KTH Computer Science
and Communication

Proof-of-Work

En modern betraktelse

Namn: Aron Sharma & Damon Shahrivar

Examenrapport vid NADA
Handledare: Mikael Goldmann
Examinator: Mads Dam

Referat

Proof-Of-Work protokollet är en konceptuell idé, tänkt att förebygga DoS, brute force samt spam-mail angrepp. Huvudidèn är att servern som får en förfrågan av en klient, inte accepterar eller bearbetar denna direkt, utan skickar tillbaka ett litet pussel till klienten som skickat förfrågan. Pusslet måste lösas av klienten innan denne får tillgång till servern och själva poängen med protokollet är att det skall ta mer cpu tid för klienten att lösa pusslet än vad det tar för servern att hitta en lösning på klientens förfrågan. Idén är att detta skall hjälpa mot ovan nämnda angrepp. Projektet kommer att studera Proof-Of-Work protokollet och flera av dess implementationer. Projektet inefattar även en undersökning av prestandaskillnaden mellan olika implementationer på olika plattformar och en reflektion som behandlar protokollets användbarhet.

Abstract

Proof-of-Work in a modern day observation

Proof-Of-Work Protocol is a conceptual idea, meant to prevent DoS attacks, brute force and spam-mail attacks. The main idea is that when a server receives a request from a client, it does not accept or process it directly. Instead it sends back a small puzzle to the client that sent the request. The puzzle must be solved by the client before being given access to the server. The point of the protocol is that it will cost the client more cpu time to solve the puzzle than what it takes for the server to find a solution to the client's request. The idea is that this will help against the attacks mentioned above. This project will study the Proof-Of-Work Protocol and a few of its implementations. In addition to this, the project also includes a study of the difference in performance between different implementations on different platforms, and a reflection of the Protocol that deals with usability and the protocol's place on the modern day web.

Innehåll

Kapitel 1

Introduktion

Inledning

I dagens moderna samhälle där datalogi har blivit en del av vardagen växer behovet av datasäkerhet för var dag som går. För ett företag är en hemsida idag nästan lika självklart som en logotyp, och en grundläggande kunskap inom datalogi är därför ett måste inom många branscher. Behovet av detta har ökat snabbt under de senaste årtiondena, och det är en av de största förändringar som vårt samhälle genomgått på senare tid.

Historien visar att vid stora förändringar följer också komplikationer. Säkerheten blir allt viktigare då produktionssamhället är beroende av effektivitet. En server som är nere en timme betyder att vi under en timme inte har något produktion över huvud taget vilket i sin tur resulterar i förluster.

Datorer möter många hot i samhället, ett av dessa är Denial of service attacker, också kallade DoS attacker. Dessa har visat sig vara en populär taktik för hackare överallt i världen, och attackerna har drabbat allt från privatpersoner till Europeiska regeringar[11]. De har vuxit till att bli ett allvarligt hot mot vår samhällsstruktur, och därför har det också börjat utvecklas metoder för att motverka denna typ av attacker. Man har använt sig av brandväggar samt Access "Control List" begränsningarna i switchar och routrar med positiva resultat. Ett annat problem som vanliga användare ofta möter är spam. Detta arbete rör ett protokoll som är ämnat att hantera de ovan nömnda problemen. Nämligen Proof-Of-Work protokollet.

Samarbetets struktur

Arbetet har genomförts av Aron Sharma och Damon Shahrivar.

Aron Sharma Research, androidimplementation, testning och uppsatsskrivning.

Damon Shahrivar Research, val av processorer, windowsimplementation, testning och uppsatsskrivning.

Syfte

Syftet med detta arbete är att granska olika implementationer av Proof-Of-Work protokollet, och därefter diskutera hur effektivt det egentligen är i dagens användarmiljö. Vi kommer här att jämföra hash-cash metoden samt en minnesbunden implementation. Vi kommer att använda oss av äldra benchmarks, men även testa att implementera protokollen på moderna processorer och mobiltelefoner. Denna rapport är en del av vårt kandidatexamens-arbete, i kursen DD143X.

Tes

Problemet med spam och DoS attacker har sedan länge setts som ett resultat av att det inte riktigt finns något på nätet som försöker förhindra att en person gör detta. Idén med Proof-Of-Work protokollet är att om det "kostade" angriparen att utföra en attack så skulle det inte längre vara lönsamt att spamma, och man skulle på detta sätt kunna minska antalet spam-mail. Att dömma av intrycket vi får av dagens internetklimat gällande att använda riktiga pengar, så verkar det inte få stöd hos den breda massan. Idén som framfördes av Dwork och Naor [4] som istället pekats ut som alternativet till riktiga pengar går ut på att låta sändaren av ett email utföra en avancerad beräkning och skicka till mottagardatorn som bevis på att emailet som skickats är värt att ta emot. Processorkraften som går åt för att sända ett email med denna metod är fortfarande gratis, men den är begränsad vilket förhindrar att tjänsten missbrukas.

