av aktiva anslutningar utan måste använda sig utav paketfiltrering för att avgöra vad som får komma in och ut ur nätverket.

Något som används aktivt inom brandväggar och nätverk är maximum transmission unit (MTU). MTU är den största mängd nätverkslagerdata som kan gå igenom i en nätverkstransaktion. (Information Sciences Institute, 1981). MTU brukar gå att justera hos olika nätverksgränssnitt och är vanligt att justera hos en brandvägg för att matcha trafikbehovet. En högre MTU ger en större genomströmning eftersom varje nätverkspaket kan ta med mer data. Detta gör att en större mängd data kan skickas. Högre MTU kommer dock med potentiella nackdelar som till exempel, ökad latens och paketförlust. Vid större paket kan latensen påverkas, då större paket tar längre tid att kontrollera och därmed tar längre tid ta sig igenom nätverket. Även paketförlust kan bli ett problem vid högre MTU, eftersom det bara krävs en korrupt bit för att hela paketet ska behövas skickas om.

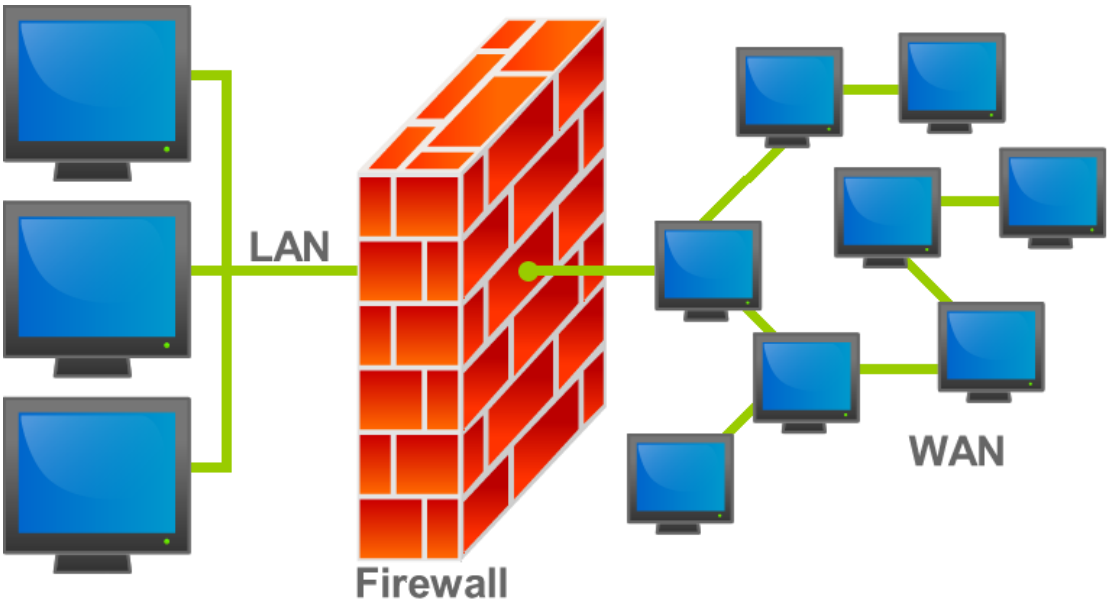


Bild 2.2 The purpose of a firewall (Bruno Pedrozo, 2007)

2.2 Genomströmning, latens och CPU

Nätverksbrandväggar placeras i regel som ett filter mellan nätverket och omvärlden (se bild 2.2). Då samtliga paket till och från nätverket ska hanteras av brandväggen, så är brandväggens prestanda viktig. För att en brandvägg ska vara effektiv måste den kunna prestera inom följande områden, genomströmning, latens och paketförlust.

Genomströmning är den mängd data som kan komma igenom brandväggen per tidsenhet. En anledning till låg genomströmning kan vara den hårdvara brandväggen baseras på. De två primära hårdvarurelaterade orsakerna som kan sänka totala genomströmningen är nätverkskortet och processorn, vilka riskerar att bli flaskhalsar för brandväggen (PFsense 2017a). Skulle ett nätverkskort som inte klarar av trafiken användas, kommer genomströmningen enbart vara så hög som nätverkskortet klarar av. När trafik ankommer till brandväggen så måste paketet bearbetas och detta kräver CPU-kraft. Klarar inte processorn av att hantera all trafik så kan detta medföra låg genomströmning, vilket kan medföra en hög latens och paketförlust. Ett exempel på detta kan vara att en hemsida inte helt laddas för att all information inte kommer fram till slutanvändaren.

Latens är den tiden det tar från att paketet skickas, till dess att paketet når sin slutdestination. Två metoder för att mäta ett pakets latens är med round-trip time(RTT) och one-way delay (OWD). Med RTT mäts tiden det tar för ett paket att skickas från källa till destination, kombinerat tiden det tar att få en bekräftelse på att paketet har kommit fram. Med OWD så mäts enbart tiden från källa till destination (G. Almes, S. Kalidindi, M. Zekauskas. 1999).

En brandvägg vill uppnå låg latens för att ha så liten inverkan på paketets färdtid som möjligt. En orsak till hög latens på en brandvägg kan vara på grund av låg genomströmning eller paketförlust. Kan trafiken inte direkt kontrolleras av brandväggen så kommer den få vänta i brandväggens kö, där inväntar paketet inspektion av brandväggen. Detta skapar latens. Om brandväggens kö blir så kan paketförlust ske. Paketet slängs då bort och måste skickas om. Detta ökar tiden tills rätt paketet når slutmålet. En hög latens kan medföra att tjänster slutar fungera korrekt eller visar fel information.

Processorn är en flaskhals hos brandväggar som kan leda till många av de ovannämnda prestandadegraderingarna. Det är då viktigt att ha en brandvägg som är resurseffektiv i sin hantering av paket. Idag är det vanligt att en server har ett flertal 10Gbps nätverkskort, och nätverkskort på 40Gbps har blivit vanligare. Men CPU-prestandan har inte växt lika snabbt som nätverkskortens hastigheter. (Zilberman et al. 2015). År 2017 så hittades Spectre- och Meltdownbuggen. Dessa buggar gav oprivilegierade användare tillgång till olika processer och CPU-kärnans minne. Vid uppdatering av dessa buggar så minskades CPU-prestandan (Kocher et al. 2018). Detta medför att processorn är en större flaskhals för brandväggar nu än tidigare.

Den generella prestandan hos en brandvägg påverkar inte enbart tjänster och användare. Säkerheten i ett nätverk blir också påverkat av en låg prestanda hos en brandvägg. Om en brandvägg inte har tillräckligt hög prestanda att försvara sig mot distributed denial of service (DDoS) attacker så kan hela det skyddade nätverkets säkerhet bakom brandväggen också vara äventyrat (Salah, K., Elbadawi, K. and Boutaba, R. 2012).

2.3 Regler i brandväggar

Att använda regler i brandväggar är en av de vanligaste metoderna att försvara ett nätverk på, men att använda paketfiltreringsregler för att kontrollera all trafik kan resultera i komplexa nätverk där enskilda brandväggar kan ha stora mängder regler inlagda i paketfiltret (Jeffrey, A. & Samak, T. 2009).

När fler regler adderas till paketfiltret degraderas prestandan i brandväggen (Tihomir K & Predrag P, 2007; Yoon, M., Chen, S. & Zhang, Z. 2010). Detta utgör ett problem då både latens, genomströmning och CPU-användning blir påverkade av utökade regeluppsättningar. Många av de tjänster som används i dagens IT-topologier påverkas av prestandadegradering. Ett exempel på detta är voice over IP(VoIP) som används inom IP-telefoni där hög latens kan bli ett problem eftersom VoIP är realtidsbaserat. Om latensen är för hög medför det att kvalitén på samtalet försämras drastiskt (Sulaiman, N., Carrasco, R. & Chester, G. 2007).

2.4 IPtables och FreeBSD PF

IPtables och FreeBSD PF är två populära open-sourcebrandväggar som är gjorda till två separata unix-baserade operativsystem. Dessa används idag runtom i världen i olika miljöer. Många andra brandväggar som idag finns ute på marknaden använder hela eller delar utav kodbaserna som dessa två brandväggar baseras på. Ett exempel på detta är PFsense, som är baserat på PF och operativsystemet FreeBSD (PFsense, 2017b). Det finns skillnader i hur de två brandväggarna arbetar med regeluppsättningar, vilket gör de unika när det kommer till paketfiltrering.

Med PF går paketet igenom pf.conf filen. Om ett paket matchar en regel fortsätter PF ändå att bearbeta paketet hela vägen igenom konfigurationsfilen. Endast om en regel innehåller "snabb”-alternativet stoppar PF behandlingen och vidtar åtgärder innan den når slutet på regeluppsättningen. Om ett paket kommer till slutet av konfigurationsfilen, är det den sista åtgärden som anges från en regel som matchade paketet som utförs (OpenBSD PF 2017).

När IPtables används behandlas paket av olika kedjor i olika ordning, beroende på källa och paketets destination. Det finns fem stycken kedjor. INPUT, FORWARD, OUTPUT, PREROUTING, POSTROUTING. Ett exempel på en kedja är OUPUT-kedjan, som bearbetar

utgående paket i brandväggen. Olika regler inom en kedja kan medföra att bearbetningen hoppar över till en annan uppsättning regler i en egendefinierad kedja, eller så vidtas åtgärder på ett paket direkt med den nuvarande kedjan. När ett paket väl matchar en regelbeskrivning stoppas behandlingen i den kedjan omedelbart och åtgärden utförs (Netfilter 2017).

2.5 Tidigare studier

Hartmeier (2002) skriver om det då nysläppta OpenBSD PF till OpenBSD operativsystemet. Där förklaras designen av paketfiltret och tester och jämförelser görs med IPfilter, IPtables och PF. I testerna jämförs genomströmning och latens med 100 regler tillagda i paketfiltren. I slutresultaten visas att IPtables har bättre latens och genomströmning än IPfilter och PF.

Salah, K. et al (2012) skapade en ny analytisk kö-modell för regleruppsättningar i IPtables. Denna modell användes för att analysera prestanda hos regelbaserade brandväggar när de får normal trafik, samt när de utsätts för DDoS attack. Studien tittade på tre stycken attribut hos en brandvägg. genomströmning, latens och brandväggens CPU-användning. Studien kom fram till att DDoS attacker som fokuserar på att angripa regler längre ner i en brandväggsregeluppsättning kan försämra prestandan kraftigt hos en brandvägg. Studien rekommenderade att minska regeluppsättningarnas mängd om det är möjligt eller att använda dynamisk regeluppsättning så att kalkylerade DDoS attacker inte kan använda sig av en attack som riktar sig mot regler.

3 Problemformulering

Denna del kommer innehålla en frågeställning som ska besvaras, vilka mål som kommer sättas upp för att svara på frågeställningen och avgränsningar i studien samt vad denna studie medför till forskningsområdet.

3.1 Frågeställningar

Studien utgår ifrån följande frågeställning;

”Hur skiljer sig prestandan hos FreeBSD PF och IPtables vid olika mängder paketfiltreringsregler?”

Målet med studien är att skapa en miljö som kan producera ett resultat för att besvara denna fråga. För att hjälpa till att besvara frågeställningen har följande delmål skapats:

-Hur skiljer sig genomströmningen mellan FreeBSD PF och IPtables vid ökad mängd paketfiltreringsregler?

-Hur skiljer sig latensen mellan FreeBSD PF och IPtables vid ökad mängd paketfiltreringsregler?

-Hur skiljer sig CPU-användningen mellan FreeBSD PF och IPtables vid ökad mängd paketfiltreringsregler?

Dessa delmål kommer hjälpa till att förklara om det finns några skillnader i prestanda mellan de två brandväggarna. För att besvara dessa frågor skapades en fysisk experimentmiljö att utföra tester i.

