Distribuerad Simulation av Multihop Routing Baserat på HLA

Lukas Pohlman
lukpo970@student.liu.se

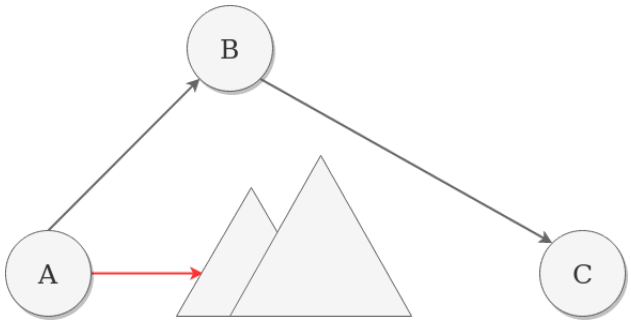
Mars, 2020

Innehåll

1 Inledning

1.1 Motivering

Pitch Technologies AB (hädanefter kallad Pitch) har utvecklat en simulerad miljö där militär effektivt kan öva på stridsföring. I miljön kan man bland annat simulera kommunikation mellan radioapparater, men det finns ännu inget stöd för att simulera multihop routing. Multihop routing är en teknik som moderna radionätverk kan använda för expandera radioapparaternas räckvidd i nätverket. Detta möjliggörs genom att kommunikationen reläas genom andra radioapparater fram till destinationen om den direkta kommunikationen mellan två radioapparater inte är möjlig. Den befintliga lösningen har endast stöd för att simulera kommunikationen mellan radioapparater direkt kopplade till varandra i nätverket. Om terräng eller avstånd hindrar signalen att nå mottagaren så kan de alltså inte kommunicera. I denna rapport utvecklas ett verktyg för att simulera ett nätverk som även har stöd för multihop routing. Simulatorn avgör vilka vägar som kommunikationen kan ta genom nätverket och väljer den kortaste vägen till målet.



Figur 1: Nod A vill kommunicera med nod C, men signalen kommer inte fram. Nod B, som har uppkoppling med både A och C, kan då användas för att reläa kommunikationen vidare till C.

I denna simulator avgör både vilken typ av kommunikation ska föras över nätverket samt förbindelsernas egenskaper om kommunikation mellan två noder är möjlig. I kontexten av radionätverk så kan förbindelsernas egenskaper handla om till exempel latens, bandbredd och paketförlust, medans vad som förs över nätverket kan vara datakommunikation, såsom text och symboler, eller samtalskommunikation. Det är viktigt att veta vad som ska föras

över nätverket eftersom olika typer av kommunikation kräver olika resurser av nätverket. En uppkoppling som är tillräckligt bra för datakommunikation behöver inte vara tillräckligt bra för en samtalskommunikation, och vice versa.

1.2 Syfte

Eftersom nyare radioapparater har stöd för multihop routing så är syftet att den större simulatorn i och med integrationen ska representera radioapparaternas faktiska förmågor bättre. Målsättningen är sedan att i framtiden integrera denna simulator med den större simulator som utvecklats av Pitch i uppdrag av den svenska militären. Rapporten utreder olika arkitekturer och lösningar på simulatorn som möjliggör denna integration.

1.3 Problemformulering

Algoritmen som hittar vägen mellan två radioapparater i nätverket är en central del av simulatorn och behöver studeras med avseende på hur väl de presterar i olika praktiska sammanhang.

Simulatorn kommer att behöva ta hand om väldigt många tunga beräkningar på kort tid. Det finns användningsfall med upp till 200 noder i nätverket, och varje nod kan potentiellt behöva veta den kortaste vägen till alla andra noder i nätverket varje sekund. Vi behöver ta fram en simulator som kan utföra beräkningarna den tar emot snabbare än vad den får nya. Lösningen kommer att utvärderas med hänsyn till dess effektivitet (hur snabb är den / hur många beräkningar den klarar av), hur skalbar är den (tidskomplexitet), resurshantering (hur slösaktig den är med hård- och mjukvara), ansvarsfördelning (om simulatorn hand om saker som andra borde ta hand om), och hur enkel den är att implementera i praktiken. Simulatorn skall även vara designad och implementerad enligt standarden High Level Architecture (HLA).

