RaiBlocks: Ett distribuerat och avgiftsfritt kryptovalutenätverk

Colin LeMahieu
clemahieu@gmail.com

***Abstrakt*—På senare tid så har hög efterfrågan och begränsad skalbarhet resulterat i en ökning utav transaktionstider och avgifter hos populära kryptovalutor, vilket lett till en minst sagt bristfällig upplevelse. Här introducerar vi RaiBlocks, en kryptovaluta med en ny block-gitter-arkitektur där varenda konto har en egen blockkedja, som levererar blixtsnabba transaktioner med obegränsad skalbarhet. Varenda användare har en egen blockkedja som låter ägaren att uppdatera den asynkront till resten av nätverket, vilket resulterar i snabba transaktioner med minimal overhead. Transaktioner bearbetar kontons saldon istället för ett transaktionsbelopp, vilket tillåter aggresiv databasbeskärning utan att kompromissa säkerheten. Hittills så har Raiblocks-nätverket bearbetat 4.2 miljoner transaktioner med en icke-beskärd huvudboksstorlek på endast 1.7 GB. RaiBlocks avgiftsfria och momentana transaktioner gör sig till den främsta kryptovalutan för konsumenttransaktioner.**

***Indextermer*—kryptovaluta, blockkedja, raiblocks, distribuerad huvudbok, digital, transaktioner**

I. Introduktion

SEDAN implementationen av Bitcoin 2009 så har det uppstått ett växande skifte från traditionella, statsstödda valutor och finansiella system till moderna betalningssystem baserat på kryptografi, vilket tillåter användare att lagra och överföra medel på ett säkert och icke-förtroendebaserat sätt [1]. För att kunna fungera effektivt, så måste en valuta vara enkelt överförbar, ej reversibel, och ha begränsade eller inga avgifter. De ökade transaktionstiderna, de stora avgifterna, och frågorna om nätverkets skalbarhet har lett till skepticism i huruvida Bitcoin verkligen är lämpad som en alldaglig valuta.

I det här dokumentet så introducerar vi Raiblocks, en låg-latents kryptovaluta byggd på en innovativ block-gitter-datastruktur som erbjuder obegränsad skalbarhet och inga avgifter. RaiBlocks är enligt design ett simpelt protokoll med det enda målet att vara en högpresterande kryptovaluta. RaiBlocks-protokollet kan köras på effektsnål hårdvara, som låter den vara en praktisk, decentraliserad kryptovaluta för vardagligt användande.

Statistik om kryptovalutor som benämns i detta dokument är korrekta vid publiceringsdatumet.

II. Bakgrund

År 2008 publicerade en anonym individ under pseudonymen Satoshi Nakamoto en vitbok som beskriver världens första decentraliserade kryptovaluta, Bitcoin [1]. En innovation som Bitcoin introducerade var blockkedjan, en offentlig, oföränderlig och decentraliserad datastruktur vilket används som huvudbok för valutans transaktioner. Tyvärr, i takt med att Bitcoin började växa, så ledde flera problem i protokollet till att Bitcoin blev mer och mer olämpligt inom många områden:

1) Dålig skalbarhet: Varenda block i blockkedjan kan bara lagra en begränsad mängd data, vilket innebär att systemet endast kan hantera ett visst antal transaktioner per sekund. Detta leder till att platser i ett block blir en sorts handelsvara. I skrivande stund så ligger medianavgiften för en transaktion på $10.38 [2].
2) Hög latens: Den genomsnittliga konfirmationstiden är 164 minuter [3].
3) Ineffektivt: Hela Bitcoin-nätverket konsumerar uppskattningsvis 27.28 TWh per år, och en enkel transaktion beräknas konsumera 260 KWh [4].

Bitcoin, och andra kryptovalutor, fungerar genom att uppnå konsensus på dess globala huvudbok för att verifiera transaktioner, samtidigt som de motarbetar skadliga aktörer. Bitcoin uppnår konsensus genom en ekonomisk åtgärd som kallas Proof of Work (PoW). I ett PoW-system så konkurrerar deltagare genom att beräkna ett nummer som kallas för ett *nonce*, så att hashen av hela blocket är inom ett giltigt intervall. Detta giltiga intervall är proportionellt omvänt till den kumulativa beräkningskraften av hela Bitcoin-nätverket för att upprätthålla en konsekvent genomsnittlig tid för att hitta ett giltigt nonce. Den som hittat ett giltigt nonce får sedan rätten att lägga till ett block på blockkedjan; således, de som tillsätter mer beräkningskraft till nätverket för att beräkna ett giltigt nonce har därför mer ansvar när det kommer till blockkedjans integritet. PoW ger motstånd till en så kallad Sybil-attack, där en enhet uppträder som flera enheter i syfte att få mer makt i ett decentraliserat system, och reducerar race-conditions kraftigt vilket är en oundviklig konsekvens av att modifiera globala datastrukturer.

Ett alternativt konsensus-protokoll, Proof of Stake (PoS), introduceras först 2012 av Peercoin [5]. I ett PoS-system så röstar deltagarna med en vikt som motsvarar den mängd de äger av en given kryptovaluta. Med det här tillvägagångssättet så får de med större finansiella investeringar mer makt i nätverket och blir därmed motiverade att upprätthålla nätverkets integritet för att inte tappa sina investeringar. PoS fungerar utan något sorts slöseri av beräkningskraft likt det som krävs av ett PoW-system, och kan köras på effektsnål hårdvara med enkel mjukvara.