3.2 Syfte

Om en brandvägg har dålig paketfiltreringsprestanda finns en risk för att detta kan komma att påverka klienter och servrar i nätverket. I dagens komplexa nätverksmiljöer behövs större mängder brandväggsregler för att hålla nätverket säkert och fungerande. Syftet med denna studie är att jämföra två stycken populära brandväggar inom operativsystemen Ubuntu och FreeBSD samt undersöka hur brandväggarna påverkas av olika mängder regeluppsättningar, för att sedan tillsätta jämförelsen i ett verklighetsbaserat scenario.

Inom forskningen jobbas det aktivt med att försöka optimera och hitta lösningar för att underlätta paketfiltrering hos brandväggar. Denna studie fokuserar på att jämföra redan existerande brandväggslösningar och se hur de skiljer sig i avseende till prestanda. Det finns liknande studier inom det valda ämnet men många utav studierna är gamla och inte uppdaterade för dagens trafik, operativsystem och topologier. Att kunna ge resultat som är relevanta för nutidens brandväggar är motivationen för att göra denna studie.

3.3 Avgränsningar

Studien ifråga fokuserar enbart på brandväggarna IPtables och PF. Det finns olika typer av trafik som kan användas i experimentet. I denna studie så fokuseras enbart på att skicka lager 3 trafik igenom brandväggarna. Dessa två val görs på grund av att studiens omfattning annars hade blivit större tidsramen tillåter. Ingen etisk diskussion kommer tas upp i studien eftersom inga etiska aspekter identifierades under experimentets planering.

4 Metod

De valda metoderna för denna studie är beskrivna nedan samt med steg för att utföra studien.

4.1 Steg för utförande av studien

För att utföra denna studie och svara på delmålen till frågeställningen måste ett antal steg utföras för att resultatet ska bli pålitligt. Experimentet kommer till stor del att följa processerna som finns tillgängliga i Wohlin, C. et al. (2012), kombinerat med metodik och topologi från RFC3511 för att utföra tester på brandväggar (Hickman, B., Newman, D., Tadjudin, S. & Martin, T. 2003) och RFC1944 för att testning nätverksanslutningsenheter (S. Bradner, J. McQuaid 1996). Studien kommer att utföras med hjälp av dessa punkter:

1. **Samla information om ämnet** – Samla information om brandväggarna, regeluppsättningar och om tidigare arbeten. Detta kan gälla allt från data till andra studier som kan hjälpa bredda kunskapen inom ämnet
2. **Designa experimentet** – Nästa steg blir att designa experimentet. Då ses val av topologi över, hur implementation ska utföras, vad som ska mätas och hur eventuella validitetshot förhindras.
3. **Utföra experimentet** – I detta steg kommer experimentet utföras och all den data som behövs för att besvara studiens frågeställningar kommer att samlas in.
4. **Analysera informationen** – I det slutgiltiga steget kommer den data som samlats in under experimentet att analyseras med hjälp av applicerbara statistikmetoder och resultatet kommer sammanfattas och diskuteras.

4.2 Val av metod

Ett praktiskt experiment ska utföras för att frågeställningen ska kunna besvaras på bästa sätt. I sin bok experimentation in software engineering skriver Wohlin et al. (2012) att fem processer behöver tas i aktning för att göra ett experiment. De fem processerna visas i punktlistan under:

1. Omfattning

2. Planering

3. Utförande

4. Analys & tolkning

5. Presentation och paketering

Dessa processer kommer inkluderas i studien för att öka dess validitet samt för att ha ett ramverk för experiment att jobba med. Denna del kommer även gå i samman med punkt ett och delvis punkt två i kapitel 3.3 och för utförande av studien.

4.3 Omfattning

Under omfattningsprocessen så är grunden i ett experiment formulerat. Denna del handlar om varför ett experiment ska utföras. Om omfattningen av experimentet inte är tillräcklig finns det risk att arbetet behöver omarbetas eller att det inte kan användas alls för det som ska studeras. Målet med omfattningsdelen är att definiera målet med ett experiment. Genom att använda en mall gjord av Wohlin et al. (2012) skapas en grund till denna studie. I bilaga A så finns mallen och en förklaring till varje del. Såhär blev resultatet av mallen efter allt fyllts i med denna studies parametrar:

Analysera **de valda brandväggarna**

Med ändamålet av att **jämföra brandväggarnas prestanda**

Med avseende på deras **effektivitet vid olika regeluppsättningar**

Från synvinkel av **forskaren**

I kontexten av **NSA-labbet på Högskolan i Skövde**

4.4 Planering

Vid Planeringsprocessen ska experimentet planeras. Här definieras förberedelsen till experimentet och hur det ska utföras. Detta görs för att kunna ha en kontrollerad miljö i experimentet. Vid planeringsprocessen finns fem steg som visas i punktlistan under.

1. Val av kontext
2. Val av variabler
3. Experimentdesign
4. Instrumentation
5. Validitetsutvärdering

Dessa steg kommer att gå igenom de delar som behövs för att planera ett experiment. Många av dessa delar hör samman med steg för utförande av studien. Denna process specifikt hör samman med att designa experimentet, vilket lägger grund till vad som ska undersökas. studien kommer gå igenom stegen i ordningen som punktlistan ovan visar.

4.4.1 Kontext

För att kunna producera de mest generella resultat från ett experiment krävs stora mjukvaruprojekt med professionell personal som hanterar projektet. Detta är inte alltid möjligt och då behövs det göras ett val för i vilken kontext experimentet ska förhålla sig till. Enligt Wohlin finns tre stycken delar att kolla emot, vilket finns i nummerlistan under med även ett svar under varje punkt för denna studies kontext.

1. I vilken miljö utförs experimentet? Produktionsmiljö eller laborationsmiljö?
- Laborationsmiljö
2. Vilka personer utför experimentet? Professionella eller studenter?
- En student kommer ta hand om experimentet
3. Är experimentet giltigt i specifik kontext eller kan det appliceras generellt för hela forskarområdet?
- Den är giltig i en specifik kontext.

Här under kommer varje punkt förklaras och motivationen till varför kontexten av arbetet är utformat som den är.
1. Detta experimentet är tidsbegränsat och att leta upp en aktiv miljö som kan skicka stor mängd riktig trafik ansågs vara för tidskrävande för tidsspannet som finns.
2. Att hyra professionella att utföra experimentet var inte ett val för ingen budget finns för att hyra personal. Experimentet styrs och utförs av författaren till detta examensarbete.
3. Den är giltig i en specifik kontext, en generell slutsats kräver att testa på en stor mängd miljöer och inställningar. Detta är inte möjligt att utföra på grund av arbetets tidsram på en termin.

4.4.2 Val av variabler

I denna del bestäms vilka variabler som finns i experimentet och vad som ska mätas. Det finns oberoende och beroende variabler. En oberoende variabel går att kontrollera och ändra i experimentet, de oberoende variabler ska ha någon effekt på de beroende variablerna. De beroende variablerna är de variabler i experimentet som det mäts effekten på från de oberoende variablerna. Dessa mätredskap behöver valideras och testas innan för det kan ha en påverkan på slutresultatet baserat på vilka oberoende som används.

I denna studie har paketfiltreringsregler och paketström identifierats som oberoende variabler och FreeBSD PF och IPtables har identifierats som beroende variabler. För att se effekten hos de beroende variablerna har denna studie tagit inspiration från en tidigare studie om brandväggar som mätte CPU-användning, genomströmning och latens. (Salah, K., Elbadawi, K. & Boutaba, R. 2012).

4.4.3 Experimentdesign

Denna del fokuserar på att identifiera vilken design experimentet har för att hitta rätt sätt att utföra den statistiska analysen på. Det finns olika designtyper som kan användas, beroende på hur många objekt som spelar roll i experimentet och hur många åtgärder som ska appliceras i experimentet. Objekten i denna studie är brandväggarna och de valda åtgärder som kommer användas är regeluppsättningarna i brandväggarna och olika paketmängder som kommer skickas igenom brandväggarna. Mätning på brandväggarna kommer även göras utan någon behandling, detta för att ha opåverkade värden som grund för att sedan kunna se effekten av behandlingarna.

4.4.4 Instrument

I planeringen för ett experiment väljs instrumenten till studien. Instrumentation delas upp i tre typer, experimentobjekt, riktlinjer och mätningsinstrument. Det generella målet med instrumentation är att hitta redskapen för att utföra experimentet och kunna övervaka det. Detta kommer påverka effekten av experimentet och om det är dåligt designat kan påverka experimentet negativt.

Experimentobjekt är det som ska undersökas. Det är bra att veta bakgrundsfakta och vad som kan vara viktigt att vara uppmärksam på i det experiment som ska utföras. I detta fall är experimentobjektet brandväggar. För att få en bättre insikt i vad som behöver undersökas läses två request for comments(RFC). Detta är dokument som beskriver en föreslagen standard inom ett område. De två som används är RFC3511, för att utföra tester på brandväggar (Hickman, et al 2003), och RFC1944, för testning nätverksanslutningsenheter (S. Bradner, J. McQuaid 1996). Utifrån dessa identifieras vilka parametrar som ska testas när brandväggarna jämförs.

Mätinstrumenten är de som används när datainsamling ska göras till experimentet. Mätinstrumenten för detta experiment kommer vara iperf3 och Ping som kommer generera och logga trafiken som går igenom brandväggen. Iperf3 är ett nätverksmätningsredskap med ett stort antal alternativ i programmet för att mäta och jämföra trafik. Ping är ett kommando som används för att mäta latens med hjälp av RTT. Med dessa mätinstrument ska latens och genomströmning av trafiken mätas. Dessa valdes eftersom de är effektiva mätredskap som anses passa för detta experiment. För CPU så kommer Top användas, Top är ett kommando

som både FreeBSD och Ubuntu har tillgängligt och som fungerar på samma sätt för båda operativsystemen.

Riktlinjer är vad som behövs för att guida experimentets genomförande. Riktlinjer kan innehålla checklistor eller processbeskrivningar. Riktlinjerna för detta experiment är att fem regeluppsättningar med tre olika paketstorlekar kommer att testas (S. Bradner, J. McQuaid 1996). Varje regeluppsättning kommer testas med varje paketmängd i 900 sekunder (15 minuter) och kommer återupprepas 2 gånger i följd (Karrer, R. P., Botta, A. and Pescapé, A. 2008). Vanlig trafik kommer normalt sätt inte via ett flöde, därför skapas fem separata flöden med trafik. Efter all data är insamlad så kommer informationen formateras, för att sedan analyseras.

4.4.5 Validitetsutvärdering

Enligt Wohlin, C. et al. (2012) definieras validiteten hos en studie av resultatets trovärdighet, validiteten skildrar i vilken utsträckning resultaten är sanningsenliga. Det finns fyra stycken kriterier som behövs uppfyllas för att få högre tillförlitlighet. Dessa är följande: begreppsvaliditet, internvaliditet, externvaliditet, Pålitlighet

Begreppsvaliditet är att de åtgärder och begrepp som görs i studien representerar vad som ska undersökas i frågeställningen. Ett problem med detta kan vara om delar av studien kan tolkas på olika sätt. Det är ett hot för validiteten. För att förhindra detta kommer de termer som tas upp i denna studie att förklaras.

Internvaliditet är om det under experimentets gång finns en okänd faktor som påverkar resultatet. Detta kan göra att felaktiga slutsatser görs på grund av att laborationsmiljön inte varit säkrad. Ett exempel på detta är när ett test som ska prova WIFI-styrka men har ett elskåp över testområdet som kan påverka resultatet. Mer hur dessa hot har tagits hand om finns i bilaga B.

Externvaliditet innebär om den metod och resultat som används och det resultat som uppnåtts kan jämföras med liknande undersökningar. Detta vill uppnås för att undvika metoder som inte går att använda under generella omständigheter. För att hantera detta så kommer alla mätinstrument och redskap i studien förklaras, samt vilka kommandon som kommer användas i studien. För att uppnå den externa validiteten så har alla mätinstrument i denna studie använts i liknande studier innan. Även de beroende variabler som kommer studeras under experimentet för att svara på frågeställningen har jämförts med tidigare studier.

