En serverbaserad skyddsalgoritm mot distribuerade överbelastningsangrepp

(HS-IKI-EA-04-104)

Erik Josefsson (c99erijo@student.his.se)

Institutionen för kommunikation och information (IKI)
Högskolan i Skövde, Box 408
S-54128 Skövde, SWEDEN

Examensarbete på programmet för systemprogrammering under vårterminen 2004.

Handledare: Jonas Mellin

En serverbaserad skyddsalgoritm mot distribuerade överbelastningsangrepp

Examensrapport inlämnad av Erik Josefsson till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information (IKI).

2004-06-07

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

En serverbaserad skyddsalgoritm mot distribuerade överbelastningsangrepp

Erik Josefsson (c99erijo@student.his.se)

Sammanfattning

Till följd av att användningen av Internet har ökat kraftigt de senaste åren har många företag etablerat sig med webbplatser på detta medium. Ett av hoten mot webbservrar är distribuerade överbelastningsangrepp vilka kan skada ett företags rykte genom att göra deras webbplats otillgänglig. Angreppen har som mål att översköljа en webbserver med data och därmed göra den oanträffbar för andra användare. Det är svårt att skydda sig mot ett angrepp av det här slaget eftersom datamängden är mycket stor och den normala trafiken får svårt att nå webbservern. Målsättningen i detta projekt är att utveckla och undersöka en skyddsmetod som kan användas på en webbserver för att skydda den mot överbelastningsangrepp. Resultatet av utförda simuleringar och en jämförelse med en liknande metod visar att den framtagna skyddsmetoden fungerar som det är tänkt i en simulerad miljö och är konkurrenskraftig.

Nyckelord: Distribuerade överbelastningsangrepp, serverbaserad, nätverkssäkerhet, Internethot, webbserver, world wide webb, TCP/IP.

Tack till

Jag vill tacka min flickvän och min familj för allt stöd under arbetet med detta examens projekt. Ett stort tack till min handledare Jonas Mellin och min examinator Henrik Engström för deras kommentarer som har varit till stor hjälp för mig under arbetets gång.

Innehållsförteckning

1 Introduktion 1

2 Bakgrund 3

2.1 Existerande attackprogram 4

2.2 Svagheter i TCP/IP protokollet 5

2.3 Framtagna skyddsmetoder 7

2.3.1 Routerbaserade skyddsmekanismer 10

2.3.2 Serverbaserade skyddsmekanismer 12

2.3.3 Router och serverbaserade skyddsmetoder 13

3 Problembeskrivning 16

3.1 Problemprecisering 16

3.2 Mål 16

4 Metod 17

4.1 Algoritmens uppbyggnad 17

5 Simuleringsresultat 24

5.1 Implementation av SeFFiM 28

5.2 Simuleringsresultat 29

5.3 Relaterat arbete 36

6 Slutsats 37

6.1 Framtida arbete 38

7 Diskussion 40

Referenser 42

Appendix A – Grafer

Appendix B – Källkod till simulering

Appendix C – Källkod till SeFFiM

1 Introduktion

På senare år har Internet växt till ett världsomspännande nätverk med en oerhörd mängd datorer som är uppkopplade och delar information. Många företag har numera en stor del av sin verksamhet kopplat till detta stora nätverk. Denna verksamhet är beroende av att deras webbservrar och Internetuppkoppling fungerar.

Före år 2000 så var enligt McClure, Scambray och Kurtz (2003) distribuerade överbelastningsangrepp mera en teoretisk idé än en praktisk tillämpning. I februari 2000 blev ett flertal stora webbplatser angripna och detta ledde till att de blev oåtkomliga. Exempel på webbplatser som blev utsatta för dessa angrepp var yahoo.com, eBay.com och cnn.com. Enligt McClure mfl. (2003) verkade det som dessa angrepp inte var utförda av en grupp kunniga hackare, utan av vanliga användare med tillgång till gratisprogram, skrivna av erfarna hackare. Dessa program som enligt McClure m.fl. (2003) finns i flera olika varianter, är inriktade på att skicka en mängd data till den angripna värddatorn i syfte att överbelasta den, så att den blir otillgänglig för andra användare.

När ett distribuerat överbelastningsangrepp pågår uppkommer ett antal problem. Det huvudsakliga problemet är att den angripna datorn blir oåtkomlig. Ett annat av problemen är enligt Lan, Hussain och Dutta (2003) att nätverket mellan de datorer som medverkar i angreppet och den dator som är angripen blir överbelastat. Detta gör att normal trafik kan bli försenad eller hindrad från att komma fram, även om varken mottagare eller sändare drabbas av, eller medverkar i det pågående angreppet. Ytterligare ett problem är enligt Lan m.fl. (2003) att de domännamnservrar (eng. domain name services, DNS), som används för att översätta domännamnet till IP-nummer blir överbelastade. Detta gäller i de fall då angreppet riktar sig mot en domän och inte en specifik IP-adress. I analyser av verkliga angrepp har Lan m.fl. (2003) visat att väntetiden för en DNS förfrågan, den tid det tar från det att förfrågan har skickats till dess svar kommer fram till användaren, ökade i medeltal med 230 %.

För att dessa angrepp ska kunna startas behövs ett stort antal datorer som medverkar. En hackare kan samla ihop dessa på ett flertal olika sätt. Ett sätt är att använda en mask eller en trojansk häst. Dessa koncept finns beskrivna i kapitel 2.1. Trots att det finns en, potentiellt sett, mycket stor risk med att bli angripen när en dator kopplas upp på Internet så finns det enligt Lejeune (2002) miljoner datorer som helt saknar skydd, såsom brandvägg och antivirusprogram. En hackare kan bryta sig in i dessa datorer och installera ett program som sedan kan användas i överbelastningsangrepp.

Den typ av överbelastningsangrepp mot webbservrar som beskrivs i denna rapport har sin grund i TCP/IP protokollets uppbyggnad. Protokollet har enligt McClure m.fl. (2003) ett antal svagheter och en av dessa är hur en anslutning mellan två datorer skapas. Enligt Bellowin (1989) finns en möjlighet att skicka ett paket med en förfalskad IP-adress. I kapitel 2.1 finns problemen med TCP/IP protokollet beskrivna mera utförligt.

Ett koncept som är framtaget av Laprie (1994) behandlar pålitlighet hos system och dess relevans till skyddsmetoder mot distribuerade överbelastningsangrepp beskrivs i kapitel 2.3. Återstoden av kapitel 2.3 tar upp ett antal faktorer som kan påverka valet av skyddsmekanism och en modell som klassificerar skyddsmetoderna utifrån deras lokalitet. Ett par skyddsmetoder från varje klass beskrivs och deras respektive för- och nackdelar presenteras.

1. Introduktion

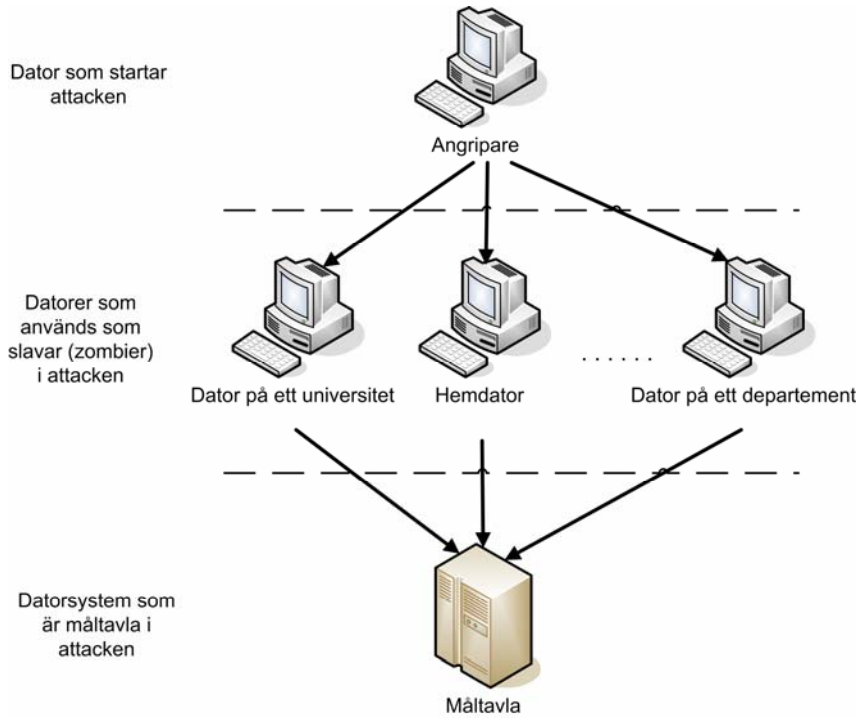
Problemet som behandlas i denna rapport består i att utveckla och undersöka en skyddsmetod som kan användas för att upprätthålla servicen från en webbserver[1] under ett distribuerat överbelastningsangrepp. Algoritmen som ligger till grund för skyddsmetoden presenteras och beskrivs ingående i kapitel 4. Skyddsmetoden är serverbaserad och har givits namnet **Se**rverbaserad **F**ullständig **Fi**ltrerings**m**etod (SeFFiM). Undersökningen av metodens egenskaper består av ett antal simuleringar. För att jämföra resultaten med liknande skyddsmetoder, genomförs också simuleringar med en annan skyddsmetod, ProbQueue, som utvecklats av Voorhies, Hyunyoung, och Klappenecker (2003). Det som skiljer de två undersökta metoderna åt, är att ProbQueue använder sig av en router som en del av skyddet samt har en slumpfunktion för att klassificera användare. SeFFiM är serverbaserad och klassificerar användare utan slumpfunktion. Framtida trafik från en angripare blockeras medan övrig trafik skickas vidare av metoden till webbserverapplikationen. Ytterligare en metod, DropTail, simuleras. Denna metod har inget inbyggt skydd mot distribuerade överbelastningsangrepp utan visar hur en oskyddad webbservers egenskaper förändras under ett angrepp.

Resultatet av undersökningen presenteras i kapitel 5. De visar hur responstiden (pingvärdet) ändras när ett angrepp sker på webbservern. Dessutom presenteras hur webbserverns bandbredd utnyttjas av angripare respektive vanliga användare före och under angreppet. Diagrammen visar att SeFFiM fungerar som det är tänkt och står sig bra i en jämförelse med ProbQueue. I ett verkligt scenario är det inte säkert att skyddsmetoden fungerar lika bra och detta diskuteras vidare i kapitel 6 och 7.

[1] I denna rapport avses termen webbserver som datorn eller de datorer som webbserverapplikationen körs på.

2 Bakgrund

Detta kapitel inleds med ett exempel hur ett generellt angrepp kan gå till och en definition av begreppet distribuerad överbelastningsangrepp. Dessutom presenteras forskning som lett fram till ett antal olika skyddsmetoder mot distribuerade överbelastningsangrepp.



Figur 1 - Exempel på ett distribuerat överbelastningsangrepp. Liknande bild återfinns i McClure m.fl. (2003)

Figur 1 beskriver hur ett överbelastningsangrepp kan gå till. En angripare bryter sig in i ett antal datorer och installerar ett klientprogram som sedan används i angreppet som s.k. slavar. Enligt McClure m.fl. (2003) kan angreppet startas mot ett utvalt mål på en given signal från attackprogrammet. Antalet slavar som används i ett angrepp är godtyckligt. Ur en angripares synvinkel så är dessa helst utspridda över många olika nätverk, eftersom detta gör angriparna svårare att spåra. Detta beror enligt Peng, Leckie och Ramamohanarao (2002b) på att den totala mängden trafik som deltar i angreppet är relativt liten från varje enskilt nätverk i jämförelse med respektive nätverks totala trafik vid en hög distribution av angripande datorer.

Följande citat ger en precis beskrivning av vad ett överbelastningsangrepp (eng. Denial of service (DoS) attack) innebär:

> "Denial of Service (DoS) is an attack designed to render a computer or network incapable of providing normal services. The most common DoS attacks will target the computer's network bandwidth or connectivity. Bandwidth attacks flood the network with such a high volume of traffic, that all available network resources are consumed and legitimate user requests can not get through. Connectivity attacks flood a computer with such a high volume of connection requests, that all available operating system resources are consumed, and the computer can no longer process legitimate user requests." (Stein och Stewart (2002))

Som citatet ovan beskriver, så går ett överbelastningsangrepp ut på att överskölja en dator eller nätverk med data så att dessa blir oanträffbara.

2.1 Existerande attackprogram

McClure m.fl. (2003) och Kargi, Maier & Weber (2001) beskriver ett antal program som kan användas i syfte att utföra ett överbelastningsangrepp. Exempel på dessa är Tribe Flood Network (TFN), Trinoo, Stacheldraht och TFN2K. Alla dessa är byggda på en liknande grundidé. De består av en serverdel och en klientdel. Serverdelen används av angriparen och klientdelen installeras på slavarna. Det finns tre sätt att installera klientdelen: (*i*) angriparen bryter sig in på en dator och installerar klientdelen, (*ii*) en nätverksmask sprider klientdelen, och (*iii*) en trojansk häst installeras av den som administrerar datorn. Dessa metoder finns beskrivna längre ner i detta kapitel. Klientprogrammet består av en demon som körs på den angripne klientdatorn utan användarens vetskap. Demonen ligger som en process i bakgrunden och lyssnar på en viss port efter inkommande meddelanden från serverprogrammet. Vilken port som används och hur serverprogrammet kommunicerar med klientprogrammen varierar.

Ett antal sätt att skydda sig mot dessa program beskrivs av McClure m.fl. (2003). Det mest uppenbara är att förhindra att en angripare kan bryta sig in på systemet och installera klientprogrammet. Detta kan göras genom att gå igenom ett antal steg som beskrivs i McClure m.fl. (2003). De går ut på att stänga av olika tjänster och stänga de portar som inte används på systemet. Dessa motåtgärder skyddar dock bara mot att den aktuella datorn inte används i ett angrepp och inte som ett skydd mot ett angrepp utifrån.

Ett sätt att bryta sig in i en dator och göra den till en slav i ett distribuerad överbelastningsangrepp är att, som tidigare nämnts, använda sig av en mask eller trojansk häst. En mask är enligt Russel och Gangemi (1991) ett oberoende program som sprider sig själv över ett nätverk, genom att kopiera sig från dator till dator. Till skillnad från ett virus är inte en masks mål att förstöra data utan de används i syfte att belasta det angripna nätverket eller att ge en viss användare tillgång till nätverkets resurser.

En trojansk häst är en annan väg för en angripare att få kontroll över ett system. Enligt Russel m.fl. (1991) är en trojansk häst ett program som döljer sig inne i ett annat, i

syfte att vilseleda en användare att köra programmen i tron att programmet enbart gör någonting meningsfullt. I många fall utför programmet det som användaren förväntar sig men aktiverar också samtidigt trojanen. Trojanen kan sedan öppna datorn för angriparen så att denne kan installera ett klientprogram till ett distribuerad överbelastningsangrepp som beskrevs tidigare i detta kapitel.

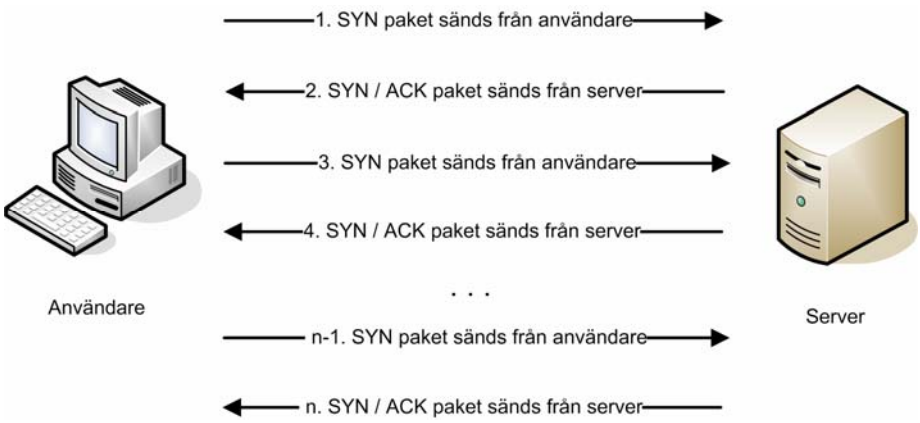
2.2 Svagheter i TCP/IP protokollet

De distribuerade överbelastningsangreppen använder svagheter i TCP/IP protokollet för att uppnå sitt mål att göra en tjänst obrukbar. En av dessa svagheter ligger i handskakningsprocessen. En annan svaghet i protokollet är att det inte finns någon inbyggd kontroll på att rätt avsändaradress skrivs in i ett paket. Ett tredje problem är att TCP/IP bygger på en tillståndslös (eng. stateless) modell, ett problem som beskrivs längre ner.

För att upprätta en anslutning mellan en användare och en server så använder TCP/IP protokollet av en trestegs handskakningsprocedur som illustreras i Figur 2.

Figur 2 - Exempel på TCPs handskakningsprocedur (McClure m.fl. 2003)

När en ny anslutning ska upprättas skickar användaren ett synkroniseringsmeddelande (SYN) till en server. När paketet mottas på servern skickas ett synkroniserings- och bekräftelsemeddelande (SYN/ACK) till användaren. Därefter väntar servern på en bekräftelse (ACK). Efter denna procedur är klar är anslutningen upprättad. Anslutningen existerar tills en av parterna avslutar förbindelsen eller något annat fel inträffar som orsakar att anslutningen avbryts. Att avsluta förbindelsen fungerar ungefär på samma sätt som handskakningen (Wang, Zhang och Shin 2002). Enligt Forouzan, (2003) så skickas alla förfrågningar i Figur 2 som TCP paket.



Figur 3 - Exempel på en handskakningsprocedur under ett överbelastningsangrepp

Figur 3 illustrerar hur ett överbelastningsangrepp kan gå till. De två första stegen i handskakningsprocessen går till som beskrevs i Figur 2. Skillnaden är att steget när angriparna skickar ett paket med bekräftelsen tillbaka till servern inte utförs. Enligt Wang m.fl. (2002) så har en server en buffert för inkommande anrop, så att flera kan tas om hand samtidigt. När denna buffert är full så blir ytterligare SYN-anrop till servern nekade. Eftersom angriparen i Figur 3 inte skickar paketet med bekräftelsen så ligger dennes begäran kvar i serverns buffert tills en viss tid har passerat. Denna tid varierar mellan servrar men enligt Wang m.fl. (2002) är den vanligtvis 75 sekunder. På detta sätt fylls serverns buffert snabbt med oavslutade anslutningar och blir obrukbar för andra användare.

Ett annat problem med TCP/IP protokollet, som nämndes tidigare, är att det inte finns någon kontroll på protokollnivå att rätt avsändaradress skrivs in i pakethuvudet när detta skickas. Enligt Bellowin (1989) är det sändarens ansvar att sätta in rätt avsändaradress i TCP/IP paketet. Detta kan missbrukas genom att en fingerad adress sätts in istället för den riktiga. Detta försvårar enligt Eronen (2000) möjligheten att spåra var ett paket kommer ifrån och därmed även möjligheten att stoppa ett framtida angrepp när en falsk IP-adress används.

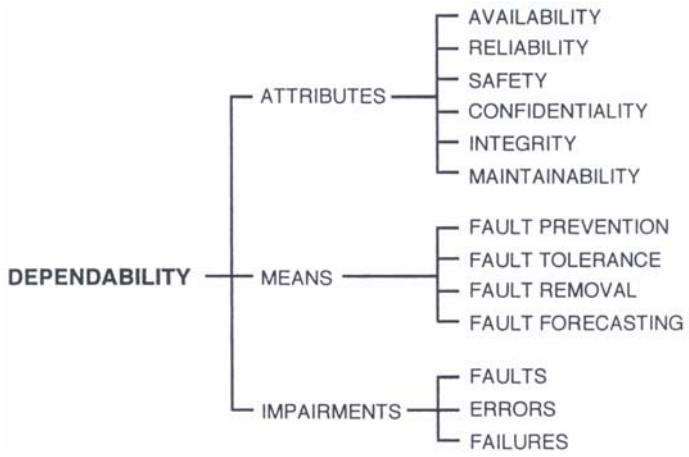
Ett sätt att ta reda på om ett paket kommer ifrån en dator som har förfalskat avsändaradressen på detta sätt presenteras av Xu och Lee (2003). Denna metod använder sig av en hemlig nyckel som tas fram av bitar ur IP-numret och portnumret. Nyckeln delas sedan mellan olika routrar. För att stoppa ett paket med en förfalskad IP-adress så skickas ett meddelande tillbaka till paketets källa. Om källan är legitim kan IP-adressen svara på detta meddelande och ytterligare trafik från detta IP-nummer kan skickas. Om källan skulle vara förfalskad får den inte något meddelande att svara på, eftersom meddelandet som kräver svaret inte kommer fram till rätt IP-adress och ingen nyckel räknas fram. Att paketet inte kommer fram till rätt IP-nummer beror enligt Xu m.fl. (2003) på att paketets MAC[2] adress inte stämmer när IP-numret är förfalskat. På detta sätt kan de routrar som delar på nycklarna räkna fram vilka paket som har förfalskade IP-adresser och kassera dessa.

[2] Media access control (MAC): Fysisk adress hos en nätverksenhet. (Parker och Sportack 2000)

Ytterligare ett problem med TCP/IP protokollet är enligt Ye (2002) att det baseras på en tillståndslös (eng. stateless) modell samt att modellen är baserad på en ”efter bästa förmåga” (eng. ”best-effort”) filosofi. Detta innebär, till exempel, att en router inte har kontroll över hur många paket den skickar till en viss mottagare. Routern skickar så många av de inkommande den kan till servern oavsett serverns belastning. Detta gör enligt Ye (2002) Internet mycket sårbart för överbelastningsangrepp. Vidare menar Ye (2002) att den tillståndslösa modellen fungerar bäst i mindre nätverk, inte ett världsomspännande nätverk som Internet är idag. Om TCP/IP istället skulle vara baserat på en tillståndstyrd (eng. stateful) modell skulle inte problemet med överbelastningsangrepp vara lika stort. Då skulle det enligt Ye (2002) kunna finnas en form av resurshanteringssystem. En resurs på Internet, till exempel en webbserver, skulle inte kunna användas utöver sin kapacitet. Systemet skulle kunna likna de resurshanteringssystem som bl.a. återfinns i en dators hårdvara (Galvin 1999) och i vissa databassystem (Elmasri och Navathe (2000).

2.3 Framtagna skyddsmetoder

En skyddsmetod är tänkt att öka pålitligheten (eng. dependability) och därmed också tillgängligheten hos en server. Enligt Laprie (1994) kan begreppet pålitlighet delas upp i olika delar som Figur 4 illustrerar. Denna modell är främst tänkt för programutveckling och testning av program men modellen kan även användas till andra ändamål som till exempel i detta fall för att indela skyddsmetoder mot distribuerade överbelastningsangrepp beroende på deras funktionalitet.



Figur 4 - Uppdelning av begreppet pålitlighet (Laprie 1994)

Det främsta målet för skyddsmetoden är att uppnå tillgänglighet (eng. availability) hos webbservern. För att uppnå detta kan enligt Laprie (1994) någon av de metoder (eng. means) som visas i Figur 4 användas.

Den metod som används av samtliga skyddsmetoder som presenteras i denna rapport är att hantera felet när det uppstår (eng. fault tolerance) och det innebär i detta fall att metoden byggs så att den klarar att upprätthålla serverns funktion, även under ett överbelastningsangrepp. Att hantera felet kostar mest systemresurser men är den enda metodtyp som är möjlig med de protokoll som används idag på Internet. En alternativ

metod är att förebygga att felet uppstår (eng. fault prevention). Detta uppnås genom att någon av de krav som ställs för att felet ska uppstå byggs bort ur systemet. Metoden är mycket kostnadseffektiv eftersom den inte behöver köras hela tiden och systemets resurser kan användas till annat än att kontrollera om ett fel har uppstått och hantera felet under körning. För att uppnå detta kan tekniker som används för förebyggande av att dödlägen (eng. deadlocks) inträffar i systemet användas (Galvin 1999).

En annan lösning kan vara att hindra att felet uppstår (eng. fault avoidance) Det innebär i detta fall att en metod används som hindrar att överbelastningsangreppet startar. En sådan metod kan inte används i dagsläget eftersom det troligen krävs någon form av säkerhetskontroll av varje paket som till exempel verifiering (eng. authentication). Enligt Halsall (2001) finns verifieringssystem i dagsläget för digital signering av bland annat e-post och det är tänkbart att liknande system också kan användas på paketnivå för att hindra överbelastningsangrepp.

Eftersom distribuerade överbelastningsangrepp är förödande mot webbservrar pågår det mycket forskning inom området. Det finns tre större delproblem. (*i*) att upptäcka att ett angrepp pågår, (*ii*) att hålla webbplatsen tillgänglig under ett angrepp och (*iii*) att spåra de datorer som står bakom angreppet.

Innan något annat kan göras mot ett överbelastningsangrepp så måste det upptäckas. Trots att det är möjligt att hålla en skyddsmetod mot angreppen kontinuerligt i drift, så är det inte önskvärt, eftersom alla skyddsmetoder tar extra resurser i anspråk. Dessa resurser är värdefulla för företagen som driver webbplatser för att kunna använda till att leverera webbsidor åt kunder.

När angreppet pågår och är upptäckt, uppkommer ett annat problem. Webbplatsen är kanske livsviktig för företaget som driver den och för att upprätthålla den och leverera webbsidor till (potentiella) kunder så måste den även fungera under ett överbelastningsangrepp.

För att stoppa framtida överbelastningsangrepp måste de som är ansvariga för angreppet kunna spåras. Om detta kan göras på ett effektivt sätt så att den som satte igång angreppet kan hittas så kan åtgärder vidtas av dess Internetleverantör. Exempel på åtgärder skulle kunna vara att stänga av angriparens Internetuppkoppling eller vidta rättsliga åtgärder. Detta kan kanske avskräcka personer från att sätta igång angrepp i framtiden. Problemet med trojaner och maskar som är förprogrammerade att utföra ett angrepp vid en viss tidpunkt kvarstår dock. Detta beror på att, om ett angrepp härrör från en mask är inte upphovsmännen bakom masken aktivt deltagande i angreppet eftersom en mask agerar på egen hand efter ett förprogrammerat mönster.

När ett företag bestämt sig att installera någon form av skydd mot distribuerade överbelastningsangrepp så finns ett antal olika faktorer som påverkar valet av skyddsmetod.

- **Installerbarhet**

  För att en skyddsmetod ska användas i större skala hos slutanvändaren är det viktigt att den är enkel att installera och underhålla.

- **Systemkrav**

  Metoden ska helst kräva så lite resurser av systemet som möjligt. Om det är en serverbaserad lösning så ska merparten av serverns resurser vara tillgängliga för användarna av servern, även under ett angrepp. Därför är det ett krav att

algoritmen kräver minimalt med processorkraft och minne. Det är önskvärt att skyddsmetoden inte är igång hela tiden, utan bara när ett angrepp pågår, för att spara så mycket resurser som möjligt.

- **Administrerbarhet**

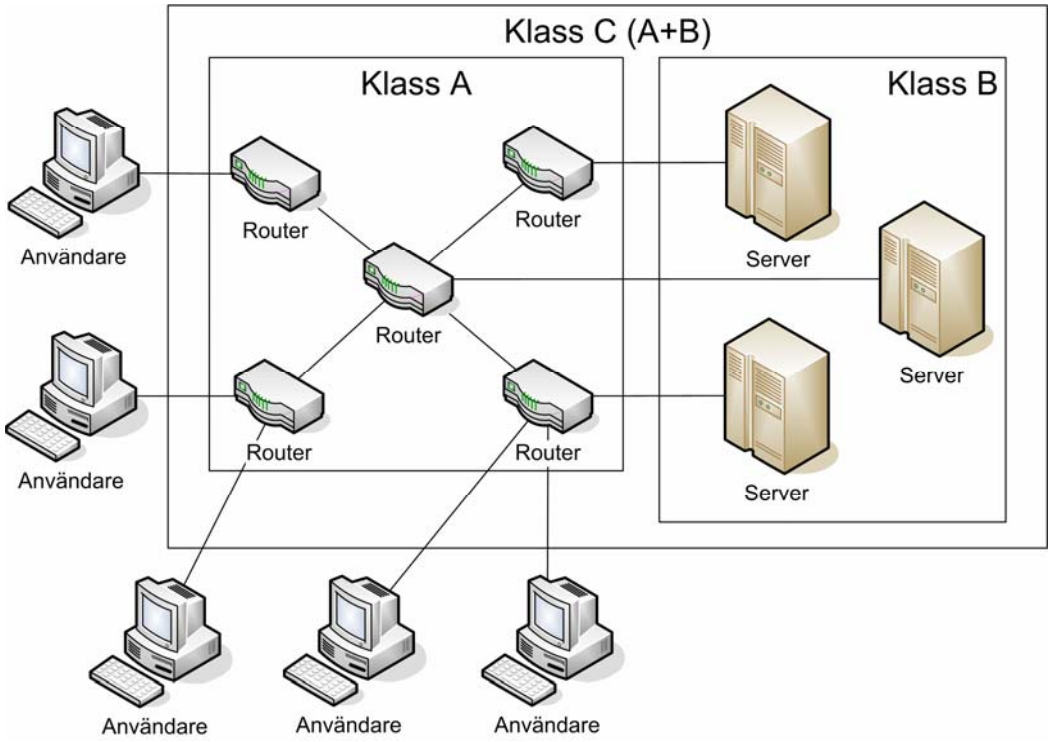
Metoden ska helst vara automatisk och inte kräva manuell övervakning, eftersom detta underlättar för administrationen av webbservern. Detta kräver att det finns en process som på något sätt upptäcker ovanligheter i nätverkstrafiken som kan tyda på att en angrepp är pågående.

- **Effektivitet**

Metoden ska hindra en majoritet av angriparens trafik samtidigt som den ska hindra en minoritet av de vanliga användarnas trafik

- **Kostnad på utrustningen**

Det är önskvärt att skyddsmetoden går att använda på en webbserver utan att ytterligare investeringar i form av ny hårdvara behövs.