Det ursprungliga RaiBlocks-dokumentet och den första beta-implementationen publicerades i December, 2014, och var en och de första kryptovalutorna baserade på Directed Acyclic Graph (DAG) [6]. Inte långt efteråt så började andra DAG-

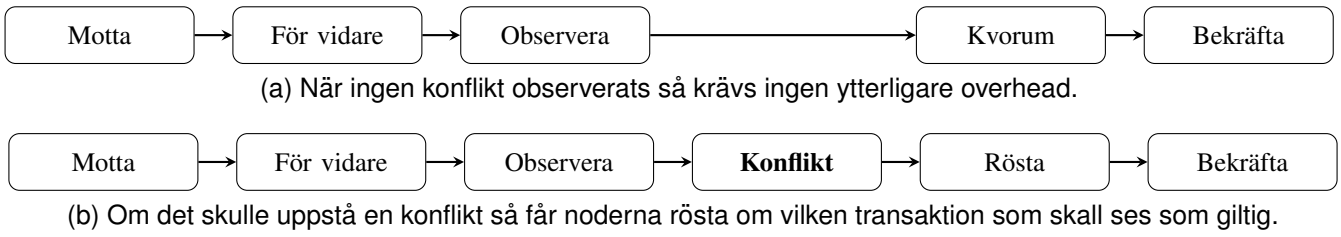

Fig. 1. RaiBlocks kräver ingen extra overhead för typiska transaktioner. Om en konflikt skulle uppstå så röstar noderna om vilken transaktion som skall bestå

kryptovalutor utvecklas, mest känd var DagCoin/ByteBall och IOTA [7], [8]. Dessa DAG-baserade kryptovalutor var de första med att markant bygga ut på det existerande blockkedje-protokollet, vilket resulterat i förbättringar i både prestanda och säkerhet. Byteball uppnår konsensus genom att förlita sig på en "huvudkedja" bestående av ärliga, välkända och tillförlitliga "vittnen", medan IOTA uppnår konsensus via den kumulativa PoW av staplade transaktioner. RaiBlocks uppnår konsensus via en saldo-viktad röst mot stridande transaktioner. Detta konsensus-system tillåter snabbare och mer deterministiska transaktioner samtidigt som man upprätthåller ett starkt, decentraliserat system. RaiBlocks fortsätter denna utveckling och har positionerat sig som en av de mest högpresterande kryptovalutorna.

III. Komponenter i RaiBlocks

Innan RaiBlocks-arkitekturen beskrivs så börjar vi med att definiera upp de individuella komponenter som utgör systemet.

A. Konto

Ett konto utgör den offentliga delen av en digital signaturs nyckelpar. Den offentliga nyckeln, även kallad adressen, delas med andra nätverksdeltagare medan de privata nycklarna hålls hemliga. Ett digitalt signerat datapaket säkerställer att innehållet godkändes av den privata nyckelns ägare. En användare kan kontrollera många konton, men endast en offentlig adress kan existera per konto.

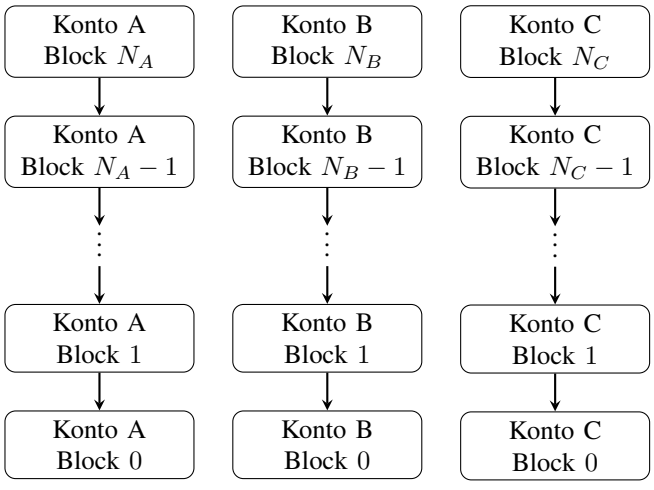

Fig. 2. Varje konto har sin egna blockkedja som innehåller historiken för kontots saldo. Block 0 måste vara en open-transaktion (Sektion IV-B)

B. Block/Transaktion

Termerna "block" och "transaktion" är ofta utbytbara, där ett block innehåller en enstaka transaktion. Transaktioner refererar till själva åtgärden medan block refererar till den digitala kodningen av transaktionen. Transaktioner signeras av den privata nyckel som tillhör det kontot som transaktionen utförs på.

C. Huvudbok

Huvudboken är den globala uppsättningen konton där varje konto har sin egen transaktionskedja (Figure 2). Detta är en nyckeldesignskomponent som har till syfte att ersätta en körtids-överrenskommelse med en designtids-överrenskommelse; alla instämmer via underskrift och kontrollerar att endast en kontoägare kan ändra deras egna kedja. Detta omvandlar en till synes delad datastruktur, en distribuerad huvudbok, till en uppsättning icke-delade.

D. Nod

En *nod* är en mjukvara som körs på en dator som överensstämmer med RaiBlocks-protokollet och deltar i dess nätverk. Mjukvaran hanterar huvudboken och eventuella konton som noden rår över. En nod kan antingen lagra hela huvudboken eller en beskärd historik som bara innehåller de sista få blocken av varje kontokedja. När en ny nod skapas så rekommenderas det att verifiera hela historiken och göra beskärningen lokalt.