1.4 Avgränsningar

1. Eftersom radiokommunikationens kvalité avtar mellan varje hopp så är det sagt att antalet hopp från den ursprungliga noden inte får överstiga fyra. Mer specifikt kommer det maximala antalet tillåtna hopp att ligga mellan 2-4, och beror på vilken typ av data som skickas över

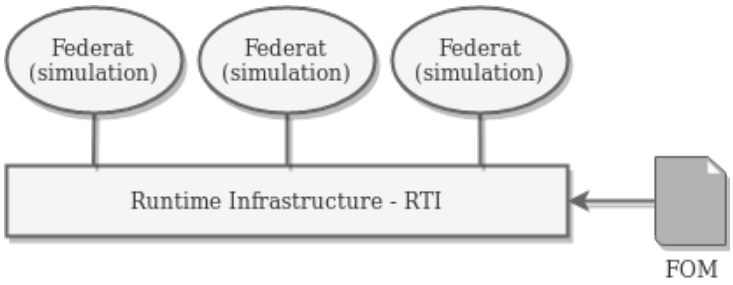
nätverket. Det innebär att kraven på simulatorns prestanda är lägre eftersom trädet som måste vandras genom garanterat aldrig har en höjd större än fyra.

2. I denna rapport hittas endast en väg mellan två noder som har tillräckligt bra uppkoppling för att kunna kommunicera, men det kan även vara intressant att hitta den väg genom grafen med bäst uppkoppling. Att hitta den bästa uppkopplingen mellan två noder är ett problem som hade krävt att alla noder i nätverket vandrats genom.

2 Bakgrund

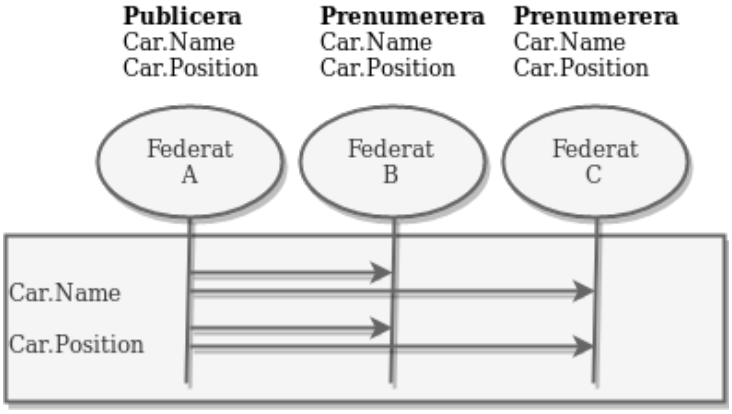
2.1 High Level Architecture

Simulatorn är designad i enlighet med standarden High Level Architecture (HLA)[5]. HLA är ett kraftfullt verktyg som utvecklades år 1990 för distribuerade simuleringar. Syftet är interoperabilitet: att tillåta olika simuleringssystem att kommunicera och arbeta tillsammans. Med hjälp av en sådan arkitektur blir det enkelt att koppla ihop många små simulationer (kallade federater) till en så kallad Runtime Infrastructure (RTI) som fungerar som en servicebuss. Figur 3 illustrerar topologin i en HLA-arkitektur. Ett godtyckligt antal system har en, och endast en, koppling till RTI:n. HLA beskriver vilka gränssnitt och egenskaper en federat måste ha för att kunna interagera med RTI:n. RTI:n förser federaterna med information samt koordinerar och synkroniserar alla komponenter. En Federation Object Model (FOM), beskriver datautbytet i federationen och kan ses som federationens språk.



Figur 2: Topologi i High Level Architecture

Federationen tillåter en federat att publicera sin egen information och prenumerera på information från andra federater enligt Figur 3.



Figur 3: Publikation och prenumeration

Dahmann [4] beskriver att HLA kommer med ett antal utmaningar som inkluderar att förbättra prestandan på RTI:n och att hitta nya innovativa implementationer av RTI:n.

2.2 Inmatningsformat

En objektmodell, FOM, skapas för information om förbindelserna mellan noderna i nätverket. Objektmodellen bidrar med information som kan liknas vid grannmatriser för viktade grafer. Det bör noteras att denna simulator kommer avgöra grafens utseende beroende på ett antal olika grannmatriser, och inte endast på en enda grannmatris. Enligt Tabell 1(a) är till exempel latensen från Nod A till Nod B 128 millisekunder. Notera att exempelvis latensen från nod A till nod B inte nödvändigtvis är densamma som latensen från nod B till nod A. Ett godtyckligt antal tabeller likt dessa kan användas för att beskriva förbindelsen mellan ett godtyckligt antal noder i ett nätverk.

