Implementation av centraliserad Multihop Routing med High Level Architecture

En empirisk undersökning av kontextspecifika heuristiker för effektiv grafsökning

Lukas Pohlman
lukpo970@student.liu.se

17 februari 2022

Examinator: Jonas Wallgren
Handledare: John Tinnerholm

LINKÖPING UNIVERSITY

LIU-IDA/LITH-EX-G–21/015—SE

Upphovsrätt

Copyright

Abstrakt

I detta arbete har en trådad simulator tagits fram enligt standarden High Level Architecture (HLA). Simulatorn är kapabel att avgöra den kortaste vägen från alla noder till alla andra noder i ett radionätverk med 200 noder på i genomsnitt 263 millisekunder. Tidigare var det endast möjligt att simulera kommunikation mellan två noder i ett nätverk som hade direkt förbindelse med varandra. I och med detta tillägg kan kommunikationssignalen reläas fram genom nätverket om en direkt förbindelse inte är möjlig. Simulatorn, eller federatet som det kallas i HLA, bygger på en centraliserad routingalgoritm och kan konfigureras till att beräkna specifika vägar på begäran alternativt beräkna alla möjliga vägar genom nätverket utan att någon efterfrågan behövs. Simulatorn använder sig av en A*-algoritm som kan använda en av två heuristiker där den ena heuristiken tar fram den kortaste vägen mellan två noder i nätverket och den andra heuristiken tar fram den väg med bäst signalkvalitet mellan två noder.

Nyckelord: Distribuerade simuleringar, High Level Architecture, Routingalgoritmer, Grafsökning

Abstract

This paper presents a threaded simulator designed according to the standard High Level Architecture (HLA). The simulator is capable of determining the shortest path from all nodes to all other nodes in a radio network with 200 nodes in 263 milliseconds on average. It was previously only possible to simulate communication between two nodes which had direct connection. As of this addition, the communication can be relayed through other nodes in the network if direct connection is not possible. The simulator, or federate as it is called in HLA, implements a centralised routing algorithm and can be configured to find specific paths on the basis of requests alternatively find all paths through the network without the need for any request. The simulator uses an A* (A-star) algorithm which can use one of two heuristics, one of which returns the shortest path and the other returns the path with the best signal quality.

Keywords: Distributed simulations, High Level Architecture, Routing algorithms, Graphsearch

Författarens tack

Jag vill tacka min examinator Jonas Wallgren och min handledare John Tinnerholm på Linköpings Universitet för all hjälp med rapporten. Jag vill även tacka Pitch för möjligheten att utföra arbetet, och speciellt Thomas Brännström, som väglett, undervisat, och hjälpt mig med den teoretiska bakgrunden. Ett stort tack går även till Fredrik Adolfsson som hjälpte till med rapportens första utkast.

Akronymer

HLA High Level Architecture, en standard för distribuerade simuleringar.

Federat En applikation som via HLA kan delta i en simulering.

RTI Run Time Infrastructure, en servicebuss för kommunikation mellan federat.

Federation En samling av federat uppkopplade till en och samma RTI.

FOM Federation Object Module. En objektmodell som används för datautbytet mellan federat i en federation.

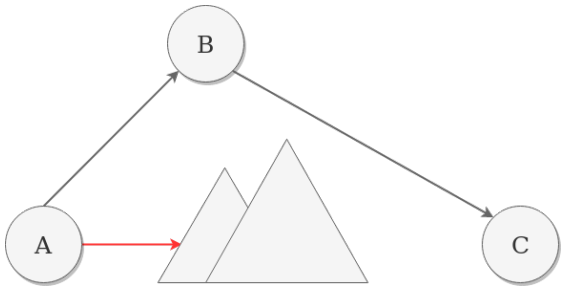
Innehåll

1 Inledning

Detta kapitel är uppdelat i fyra delar. Den första delen motiverar och beskriver bakgrunden, den andra delen beskriver syftet med arbetet, den tredje delen beskriver frågeställningarna, och den fjärde delen beskriver avgränsningarna.