Pålitlighet behövs för att kunna verifiera att det resultat som skapats inte är på grund av den specifika forskaren. Det vill säga att om en annan person utför experimentet ska samma resultat ske. För att experimentet ska kunna återupprepas kommer så kommer de skript som används för att utföras tas med i Bilagor D-I.

4.4.6 Sammanfattningsvaliditet

Hot mot sammanfattningsvaliditeten är när en studie får svårt att dra rätt slutsats från det resultat som kommit ut från experimentet skriver Wohlin, C. et al. (2012). Olika hot kan försvaga sammanfattningsvaliditeten. Låg styrkefunktion är ett hot där informationen resultatet gett inte är starkt nog för att kunna dra en slutsats. Letande efter bestämt resultat är ett hot mot sammanfattningsvaliditeten, då forskaren aktivt strävar efter ett specifikt resultat vilket gör att experimentet inte längre är opartiskt, och kan leda till att slutsatsen blir felaktig.

Att ha förutfattade meningar om vilken av brandväggarna som är bäst kan resultera i att felaktigheter kan ske på grund av detta. Detta är bra att ha i åtanke experimentet utförs.

4.5 Utförande, analys, tolkning, presentation och paketering

Dessa delar kommer att vara del av andra moment i arbetet.

Utförande – i denna del är det viktigt att se till att alla delar i experimentet är förberedda innan start. Samt att de saker som planerats följs upp och att de utförs.

Analys och tolkning – här kommer all data som samlats att analyseras med hjälp av deskriptiv statistik.

Presentation och Paketering – Hur den data visas som experimentet resulterat i.

5 Experimentdesign

Denna del tar upp hur experimentmiljön utformas, vilka mätredskap används och förklarar de ramar som experimentet kommer utföras med. Denna del kopplar samman med punkt två i kapitel 3.3.

5.1 Topologi

Experimentets topologi kommer baseras på två nätverkssegment med ”dual-homed” design på nätverket (Hickman, B. et al 2003). Detta innebär ett externt och ett internt nätverk skapas, där enkelriktad trafik kommer från det externa nätverket igenom brandväggen till det interna nätverket. För detta ska fungera kommer brandväggen föra trafiken från det externa till det interna nätverket med hjälp av vidarebefordran av trafik. Till IPtables kommer Ubuntu 16.04.3 LTS användas med minimuminstallation. Detta för att se till att så få andra tjänster i systemet som möjligt påverkar resultatet. FreeBSD 11.1-STABLE är den senaste stabila versionen av FreeBSD och det är denna variant som kommer användas i experimentet. För att simulera och mäta kommer två stycken Ubuntu 16.04.3 LTS maskiner skapas. En i det externa nätverket och en i det interna nätverket. Dessa kommer användas för att skicka trafik. För att se specifikationerna av maskinerna (Bilaga C). För att se hela topologin i detalj (bild 5.1 Topologi).

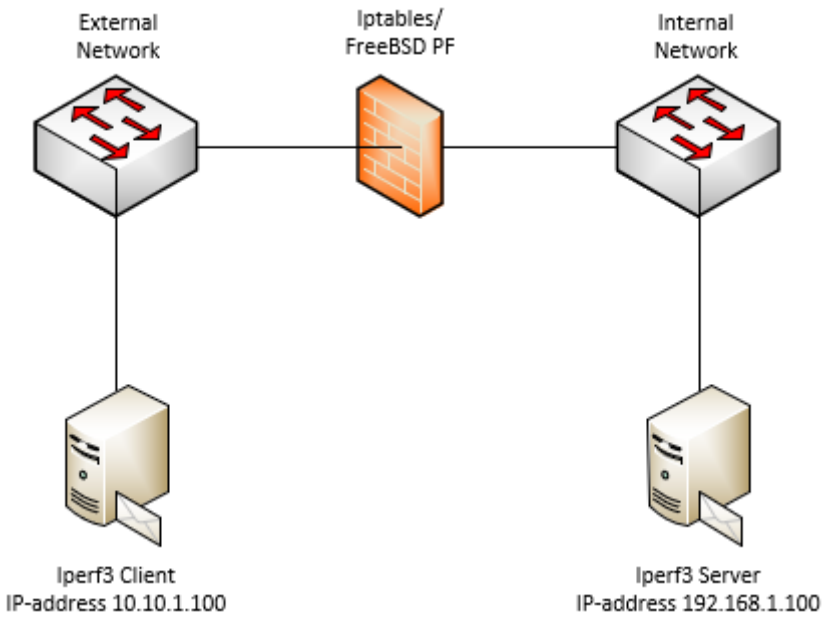


Bild 5.1 Topologi (Författarens egen)

5.2 Mätredskap

För att simulera trafik mellan det interna och externa nätverket samt kolla genomströmning så används Iperf3. Iperf3 skickar enkelriktad trafik från iperf3 klienten till iperf3 servern. För att mäta latensen så används Ping från Iperf3 klienten till Iperf3 servern. För att se hur de är placerade i topologin se bild 5.1.Vid mätning av CPU-användning används Top under tidsperioden trafiken går igenom.

5.3 Konfiguration av brandväggarna

Både brandväggarna uppdateras med de senaste säkerhetsuppdateringarna. Utöver senaste säkerhetsuppdateringarna så har båda brandväggarna installerats med Vim. Vim är en textredigerare av filer, Denna anses inte ha någon inverkan på experimentet, eftersom den är inte är aktiv under utförandet av experimentet, samt att ingen känd inverkan finns på CPU eller nätverk när den är inaktiv.

5.4 Paketström och paketstorlek

En pilotstudie görs för att få ut vilken paketström och paketstorlek som används. 20 tester görs mellan 50 Mbit och 1000Mbit bandbredd och med inspiration av RFC1944 av S. Bradner, J. McQuaid. 1996 väljs följande paketstorlekar; 64, 128, 256, 512, 1024, 1280 och 1518. Varje test gjordes i 30 sekunders intervaller på båda brandväggarna.

Efter pilotstudien väljs tre stycken paketströmmar att kolla på; 100Mbit, 500Mbit och 1000Mbit. Paketstorleken som valdes var 64, 512 och 1518. Paketströmmarna väljs eftersom det verkar vara de punkterna där Nätverkskorten ligger i låg/medel/hög användning. Paketmängderna väljs av liknande orsak där skillnaden är att ju längre brandväggen är från sin normala MTU (1500 bytes för båda operativsystemen) desto svårare blir det för brandväggarna att ta hand om trafiken. Då väljs 64, 512, 1518. Där brandväggarna har svårast att hantera trafik på 64 bytes.

5.5 Regeluppsättningar

Baserat på den tidigare pilotstudien om paketmängd och paketstorlek utförs en till pilotstudie för att se vilken mängd regeluppsättningar som experimentet ska innehålla. Tester med 1 regel upp till 4000 regler gjordes och följande regeluppsättningar ansågs vara intressanta att kolla på; 100, 500 1000, 2000 och 4000 regler.

Att notera är att syntaxen för reglerna i de olika brandväggarna är inte samma, vilket skapar utmaningar att göra så likvärdiga regeluppsättningar som möjligt. För att öka chansen till likvärdiga resultat så skapades några kriterier på regeluppsättningarna; som första regel ska de tappa alla paket som inte är tillåtna. Reglerna ska inte innehålla några extra parametrar utöver skicka trafik från A till B. I slutet ska varje regeluppsättning ha en regel som låter trafiken gå igenom. Detta gör så att varje regel bearbetas innan trafiken går igenom hos båda brandväggarna. Under så kommer två exempel på regler i brandväggarna som används under experimentet.

```
FreeBSD PF regel
pass in on em0 from 10.10.1.2 to 13.112.224.240

IPtables regel
iptables -A FORWARD -s 10.10.1.2/32 -j ACCEPT
```

De skiljer sig på olika sätt. PF regeln är tillsatt på ett specifikt nätverksgränssnitt och IPtables regeln är tillsatt på FORWARD kedjan. PF regeln är även specifikt till en annan IP adress. Orsaken bakom detta är att PF har en funktion som gör att liknande eller redundanta regler summeras till en regel. Detta går inte att inaktivera, då behövdes länkning till separata IP-adresser för att se till så att PF inte summerade regeluppsättningarna.

6 Insamling av data

Denna del kommer ta upp insamling av data och hur rå data hanteras för att kunna användas i studien.

6.1 Skript

När insamling av data ska utföras så skapas tre bash skript; Genomströmning.sh(Bilaga D), Latens.sh(Bilaga E) och CPU.sh(Bilaga F, Bilaga G). Dessa används för att skapa, samla in och formatera den data som kommer in till experimentet. Genomströmning.sh skapar fem parallella flöden med den specifika paketström och paketstorleken som ska testas till Iperf3 servern och sparar därefter ner informationen till den lokala klienten. Latens.sh skapar ett flöde av Ping i 900 sekunder samt sparar ner latensresultatet ner till Iperf3-klienten. CPU.sh mäter CPU-användning per sekund och sparar informationen på den lokala brandväggen efter en körning av testet är klart.

6.2 Insamlingsparametrar

Med hjälp av skripten ovan så testas en regeluppsättning med alla paketstorlekar och hastighetskombinationer nämnt i kap 5, dessa körs i 900 sekunder var. Efter en testkörning så väntas 30 minuter innan nästa test utförs. Dessa test återupprepades tre gånger innan nästa regeluppsättning laddades in och nästa test för den nya regeluppsättningen fortgår. För att automatisera processen så används schemaläggaren crontab för alla skript på Iperf3 klienten. För att synkronisera så att skripten kör samtidigt mellan brandväggarnas CPU-skript och Iperf3-klientens Ping-skript och Genomströmnings-skript så skapas en ssh uppkoppling med hjälp av skriptet i Bilaga H.

När CPU mäts så har Top kommandot mindre skillnader i parametrar mellan Ubuntu och FreeBSD operativsystemen. Dessa parametrar ändras för att ge lika resultat på båda brandväggarna. Skillnaderna kan observeras i Bilaga F och Bilaga G.

6.3 Filtrering av data

Den ursprungliga informationen som samlas in under experimentet är inte enkelt läsbart utan måste summeras och filtreras för att kunna användas i studien. De filer som mäter all trafik som passerat brandväggen går inte summera effektivt i deras råa format. Då används delar av tidigare nämnda skript för att sammanställa informationen från de olika trafikflödena till ett läsbart format.
För att få CPU i ett användbart format används ett bashkommando som finns i Bilaga F, rad 4 för PF och Bilaga G, rad 6 för IPtables. Dessa kollar på inaktiviteten hos en CPU varje sekund. Genom att få ut mängden inaktivitet, kunde aktiviteten hos CPU uträknas.

Så här ser den data som filtrerades ut efter insamlingen.

```
1    99.9
2    91.1
3    92.9
4    92.4
5    92.2
6    92.5
7    92.2
```

När all CPU data har samlats in, så skickas den data för analys. Vid analys upptäcktes det att Excel inte accepterar punktformat på decimaler när uträkningar utförs, ett mindre Powershell skript (bilaga I) skapas för att formatera om punkterna till kommatecken på alla filer så all data kan läggas in i ett Excel dokument.

För att undersöka genomströmning så sparas iperf3 informationen i textfiler. När informationen är samlad så används reguljära uttryck för att skapa en fil med bara slutresultatet av en körning av experimentet. Dessa slutresultat från alla körningar läggs sedan ihop i en fil, dessa analyseras och summeras sedan med hjälp av Excel.

Vid latens används samma teknik som genomströmning, med hjälp av reguljära uttryck samlas slutresultatet från alla filer till en fil som sedan kan analyseras med hjälp av Excel. Dessa reguljära uttryck visas de i Bilaga D för genomströmning och bilaga E för latens.

7 Dataanalys

I delen dataanalys beskrivs hur insamlad data analyseras och den metod som används för att göra detta. Här förklaras även hur data kommer sättas upp för att göra en jämförelse mellan brandväggarna.

7.1 Paketstorlekar

Istället för att presentera varje paketstorlek görs ett genomsnitt av alla paketstorlekars resultat för en viss hastighet och regeluppsättning. Det är dessa genomsnitt som presenteras i studien.