Nod	A	B	C	D
A	0	128	97	95
B	57	0	104	111
C	71	45	0	91
D	95	124	136	0

(a) Latens (ms) mellan noderna

Nod	A	B	C	D
A	0	1028	797	395
B	857	0	1004	711
C	761	453	0	931
D	954	1240	736	0

(b) Bandbredd (kB/s) mellan noderna

Tabell 1: Exempel på inmatning från RTI

3 Teori

Radionätverket kan betraktas som en graf där varje nod representerar en radio och varje båge representerar en förbindelse över vilken direkt kommunikation mellan två radioapparater är möjlig. Om en nod vill upprätta en förbindelse till en nod som den inte kan kommunicera direkt med, så kan en alternativ väg hittas genom grafen. Tekniken att hitta en alternativ väg mellan två noder som går genom en sekvens av andra noder i grafen kallas för multihop routing. I fallet som studeras är det intressant att hitta en lösning på hur noderna på ett så effektivt sätt som möjligt kan kommunicera med alla andra noder i nätverket. För vår del så bör lösningen kunna hantera att beräkna den kortaste vägen från alla noder till alla andra noder på under en sekund. Med upp till 200 noder i nätverket så kan det handla om 39800 stycken kortaste vägar som ska beräknas varje sekund. Olika lösningar studeras för att tillfredsställa detta krav. I detta kapitel kommer vi att diskutera hur valet av centraliserade- eller distribuerade sökalgoritmer hade påverkat prestationen av simuleringen. Därefter kommer vi att gå igenom specifikt vilken algoritm som är mest passande och sedan hur distribuerade simulationer har implementerats tidigare.

3.1 Centraliserade och distribuerade modeller

Många routingalgorimer bygger på en distribuerad arkitektur [12, 11, 6]. Dessa arkitekturer kräver ingen central maskin som utför beräkningar åt dem, dessa presterar generellt sätt väldigt bra och är skalbara. Andra är centraliserade eller semicentraliserade [10, 7, 8] och bygger på en server (eller en mindre mängd servrar) som beräknar alla kortaste vägar i nätverket. Servern håller på global kunskap om nätverket och kräver att mycket kommunikation förs mellan noderna och servern för att servern ska veta hur den bör rutta.

Peterson et al. [13] argumenterar att ett nytt intresse i centraliserad routing har uppstått i och med att kommunikationsmedel har blivit mer pålitliga och presterar bättre. I jämförelse med den distribuerade modellen så bidrar en centraliserad modell med en överlägsen styrförmåga över hur kommunikationen ska ruttas genom nätverket och en klarare syn över resultaten. Alla prestandakrav läggs på ett ställe med en centraliserad modell, allt underhåll och alla uppdateringar behöver bara göras på en maskin. Något som ändå talar för en distribuerad modell är att om servern vill att alla noder i nätverket

ska veta positionen av alla andra noder i nätverket så blir tidskomplexiteten $\mathcal{O}(N^2)$. Om istället alla noder i nätverket själva hade skickat sin position till alla andra noder så hade tidskomplexiteten legat på $\mathcal{O}(N)$ per maskin. Lösningen är med andra ord väldigt oskalbar, komplexiteten ökar drastiskt med antal noder i nätverket, och mycket resurser såsom hårdvara, mjukvara och arbetskraft behövs för att bygga upp en tillräckligt effektiv simulator som tar hand om alla förfrågningar från noderna. Den centraliserade tekniken kan utökas med en mindre mängd servrar som kan dela på belastningen och fungera som backups vid krasch. På så vis kan flera av de stora nackdelarna med en centraliserad modell kringgås. Det bör också nämnas att eftersom det handlar om en simulering av nätverket så är hotet för (och allvarlighetsgraden i) en krasch av den centrala servern mycket lägre än om nätverket skulle implementerats på riktigt".

En federation kan även innehålla mer än bara maskiner som är med i den faktiska simulationen. Till exempel så kan det finnas passiva observatörer och simulationssurrogater. Cai et al. [2] beskriver ett Load Management System för att även distribuera lasten över de maskiner som inte är aktivt deltagande i simulationen, men som ändå har resurser för att ta hand om beräkningar. Detta är en intressant lösning att ha i åtanke, men eftersom hårdvara inte är en vidare bristvara för Pitch så är en sådan åtgärd inte nödvändig i nuläget.