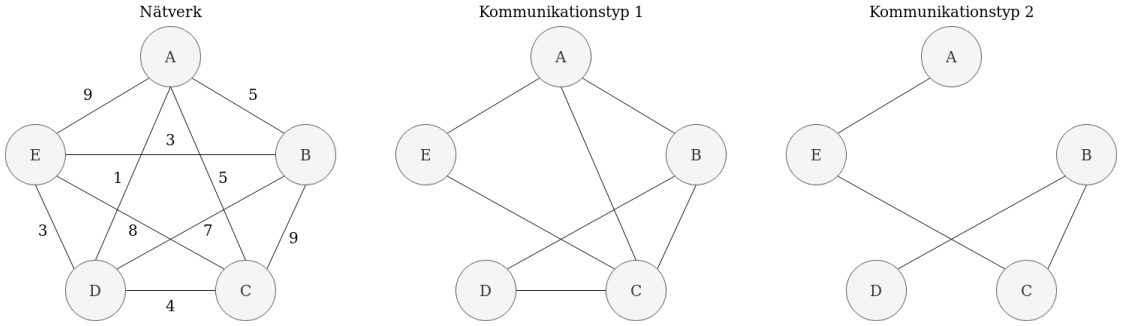
1.1 Motivering

Pitch Technologies AB (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 använder för expandera radioapparaternas räckvidd i nätverket. Detta möjliggörs genom att kommunikationssignalen skickas genom andra radioapparater fram till destinationen om direkt kommunikation mellan två radioapparater inte är möjlig enligt Figur 1. Den befintliga lösningen har endast stöd för att simulera kommunikationen mellan radioapparater med direkt förbindelse till varandra i nätverket. Om terräng eller avstånd hindrar signalen att nå mottagaren så kan de omöjligen 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 en väg 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 avgörs både vilken typ av kommunikation som ska föras över nätverket och förbindelsernas egenskaper om kommunikation mellan två noder är möjlig. Kommunikationen 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. Det innebär att i ett radionätverk med olika förbindelsekvaliteter kan vi extrahera olika grafer för varje kommunikationstyp som består av de förbindelserna som är av tillräcklig kvalitet för att kunna traverseras av just den kommunikationstypen. Detta illustreras i Figur 2.



Figur 2: Nätverket till vänster används för att bygga upp två grafer för kommunikationstyp 1 respektive 2. Graferna består endast av de bågar som har tillräckligt hög förbindelsekvalité i nätverket (se bågvikterna) för kommunkationstyperna att traversera.

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 att i framtiden integrera denna simulator med en större simulator som utvecklats av Pitch i uppdrag av den svenska militären.

1.3 Frågeställningar

Simulatorn kommer att hantera många beräkningar på kort tid. Det finns användningsfall med upp till 200 noder i nätverket, och varje nod kan potentiellt behöva veta en väg till alla andra noder i nätverket varje sekund. Behov föreligger att ta fram en simulator som kan utföra de beräkningar den tar emot snabbare än vad den får nya. Simulatorn skall vara designad och implementerad enligt standarden High Level Architecture (HLA).

1. Är en centraliserad routingalgoritm tillräckligt effektiv för att utföra en uttömmande sökning av alla vägar, det vill säga vägen mellan alla par av noder, eller måste den arbeta på efterfrågan om en väg? Hur skalbar är en uttömmande sökning?

2. I vilken utsträckning kan trådning användas för att minska exekveringstiden hos den centraliserade routingalgoritmen?

3. Vilken i kontexten lämplig heuristik bör användas i en A*-algoritm för att hitta vägen mellan två noder i nätverket som söker kommunikation med varandra? Först och främst söks den kortaste vägen mellan två noder. Om algoritmen är tillräckligt effektiv så kan det även finnas intresse av att hitta den väg som resulterar i bäst förbindelsekvalitet. Hur presterar algoritmerna som hittar dessa två typer av vägar i jämförelse med varandra?

1.4 Avgränsningar

1. Eftersom radiokommunikationens kvalitet avtar mellan varje hopp så är det sagt att antalet hopp från den ursprungliga noden begränsas till max fyra. Det innebär att kraven på simulatorns prestanda är lägre eftersom trädet som måste traverseras garanterat aldrig har en höjd större än fyra.