IV. Systemöversikt

Till skillnad från blockkedjor som används i många andra kryptovalutor så använder RaiBlocks vad som kallas för en *block-gitter*-struktur. Varenda konto har en egen blockkedja (kontokedja) som motsvarar historiken för kontots transaktioner/saldo (Figur 2). En kontokedja kan bara uppdateras av kontots ägare; detta gör att varenda kontokedja kan uppdateras omedelbart och asynkront med resten av block-gittret vilket resulterar i snabba transaktioner. RaiBlocks protokoll är extremt simpelt; en transaktion är så pass liten att den passar inom de nödvändiga ramarna för ett UDP-paket för överföring via internet. Hårdvarukrav för noder är också minimala, eftersom noder bara behöver skicka vidare block för de flesta transaktioner och inte göra någon PoW (Figure 1).

Systemet initieras med ett *genesiskonto* som innehåller *genesissaldot*. Genesissaldot är en fast mängd och kan aldrig ökas. Genesissaldot delas upp och skickas vidare till andra

konton som är registrerade på genesiskontots kedja via send-transaktioner. Summan av saldon på alla konton kommer aldrig att överstiga det initiala genesissaldot vilket ger systemet en övre gräns för kvantitet och ingen möjlighet att öka den.

Den här sektionen går igenom hur olika typer av transaktioner skapas och propageras genom nätverket.

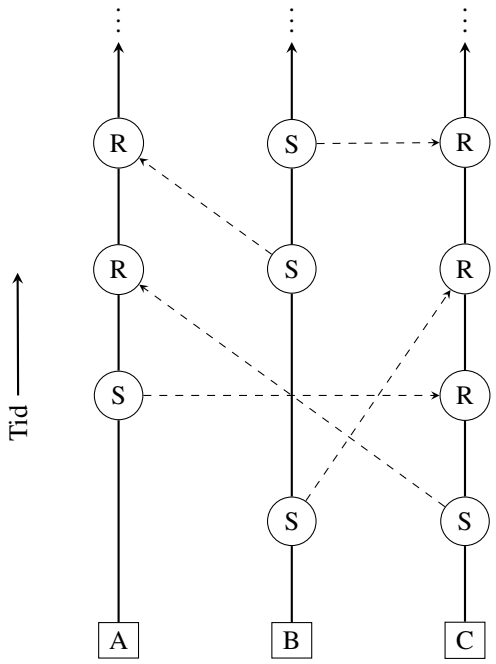

Fig. 3. Visualisering av block-gittret. Varenda förflyttning av summor kräver ett send-block (S) och ett receive-block (R), var och en signerad av kontokedjans ägare (A, B, C)

A. Transaktioner

Överföring av medel från ett konto till ett annat kräver två transaktioner: en *send* som drar en summa från avsändarens konto och en *receive* som lägger till nämnd summa i mottagarens konto (Figure 3).

Att ha denna uppdelning tjänar några viktiga syften:

1) Sekvensering av inkommande överföringar som i sig är asynkrona.
2) Håller transaktioner små så att de passar i UDP-paket.
3) Underlättar beskärning av huvudboken genom att minimera datafotspår.
4) Isolerar uppklarade och ouppklarade transaktioner.

Mer än ett konto som utför transaktioner till samma destination är en asynkron operation; nätverksfördröjning och faktumet att de sändande kontona inte nödvändigtvis är synkroniserade med varandra innebär att det inte finns något sätt att klargöra vilken transaktion som skedde först. Då addition är associativ betyder det att ordningen som transaktionerna kommer in i inte spelar någon roll, och vi behöver bara en global överenskommelse. Detta är en nyckeldesignskomponent som omvandlar en körtids-överenskommelse till en designtids-överenskommelse. Kontot som mottar transaktionerna har makten att välja vilken som anlände först och uttrycks av den signerade ordningen av de inkommande blocken.

Om ett konto vill göra en stor överföring som togs emot som flera små överföringar så vill vi representera det på ett sätt som passar i ett UDP-paket. När ett konto tar emot och skickar transaktioner så håller den en löpande totalsumma av sitt saldo, vilket gör att den har förmågan att när som helst göra en transaktion med fast storlek. Detta skiljer sig från in/ut-modellen för transaktioner som Bitcoin och många andra kryptovalutor använder.

Vissa noder är ointresserade av att använda resurser till att lagra kontons fullständiga transaktionshistorik; de är bara intresserade i kontons nuvarande saldo. När ett konto gör en transaktion så kodas det för sitt nuvarande saldo, och dessa noder behöver bara hålla koll på senaste blocket, vilket låter noderna ignorera historisk data samtidigt som korrektheten upprätthålls.

Även med ett fokus på designtids-överenskommelse så finns det ett fördröjningsfönster vid valideringen av transaktioner då eventuella dåliga aktörer måste identifieras och hanteras. Eftersom överenskommelser i RaiBlocks sker på bråkdelen av en sekund så kan vi presentera användaren med två kategorier av inkommande transaktioner: uppklarade och ouppklarade. Uppklarade transaktioner är transaktioner där ett konto har genererat receive-block. Ouppklarade transaktioner är de som ännu inte har slagits ihop med mottagarens ackumulerade saldo. Detta ersätter de mer komplexa och obekanta tillvägagångssätten för bekräftning som används hos andra kryptovalutor.