Tekniken för att implementera ett Proof-Of-Work protokoll har funnits sedan länge. Dagens debatt handlar istället om ifall tekniken är effektiv eller inte.[9][10] Det är lätt att i teorin få metoden att fungera, men i praktiken kan många problem uppenbara sig. Protokollet måste uppfylla sitt syfte utan att begränsa möjligheterna för genuina användare. Det finns två huvudvarianter av protokollet, Challenge-response samt Solution verification. Med solution verification löser man ett fördefinerat problem och har en stämpel i sitt email med svaret som sedan verifieras av servern. Allt detta sker innan man har en uppkoppling till servern. Stämpel fungerar som en header som bevisar att klienten har spenderat viss CPU tid för att lösa problemet och därför med största sannolikhet inte är en spammare. I dagens läge finns det ett stort antal implementationer av Proof-Of-Work, varav en av de mest kända är Adam Back's Hashcash metod som är av typen solution verification. Challenge-response metoden är inte lika utbredd som solution verification och den går ut på att en klient skickar en request till servern och får tillbaks ett problem som måste lösas. Klienten får sedan åtkomst när problemet är löst. Svagheten med detta är att en massa resurser går åt för själva processen med att endast få tillgång

till problemet[2]. Själva kommunikationen blir problematiskt och som en följt av detta blir protokollet mindre effektivt.

En annan komplikation som uppstått idag, och som Proof-Of-Work skaparna inte räknat med är att våra nätverk idag består av mer än vanliga datorer. Nu för tiden är vi även uppkopplade med telefoner, spelkonsoler och läsplattor. Detta blir problematisk då ett problem som kan lösas på 10 sekunder med en dator kan ta flera minuter med en smartphone. För att implementera en smart lösning måste faktorer som denna tas med i utveckligen.

Målet med Proof-Of-Work är att motverka DoS attacker och spam, och om man väljer ett pussel som är för svårt kan det göra så att legitima användare inte kan skicka email eller utföra andra typer av requests. För att implementera protokollet måste ett pussel väljas, som är ungefär lika krävande på de flesta system och som inte är allt för svårt. Då de flesta implementationerna är menade att fylla samma syfte så kommer problemen som behandlas vara snarlika.

Andra användningsområden

Förutom att motverka spam så kan Proof-Of-Work användas i flera andra områden, till exempel för att motverka andra typer av denial-of-service angrepp, motivera samarbete i peer-to-peer system och mäta antalet besök en hemsida har fått. Rent allmänt, så kan Proof-Of-Work användas som accesskontroll där det är viktigt att skydda resurser.

CPU problematik

Huvudidén med att använda Proof-Of-Work för att bekämpa spam är att öka tillförlitligheten för genuin e-post genom att förse meddelandena med bevis i form av att de använder sig av sändarens resurser. Fram tills nyligen var processortiden den enda resursen som ansågs vara användbar för denna sorts bevis, men dessa saktar dock ner email processen för den generella användaren. Det kan beräknas snabbare om en spammare bestämmer sig för att använda en bättre processor. En spammare kan med en engångsinvestering enkelt göra sig kvitt de flesta Proof-Of-Work problemen då man kan räkna med att protokollens framtida ökning av prestandakrav enbart är marginell.

Tillvägagång

Vi kommer att testa ett par olika implementationer av Proof-Of-Work protokollet. Vi kommer att implementera en version av hashcash samt en annan implementation kallad mbound. Dessa kommer att testas på olika klienter. Då det redan finns en hel del data och resultat av tester gjorda på datorer, så har vi valt att främst fokusera våra tester på dagens nya teknik, nämligen mobiltelefoner. Vi kommer därför under

testningsfasen att jämföra implementationerna som redan gjorts på olika datorer med testresultat som vi får av våra egna tester på android-telefoner.

Kapitel 2

Bakgrund

2.1 De olika metoderna

Som tidigare nämnt finns det ett flertal olika implementationer. Somliga är enbart teortiska, medan andra har implementerats. Det finns även säkerhetssystem överallt på nätet som bygger på samma princip som Proof-Of-Work protokollet. Man skulle till exempel kunna likna säkerhetsåtgärderna captcha [1] systemen som används vid ifyllning av formulär på nätet där man bes om att fylla i ett antal siffror och bokstäver visade i en förvrängd bild. Detta kräver arbete som bevis för att man inte är en maskin som försöker skapa ett konto.

2.1.1 SHA-1

SHA funktionerna är kryptografiska hash-funktioner designade av NSA (National Security Agency). SHA står för secure hash algorithm, och den används i hashcash metoden samt många andra Proof-Of-Work implementationer. SHA-1 algoritmen producerar output om 160 bitar. SHA-1 används i de flesta Proof-Of-Work implementationerna som rapporten kommer att gå igeonom.