Figur 5 - Lokalitet på skyddsmekanismer

I Figur 4 visas en förenklad bild av hur en nätverkstopologi kan se ut. Bilden beskriver en klassificering och lokalitet, som tagits fram i samråd med J. Mellin (personlig kontakt 24 mars, 2004), av olika skyddsmetoder mot överbelastningsangrepp. I klass A ingår de skyddsmetoder som är baserade på enbart routrar och i klass B ingår de skyddsmetoder som enbart körs på servrar som är mål i angreppet. Klass C är en hybrid variant av klass A och B. Dessa skyddsmetoder har en del på webbservern och en annan del av skyddet på en eller flera routrar. I de tre kategorierna har skyddsmetoder tagits fram som alla har både för- och nackdelar.

2.3.1 Routerbaserade skyddsmekanismer

En routerbaserad skyddsmetod mot distribuerade överbelastningsangrepp innebär att en eller flera algoritmer implementeras och körs på en programmerbar router. Routern kan i vissa fall även samarbeta med andra routrar utanför företagets egna nätverk.

Fördelar:

- En routerbaserad lösning är effektiv eftersom många routrar kan samarbeta och kassera paket nära angripare.
- De motverkar att företagets interna nätverk blir fyllt med trafik under ett angrepp eftersom paketen tas bort innan de kommer in på nätverket.
- Belastar inte webbservern eftersom filtreringen sker på routern.

Nackdelar:

- En routerbaserad lösning går inte att implementera på alla routrar eftersom inte alla routrar är programmerbara och det kan krävas att utrustningen på företaget uppgraderas för att skyddsmetoden skall kunna användas.
- Vissa skyddsmetoder kräver hjälp av routrar som inte företaget har kontroll över.
- Vissa skyddsmetoder kräver att information lagras i IP-paketets huvud för att vara effektiva. Denna överlagring gör att vissa delar av protokollets funktioner inte längre kan användas som vanligt.
- En routerbaserad lösning måste dessutom ta större hänsyn till processor- och minnesåtgång än en serverbaserad dito, eftersom en router troligtvis har en mer begränsad tillgång till resurser av denna typ än en webbserver.

En routerbaserad skyddsmetod för att upptäcka överbelastningsagrepp föreslås av Peng, Leckie och Ramamohanarao (2002a). Skyddsmetoden analyserar de inkommande paketen och övervakar antalet IP-nummer. När en plötslig ökning i antalet IP-nummer sker så varnar metoden att ett angrepp kan vara pågående. Den skyddsmetod som presenteras av Peng m.fl. (2002a) kan generera en stor mängd felaktiga varningar, eftersom den bygger på en ökning i antalet IP-adresser.

En metod för att hålla en webbserver åtkomlig under ett överbelastningsangrepp har föreslagits av Xu m.fl. (2003). Den går ut på att införa en mekanism som begränsar bandbredden för anslutna datorer. Om en användare använder för mycket av routerns bandbredd anses den vara en angripare och får då en minimal tillgång till bandbredden. På detta sätt minskar skadan från en angripande. För att ytterligare assistera webbservern under en ett angrepp så används routrar längs vägen till den angripna datorn för att kassera paket som används i angreppet före de når sitt mål. Det skydd som presenteras av Xu m.fl. (2003) behöver endast små mängder resurser hos routern eftersom den tar hänsyn till antalet IP-nummer istället för den totala mängden paket som skickas. En nackdel med skyddsmetoden är att skyddet måste installeras på en mängd routrar, även de utanför företaget, för att vara effektiv.

Tanachaiwiwat och Hwang (2003) introducerar ett koncept kallat ”sustained quality of service”. Med detta menas att de normala användarna ska få så bra kvalitet på servicen från till exempel en webbserver som möjligt. För att uppnå detta så presenterar Tanachaiwiwat m.fl. (2003) en routerbaserad skyddsmetod som filtrerar nätverkstrafiken och reagerar när en hastig ökning i antalet paket inträffar. Skyddsmetoden använder sig av tre olika prioritetsklasser, säker, normal och riskfylld

samt en räknare för att hålla reda på ett IP-nummers rykte vilket lagras i en tabell tillsammans med de IP-nummer som aktiva användare av routern. IP-numrets rykte beskriver hur stor sannolikheten är att ett IP-nummer är angripare. Om trafiken från ett visst IP-nummer verkar misstänkt så sänks IP-numrets rykte stegvis. Misstänkt trafik kan vara trafik från ett förfalskat IP-nummer eller ett IP-nummer som sänder en onormalt stor mängd paket.

Peng, Leckie och Ramamohanarao (2003) presenterar en routerbaserad skyddsmetod som använder sig av historiken över de IP-nummer som besökt webbservern tidigare. Under ett angrepp används denna historik för att kontrollera om ett visst IP-nummer är tillåtet på webbservern eller inte. Skyddsmetoden aktiveras då trafiken blir så stor på servern så att paket måste kasseras. Ett problem med detta är att en ökning i trafiken på detta sätt kan inträffa även om servern inte är utsatt för ett angrepp. Detta benämner Peng m.fl. (2003) som en ”flash crowd” och kan inträffa då en nyhetswebbplats går ut med en stor nyhet som lockar till sig en stor mängd besökare. Enligt Peng m.fl. (2003) har ofta de som besöker webbplatsen under en ”flash crowd” varit besökare tidigare (ca 80 %) och är med i historiken som algoritmen använder. De IP-adresser som utgör angripare har oftast inte varit besökare på webbservern tidigare. Enligt Peng m.fl. (2003) är andelen tidigare besökare som agerar angripare ca 0,6 - 14 %. I experiment som Peng m.fl. (2003) utfört tränades en webbserver med trafik som utgjordes av verklig data från Aucklands Universitets Internetförbindelse, för att lära sig vilka IP-nummer som var frekventa besökare på webbservern. Sedan gjordes ett angrepp mot webbservern. Med denna data var algoritmen effektiv och kunde skydda 90 % av den legitima trafiken med en minnesåtgång på 4MB.

Ett problem med skyddsmetoden enligt Peng m.fl. (2003) är att det finns en möjlighet för en angripare att gå runt skyddet genom att besöka webbservern med adresserna som är tänkta att delta i angreppet innan det inleds. På detta sätt så läggs adresserna till i historiken och paket från dessa släpps sedan igenom under ett angrepp. Ytterligare ett sätt att vilseleda skyddsmetoden är att hålla mängden trafik under gränsen för att skyddsmetoden ska ingripa mot angreppet.

En skyddsmetod att spåra de datorer som medverkar i angreppet presenteras av Savage, Wetherall, Karlin, & Anderson (2000). Idealet enligt Savage m.fl. (2000) är om det går att spåra den som utför angreppet istället för de slavdatorer som medverkar. Detta är dock ett större problem än att spåra de som medverkar direkt. Eftersom det finns en mängd sätt för en angripare att gömma sin identitet på. Exempelvis genom kedjor av falska konton på oskyldiga datorer och att använda sig av andra datorer för att vidarebefordra meddelanden. Ett problem med att spåra en dator är att IP-adressen kan vara falsk. Metoden som presenteras av Xu m.fl. (2003) fungerar inte om datorn ska spåras, bara om den ska hindras från att skicka paket till måldatorn. Den skyddsmetod som föreslås av Savage m.fl. (2000) har flera fördelar. En av den är att den inte kräver aktiv medverkan av någon Internetleverantör eftersom den går att göra automatiskt. En annan fördel är att den inte genererar överflödig trafik på nätverket och att den kan göras i efterhand, när angreppet är över.

Skyddsmetoden som presenteras av Savage m.fl. (2000) bygger på att märka paket med adresserna på de routrar som paketet passerar. Problemet med skyddsmetoden är att adresserna till de olika routrar som paketet passerar måste lagras någonstans. Savage m.fl. (2000) har valt att lagra dessa i IP-paketets huvud där information om fragment av IP-paket i vanliga fall lagras. Savage m.fl. (2000) noterar att detta kan leda till kompatibilitetsproblem, om data i paketet är fragmenterad i flera paket, men menar att detta inte är något större problem eftersom studier visar att paket sällan

(0,25 %) är fragmenterade. Dessutom så har nyare implementationer av TCP/IP protokollet en funktion som motverkar uppdelning av paket. Savage m.fl.(2000) presenterar en lösning på problemet med det omskrivna IP-huvudet. En går ut på att inte markera paketfragment och en annan att ha flaggan ”don’t fragment” satt i paketet. Båda dessa metoder motverkar uppdelning av paket.

Ett sätt som paket från falska adresser kan filtreras ut på är att använda tillträdesfiltrering (eng. ingress filtering). Detta fungerar enligt Tupakula och Varadharajan (2003) så att en Internetleverantörs router kasserar alla inkommande paket som har adresser som inte finns i det nätverk paketen kommer ifrån. Vidare så fungerar utträdesfiltering (eng. egress filtering) så att den kasserar alla de utgående paketen från en Internetleverantörs nätverk som har adresser som inte finns i Internetleverantörens nätverk. Denna filtreringsmetod är resurskrävande eftersom varje paket kontrolleras. Dessutom klarar den inte av att filtrera paket från en förfalskad adress om adressen existerar i nätverket och IP-numret har tagits från en annan dator.

2.3.2 Serverbaserade skyddsmekanismer

En skyddsmetod som inte kräver inblandning av en router delas enligt figur 4 in i klass B. Detta innebär att det inte är nödvändigt att en router är inblandad i skyddet mot överbelastningsangreppet utan webbservern utför filtreringen av paket på egen hand.

Fördelar:

- Inga utomstående routrar måste delta i skyddet och installationen blir därmed förenklad.
- Datakraften och minnesutrymmet hos en webbserver är ofta större än hos en router och därmed kan mera avancerade algoritmer tas i bruk.

Nackdelar:

- Den största nackdelen med denna lösning är att trafiken från angreppet tar sig in på företagets interna nätverk. På grund av detta så är det få skyddsmetoder som enbart baserar sig på en server.
- Datakraft tas från användarna på webbservern och används istället för att motverka angreppet.

Ett förslag på en skyddsmetod för att upptäcka att en överbelastningsangrepp är beskriven av Wang m.fl. (2002). Artikeln föreslår en metod som baseras på statistik över antalet SYN och FIN[3] paket. Enligt författarna är det en stor överrepresentation av SYN-paket i nätverket när ett angrepp pågår. Detta eftersom ett angrepp går ut på att hålla så många anslutningar öppna på servern som möjligt, se figur 3. En fördel med denna metod är att den också kan vara en klass A metod. Denna router kan då ta hand om att upptäcka angreppet och samtidigt förhindra att det interna nätverket blir fyllt med trafik. Wang m.fl. (2002) ger ett exempel på en tillståndsbaserad mekanism. Den består av en brandvägg som är designad att stå emot ett överbelastningsangrepp.

[3] FIN paket används för att avsluta en anslutning i TCP/IP protokollet

Enligt experiment som Wang m.fl. (2002) beskriver så krävde denna brandvägg en stor mängd resurser och kunde inte klara att bearbeta mer än 14000 inkommande paket per sekund. Skyddsmetoden som presenteras av Wang m.fl. (2002) är tillståndslös och kräver därför ett minimum av resurser för att köras. Experiment, som Wang m.fl. (2002) redovisar visade att skyddsmetoden fungerade bra och har en stor träffsäkerhet när det gäller att upptäcka ett överbelastningsangrepp. En nackdel med denna skyddsmetod är att den är lätt att gå förbi för en inkräktare. Som Wang m.fl. (2002) noterar, så kan en angripare som har vetskap om att metoden används gå runt den genom att sända FIN paket och på så sätt jämna ut statistiken mellan SYN och FIN paket. När detta inträffar blir skyddsmetoden oanvändbar och tjänar faktiskt angriparens syfte genom att konsumera resurser på systemet vilket är målet för ett överbelastningsangrepp.

2.3.3 Router och serverbaserade skyddsmetoder

I en hybrid skyddsmetod som involverar både en router och en server klassificeras enligt Figur 4 som klass C. Dessa skyddsmetoder kräver att det finns kommunikation mellan webbservern och routern.

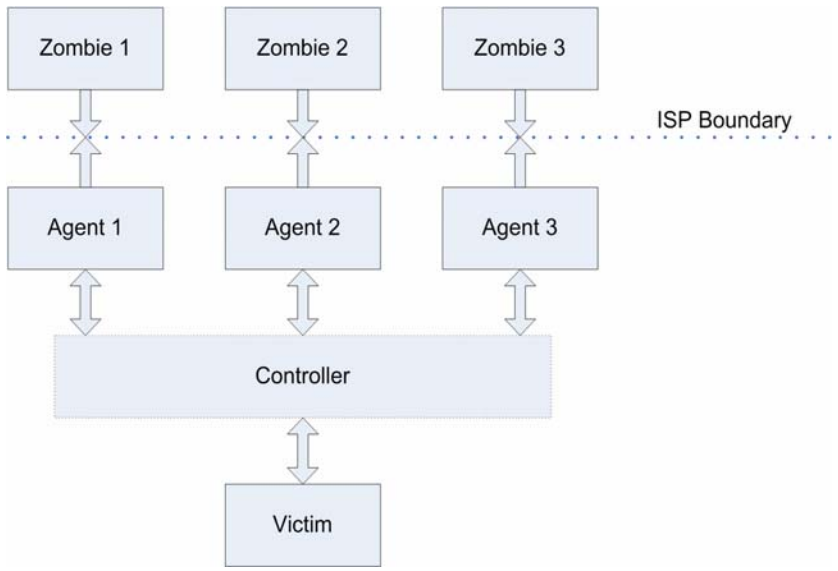
Fördelar:

- Att dela upp arbetet mellan server och router är avlastande för båda parter.
- Har fördelarna av en routerbaserad lösning, som ställts upp i kapitel 2.3.1 och fördelen att datakraften och minnesutrymmet hos en webbserver är ofta större än hos en router och därmed kan mera avancerade algoritmer tas i bruk under förutsättning att beräkningen görs på webbservern.

Nackdelar:

- Installationen är mer omfattande eftersom det är flera maskiner inblandade.
- Kan vara svårt att få alla delar i systemet att fungera tillsammans på ett bra sätt. En brist i kommunikationen mellan enheterna kan få skyddsmetoden att fallera.
- Har nackdelarna från kapitel 2.3.1 och att datakraft tas från användarna på webbservern och används istället för att motverka angreppet om webbservern används för beräkningar och analys av trafiken.

En hybridmetod för att hålla en webbserver åtkomlig under ett överbelastningsangrepp presenteras av Voorhies m.fl. (2003). Skyddsmetoden går ut på att övervaka hur mycket data en dator sänder från ett visst IP-nummer. Enligt Voorhies m.fl. (2003) är det troligt att en angripande dator skickar betydligt mera data till den utsatta webbservern än en normal användare gör. Skyddsmetoden väljer ut paket att behandla med en viss sannolikhet. De utvalda paketens IP-nummer läggs sedan i en lista. Andledningen till att endast vissa paket väljs ut, är att om alla pakets IP-nummer skulle läggas på listan så kräver detta enligt Voorhies m.fl. (2003) för mycket dataresurser eftersom mängden paket oftast är stor. Om en för stor mängd paket kommer från samma IP-adress så kasseras framtida paket från denna dator och dess öppna anslutningar stängs. En förutsättning för att denna metod ska fungera så krävs det enligt Voorhies m.fl. (2003) att mängden förfalskade IP-nummer är liten. Detta kan uppnås med hjälp av tillträdes- och utträdes filtrering.



Figur 6 - Representation av Control-agent modellen (Tupakula m.fl. 2003).

Figur 6 illustrerar en modell som presenteras av Tupakula m.fl. (2003). Den består av ett antal agenter som styrs med hjälp av en kontrollmodul. Agenterna är inbyggda i en eller flera routrar och dessa fungerar som vanliga routrar när ett angrepp inte pågår. När måldatorn upptäcker ett angrepp sänder denna ett meddelande till kontrollmodulen med en signatur som beskriver hur trafiken från angriparna ser ut. Kontrollmodulen signalerar i sin tur agenterna med signaturen. Med hjälp av signaturen så filtrerar agenterna trafiken innan den når fram till måldatorn. Signaturen tas fram i måldatorn genom att de inkommande paketen analyseras. Under analysen görs en sökning i IP-paketets huvud efter ett ID nummer i ID fältet. Detta nummer läggs till av andra routrar i andra nätverk, helst så nära de angripande datorerna som möjligt. För att spara plats i ID fältet används ett ID nummer istället för ett vanligt 32-bitars IP-nummer.

Ett alternativ till metoden som presenteras av Tupakula m.fl. (2003) för att filtrera ut paket från en förfalskad IP-adress presenteras av Cheng, Wang och Shin (2003). Den bygger på att jämföra TTL (Time to live) värdet i IP-paketets huvud med det verkliga antalet routrar ett paket har passerat. När ett paket passerar en router så räknas dess TTL-värde ner och på så sätt går det att ta reda på hur många routrar ett paket har passerat. Sedan lagras detta värde i en tabell. Tabellvärdet jämförs sedan med inkommande paket med samma IP-nummer. Om antalet routrar som paketet gått igenom inte överensstämmer med det lagrade värdet så anses paketet ha en falsk IP-adress. Värdet kan också överensstämma även om ett falskt IP-nummer används, men detta är enligt Cheng m.fl. (2003) inte så vanligt förekommande.

Metoden har två tillstånd, alert och aktivt. I det alerta tillståndet så kontrolleras paketen utan att de misstänkta kasseras. Om en plötslig ökning i antalet paket med förfalskade adresser inträffar byter algoritmen tillstånd och övergår i ett aktivt tillstånd. Då kasseras misstänkta paket om de upptäcks. Metoden klarar enligt Cheng m.fl. (2003) att upptäcka nästan 90 % av de paket med förfalskad IP-adress utan att andelen paket med legitim adress klassificeras felaktigt. Eftersom denna typ av förfalskade IP-adresser är vanliga i distribuerade överbelastningsangrepp kan en stor

del av trafiken som genereras av angriparna filtreras bort genom att en metod mot förfalskade IP-adresser används.

Att basera metoden på att andra routrar markerar paketen kan leda till en rad problem. Utöver de problem som finns med routerbaserade lösningar så finns en möjlighet att en angripande dator själv lägger till ett giltigt ID nummer i ID fältet. Då kan paketen från denna ändå komma fram till måldatorn. Ett annat problem med att lägga till extra information i IP-huvudet är att något annat inte får plats där. Överlagringen sker i fältet där information om fragmenterade paket normalt finns. I den nuvarande versionen av metoden som presenteras av Tupakula m.fl. (2003) så kasseras paket som är fragmenterade under ett angrepp även om de kommer från en dator som inte är angripare. Men eftersom detta sker enbart under ett angrepp är inte problemet så stort som det hade varit om skyddet hade varit igång utan att ett angrepp var pågående. Skyddet för måldatorn blir effektivt eftersom försvaret består av flera olika routrar och filtreringen av paketen enligt en signatur som dessutom kan varieras dynamiskt under pågående angrepp är en effektiv lösning, enligt Tupakula m.fl. (2003).

3 Problembeskrivning

I detta projekt ligger fokus på att ta fram och beskriva en serverbaserad skyddsmetod som kan användas på en webbserver, som är den dator som förser användare med webbtjänsten. Som beskrivits i kapitel 2 har problemet med distribuerade överbelastningsangrepp ingen trivial lösning. I valet av skyddsmetod får fördelar vägas mot nackdelar eftersom samtliga klasser av skyddsmetoder har olika fördelar och nackdelar. Kapitel 2 visar också att det finns en avsaknad av renodlade klass B metoder. För ett mindre företag med en webbserver som inte har tillgång till en programmerbar router skulle en serverbaserad skyddsmetod kunna vara en bra lösning. Valet att fokusera på en webbserver beror på att det är rimligt att anta att dessa är bland det mest utsatta på företagen. Vidare menar Badre (2002) att ett företags webbplats ofta är dess ansikte utåt på Internet. Om webbsidor som ligger på en webbserver skulle vara otillgänglig kan företagets rykte skadas.

3.1 Problemprecisering

Problemet består i att ta fram en algoritm som kan skydda en webbserver mot ett överbelastningsangrepp. Denna algoritm ska kunna stå emot ett angrepp från en stor mängd angripare. Detta kräver att algoritmen kan identifiera om en dator deltar i angreppet eller är en legitim användare av webbplatsen. Algoritmen ska kunna gå att applicera på en webbserver och ska fungera utan att ytterligare hårdvara i form av router behöver vara en del av skyddet. För att få ett mått på hur bra algoritmen klarar uppgiften kontrolleras om vanliga användare har möjlighet att koppla upp sig mot webbservern under ett pågående angrepp. För att bekräfta resultaten kan en jämförelse göras med andra liknande skyddsmetoders resultat. Dessutom är det önskvärt att implementationen av algoritmen konsumerar endast en begränsad mängd dataresurser.

3.2 Mål

Målet med den framtagna algoritmen är att den ska vara ett direkt applicerbart skydd som kan installeras på en webbserver i form av en modul och en utökning av nätverkshanteringen på webbservern. Algoritmen ska kunna ge ett tillräckligt bra skydd för att kunna hålla webbservern tillgänglig för vanliga användare under ett distribuerat överbelastningsangrepp.

4 Metod

För att undersöka hur den framtagna algoritmen fungerar i förhållande till liknande algoritmer och hur effektivt den skyddar mot ett distribuerat överbelastningsangrepp kan ett antal olika undersökningsmetoder användas. De metoder som har identifierats som tänkbara till detta projekt är att genomföra en matematisk analys av skyddsmetoden eller simulera och sedan analysera skyddsmetoden utifrån resultatet av simuleringen eller att genomföra en fallstudie.

Den metod som används av Tanachaiwiwat och Hwang (2003) är att genomföra en matematisk analys. Att analysera en metod matematiskt har nackdelar. Framförallt är det svårt att använda sig slumpvisa förlopp på ett sätt som stämmer överens med verkligheten. Enligt Law och Kelton (2000) används ofta någon form av matematisk fördelning för detta ändamål. I undersökningen av skyddsmetoden är det trafiken som ska analyseras och det är viktigt att denna stämmer överens med verkligheten på ett bra sätt. En matematisk fördelning ansågs inte vara tillräcklig för detta ändamål. Den undersökningsmetod som ansågs kunna konstruera den mest verklighetstrogna bilden av hur trafiken ser ut i ett verkligt scenario är en fallstudie. En fallstudie genomförs genom att implementera algoritmen på ett antal verkliga datorer och utföra tester i en verklig miljö. Detta var inte pratiskt möjligt att genomföra eftersom hårdvaran som skulle krävas för testerna inte fanns tillgänglig. Den metod som återstår är att genomföra en simulering. I en simulering kan trafiken genereras på ett verklighetstroget sätt och genomförandet av en simulering har inga speciella krav på hårdvara utan en dator kan användas till undersökningar med hundratals virtuella datorer inblandade. Därför valdes simulering som undersökningsmetod till detta projekt.

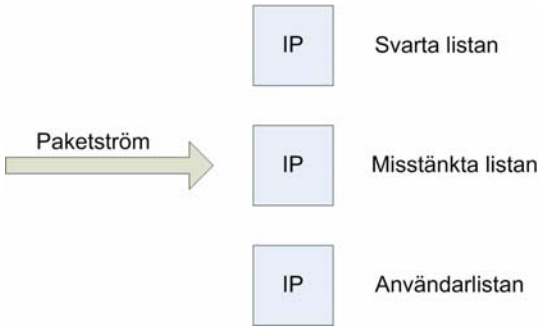
4.1 Algoritmens uppbyggnad

SeFFiM är en serverbaserad skyddsmetod som ingår i klass B och utformningen av algoritmen som ligger till grund för SeFFiM, som tagits fram i detta projekt, har hämtat inspiration från två andra skyddsmetoder. Den ena skyddsmetoden beskrivs av Voorhies m.fl. (2003). Den använder sig av en slumpfunktion som plockar ut paket från paketströmmen vilket inte används av SeFFiM i nuvarande impelementation. IP-numren i de paket som blir utvalda läggs på en kö. När frekvensen av ett visst IP-nummer blir för högt på kön så anses IP-numret vara deltagare i angreppet och paket från dessa IP-nummer kasseras i framtiden. Denna metod grundar sig på antagandet att ett angripande IP-nummer sänder en stor mängd paket. Chansen för att ett paket från en angripare väljs ut är större än från en vanlig användare eftersom en vanlig användare sänder betydligt färre paket.

Den andra skyddsmetoden som SeFFiM bygger på presenteras av Tanachaiwiwat m.fl. (2003). Den använder sig av en IP-tabell som består av tre köer med olika prioritet. IP-numren flyttas mellan tabellerna beroende på hur frekvent ett visst IP-nummer sänder paket.

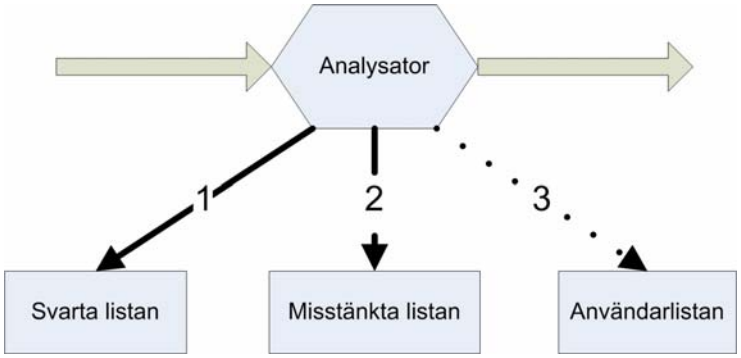
Figur 7 visar en förenklad bild av hur SeFFiM fungerar. Paketströmmen från nätverket kommer in till webbservern och IP-numren från paketet delas sedan upp i 3 olika klasser. Vilken klass de hamnar i påverkar hur framtida paket från IP-numret hanteras. De olika klasserna utgörs av listor som benämns svarta listan, misstänkta listan och användarlistan. Den svarta listan innehåller de IP-nummer som deltar i

angreppet, den misstänkta sådana IP-nummer som kanske är deltagare i angreppet och användarlistan innehåller de IP-nummer som tillhör de normala användarna av webbservern.



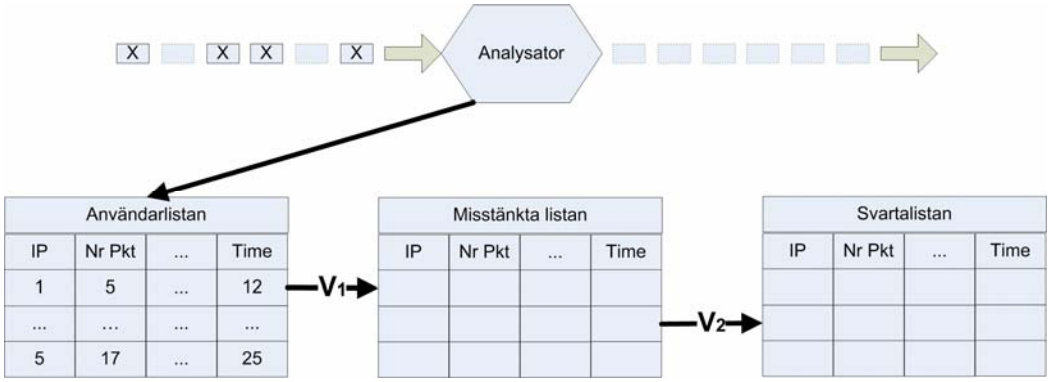
Figur 7 – Klassificering av IP-nummer

Schematiskt passerar paketen först genom en analysator. Analysatorn kontrollerar de olika listorna och beroende på vilken lista IP-numret befinner sig för tillfället så skickas paketet vidare till webbserver eller kasseras. I figur 8 illustreras detta förfarande.



Figur 8 – Listorna kontrolleras

Först kontrolleras den svarta listan efter det aktuella IP-numret. Om det finns där så kasseras paketet omgående. I nästa steg kontrolleras listan över misstänkta IP-nummer. Om IP-numret återfinns i denna lista uppdateras informationen som tillhör numret. Samma förfarande gäller användarlistan som listan över misstänkta nummer. Om IP-numret inte återfinns i någon av listorna så skapas ett nytt element i användarlistan och informationen om IP-numret läggs dit.



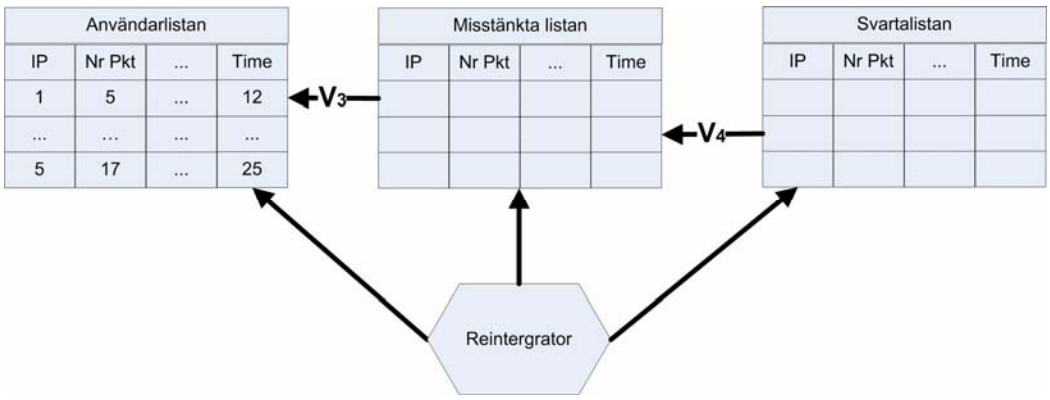
Figur 9 - Förflyttning mellan listorna

Figur 9 visar hur elementen med informationen om IP-numren förflyttar sig mellan de olika listorna. Informationen om ett IP-nummer som innefattar hur många paket som IP-numret sänt till webbservern, hur många av de som anses misstänkta och ett par sparade tidsstämplar. Algoritmen innehåller ett antal gränsvärden som är inställbara. Inställningen av gränsvärdena kan till exempel göras genom att den normala trafiken analyseras och ett mått på hur den normala trafiken ser ut tas fram.

$$n_i / t_i > x_i$$

Figur 10 - Formel för V_1 och V_2

Som Figur 10 visar så är det tre av dessa gränsvärden som påverkar förflyttningen mellan användarlistan och listan över misstänkta IP-nummer. Den tid t_i som ett visst antal paket n_i får skickas. Om detta över ett visst antal x_i så flyttas IP-numret från användarlistan till listan över misstänkta. Detta gäller både för V_1 och V_2 som Figur 9 visar men värdet på n_i, t_i och x_i är olika för V_1 och V_2. För att undvika att oskyldiga IP-nummer hamnar på den svarta listan eller anses misstänkta, finns även en funktion som flyttar tillbaka IP-nummer till användarlistan. Detta visas schematiskt i Figur 11.