B. Skapandet Av Ett Konto

För att skapa ett konto så måste det först utfärdas en *open*-transaktion (Figure 4). En open-transaktion är alltid den första transaktionen på varje kontokedja och skapas första gången en summa mottas. *Account*-fältet lagrar den offentliga nyckeln (adressen), som hör ihop med den privata nyckeln som används för signering. *Source*-fältet innehåller en hash av den transaktion som skickade summan. När kontot skapas så måste en representant väljas i syfte att rösta åt dig; detta kan ändras senare (Sektion IV-F). Kontot kan förklara sig som sin egen representant.

```
open {
   account: DC04354B1...AE8FA2661B2,
   source: DC1E2B3F7C...182A0E26B4A,
   representative: xrb_1anr...posrs,
   work: 0000000000000000,
   type: open,
   signature: 83B0...006433265C7B204
}
```

Fig. 4. Anatomin av en open-transaktion

C. Kontons Saldo

Kontosaldot registreras i huvudboken själv. Istället för att registrera beloppet av en transaktion, så kräver verifieringen (Sektion IV-I) istället skillnaden i saldo mellan send-blocket och saldot i föregående block. Mottagarkontot kan sedan

öka föregående saldo, mätt i det slutliga saldot som ges i det nya receive-blocket. Detta görs för att förbättra bearbetningshastigheten vid nedladdning av höga blockvolymer. När kontohistorik begärs så är saldot redan givet.

D. Skicka Från Ett Konto

För att skicka från en adress så måste adressen redan ha ett befintligt open-block, och därmed ett saldo (Figure 5). *Previous*-fältet refererar till en hash av föregående block i kontokedjan. *destination*-fältet innehåller adressen dit beloppet ska skickas. Ett send-block är inte reversibelt när den väl blivit bekräftad. När send-blocket sänts till nätverket så dras beloppet direkt av från kontots saldo och betraktas som *pending* tills det att mottagaren signerat ett block och accepterat beloppet. Transaktioner med statusen pending bör inte betraktas som att de väntar på bekräftelse, då avsändaren inte kan annulera transaktionen.

```
send {
   previous: 1967EA355...F2F3E5BF801,
   balance: 010a8044a0...1d49289d88c,
   destination: xrb_3w...m37goeuufdp,
   work: 0000000000000000,
   type: send,
   signature: 83B0...006433265C7B204
}
```

Fig. 5. Anatomin av en send-transaktion

E. Ta Emot En Transaktion

För att slutföra en transaktion så måste mottagaren av beloppet skapa ett receive-block på deras egna kontokedja (Figure 6). Source-fältet refererar till en hash av transaktionen som skickade summan. När det här blocket har skapats och sänts till nätverket så uppdateras saldot och beloppet har då officiellt flyttats till kontot.

```
receive {
   previous: DC04354B1...AE8FA2661B2,
   source: DC1E2B3F7C6...182A0E26B4A,
   work: 0000000000000000,
   type: receive,
   signature: 83B0...006433265C7B204
}
```

Fig. 6. Anatomin av en receive-transaktion

F. Tilldelning Av En Representant

Att kontoinnehavare har möjlighet att välja en representant att rösta åt dem är ett kraftfullt decentraliseringsverktyg som inte återfinns i vanliga PoW eller PoS-system. I konventionella PoS-system så måste kontoägarens nod vara igång för att delta i omröstningar. Att ständigt köra en nod är opraktiskt för många användare; att ge en representant befogenhet att rösta åt ett konto gör det hela lättsamt. Kontoinnehavare har möjlighet att ändra konsensus till ett annat konto när som helst. En *change*-transaktion ändrar representanten för ett konto genom att subtrahera röstvikt från den gamla representanten och lägga vikten på den nya representanten (Figure 7). Inga belopp flyttas i denna transaktion, och representanten kan inte spendera det belopp som finns på kontot.

```
change {
   previous: DC04354B1...AE8FA2661B2,
   representative: xrb_1anrz...posrs,
   work: 0000000000000000,
   type: change,
   signature: 83B0...006433265C7B204
}
```

Fig. 7. Anatomin av en change-transaktion

G. Avgrening Och Röstning

En avgrening sker när j signerade block $b_1, b_2, \ldots, b_j$ pekar på samma block som deras föregångare (Figure 8). Dessa block orsakar en motsägande åsikt av kontots status och måste lösas. Endast kontots ägare har möjligheten att signera block in i kontokedjan, så en avgrening måste vara ett resultat av antingen dålig programmering eller skadlig avsikt (dubbelspendering).

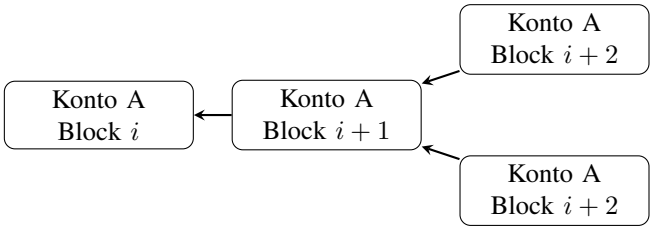

Fig. 8. En avgrening sker när två (eller fler) signerade block refererar samma föregående block. Äldre block ses till vänster och nyare block kan ses till höger