2. A*-sökningen som i denna rapport hittar den kortaste vägen mellan två noder i nätverket använder sig av en heuristik för att guida sökningen. Endast två heuristiker kommer att undersökas i detta syfte. De två heuristikerna bygger på avståndet mellan en nod och destinationsnoden respektive på en variabel som definierar uppkopplingskvaliteten mellan en nod och destinationsnoden.

3. I detta arbete implementeras endast en centraliserad routingalgoritm. Den distribuerade motsvarigheten lämnas till vidare arbete.

4. Floyd Warshalls algoritm [6] hittar alla kortaste vägar mellan alla par av noder i en graf, men den tar inte hänsyn till antalet hopp som gjorts från ursprungsnoden och den sparar inte vägen som traverserats. Dessa problem skulle möjligen kunna lösas genom modifikationer av algoritmen, men den är oavsett inte kompatibel med en request-responselösning och implementeras därför inte i detta arbete.

2 Bakgrund

I detta kapitel ges bakgrunden för rapporten fördelat över tre avsnitt vari uppdragsgivaren, standarden High Level Architecture, och simulatorn utvecklad av Pitch presenteras.

2.1 Pitch Technologies

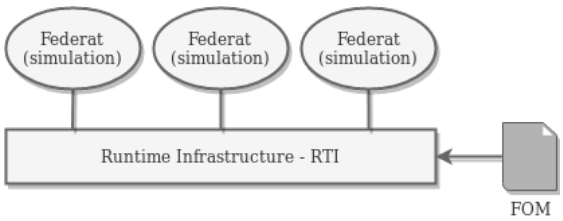
Pitch Technologies (Pitch) grundades 1991 och är en ledande leverantör av produkter, tjänster och lösningar som är kompatibla med öppna standarder[1] för interoperabla distribuerade simuleringssystem. Deras produkter och lösningar används av några av de största och mest komplexa systemutvecklings-, integrations-, test-, tränings- och utbildningsleveransprogrammen inom ett globalt sortiment av sektorer inklusive försvar, flygkontroll, luftfartskontroll, säkerhet, transport, energi och medicin.

2.2 Simulatorn

Pitch har utvecklat en simulator till en ledningsträningsanläggning (LTA) som stöd i träningen av bataljonsstaber. LTA:n bidrar med möjligheten att träna grundläggande stridssituationer, orderträna, och vidmakthålla färdigheter. I LTA:n skapas en virtuell miljö så lik verkligheten som möjligt för att kunna öva staber i ledning av förband på ett så realistiskt sätt som möjligt. Anläggningen erbjuder även möjligheten att öva chefer på kompani- och plutonsnivå, samt för gruppnivå och för olika specifika funktioner som logistik, sjukvårdstjänst, indirekt bekämpning med mera.

2.3 High Level Architecture

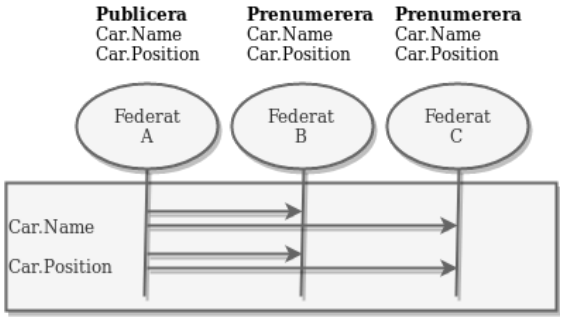
Simulatorn är designad i enlighet med standarden High Level Architecture (HLA)[4]. HLA är ett kraftfullt verktyg som utvecklades år 1990 för distribuerad simulering. Syftet är interoperabilitet: att tillåta olika simuleringssystem att kommunicera och arbeta tillsammans, och återanvändning: att modeller lätt kan användas i nya kombinationer för att skapa nya simuleringar. Med hjälp av en sådan arkitektur blir det enkelt att koppla ihop många små federat 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 ett federat måste ha för att kunna interagera med RTI:n. RTI:n förser federaten med information samt koordinerar och synkroniserar alla komponenter. En Federation Object Model, FOM, beskriver datautbytet i federationen och kan ses som federationens språk.