Figur 11 – Tillbakaflyttning av IP-nummer

Kontrollen av ”Reintegratorn” görs periodiskt. Även dessa kan vara olika och V_4 beräknas ett par timmar eller dagar efter det att angreppet har avslutats och V_3 kan beräknas under ett angrepp för att ge bästa möjliga service till användarna av webbservern. Formeln som används i beräkningarna av V_3 och V_4 återfinns i Figur 12.

$$n_i / t_i < x_i$$

Figur 12 - Formel för V_3 och V_4

Algoritmen använder sig av tre listor för att lagra de olika IP-numren. Anledningen till detta är att det ska finnas möjlighet att undvika att en ”flash crowd” som beskrivs av Peng m.fl. (2003) och togs upp i kapitel 2.3, svartlistar vanliga användare. Under en ”flash crowd” så skickar en mängd olika IP-nummer en mängd paket och detta kan göra att algoritmen tror att IP-numret är en angripare och kasserar framtida paket ifrån detta. När tre listor används på det sätt som beskrivs i figur 11 finns det en möjlighet för ett IP-nummer att flyttas tillbaka från listan över misstänkta adresserna till användarlistan igen.

Denna behandling av IP-numret kan naturligtvis utnyttjas av en angripare. Om en angripare har vetskap om att denna skyddsmetod används och på något sätt fått reda på vilka gränsvärdena är så finns en möjlighet att sända paketen i en takt så inte gränsvärdena överskrids. Denna problematik kan förhindras om gränsvärdena görs dynamiska. Till exempel kan en extra process räkna om dem vid vissa tidpunkter.

När angreppet är över så är det önskvärt att listan över de svartlistade IP-numren rensas efter en viss tid. Om inte detta sker och listan byggs på efter varje angrepp finns risken att en ny typ av angrepp blir möjliga. Det kan tänkas att en angripare använder ett specifikt IP-nummer i syfte att svartlista detta från en viss server. Dessutom blir listan stor och tar mycket utrymme om den fylls på efter ett antal angrepp.

En större nackdel med algoritmen är att den inte hindrar att trafiken från ett angrepp att komma in på företagets interna nätverk och använder bandbredd som kunde användas av företagets anställda istället. Detta beror på att lösningen inte använder sig av någon router för att kassera paket vid inträdet i nätverket. En router skulle kunna användas för detta och problemet med att nätverket blir överbelastat kunde då undvikas. I detta scenario skulle en router till exempel ha en kopia på listan som innehåller de IP-nummer som är svartlistade. Detta kräver som tidigare diskuterats i kapitel 2.3 att en mer avancerad programmerbar router används och algoritmen får en mera omfattande installation.

Från Voorhies m.fl. (2003) togs idén att lägga IP-nummer på en kö och spara information om dessa istället för att paketen skulle utgöra nyckelposten i elementen i listan. På detta sätt sparas en mängd dataresurser eftersom antalet paket är mycket större än antalet IP-nummer under ett angrepp. Om varje paket från paketströmmen lagrades som ett element i listan tillsammans med IP-nummer skulle listan bli mycket stor och svårhanterlig. Idén att basera metoden på hur frekvent ett IP-nummer skickar paket togs också från denna artikel. Skillnaden mellan SeFFiM och den som är framtagen av Voorhies m.fl. (2003) är att i SeFFiM så används ingen form av slumpfunktion vid valet av paket från paketströmmen utan alla paket i SeFFiM utgör grunden för hur ett visst IP-nummer klassificeras. Detta val gjordes, eftersom även om sannolikheten är hög för att en angripare ska klassificeras rätt med hjälp av en

beräkning som grundar sig på en slumpvis utvald mängd paket så kan slumpfunktionen missa en stor mängd paket som utgör angreppet. Istället kan det i värsta fall klassificera vanliga användare som angripare. Denna risk minskar när sannolikheten för att ett paket ska väljas ut ökar, men är inte helt borta om inte alla paket undersöks.
Från skyddsmetoden som presenteras av Tanachaiwiwat m.fl. (2003) kommer idén att använda sig av flera listor att klassificera IP-numren i. Anledningen till att flera listor används är framförallt att listornas storlek blir mindre och därmed kan sökning och insättning göras effektivare. Skillnaden mellan skyddsmetoden som presenteras av Tanachaiwiwat m.fl. (2003) och SeFFiM är att metoden som Tanachaiwiwat m.fl. (2003) presenterar tillhör klass A och SeFFiM tillhör klass B, enligt figur 4.

Listorna i SeFFiM kan implementeras med hjälp av binärträd. Ett binärträd är enligt Skansholm (2000) en rekursiv lagringsstruktur som möjliggör en snabb och effektiv sökning av element som finns lagrade i trädet. Eftersom det är rimligt att antalet paket i ett distribuerat överbelastningsangrepp är större än antalet datorer som deltar i angreppet så är det flera sökningsoperationer än insättningsoperationer på listorna i SeFFiM. En insättningsoperation utförs när ett nytt IP-nummer tillkommer till webbserver samt när ett IP-nummer ska flyttas till en annan lista. Sökningsoperationen utförs när ett paket anländer till webbservern efter IP-numret som finns i paketet.

SeFFiM och metoden som presenteras av Voorhies m.fl. (2003) kan implementeras som binärträd respektive en kö. I Cormen, Leiserson, Rivest och Stein (2001) återfinns matematiska bevis för hur tidskomplexiteten utfaller för olika operationer med olika datastrukturer. Tabell 1 ger en beskrivning på hur en jämförelse mellan SeFFiM och skyddsmetoden som presenteras av Voorhies m.fl. (2003) ser ut om bevisen från Cormen m.fl. (2001) används.

Tabell 1 -Tidsjämförelse mellan SeFFiM och Voohries m.fl. (2003)

Operation	**SeFFiM**	**Voorhies m.fl. (2003)**
Sökning	*O(log i)*	*O(i)*
Insättning	*O(log i)*	*O(1)*
Uppdatering	*O(log i)*	*O(i)*[4]
Borttagning	*O(log i)*	*O(i)*

Tiden det tar för en sökningsoperation är i genomsnitt mindre i SeFFiM jämfört med Voorhies m.fl. (2003). Eftersom sökning sker frekvent i SeFFiM är det av yttersta vikt att denna blir effektiv. Insättningsoperationer tar längre tid i SeFFiM än i Voorhies m.fl. (2003) skyddsmetod. Detta beror på att i Voorhies m.fl. (2003) så läggs IP-numret sist i listan hela tiden och det utförs ingen kontroll om IP-numret redan finns i listan vid detta tillfälle. Vad gäller uppdateringsoperationer så utförs inte dessa i Voorhies m.fl. (2003) medan de utförs mycket frekvent i SeFFiM. I nuvarande implementation sker en uppdatering av ett element vid varje nytt inkommande paket. Borttagningsoperationer utförs i båda metoderna. I fallet med SeFFiM gör en

[4] Används inte av algoritmen

borttagning när ett IP-nummer flyttar till en annan lista och i fallet med Voorhies m.fl. (2003) utförs en borttagning då den interna listan med IP-nummer blir full. Trots att Voorhies m.fl. (2003) har möjligheten att avstå från att behandla samtliga paket i paketströmmen kan deras algoritm bli mera resurskrävande än SeFFiM när det gäller hanteringen av listan eftersom det skapas ett nytt listelement vid varje paketankomst när det i fallet med SeFFiM utförs en uppdatering istället. En jämförelse hur effektiva SeFFiM och ProbQueue är att stoppa ett distribuerat överbelastningsangrepp och hur de vanliga användarna av webbserver klarar sig under angreppet redovisas i kapitel 5.

En mera precis beskrivning av algoritmen som ligger till grund för SeFFiM och som beskrivits konceptuellt i figur 7-11 återfinns i algoritm 1. Källkoden till algoritmen återfinns i Appendix C.

Algoritm 1 (pseudokod)

Kontrollera om paketets IP-nummer finns i svarta listan

- Om ja, kassera paketet
- Om inte, kontrollera om det finns i den misstänkta listan
  - Om ja, kontrollera om paketet inkom för snabbt jämfört med de tidigare inkomna paketen
    - Om ja, flytta IP-numret till svarta listan
    - Om inte, kontrollera om tiden mellan de senaste paketen var tillräckligt lång så att IP-numret inte längre anses som misstänkt
      - Om ja, flytta IP-numret tillbaka till användarlistan och addera paketet till kön mot serverprocessen
      - Om inte, låt IP-numret vara kvar i de misstänktas lista och addera paketet till kön mot serverprocessen
  - Om inte, kontrollera användarlistan
    - Om ja, kontrollera om paketet inkom för snabbt jämfört med de tidigare inkomna paketen och att inte IP-numret har sänt för många paket totalt
      - Om ja, flytta IP-numret till misstänkta listan
      - Om inte, addera paketet till kön mot serverprocessen
    - Om inte, addera paketets IP-nummer till användarlistan

Algoritmen kräver i sin nuvarande form att en TCP/IP implementation med full duplex används på nätverket. Enligt Black (1994) innebär full duplex att det är möjligt att sända och ta emot data i båda riktningarna oberoende av varandra. För webbservern är detta väsentligt eftersom det är möjligt att sända data och därmed upprätthålla servicen till användarna även om mycket stora delar av bandbredden uppströms mot webbservern utgörs av trafik från angriparna. Dessutom krävs det att mängden paket med förfalskade IP-nummer minimeras med hjälp av till exempel till tillträdesfiltrering och utträdesfiltrering. Om detta denna filtrering utförs kan en

angripare gå runt SeFFiM helt genom att generera paket med en slumpvis vald IP-adress. SeFFiM antar att paket med olika adresser kommer från olika avsändare.

Fördelar med algoritmen

- Algoritmen är enkel att installera eftersom det enda som krävs är att en modul för den aktuella webbservern och utökning av nätverkshanteringen installeras och körs.
- Algoritmen har låga systemkrav eftersom den inte kräver en programmerbar router.

Nackdelar med algoritmen

- De paket som är en del av angreppet stoppas inte i företagets router utan kommer in på det interna nätverket. Detta medför att det interna nätverket blir fullt med trafik.
- Nuvarande algoritm körs hela tiden på webbservern. Om en skyddsmetod som var mindre krävande kördes för att upptäcka ett angrepp skulle dataresurser kunna sparas.
- Filtrering av paket med förfalskade IP-nummer krävs eftersom algoritmen bygger på statistik från olika IP-nummer.

5 Simuleringsresultat

Detta kapitel inleds med beskrivningar och förklaringar av vilka beslut som togs angående hur simuleringarna skulle utformas. Därefter presenteras resultatet av ett par simuleringar med diagram och beskrivning samt en jämförelse av resultaten i detta projekt med resultat från andra undersökningar.

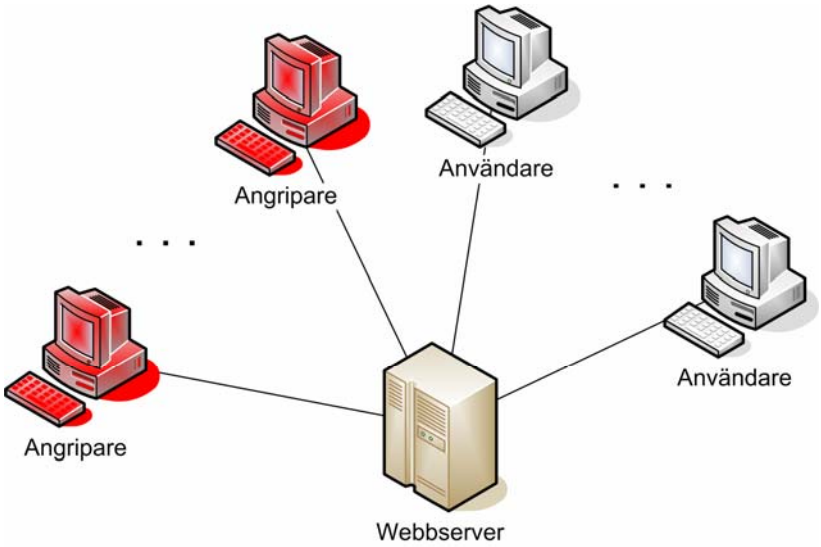
Valet av simulator föll på ”The network simulator – ns-2”. Detta val gjordes främst på grund av att ns-2 har används av de som gjort liknande undersökningar tidigare och av de som resultaten kommer att jämföras med. Simulatorn är enligt Chung och Claypool (2003) en händelsestyrd och objektorienterad nätverkssimulator. Den har utvecklats av UC Berkeley och finns tillgänglig som freeware vilket innebär att den är fri att använda och modifiera efter behov. Modifieringar och tillägg till simulatorn skrivs i C++ och de olika simuleringarna skrivs som skript med Tcl/Tk som språk.

När en simulering genomförs i ns-2 skapas objekt med de datorer och routrar som är aktuella i simuleringen. Noder kan i detta fall vara till exempel datorer eller routrar. För att data ska kunna skickas mellan dessa så skapas länkar som länkar ihop noderna med varandra. Länkarna kan använda sig av en rad olika överföringsprotokoll. Exempel på protokoll är TCP, UDP och en rad trådlösa protokoll.

För att kunna skicka data mellan noderna över länkarna skapas ett antal agenter som kopplas ihop med noderna. När UDP används som protokoll kan UDP-agenterna generera och skicka data till andra noder i nätverket. Det är också agenternas uppgift att räkna mottagen data och på så sätt få fram statistik om vad som händer under en simulering. Under pågående simulering kan resultatet skrivas ner i filer som sedan kan analyseras. I detta projekt har Microsoft® Excel® används för att bearbeta utdata och visualisera denna i diagram.

Målet med simuleringarna är att analysera hur bra algoritmen är på att hjälpa klienter att koppla upp sig mot en webbserver även om den är utsatt för en distribuerad överbelastningsangrepp. För att få ett mått på hur effektiv SeFFiM är kan responstiden mätas både när servern är i normal drift och under ett angrepp. Med responstid menas tiden från det användaren skickar en begäran i form av ett paket till webbserver tills användaren får svar på denna begäran från servern igen. Responstiden är intressant att simulera eftersom det ligger nära ett verkligt scenario.

Strukturen på nätverkstopologin som används i simuleringar av algoritm 1 är illustrerad i figur 13. Topologin är en förenklad bild av verkligheten då en router inte är inritad. Detta är på grund av att en router inte används i simuleringarna eftersom SeFFiM inte kräver någon sådan. I de olika simuleringarna så varieras antalet angripare och användare.



Figur 13 – Nätverkstopologi som används i simuleringarna

Variationen av antalet datorer i simuleringen sker stegvis med början vid 10 angripare och 1 användare som använder webbservern när ett angrepp inträffar, 1 användare som kopplar upp sig innan angreppet startar och ytterligare 1 som försöker koppla upp sig när angreppet är igång. Sedan ökas antalet angripare 10 gånger så antalet angripare blir 100 och antalet användare sätts till totalt 20 stycken i simuleringen. För att ytterligare kontrollera om antalet angripare och användare har någon effekt på algoritmens effektivitet görs ytterligare en simulering med 1000 angripare och 100 användare vid de tre tillfällen användarna introduceras i simuleringen. Andledningen till att öka antalet angripare och användare på detta sätt är att det är ett effektivt sätt att hitta den tidpunkt då intressanta händelser kan observeras av. Om något intressant inträffar i simuleringen med 1000 angripare i jämförelse med simuleringen med 100 angripare kan det vara av intresse att köra ytterligare simuleringar med till exempel 500 angripare och se om liknande saker inträffar och på detta sätt få fram den mest intressanta kombinationen av angripare och användare. I detta fall kan intressanta kombinationer vara när webbservern inte kan hantera den inkommande trafiken utan hjälp från en skyddsmetod. Detta inträffar någonstans mellan 10 och 100 angripare men eftersom denna undersökning inte har som mål att hitta antalet angripare som webbservern klarar av att hantera gjordes inga ytterligare simuleringar med ett antal angripare i intervallet 10 till 100.

Användarna försöker koppla upp sig vid tre tillfällen mot webbservern. En del användare är uppkopplande när simuleringen börjar. En annan del kopplar upp sig efter 25 % av tiden, innan angreppet börjar och den sista delen användare kopplar upp sig under angreppet, efter 75 % av tiden. Angreppet startar efter halva tiden.Valen av tidpunkter gjordes eftersom det är viktigt när simuleringar genomförs att de utförs med identiska förutsättningar. Det som inträffar vid 75 % av tiden är särskilt intressant eftersom den beskriver om användare kan använda webbservern under ett överbelastningsangrepp eller inte.

Protokollet som valts för att simulera förfarandet som beskrivs i figur 3 är UDP. Alternativet till detta var att modifiera den existerande implementationen av TCP-protokollet så att en ström med SYN-paket skickas och inga svar på dessa skickas då servern begär detta. Valet föll på att använda en ström av UDP-paket, dels för att

lösningen inte var så komplicerad som en modifikation av TCP-protokollet skulle ha varit och dels för att bland andra Voorhies m.fl. (2003) och Yan, Lui och Liang (2002) använt sig av en ström av UDP-paket i sina simuleringar. Eftersom en jämförelse med resultat från andra rapporter ska göras så valdes denna lösning eftersom det är önskvärt att förutsättningarna i simuleringar liknar varandra.

Enligt Parker och Sportack (2000) är den största skillnaden mellan UDP och TCP att TCP garanterar att paket kommer fram till målet medan UDP inte har några garantier alls. Vidare så har UDP inte någon handskakning som TCP har, utan sänder paketen till mottagaren utan någon kontroll över om de kommer fram eller inte. Enligt Parker och Sportack (2000) har UDP fördelen att paketen har lite extra information utöver de data som paketet transporterar (eng. overhead). TCP har däremot relativt mycket overhead. I detta projekt simuleras en användares trafik genom att UDP-paket sänds till servern. Simuleringarna går i detta fall ut på att visa om användare av webbservern kan koppla upp sig mot denna både före och under ett angrepp. Till detta ändamål användes ett enkelt pinganrop för att kontrollera om användarens dator kunde komma åt webbservern och att simulera handskakningsprocessen i Figur 2. Ett pinganrop som fungerar räknas som en lyckad handskakningsprocess. Ett pinganrop fungerar enligt Parker m.fl. (2000) så att en dator sänder ett paket till en annan dator som sänder ett svar på detta. Tiden det tar från det anropet sker till dess svar kommer fram utgör utdata från pingfunktionen. Om ett pinganrop lyckas så kan därmed användarens dator koppla upp sig mot webbservern vid en viss tidpunkt. Ett pingpaket är litet (60 byte) i jämförelse med de paket som skickas av angripare och användare (500 byte). Dessutom skickas pingpaketen endast en gång per sekund från användarna till servern medan övrig trafik består av hundratals paket per sekund. Den extra trafik som pingpaketen genererar är mycket liten i jämförelse med övrig trafik. Denna approximation används också av Voorhies m.fl. (2003). Några argument för att använda UDP istället för en modifiering av TCP-protokollet utöver att detta är enklare att implementera kunde inte utläsas i Voorhies m.fl. (2003) eller Yan m.fl (2002).

I ett verkligt scenario med ett överbelastningsangrepp så är det serverns buffert över antalet samtidiga anslutningar som fylls upp under angreppet. Eftersom valet av paketström föll på UDP-paket simuleras serverns anslutningsbuffert att en angripare inte får sända mera paket än ett visst antal under en viss tid istället för att antalet aktiva anslutningar i bufferten räknas. Om en angripare sänder för många paket under en viss tid blir denna så småningom svartlistad från servern och framtida paket från detta IP-nummer kasseras.

De värden som kan varieras under simuleringarna är:

- Antalet angripare
- Intervallet mellan paketen i strömmen av UDP paket från angriparna
- Antalet användare vid de tre tillfällen då klienterna försöker koppla upp sig mot webbservern.
- Hur paketströmmen ska se ut från användarna
- Tiden mellan användarnas uppkopplingsförsök och tiden de väntar på svar från servern innan ytterligare försök genomförs.
- De värden som illustreras av Figur 10 och 12 kan varieras. Tiden som antalet paket mäts under och antalet paket som utgör det maximala en användare får sända under en viss tid.

- Om webbrus ska vara med i simuleringen. När webbrus är en del av simuleringen så innebär detta att en simulerad bakgrundstrafik läggs till. Detta påverkar resultatet genom att tiden det tar för ett paket att komma fram kan variera och att vissa paket kan försvinna längs vägen till målet. Nätverket mellan de anslutna användarna och webbservern blir inte belastat ytterligare av detta på något sätt eftersom inga ytterligare paket skickas.

Antalet angripare är en mycket väsentlig faktor i ett distribuerat överbelastningsangrepp. Om antalet angripare är högt är möjligheterna för dessa att skicka ut stora strömmar med paket större. Metoden som försvarar en webbserver ställs också inför en större utmaning eftersom antalet olika IP-nummer är stort. Hur paketströmmen ser ut från angriparna är också väsentlig. I dessa simuleringar så skickas så mycket paket som möjligt från samtliga angripare. Detta kan ses som en fördel för algoritmen som ska skydda webbservern genom att det i detta fall blir lättare att urskilja trafiken som tillhör angriparna. Men det är rimligt att anta att trafiken från angripare kan se ut på detta sätt eftersom en angripare troligtvis vill åstadkomma maximal skada för den webbserver den riktar sitt angrepp emot.

Hur många användare som använder webbservern och hur deras paketström ser ut är inte fullt så väsentligt som hur angriparnas trafik ser ut. Detta beror på att antalet klienter i dessa simuleringar är mycket färre än angriparna och paketströmmen från dessa är betydligt mindre omfattande. Det är högst troligt att användare och angripare i ett verkligt scenario förhåller sig på ett liknande sätt. Som beskrivits av Peng m.fl. (2003) kan en ”flash crowd” inträffa och då kan ett IP-nummer från en vanlig användare blir klassificerat som misstänkt. För att motverka detta finns en möjlighet att flytta tillbaka IP-nummer från listan över misstänkta IP-nummer till användarlistan igen. Det kan också inträffa att en ”flash crowd” från ett visst IP-nummer blir långvarig och detta kan leda till att SeFFiM klassificerar IP-numret som en angripare. Eftersom en ”flash crowd” liknar ett riktigt angrepp så är det mycket svårt att skilja dessa åt. Ett rimligt antagande är att ett riktigt angrepp från ett IP-nummer varar under en längre tid än en ”flash crowd” gör. I detta fall går det att ställa in de tröskelvärden som används i klassificeringen av IP-numren så att de tillåter att en ”flash crowd” kan pågå under en viss tid. Detta kan dock utnyttjas av en angripare om denne känner till hanteringen av ”flash crowds” genom att skicka ut paket i perioder för att vilseleda skyddsmetoden att tro att det är vanliga användare som står för trafiken. För att skydda en webbserver mot en angripare som utnyttjar ”flash crowd” hanteringen skulle någon form av historik kunna användas för att spärra IP-nummer som skickar en paketström som liknar en ”flash crowd” upprepade gånger. I nuvarande version av SeFFiM är inget skydd för vanliga användare som genererar en ”flash crowd” på webbservern implementerat, vilket kan leda till att IP-numret svartlistas. Utan hanteringen av ”flash crowds” lämnas som framtida arbete.

Ytterligare faktorer som påverkar resultatet på simuleringarna är hur de värden som reglerar hur många paket som får sändas under en viss tid är satta. Dessa har tagits fram genom att analysera trafiken med hjälp av beräkningar på hur trafiken ser ut från användare respektive angripare. Användarnas och angriparnas trafik har satts utifrån de värden för dessa som använts i simuleringar som genomförts av Voorhies m.fl. (2003). Användarna sänder en paketström av ca 30 paket/s och angriparnas trafik utgörs av ca 330 paket/s. Eftersom metoden har som mål att skydda från så många angripare som möjligt sattes tröskelvärdena som bestämmer hur stor paketströmmen får vara relativt nära de värden som representeras av användarnas trafik.

Tröskelvärdena är satta enligt följande rader som återfinns i källkoden till simuleringen i Appendix B.

```
Queue/SeFFiM set minTimeLimit_ 60
Queue/SeFFiM set minPktLimit_ 40
Queue/SeFFiM set maxTmpPkt_ 15
```

Tiden är satt till 60 ms, antalet paket som maximalt får skickas under denna tid är 15 st. Hur många gånger gränsen på 15 paket får överskridas innan ett listbyte sker är satt till 40 gånger. En dator får med dessa värden skicka maximalt 375 paket under 10 s.
I ett verkligt scenario behöver dessa värden ställas in på något sätt. Antingen manuellt av administratören av systemet eller beräknas på något sätt. Denna beräkning skulle kunna utföras genom att historik av normal trafik används för att få en bild över hur många paket som skickas och hur många anslutningar som utförs normalt.

5.1 Implementation av SeFFiM

Till detta projekt implementerades algoritm 1 som en arvinge till den inbyggda köklassen i ns-2. Enligt Chung m.fl. (2003) har all C++ högre prioritet när en simulering körs, än Tclkoden som simuleringen skrivs i. Eftersom detta var fallet behöver inte lagringsstrukturen som tabellerna implementerades i vara den absolut effektivaste. Därför föll valet på lagringsstruktur på standardklassen vector som ingår i C++ standarden. Fördelen med detta val är att standardklasserna är vältestade och tiden för implementation kan fokuseras på att implementera algoritm 1. Istället för att bygga en egen klass för lagringsstrukturen och spendera tid på testning. Nackdelen med att använda en standardklass på detta sätt är att det inte går att optimera den underliggande lagringsstrukturen speciellt för den överliggande applikationens funktioner. I detta fall behövdes inte några funktioner som inte kunde implementeras med hjälp av de inbyggda funktionerna hos vectorklassen så därför valdes detta som lagringsstruktur. Alternativet till vector hade varit att implementera listorna med hjälp av binärträd som diskuterades i kapitel 4.1. Binärträd finns också att tillgå som standardklass i C++ men strukturen är mer komplicerad än en vector och eftersom valet inte har inverkan på simuleringsresultatet valdes den enklare vector strukturen för implementeringen av de tre listorna.

Ovanpå vectorklassen byggdes en egen köstruktur för att underlätta implementationen av algoritm 1 samt en nodklass för att hålla reda på information om elementen i köklassen. Strukturen på kön är illustrerad i tabell 2. Varje element i kön representeras av en rad i tabellen.

Tabell 2 - Struktur på kö

IP-nummer (nod id)	Antal (paket skickade)	Antal misstänkta paket	Timestamp (senaste paketet)	Timestamp (första paketet)
3	12	0	345	2
15	35	2	265	202
…	…	…	…	…
32	67	5	125	5

Ns-2 lagrar ett IP-nummer som ett heltal och varje dator och router i en simulering har ett unikt nummer. Den andra kolumnen innehåller de antal paket som skickats från IP-numret. Detta värde lagras därför att det finns möjlighet för algoritmen att flytta paket mellan listorna baserad på den totala mängden paket som skickats från ett visst IP-nummer. I den tredje kolumnen lagras antalet paket som anses misstänkta. Detta värde avgör om IP-numret ska flyttas beroende på hur stort detta värde är. I de två sista kolumnerna lagras den värdet på den logiska klocka som ns-2 använder. Gränsvärdena för värdena i kolumn 2-4 är inställbara från tcl-skriptet som källkoden simuleringen skrivs i.

Hur algoritm 1 implementeras i ett verkligt scenario kan skilja sig lite beroende på vilket operativsystem som används på datorn som webbservern körs på. Enligt McClure m.fl. (2003) är Apache en av de vanligaste webbserverapplikationerna och det vanligaste operativsystemet att köra en Apacheserver på är UNIX samt besläktade varianter. På ett sådant system skulle implementationen av SeFFiM delas upp i två delar. Den del, som i fallet med simuleringarna består av själva algoritmen och listorna, kan implementeras som en utökning av nätverkshanteringen i kärnan av UNIX systemet och delen som sätter de olika tröskelvärdena som en egen del t.ex. som en modul till Apacheservern. Andledningen till att dela upp implementationen är att SeFFiM helst ska ha tillgång till paketströmmen från nätverket på ett tidigt stadium när denna anländer till webbservern. Det är önskvärt att de paket som kommer från de svartlistade IP-numren kasseras och deras anslutningar plockas bort från bufferten på servern så fort som möjligt för att kunna lämna plats för de vanliga användarnas anslutningar. Den andra delen av implementationen som består av tilläggsprogrammet till webbservern innehåller funktioner för att sätta de gränsvärden som algoritmen använder för att klassificera trafiken.

5.2 Simuleringsresultat

De simuleringar som genomförts till detta projekt har visat att SeFFiM fungerar enligt beskrivningen som presenterats i kapitel 3. Simuleringarna visar hur pingvärdet förändras för en användare när ett distribuerat överbelastningsangrepp sker på webbservern. Samtliga diagram för de olika simuleringarna återfinns i appendix A och källkoden till simuleringarna och SeFFiM återfinns i appendix B och C. Vilka program och vilken version av respektive program som användes framgår av tabell 3.

Tabell 3 - Programversioner som används i simuleringarna

Program	Version
ns-2	2.26
Tcl/tk	8.3.2
Gcc	3.3.1
Mandrake Linux	9.2
Kernel	2.4.22

En av metoderna som simuleringar genomförts med är DropTail. Detta är en av de inbyggda metoderna i ns-2 för köhantering. Metoden fungerar så att den kontrollerar om mottagaren av paketet har möjlighet att ta emot det. Om mottagaren har möjlighet att ta emot paketet skickas detta till mottagaren för ytterligare behandling. Om mottagaren är upptagen med att hantera andra paket kasseras det aktuella paketet av metoden istället. I simuleringarna till detta projekt utgör DropTail en bas att jämföra de andra simuleringsresultaten med för att se hur resultatet av ett simuleringsscenario ser ut utan att något skydd mot överbelastningsangrepp används.