Vid upptäckt så kommer en representant skapa en omröstning som refererar block $\hat{b}_i$ i sin huvudbok och sedan sända det till nätverket. Vikten av en nods röst, w_i, är summan av saldot i alla konton som har utgett noden som sin representant. Noden kommer observera inkommande röster från de andra M representanterna och hålla en kumulativ räkning i 4 röstperioder, 1 minut totalt, och bekräfta det vinnande blocket (Ekvation 1).

$$v(b_j) = \sum_{i=1}^{M} w_i \mathbb{1}_{\hat{b}_i = b_j} \tag{1}$$

$$b^* = \arg\max_{b_j} v(b_j) \tag{2}$$

Det mest populära blocket b^* kommer ha majoriteten av alla röster och får därmed behållas i nodens huvudbok (Ekvation 2). De block som förlorar omröstningen kastas bort. Om en representant ersätter ett block i sin huvudbok så kommer det

att skapas en ny röst med ett högre sekvensnummer som sänds till nätverket. Det här är det **enda** scenariot där representanter röstar.

Under vissa omständigheter kan korta nätverksproblem orsaka att ett sänt block inte accepteras av alla noder. Alla efterföljande block på det här kontot kommer att ignoreras som ogiltiga av de noder som inte såg den ursprungliga sändningen. En omsändning av detta block kommer att accepteras av de resterande noderna och efterföljande block kommer då att hämtas automatiskt. Även när en avgrening eller ett saknat block uppstår, påverkas endast de konton som är delaktiga i transaktionen; resten av nätverket fortsätter bearbeta transaktioner för alla andra konton.

H. Proof of Work

Alla fyra transaktionstyper har ett work-fält som måste fyllas i korrekt. Här måste skaparen av transaktionen beräkna en nonce så att hashen av noncen konkatenerat med previous-fältet i receive/send/change-transaktioner eller account-fältet i en open-transaktion ligger under ett visst tröskelvärde. Till skillnad från Bitcoin används PoW i RaiBlocks helt enkelt som ett verktyg för att motverka spam (som liknar Hashcash [9]), och det kan beräknas snabbare än en sekund. När en transaktion skickats kan PoW för det efterföljande blocket räknas ut i förväg då previous-fältet är känt; detta leder till att transaktioner upplevs som omedelbara för användaren så länge tiden mellan transaktioner är längre än den tid det tar att utföra PoW.

I. Verifikation Av Transaktioner

För att ett block skall anses som giltigt så måste den ha följande attribut:

1) Blocket kan inte redan existera i huvudboken (dubblett).
2) Måste vara signerad av kontots ägare.
3) Det föregående blocket är huvudblocket i kontokedjan. Om det existerar men inte är huvudblocket så är det en avgrening.
4) Kontot måste ha ett open-block.
5) Den uträknade hashen måste möta kraven för PoW:ens tröskelvärden.

Om det är ett receive-block, kolla om källblockets hash är väntande, vilket innebär att den inte blivit utlöst. Om det är ett send-block så måste saldot vara mindre än det föregående saldot.

V. Attackvektorer

RaiBlock, som alla decentraliserade kryptovalutor, kan bli attackerade av skadliga parter i försök till ekonomisk vinst eller i syfte att ta död på systemet. I den här sektionen beskriver vi några möjliga angreppsscenarier, konsekvenserna av en sådan attack och hur RaiBlocks protokoll tar förebyggande åtgärder.

A. Blockgaps-synkronisering

I Sektion IV-G så diskuterades scenariot där ett block kanske inte sänds korrekt, vilket leder till att nätverket ignorerar efterföljande block. Om en nod observerar ett block som inte har det refererade föregående blocket så har den två alternativ:

1) Ignorera blocket eftersom det kan vara dåligt/skadligt.
2) Begär en omsynkronisering med en annan nod.

Vid en omsynkronisering så måste en TCP-anslutning formas med en bootstrapping-nod för att underlätta den ökade trafiken som en omsynkronisering kräver. Om blocket skulle visa sig vara dåligt så var omsynkronisering onödig och har onekligen ökat trafiken på nätverket. Detta är en nätverksförstärkningsattack och resulterar i denial-of-service.

För att undvika onödig omsynkronisering så väntar noder tills det uppnåtts ett visst tröskelvärde av röster för ett potentiellt dåligt block innan en hel omsynkronisering görs. Om ett block inte får tillräckligt med röster anses det vara dåligt och ignoreras.

B. Transaktionsöversvämning

En skadlig enhet kan skicka många onödiga men giltiga transaktioner mellan konton i ett försök att mätta nätverket. Då det inte finns några transaktionsavgifter så skulle en sådan attack kunna pågå hur länge som helst. Den PoW som krävs för varje transaktion begränsar dock frekvensen som den skadliga enheten skulle kunna generera utan att investera avsevärt i beräkningsresurser. Även under en sådan attack i ett försök att översvämma huvudboken så kan noder beskära gamla transaktioner från kedjan, vilket resulterar i att ett konstant lagringsutrymme används.

C. Sybil-attack

En enhet kan skapa flera hundra RaiBlocks-noder på en maskin; dock, eftersom att röstningssystemet är viktat baserat på kontons saldo, så kommer inte fler noder ge enheten fler röster. Det finns därför ingenting att vinna via en Sybil-attack.