2.1.2 HashCash

En viktig egenskap hos Adam Back's hashcash protokoll och dess pussel är att det är väldigt dyrt för en sändare att lösa pusslet medans de samtidigt är simplelt och billigt för mottagaren att verifiera. För att kunna binda hashcash pusslet till ett email så måste man ange mottagarens email adress, en tidsstämpel. Dessa används med SHA-1 för att skapa en hash som sedan kommer att undersökas för att hitta de mest signifikanta bitarna. Dessa skall i hashcash vara noll. För att visa hur Proof-Of-Work protokoll är uppbyggda, låt oss ta en närmare titt på hashcash metoden. För att kunna sända ett meddelande till en mottagare så måste sändaren hitta stämpeln k där de L mest signifikanta bitarna av y = SHA-1(time||message||recipient||k) är nollor. Det enklaste sättet att göra detta på är att välja ett slumpvist hash och kolla om de flesta signifikanta bitarna faktiskt är nollor. Detta är en brute-force lösning

och är väldigt tung på en dators CPU. Efter lyckad beräkning så är k bifogad i ett email som Proof-Of-Work mottagaren får användning av bevis k och beräknar hashfunktionen en gång för att se om de mest signifikanta bitarna är nollor. Om beviset är korrekt släpps emailet igenom spamfiltret.

computing a Hashcash Proof-Of-Work
L is a parameter indicating the hardness of this Proof-Of-Work

```
function ComputeProofOfWork(message, recipient, time) : k
loop
  trial := Random() {Man kan sätta trial som ett slumpvist tall innan loopen
  börjar och sedan bara öka trial med 1 i varje iteration}
  hashed := SHA-1(time||message||recipient||trial)
  zeros := CountMostSignificantZeros(hashed)
  if (zeros >= L) then
    return trial
  end if
end loop
```

2.2 Olika angreppsmodellerna

2.2.1 Denial of service(DoS)

Basic

Denial of service, även känt som DoS är en sorts attack som förstör möjligheterna för användning av nätverk, system eller applikationer genom att överbelasta resurser som till exempel CPU, minne, eller bandbredd. Det finns olika sorters DoS attacker, men de bygger alla på samma grundconcept.

Distributed Denial of Service Attacks (DDoS)

Distributed Denial of Service Attack, förkortat DDoS fungerar på samma sätt som vanliga DoS attacker. Skillnaden är att man använder fler clienter att attackera från under DDoS attacken. Denna variant av DoS attacker har ställt till det ordentligt för Proof-Of-Work protokollet då även en stark dator kommer få problem med tusen datorer som attackerar den.

2.2.2 SPAM

Laurie och Clayton kommer fram till i [9] att ett spam-post som högst kan kosta sändaren 0,005 amerikanska cents för att spam-verksamheten ska vara vinstdrivande. Denna siffra har räknats ut med priset för en medelstark dator samt elektricitet och andra regelbundna kostnader i åtanke. Enligt Laurie och Claytons beräkningar

måste en spammare därför skicka 15000 epost om dagen för att verksamheten skall gå runt. Detta betyder att ett epost kan ta som högst 5,8 sekunder för att skickas för att verksamhten skall vara vinstdrivande. Undersökningen visar dock att det finns många spammare som skulle kunna kringgå problemen som en låg tidsspärr skulle medföra. Med detta i åtanke har Laurie och Clayton[9] kommit fram till att för att effektivt bekämpa spammare bör deras epostkvot begränsas till 1750 stycken om dagen, dvs varje epost ska ta runt 50 sekunder att skicka.

2.3 Problemet

Utöver det klassiska problemet om hur svårt ett problem ska vara så finns det andra svårigheter med Proof-Of-Work. Ett av dessa är hur det påverkar den generella användarvänligheten. Att lösa kryptografiska problem som kräver väldigt mycket resurser gör att man ej kan använda andra resurser under tiden och utöver detta så slösar man oftast bort en massa cpu-cykler då dessa inte används till att lösa något produktivt. Problemen man löser är en form av betalning, men lösningen gör egentligen ingen ordentlig nytta. I värsta fall så kan Proof-Of-Work kosta tid (Observera att det inte handlar om CPU-tid, utan faktiskt tid).

Om man tar exemplet att en hemsida, säg Gmail använder sig av Proof-Of-Work protokollet och man måste lösa ett pussel som tar 30 sekunder innan man kan skicka iväg ett mail. Då pusslet är väldigt resurskrävande är det inte möjligt att jobba med något annat simultant vilket blir ett problem i de fall då man måste skicka hundratals email per dag. Då vi har två olika sorters Proof-Of-Work, challenge response och solution verification, så varierar även en del av problemen vi stöter på beroende vilken sorts implementation vi använder oss av. Vid solution verification så har vi ett stort problem då vi i detta fall redan har ett antal fördefinierade problem som skall lösas av användaren. En användare kan här i teorin lösa problem och spara de olika stämplarna. När detta skett tillräckligt många gånger så kommer hackaren kunna uppvisa bevis på lösningar av ett visst problem, och därefter utföra någon form av DoS attack.