3.2 Bruteforce och request-response

Beräkningen av den kortaste vägen kan antingen göras då den efterfrågas (request-response), eller så kan alla kortaste vägar beräknas i ett förebyggande syfte (bruteforce). Vilken metod som är mest lämplig beror dels på hur stor andel av de kortaste vägar som ändå kommer behöva beräknas och, om en centraliserad modell implementeras, på hur stor tidsmässig fördröjning som följer att en nod skickar iväg en förfrågan om den kortaste vägen till servern. Eftersom det handlar om en tidsfördröjning i millisekunder att nå servern över RTI:n, så betraktas den däremot inte som något större hot. Det kan därför sägas att en request-response lösning är mer förmånlig om det finns flera noder som inte behöver ha svar på kortaste vägen i grafen. Denna lösning är även hållbar på lång sikt: simulatorn kommer att prestera proportionerligt till antalet förfrågningar oavsett antalet noder i nätverket.

3.3 Multihop routing

Många sökalgoritmer och tekniker finns för att hitta den kortaste vägen genom en graf från en nod till en annan. Vilken av dessa algoritmer som är mest effektiv beror på grafens utseende, egenskaper, och vilken information som finns tillgänglig som skulle kunna hjälpa till i sökningen. I fallet som studeras så kan det konstateras att grafen kan bestå av upp till 200 noder, och den genomsnittliga förgreningsfaktorn i grafen ligga mellan 0 och 199. Eftersom maximalt fyra reläer tillåts innan den sökta noden betraktas som omöjlig att nå, så kommer grafen kunna sägas ha en höjd på maximalt fyra.

För att hitta den kortaste vägen mellan två godtyckliga noder i den här grafen kommer så kommer vi titta på hur routing implementeras i andra sammanhang, och sedan kommer två olika alternativ till sökalgoritmer diskuteras.

Många av de traditionella routingalgoritmerna bygger på att beräkna den kortaste vägen till målet och sedan skicka kommunikationen genom den vägen. Biswas och Morris [1] beskriver en annan lösning som bygger på att noderna broadcastar paketen till alla sina grannar, och väljer efter att ha fått reda på vilka noder som tagit emot paketen en nod som ska sända paketen vidare på likartat sätt. Denna distribuerad lösning kan vara mycket krävande för simuleringen eftersom alla noder hade behövt broadcasta varje sekund till alla sina grannar.

En bredden först sökning (BFS) i en graf $G = (V, E)$ har tidskomplexiteten $\mathcal{O}(V + E)$, och minneskomplexiteten $\mathcal{O}(V)$. Minneskomplexiteten är i jämförelse med djupet först sökningar (DFS) högre, men till skillnad från DFS så är den optimal[3]. Det betyder att så fort en väg har hittats så kan det garanteras att den vägen är den kortaste vägen till destinationen. Algoritmens optimalitet är en försäkran på att inte radionätverket belastas mer än absolut nödvändigt. BFS är därvid mer lämplig i praktiken än DFS.

Information finns även tillgängligt som tillåter oss att beräkna det geografiska avståndet mellan två godtyckliga noder. Den informationen kan användas som en heuristik i en A* sökning, alltså ett sätt att guida sökningen i rätt riktning. I denna rapport så kommer vi att använda oss av just en A* sökning, som är optimal och komplett likt BFS, men eftersom den är guidad så mins-

kas generellt antalet utforskade noder.

3.4 Multivariabla egenskaper i förbindelserna

I vissa fall kan det vara intressant att undersöka om det finns en annan väg som har en bättre uppkoppling än den väg som hittades först. I praktiken kan man tänka sig att den faktiska vägen som väljes av algoritmen är en väg som inte bara *har* en uppkoppling, utan den väg som har den absolut bästa uppkopplingen. Det kan alltså finnas anledning att fortsätta leta efter vägar till destinationen även efter att vi redan hittat en väg som fungerar. För att hitta den väg med bäst uppkopplingskalité så hade alla noder i grafen behövts vandras för varje förfrågning som görs. Detta görs inte i denna rapport, utan lämnas till vidare arbete.

Vissa av förbindelsens egenskaper mellan två noder i nätverket beror på hela den gångna sträckan från startnoden. Dessa egenskaper, som hädanefter kallas multivariabla egenskaper, kan vara exempelvis paketförlust, som beror på paketförlusterna över alla förbindelser som vandrats över. Generellt kan den resulterande paketförlusten mellan nod A och nod X, Pf_{ax}, beräknas

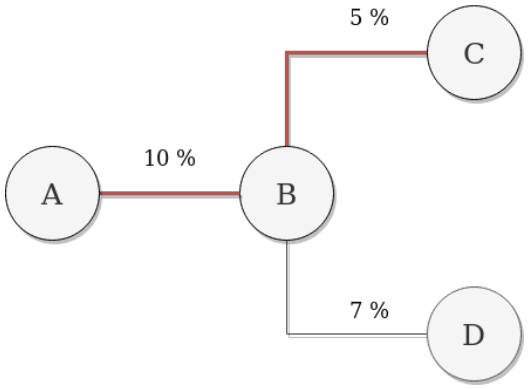
$$Pf_{ax} = 1 - (1 - Pf_{ab}) \times (1 - Pf_{bc}) \times \ldots \times (1 - Pf_{yx})$$