D. Pennyspenderingsattack

En pennyspenderingsattack går till så att en enhet spenderar infinitesimala kvantiteter till ett stort antal konton för att slösa lagringsresurser hos noder. Blockpublicering är hastighetsbegränsad av PoW, så det begränsar till viss del skapandet av konton och transaktioner. Noder som inte lagrar hela historiken kan beskära konton under en statistisk gräns där kontot med största sannolikhet inte är ett giltigt konto. RaiBlocks är även inställt på att använda minimalt med lagringsutrymme, och det utrymme som krävs för att lagra ytterligare ett konto är proportionellt mot storleken på ett öppet block + indexeringen = 96B + 32B = 128B. Detta motsvarar att 1GB kan lagra totalt 8 miljoner konton för pennyspendering. Om noder vill beskära mer aggressivt kan de beräkna en distribution baserat på åtkomstfrekvens och delegera konton som sällan används till långsammare lagring.

E. Förberäknad PoW-attack

Eftersom ägaren av ett konto kommer vara den enda enheten som lägger till block i kontokedjan kan sekventiella block beräknas tillsammans med deras PoW innan de sänds till nätverket. Här genererar angriparen en otalig mängd av sekventiella block, var och en med minimala värden, över en längre tidsperiod. Vid en viss punkt så utför angriparen en Denial of Service (DoS) genom att översvämma nätverket med massvis av giltiga transaktioner, vilket de andra noderna kommer behandla och skicka vidare så snabbt som möjligt. Detta är en avancerad version av transaktionsöversvämning som beskrivs i Sektion V-B. En sådan attack skulle bara fungera kortfattat, men skulle kunna användas i samband med andra attacker, till exempel en >50%-attack (Sektion V-F) för att öka effektiviteten. Hastighetsbegränsning för transaktioner och andra tekniker undersöks för närvarande för att mildra attacker.

F. >50%-attack

Raiblocks använder ett saldo-viktat röstningssystem för att uppnå konsensus. Om en angripare lyckas få kontroll på över 50% av rösterna så skulle det vara möjligt att sprida konsensus över nätverket, vilket skulle rendera systemet förstört. En angripare kan sänka saldot som de måste förlora genom att förhindra att goda noder röstar via en nätverks-DoS. RaiBlocks vidtar följande åtgärder för att förhindra en sådan attack.

1) Det grundläggande försvaret mot denna typ av attack är att röstvikt är direkt bunden till mängden investeringar i systemet. Kontoägare motiveras till att upprätthålla systemets integritet i syfte att skydda deras investeringar. Att manipulera huvudboken skulle vara skadligt för systemet vilket leder till att investeringarna fördärvas.
2) Kostnaden för en attack av denna typ är proportionell mot RaiBlocks marknadsvärde. I PoW-system kan teknik uppfinnas som ger oproportionerlig kontroll jämfört med monetära investeringar och om attacken lyckas så kan denna teknik fortsätta användas efter att attacken är avslutad. Med RaiBlocks så skalar kostnaden för att attackera systemet med storleken på systemet i sig, och om en attack skulle bli lyckad så kan investeringen i attacken inte återanvändas.
3) För att alltid ha så många tillgängliga röstare som möjligt så använder vi oss utav så kallad representativ röstning. Kontoägare som inte kan delta i omröstningar på grund utav anslutningsproblem eller liknande kan välja en representant som röstar med kontots saldo. Att maximera antalet röstare resulterar i ett starkt och hållbart nätverk.
4) Avgreningar i RaiBlocks uppstår aldrig av misstag, så noder får göra ett eget val i hur hanteringen av dessa avgrenade block ska ske. Enda scenariot när icket-attackerande konton är sårbara för avgrenade block är om de mottar ett saldo från ett attackerande konto. Konton som vill vara säkra från avgreningar kan vänta ett längre tag innan de tar emot från ett konto som genererat dessa. De kan även välja att inte ta emot någonting överhuvudtaget. Mottagare kan också generera separata konton att använda för att ta emot summor från tvivelaktiga konton, i syfte att isolera sina andra konton.
5) En sista försvarsmekanism som ännu inte implementerats är *blockcementering*. RaiBlocks gör stora insatser för att snabbt kunna hantera avgreningar via omröstningar. Noder kan konfigureras till att cementera block, vilket skulle förhindra att de rullas tillbaka efter en viss tidsperiod. Nätverket är tillräckligt säkert då fokus ligger på snabba transaktionstider vilket förhindrar tvetydiga avgreningar.

En mer sofistikerad version av en $> 50\%$-attack anges i Figur 9. “Offline” är andelen representanter som har blivit identifierade men som inte är uppkopplade för att rösta. “Stake” är summan av alla saldon som attackeraren röstar med. “Active” är representanter som är uppkopplade och som röstar enligt protokollet. En angripare kan minska summan de måste sätta in genom att tvinga bort övriga deltagare från omröstningen genom en nätverks-DoS. Om den här attacken kan upprätthållas så kommer de representanter som attackeras bli osynkroniserade, vilket demonstreras med “Unsync”. Slutligen kan en angripare få en kort bristrelaterad röstningsstyrka genom att byta anfall till en ny grupp representanter medan den gamla gruppen omsynkroniserar sin huvudbok, vilket demonstreras med “Attack”.

Offline	Unsync	**Attack**	Active	Stake

Fig. 9. Ett potentiellt röstningsarrangemang som kan sänka kraven för en 51%-attack.