Vid challenge response har vi istället en annan problematik som uppenbarar sig. Vad som här skulle kunna orsaka stora problem för vanliga användare är att om till exempel varje hemsida på nätet enbart tillåter att man ansluter till den en eller två gånger per minut för att undvika överbelastning och på grund av detta implementerar ett challenge-response pussel som tar 30 sekunder att lösa. Scenariot som uppstår då är att man enbart kan ansluta till en sida åt gången vilket blir väldigt frustrerande i längden då man ofta vill besöka flera hemsidor på en gång. Problemet i sig är inte unikt för challenge response, men i solution verification fallet kan man kringgå detta genom att lösa problem i förväg (exempelvis då datorn är satt i viloläge).

2.4 Lösningen

2.4.1 Moderately hard memory-bound functions

Lösningen till det klassiska svårighetsproblemet är minnestunga pussel. Problem som förlitar sig på att göra många minnesaccesser istället för att räkna ut stora uttryck. Detta eftersom en ny dators minne och en gammal som mest har 4:1 skillnad i prestanda[3]. Med detta så kan ett problem implementeras som är lika svårt på de flesta maskiner. Det viktiga med funktioner som förlitar sig på detta är att det är dålig lokalitet på de olika värdena som lagras, för att undvika att datorn cachar värden dvs man vill ha cache-missar. Dessa problem kan självklart förenklas med speciell hårdvara som stora cache-minnen. Dessa är dock väldigt dyra vilket gör att det blir svårare att skicka spam, därav behöver detta inte ses som ett problem.

Säg att man räknar ut $y = f(x)$, och nu vill man få reda på x, och det finns en funktion sådan att, $f^{-1}(y) = x$ men detta är svårare att göra än att kolla upp $y = f(x)$ i en tabell som har lagrats. Ett problem skulle kunna se ut så här, hitta x då y är ett visst värde. Klienten måste titta igenom en lång array för att hitta ett värde x som genererar y med en viss funktion f. Då klienten har hittat värdet x och skickat detta så kan detta enkelt verifieras hos mottagaren(servern) genom att räkna ut $y = f(x)$.

MBound Vi har använt oss av algoritmen MBound som fungerar på följande vis: algoritmen använder en array A där den totala storleken av A är större än blocksize(för att undvika att cacha stigar) och ett par hashfunktioner H_0, H_1, H_2 och H_3. Funktionen H_0 används under initiering av en stig(en stig är en mängd av element i arrayen T), den tar input m, sändarens namn S, mottagarens namn R, och datum d tillsammans med ett försökstal k och producerar arrayen A(A är en array som innehåller olika mängder av element för att matcha stigar i T). H_1 tar A som input och producerar index C i tabellen T. Funktionen H_2 tar A som input samt ett element av T och skapar en ny array A. H_3 används för att stigen man har hittat och se om denna är riktig(skapar en 4w lång sträng).

A = H_0(m, R, S, d, k){Initiering, skapa en slumpmässig första stig}
for l steg **do** {l är längden av A}
  C $\leftarrow$ H_1(A) {hitta motsvarande index i T}
  A $\leftarrow$ H_2(A, T[C]) {updatera A}
  När de e sista bitarna i H_3(A) är noll då är vi klara {dvs. efter L steg så kommer man att testa stigen genom att utföra H_3 om det inte uppfylls görs en till iteration}
end for

Observera att pseudokoden är förenklad. Se Bilaga A för närmare beskrivning och komplett pseudokod från [3].

2.4.2 Andra förslag

Marek Klonowski and Tomasz Struminski hittade en alternativ lösning till spam problemet kallat proof of communication eller POC[8]. Istället för att lösa svåra minnestunga pussel så krävs det av en användare att behandla anslutningar med slumpmässigt valda entiteter för att bevisa sitt legitima skäl till att logga in på någon form av webservice eller skicka epost. Klonowski et al bestämde i [8] att en användare skulle få tillgång till en viss information från internet och fördelarna med detta är att flaskhalsen här är just internet och inte en viss dators CPU. Klonowski et al, menar att fördelarna med detta är att det inte är beroende på skickarens CPU och kan modifieras och kombineras med existerande Proof-Of-Work lösningar. Proof of communication ska inte heller ses som en lösning i sig självt utan ska ses som ett verktyg att kombinera med existerande metoder för att motverka spam. Absolut största nackdelen med proof of communication är hur den kan utnyttjas för att utföra DoS angrepp. En spammare skickar 100-tals emails med bevis där data har samlats från en och samma entitet. När beviset ska verifieras hos mottagaren kommer alla dessa emails att öppna anslutningar med målet att överbelasta servern.

Kapitel 3

Implementationen

3.1 HashCash

Hashcash metoden utformades under slutet av 90-talet, och det är ett av de mest populära Proof-Of-Work implementationerna. Vi har i våra tester implementerat protokollet samt provat att köra det på olika svårighetsgrader på ett antal nyare processorer. Vi har förutom detta även utvecklat en applikation till Googles android operativsystem(detta med hjälp av Gregory Rubins open source projekt, `http://www.nettgryppa.com/code/`) och provat protokollet på två av dagens mest populära telefoner på mobilmarknaden, nämligen HTC Desire och ZTE Blade. För att komplettera studien har vi även använt oss av Adam Back's benchmarks som kan hittas på dennes hemsida(`http://www.hashcash.org/benchmark/`).