För att kontrollera hur bra SeFFiM fungerar i jämförelse med andra liknande skyddsmetoder gjordes simuleringar med en metod kallad ProbQueue som tagits fram av Voorhies m.fl. (2003). Hur ProbQueue fungerar finns beskrivet i kapitel 2.2.3

ProbQueue har två värden som är unika för skyddsmetoden. En av dessa är vilken sannolikhet ett visst IP-nummer har att hamna på den interna tabell som har en variabel maximal storlek. Tanken med dessa simuleringar var att med en sannolikhet på 100% och en tabellstorlek som representerade antalet angripare så skulle skyddsmetoden vara mest likt SeFFiM. För att se om utfallet ändrades om sannolikheten och tabellstorleken ändrades genomfördes även andra simuleringar med andra värden på tabellstorleken och sannolikheten.

Noterbart är att ProbQueue utvecklats som en skyddsmetod som behöver hjälp av en router för att kassera inkommande paket. Simuleringarna som har genomförts har dock visat att ProbQueue fungerar även utan hjälp av en router och därför gjordes valet att genomföra jämförelsen med SeFFiM även om resultatet sannolikt hade sett bättre ut med en router för ProbQueue. Det som testades av Voorhies m.fl. (2003) med simuleringarna av ProbQueue var andelen paket från den vanliga trafiken som behandlades av webbservern istället för responstiden. Detta ger också ett mått på algoritmens effektivitet med avseende på att göra webbservern tillgänglig för användarna under ett angrepp. Andledningen till att simuleringen i detta projekt fokuserades på att mäta responstiden istället för andelen paket av användarnas trafik som behandlas av webbservern var att responstiden ger ett mera exakt mått på hur kvalitén på servicen från en webbserver ser ut. Enligt Xu m.fl. (2003) lämnar en användare en webbsida om det tar mer än 8 sekunder för webbsidan att visa en bekräftelse på att den är tillgänglig. I detta fall är det intressant för skyddsmetoden att upprätthålla ett pingvärde som understiger 8 sekunder under ett angrepp, vilket inte simuleringarna som Voorhies m.fl. (2003) genomfört visar. Diagrammen som illustrerar responstiden i sekunder visar den relativa eventuella förändringen i responstid. Andledningen till att den faktiska tiden i sekunder inte visas är att det som är intressant att visa är om tiderna ser lika ut innan och under ett angrepp. Om den faktiska tiden hade visats hade det bara påverkat skalan i diagrammen.

Från de simuleringar som genomförts av Voorhies m.fl. (2003) har de värden som bestämmer hur trafiken ska se ut från angripare och användare tagits. Detta val gjordes för att simuleringarna i detta projekt skulle likna de simuleringar som genomförts med ProbQueue av Voorhies m.fl. (2003). Angriparnas trafik är satt till 100 gånger större än de vanliga användarnas.

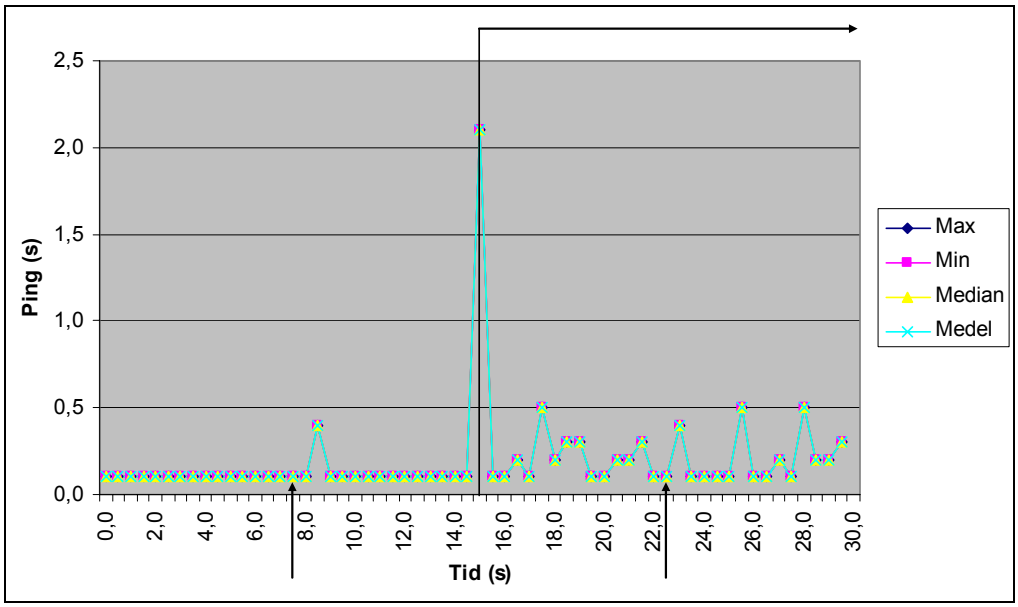
I de simuleringar som genomförts har följande parametrar varierats.

1. Antalet angripare
2. Antalet användare av webbservern
3. Tabellstorleken och sannolikhetsvärdet i simuleringarna med ProbQueue

Antalet angripare varierades för att visa hur väl skyddsmetoden klarar av att filtrera en större paketström och antalet användare för att se om pingvärdet för användarna ändras när antalet användare ökar. I simuleringarna med ProbQueue var antalet användare och angripare konstant. Det som varierades var storleken av den interna tabellen som lagrar IP-nummer av inkommande paket och sannolikheten att IP-numret från paketet läggs på metodens interna lista.

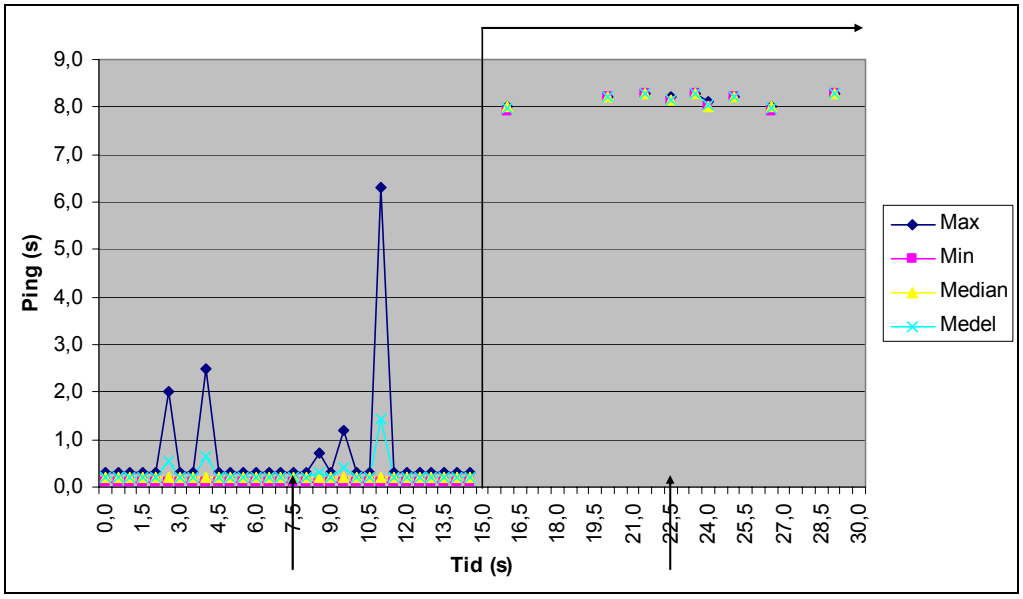
För att visa hur väl trafiken filtreras vid olika tillfällen så kopplar användarna upp vid olika tillfällen under simuleringen och gör ett test om de lyckas få kontakt med webbservern. De pilar som finns i diagrammen markerar de händelser som beskrevs i tidigare i detta kapitel. De två kortare pilarna markerar tidpunkter då användare försöker koppla upp sig och den längre pilen i mitten av diagrammet markerar när angreppet börjar samt att det pågår till slutet av simuleringen.

Figur 14 visar simuleringen som genomfördes med 10 angripare och totalt 3 användare visade att pingvärdet inte ändrar sig nämnvärt under ett angrepp i jämförelse normala förhållanden. Värt att notera är att pingvärdet ökar markant i det ögonblick då angreppet startar och sedan sjunker lika snabbt igen. Vad denna ökning beror på är oklart men eftersom denna simulering endast visar data från en användare kan det bero på variationer i trafiken som utgörs av webbrus. Ett antal simuleringar genomfördes med dessa inställningar men samtliga gav exakt samma resultat varje gång. Varför detta inträffade diskuteras i kapitel 7.



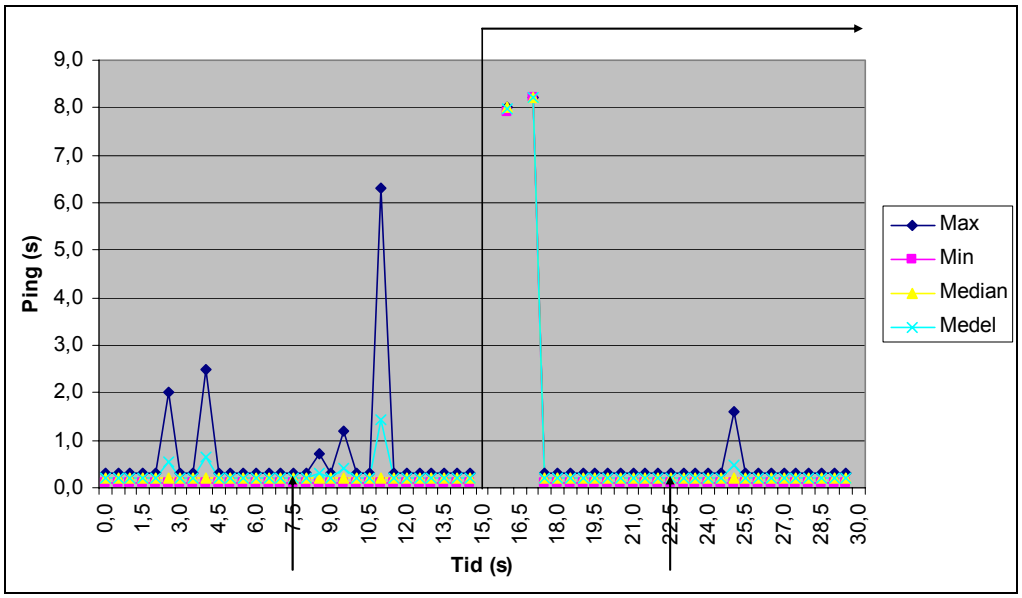
Figur 14 - Simulering med 10 angripare och 3 användare utan skyddsmetod (DropTail) och visar användare som är uppkopplade innan angreppet.

Eftersom pingvärdet inte ändras nämnvärt när ett angrepp startar utelämnas analysen av resterade diagram från simuleringar med 10 angripare och 3 användare.



Figur 15 – Simulering med 100 angripare och 20 användare utan skyddsmetod (DropTail) och visar användare som är uppkopplade innan angreppet.

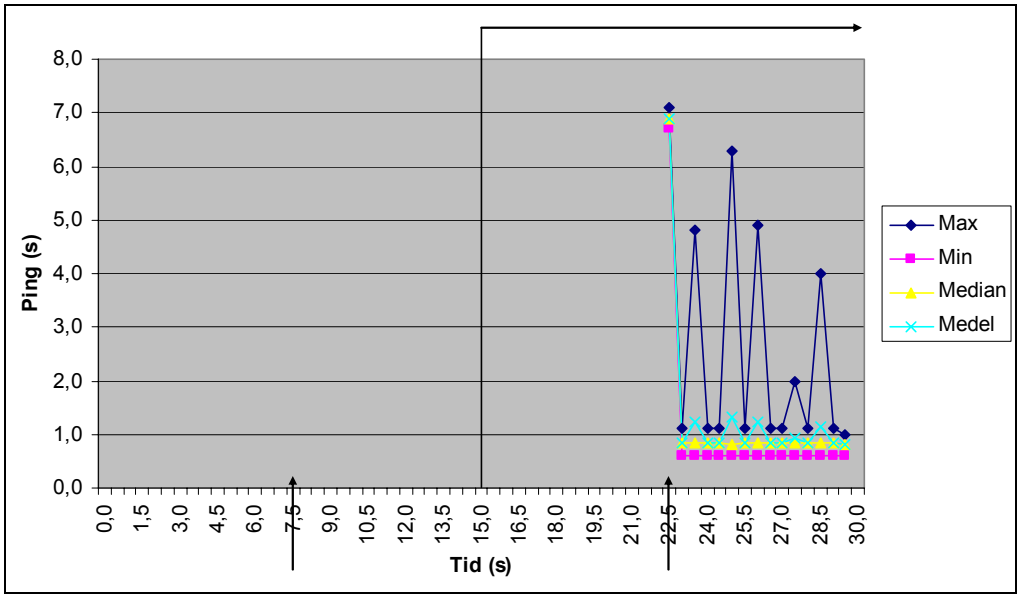
Av Figur 15 går att utläsa att när antalet angripare blir så stort att de kan fylla upp hela bandbredden påverkat serverns förmåga att svara på användarnas pinganrop mycket negativt. Endast ett fåtal av anropen blir besvarade och servern är i det närmaste otillgänglig för användarna.



Figur 16 - Simulering med 100 angripare och 20 användare med SeFFiM som skyddsmetod och visar användare som är uppkopplade innan angreppet startar.

Figur 16 visar en simulering med de parametrar som användes i simuleringen som figur 15 med det undantaget att i figur 16 så har SeFFiM använts som skyddsmetod. Grafen i figur 16 visar tydligt att SeFFiM ger betydligt bättre responstider till

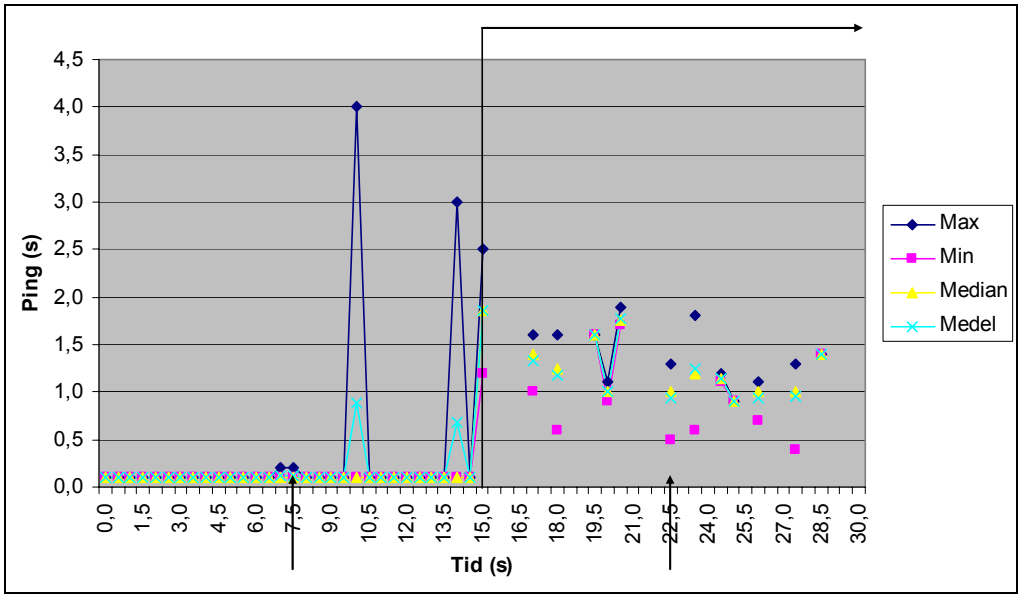
användarna under ett angrepp än när ingen skyddsmetod användes. Metoden tar ca 2 sekunder av simuleringstiden på sig att klassificera vilka av de uppkopplade datorerna som är angripare. Under denna tid är webbservern visserligen oanvändbar men denna tid är betydligt kortare än om ingen skyddsmetod alls används. Det är mycket sannolikt att denna tid går att ändra om andra värden används för att klassificera IP-numren men detta lämnas som framtida arbete att undersöka. Även de användare som försöker få kontakt med webbservern under ett angrepp lyckas med detta vilket figur 17 illustrerar.



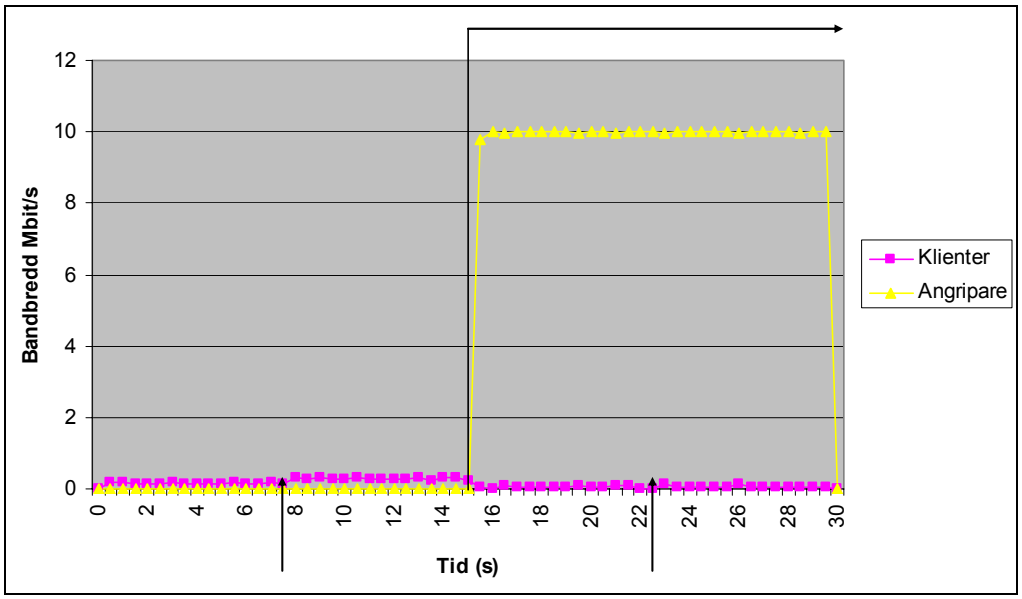
Figur 17 – Simulering med 100 angripare och 20 användare med SeFFiM som skyddsmetod och visar användare som kopplar upp sig under angreppet.

Noterbart är dock att användarnas pingvärde är större än de användare som var uppkopplade mot webbservern innan angreppet. Detta beror sannolikt på att ns-2 behandlar noder i en simulering i ordning med utgångspunkt i vilken ordning de skapades i skriptet och därmed vilket ID-nummer de har. De användare som kopplade upp sig först har alla lägre ID-nummer än de som kopplar upp sig senare. Angriparna har alla ID-nummer som är högre än användarna. Detta kan ses som en fördel för metoden men förutsättningarna är exakt likadana för alla de tre testade metoderna så ordningen på id-numren har mindre betydelse för slutresultatet av jämförelsen metoderna sinsemellan. En diskussion om hur detta hade kunnat förhindras finns i kapitel

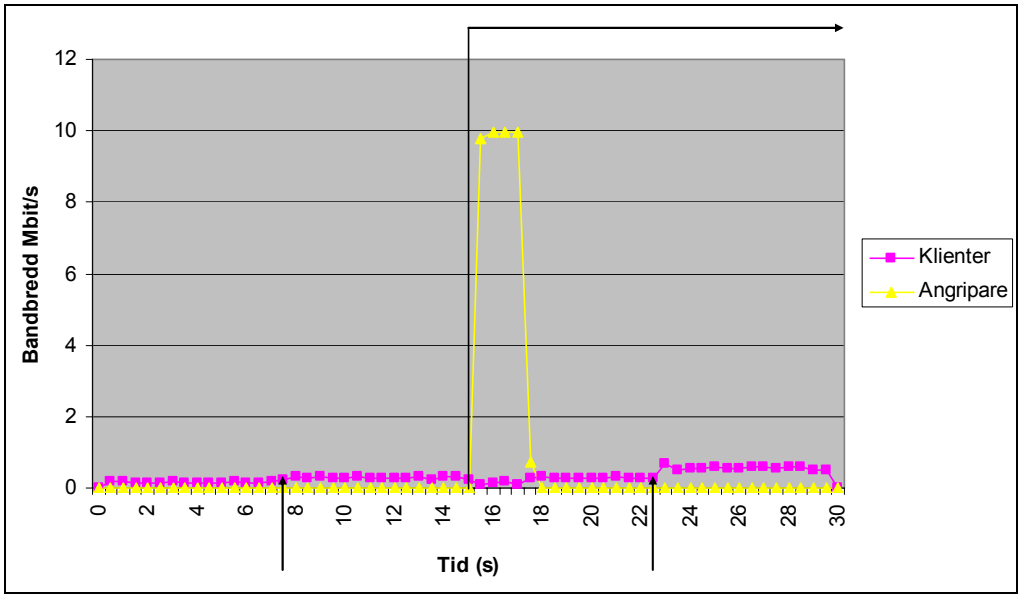
Figur 18 visar en simulering med ProbQueue. Diagrammet visar att ProbQueue lyckas att filtrera ut och leverera en del av användarnas trafik till webbservern trots att ingen router används för att kassera paketen som tillhör angriparnas trafik. Figur 19 visar att angriparna upptar i princip hela webbserverns ingående bandbredd. Detta beror på att metoden inte har hjälp av någon router som kasserar paketen från angriparnas trafik.



Figur 18 – Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 50 och visar användare som är uppkopplade före angreppet



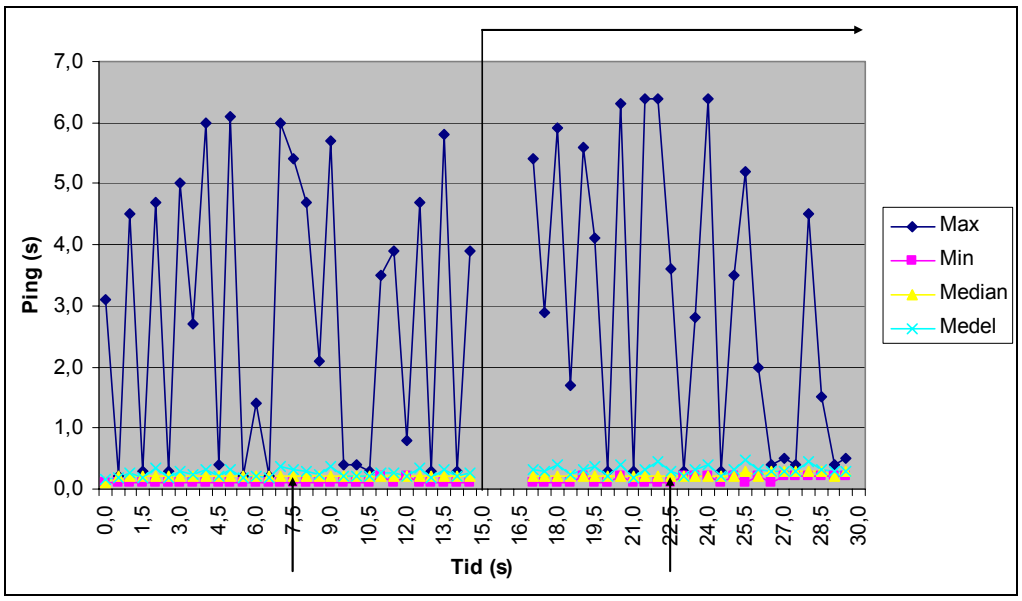
Figur 19 – Bandbredd som används av 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 100.



Figur 20 – Bandbredd som används av 100 angripare och 20 användare med SeFFiM som skyddsmetod.

Figur 20 visar tydligt att SeFFiM effektivt klassificerar vilken trafik som tillhör angriparna. Diagrammet visar att klassificeringen tar ca 2 sekunder av simuleringstiden. Efter klassificeringen kasseras paketen som angriparna skickar och webbservern har bandbredd över för att göra webbservern tillgänglig för de vanliga användarna. Eftersom SeFFiM är serverbaserad är figur 20 inte ett diagram över bandbredden mellan server och användare utan mellan nätverkskortet i datorn som webbservern är installerad på och webbserverapplikationen. Avlastningen som SeFFiM ger webbservern är dock betydande som visas i figur 20.

Figur 21 visar att SeFFiM klarar av att filtrera trafiken på ett effektivt sätt när antalet angripare ökar med ytterligare 10 gånger i jämförelse med figur 16. Skillnaden är att responstiden ökar för användarna när antalet datorer som är uppkopplade mot webbservern är högt. Detta beror på att de vanliga användarnas trafik konsumerar en stor del av webbserverns bandbredd och därmed blir responstiden längre eftersom webbservern har en större mängd trafik att behandla.



Figur 21 - Simulering med 1000 angripare och 200 användare med SeFFiM som skyddsmetod och visar användare som är uppkopplade innan angreppet startar.

Resultaten av simuleringarna har visat att SeFFiM fungerar under de förutsättningar som den simulerade miljön ger. Algoritmen SeFFiM ger ett tillräckligt bra skydd mot distribuerade överbelastningsangrepp och klarar att hålla webbservern tillgänglig för vanliga användare under ett angrepp. I sin nuvarande form är inte algoritmen ett direkt applicerbart skydd eftersom en simulerad miljö kan skilja sig en del från ett verkligt scenario och för att undersöka detta krävs att en fallstudie genomförs.

5.3 Relaterat arbete

I kapitel 2 presenterades ett antal metoder som kan användas som skydd mot distribuerade överbelastningsangrepp. Det som ansågs mest intressant var att relatera resultaten av SeFFiM till de metoder som bidragit med idéer till utformningen av SeFFiM. Resultaten av simuleringarna med ProbQueue som utförts av Voorhies m.fl. (2003) visar att metoden fungerar bra när en router är en del av lösningen. I deras simuleringar undersöktes hur effektivt ProbQueue sorterade ut de paket som tillhörde de vanliga användarna av webbservern i jämförelse med ett antal andra algoritmer. Undersökningen gick ut på att visa vilken av algoritmerna som kasserade minst paket från de vanliga användarnas trafik. Resultatet av simuleringarna visar att ProbQueue kunde svara på anrop för 80 % av användarna under ett angrepp vilket var betydligt mer än någon annan av de andra algoritmerna som deltog i undersökningen.

Den andra metoden som bidragit med idéer till SeFFiM presenterades av Tanachaiwiwat m.fl. (2003). I deras rapport har inte metoden simulerats utan en matematisk analys användes för att visa hur metodens skydd fungerar. Enligt analysen fungerar metoden 45 % bättre än tillträdes- och utträdesfiltrering som används till att kassera paket med förfalskad avsändare.

6 Slutsats

Problemet med distribuerade överbelastningsangrepp har ingen enkel lösning och kan vara mycket kostsamma för ett företag som blir utsatt. I detta projekt har målet varit att utveckla och undersöka en metod som kan vara ett direkt applicerbart skydd mot överbelastningsangrepp på webbservrar. Fokus har lagts på webbservrar därför att dessa är ofta företagets ansikte mot kunden och företagets rykte kan skadas om webbservern blir otillgänglig. Den övriga forskning som har bedrivits på området har främst fokuserat på lösningar som använder routrar på något sätt. För att kunna använda en router som en del av skyddsmetoden krävs det att routern är programmerbar vilket inte är fallet med alla routrar. Den största nackdelen med en helt serverbaserad lösning är att angriparnas trafik kommer in på företagets interna nätverk.

För att undersöka om den framtagna metoden, SeFFiM, fungerade genomfördes ett antal simuleringar. I simuleringarna mättes responstiden som användarna fick från webbservern före och under ett överbelastningsangrepp. För att få en bild av hur effektivt angriparnas trafik filtrerades bort genomfördes även simuleringar som visade hur mycket av webbserverns bandbredd som användes av angripare respektive användare. Resultaten från simuleringarna visade att SeFFiM klarade att filtrera bort angriparnas trafik och ge användarna bra responstider även under ett angrepp.

En jämförelse med resultaten från simuleringarna med SeFFiM genomfördes med simuleringar med en annan skyddsmetod mot överbelastningsangrepp. Metoden som SeFFiM jämförs med är ProbQueue och har den har tagits fram av Voorhies m.fl. (2003). Trots att ProbQueue egentligen är en hybridmetod som använder sig av en router så gjordes en jämförelse med SeFFiM när endast en webbserver användes. Resultaten från denna jämförelse visade att SeFFiM var bättre på att göra webbservern tillgänglig för användarna än ProbQueue under dessa förhållanden. Simuleringar som utförts av Voorhies m.fl. (2003) när en router användes visade att metoden fungerade bättre än den gjorde i simuleringarna som presenterades i kapitel 5, men en direkt jämförelse med SeFFiM med dessa resultat är svår att göra, eftersom olika saker undersöktes.

Resultaten av denna undersökning har visat att i en simulerad miljö fungerar en serverbaserad skyddsmetod mot distribuerade överbelastningsangrepp bra och uppfyller kraven på tillgänglighet som diskuterades i kapitel 2.3. Även om det är osäkert om SeFFiM fungerar lika bra i ett verkligt scenario kan den vara ett tillräckligt bra skydd. Problemet med att mycket trafik kommer in på företagets interna nätverk kan uppvägas av att full duplex används i det interna nätverket och därmed gör det möjligt att skicka data till användarna.

Alla som kör en webbserver där webbsidorna är kritiska för ett företags rykte borde installera ett skydd mot överbelastningsangrepp. Detta projekts framtagna skyddsmetod kan utvecklas vidare till ett skydd som är direkt applicerbart på en webbserver. Dessutom kan idéer som låg till grund för utformningen av algoritmen också användas med andra typer av Internettrafik. Inom en snar framtid är det högst sannolikt att Internet används till telefoni, tv-utsändningar och andra liknande tjänster i större skala än idag. När användningen av Internet ökar blir också sårbarheten för störningar i kommunikationen större. Resultaten från detta projekt är tänkta att hjälpa till i utvecklingen av skyddsmetoder som är tillräckligt effektiva och inte kräver ytterligare investeringar för att fungera. Den bakomliggande tanken är att det är bättre att utveckla en metod som är kostnadseffektiv att använda och är ett tillräckligt bra

skydd än en metod som fungerar utmärkt men kräver kostsam utrustning för att fungera. Detta projekt har bidragit med att framhäva att det är relevant att undersöka skyddsmetoder som är enkla att använda men kanske inte de mest effektiva. Eftersom det är sannolikt att ett företag relativt sällan utsätts för ett distribuerat överbelastningsangrepp är en skyddsmetod som kräver nya investeringar inte attraktiv att satsa på.