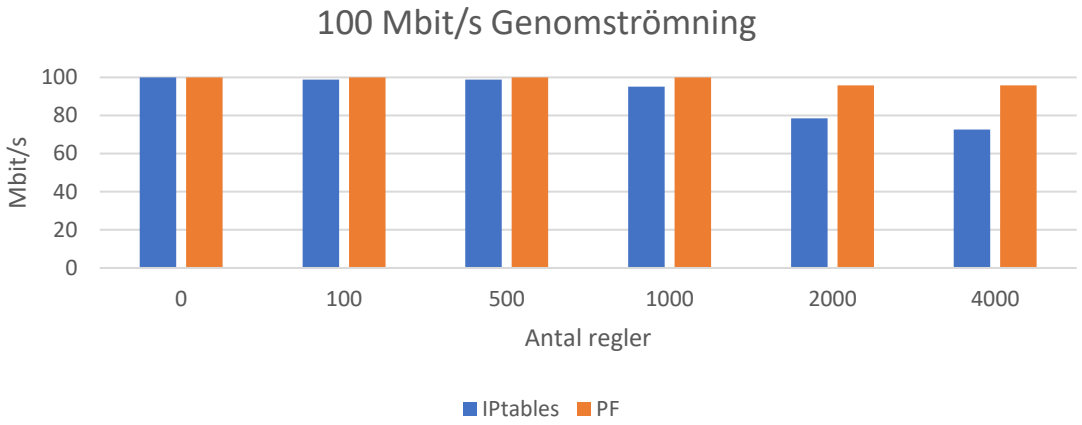
7.2 CPU, latens och genomströmning

Som tidigare nämnt så krävdes formatering av den data som kom in med hjälp av olika skript för att få en mer hanterbar data. När data sen är formaterad så läggs den in i Excelark så genomsnittet på varje resultat sammanställs.

Denna data matas sedan in i grafer där genomsnittet och standardavvikelsen av varje resultat visas. På grund av ett mindre fel i tidsberäkning missas två sekunder av CPU-beräkningen i början av varje test på brandväggarna. Detta är kan anses som en acceptabel dataförlust just på grund av den stora mängd data som är insamlad under experimentets gång.

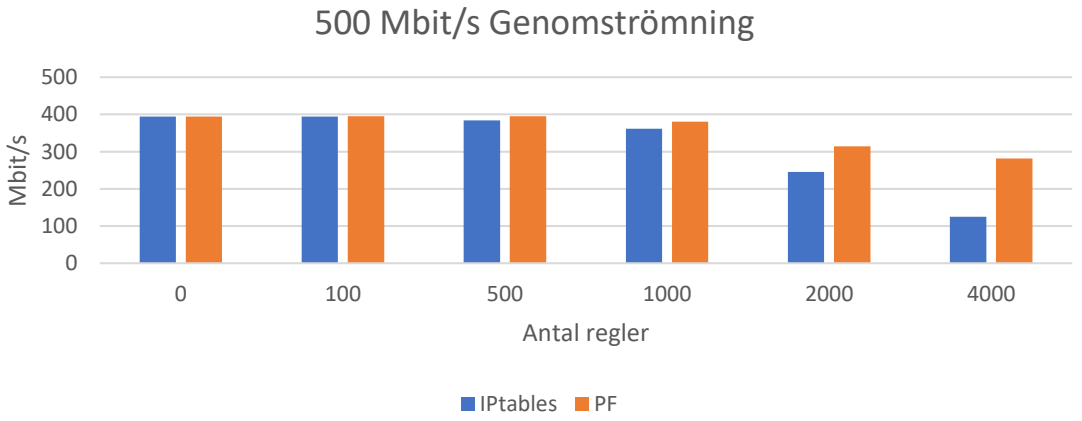
8 Resultat

8.1 Genomströmning



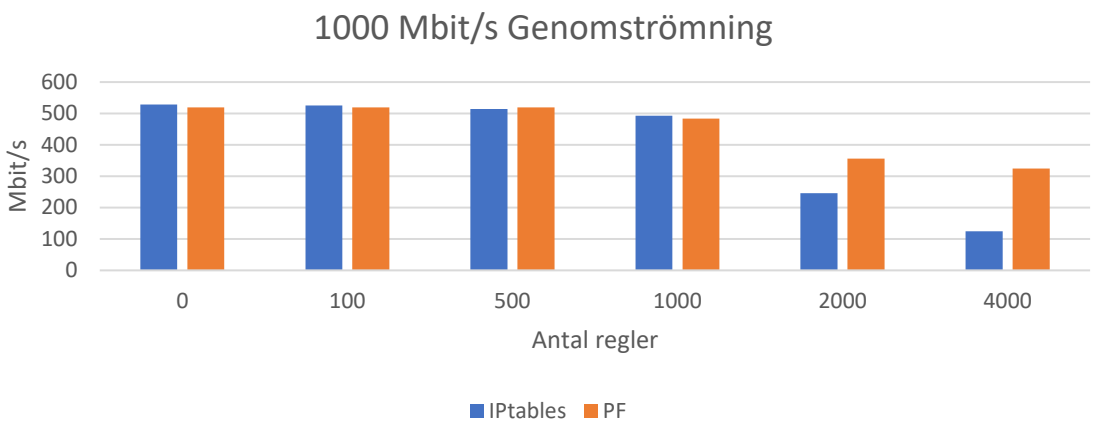
Graf 8.1 100Mbit/s genomsnittlig genomströmning

Vid 100 Mbit/s finns små skillnader i genomströmning ända upp till 1000 regler. Vid 1000 regler får PF en 4,9% högre genomströmning jämfört med IPtables där PF bibehåller 100 Mbit/s och IPtables får 95,1 Mbits/s. Vid 2000 regler blir det en tydlig genomströmningsskillnad mellan IPtables och PF, IPtables har en genomströmning på 78,46 Mbit/s och PF får en genomströmning på 95,9 Mbit/s, PF har 15,6% högre genomströmning än IPtables vid 2000 regler. Detta ökar vid 4000 regler där PF bibehåller liknande resultat med 95,7 Mbits/s medans IPtables får en genomströming på 72,7 Mbits/s, PF har 24% mer genomströmning vid 4000 regler jämfört med IPtables.



Graf 8.2 500Mbit/s genomsnittlig genomströmning

Vid 500Mbit/s är det små skillnader i genomströmning vid 0–500 regler. Vid 1000 regler har IPtables en genomströmning på 361,8 Mbits/s medans PF har en genomströmning på 380,6 Mbits/s, här har PF 4,9% högre genomströmning än IPtables. Vid 2000 regler så blir det en högre genomströmningsskillnad. IPtables har 245,7 Mbits/s och PF har 314,3 Mbits/s, PF har 21,8% högre genomströmning än IPtables vid 2000 regler. Vid 4000 regler har IPtables 124,8 Mbits/s och PF har 281,6 Mbits/s genomströmning, PF har 56% mer genomströmning jämfört med IPtables vid 4000 regler.



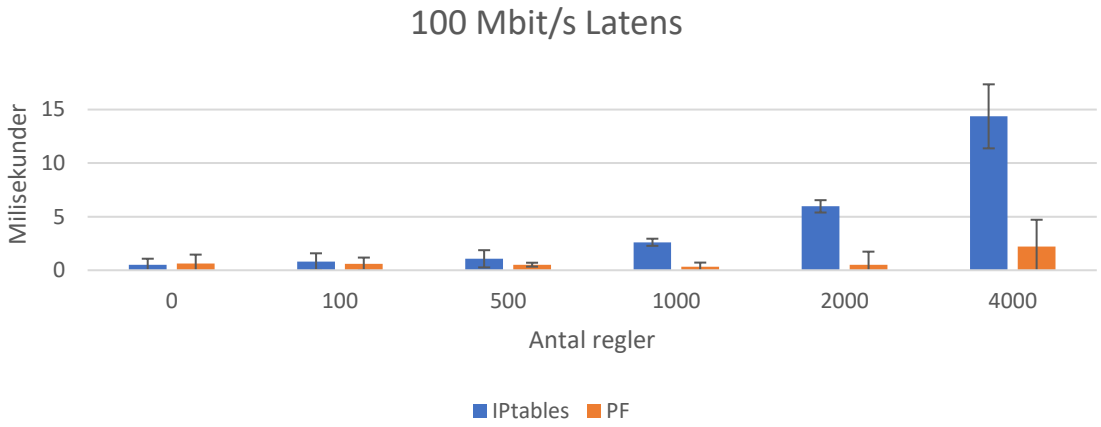
Graf 8.3 1000Mbits/s genomsnittlig genomströmning

Vid 1000Mbit/s det relativt små skillnader på genomströmning vid 0–1000 regler, men vid 2000 regler blir det en högre genomströmningsskillnad mellan brandväggarna. PF har en genomströmning på 355,6 Mbit/s och IPtables får en genomströmning på 245,7 Mbit/s, där PF har 30,9% mer genomströmning jämfört med IPtables. Vid 4000 regler har PF en genomströmning 324 Mbit/s och IPtables en genomströmning på 125 Mbit/s, PF har 61,4% högre genomströmning än IPtables vid 4000 regler.

8.1.1 Hur skiljer sig genomströmningen mellan FreeBSD PF och IPtables vid ökad mängd paketfiltreringsregler?

Resultatet visar att genomströmningen vid regeluppsättningar från 0–500 regler skiljer sig brandväggarna sig inte mycket i genomströmning från varandra oavsett trafikmängd som tillsätts till brandväggen. Vid 1000 regler märks en mindre skillnad där IPtables i genomsnitt 4,9% mindre genomströmning vid alla trafikhastigheter. Vid 2000 regler förlorar IPtables mer paket än PF ju mer mängd trafik som går igenom brandväggen med 15,6% förlust vid 100 Mbits/s, 21,8% vid 500 Mbits/s och 30,9% vid 1000 Mbits/s. Vid 4000 regler är skillnaden mellan brandväggarna som störst, IPtables följer samma mönster som vid 2000 regler fast med större genomströmningsskillnad jämfört med PF än tidigare, 24% vid 100 Mbits/s, 56% vid 500 Mbits/s och 61,4% vid 1000 Mbits/s.

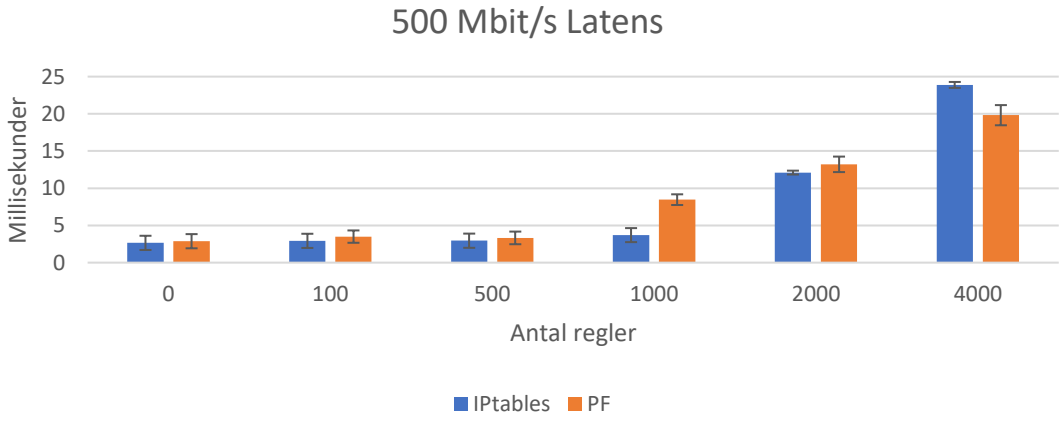
8.2 Latens



Graf 8.4 100Mbit/s genomsnittlig latens

Vid 100Mbit/s finns små skillnader i latens hos brandväggarna upp till 500 regler. Vid 500 regler har IPtables 1,06 millisekunder emot 0,52 millisekunder hos PF, en skillnad på 0,5 millisekunder. Vid 1000 regler ökar har IPtables 6 millisekunder latens jämfört med 0,5 millisekunder hos PF, en IPtables latens till 2,6 millisekunder jämfört med 0,3 millisekunder hos PF, en skillnad på 2,3 millisekunder. Vid 2000 regler skillnad på 5,5 millisekunder. Vid 4000 regler uppnår IPtables en latens på 14,4 millisekunder och PF en latens på 2,2 millisekunder, en skillnad på 12,2 millisekunder.