Figur 2 visar ett nätverk med paketförlusten över respektive förbindelse. Då vi vandrat från nod A till nod B så är paketförlusten 10%, och vid nod C så är förlusten 5%. Den slutgiltiga paketförlusten mellan nod A och nod C i figuren beräknas

$$Pf_{ac} = 1 - (1 - 0.10) \times (1 - 0.05) = 0.145 = 14.5\%$$

Samma princip gäller för bandbredden mellan nod A och X, Bb_{ax} . Samtliga bandbredder i vägen mellan nod A och nod X måste beaktas. Den resulterande bandbredden mellan nod A och X är den minsta av förbindelsernas bandbredder.

$$Bb_{ax} = min(B_{ab}, B_{bc}, \ldots, Bb_{yx})$$



Figur 4: Paketförlust över flera förbindelser i ett nätverk. Den totala paketförlusten över förbindelsen mellan A till C blir i detta exempel 14.5%.

3.5 Data Distribution Management

Många Distributed Interactive Simulation (DIS) system broadcastar alla uppdateringar till alla simulatorer i federationen vilket innebär att antalet kommunikationer som görs över RTI:n blir $\mathcal{O}(N^2)$. Bandbredden som krävs ökar även den orimligt mycket då N ökar. Många försök har gjorts för att attackera detta problem, och i stort sett alla har försökt minimera komplexiteten genom att bara skicka uppdateringar till de simulatorer som begär att de vill ha en uppdatering. En studie av detta gjordes av Morse et al. [9]. Detta kallas för data distribution management (DDM) [14].

4 Metod

4.1 Grafsökning

Fem javaklasser skapades

1. Main.java
   Innehåller ett huvudprogram. Programmet startas och ser till att grafen uppdateras så fort en ny inmatning kommer in från RTI:n.

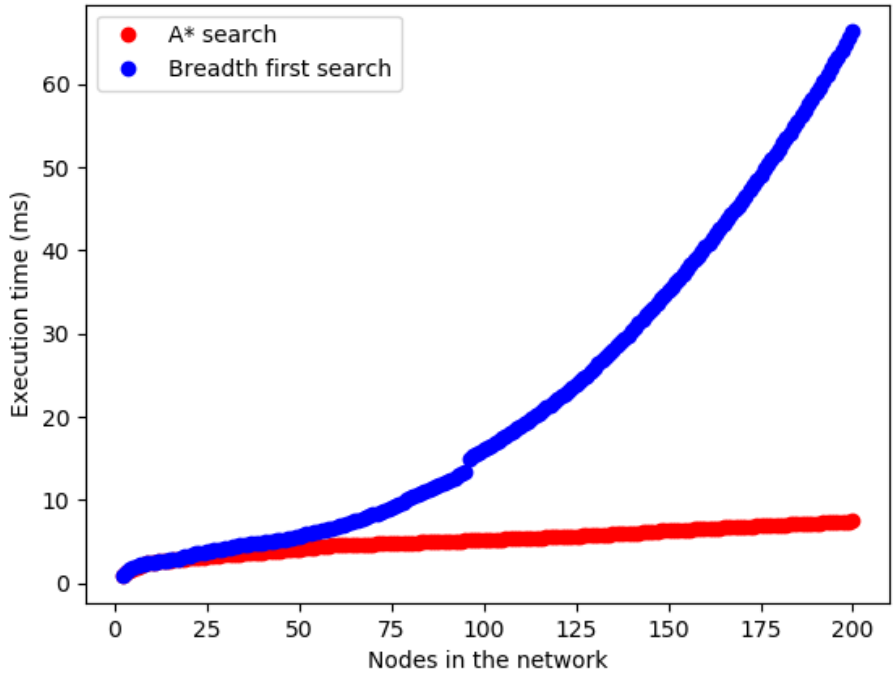
2. Graph.java
   Bygger upp grafen och utför en grafsökningen. Används av RTI_RW.java för att tillfredsställa förfrågningar.

3. Radio.java
   I denna klass är det definierat för varje typ av kommunikation vad som krävs av förbindelserna för att kommunikationen ska kunna föras över nätverket. Används av Graph.java för att bygga upp rätt graf för en viss typ av kommunikation.