6.1 Framtida arbete

I sin nuvarande utformning är inte SeFFiM ett direkt applicerbart skydd som var målet i kapitel 3. Det som återstår innan detta kan bli verklighet är en fallstudie. I en fallstudie går det att undersöka om SeFFiM är applicerbar i verkligheten genom att på ett mera realistiskt sätt simulera ett angrepp än vad som är möjligt med en simulator. De approximationer som gjorts i simuleringarna behöver inte göras i en fallstudie. Exempelvis kan TCP användas istället UDP som protokoll för både angripare och användare. I detta fall kan, vilket illustreras i figur 3, utnyttjandet av handskakningsprocessens inverkan på skyddsmetodens egenskaper undersökas. Även vad som händer när webbserverns buffert med anslutningar blir full och hur skyddsmetoden hanterar detta kan undersökas om en fallstudie genomförs.

Det finns dock möjlighet att genomföra simuleringar utan approximationerna som gjorts i detta projekt. Detta kräver dock en hel del modifikationer av ns-2 och tidsramen för detta projekt tillät inte att dessa förändringar gjordes.

För att undersöka hur effektivt SeFFiM använder dataresurser och vilken prestanda skyddsmetoden har, kan en fallstudie också genomföras. Denna typ av undersökning är svår att genomföra i en simulator som ns-2 eftersom den bygger på händelser och har en egen klocka under simuleringarna och använder inte verklig tid. En prestandaundersökning är intressant eftersom en av målsättningarna i utvecklingen av SeFFiM var att göra metoden resurssnål och ändå hantera ett angrepp på ett effektivt sätt med en jämförelse med andra skyddsmetoder dylika egenskaper. Den implementation som användes i simuleringarna av de tre listorna var en standard c++ vector vilken inte den mest optimala. Inför prestandastudien måste

Inför fallstudien kan ytterligare simuleringar som undersöker variationer i användarnas trafik som till exempel en ”flash crowd” genomföras. SeFFiM klarar att klassificera användare som sänder ut för mycket trafik en kort stund, fast om det pågår en längre tid blockeras framtida trafik från användaren. Detta har inte simulerats i detta projekt och är tänkbart i en framtida undersökning. En tänkbar metod för att undersöka hur ”flash crowds” hanteras är att använda sig av trafikloggar från verkliga webbplatser. På detta sätt kan normal trafik på ett realistiskt sätt analyseras och de ett bättre undersökningsresultat än vad som är möjligt då trafiken är genererad som i simuleringarna i detta projekt.

En möjlig utveckling av SeFFiM som kan göra metoden mindre resurskrävande är att frångå att behandla varje paket i den ingående paketströmmen liknande den funktion som slumpvis väljer ut vilka paket som ska behandlas i ProbQueue av Voorhies m.fl. (2003) och/eller att införa någon form av bandbreddsbegränsning liknande den som återfinns i metoden som presenteras av Xu m.fl. (2003).

Under ett distribuerat överbelastningsangrepp är det sannolikt att angriparnas paketström av SYN-paket är mycket stor. Att behandla varje paket, som är fallet i nuvarande implementation av SeFFiM kan vara mycket resurskrävande. Det är möjligt att varje paket inte behöver behandlas utan det kan vara tillräckligt att

behandla en delmängd av paketströmmen. I den simulerade miljön fungerar det utmärkt att behandla varje paket med undantaget att det tar extra simuleringsstid att göra detta. Hur stor delmängden av paketströmmen måste vara för att åstadkomma en lika bra klassificering som när alla paket används kan påvisas med hjälp av en fallstudie.

Om fallstudien ger ett accepterbart resultat som liknar det som presenteras i kapitel 5 kan implementationen av en modul till en webbserver genomföras. Dessutom är det önskvärt om en metod som beräknar de tröskelvärden som SeFFiM använder för att undvika att dessa behöver sättas manuellt. Först då uppnås målet om ett direkt applicerbart skydd som sattes upp i kapitel 3.

7 Diskussion

I detta projekt har en simulator används för att undersöka den framtagna metoden. Resultaten visar att metoden fungerar utmärkt i den simulerade miljön, men det är långtifrån säkert att den gör det i en fallstudie. Andledningarna till detta kan vara brister i ns-2 som gör att den inte speglar verkligheten på ett bra sätt eller så kan det vara brister i simuleringsskriptet som gör att metoden framstår som den fungerar bättre än den i själva verket gör.

I den simulerade världen kan simulatorn behandla varje paket ur paketströmmen utan problem. Detta är mycket svårt att göra i ett verkligt scenario under ett angrepp. Även under normala förhållanden kan det vara problem för exempelvis en router att behandla paketen i den takt de inkommer.

Resultaten från simuleringarna med SeFFiM blev förvånansvärt bra, vilket visar på att algoritmen fungerade som det var tänkt. Tiden som gick åt för varje simulering var relativt kort och detta kan bero på en smart implementering av algoritmen.

En brist i de genomförda simuleringarna har varit hur trafiken från de inblandade datorerna har genererats. Paketen har skickats i ordning till servern istället för slumpvis som hade varit fallet i verkligheten. Detta har inte påverkat resultatet nämnvärt annat än diagrammen visar att de som kopplar upp sig under ett angrepp har fått aningen längre responstid än övriga. Denna brist hade kunnat förhindras genom att en slumpfunktion hade bestämt i vilken ordning användarna sände paketen.

Ytterligare en brist i simuleringarna har varit att de endast har körts i en upprepning och inte en mängd upprepningar med lika förutsättningar för att räkna fram ett medelvärde. Speciellt gäller detta simuleringen när en användare kopplar upp sig i taget. För att ge ett mer korrekt resultat av simuleringarna med få användare kan en slumpfunktion användas. Slumpfunktionen kan använda olika frön[5] för att generera trafiken i de olika upprepningarna. På så sätt kan ett medelvärde räknas fram med hjälp av en mängd simuleringar istället för en enda som var fallet i detta projekt. Troligen hade resultatet inte skilt sig nämnvärt men det hade gett mera trovärdighet och säkrare slutsatser hade kunnat dras av simuleringarna med ett fåtal användare. Hur en sådan implementering skulle se ut i ns-2 fanns inte tid till att komma fram till men anses vara fullt genomförbart med en större tidsram för implementeringsfasen av projektet.

Approximationerna som gjorts i detta projekt hade kunnat undvikas. Till exempel hade en modifiering av TCP-protokollet mycket väl kunna genomföras och det skulle gå att genomföra simuleringar där antalet anslutningar i serverns buffert togs hänsyn till istället för enbart paketströmmen. Anledningen till att approximationerna gjordes var främst att den avsatta tiden för projektet var begränsad, speciellt i implementationsfasen och att det fanns flera exempel på andra arbeten där liknande approximationer gjorts.

Algoritmens uppbyggnad i SeFFiM med tre separata listor och förflyttningar mellan dessa kan ge upphov till prestandaförsämringar. Speciellt när ett IP-nummer ska flyttas, då detta kräver ett flertal operationer på de aktuella listorna. Om framtida tester visar att ett IP-nummer ofta flyttas mellan listorna kan det vara aktuellt att undersöka om en metod som använder en lista är mindre resurskrävande och ger

[5] Startvärde som används i uträkningen av slumptalet.

bättre prestanda. I detta fall skulle klassificeringen av ett IP-nummer representeras med ytterligare ett fält i varje element. Denna lösning kan dock leda till att listan blir ohanterligt stor om det är många IP-nummer som är aktuella på webbservern samtidigt. Om listan är relativt liten kan det ge prestandavinster, gentemot lösningen med tre listor, genom att element inte behöver skapas och tas bort upprepade gånger under ett angrepp. I simuleringarna som genomfördes till detta projekt fungerade varianten med tre listor bra och empiriska erfarenheter av simuleringarna var att de slutfördes på relativt kort tid och var inte nämnvärt minneskrävande. Däremot var simuleringarna med ProbQueue mycket minneskrävande och tog betydligt längre tid att genomföra med identiska förutsättningar. Vad detta beror på är oklart eftersom prestanda och resurstester inte genomförts, men det kan bero på att en router inte användes vid simuleringarna som kasserade paket från paketströmmen eller att nya element skapades i listan vid varje paketankomst.

I diagrammen över resultaten saknas en hel del värden. Detta beror med största sannolikhet på att pinganropet tog för lång tid och paketet inte kom fram (eng. timeout). Implementationen av pingprotokollet som användes var inbyggd i ns-2 och saknade inställningsmöjligheter för timeout-tiden. Om denna implementation hade modifierats hade en inställbar tidsgräns varit möjlig att använda och en del luckor i diagrammen hade kunnat fyllas med relevanta värden istället.

Arbetet med projektet i övrigt har gått bra. Information, i form av artiklar och böcker, om ämnet har inte varit svårt att hitta och det har varit intressant att lära sig om problematiken kring ämnet. Dessutom har det känts aktuellt och relevant att arbeta med. Nackdelen med omfattningen på informationsmaterialet har varit att det varit svårt att hitta en problemställning som inte använts i tidigare undersökningar. Därför föll valet på en serverbaserad lösning eftersom det fanns få existerande lösningar som liknande denna.

Tidsåtgången har varit svår att förutsäga och den avsatta tiden till de olika momenten hade med största sannolikhet ändrats om ett liknande projektet genomförts på nytt. Speciellt tiden som gick åt till att undersöka hur ns-2 fungerade och hur algoritmen skulle implementeras tog mycket längre tid än beräknat. Detta hade kunnat undvikas om kunskap om ns-2 hade inhämtats före projektets start men det är svårt att veta vilken metod som är lämpligt för att lösa projektets problem innan problemet definierats ordentligt.

Referenser

Badre, A.N. (2002) *Shaping web usability*, Pearson Education, Inc.

Black, U (1994) *TCP/IP & related protocols (2nd edition)*, McGraw-Hill Inc.

Bellowin, S. M. (1989) Security problems in the TCP/IP Protocol suite. *Computer Communication Review, 19,* 32-48.

Cheng, J., Wang, H. & Shin, K.G. (2003) Hop-count filtering: an effective defense against spoofed DDoS traffic, *Proceedings of the 10th ACM conference on Computer and communication security*.

Chung, J. & Claypool, M. (2003) *Ns by Example*. Tillgänglig på Internet: http://nile.wpi.edu/NS/ [Hämtad 2004-03-28]

Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001) *Introduction to algorithms* (2nd edition) The Massachusetts Institute of Technology.

Elmasri, R. & Navathe, S. (2000) *Fundamentals of database systems* (3rd edition) Addison-Wesley

Eronen, P (2000) Denial of service in public key protocols, *In proceedings of the Helsinki University of Technology seminar on network security, 2000.*

Forouzan, B.A. (2003) *TCP/IP protocol suite* (2nd edition), McGraw-Hill Higher Education.

Galvin, S. (1999) *Operating system concepts* (5th edition), John Wiley & Sons Inc.

Halsall, F. (2001) *Multimedia communications: Applications, networks, protocols and standards,* Pearson Education Limited.

Kargi, F., Maier, J. & Weber, M. (2001) Protecting web servers from distributed denial of service attacks. *In proceedings of 10th international WWW conference 2001.*

Lan, K., Hussain, A. & Dutta, D. (2003) Effect of malicious traffic on the network, *Passive and Active Measurement Workshop (PAM), San Diego, April, 2003.*

Laprie, J.C (1994) Dependability: Basic concepts and terminology, *IFIP WG 10.4 – Dependable computing and fault tolerance.*

Law, A. & Kelton, W. (2000) *Simulation modeling and analysis* The McGraw-Hill Companies Inc.

Lejeune, M. A (2002) Awareness of distributed denial of service attacks' dangers: role of Internet pricing mechanisms. *Netnomics, Kluwer Academic Publishers, 4,* 145-162.

McClure, S., Scambray, J. & Kurtz, G. (2003) *Hacking i focus* (4th edition). Pagina förlags AB.

Parker, P. & Sportack, M. (2000) *TCP/IP Unleashed* (2nd edition). Sams Publishing.

Peng, T., Leckie, C. & Ramamohanarao, K. (2002a) Detecting distributed denial of service attacks by sharing distributed beliefs. *9th IEEE International Conference on Telecommunications, Beijing, China, June, 2002.*

Peng, T., Leckie, C. & Ramamohanarao, K. (2002b) Detecting distributed denial of service attacks using source IP address monitoring. *Submitted to Infocom 2004, Hongkong, P.R. China, March, 2004.*

Peng, T., Leckie, C. & Ramamohanarao, K. (2003) Protection from Distributed Denial of Service Attack Using History-based IP Filtering, *IEEE International Conference on Communications (ICC 2003)*

Russel, D. & Gangemi Sr., G.T. (1991) *Computer security basics*. O'Reilly & Associates Inc.

Savage, S., Wetherall, D., Karlin, A. & Anderson, T. (2000) Practical support for IP Traceback. *SIGCOMM'00 Stockholm, Sweden*

Skansholm, J. (2000) *C++ direkt*, Studentlitteraur, Lund

Stein, L.D., Stewart, J.N. (2002) *The world wide web security faq*, W3C Tillgängligt på Internet: http://www.w3.org/Security/faq/ [Hämtad 2004-03-15].

Tanachaiwiwat, S. & Hwang, K. (2003) Differential packet filtering against DDoS flood attacks. *ACM Computer Conference, 2003.*

The network simulator - ns-2 (Version 2.26) (2003) [Datorprogram] VINT project Tillgänglig på Internet http://www.isi.edu/nsnam/ns/ [Hämtad 2004-04-01]

Tupakula, U. & Varadharajan, V. (2003) A practical method to counteract denial of service attacks. *25th Australasian Computer Science Conference (ACSC2003)*

Voorhies, S., Hyunyoung, L. & Klappenecker, A. (2003) A probabilistic defense against distributed denial of service attacks. *Submitted to a journal, Department of Computer Science, University of Denver.*

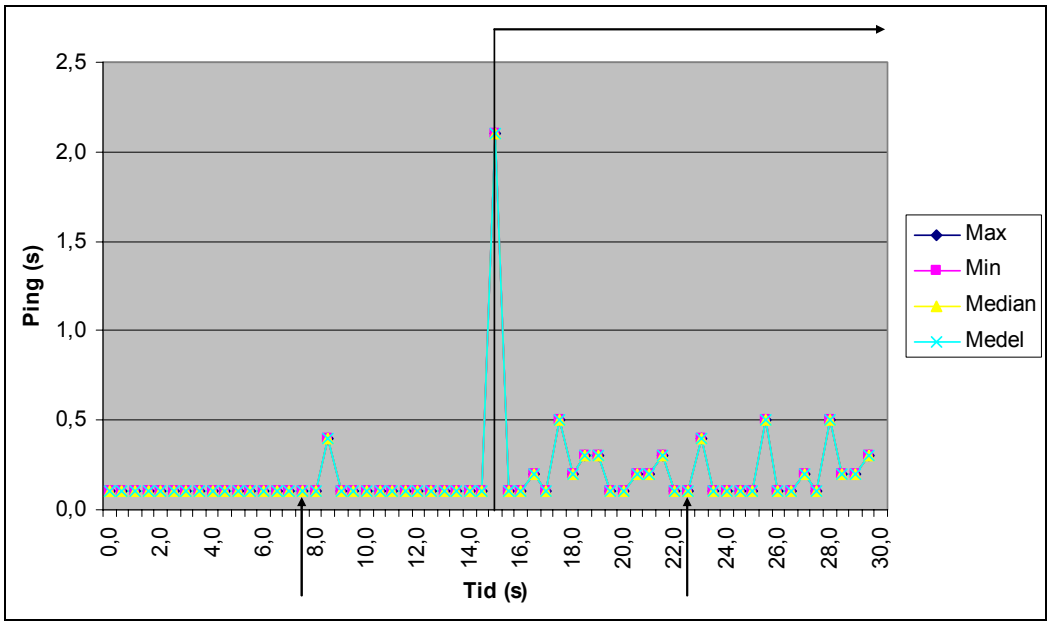
Wang, H., Zhang, D. & Shin, K.G. (2002) Detecting SYN flooding attacks. *Proceedings of IEEE Infocom June 2002*

Yan, D., Lui, J. & Liang, F. (2002) Defending against distributed denial of service attacks with max-min fair server centric router throttles. *Proceedings of IEEE International Workshop on Quality of Service (IWQoS), Miami Beach, FL, May 2002.*

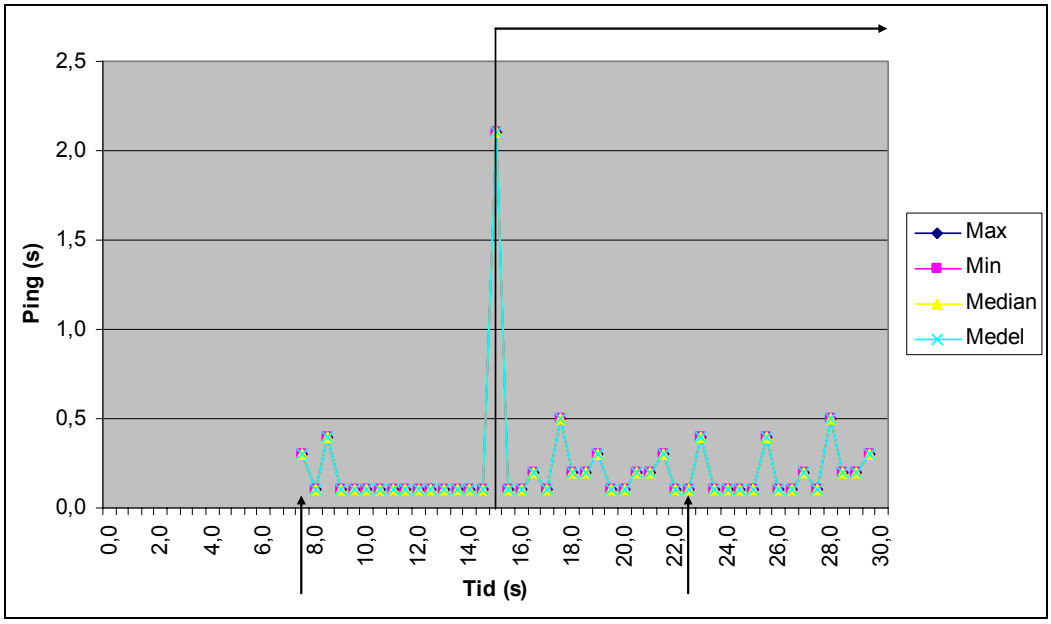
Xu, J. & Lee, W. (2003) Sustaining availability of web services under distributed denial of service attacks. *IEEE Transactions on computers, 52:2,* 195-208.

Ye, N. (2002) QoS-centric stateful resource management in information systems. *Netnomics, Kluwer Academic Publishers, 4:2,* 149-160.

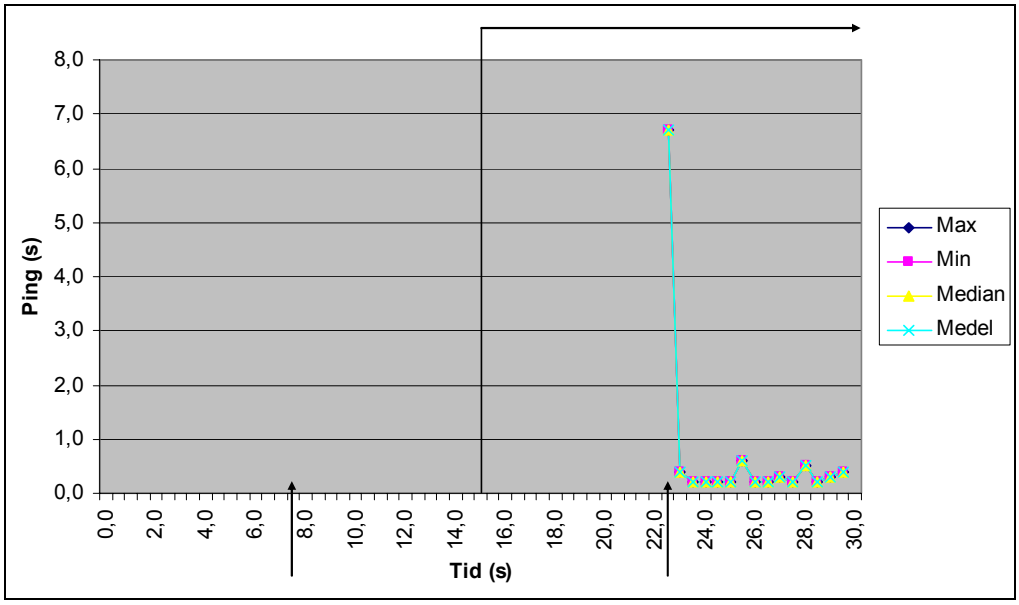
Appendix A - Grafer



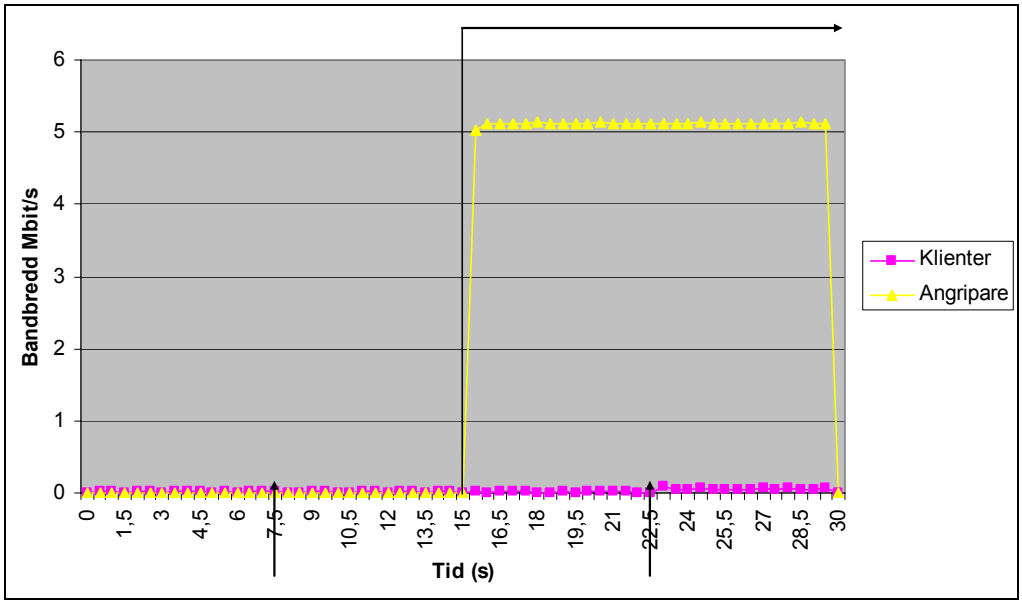
Figur A 1 - Simulering med 10 angripare och 3 användare utan skyddsmetod (DropTail) och visar användare som är uppkopplade innan angreppet.



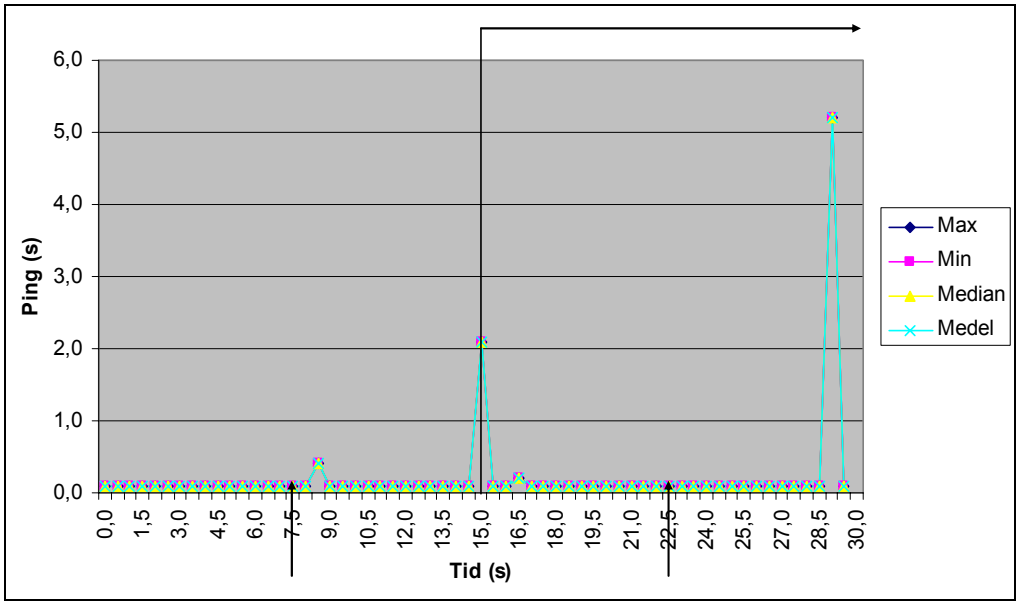
Figur A 2 - Simulering med 10 angripare och 3 användare utan skyddsmetod (DropTail) och visar användare som kopplar upp innan angreppet.



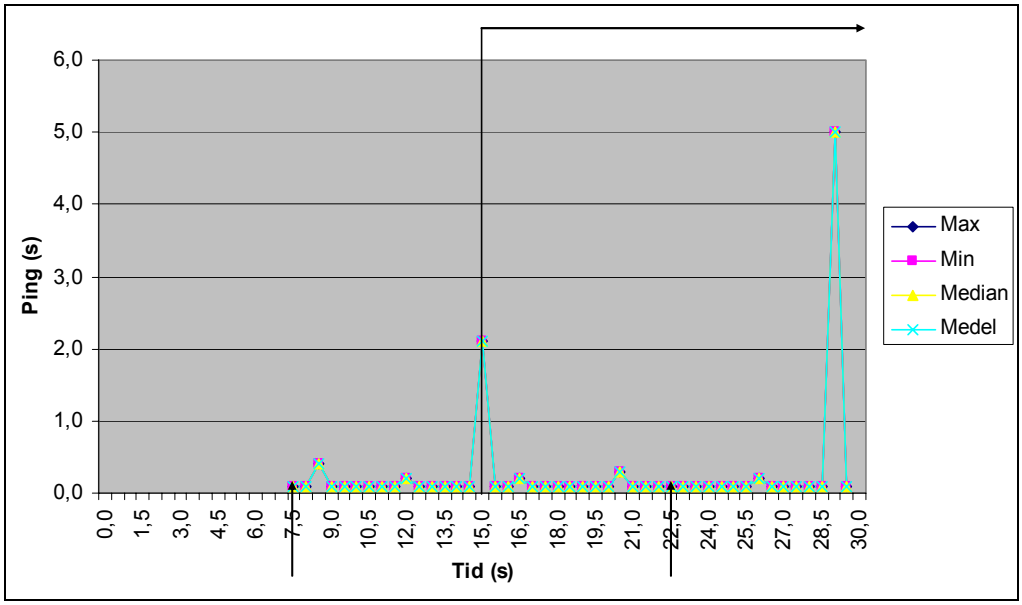
Figur A 3 - Simulering med 10 angripare och 3 användare utan skyddsmetod (DropTail) och visar användare som kopplar upp under angreppet.



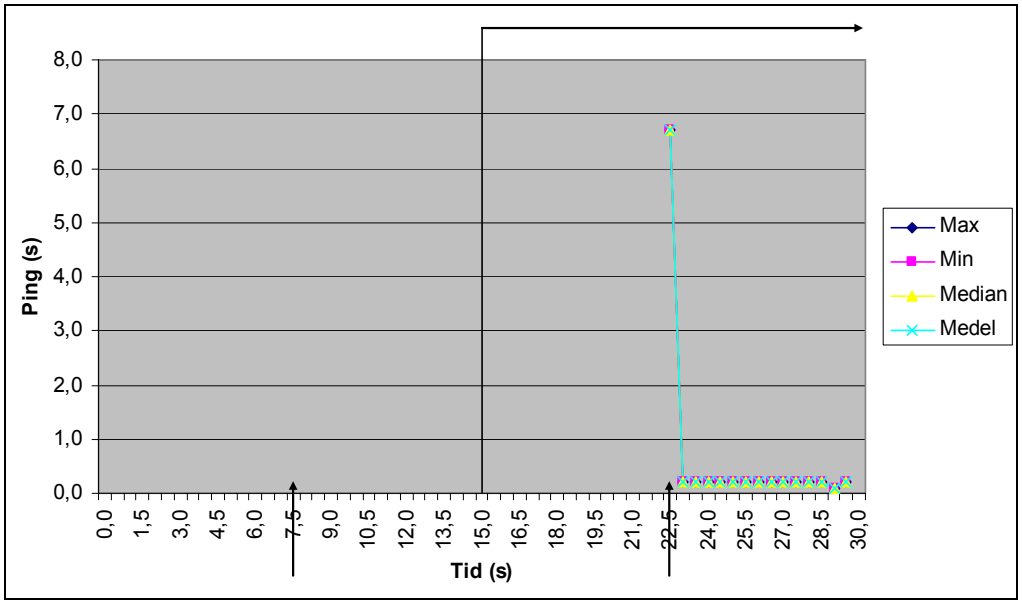
Figur A 4 - Bandbredd används av 10 angripare och 3 användare utan skyddsmetod (DropTail).