Figur 3: Topologi i High Level Architecture. [13]

[1] En öppen standard är en standard som tillåts att implementeras av vem som helst utan hinder från ägaren. (https://open-stand.org/about-us/principles/, september 2020)

Ett federat kan välja att publicera viss information och prenumerera på viss information enligt Figur 4. Att publicera innebär att federatet skickar ut information via RTI:n, och att prenumerera innebär att federatet hämtar upp information som andra federat har publicerat. På så vis sker kommunikationen mellan federaten i HLA.



Figur 4: Publikation och prenumeration i HLA. Tre federat simulerar en bil var. Federat A publicerar sitt namn och sin position som de andra federaten, B och C, tar del av genom prenumeration. [13]

Dahmann [3] 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.

3 Teori

Radionätverket 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. I nätverket är det möjligt att kommunikation endast kan föras åt det ena hållet över en viss förbindelse. Grafen är med andra ord riktad vilket innebär att den kortaste vägen från nod A till nod B inte nödvändigtvis är densamma som från nod B till nod A. 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. Det är önskvärt att lösningen ska kunna beräkna den kortaste vägen från alla noder till alla andra noder på under en sekund. Med upp till 200 noder i nätverket kan det handla om 39800 kortaste vägar som ska beräknas varje sekund. Olika lösningar studeras för att tillfredsställa detta krav. I detta kapitel förs en diskussion om hur valet av centraliserade eller distribuerade sökalgoritmer påverkar simuleringens prestanda. Därefter redogörs för vilken algoritm som är mest passande och sedan hur distribuerade simuleringar har implementerats tidigare.

3.1 Centraliserade och distribuerade routingalgoritmer

Många routingalgorimer bygger på en distribuerad arkitektur [7, 14, 15]. Dessa arkitekturer kräver ingen central maskin som utför beräkningar åt dem, dessa presterar bra och är skalbara. Andra är centraliserade eller semicentraliserade [9, 10, 12] och bygger på en server (eller en mindre mängd servrar) som beräknar alla kortaste vägar i nätverket. Servern tillhandahåller information om nätverkets tillstånd och kräver att mycket kommunikation förs mellan noderna och servern för att servern ska veta hur den bör dirigera.

Klein et al. [8] simulerar fordonstrafik och argumenterar för att modulära simuleringsmodeller är lättare att utveckla, testa och underhålla än monolitiska modeller. Samma sak gäller när det kommer till att centralisera eller distribuera sökalgoritmen. Det finns däremot ingenting som talar för att en centraliserad modell inte är möjlig att utveckla. Peterson et al. [16] argumenterar till och med för att ett nytt intresse för 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 dirigeras 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 den centraliserade servern vill beräkna den kortaste vägen från alla noder till alla andra noder i nätverket så blir antalet kortaste vägar som behöver beräknas med N noder i nätverket $\mathcal{O}(N^2)$. Om istället alla noder i nätverket själva hade beräknat den kortaste vägen till alla andra noder så hade antalet kortaste vägar som beräknats per maskin varit $\mathcal{O}(N)$. Den centraliserade arkitekturen är med andra ord mindre skalbar, 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 backuper 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 simuleringen. Till exempel så kan det finnas passiva observatörer och simuleringssurrogater. Cai et al. [2] beskriver ett lasthanteringssystem för att även distribuera lasten över de maskiner som inte är aktivt deltagande i simuleringar, 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 någon bristvara för Pitch så är en sådan åtgärd inte nödvändig i nuläget.