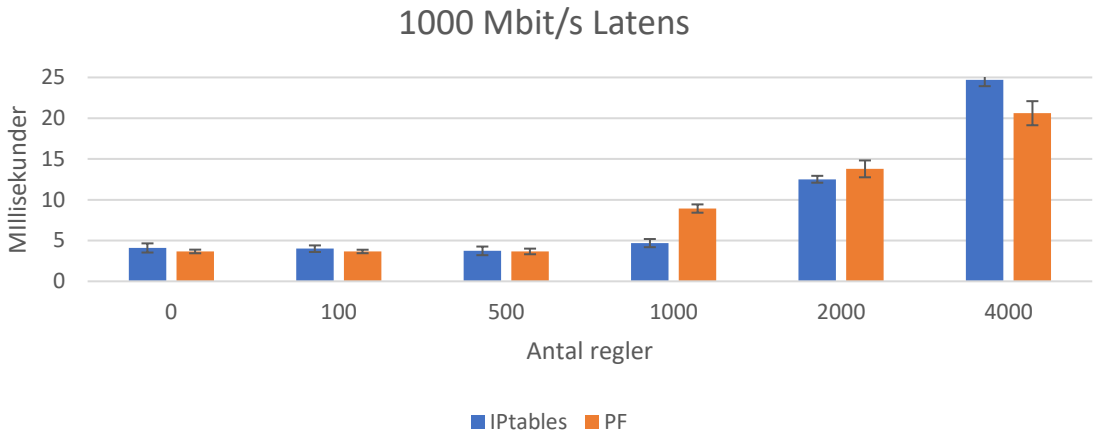
För 0–100 regler förblir skillnaden lika. Vid 500 regler ökar IPtables med en mindre skillnad vid 500 regler på 0,5 millisekunder och vid 1000 regler på 2,3 millisekunder Denna trend fortsätter vid 2000–4000 regler där IPtables får en kraftigt högre latensskillnad jämfört med PF.



Graf 8.5 500Mbit/s genomsnittlig latens

Vid 500Mbit/s finns mindre skillnader på latens ända upp till 1000 regler. Vid 1000 regler ökar PF latens märkbart med 8,5 millisekunder jämfört med 3,7 millisekunder hos IPtables, en skillnad på 4,8 millisekunder. Vid 2000 regler har PF 13,2 millisekunder jämfört med 12,1 millisekunder hos IPtables med en skillnad på 1,1 millisekunder. Vid 4000 regler uppnår IPtables en högre latens med 23,9 millisekunder jämfört med PF 19,9 millisekunder, en skillnad på 4,9 millisekunder.

Vid 0–500 regler skillnaden mellan brandväggarna relativt lika med, men vid 1000 ökar latensen på PF till 8,4 millisekunder jämfört med 3,3 millisekunder vid 500 regler. IPtables bibehåller nära samma latens på 2,9 millisekunder vid 500regler och 3,6 millisekunder vid 1000 regler. Vid 2000–4000 regler ser IPtables en kraftigare latensökning än PF där vid 2000 regler har är de närmare varandra latensmässigt och vid 4000 regler har IPtables en högre latens än PF.



Graf 8.5 1000Mbit/s genomsnittlig latens

Vid 1000Mbit/s finns små skillnader på latens ända upp till 1000 regler. Vid 1000 regler PF's latens ökar märkbart med 8,9 millisekunder jämfört med 4,7 millisekunder hos IPtables, en skillnad på 4,2 millisekunder. Vid 2000 regler har PF 13,8 millisekunder jämfört med 12,5 millisekunder hos IPtables med en skillnad på 1,3 millisekunder. Vid 4000 regler uppnår IPtables en högre latens med 24,7 millisekunder jämfört med PF med 20,6 millisekunder, en skillnad på 3,9 millisekunder.

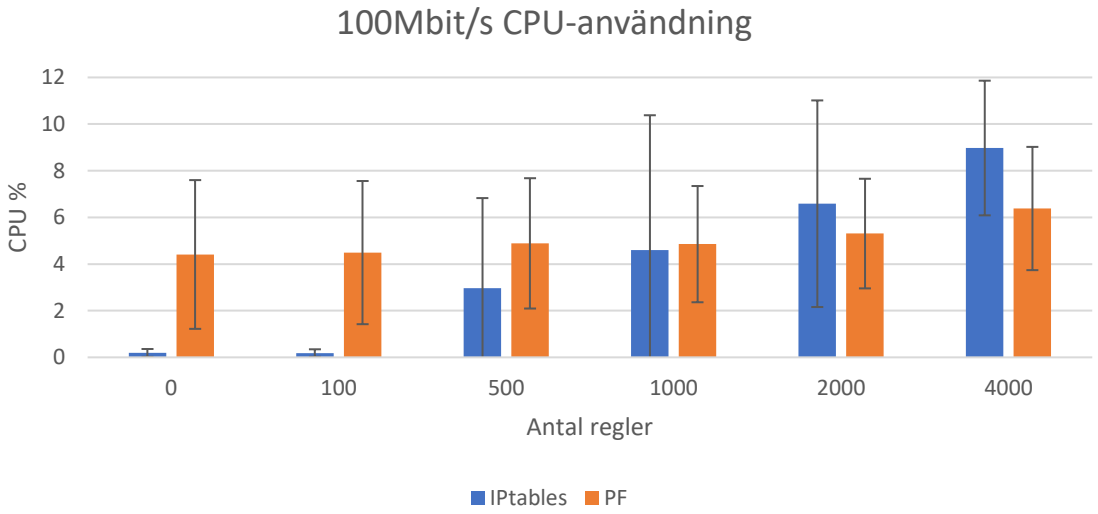
1000Mbit/s följer samma mönster som 500Mbit/s, men med generellt sätt högre latens för både brandväggarna. Vid 0–500 förblir skillnaden mellan brandväggarna relativt lika, men vid 1000 regler ökar latensen på PF till 8,9 millisekunder jämfört med 3,7 millisekunder vid 500 regler. IPtables bibehåller nära samma latens på 3,7 millisekunder vid 500regler och 4,7 millisekunder vid 1000 regler. Vid 2000–4000 ser IPtables en kraftigare latensökning än PF då både brandväggarna börjar närma sig varandra vid 2000 regler och vid 4000 regler har IPtables en högre latens än PF.

8.2.1 Hur skiljer sig latensen mellan FreeBSD PF och IPtables vid ökad mängd paketfiltreringsregler?

Genom att analysera graferna kunde generella trender hos brandväggarna identifieras. 100Mbit/s grafen skiljer sig från de andra trafikmängderna med en låg latens för PF genom alla regeluppsättningarna och en exponentiell latensökning vid 1000–4000 regler för IPtables.

500Mbit/s och 1000Mbit/s graferna följer samma mönster med inga större skillnader mellan 0-500 regler, men en stor skillnad i latens vid 1000 regler där PF får en 47,5% högre latens än IPtables vid 500 Mbit/s och 56,3% högre latens vid 1000Mbit/s, vid 2000 regler har skillnaden mellan PF och IPtables jämnats ut men där PF har fortsatt högre latens med 8,4% vid 500 Mbit/s och 9,2% högre latens vid 1000Mbit/s. Vid 4000 regler har IPtables en högre latens än PF med 16,4% vid 500 Mbit/s och 17% högre latens vid 1000Mbit/s.

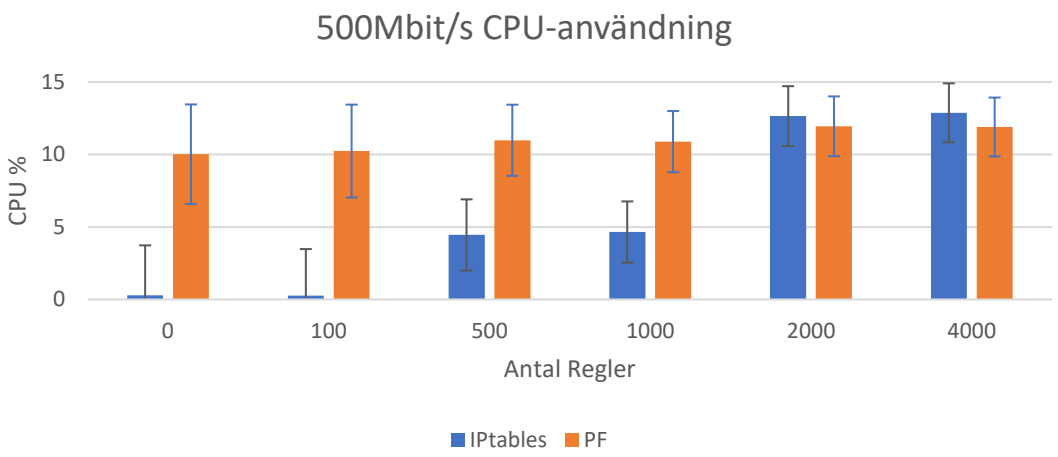
8.3 CPU-användning



Graf 8.6 100Mbit/s genomsnittlig CPU-användning i procent

Vid 100Mbit/s visar PF en högre CPU-användning vid lägre regeluppsättningar, 4,41% vid noll regler och 4,49% vid 100 regler jämfört med IPtables 0,18% vid 0 och 100 regler. Vid högre regeluppsättningar så ökar mängden CPU-användning graduellt efter varje högre regeluppsättning tillsätts till IPtables. PF bibehåller liknande CPU-användning genom alla tester och mäts som lägst vid 0 regler med 4,41% CPU-användning och som högst 6,38% vid 4000 regler.

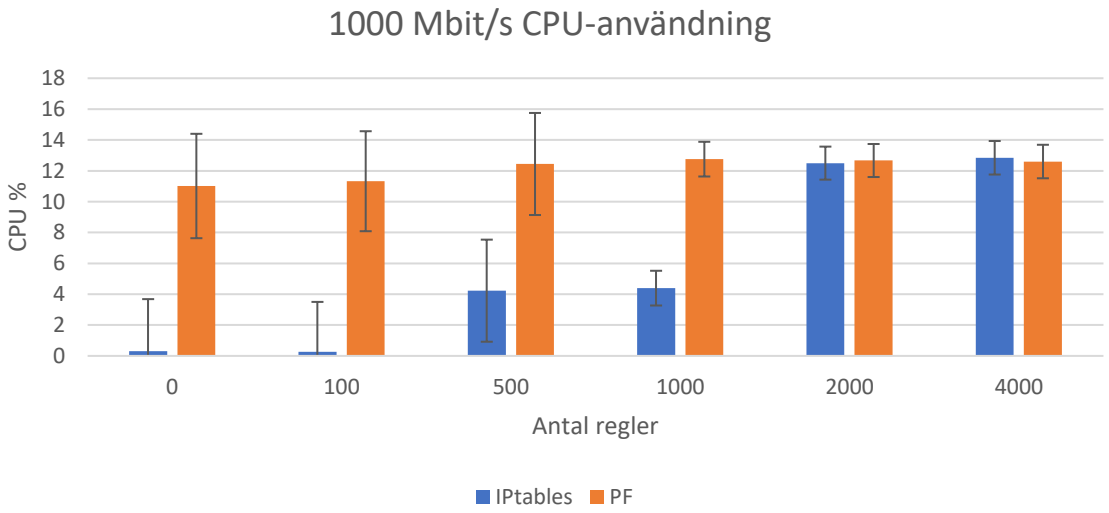
Vid 0–100 regler är skillnaden stor mellan brandväggarna, CPU-användningen hos IPtables är 95,7% lägre än PF vid 0 regler och 95,9% lägre vid 100 regler. Vid 500–1000 regler är CPU-användningen jämnare hos brandväggarna där PF har 39,3% högre CPU-användning jämfört med IPtables vid 500 regler och 5,1% högre vid 1000 regler. Vid 2000–4000 regler har IPtables högre CPU-användning än PF med en skillnad på 19,4% vid 2000 regler och 28,8% vid 4000 regler.