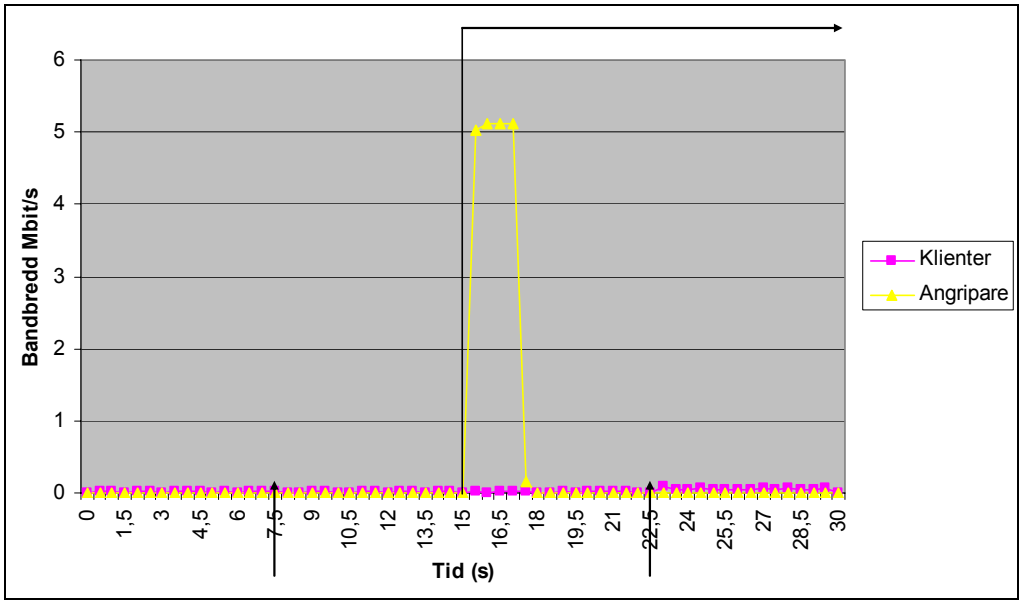
Figur A 5 - Simulering med 10 angripare och 3 användare med SeFFim som skyddsmetod och visar användare som är uppkopplade innan angreppet.



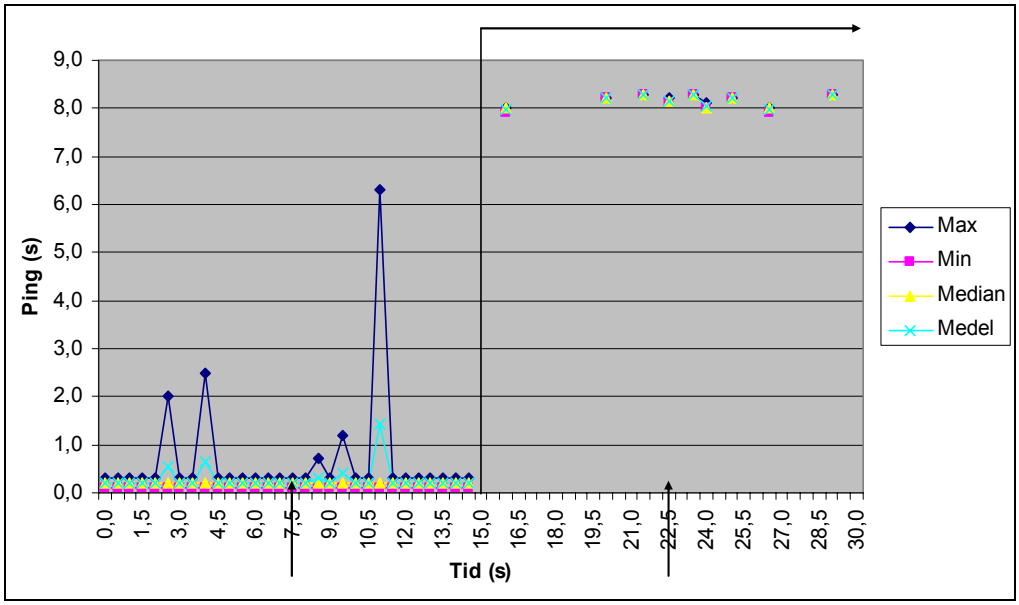
Figur A 6 - Simulering med 10 angripare och 3 användare med SeFFim som skyddsmetod och visar användare som kopplar upp innan angreppet.



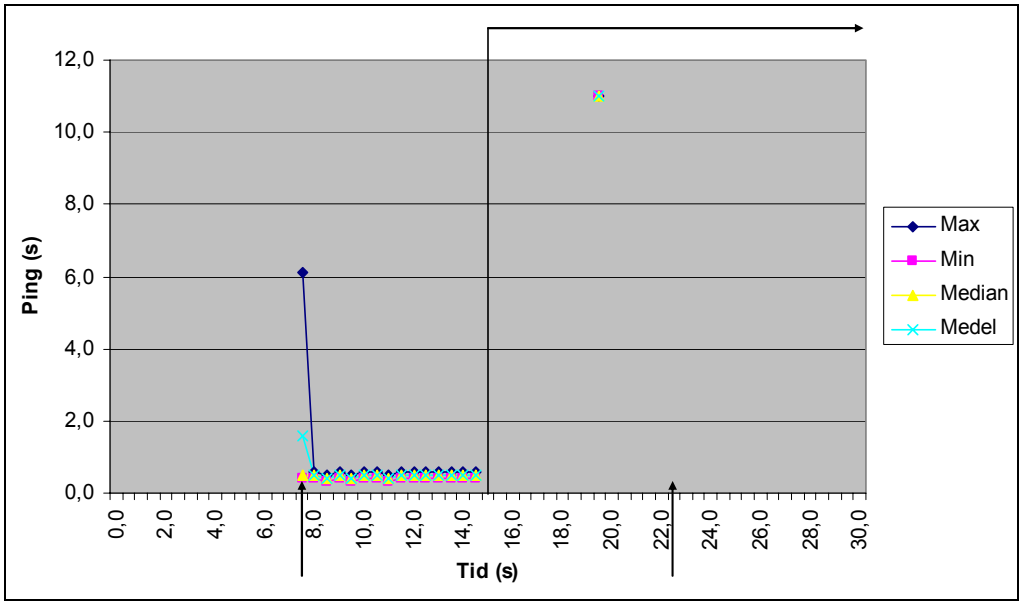
Figur A 7 - Simulering med 10 angripare och 3 användare med SeFFim som skyddsmetod och visar användare som kopplar upp under angreppet.



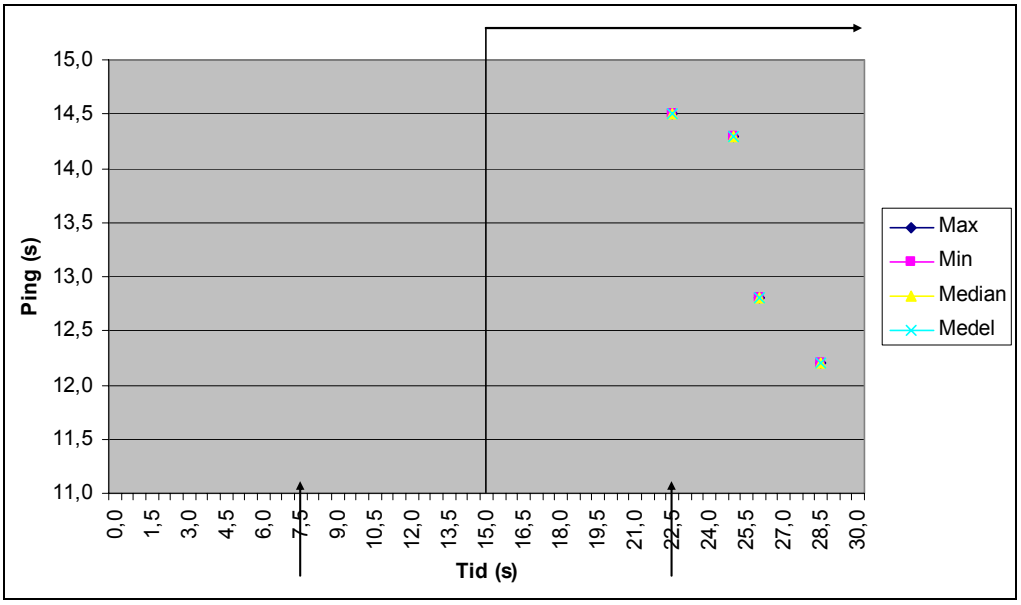
Figur A 8 - Bandbredd används av 10 angripare och 3 användare med SeFFiM som skyddsmetod.



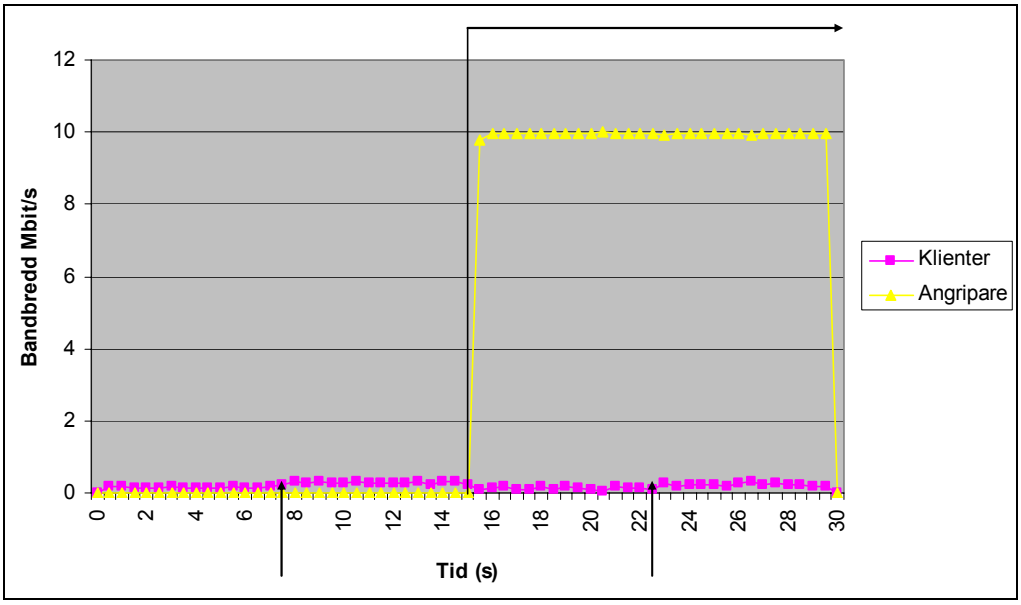
Figur A 9 - Simulering med 100 angripare och 20 användare utan skyddsmetod (DropTail) och visar användare som är uppkopplade innan angreppet.



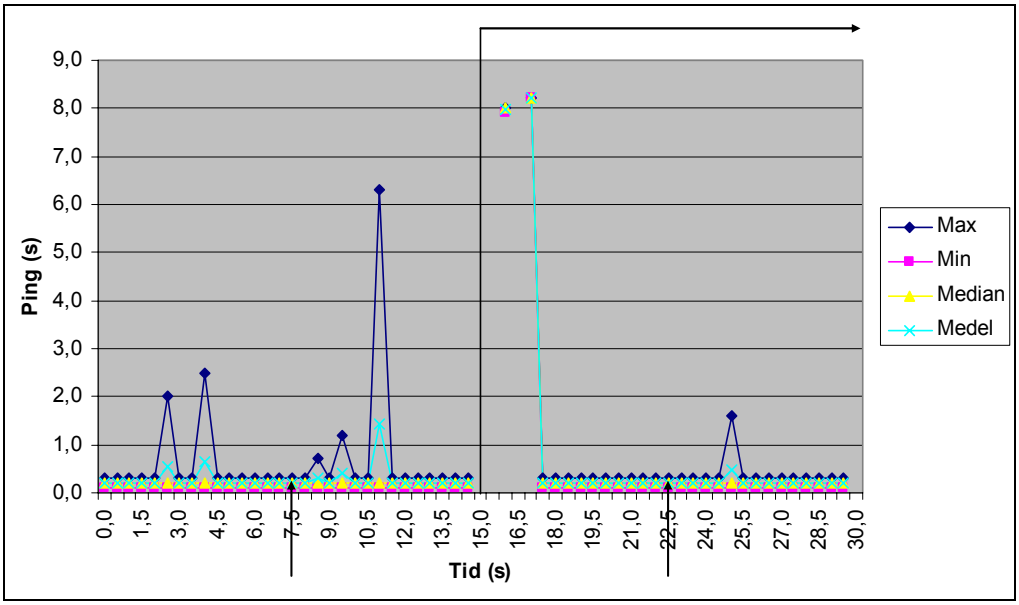
Figur A 10 - Simulering med 100 angripare och 20 användare utan skyddsmetod (DropTail) och visar användare som kopplar upp innan angreppet.



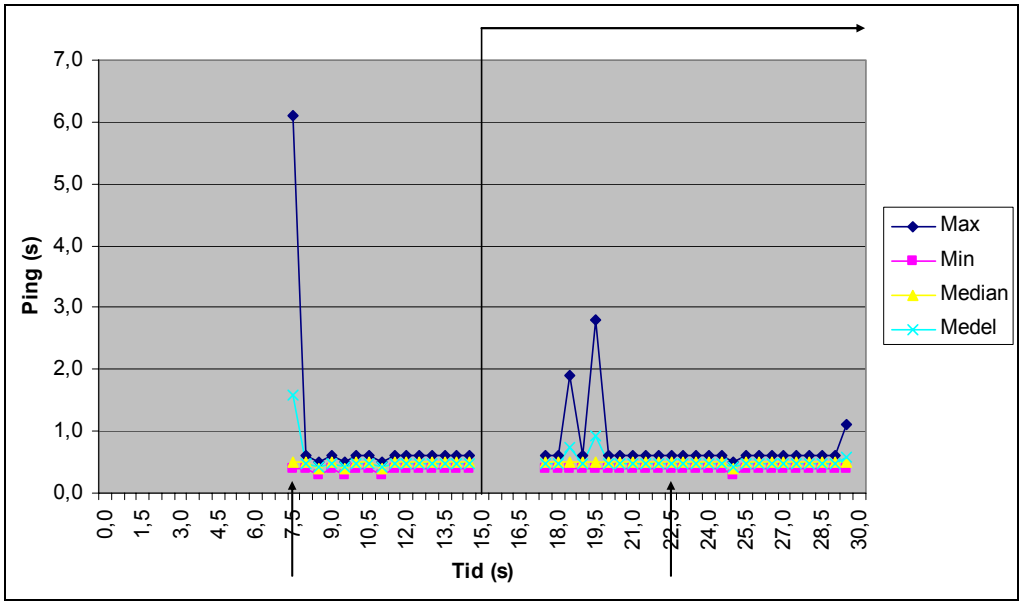
Figur A 11 - Simulering med 100 angripare och 20 användare utan skyddsmetod (DropTail) och visar användare som kopplar upp efter angreppet.



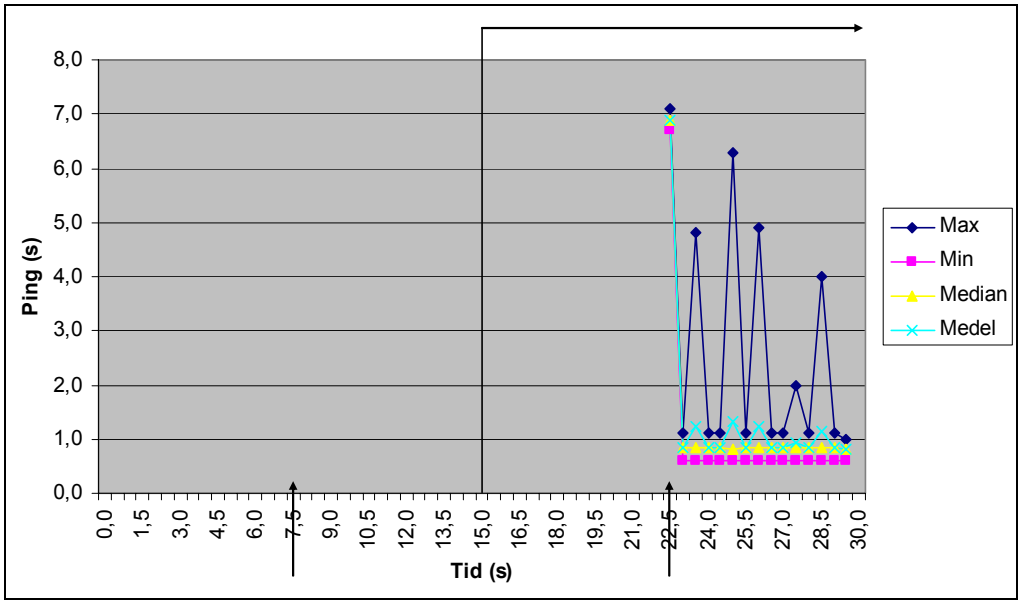
Figur A 12 - Bandbredd används av 100 angripare och 20 användare utan skyddsmetod (DropTail).



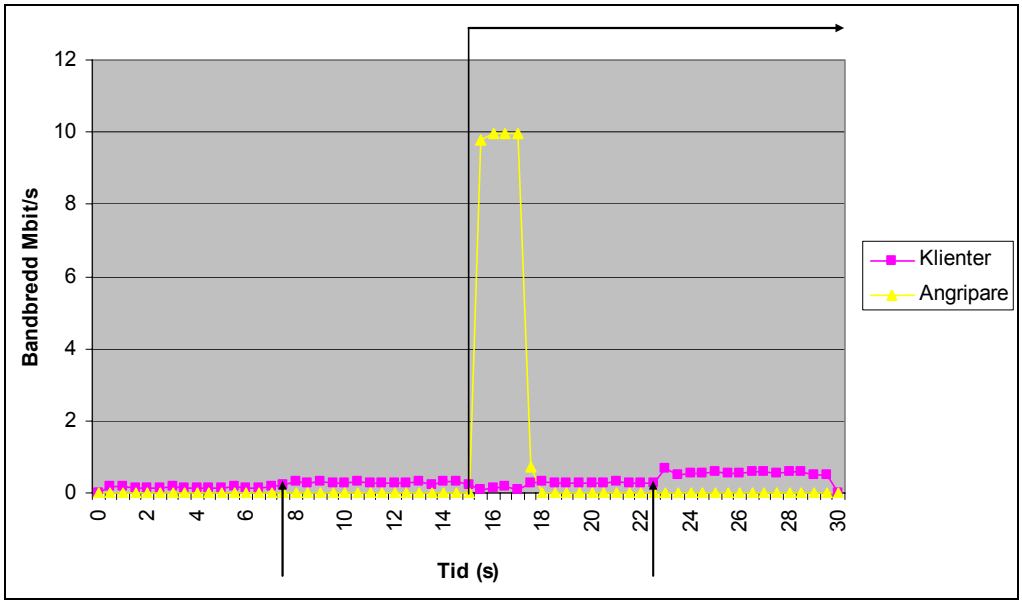
Figur A 13 - Simulering med 100 angripare och 20 användare med SeFFim som skyddsmetod och visar användare som är uppkopplade innan angreppet.



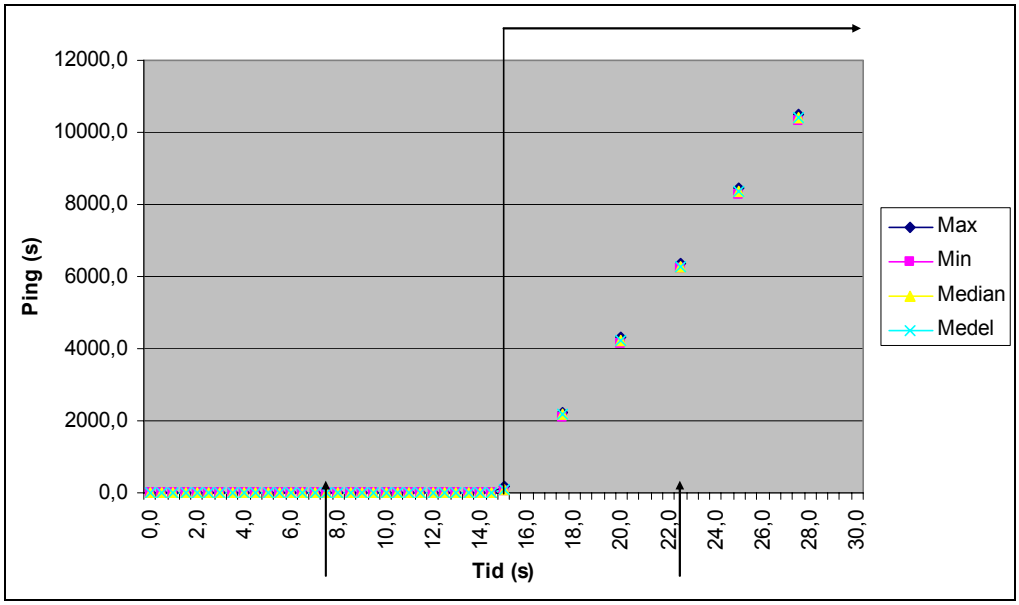
Figur A 14 - Simulering med 100 angripare och 20 användare med SeFFim som skyddsmetod och visar användare som kopplar upp innan angreppet.



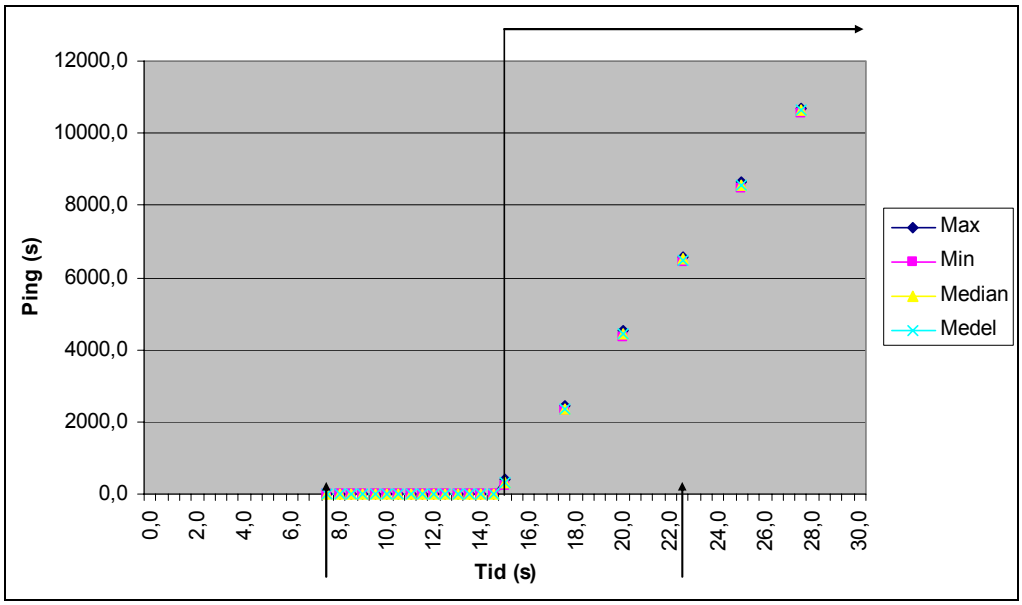
Figur A 15 - Simulering med 100 angripare och 20 användare med SeFFim som skyddsmetod och visar användare som kopplar upp under angreppet.



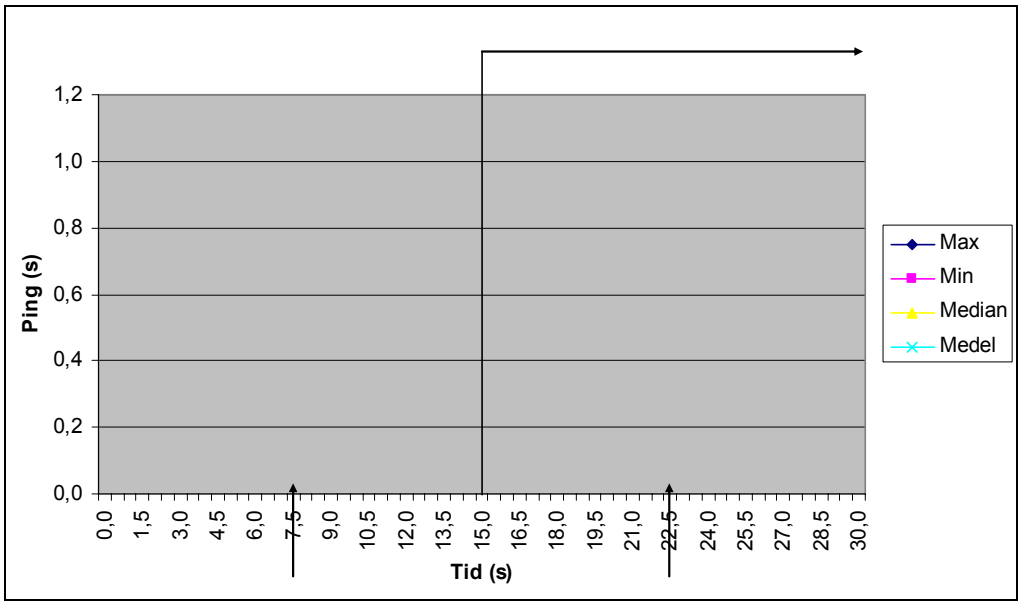
Figur A 16 - Bandbredd används av 100 angripare och 20 användare med SeFFiM som skyddsmetod.



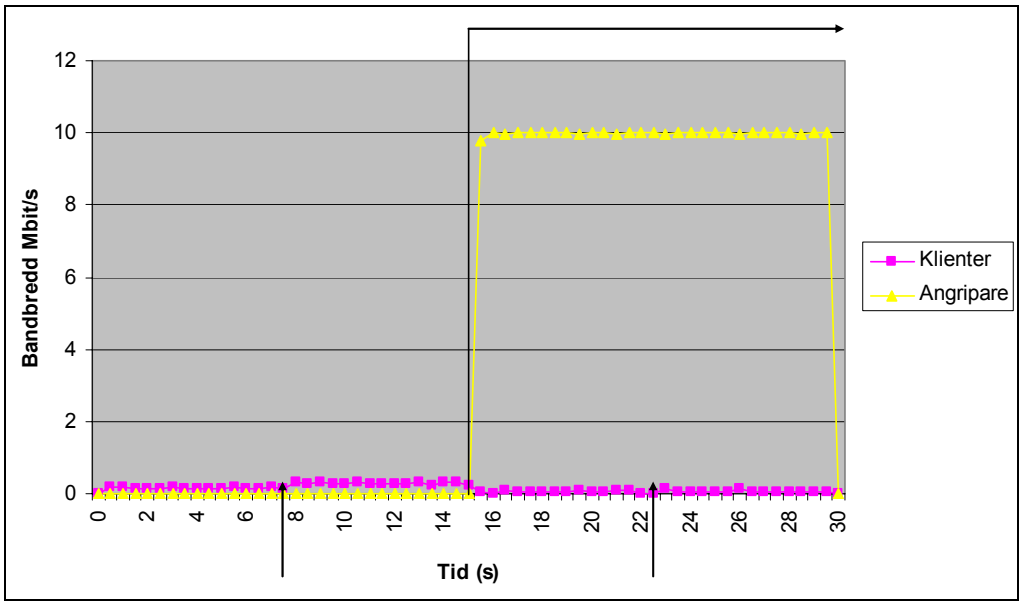
Figur A 17 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 50 och visar användare som är uppkopplade före angreppet.



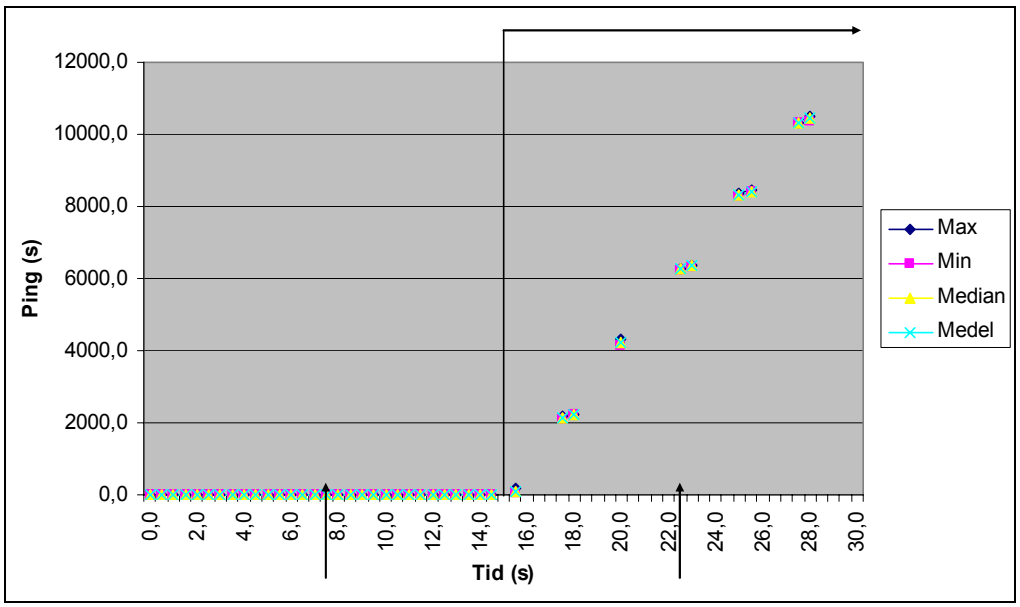
Figur A 18 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 50 och visar användare som kopplar upp före angreppet.



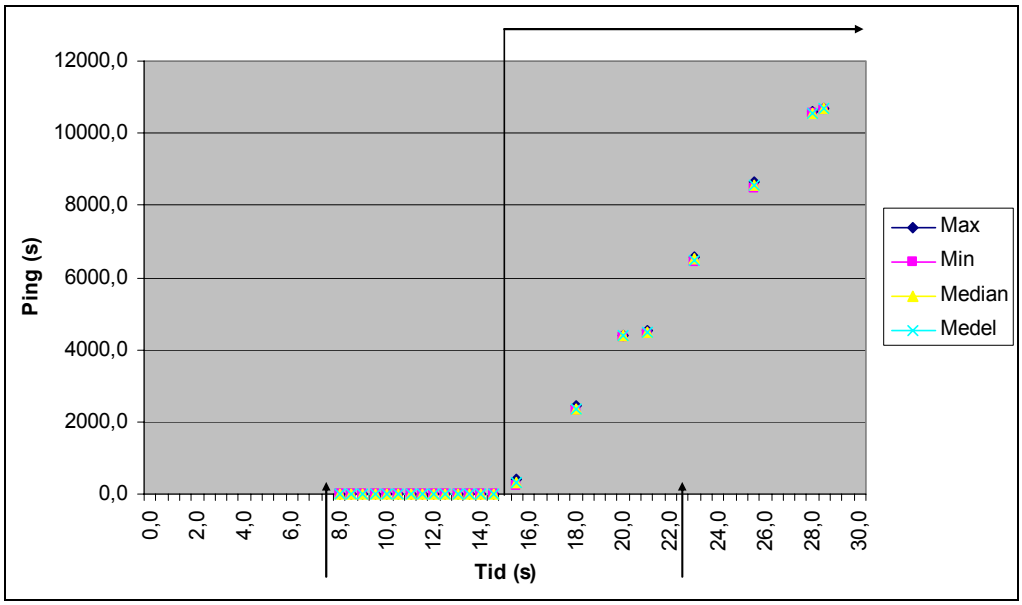
Figur A 19 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 50 och visar användare som kopplar upp under angreppet.