Om en angripare lyckas orsaka Stake $>$ Active genom att kombinera dessa förhållanden så skulle det vara möjligt att manipulera omröstningar på huvudboken under angriparens förutsättningar. Vi kan estimera hur mycket en attack av den här typen skulle kunna kosta genom att examinera marknadsvärdet hos andra system. Om vi antar att 33% av representanter inte är uppkopplade eller under DoS-attack, så skulle en angripare behöva köpa 33% av marknadsvärdet för att kunna attackera systemet via röstning.

G. Bootstrapförgiftning

Ju längre en angripare har möjlighet att hålla en gammal privat nyckel med ett saldo, desto högre är sannolikheten att saldon som existerade vid senaste uppdateringen inte kommer att ha medverkande representanter då dess saldon eller representanter har flyttats till nyare konton. Det här innebär att om en nod blir uppstartad till en gammal representation av nätverket där en angripare håller en majoritet av rösterna, så skulle de kunna skicka röstbeslut till den noden. Om denna nya användare skulle vilja interagera med någon annat förutom den attackerande noden så skulle alla dess transaktioner nekas då de har olika huvudblock. Huvudproblemet är alltså att noder kan få andra, nyskapade noder, att slösa tid genom att mata dem med felaktig information. För att förhindra detta kan noder kopplas samman med en initial databas som innehåller

konton och pålitliga huvudblock. Detta istället för att behöva ladda ner databasen hela vägen tillbaka till genesisblocket. Ju närmare nerladdning är till att vara aktuell, desto större är chansen att kunna avvärja en attack av denna typ. I slutändan så är dessa attacker förmodligen inte allvarligare än att skicka skräpdata till noder medan de startas upp, eftersom de inte skulle kunna utföra transaktioner med någon som har en aktuell databas.

VI. Implementation

Referensimplementationen är för nuvarande implementerad i C++ och har aktivt arbetats på sedan 2014 på GitHub [10].

A. Designegenskaper

Implementeringen av RaiBlocks följer arkitekturstandarden som beskrivs i detta dokument. Ytterligare specifikationer beskrivs här.

1) Signeringsalgoritm: RaiBlocks använder en modifierad ED25519 elliptic curve-algoritm med Blake2b hashning för alla digitala signaturer [11]. ED25519 valdes för dess snabba signering, snabba verifikation, och höga säkerhet.

2) Hashalgoritm: Eftersom hashalgoritmen endast används för att motverka nätverksspam så är valet mindre viktigt jämfört med mining-baserade kryptovalutor. Vår implementation använder Blake2b som en digest-algoritm för blockinnehåll [12].

3) Härledningsfunktion för nycklar: I referensplånboken krypteras nycklarna med ett lösenord. Det här lösenordet matas genom en härledningsfunktion för att skydda mot crackningsförsök av ASIC:s. I nuläget så är Argon2 [13] vinnaren av den enda offentliga tävlingen som haft som mål att ta fram en beständig härledningsfunktion.

4) Blockintervall: Eftersom varenda konto har sin egna blockkedja, kan uppdateringar utföras asynkront i nätverket. Det finns därför inga blockintervaller och transaktioner kan publiceras omedelbart

5) UDP Message Protocol: Vårt system är designat för att fungera oändligt länge med så lite datorresurser som möjligt. Alla meddelanden i systemet är utformade för att vara stateless och få plats i ett enda UDP-paket. Detta gör det också lättare för parter med dålig anslutning att delta i nätverket då de inte ständigt behöver skapa nya TCP-anslutningar. TCP används endast för nya parter när de vill synkronisera blockkedjorna i ett enda svep.

Noder kan vara säkra på att deras transaktion tagits emot av nätverket genom att observera trafiken från andra noder, då den kommer få flera kopior ekande tillbaka till sig själv.

B. IPv6 och Multicast

Genom att bygga ovanpå anslutningslösa UDP så kan framtida implementationer använda sig utav IPv6 multicast som ersättning för traditionell transaktion- och röstningssändning. Detta kommer att minska förbrukningen av nätverksbandbredd och ge mer flexibilitet till noder.

C. Prestanda

När detta skrivs så har 4.2 miljon transaktioner behandlats på RaiBlocks-nätverket, vilket resulterar i en blockkedjestorlek på 1.7 GB. Transaktionstider mäts i sekunder. En nuvarande referensimplementation på en SSD kan behandla över 10’000 transaktioner per sekund, och är huvudsakligen IO-bundet.

VII. Resursanvändning

Detta är en översikt över resurser som som används av en RaiBlocks-nod. Vi tar dessutom upp idéer för att minska resursanvändningen för specifika användningsfall. Reducerade noder benämns typiskt för lätt, beskärd, eller en simplified payment verification-nod (SPV).

A. Nätverk

Mängden nätverksaktivitet beror på hur mycket noden i fråga bidrar till nätverkets integritet.

1) Representant: En representativ nod kräver maximala nätverksresurser eftersom den observerar röstningstrafik från andra representanter och publicerar sina egna röster.

2) Icke-förtroendebaserad: En icke-förtroendebaserad nod är väldigt lik en representativ nod men observerar endast; den innehar inte en representants privata nyckel och publicerar inte egna röster.

3) Litande: En litande nod observerar röstningstrafik från en representant som den förutsätter utför konsensus korrekt. Det här skär ner på inkommande röstningstrafik från representanter som pekar på den här noden.

4) Lätt: En lätt nod är en litande nod som endast observerar trafik för konton som den är intresserad av, vilket resulterar i minimal nätverksanvändning.