4. RTI_RW.java
   Tar hand om ut- och inmatning till och från RTI:n. Används av Graph.java för att bygga upp grafen med hjälp av grannmatriser, och av DynamicQueue.java för att schemalägga förfrågningar.

5. DynamicQueue.java
   Schemalägger förfrågningar. Används av Graph.java för att hämta förfrågningarna.

Om en nod vill skicka sin geografiska position till en annan nod så behöver först säkerställas att latensen, paketförlusten, och bandbredden är tillräcklig mellan dessa noder för att skicka just denna information. För att göra detta så skapades två metoder i Radio.java: voiceCom() och dataCom(), som returnerar om det är möjligt att föra samtalskommunikation respektive datakommunikation över en viss förbindelse. Dessa funktioner används av Graph.java när grafen byggs upp för att exkludera de förbindelser som definitivt inte har tillräckliga nätverksresurser för den specificerade kommunikationstypen. Då grafsökningen utförs måste sökningen fortfarande ta hänsyn till förbindelsernas multivariabla egenskaper, beskrivet i teorikapitlet. En studie utfördes för att säkerställa att en A* sökning är bättre än en bredden först sökning även för just vårt fall där vi inte får hoppa längre än 4 steg från ursprungsnoden. Detta illustreras i Figur 5.



Figur 5: Exekveringstid för en A* sökning och en bredden först sökning som funktion av antalet noder i nätverket. A* är uppenbart överlägsen tidsmässigt -oberoende av antalet noder i nätverket.

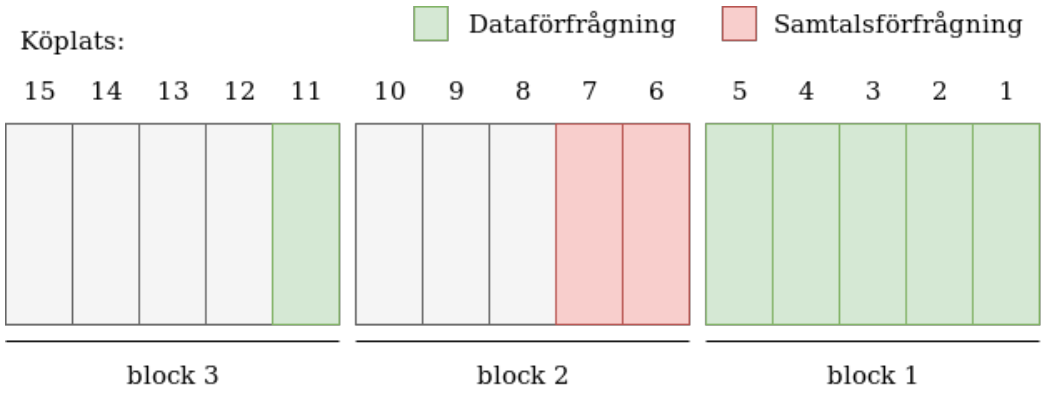
4.2 Centraliserad routing

Denna simulator implementeras med centraliserad routing. En server tillägnas att ta hand om att beräkna den kortaste rutten från alla noder till alla andra noder ungefär varje sekund. Centraliserad routing är väldigt behaglig att implementera i fallet som studeras här eftersom det inte kräver någon installation på andra maskiner i nätverket. Detta är förmånligt eftersom komponenterna som kör simulationen inte nödvändigtvis är lättillgängliga. De kan till exempel stå i något annat land eller kan av andra skäl inte uppdateras med ny mjukvara. Vi kan heller inte förutsätta att alla deltagande maskiner har tillräckligt bra hårdvara för att göra alla de tunga beräkningar som förväntas av dem. Vi har dessutom avgjort att en request-response lösning kommer att implementeras hellre än en brute-force. För att en federat ska