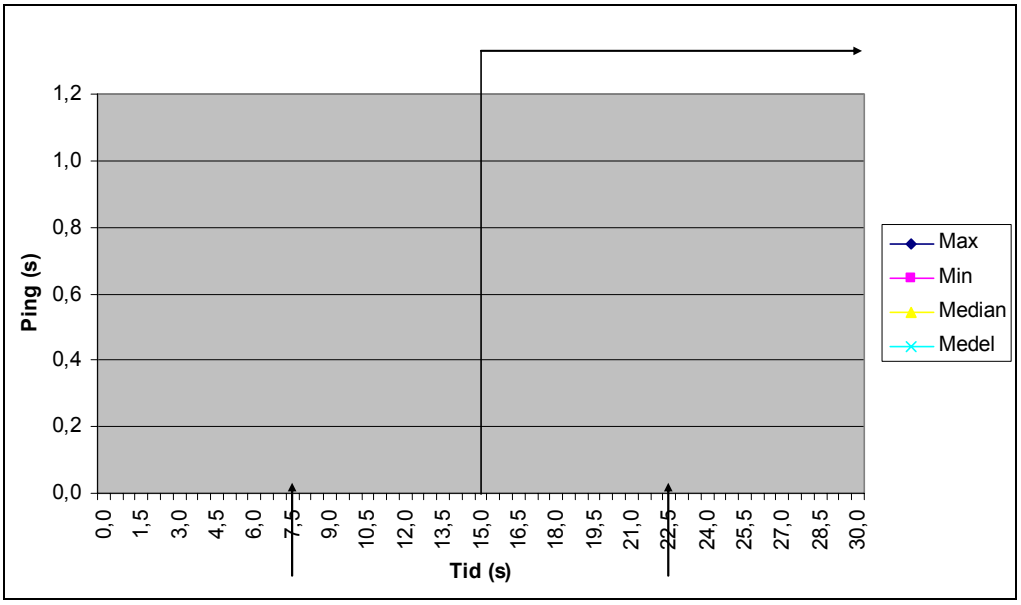
Figur A 20 - Bandbredd används av 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 50.



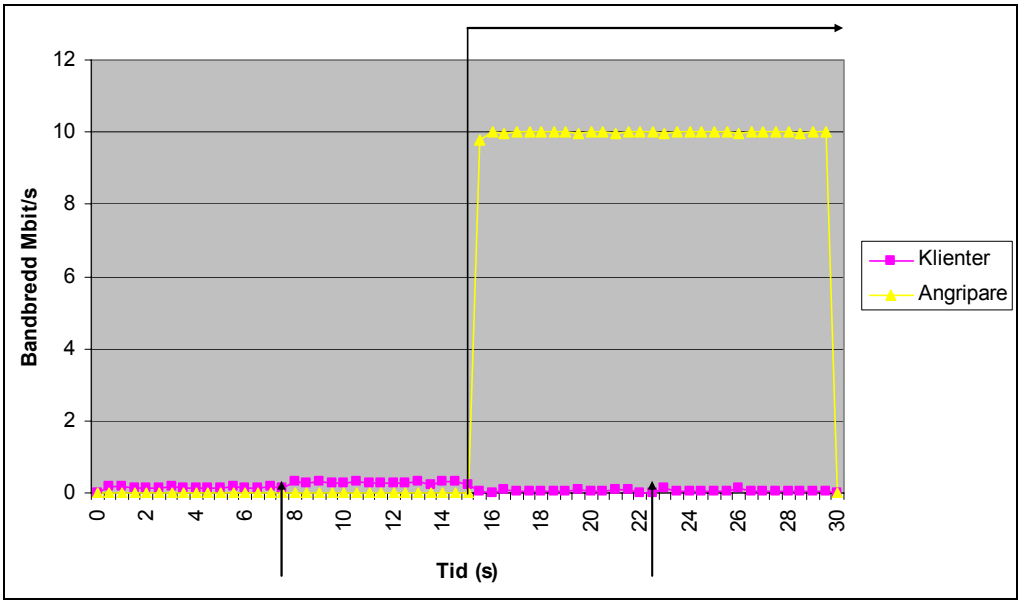
Figur A 21 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 100 och visar användare som är uppkopplade före angreppet.



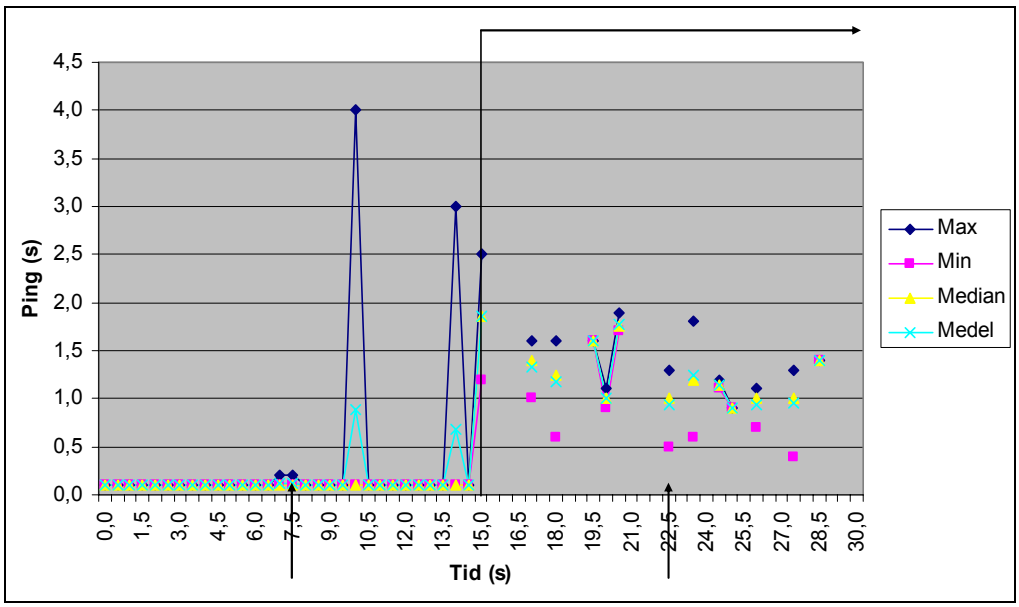
Figur A 22 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 100 och visar användare som kopplar upp före angreppet.



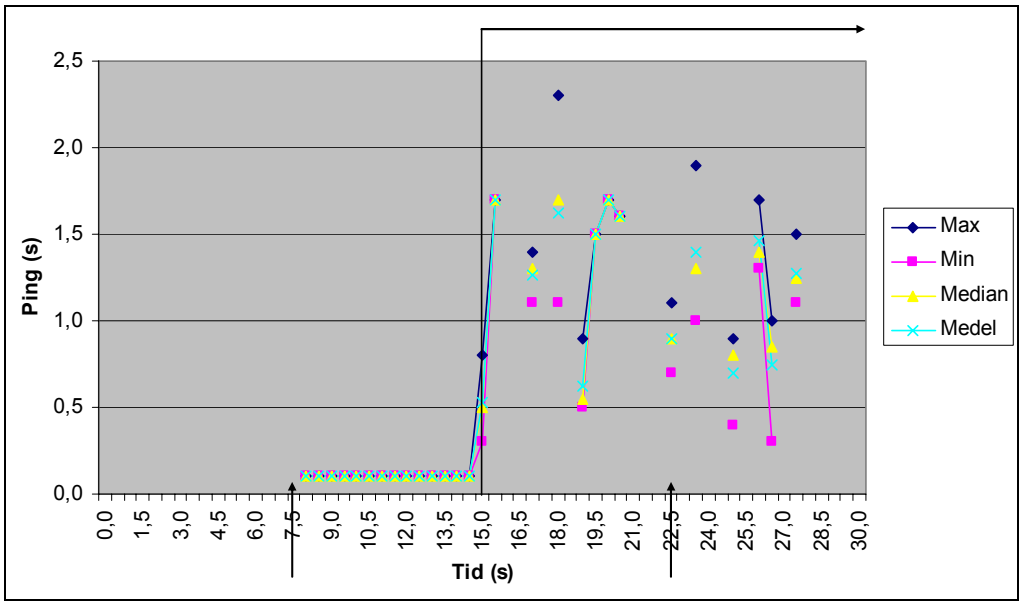
Figur A 23 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 100 och visar användare som kopplar upp under angreppet.



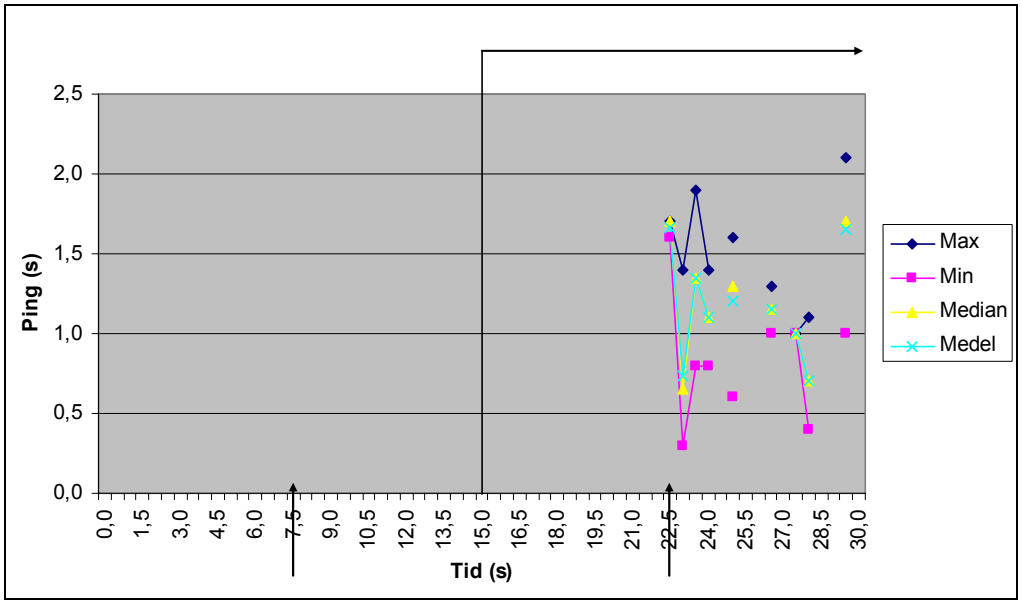
Figur A 24 - Bandbredd används av 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 75 och tabellstorlek 100.



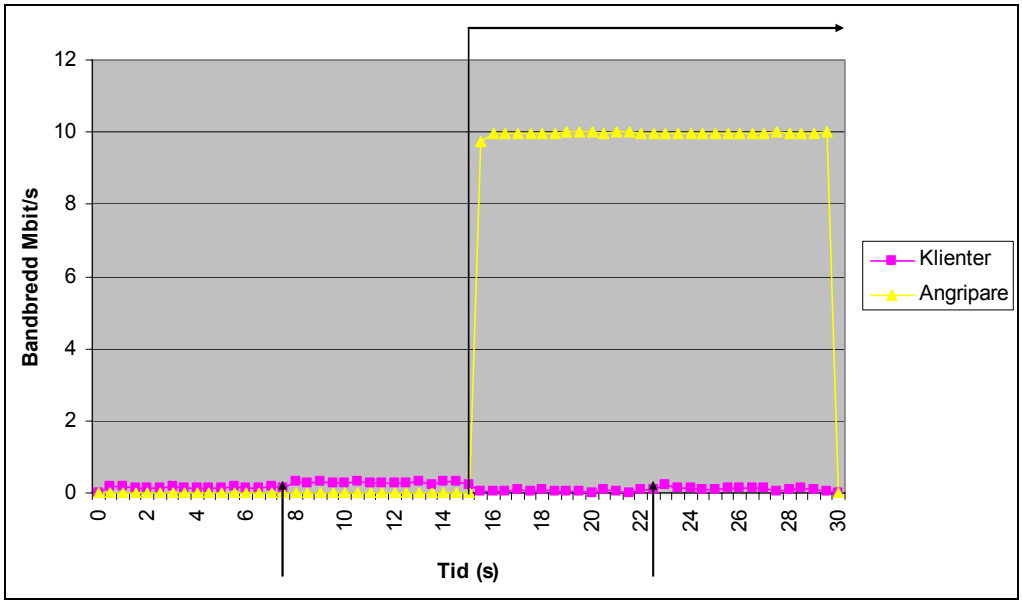
Figur A 25 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 50 och visar användare som är uppkopplade före angreppet.



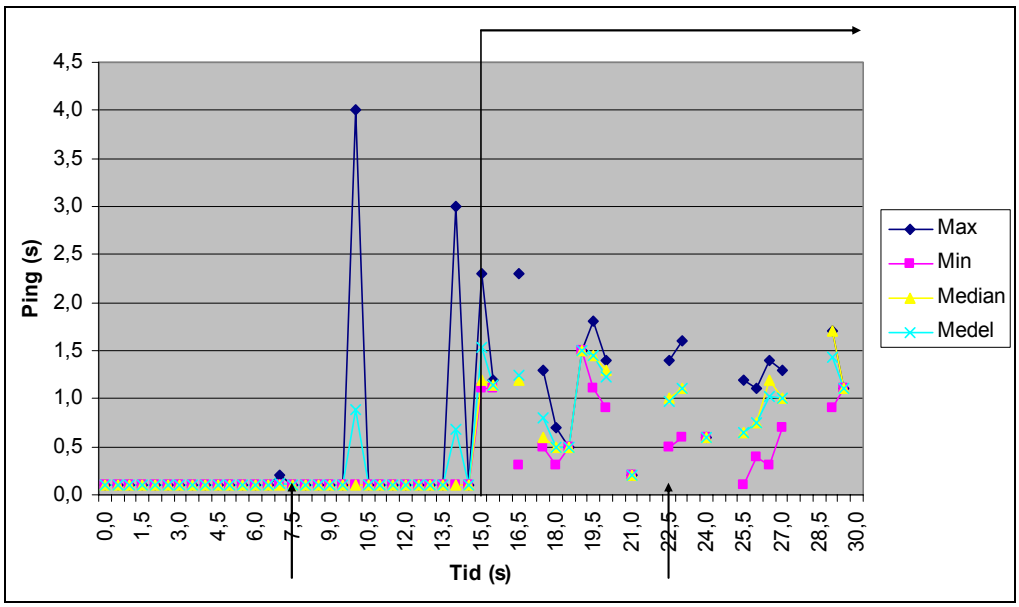
Figur A 26 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 50 och visar användare som kopplar upp före angreppet.



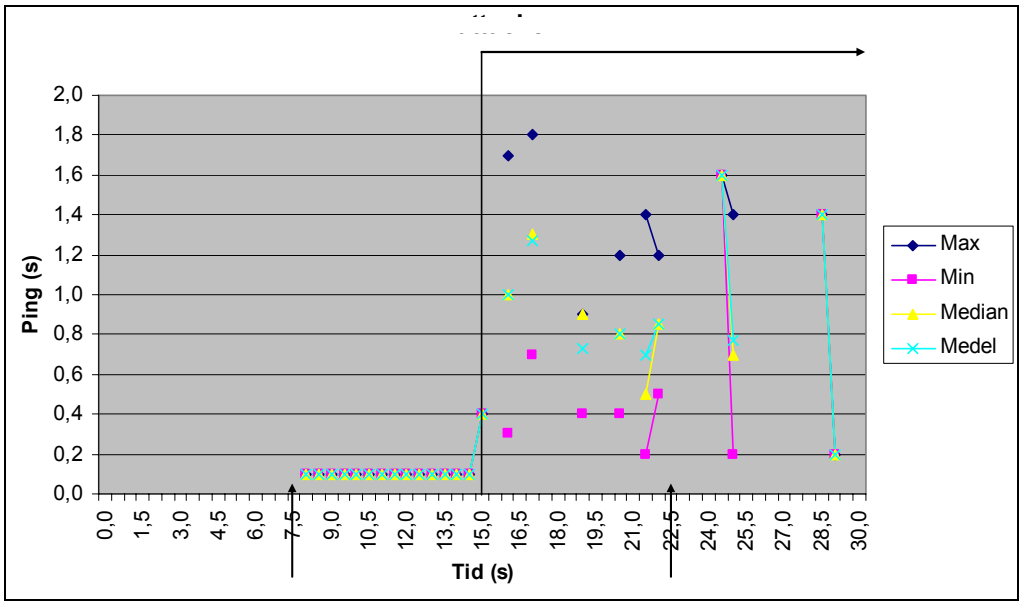
Figur A 27 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 50 och visar användare som kopplar upp under angreppet.



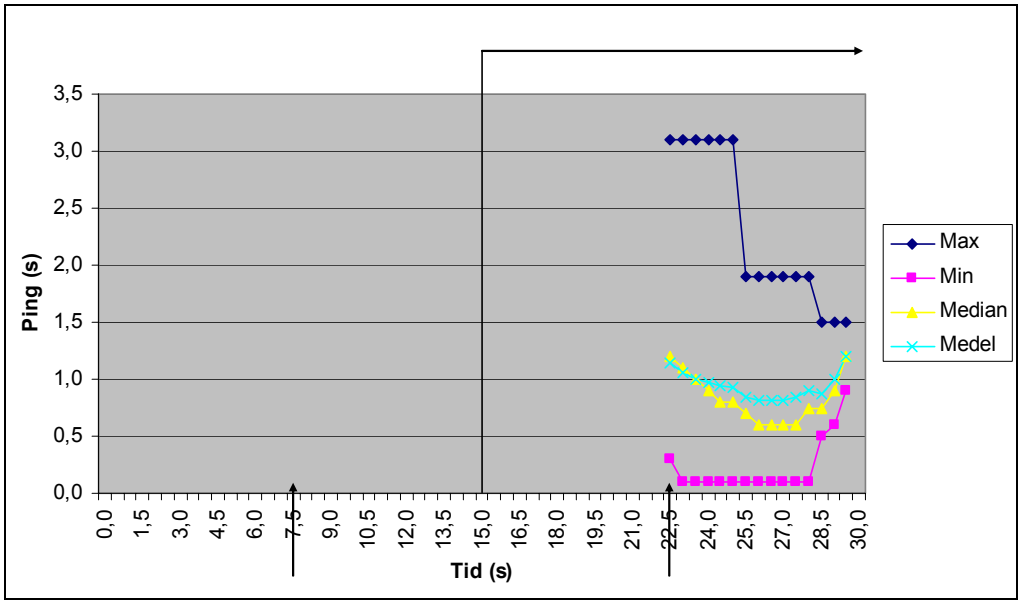
Figur A 28 - Bandbredd används av 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 50.



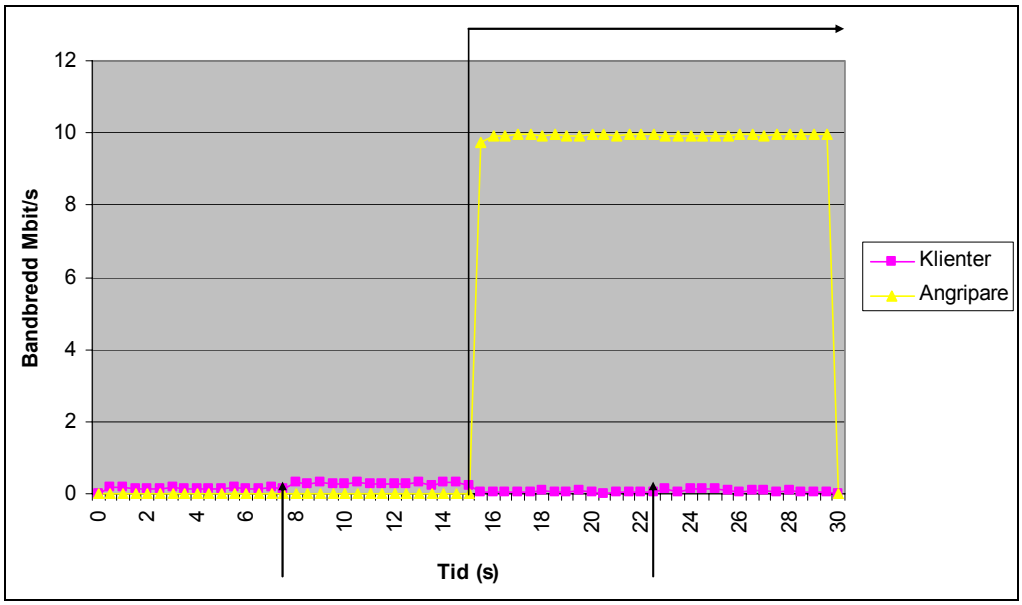
Figur A 29 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 100 och visar användare som är uppkopplade före angreppet.



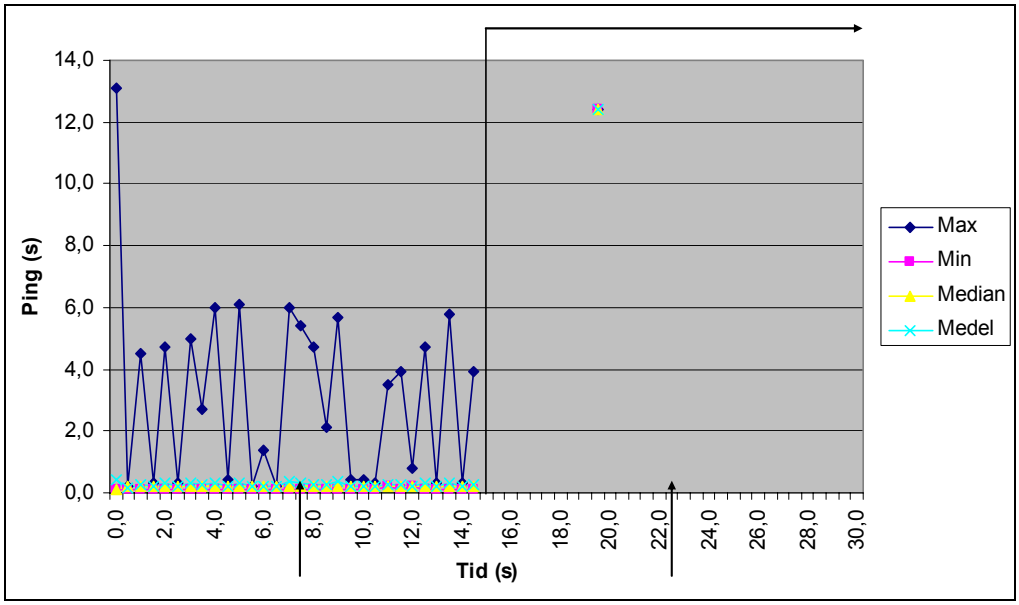
Figur A 30 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 100 och visar användare som kopplar upp före angreppet.



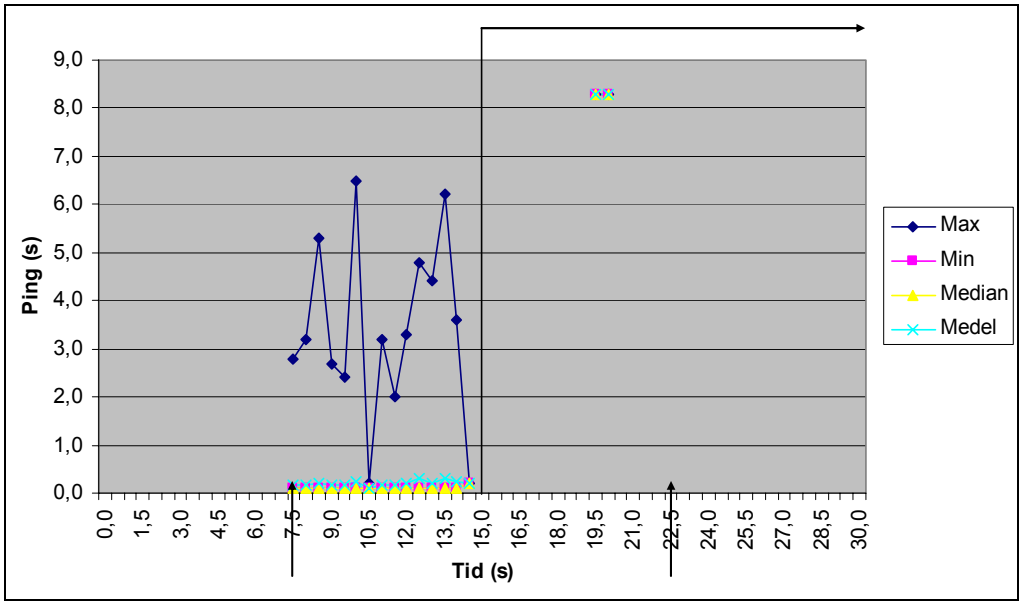
Figur A 31 - Simulering med 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 100 och visar användare som kopplar upp under angreppet.



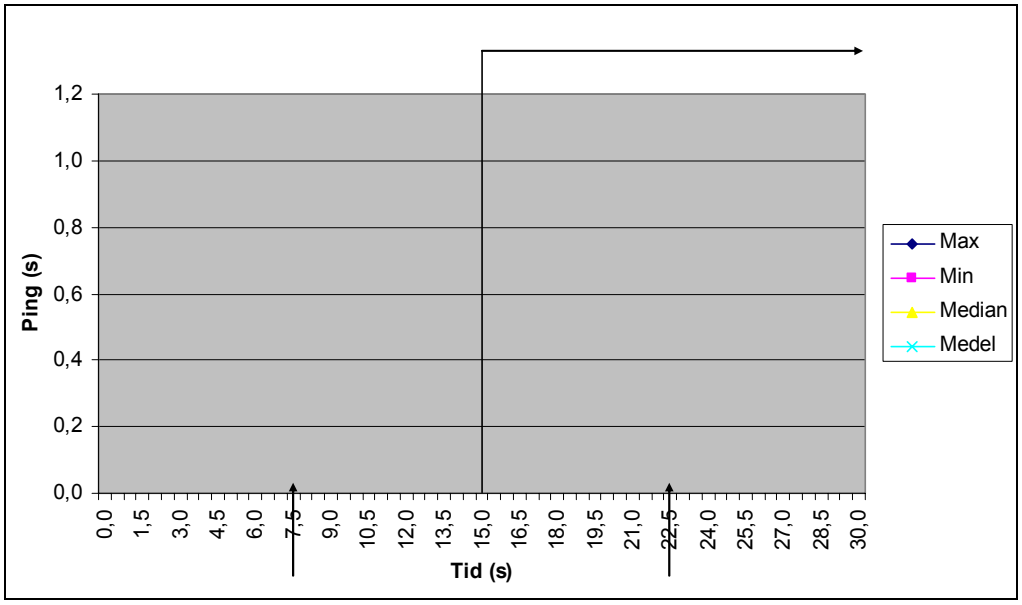
Figur A 32 - Bandbredd används av 100 angripare och 20 användare med ProbQueue som skyddsmetod med sannolikhet 100 och tabellstorlek 100.



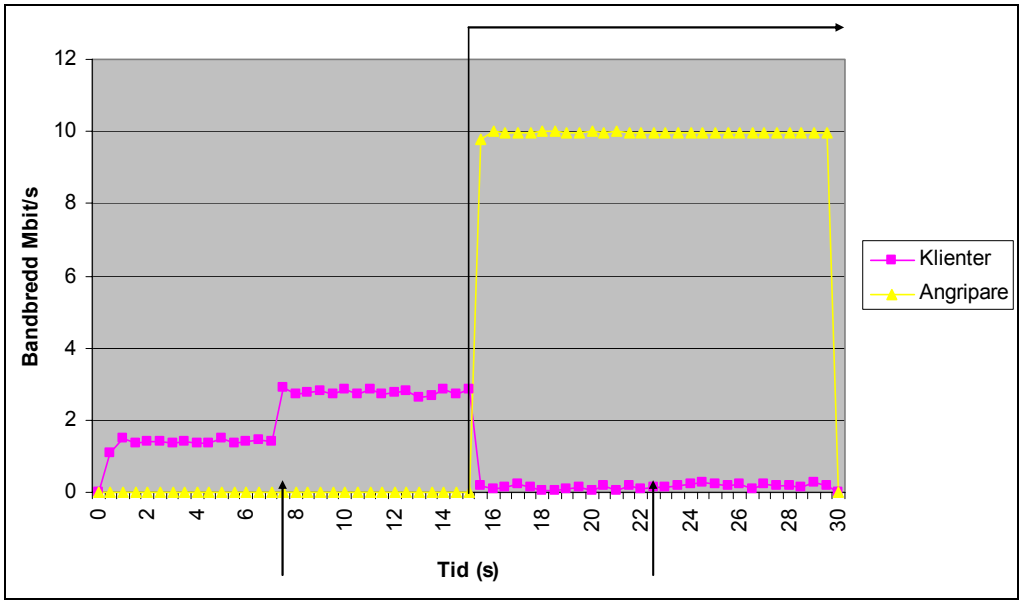
Figur A 33 - Simulering med 1000 angripare och 200 användare utan skyddsmetod (DropTail) och visar användare som är uppkopplade innan angreppet.