5) Synkronisering: En synkroniseringsnod skickar ut delar eller hela huvudboken till noder som behöver synkroniseras med nätverket. Detta görs via en TCP-anslutning i stället för UDP eftersom det oftast är en stor mängd data som kräver avancerad flödesstyrning.

B. Lagringskapacitet

Beroende på användarens krav så kommer olika nodkonfigurationer kräva olika lagringsbehov.

1) Historisk: En nod som är intresserad av att lagra en fullständig historik över alla transaktioner kommer kräva en maximal lagringsmängd.

2) Löpande: I och med designvalet att lagra det ackumulerade saldot i blocken så behöver noder bara lagra det senaste blocken för varje konto. Om en nod skulle vilja skära ner på det ännu mer så är det fullt möjligt att endast lagra huvudblocken.

3) Lätt: En lätt nod behåller ingen lokal huvudbok utan deltar bara i nätverket för att observera konton som den är intresserad av, men kan även skapa nya transaktioner med eventuella privata nycklar som den kan ha i sin ägo.

C. CPU

1) Transaktionsgenerering: En nod som är intresserad av att skapa nya transaktioner måste producera en giltig Proof of Work-nonce för att ta sig förbi RaiBlocks hastighetsbegränsningsmekanism. En jämförelse i beräkningskraft mellan olika hårdvaror återfinns i Appendix A.

2) Representant: En representant måste verifiera signaturer för block, röster, och även skapa egna signaturer för att kunna delta i konsensus. Mängden CPU-resurser som krävs för en representativ nod är betydligt mindre än det som krävs för transaktionsgenerering och bör gå att köra på vilken modern dator som helst.

3) Observatör: En observatörsnod genererar inte egna röster. Eftersom signaturgenereringens overhead är minimal är CPU-kraven nästan identiska med att köra en representativ nod.

VIII. Slutsats

I detta dokument presenterade vi ramverket för en icke-förtroendebaserad, avgiftsfri, låg-latens kryptovaluta som använder sig av en ny block-gitter-datastruktur och delegerad Proof of Stake-röstning. Nätverket använder minimala resurser, har ingen kraftkrävande mining, och klarar av höga transaktionshastigheter. Allt detta är möjligt tack vare att varenda konto har en egen individuell blockkedja, vilket eliminerar åtkomstproblem och ineffektiviteten i en global datastruktur. Vi identifierade möjliga attackvektorer i systemet och presenterade argument för hur RaiBlocks är immun mot dessa typer av attacker.

Bilaga A
PoW Hardware benchmarks

Som tidigare nämnt så har PoW som syfte att minska nätverksspam. Vår nod-implementation kan utnyttja OpenCL-kompatibla GPU-enheter för acceleration i beräknandet. Table I visar en jämförelse av hur olika hårdvaror presterar i verkligheten. I skrivande stund så är PoW-tröskeln konstant, men ett adaptivt tröskelvärde kan komma att implementeras i framtiden i och med att den genomsnittliga datorkraften ökar.

Tabell I
Hardware PoW Performance

Enhet	Transaktioner per sekund
Nvidia Tesla V100 (AWS)	6.4
Nvidia Tesla P100 (Google,Cloud)	4.9
Nvidia Tesla K80 (Google,Cloud)	1.64
AMD RX 470 OC	1.59
Nvidia GTX 1060 3GB	1.25
Intel Core i7 4790K AVX2	0.33
Intel Core i7 4790K,WebAssembly (Firefox)	0.14
Google Cloud 4 vCores	0.14-0.16
ARM64 server 4 cores (Scaleway)	0.05-0.07

Acknowledgment

Vi vill tacka Brian Pugh för att ha sammanställt och formaterat detta dokument.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008. [Online]. Available: http://bitcoin.org/bitcoin.pdf
[2] “Bitcoin median transaction fee historical chart.” [Online]. Available: https://bitinfocharts.com/comparison/bitcoin-median_transaction_fee.html
[3] “Bitcoin average confirmation time.” [Online]. Available: https://blockchain.info/charts/avg-confirmation-time
[4] “Bitcoin energy consumption index.” [Online]. Available: https://digiconomist.net/bitcoin-energy-consumption
[5] S. King and S. Nadal, “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake,” 2012. [Online]. Available: https://peercoin.net/assets/paper/peercoin-paper.pdf
[6] C. LeMahieu, “Raiblocks distributed ledger network,” 2014.
[7] Y. Ribero and D. Raissar, “Dagcoin whitepaper,” 2015.
[8] S. Popov, “The tangle,” 2016.
[9] A. Back, “Hashcash - a denial of service counter-measure,” 2002. [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf
[10] C. LeMahieu, “Raiblocks,” 2014. [Online]. Available: https://github.com/clemahieu/raiblocks
[11] D. J. Bernstein, N. Duif, T. Lange, P. Shwabe, and B.-Y. Yang, “High-speed high-security signatures,” 2011. [Online]. Available: http://ed25519.cr.yp.to/ed25519-20110926.pdf
[12] J.-P. Aumasson, S. Neves, Z. Wilcox-O’Hearn, and C. Winnerlein, “Blake2: Simpler, smaller, fast as md5,” 2012. [Online]. Available: https://blake2.net/blake2.pdf
[13] A. Biryukov, D. Dinu, and D. Khovratovich, “Argon2: The memory-hard function for password hashing and other applications,” 2015. [Online]. Available: https://password-hashing.net/argon2-specs.pdf