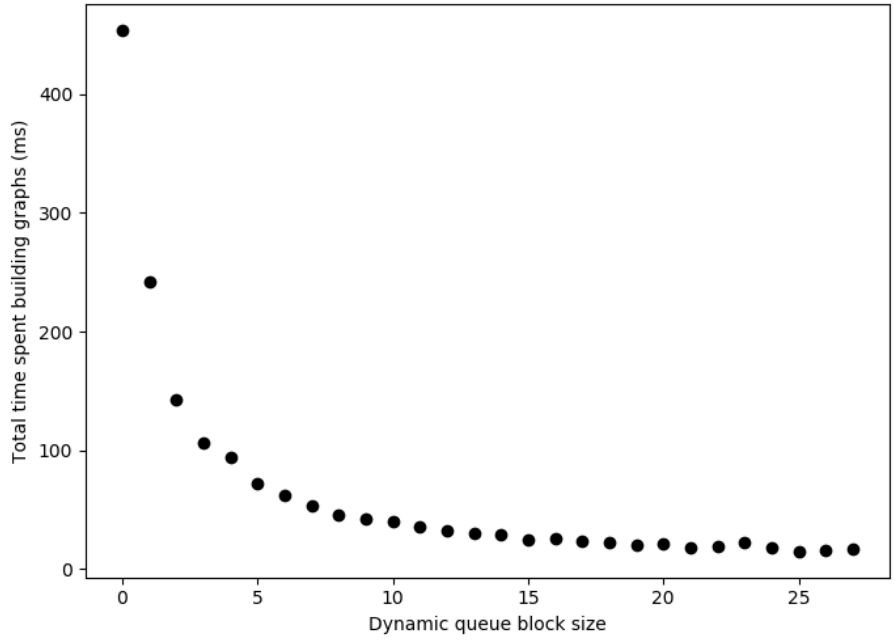
få reda på den kortaste vägen till en nod så kommer den att behöva ställa en fråga till servern, som sedan räknar ut den kortaste vägen, och slutligen skickar svaret tillbaka till frågeställaren.

Federationen kommer att göra förfrågningar av kortaste vägen från en viss nod till en annan nod i nätverket. Eftersom vilken graf som ska vandras beror både på det totala nätverkets egenskaper och på typen av kommunikation så behöver vi bygga om grafen varje gång antingen nätverket uppdateras eller en förfrågning om en ny typ av kommunikation kommer in. För att programmet ska vara så effektivt som möjligt så vill vi minimera antalet gånger grafen behöver byggas om. För att minimera problemet med att grafen behöver byggas om varje gång en förfrågning om en ny kommunikationstyp kommer in så kan en idé vara att dela upp dataförfrågningarna och samtalsförfrågningarna. Vi kan antingen tänka oss att vi varannan sekund tar hand om dataförfrågningar, och varannan samtalsförfrågningar. En annan idé kan vara att tråda programmet: en tråd tar hand om samtal, och en annan tar hand om data. Båda dessa lösningar är ineffektiva eftersom resurser alltid kommer att läggas på att ta hand om exempelvis dataförfrågningar även om inga sådana finns för tillfället. Implementationen i denna artikel använder en gemensam, dynamisk, kö för både samtal och data. I kön schemaläggs i största möjliga mån ett visst antal förfrågningar av samma typ i rad (i ett block) för att på så sätt öka tidseffektiviteten i programmet genom att minska antalet gånger som grafen måste byggas om. Figur 6 illustrerar denna kö med en blockstorlek av 5. Något som måste förhindras i denna implementation är så kallad resurssvält. Detta sker om block 2 aldrig blir servat för att block 1 fylls på med lika många förfrågningar som den tillfredsställer. För att förhindra detta så är det sagt att ett block inte ta emot fler än den antal förfrågningar som får plats i ett block.

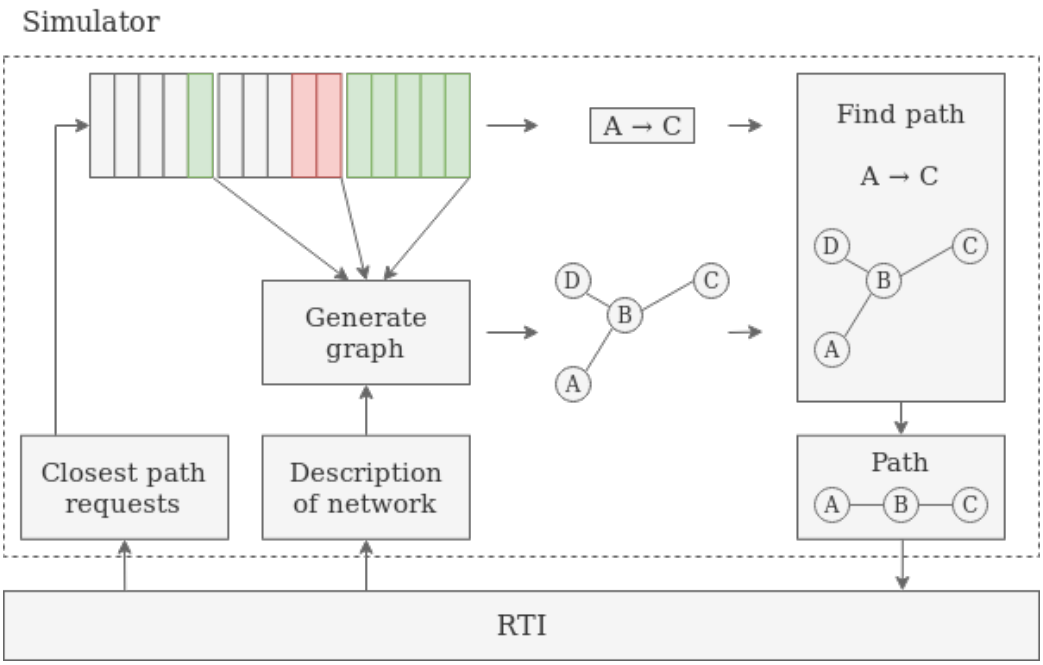
Figur 7 visar hur gruppstorleken i kön kan användas för att minimera tiden som programmets lägger på att bygga grafer. Figur 8 (som inte finns, men caption = Genomsnittlig fördröjning per förfrågning för 1000 förfrågningar (500 data, 500 samtal) som funktion av gruppstorlek”). Dessa två undersökningar användes för att komma fram till att en gruppstorlek av X förfrågningar i varje grupp maximerar programmets effektivitet samtidigt som fördröjningen för att en förfrågan besvaras hålls på acceptabel nivå. Figur 8 visar hur förhållandet mellan antalet data till samtalsförfrågningar påverkar programmets effektivitet.