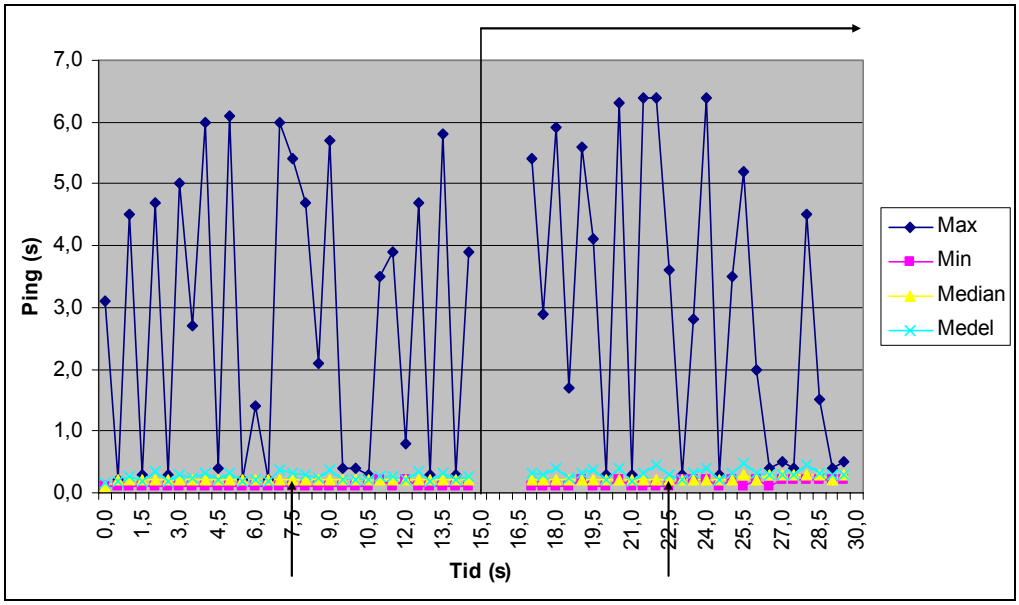
Figur A 34 - Simulering med 1000 angripare och 200 användare utan skyddsmetod (DropTail) och visar användare som kopplar upp innan angreppet.



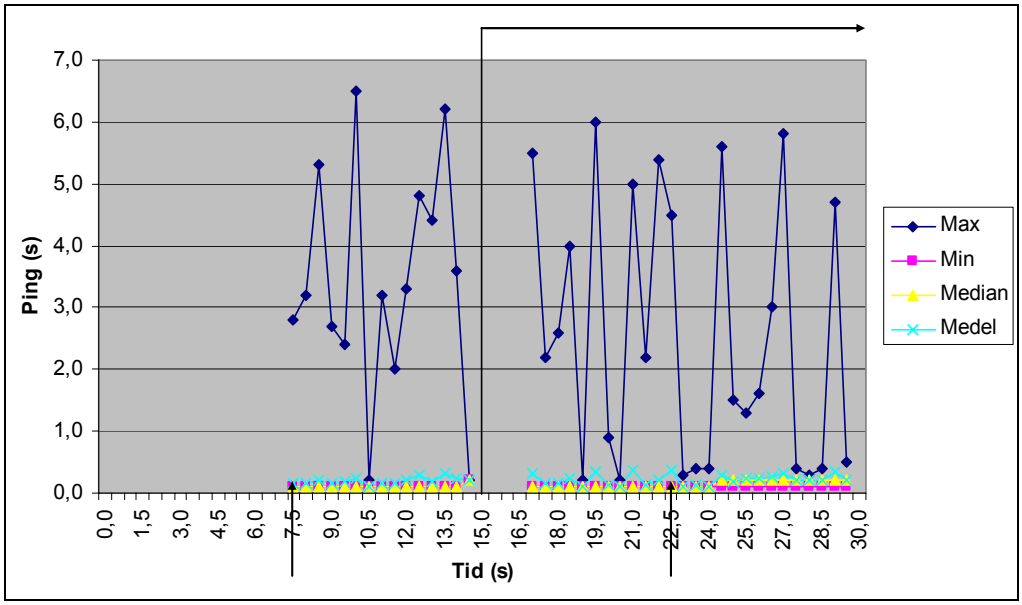
Figur A 35 - Simulering med 1000 angripare och 200 användare utan skyddsmetod (DropTail) och visar användare som kopplar upp under angreppet.



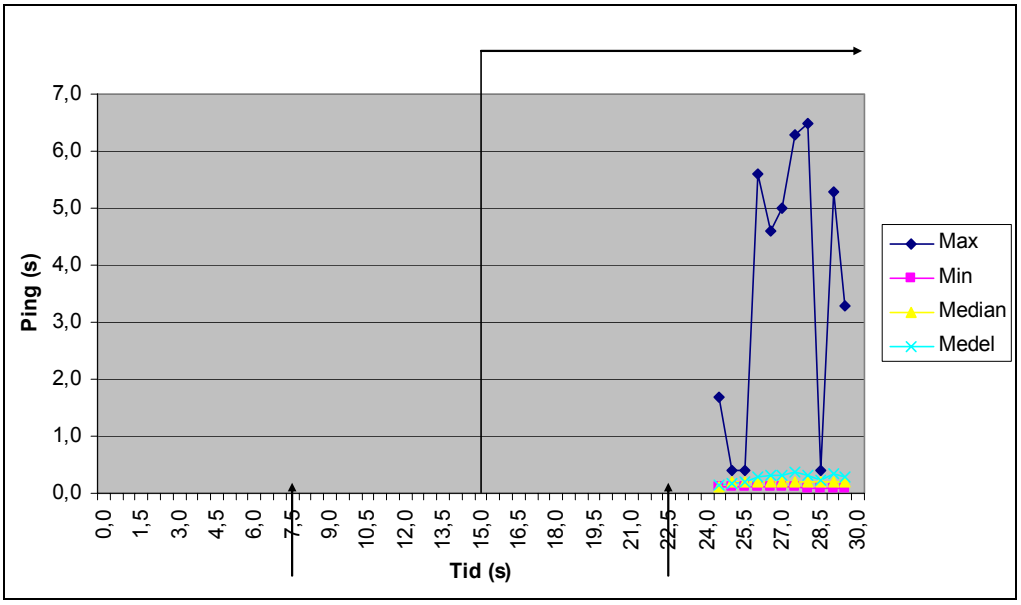
Figur A 36 - Bandbredd används av 1000 angripare och 200 användare utan skyddsmetod (DropTail).



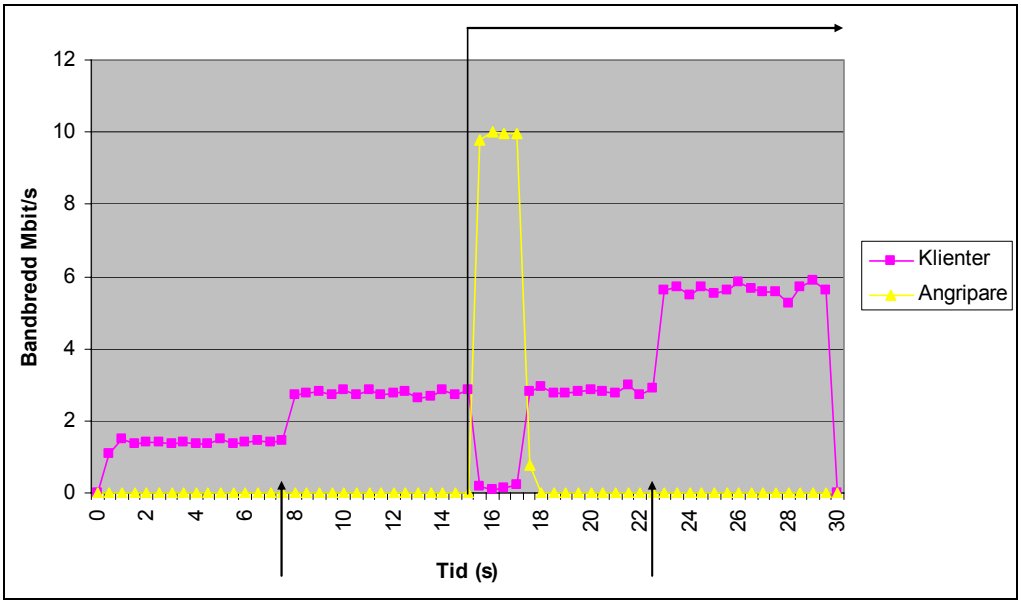
Figur A 37 - Simulering med 1000 angripare och 200 användare med SeFFiM som skyddsmetod och visar användare som är uppkopplade innan angreppet.



Figur A 38 - Simulering med 1000 angripare och 200 användare med SeFFiM som skyddsmetod och visar användare som kopplar upp innan angreppet.



Figur A 39 - Simulering med 1000 angripare och 200 användare med SeFFiM som skyddsmetod och visar användare som kopplar upp under angreppet.



Figur A 40 - Bandbredd används av 1000 angripare och 200 användare med SeFFiM som skyddsmetod.

Appendix B – Källkod till simulering

```
Queue/SeFFiM set minTimeLimit_ 60
Queue/SeFFiM set minPktLimit_ 40
Queue/SeFFiM set maxTmpPkt_ 15

Queue/ProbQueue set probability_ 50
Queue/ProbQueue set table_size_ 75

set agent SeFFiM
set trace_ok 0
set pi(0) [open ping_a100_c20_var_prq_p50_ts75.tr w]

set nrOfAttackers 100
set attackInterval 0.003
set attackDelay 5ms
set attackBandwidth 512k
set attackPktSize 500

set nrOfClients 20
set firstClients 5
set secondClients 5
set thirdClients 10

set clientInterval 0.3
set clientDelay 10ms
set clientBandwidth 256k
set clientPktSize 500

set serverDelay 1ms
set serverBandwidth 10000k
set overallSampleSize 3

set totalNodes [expr $nrOfAttackers + $nrOfClients]

puts "set totalNodes [expr $nrOfAttackers + $nrOfClients]"

set ns [new Simulator]

# Open a trace file
if {$trace_ok == 1} {
      set nf [open out.nam w]
      $ns namtrace-all $nf
}

set f(0) [open out0.tr w]
set f(1) [open out1.tr w]
set f(2) [open out2.tr w]
set f(3) [open out3.tr w]
set f(4) [open out4.tr w]
set f(5) [open out5.tr w]
set f(6) [open out6.tr w]
set f(7) [open out7.tr w]
set f(8) [open out8.tr w]
set f(9) [open out9.tr w]

# set two flow to identify the attackers flow and the legacy flow
$ns color 1 red
$ns color 2 blue
```

```
# server node
set server [$ns node]
set center [$ns node]

for {set i 0} {$i < $totalNodes} {incr i} {
      set n($i) [$ns node]
}

# connect the server to the center
$ns duplex-link $server $center $serverBandwidth $serverDelay $agent

# setup the connections - clients
for {set i 0} {$i < $nrOfClients} {incr i} {
      $ns duplex-link $n($i) $center $clientBandwidth $clientDelay
$agent
}

# setup the connections - attackers
for {set i $nrOfClients} {$i < $totalNodes} {incr i} {
      $ns duplex-link $n($i) $center $attackBandwidth $attackDelay
$agent
}

proc attach-expoo-traffic { node sink size burst idle rate interval }
{
      # Get an instance of the simulator
      set ns [Simulator instance]

      #Create an UDP agent and attach it to the node
      set source [new Agent/UDP]
      $ns attach-agent $node $source

      #Create an Expoo traffic agent and set its configuration
parameters
      set traffic [new Application/Traffic/CBR]
      $traffic set packet-size $size
      $traffic set interval_ $interval
      $traffic set random_ 1

      # Attach traffic source to the traffic generator
      $traffic attach-agent $source
      # Connect the source and the sink
      $ns connect $source $sink
      return $traffic
}

for {set i 0} {$i < $totalNodes} {incr i} {
      set sink($i) [new Agent/LossMonitor]
      $ns attach-agent $server $sink($i)
}

for {set i 0} {$i < $nrOfClients} {incr i} {
      set source($i) [attach-expoo-traffic $n($i) $sink($i)
$clientPktSize 4s 2s 100k $clientInterval]
      puts "Clients: $i\n"
}

for {set i $nrOfClients} {$i < $totalNodes} {incr i} {
      set source($i) [attach-expoo-traffic $n($i) $sink($i)
$attackPktSize 2s 1s 100k $attackInterval]
      puts "Attackers: $i\n"
}
```

```
proc finish {} {

      global f[] ns nf nrOfClients pi[] trace_ok
      $ns flush-trace
      close $nf
      # Close the output files
      close $f(0)
      close $f(1)
      close $f(2)
      close $f(3)
      close $f(4)
      close $f(5)
      close $f(6)
      close $f(7)
      close $f(8)
      close $f(9)
      close $pi(0)

      if {$trace_ok == 1} {
            exec ./nam out.nam
      }
      exec ./xgraph out0.tr out1.tr out2.tr out3.tr out4.tr out5.tr
out6.tr out7.tr  -geometry 800x400 &
      exit 0
}


for {set i 0} {$i < $nrOfClients} {incr i} {
      set p0($i) [new Agent/Ping]
      set p1($i) [new Agent/Ping]
      $ns attach-agent $server $p0($i)
      $ns attach-agent $n($i) $p1($i)
      $ns connect $p0($i) $p1($i)
}

proc record {} {

      global nrOfClients nrOfAttackers totalNodes overallSampleSize
halfClients
      global sink[] f[] p1[] p0[] pi[]

      # Get an instance of the simulator
      set ns [Simulator instance]
      # Set the time after which the procedure should be called again
      set time 0.5

      set now [$ns now]
      # How many bytes have been received by the traffic sinks?

      set bw(0) [$sink(0) set bytes_]
      set bw(1) [$sink(1) set bytes_]
      set bw2 [$sink2 set bytes_]

      # ping stuff

      Agent/Ping instproc recv {from rtt} {
            global pi[] ns
            set now [$ns now]

            $self instvar node_
            puts $pi(0) "$now [$node_ id] $from $rtt"
```

```
        }

        for {set j 0} {$j < $halfClients} {incr j} {
              $ns at [expr $now + [expr $j * 0.01]] "$p1($j) send"
        }

        if {$now > 22.0} {
              for {set j $halfClients} {$j < $nrOfClients} {incr j} {
                    $ns at [expr $now + [expr $j * 0.02]] "$p1($j)
send"
              }
        }

        for {set i 0} {$i < $totalNodes} {incr i} {
                    set bw($i) [$sink($i) set bytes_]
        }

        set tmpC 0
        set tmpA 0

        if {$nrOfAttackers > 0} {

        set tmpC1 0
        set tmpC2 0
        set tmpC3 0
        set tmpA1 0
        set tmpA2 0
        set tmpA3 0
        set CsampleSize $overallSampleSize
        set AsampleSize [expr $nrOfClients + $CsampleSize]
        set AsampleSize2 [expr $AsampleSize + $CsampleSize]
        set AsampleSize3 [expr $AsampleSize2 + $CsampleSize]
        set CsampleSize2 [expr $CsampleSize + $CsampleSize]
        set CsampleSize3 [expr $CsampleSize2 + $CsampleSize]

        for {set i 0} {$i < $CsampleSize} {incr i} {
              set tmpC1 [expr $tmpC1 + $bw($i)]
        }

        for {set i $CsampleSize} {$i < $CsampleSize2} {incr i} {
              set tmpC2 [expr $tmpC2 + $bw($i)]
        }

        for {set i $CsampleSize2} {$i < $CsampleSize3} {incr i} {
              set tmpC3 [expr $tmpC3 + $bw($i)]
        }

        for {set i $nrOfClients} {$i < $AsampleSize} {incr i} {
              set tmpA1 [expr $tmpA1 + $bw($i)]
        }

        for {set i $AsampleSize} {$i < $AsampleSize2} {incr i} {
              set tmpA2 [expr $tmpA2 + $bw($i)]
        }

        for {set i $AsampleSize2} {$i < $AsampleSize3} {incr i} {
              set tmpA3 [expr $tmpA3 + $bw($i)]
        }

        for {set i 0} {$i < $nrOfClients} {incr i} {
              set tmpC [expr $tmpC + $bw($i)]
        }
```

```
        for {set i $nrOfClients} {$i < $totalNodes} {incr i} {
              set tmpA [expr $tmpA + $bw($i)]
        }

}
        #Calculate the bandwidth (in MBit/s) and write it down to the
files

        puts $f(0) "$now [expr ($tmpC/$time*8/1000000)/$nrOfClients]"

        puts $f(1) "$now [expr ($tmpA/$time*8/1000000)/$nrOfAttackers]"

        puts $f(2) "$now [expr ($tmpC1/$time*8/1000000)]"
        puts $f(3) "$now [expr ($tmpC2/$time*8/1000000)]"
        puts $f(4) "$now [expr ($tmpC3/$time*8/1000000)]"
        puts $f(5) "$now [expr ($tmpA1/$time*8/1000000)]"
        puts $f(6) "$now [expr ($tmpA2/$time*8/1000000)]"
        puts $f(7) "$now [expr ($tmpA3/$time*8/1000000)]"
        puts $f(8) "$now [expr ($tmpC/$time*8/1000000)]"
        puts $f(9) "$now [expr ($tmpA/$time*8/1000000)]"



        # 3 samples from each

#       puts $f(0) "$now [expr $bw(0)/$time*8/1000000]"
#       puts $f(1) "$now [expr $bw(1)/$time*8/1000000]"
#       puts $f(2) "$now [expr $bw(2)/$time*8/1000000]"
#       set tmpOne [expr $totalNodes - 1]
#       set tmpTwo [expr $totalNodes - 2]
#       set tmpThree [expr $totalNodes - 3]
#       puts $f(3) "$now [expr $bw($tmpOne)/$time*8/1000000]"
#       puts $f(4) "$now [expr $bw($tmpTwo)/$time*8/1000000]"
#       puts $f(5) "$now [expr $bw($tmpThree)/$time*8/1000000]"

        # Reset the bytes_ values on the traffic sinks

        for {set i 0} {$i < $totalNodes} {incr i} {
              $sink($i) set bytes_ 0
        }

        # Re-schedule the procedure
        $ns at [expr $now+$time] "record"
}

$ns at 0.0 "record"

# start the first clients at beginning of simulation
for {set i 0} {$i < $firstClients} {incr i} {
        $ns at 0.1 "$source($i) start"
}

# start the second clients at 7.5 secs into simulation
for {set i $firstClients} {$i < $secondClients} {incr i} {
        $ns at 7.5 "$source($i) start"
}

# start the clients who tries to connect during the attack
for {set i $halfClients} {$i < $nrOfClients} {incr i} {
        $ns at 22.5 "$source($i) start"
}
```

Appendix B

```
# start the attackers at 1/2 into simulation
for {set i $nrOfClients} {$i < ($totalNodes)} {incr i} {
      $ns at 15.0 "$source($i) start"
}

# stop at the end of simulation
for {set i 0} {$i < $totalNodes} {incr i} {
      $ns at 29.9 "$source($i) stop"
}

$ns at 30.0 "finish"
$ns run
```

Appendix C – Källkod till SeFFiM

SeFFiM.h

```
#ifndef ns_SeFFiM_h
#define ns_SeFFiM_h

#include <vector>
#include "queue.h"
using namespace std;

typedef int PktC;
typedef int IPnr;
typedef double Stamp;

enum listNames { BlackList, SuspectedList, UserList };

class myNode {
	public:
		myNode();
		myNode(IPnr IP, PktC nrPkt, PktC suspPkt, Stamp
timeStamp);
		~myNode();
		void printmyNode();
		void setNrPkt(PktC p);
		void setSuspPkt(PktC p);
		void setTimeStamp(Stamp t);
		void setFirstStamp(Stamp s);
		void setNrTmpPkt(int nr);
		IPnr getIP();
		PktC getNrPkt();
		PktC getSuspPkt();
		Stamp getTimestamp();
		Stamp getFirstStamp();
		int getNrTmpPkt();

		friend class MyList;

	private:
		IPnr IP;
		PktC nrPkt;
		PktC suspPkt;
		Stamp timeStamp;
		Stamp firstStamp;
		int nrTmpPkt;
};

class MyList {
	public:
		MyList();
		~MyList();
		void addmyNode(IPnr IP, PktC nrPkt, PktC suspPkt, Stamp
timeStamp);
		void removemyNode(int pos, IPnr IP);
		int searchmyNode(IPnr IP);
		int getSuspPkt(int pos, IPnr IP);
```

```
            int getNrPkt(int pos, IPnr IP);
            int getNrTmpPkt(int pos, IPnr IP);
            Stamp getFirstStamp(int pos, IPnr IP);
            void updatePacketFlow(int pos, IPnr IP, int nr);
            bool checkPacketFlow(int pos, int maxTmpPkt, Stamp
timeStamp);
            void updatemyNode(int pos, IPnr IP, PktC nrPkt, Stamp
timeStamp, Stamp minTimeLimit, double minTimeMulti, PktC maxTotalPkt,
PktC maxTmpPkt);
            void resetList();
            bool isEmpty();
            int getListSize();

            // debugging
            void printList();

      private:
            vector<myNode*> v;
};

class SeFFiM : public Queue {
      public:
            SeFFiM() {
                  q_ = new PacketQueue;
                  pq_ = q_;
                  bind_bool("queue_in_bytes_", &qib_);
                  bind_bool("summarystats_", &summarystats);
                  bind("mean_pktsize_",&mean_pktsize_);
                  bind("minTimeLimit_", &minTimeLimit);
                  bind("minPktLimit_", &minPktLimit);
                  bind("maxPktLimit_", &maxPktLimit);
                  bind("minTimeMulti_", &minTimeMulti);
                  bind("maxTotalPkt_", &maxTotalPkt);
                  bind("maxTmpPkt_", &maxTmpPkt);
            }

            ~SeFFiM() {
                  delete q_;
            }

      protected:
            void enque(Packet*);          /* add node */
            Packet* deque();              /* remove node */
            void cleanQueue();            /* clean entire queue */
            int command(int argc, const char* const* argv);
            void reset();                 /* reset FIFO queue */
            void print_summarystats();    /* Print stuff */

            int summarystats;
            PacketQueue *q_;
            int qib_;
            int mean_pktsize_;
            MyList blackList;
            MyList suspectList;
            MyList userList;
            Stamp minTimeLimit;
            int minPktLimit;
            int maxPktLimit;
            double minTimeMulti;
            PktC maxTotalPkt;
            PktC maxTmpPkt;
};
```

```
#endif
```

SeFFiM.cc

```
#include <vector>
#include <string>
#include "myqueue.h"

using namespace std;

myNode::myNode() {

};

myNode::myNode(IPnr IP, PktC nrPkt, PktC suspPkt, Stamp timeStamp) {
	this->IP = IP;
	this->nrPkt = nrPkt;
	this->suspPkt = suspPkt;
	this->timeStamp = timeStamp;
	nrTmpPkt = 0;
	firstStamp = 0.0;
}

myNode::~myNode() {

};

// debuggning
void myNode::printmyNode() {
	printf("\nmyNode IP, %d - NrPkt: %d - SuspPkt: %d - Timestamp:
%d\n\n ", IP, nrPkt, suspPkt, timeStamp);
};

void myNode::setNrPkt(PktC p) {
	nrPkt += p;
};

void myNode::setSuspPkt(PktC p) {
	suspPkt += p;
};

void myNode::setTimeStamp(Stamp t) {
	timeStamp = t;
};

void myNode::setFirstStamp(Stamp s) {
	if (s > 0.0) {
		firstStamp = s;
	} else {
		firstStamp = 0.0;
	}
};

void myNode::setNrTmpPkt(int nr) {
	if (nr > 0) {
		nrTmpPkt += nr;
	} else {
		nrTmpPkt = 0;
	}
};
```

```
IPnr myNode::getIP() {
      return IP;
}

PktC myNode::getNrPkt() {
      return nrPkt;
};

PktC myNode::getSuspPkt() {
      return suspPkt;
};

Stamp myNode::getTimestamp() {
      return timeStamp;
};

Stamp myNode::getFirstStamp() {
      return firstStamp;
};

int myNode::getNrTmpPkt() {
      return nrTmpPkt;
};

MyList::MyList() {
      v.reserve(100);
};

MyList::~MyList() {
      resetList();
};

void MyList::addmyNode(IPnr IP, PktC nrPkt = 0, PktC suspPkt = 0,
Stamp timeStamp = 0) {

      if (searchmyNode(IP) == -1) {
            myNode *n = new myNode(IP, nrPkt, suspPkt, timeStamp);
            v.push_back(n);
      }
};

int MyList::searchmyNode(IPnr IP) {

      for (int i = 0; i < v.size(); i++) {
            if (v[i]->getIP() == IP) {
                  return i;
            }
      }
      return -1;
};


void MyList::removemyNode(int pos, IPnr IP) {
      if (v[pos]->getIP() == IP) {
            if (pos >= 0) {
                  delete v[pos];
                  vector<myNode*>::iterator where = v.begin() + pos;
                  v.erase(where);
            }
      }
};
```

```
void MyList::printList() {
	for (int i = 0; i < v.size(); i++) {
		printf("myNode nr: %d", i);
		v[i]->printmyNode();
	}
};

int MyList::getSuspPkt(int pos, IPnr IP) {
	if(v[pos]->getIP() == IP) {
		return v[pos]->getSuspPkt();
	} else {
		return -1;
	}
};

int MyList::getNrPkt(int pos, IPnr IP) {
	if(v[pos]->getIP() == IP) {
		return v[pos]->getNrPkt();
	} else {
		return -1;
	}
};

int MyList::getNrTmpPkt(int pos, IPnr IP) {
	if(v[pos]->getIP() == IP) {
		return v[pos]->getNrTmpPkt();
	} else {
		return -1;
	}
};

Stamp MyList::getFirstStamp(int pos, IPnr IP) {
	if(v[pos]->getIP() == IP) {
		return v[pos]->getFirstStamp();
	} else {
		return -1;
	}
};

void MyList::updatePacketFlow(int pos, IPnr IP, int nr) {
	if(v[pos]->getIP() == IP) {
		v[pos]->setNrTmpPkt(nr);
	}
};

bool MyList::checkPacketFlow(int pos, int maxTmpPkt, Stamp timeStamp)
{
	if (v[pos]->getNrTmpPkt() >= maxTmpPkt) {
		v[pos]->setNrTmpPkt(0);
		v[pos]->setFirstStamp(timeStamp);
		return 0;
	} else {
		return -1;
	}
};

void MyList::updatemyNode(int pos, IPnr IP, PktC nrPkt, Stamp
timeStamp, Stamp minTimeLimit, double minTimeMulti, PktC maxTotalPkt,
PktC maxTmpPkt) {
```

```
	if (pos >= 0) {
		Stamp tmpStamp = v[pos]->getTimestamp();
		updatePacketFlow(pos, IP, 1);
		if (((timeStamp - v[pos]->getFirstStamp()) >=
minTimeLimit) ||
		   (v[pos]->getNrPkt() >= maxTotalPkt)) {
			if (checkPacketFlow(pos, maxTmpPkt, timeStamp) ==
0) {
				v[pos]->setSuspPkt(1);
			}
		}

		v[pos]->setNrPkt(1);
		v[pos]->setTimeStamp(timeStamp);
	}
};

void MyList::resetList() {
	for (int i = 0; i < v.size(); i++) {
		delete v[i];
	}
	v.clear();
};

bool MyList::isEmpty() {
	return v.empty();
};

int MyList::getListSize() {
	return v.size();
};

static class SeFFiMClass : public TclClass {
	public:
		SeFFiMClass() : TclClass("Queue/SeFFiM") {}
		TclObject* create(int, const char*const*) {
			return (new SeFFiM);
		};
} class_my_queue;

/* Determine which packet that should be added to the queue. Check
with the internal lists and determine
 if the packet should be dropped or put on the queue. */
void SeFFiM::enque(Packet* p) {

	hdr_ip *hdr = hdr_ip::access(p);	// access the header of the
current packet
	IPnr src_ip = hdr->saddr();		// get the ip addr (the
simulator equivalent)
	int u_id = hdr_cmn::access(p)->uid_;	// get unique packet
id
	Stamp ts = hdr_cmn::access(p)->ts_;	// get timestamp of
the current packet
	bool addPacket = false;		// add packet or not to
queue.
	int pos = -1;			// listposition of ip.
	int tmpSuspPkt = -1;
	int tmpPkt = -1;

	if (summarystats) {
		Queue::updateStats(qib_?q_->byteLength():q_->length());
```

```
	}

	if (blackList.searchmyNode(src_ip) >= 0) {
		drop(p);
	} else {
		pos = suspectList.searchmyNode(src_ip);
		if (pos >= 0) {
			suspectList.updatemyNode(pos, src_ip, 1, ts,
minTimeLimit,minTimeMulti, maxTotalPkt,maxTmpPkt);
			if (suspectList.getSuspPkt(pos, src_ip) >=
minPktLimit) {
				blackList.addmyNode(src_ip);
				suspectList.removemyNode(pos, src_ip);
				drop(p);
			} else {
				addPacket = true;
			}
		} else {
			pos = userList.searchmyNode(src_ip);
				if (pos >= 0) {
				userList.updatemyNode(pos, src_ip, 1, ts,
minTimeLimit, minTimeMulti, maxTotalPkt, maxTmpPkt);
				tmpSuspPkt = userList.getSuspPkt(pos,
src_ip);
				tmpPkt = userList.getNrPkt(pos, src_ip);
				if (tmpSuspPkt >= maxPktLimit) {
					suspectList.addmyNode(src_ip, tmpPkt,
tmpSuspPkt, ts);
					userList.removemyNode(pos, src_ip);
					addPacket = true;
				} else {
					userList.addmyNode(src_ip);
					addPacket = true;
				}
			} else {
				userList.addmyNode(src_ip, tmpPkt,tmpSuspPkt,
ts);
				addPacket = true;
			}
		}
	}

	// Deal with overflows of the queue
	if (addPacket == true) {
		int qlimBytes = qlim_ * mean_pktsize_;
		if ((!qib_ && (q_->length() + 1) >= qlim_) ||
		(qib_ && (q_->length() + hdr_cmn::access(p)->size()) >=
qlimBytes)) {
			// What to do when queue is full?
			// LIFO queue then?
			drop(p);
		} else {
			q_->enque(p);
		}
	} else {
		drop(p);
	}

};

Packet* SeFFiM::deque() {
	if (summarystats && &Scheduler::instance() != NULL) {
```

```
            Queue::updateStats(qib_?q_->byteLength():q_->length());
      }
      return q_->deque();
};

void SeFFiM::cleanQueue() {

};

int SeFFiM::command(int argc, const char* const* argv) {

      if (argc == 2) {
            if (strcmp(argv[1], "printstats") == 0) {
                  print_summarystats();
                  return (TCL_OK);
            }
      }
      return Queue::command(argc, argv);
};

void SeFFiM::reset() {
      Queue::reset();
};

void SeFFiM::print_summarystats() {
      printf("True average queue: %5.3f", true_ave_);
      if (qib_) {
            printf(" (in bytes)");
      };
      printf(" time: %5.3f\n", total_time_);
};
```