Figur 3.1. Fungerande android-implementation i simulatorn

3.2 MBound

En andra implementationen vi provat är en så kallad “memory bound function” utvecklats av Dwork et al [3], som de velat kalla Mbound. Den har en så kallad *fixed-forever table* vilket menas, att den har samma storlek tills datorer börjar använda sig av större cache-minnen. Tabellen T är dubbelt så stor som cache-minnen för att implementationen ska fungera väl, runt 16mb. Algoritmen funkar så att en användare måste söka olika vägar i tabellen T genom minnesacesser. Användaren tvingas göra detta tills “rätt” väg har hittats och vägen tillsammans med information som behövs för verifiering skickas till mottagaren. Verifieringen är proportionell till parametern l som bestämmer en vägs längd. Tabellerna bestäms av två .dat filer som är kodade i ett windowsspecifkt format.

Dwork et al nämde i [3], att mottagaren kan skicka emails också och denna har nu tillgång till tabellen T och dess innehåll, mottagaren ska inte behöva åtkomst till denna utan verifieringen ska vara möjlig med endast datan som fåtts av skickaren. De menade att en implementation med hashträd skulle vara bättre då antalet bitar som behöver att skickas är mindre. Dessutom är det problematiskt att behöva ladda ner en 16mb stor tabell med begränsad uppkoppling för att behöva uppvisa ens legitima syfte. Det finns två lösningar till detta, en teoretisk vidareutveckling av Dwork et al[5] där de har ändrat tabellen T och en av Coelho där en implementation med merkleträd har använts[2].

Input parametrar för mbound: meddelande m, skickare S, mottagare R, tid t, tabell T, hjälptabell(vägen) A

Kapitel 4

Testet

Vi har utfört tester på ett flertal olika processorer. Vi har gjort tio tester per processor och därefter tagit ut ett medelvärde på tiden som det har tagit. Detta medelvärde har vi använt som måttstock för att jämföra de olika processorerna.

4.1 HashCash

Core I5

Hashcashmetodens defaultvärde ligger på 20 bitar vilket på de flesta nya datorer är ett tämligen lättlöst problem. På en dator med en Intel core i5 processor är det förväntade värdet ca 0.87 sekunder. det förväntade värdet fås fram genom att använda hashcash's inbyggda "expected value"-funktion. Under testerna på en sådan processor visar det sig ta i genomsnitt 1114.8 ms. Anledningen till att värdet avviker så mycket från det förväntade värdet har många förklaringar. Den slutgiltiga hastigheten beror på en hel del olika parametrar (minne, program som är igång etc) och värdet blir ganska individuellt från dator till dator. Det intressanta i denna undersökning är dock jämförelsen med telefonerna. Dock värt att nämnas är att det snabbaste resultatet som uppmättes av Dwork et al [3] i var 4.44 sekunder, detta med en P4 processor som hade en klockhastighet på 3ghz.

ZTE Blade

Efter att ha skrivit en applikation för operativsystemet android, börjar vi med att implementera den på ZTE telefonen vid namn "Blade". Vid defaultvärdet 20 bitar så dödar telefonen applikationen då den tar för mycket processorkraft. Vid sänkning till 19 bitar så händer samma sak. Vid 18 bitar lyckas vi få ut resultat, men telefonen dödar fortfarande applikationen vid 8 av 10 tester. Vid 17 bitar får vi ut mer regelbundna resultat, men även här upprepas ett liknande mönster, telefonen dödar programmet vid 4 av 10 gånger. Först vid sänkning till 15 bitar fungerar protokollet stabilt för mobilen. Lösningstiderna varierar från 1 sekund till uppemot en halv minut, och medelvärdet för lösningstiden hamnar på 9.8 sekunder.

Lösningstid(ms)
15163
12744
6035
4216
2511
1090
10006
23154
18289
5048

HTC Desire

HTC Desire har en starkare processor än vad ZTE Blade har. Den kan klara defaultvärdet, men det tar över 300 sekunder och telefonen krashar 7 av 10 gånger. Vid 19 bitar krashar telefonen 4 av 10 gånger. Protokollet fungerar stabilt först vis 17 bitar. Resultatet redovisas här i nedstående tabell.

Lösningstid(ms)
30479
8488
26915
72124
46524
176297
9566
11626
208712
100229

4.2 Mbound

Core I5

Vi läste igenom Mbound-utvecklarnas avhandling [3]. Då testerna här hade använt bitvärdet 15 så valde vi att göra likadant då det på detta sätt blev enklare för oss att jämföra våra resultat med deras. Efter att ha testat Memory-Bound funktionen med defaultvärdet 15 på en Intel Core i5 processor 10 gånger och fått ut ett medelvärde på 89000 millisekunder. Värt att nämnas är att snabbaste tiden Dwork et al uppmätte var 9.15 sekunder och att samma dator presterade sämre än de andra i hashcash-testen i deras experiment, en gammal dator med en P3 processor.