Graf 8.7 500Mbit/s genomsnittlig CPU-användning i procent

Det som skiljer sig från 100Mbit/s vid 500Mbit/s flödet är att IPtables inte följer den graduella ökning som innan utan får liknande CPU-belastning vid 0–100(0,29–0,26%), 500–1000 (4,45–4,64%) och 2000–4000(12,65–12,88%) regler. PF bibehåller liknande CPU-användning genom alla tester med som lägst 10,02% Belastning vid 0 regler och som högst 11,89% vid 4000 regler.

Skillnaden vid 0–100 regler är stor, CPU-användningen hos IPtables är 97,1% lägre än PF vid 0 regler och 97,4% lägre vid 100 regler. Vid 500–1000 har PF en klart högre CPU-användning än IPtables med 59,4% mer CPU-användning vid 500 regler och 57,2% vid 1000 regler. Vid 2000–4000 regler har CPU-användningen jämnats mellan brandväggarna. IPtables har en högre CPU-användning än PF på 5,5% vid 2000 regler och 7,6% vid 4000 regler.



Graf 8.8 1000Mbit/s genomsnittlig CPU-användning i procent

Resultatet från 1000Mbits/s liknar 500Mbit/s flödet, IPtables följer inte graduella ökning som vid 100Mbits/s utan får liknande CPU resultat på 0–100(0,29–0,26%), 500–1000 (4,45–4,64%) och 2000–4000(12,65–12,88%) regler. PF bibehåller tidigare mönster av liknande CPU-användning genom testerna, som lägst 11,01% vid 0 regler och som högst 12,66% vid 2000 regler.

Vid 0–100 är skillnaden stor, IPtables har en CPU-användning vid 0 regler som är 97,3% lägre än PF och 97,8% lägre vid 100 regler. Vid 500–1000 använder PF 66% mer CPU vid 500 regler och 65,6% vid 1000 regler jämfört med IPtables. Lägre skillnad mäts vid 2000–4000 regler. Vid 2000 regler har PF 1,5% mer CPU-användning i jämförelse till IPtables, vid 4000 regler har IPtables en högre CPU-användning på 1,9% jämfört med PF.

8.3.1 Hur skiljer sig CPU-användningen mellan FreeBSD PF och IPtables vid ökad mängd paketfiltreringsregler?

Resultatet visar att IPtables har en betydligt lägre CPU-användning vid få mängder regler än PF men eskalerar i CPU-användning ju fler regler som tillsätts. Vid 100Mbit/s har IPtables en låg CPU-användning vid 0–100 regler och en kraftig ökning av CPU-användning från 500–4000 regler. Resultatet från 500Mbits/s och 1000Mbit/s liknar varandra i mönster där IPtables CPU-användning får lika resultat vid 0–100 regler, 500–1000 regler och 2000–4000 regler. Från en lägre CPU-användning vid 0–1000 regler jämfört med PF till en likvärdig CPU-användning med PF vid 2000–4000 regler. Den generella CPU-användningen hos PF vid lägre regeluppsättningar är kraftigt högre än IPtables oavsett trafikmängd som används, men ökar enbart med cirka 2% CPU-användning genom testerna av alla regeluppsättningar.

Vid 0 regler har brandväggarna störst skillnad i CPU-användning där IPtables använder 95,7% mindre CPU än PF vid 100 Mbits/s, 97,1% vid 500 Mbits/s och 97,3% vid 1000 Mbit/s. Samma trend sker vid 100 regler där IPtables 95,9% mindre CPU vid 100 Mbits/s, 97,4% vid 500 Mbits/s och 97,8% vid 1000 Mbit/s jämfört med PF. Vid 500 regler får IPtables använder 39,3% mindre CPU än PF vid 100 Mbits/s, 59,4% vid 500 Mbits/s och 66% vid 1000 Mbit/s. Vid 1000 regler får IPtables använder 5,1% mindre CPU än PF vid 100 Mbits/s, 57,2% vid 500 Mbits/s och 65,6% vid 1000 Mbit/s. Vid 2000 regler är det varierade resultat med högre CPU för IPtables med 19,4% vid 100Mbits och 5,5% vid 500 Mbit/s och vid 1000Mbit så har PF en 1,5% högre CPU. Vid 4000 regler har IPtables en högre CPU-användning än PF med 28,8% vid 100 Mbits/s, 7,6% vid 500 Mbits/s och 1,9% vid 1000 Mbit/s.

9 Diskussion

9.1 Metoddiskussion

Under studiens gång gjordes ändringar i planeringen. I den ursprungliga planen skulle experimentet utföras med hjälp av en virtualiserad plattform. Då detta testades i den interna miljön blev paketförlusten oproportionerligt stor även vid låg trafik. Eftersom pilottesterna i den virtuella miljön inte var tillförlitliga gjordes valet att använda en fysisk experimentmiljö och majoriteten av arbetet behövde göras om. Detta skapade stora utmaningar i att planera experimentet och att hinna med allt.

Redskapen ITG-Recv och ITG-Sender var först planerade att användas som instrument för att mäta genomströmning och latens. Dessa redskap erbjöd enkel datasummering samt OWD, vilket anses vara en bättre metod för att mäta latens med än RTT. På grund av opålitliga resultat i pilotstudierna användes istället Iperf3 och ping.

9.2 Resultatdiskussion

Den låga genomströmningen under experimentet vid framförallt 1000Mbit/s-testerna verkar till stor del bero på testerna med 64 bytes paketstorlek, dessa kunde som mest skicka ut 184 Mbit/s oavsett regeluppsättning, vilket drog ner genomströmningsmängden när resultaten summerades. Detta hände för båda brandväggarna vilket tyder på att det kan vara relaterat till Iperf3’s förmåga att skicka större mängder paket med 64 bytes paketstorlek.

Den låga latensen i experimentet beror delvis på den korta distansen trafiken behövde för att passera till slutdestinationen. Vid ett verkligt scenario är det troligt att trafiken hade gått igenom en större mängd nätverksenheter och rest längre fysiska distanser innan trafiken nått sin slutdestination. En sannolikhet hade varit att andra latenstider uppnåtts i den miljön. Den korta distansen trafiken reste från Iperf3 klienten till Iperf3 servern skapades för att isolera experimentet, samt för att underlätta kontrollen av experimentmiljön.

Den utökade CPU-användningen hos PF jämfört med IPtables var intressant resultatet och ett mindre experiment skapades för att hitta orsaken bakom den högre CPU-användningen. När Iperf3 eller Ping skickar trafik enskilt så bibehåller PF lika resultat med IPtables, men när ICMP paket och UDP strömmar kombineras så ökas CPU-användningen markant hos PF. Efter sökning på nätet och FreeBSD forumen så hittades ingen direkt orsak till varför detta skulle uppstå. Potentiella orsaker kan vara den aktiva konfigurationen som används, det fysiska CPU-chippet eller de Nätverkskort som används under experimentet.

9.3 Etiska aspekter

För att förhindra eventuella förutfattade meningar speglas i uppsatsen fick två studenter läsa igenom arbetet efter vinklade eller subjektiva delar. Detta görs för att stärka objektiviteten i studien. Utöver detta identifierades inga etiska hot under planeringen och experimentets gång.

9.4 Samhällsaspekter

Paketfiltrering är ett redskap som många av dagens nätverkstopologier använder sig av. Denna studie kan hjälpa individer och företag att göra ett mer informerat val inom vilken brandvägg som kan passa deras behov. Personen kan då utgå ifrån de tre attribut som mäts och kan de sedan välja den brandvägg som passar bäst i deras topologi. Denna studie hjälper

även till att upplysa om reglers påverkan hos brandväggar och de potentiella nackdelarna med att ha för stora regeluppsättningar.

9.5 Vetenskapliga aspekter

Bidraget till forskningsvärlden är en studie mellan två populära brandväggar, som kan ligga till grund för vidare forskning inom brandväggsområdet och vilka typer av optimeringar som kan utföras gällande paketfiltrering. Denna studie undersöker tre olika attribut vilket ger mer helhetsbild över hur brandväggarna reagerar vid olika regeluppsättningar. Detta kan användas för att göra en övergripande studie om hur brandväggar påverkas av brandväggsregler, vilket skulle kunna ge producera en mer generell slutsats.

Den metodik som används kan även implementeras eller agera som inspiration för liknande studier och forskning. Metodiken för experimentdesignen kan användas till ytterligare forskning där ett större antal brandväggar kan jämföras. Samma metodik kan även tillämpas i andra brandväggstester då studien baseras på de tidigare föreslagna standarderna för testning av brandväggar och nätverksanslutningsenheter.

9.6 Slutsats

I studien syntes inga större skillnader mellan brandväggarna vid 0–1000 regler. När antalet regler översteg 2000 visade studien att IPtables hade större paketförluster än PF oavsett trafikmängd.

Vid 0-500 regler har de båda brandväggarna lika låg latens. Vid 1000–2000 regler har PF en högre latens än IPtables och vid 500 regler har IPtables högre latens än PF. Då flödet genom brandväggarna var 100 Mbit/s skiljde sig dock resultatet från de andra testerna. Då hade PF låg latens vid samtliga regeluppsättningar och latensen för IPtables ökade kraftigt vid fler än 1000 regler.

Resultatet för CPU-användning visar att IPtables har en kraftigt lägre CPU-användning än PF vid mindre mängder regler, men eskalerar kraftigt i CPU-belastning ju fler regler som tillsätts. medans CPU-användningen för PF börjar med en högre CPU-användning än IPtables, men förändras bara med cirka 2% CPU-användning genom alla tester, oavsett trafikmängd som används.

Sammanfattningsvis är brandväggarnas prestanda likvärdiga i genomströmning och latens vid lägre regelmängder. CPU-belastning är dock lägre hos IPtables. Detta innebär att IPtables anses vara den bättre brandväggen för låga regeluppsättningar. Enligt studiens resultat är PF bättre anpassad för högre regeluppsättningar.

9.7 Framtida arbeten

Denna studie testade IPtables och FreeBSD PF. Det hade varit intressant att göra en större studie med fler brandväggar inkluderande. Denna studie gav ut värdena i en graf för varje regeluppsättning. Det kan vara intressant att utföra en studie som gör ett konstant trafikflöde genom brandväggarna i en längre tidsperiod, samtidigt läggs ökande mängd regler till i brandväggarna. Då kan resultatet av vilken belastning varje regel lägger på en brandvägg beräknas, samt se vilka brandväggar som klarar av störst mängd brandväggsregler. I denna studie ändrades inte kärnan hos programmen så de förblev enkeltrådade, Att utföra testet med flertrådad konfiguration på brandväggarna hade potentiellt kunna ge annat resultat och hade

kunnat öka regelmängder som kan läggas in i brandväggen innan överbelastning. En annan aspekt att kolla på hade varit att skicka en specifik trafik genom brandväggarna som till exempel VOIP trafik, då kan brandväggarna jämföras om hur lämpade de är att hantera en viss trafiktyp.

Källor

Zilberman, N. *et al.* (2015) *'Reconfigurable network systems and software-defined networking'*, *Proceedings of the IEEE*, 103(7), pp. 1102–1124. doi: 10.1109/JPROC.2015.2435732.