Figur 6: Dynamisk kö för förfrågningar. Om nya samtalsförfrågningar kommer in i kön så kommer tre stycken sådana kunna läggas i block 2, alltså framför de förfrågningar som redan ligger i block 3.



Figur 7: Genomsnittlig tid programmet spenderade på att bygga upp grafer (1000 förfrågningar: 500 data och 500 samtal) som funktion av gruppstorlek i kön



Figur 8: Simulatorns fullständiga arkitektur.

Referenser

[1] Sanjit Biswas and Robert Morris. Exor: opportunistic multi-hop routing for wireless networks. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 133–144, 2005.

[2] Wentong Cai, Stephen John Turner, and Hanfeng Zhao. A load management system for running hla-based distributed simulations over the grid. In *Proceedings. Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, pages 7–14. IEEE, 2002.

[3] Charles E.; Rivest Ronald L.; Stein Clifford Cormen, Thomas H.; Leiserson. *Introduction to Algorithms (2nd ed.)*. MIT Press, 1990.

[4] Judith S Dahmann. The high level architecture and beyond: technology challenges. In *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation. PADS 99.(Cat. No. PR00155)*, pages 64–70. IEEE, 1999.

[5] Judith S Dahmann, Richard M Fujimoto, and Richard M Weatherly. The department of defense high level architecture. In *Proceedings of the 29th conference on Winter simulation*, pages 142–149, 1997.

[6] David B Johnson. Routing in ad hoc networks of mobile hosts. In *1994 First Workshop on Mobile Computing Systems and Applications*, pages 158–163. IEEE, 1994.

[7] Azman Osman Lim, Xudong Wang, Youiti Kado, and Bing Zhang. A hybrid centralized routing protocol for 802.11 s wmns. *Mobile Networks and Applications*, 13(1-2):117–131, 2008.

[8] Subhasree Mandal, Subbaiah Venkata, Leon Poutievski, Amit Gupta, Min Zhu, Rajiv Ramanathan, James M Wanderer, and Joon Ong. Semi-centralized routing, September 9 2014. US Patent 8,830,820.

[9] Katherine L Morse et al. *Interest management in large-scale distributed simulations*. Information and Computer Science, University of California, Irvine, 1996.

[10] Siva D Muruganathan, Daniel CF Ma, Rolly I Bhasin, and Abraham O Fapojuwo. A centralized energy-efficient routing protocol for wireless sensor networks. *IEEE Communications Magazine*, 43(3):S8–13, 2005.

[11] Charles E Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM computer communication review*, 24(4):234–244, 1994.

[12] Charles E Perkins and Elizabeth M Royer. Ad-hoc on-demand distance vector routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100. IEEE, 1999.

[13] Haldane Peterson, Soumya Sen, Jaideep Chandrashekar, Lixin Gao, Roch Guerin, and Zhi-Li Zhang. Message-efficient dissemination for loop-free centralized routing. *ACM SIGCOMM Computer Communication Review*, 38(3):63–74, 2008.

[14] Ivan Tacic and Richard M Fujimoto. Synchronized data distribution management in distributed simulations. In *Proceedings of the twelfth workshop on Parallel and distributed simulation*, pages 108–115, 1998.