Figur 4.1. Android-implentation som har dött

ZTE Blade samt HTC Desire

Efter att ha testat implementationen på Intelprocessorn förstår vi snart att detta kommer att bli svårt att få effektivt på en androidtelefon. Våra telefonimplementationer var ineffektiva då de fick android OS att krasha applikationen. Problematiken med telefonimplementationerna behandlas djupare i diskussionen.

4.3 Jämförelse mellan Hashcash och Mbound

Jämförelserna som Dwork et al [3] har gjort ger ett väldigt tydligt resultat, minnestunga problem presterar snarlikt på de flesta datorer. Skillnaderna på en P3 och en P4 dator var som högst en faktor på 1,33. Liknande resultat har även uppmäts på våra experiment, där Hashcash löstes under en sekund(med en väldigt hög svårighetsgrad) och Mbound kunde ta upp emot 1 minut. Hashcash är bevisat väldigt plattformsberoende, vilket också bekräftades när vi testade med smartphones.

Kapitel 5

Diskussion

Diskussion

Frågorna som vi ställt oss själva i början av detta projekt har varit många. Är Proof-Of-Work effektivt? Har det en plats i dagens samhälle, och är det en hållbar lösning inför den nya tekniken som står för dörren? Då det är ett protokoll med så många olika implementationer och med en så stor debatt bakom sig, så har osäkerheten varit ständigt närvarande i början av projektet då vi inte varit övertygande om att kunna få konkreta svar på våra frågor. Då konceptet i sig kan verka aningen abstrakt så får man lätt känslan av att abstrakta svar kan ligga nära till hands för de människor som studerar ämnet.

För att finna svar med verklig substans, har vi funnit det lämpligast att ställa själva konceptet på sin spets. Vad är Proof-Of-Work? Varför har det tagits fram? Vad är det till för?

Som tidigare förklarats så är själva idén med protokollet att bekämpa DoS-angrepp och spam. Det skall helt enkelt göras ineffektivt rent kostnadsmässigt att utföra denna typ av angrepp. Då detta är protokollets huvudsyfte är det naturligt att detta blir den första punkten vi granskar. Efter att ha läst en hel del om protokollet så uppfattar vi det så som att Idén i sig är en mycket god sådan som baserats på vettiga tankar. De stora problem som uppstått och skapat debatt om ämnet är ett resultat av att alla parametrar inte tagits i åtanke när protokollets olika implementationer skapats.

En idé som har diskuterats och setts som en möjlig lösning, eller åtminstone en del av en lösning, är att implementera en sorts ryktesmekanism [10] (engelska: reputation mechanism). Poängen är att en användare som visat sig vara pålitlig och inte blivit anmäld för spam skall ges privilegiet att kunna skicka email till lägre kostnad, eller till slut kanske helt utan kostnad. På detta sätt skulle protokollet kunna skydda genuina användare men fortfarande kunna vara effektivt. Det har desvärre visat sig vara nästan lika svårt att bestäma en hederlighetsskala som att bestämma en universell svårighetsgrad på hashcash problemen. Då denna metod dessutom av förståeliga skäl skulle ha varit nästintill omöjlig för oss att prova själva

valde vi att lägga fokus på andra implementationer.

När vi började utföra tester av hashcash på intel core i5 processorn var vi relativt positivt inställda till protokollet eftersom det gick väldigt snabbt för datorn att lösa problemet. Det tog visserligen över en sekund, men det skilde sig inte mycket från tiden som det vanligtvis tar att skicka ett email. Även om det var längre så var det knappt märkbart. Om vi istället skulle bestämma oss för att skicka tusen meddelanden så skulle tidsskillnaden dock bli kännbar (med våra resultat skulle det tagit ca 17 minuter extra att skicka 1000 email, och om svårighetsgraden på pusslet skulle ökas med 1 bit så skulle tiden nästan fördubblas). Protokollet skulle här fylla sin funktion på ett väldigt effektivt sätt i de fall då vi försöker angripa en identisk dator genom ett DoS angrepp. Detta är dock ett av problemen med protokollet. Det förutsätter att datorn som blir angripen är ungefär jämnstark med angriparen. Protokollet kan då funka som ett skydd, men så fort ens dator blir gammal, eller så fort det kommer ut en något mer kraftfull modell, så finns det stor risk att man kan bli utsatt för denna typ av angrepp igen. Ett annat problem med många av protokollets implementationer är att det ofta finns sätt att kringgå försvarsmekanismen. Som tidigare nämnts så kan en angripare i vissa implementationer, samla på sig alla möjliga stämplar över ett par nätter och därefter utföra ett DoS angrepp utan att protokollet kan stoppa denne. Vad som är ett ännu större problem är att Denial of Service attacker nu för tiden ofta utförs av flera klienter samtidigt. Detta kallas för DDoS attacker. Denna typ av angrep utförs under samma princip som vanliga DoS attacker med skillnaden att det utförst av flera olika personer/datorer på samma gång. Detta gör det väldigt svårt att tillämpa Proof-Of-Work då en vanlig dators processor kommer att ha svårt att mäta sig med, låt säga tio andra datorers processorer. Belastningen blir här alldeles för hög. Detta beror dock helt och hållet på processorena i de inblandade klienterna.