Hickman, B., Newman, D., Tadjudin, S. & Martin, T. (2003). *Benchmarking Methodology for Firewall Performance.* Hämtad 29 Januari, 2018, från https://tools.ietf.org/html/rfc3511

G. Almes, S. Kalidindi, M. Zekauskas. (1999). *RFC2679 A One-way Delay Metric for IPPM.* http://softwareandservices.net/Documentation/public/IETF/html/dc/d31/RFC2679.html

Information Sciences Institute (1981). *Darpa Internet Program Protocol Specification.* Hämtad 14 Mars, 2018, från https://tools.ietf.org/html/rfc791

S. Bradner, J. McQuaid (1996). '*Benchmarking Methodology for Network Interconnect Devices'* Hämtad 13 Maj, 2018, från https://tools.ietf.org/html/rfc1944

Oppliger, R. (1997) 'Internet security: firewalls and beyond', *Communications of the ACM*, 40(5), pp. 92–102. doi: 10.1145/253769.253802.

Acharya, H. B. & Gouda, M. G. (2009) 'Linear-time verification of firewalls', *Proceedings - International Conference on Network Protocols*, ICNP, pp. 133–140. doi: 10.1109/ICNP.2009.5339691.

Errin W. Fulp (2005*). Optimization of network firewall policies using ordered sets and directed acyclical graphs*. Proc. of IEEE Internet Management Conference, 2005 - csweb.cs.wfu.edu

Hoffman, D., Prabhakar, D. & Strooper, P. (2003) 'Testing iptables', *Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*, pp. 80–91.

Bruno Pedrozo. (2007) The purpose of a firewall [bild] Hämtad från: https://commons.wikimedia.org/w/index.php?curid=2864257 (CC BY-SA 3.0) https://creativecommons.org/licenses/by/3.0/

Hartmeier, D. (2002) 'Design and Performance of the OpenBSD Stateful Packet Filter (pf)', *USENIX 2002 Annual Technical Conference*, Freenix Track – Paper, Pp. 171–180 of the Proceedings.

Jeffrey, A. & Samak, T. (2009) 'Model checking firewall policy configurations', *Proceedings - 2009 IEEE International Symposium on Policies for Distributed Systems and Networks, POLICY* 2009, (1), pp. 60–67. doi: 10.1109/POLICY.2009.32.

OpenBSD PF (2017) *'Packet Filtering'* Hämtad 27 februari, 2018 från: https://www.openbsd.org/faq/pf/filter.html

Netfilter (2017) *'IPtables'* Hämtade 28 februari, 2018 från: http://ipset.netfilter.org/iptables.man.html

PFsense (2017) *'Low Throughput Troubleshooting'* Hämtad 5 mars från: https://doc.pfsense.org/index.php/Low_Throughput_Troubleshooting

Yoon, M., Chen, S. and Zhang, Z. (2010) ‘Minimizing the maximum firewall rule set in a network with multiple firewalls’, *IEEE Transactions on Computers*, 59(2), pp. 218–230. doi: 10.1109/TC.2009.172.

Sulaiman, N., Carrasco, R. & Chester, G. (2007). Impact of Traffic on Multi Service IP Based Applications. *Second International Conference on Innovative Computing, Information and Control*, 2007. ICICIC '07. doi: 10.1109/ICICIC.2007.344

Wohlin, C. et al. (2012) Experimentation in software engineering, Experimentation in Software Engineering. doi: 10.1007/978-3-642-29044-2.

Vmware (2017). *‘Host-Only Networking’*. Hämtad 22 Februari, 2018, från: https://www.vmware.com/support/ws5/doc/ws_net_configurations_hostonly.html

Salah, K., Elbadawi, K. & Boutaba, R. (2012) *‘Performance Modeling and Analysis of Network Firewalls’*, 9(1), pp. 12–21.

Tihomir K, Predrag P (2007) *‘Optimization of Firewall Rules’*, pp. 685–690.

Cisco (2005) *‘Measuring Delay, Jitter, and Packet Loss with Cisco IOS SAA and RTTMON’* Hämtad 19 Mars, 2018, från: https://www.cisco.com/c/en/us/support/docs/availability/high-availability/24121-saa.html

Kocher, P. et al. (2018) ‘Spectre Attacks: Exploiting Speculative Execution’. Hämtad 7 januari, 2018, från: http://arxiv.org/abs/1801.01203

PFsense (2017b*) ‘Versions of pfSense and FreeBSD’* Hämtad 5 mars, 2018, från: https://doc.pfsense.org/index.php/Versions_of_pfSense_and_FreeBSD

Gouda, M. G. & Liu, A. X. (2005) ‘A model of stateful firewalls and its properties’, *Proceedings of the International Conference on Dependable Systems and Networks*, pp. 128–137. doi: 10.1109/DSN.2005.9.

Liu, A. X. & Gouda, M. G. (2009) ‘Firewall policy queries’, *IEEE Transactions on Parallel and Distributed Systems*, 20(6), pp. 766–777. doi: 10.1109/TPDS.2008.263.

Comerford, P., Davies, J. N. and Grout, V. (2016) ‘Reducing Packet Delay Through Filter Merging’, *Proceedings of the 9th International Conference on Utility and Cloud Computing, (Ucc 2016)*, pp. 358–363. doi: 10.1145/2996890.3007854.

Karrer, R. P., Botta, A. and Pescapé, A. (2008) ‘High-speed backhaul networks: Myth or reality?’, *Computer Communications*, 31(8), pp. 1540–1550. doi: 10.1016/j.comcom.2008.01.034.

Siegel, S., & Castellan, N. J. (1988). *Nonparametric statistics for the behavioral sciences (2nded.)*. New York: McGraw-Hill.

Bilaga A – Wohlin mall

Mall	***Förklaring***
Analysera **<Objekt som ska studeras>**	***Objekt som ska studeras*** *– är vad som ska studeras i experimentet, exempel på detta är Modeller, produkter eller teorier.*
Med ändamålet av att **<Syfte>**	***Syfte-*** *definierar vad åtanken är med arbetet. Det kan vara att kolla påverkan av två tekniker, eller kollande förhållande mellan objekt.*
Med avseende på deras **<Kvalitetsfokus>**	***Kvalitetsfokus-*** *är huvudeffekten av det som ska studeras.*
Från synvinkel av **<Perspektiv>**	***Perspektiv****- förklarar vilken synvinkel resultatet av experimentet kommer utgå ifrån.*
I kontexten av **<Kontext>**	***Kontext****- ger information om vad eller vem som är involverad i experimentet.*

BILAGA B - Validitetshot

Validitetshot	**Potentiella risker**	**Åtgärder för att undvika riskerna**
Maskinerna för brandväggarna har olika prestanda.	Studien ger ett vinklat resultat.	Båda FreeBSD och Ubuntu maskinerna får samma mängd hårdvara att utgå ifrån.
Maskinerna som brandväggarna körs på är inte identiska	Kan göra så att resultaten inte går att jämföra	Brandväggarna körs på samma hårdvara.
Testet utförs på ett gemensamt nätverk	Externdata kan förstöra experimentets validitet och resultat	Testerna görs internt inom den virtualiserade miljön med ingen externkoppling.
Andra tjänsterna körs på brandväggarna	Kan potentiellt ändra resultatet och lägga en onödig belastning på brandväggarna	På både brandväggarna så installeras med minimuminstallation och inga tjänster utöver den testade brandväggen är konfiguerade.
Det finns olika sätt att konfigurera brandväggarna	Kan påverka prestandan så att en brandvägg presterar bättre än den andra	Både brandväggarna konfigureras med så lika omständigheter som möjligt
Regler kan placeras olika ordning i brandväggen	Kan skapa att en brandvägg inte behöver läsa in alla regler. Därmed påverka resultatet.	Både brandväggarnas regeluppsättningar har samma ordning på reglerna.

BILAGA C

Virtuell Maskin	CPU	RAM
FreeBSD 11.1-STABLE - FREEBSD PF	Intel Xeon W3550, 3,07 GHz	24 GiB, 1333 MHz
Ubuntu 16.04.3 LTS - IPtables	Intel Xeon W3550, 3,07 GHz	24 GiB, 1333 MHz
Ubuntu 16.04.3 LTS - Klient	Intel Xeon W3550, 3,07 GHz	24 GiB, 1333 MHz
Ubuntu 16.04.3 LTS - Server	Intel Xeon W3550, 3,07 GHz	24 GiB, 1333 MHz

Nätverkskort – Brandväggar:
Nätverksgränssnitt 1 Broadcom BCM5764M Gigabit Ethernet
Nätverksgränssnitt 2 Intel 82541PI Gigabit Ethernet Controller

Nätverkskort – Klient, Server:
Nätverksgränssnitt 1 Broadcom BCM5764M Gigabit Ethernet

BILAGA D – Genomströmning.sh

```
1.  #!/bin/bash
2.  sleep 2
3.  iperf3 -c 192.168.1.100 -u -t 900 -V -b 20m -l 64 -
    P 5 >> /home/andreas/output/IPtables/throughput/100m64byte.txt
4.  echo "1 TP"
5.  iperf3 -c 192.168.1.100 -u -t 900 -V -b 20m -l 512 -
    P 5 >> /home/andreas/output/IPtables/throughput/100m512byte.txt
6.  echo "2 TP"
7.  iperf3 -c 192.168.1.100 -u -t 900 -V -b 20m -l 1500 -
    P 5 >> /home/andreas/output/IPtables/throughput/100m1500byte.txt
8.  echo "3 TP"
9.  iperf3 -c 192.168.1.100 -u -t 900 -V -b 100m -l 64 -
    P 5 >> /home/andreas/output/IPtables/throughput/500m64byte.txt
10. echo "4 TP"
11. iperf3 -c 192.168.1.100 -u -t 900 -V -b 100m -l 512 -
    P 5 >> /home/andreas/output/IPtables/throughput/500m512byte.txt
12. echo "5 TP"
13. iperf3 -c 192.168.1.100 -u -t 900 -V -b 100m -l 1500 -
    P 5 >> /home/andreas/output/IPtables/throughput/500m1500byte.txt
14. echo "6 TP"
15. iperf3 -c 192.168.1.100 -u -t 900 -V -b 0 -l 64 -
    P 5 >> /home/andreas/output/IPtables/throughput/1000m64byte.txt
16. echo "7 TP"
17. iperf3 -c 192.168.1.100 -u -t 900 -V -b 0 -l 512 -
    P 5 >> /home/andreas/output/IPtables/throughput/1000m512byte.txt
18. echo "8 TP"
19. iperf3 -c 192.168.1.100 -u -t 900 -V -b 0 -l 1500 -
    P 5 >> /home/andreas/output/IPtables/throughput/1000m1500byte.txt
20. echo "9 TP"
21. path="norules"
22. cd /home/andreas/output/IPtables/throughput/
23. if [[ -e $path ]] ; then
24.         i=0
25.         while [[ -e $path-$i ]] ; do
26.                 let i++
27.         done
28.         path=$path-$i
29. fi
30. echo $path
31.
32. if [ "$path" = "norules-2" ] ; then
33. w=0
34. path=100rules
35.         while [[ -e $path-$w ]] ; do
36.                 let w++
37.         done
38. path=100rules-$w
39. fi
40. echo $path
41.
42.
43.
44. if [ "$path" = "100rules-2" ] ; then
45. n=0
46. path=500rules
47.         while [[ -e $path-$n ]] ; do
48.                 let n++
49.         done
50. path=500rules-$n
51. fi
52. echo $path
53.
54.
```

```
if [ "$path" = "500rules-2" ] ; then
m=0
path=1000rules
        while [[ -e $path-$m ]] ; do
                let m++
        done
path=1000rules-$m
fi
echo $path
if [ "$path" = "1000rules-2" ] ; then
f=0
path=2000rules
        while [[ -e $path-$f ]] ; do
                let f++
        done
path=2000rules-$f
fi

if [ "$path" = "2000rules-2" ] ; then
d=0
path=4000rules
       while [[ -e $path-$d ]] ; do
                let d++
        done
path=4000rules-$d
fi



echo $path
mkdir $path

 cat 500m512byte.txt | grep ".....0.00-900.00" > FINAL500m512byte.txt
 cat 100m512byte.txt | grep ".....0.00-900.00" > FINAL100m512byte.txt
 cat 1000m512byte.txt | grep ".....0.00-900.00" > FINAL1000m512byte.txt
 cat 500m512byte.txt | grep ".....0.00-900.00" > FINAL500m512byte.txt
 cat 100m512byte.txt | grep ".....0.00-900.00" > FINAL100m512byte.txt
 cat 100m64byte.txt | grep ".....0.00-900.00" > FINAL100m64byte.txt
 cat 500m64byte.txt | grep ".....0.00-900.00" > FINAL500m64byte.txt
 cat 1000m64byte.txt | grep ".....0.00-900.00" > FINAL1000m64byte.txt
 cat 1000m1500byte.txt | grep ".....0.00-900.00" > FINAL1000m1500byte.txt
 cat 100m1500byte.txt | grep ".....0.00-900.00" > FINAL100m1500byte.txt
 cat 500m1500byte.txt | grep ".....0.00-900.00" > FINAL500m1500byte.txt

      IFS=$'\n' read -d '' -r -
   a lines < /home/andreas/output/IPtables/throughput/filename/filenames.txt
      for element in ${lines[*]}
      do
      echo "$element--$path" | cat - $element > temp && mv temp $element
      done

      results=results-$path.txt

      cat FIN* > $results
      mv *txt $path

      ssh root@10.10.1.1  "echo "1" > wait && exit"
```