Det stora problem som våra tester tydligast pekar mot har dock med något helt annat att göra än processorskillnaden datorer emellan. Det enligt oss än mer oroande problem är den väg den tekniska utveckligen har tagit. Efter att ha studera resultatrn från vår undersökning och implementerat Proof-Of-Work protokollet på flera androidtelefoner drar vi slutsatsen att dagens telefontillverkare skulle bli tvungna att öka sina telefoners processorkraft med flera hundra procent för att kunna fungera på samma sätt som de gör idag, och detta enbart för att klara av de enklare versionerna av protokollet så som hashcash. Att införa ett protokoll som detta skulle alltså i teorin kunna bli ett hot mot den moderna mobilmarknaden. Eftersom så kallade smartphones väntas bli ledande på den globala mobiltelefonmarknaden [6] inom en snar framtid så kan man dra slutsatsen att telefonsurfandet kommer att öka även det. Om vi till detta lägger till läsplattor, spelkonsoler, mp3spelare och alla andra moderna apparater som i vår tid är uppkopplade till internet och som samtidigt har betydligt svagare processorer än datorer, så ser vi att internetanvändandet skulle begränsas radikalt för alla dessa användare. Det skulle inom en snar framtid kunna röra sig om en ordentlig procentandel av alla världens internetanvändare. I Dwork och Naor's tester av memory-bound funktionen [3] som vi också testat själva, visar det sig att metoden fungerar väldigt bra. De testade under sina experiment äldre

processorer mot nyare sådana, och det visade sig att protokollet gjorde sitt jobb. Problemet med mbound protokollet är att det är relativt bundet till windows. Det går att implementera för andra operativsystem, men det kan ofta bli omständigt då det innehåller funktioner som läser från windowsspecifika filer. I dagsläget finns det många operativsystem som inte har stöd för denna Proof-Of-Work implementation. Android är ett av dessa, och vi lyckades tyvärr inte få vår egen implementation att fungera med detta operativsystem. Det som behövs idag är en plattformsoberoende version av Mbound. Om Proof-Of-Work protokollet skall ha någon form av framtidsutsikter i den värld som håller på att byggas så måste det anpassas till detta tekniska klimat, och av de implementationer vi studerat under projekttiden så har vi inte hittat någon som möter kraven som ställs av dagen teknik. Om ett företag som google skulle ta upp memory-bound funktionen och utvecka android stöd för denna så skulle det kanske finnas en chans för Proof-Of-Work att göra nytta även nu, men för tillfället tycks det inte finnas några sådana planer.

Reflektioner och slutsats

En idé kan vara enkel men samtidigt genialisk. Lika så kan den vara komplex och fullständigt värdelös. Idén om Proof-Of-Work har sedan den presenterades kategoriserats i både den första och den andra kategorin samt en drös andra kategorier däremellan. Version efter version har presenterats, men det har alltid varit något dom saknas. Det är lätt att bilda sig en uppfattning av protokollet men då idén har spridits så mycket och nu finns i så pass många olika former så finns det enligt oss inga argument som biter mot alla implementationer. Det finns ingen fungerande version av Proof-Of-Work, med det finns definitivet ingenting som bevisar att idén inte är genomförbar, därav bör den inte avfärdas till fullo.

Om det visar sig att detta protokoll fungerar så skulle det definitivt kunna bli populärt i många kretsar. Men om sådant är fallet tåls det också att fråga sig själv, om det verkligen är värt det? Är det verkligen en bra idé att att slösa processorkraft på att lösa värdelösa problem? Det finns ju så mycket annatvvall denna kraft skulle kunna användas till [7]. Ett exempel är att man kanske skulle kunna stödja folding@home.projektet och därmed hjälpa den medicinska vetenskapen med sitt projekt att bota cancer. Man kanske skulle kunna betala varje email genom att låta sin dator räkna fram ett antal proteinkombinationer och därmed stödja detta välgörenhetsprojekt. Om detta var något som gjordes till ett gemensamt mål för folk som tror på Proof-Of-Work så skulle det potentiellt kunna locka fler duktiga programmerare till projektet och därmed kanske tillslut utvecklas en version som möter alla de höga krav som ställs på mekanismen.

Det man inte heller får glömma bort är att idéer inte alltid blir som man tänkt sig. Ibland föds riktigt bra tankar som blir till produkter. Dessa produkter kan i slutändan komma att användas till något helt annat än vad det i början var tänka att användas för. Ett exempel är Alfred Nobel och dynamiten, och på ett sätt så gäller detta även Proof-Of-Work. Idén bakom funktionen “captcha” som vi refererat

till tidigare, bygger på precis samma grunder som Proof-Of-Work. Detta används i dagens läge på varje större webbforum på internet och på många andra sajter där formulär fylls i. Kanke är det i denna form Proof-Of-Work gör sig bäst?

Det som står klar är att Proof-Of-Work inte är färdigt. Det är inte redo att användas i dagens läge, men detta betyder inte att det inte skulle kunna fungera. Tid och resurser skulle behövas, och om det skulle finnas någon eller några personer med orken och beslutsamheten att vidareutveckla protokollet så skulle det kunna bli ett otroligt bra verktyg. Om inte, så kommer det dessvärre aldrig att bli något annat än en otroligt bra idé.

Med hjälp av resultatet av våra tester kan vi utan problem konstatera att protokollet funkar bra i teorin. Frågan är om det funkar bra i praktiken.

Då metoden i början är framtagen för att förhindra DoS angrepp så är det naturligt att det är effektiviteten gentemot denna typ av angrepp som vi börjat testa mot.

Litteraturförteckning

[1] The official captcha site. http://www.captcha.net/, april 2011.

[2] Fabien Coelho. An (Almost) Constant-Effort Solution-Verification Proof-of-Work Protocol based on Merkle Trees. In Serge Vaudenay, editor, *Africa Crypt 2008*, number 5023 in LNCS, pages 80–93. Springer Verlag, June 2008. Cryptology eprint Archive 2007/433.

[3] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *CRYPTO*, pages 426–444, 2003.

[4] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, pages 139–147, 1992.

[5] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In *CRYPTO*, pages 37–54, 2005.

[6] Zach Epstein. Ovum: Android to dominate smartphone growth, windows phone will beat blackberry. http://www.bgr.com/2011/03/25/ovum-android-to-dominate-smartphone-growth-windows-phone-will-beat-blackberry/, April 2011.

[7] The Radicati Group Inc. The radicati group releases "email statistics report, 2009-2013". http://www.radicati.com/?p=3237, April 2011.

[8] Marek Klonowski and Tomasz Strumiński. Proofs of communication and its application for fighting spam. In *Proceedings of the 34th conference on Current trends in theory and practice of computer science*, SOFSEM'08, pages 720–730, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Ben Laurie and Richard Clayton. Proof-of-Work Proves Not to Work. In *IN WEAS 04*, 2004.

[10] Debin Liu and L. Jean Camp. Proof of work can work. *NET Institute Working Paper No. 06-18.*, 2006.

[11] John Markoff. Before the gunfire, cyberattacks. *The New York Times*, 2008.

Bilaga A

MBound

Nedanstående beskrivningär baserat på kapitel 6 från [3] och pseudokod är från samma kapitel i avhandligen. Vi har en array A_0 med 256 slumpmässiga 32-bitars ord. I början av det k:te försöket kommer $A = H_0(m, S, R, d, k)$ räknas genom att att ta $A_0 \oplus$ en 256-bitars mask. Sätt att definera H_0 på:

1. Låt $\alpha_k = h(m, S, R, d, k)(|\alpha_k| = 128)$, där h är en stark kryptografisk funktion som SHA-1.

2. Låt $\eta(\alpha_k)$ vara en 2^{13} -bitars sträng som fås genom att konkatenera den 2^7 -biten av α_k med sig självt 2^6 *gånger*[14]. Arrayen A kan ses som en 2^{13} -bitars sträng (genom att konkatenera dess element i en radvis ordning som är standard i de flesta språk), där $A = A_0 \oplus \eta(\alpha_k)$.

Vid initiering är c, den nuvarande platsen i T , dvs de 22 sista bitarna av A (detta är då vi har definerat H_0 med hjälp av att se på A som en sträng). Varje gång $A[i]$ nämns, menas $A[i \bmod 2^8]$ och på samma sätt är $T[c]$ egentligen $T[c \bmod 22^2]$. För att hitta en stig genom mbound:

Initialize Indices:
i = 0, j = 0
for l steps **do** {l is the path length}
$i = i + 1$
$j = j + A[i]$
$A[i] = A[i] + T[c]$
$A[i]$ =RightCyclicShift($A[i]$, 11) {shift forces all 32 bits into play}
Swap($A[i]$, $A[j]$)
$c = T[c] \oplus A[A[i] + A[j]]$
end for{Success occurs if the last e bits of $h(A)$ are all 0}

Sista raden kan vara annorlunda då vi har flera definitioner av H_0, man kan välja att göra detta med hjälp av SHA-1 istället.

Description of H_1 (updates c, leaves A unchanged). The function H_1 is essentially

$i = i + 1$
$j = j + A[i]$
$v = A[i] + T[c]$ {v is a temporary variable}
v =RightCyclicShift(v, 11)
$c = T[c] \oplus A[A[j] + v]$
Description of H_2 {updates A}
$A[i] = A[i] + T[c]$
$A[i]$ = RightCyclicShift($A[i]$, 11)
Swap($A[i]$, $A[j]$)

Hashfunktionen $H_3(A)$ är en stark kryptografisk funktion med 128-bitars output, exempelvis SHA-1.