BILAGA E – Latens.sh

```
1.  #!/bin/bash
2.  sleep 2
3.  ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 100_64 \ $ping_output" >> /home/
    andreas/output/IPtables/ping/100mping64.txt; done
4.  echo "ping 1"
5.  ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 100_512 \ $ping_output" >> /home
    /andreas/output/IPtables/ping/100mping512.txt; done
6.  echo "ping 2"
7.  ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 100_1500 \ $ping_output" >> /hom
    e/andreas/output/IPtables/ping/100mping1500.txt; done
8.  echo "ping 3"
9.  ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 500_64 \ $ping_output" >> /home/
    andreas/output/IPtables/ping/500mping64.txt; done
10. echo "ping 4"
11. ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 500_512 \ $ping_output" >> /home
    /andreas/output/IPtables/ping/500mping512.txt; done
12. echo "ping 5"
13. ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 500_1500 \ $ping_output" >> /hom
    e/andreas/output/IPtables/ping/500mping1500.txt; done
14. echo "ping 6"
15. ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 1000_64 \ $ping_output" >> /home
    /andreas/output/IPtables/ping/1000mping64.txt; done
16. echo "ping 7"
17. ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 1000_512 \ $ping_output" >> /hom
    e/andreas/output/IPtables/ping/1000mping512.txt; done
18. echo "ping 8"
19. ping 192.168.1.100 -
    w 900 | while read ping_output; do echo "$(date): \ 1000_1500 \ $ping_output" >> /ho
    me/andreas/output/IPtables/ping/1000mping1500.txt; done
20. echo "ping 9"
21. cd /home/andreas/output/IPtables/ping/
22. path="norules"
23. if [[ -e $path ]] ; then
24.         i=0
25.        while [[ -e $path-$i ]] ; do
26.                 let i++
27.         done
28.         path=$path-$i
29. fi
30. echo $path
31.
32.
33. if [ "$path" = "norules-2" ] ; then
34. w=0
35. path=100rules
36.         while [[ -e $path-$w ]] ; do
37.                 let w++
38.         done
39. path=100rules-$w
40. fi
41. echo $path
42.
43.
44.
45. if [ "$path" = "100rules-2" ] ; then
```

```
n=0
path=500rules
        while [[ -e $path-$n ]] ; do
                let n++
        done
path=500rules-$n
fi
echo $path


if [ "$path" = "500rules-2" ] ; then
m=0
path=1000rules
        while [[ -e $path-$m ]] ; do
                let m++
        done
path=1000rules-$m
fi
echo $path

if [ "$path" = "1000rules-2" ] ; then
f=0
path=2000rules
        while [[ -e $path-$f ]] ; do
                let f++
        done
path=2000rules-$f
fi

if [ "$path" = "2000rules-2" ] ; then
wq=0
path=4000rules
        while [[ -e $path-$wq ]] ; do
                let wq++
        done
path=4000rules-$wq
fi

results=results-$path.txt

echo "$path                                                                   MIN   AVG
      MAX   MDEV "  >> test.txt; cat *.txt | grep rtt >> $results
echo $path
mkdir $path
mv *.txt $path
```

BILAGA F – CPU.sh – PF

```
#!/bin/bash
top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 100m64cpu.txt
rm -r topoutput.txt

top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 100m512cpu.txt
rm -r topoutput.txt

top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 100m1500cpu.txt
rm -r topoutput.txt

top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 500m64cpu.txt
rm -r topoutput.txt

top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 500m512cpu.txt
rm -r topoutput.txt

top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 500m1500cpu.txt
rm -r topoutput.txt

top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 1000m64cpu.txt
rm -r topoutput.txt

top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 1000m512cpu.txt
rm -r topoutput.txt

top -b -s 1 -d 901 >> topoutput.txt
echo "done!"
cat topoutput.txt | grep idle | cut -c 61-64 | nl > 1000m1500cpu.txt
rm -r topoutput.txt

sleep 100
#touch waittime.txt
#while [[ "$var" -ne 1 ]]; do
#   var=$(cat "wait")
#   sleep 10
#   echo "$(date) -- waiting..." >> waittime.txt
#done
echo "moving on, creating folder, adding rules."
amount="1"
path="norules"
if [[ -e $path ]] ; then
        i=0
        while [[ -e $path-$i ]] ; do
        let i++
        if [ $i -eq $amount ]; then
        echo "100rules"
        pfctl -F all
```

```
    sleep 10
    pfctl -f 100rulis
 sleep 10
        fi
        done
        path=$path-$i
fi
echo $path
if [ "$path" = "norules-2" ] ; then
w=0
path=100rules
        while [[ -e $path-$w ]] ; do
                let w++
            if [ $w -eq $amount ]; then
            echo "500rules"
sleep 10
            pfctl -F all
sleep 10
            pfctl -f 500rulis
            fi
        done
path=$path-$w
fi
echo $path

if [ "$path" = "100rules-2" ] ; then
n=0
path=500rules

        while [[ -e $path-$n ]] ; do
                let n++
            if [ $n -eq $amount ]; then
            echo "1000rules"
 sleep 10
            pfctl -F all
 sleep 10
                  pfctl -f 1000rulis
                  fi
              done
     path=$path-$n
     fi

     if [ "$path" = "500rules-2" ] ; then
     o=0
     path=1000rules

             while [[ -e $path-$o ]] ; do
                     let o++
                                if [ $o -eq $amount ]; then
                                echo "2000rules"
      sleep 10
                  pfctl -F all
      sleep 10

                                pfctl -f 2000rulis
                                fi
             done
     path=$path-$o
     fi

     if [ "$path" = "1000rules-2" ] ; then
     dq=0
     path=2000rules

             while [[ -e $path-$dq ]] ; do
                     let dq++
```

```
                                if [ $dq -eq $amount ]; then
                                echo "4000rules"
        sleep 10
                    pfctl -F all
        sleep 10
                                pfctl -f 4000rulis
        sleep 10
                                fi
                done
      path=$path-$dq
      fi

      if [ "$path" = "2000rules-2" ] ; then
      aq=0
      path=4000rules

                while [[ -e $path-$aq ]] ; do
                        let aq++
                done
      path=$path-$aq
      fi

      echo "0" > wait
      echo $path
      echo "$path.... klar cpu"
      mkdir $path
      mv *.txt $path

      #cat $path/*
```

BILAGA G – CPU.sh - IPtables

```
#!/bin/bash

top -b -d 1 >> topoutput.txt &
PID=$!
sleep 906
kill $PID
echo "done!"
cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 100m64cpu.txt
rm -r topoutput.txt

top -b -d 1 >> topoutput.txt &
PID=$!
sleep 906
kill $PID
echo "done!"
cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 100m512cpu.txt
rm -r topoutput.txt

top -b -d 1 >> topoutput.txt &
PID=$!
sleep 906
kill $PID
echo "done!"
cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 100m1500cpu.txt
rm -r topoutput.txt
top -b -d 1 >> topoutput.txt &
PID=$!
sleep 906
kill $PID
echo "done!"
cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 500m64cpu.txt
rm -r topoutput.txt

top -b -d 1 >> topoutput.txt &
PID=$!
sleep 906
kill $PID
echo "done!"
cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 500m512cpu.txt
rm -r topoutput.txt

top -b -d 1 >> topoutput.txt &
PID=$!
sleep 906
kill $PID
echo "done!"
cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 500m1500cpu.txt
rm -r topoutput.txt
top -b -d 1 >> topoutput.txt &
PID=$!
sleep 906
kill $PID
echo "done!"
cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 1000m64cpu.txt
rm -r topoutput.txt

top -b -d 1 >> topoutput.txt &
PID=$!
sleep 906
kill $PID
echo "done!"
cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 1000m512cpu.txt
rm -r topoutput.txt
```

```
62.
63. top -b -d 1 >> topoutput.txt &
64. PID=$!
65. sleep 906
66. kill $PID
67. echo "done!"
68. cat topoutput.txt | grep Cpu | cut -c 36-40 | nl > 1000m1500cpu.txt
69. rm -r topoutput.txt
70.
71. touch waittime.txt
72. while [[ "$var" -ne 1 ]]; do
73.     var=$(cat "wait")
74.     sleep 10
75.     echo "$(date) -- waiting..." >> /home/andreas/waittime.txt
76. done
77. echo "moving on, creating folder, adding rules."
78. amount="2"
79. path="norules"
80. if [[ -e $path ]] ; then
81.         i=0
82.         while [[ -e $path-$i ]] ; do
83.         let i++
84.         if [ $i -eq $amount ]; then
85.         echo "100rules"
86.          sh ./rules/iptables-100rules.sh
87.         fi
88.         done
89.         path=$path-$i
90. fi
91. echo $path
92. if [ "$path" = "norules-2" ] ; then
93. w=0
94. path=100rules
95.         while [[ -e $path-$w ]] ; do
96.                 let w++
97.             if [ $w -eq $amount ]; then
98.             echo "500rules"
99.             sh ./rules/iptables-500rules.sh
100.                 fi
101.               done
102.      path=$path-$w
103.      fi
104.      echo $path
105.
106.      if [ "$path" = "100rules-2" ] ; then
107.      n=0
108.      path=500rules
109.
110.              while [[ -e $path-$n ]] ; do
111.                      let n++
112.                  if [ $n -eq $amount ]; then
113.                  echo "1000rules"
114.                  sh ./rules/iptables-1000rules.sh
115.                  fi
116.              done
117.      path=$path-$n
118.      fi
119.
120.      if [ "$path" = "500rules-2" ] ; then
121.      wq=0
122.      path=1000rules
123.
124.              while [[ -e $path-$wq ]] ; do
125.                      let wq++
126.                              if [ $wq -eq $amount ]; then
127.                              echo "2000rules"
```

```
                                sh ./rules/iptables-2000rules.sh
                                fi
                done
        path=$path-$wq
        fi

        if [ "$path" = "1000rules-2" ] ; then
        dq=0
        path=2000rules

                while [[ -e $path-$dq ]] ; do
                        let dq++
                                if [ $dq -eq $amount ]; then
                                echo "4000rules"
                                sh ./rules/iptables-4000rules.sh
                                fi
                done
        path=$path-$dq
        fi

        if [ "$path" = "2000rules-2" ] ; then
        aq=0
        path=4000rules

                while [[ -e $path-$aq ]] ; do
                        let aq++
                done
        path=$path-$aq
        fi

        echo "0" > wait
        echo $path
        mkdir $path
        mv *.txt $path
```

Bilaga H

```
1.  ssh root@10.10.1.1 "bash cpurealdeal.sh && exit" &
```

Bilaga I – Powershell

```
$configFiles = Get-ChildItem . *.txt. -rec
foreach ($file in $configFiles)
{
    (Get-Content $file.PSPath) |
    Foreach-Object { $_ -replace "\.", "" } |
    Set-Content $file.PSPath
}
```