3.2 Uttömmande sökning och request-response

Beräkningen av en kortaste väg kan antingen göras då den efterfrågas av en nod i nätverket (request-response), eller så kan alla kortaste vägar beräknas proaktivt (en uttömmande sökning). En request-response lösning kommer att prestera proportionellt mot antalet förfrågningar som skickas oavsett antalet noder i nätverket och oavsett hur många kommunikationstyper som finns. Eftersom den uttömmande sökningen behöver beräkna alla kortaste vägar i alla grafer som finns (en graf per kommunikationstyp) så kommer antalet förfrågningar som behöver beräknas proaktivt vara proportionell mot antalet kommunikationstyper och antal noder som nätverket innehåller. Det kan finnas noder som bara kommer användas för att reläa andra noders kommunikation och som på förhand är känt aldrig kommer att skicka eller ta emot kommunikation. Den uttömmande sökningen kan minska antalet kortaste vägar som behöver beräknas genom att helt enkelt inte ta med dessa noder som avsändare och/eller mottagare. Om $f(x)$ är antalet kortaste vägar som måste beräknas med lösning x, så kan f(uttömmande sökning) och f(request-response) beräknas:

$$f(uttömmande\ sökning) = N \times (N - 1 - F) \times K \tag{1}$$

$$f(request - response) = R \tag{2}$$

N noder i nätverket
F federat som inte skickar förfrågningar
K kommunikationstyper
R förfrågningar

Den uttömmande sökningen har fördelen att ingen implementation krävs i de andra federaten. Alla kortaste vägar kommer att beräknas och publiceras utan att någon förfrågan behöver skickas. Om fler kommunikationstyper introduceras försämras däremot prestandan väsentligt. Generellt kan sägas att request-response är en mer skalbar lösning som är mindre CPU-krävande, medan en uttömmande sökning är enklare att implementera och kan även prestera likvärdigt förutsatt att färre kommunikationstyper finns i nätverket.

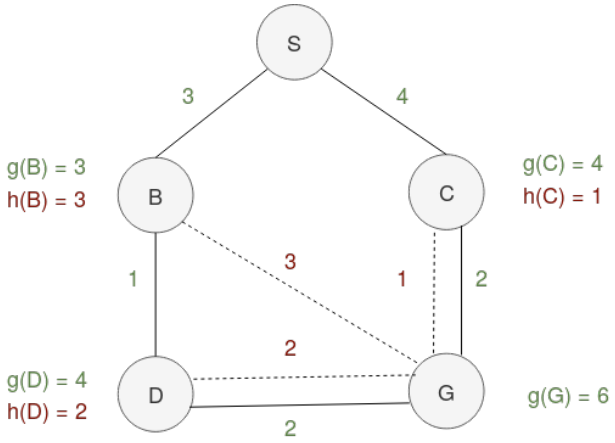
3.3 Multihop routing

Vilken algoritm som är mest effektiv för att hitta den kortaste vägen beror på grafens struktur och vilken information som finns tillgänglig som skulle kunna underlätta sökningen. I detta arbete definieras den kortaste vägen som den väg mellan två noder som traverserar minst antal bågar. Eftersom grafen som ska traverseras redan består av den delmängd av nätverkets bågar som kommunikationen faktiskt kan föra över, så behöver inte sökalgoritmen ta hänsyn till bågvikterna i syfte att undersöka om en viss förbindelse är av tillräcklig kvalitet för att traverseras. Det kan däremot finnas anledning att använda bågvikterna för att prioritera vissa vägar framför andra. I fallet som studeras så kan det konstateras att grafen består av max 200 noder, den kan innehålla loopar, och förgreningsfaktorn i grafen ligger mellan 0 och 199. Eftersom maximalt tre 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 så kommer undersökning göras av hur routing implementeras i andra sammanhang, och sedan kommer två olika alternativ till heuristiker 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 distribuerade lösning kan vara mycket krävande för simuleringen eftersom alla noder hade behövt broadcasta varje sekund till alla sina grannar.

För att effektivisera grafsökningen så kan istället en guidad grafsökning göras, till exempel med en A* (A-stjärna) sökning. I en A*-sökning så har varje nod N i grafen som ska traverseras ett värde $g(N)$ som anger den minsta kostnaden att ta sig till denna nod från en startnod S. Nod D i Figur 5 har till exempel $g(D) = 4$ eftersom den billigaste vägen är S, B, D, och bågarna mellan dessa noder har den totala kostnaden 4. Varje nod har även ett värde $h(N)$ som kallas för en heuristik och används för att guida sökningen mot destinationsnoden G. I figuren representeras heuristiken av de streckade linjerna. Heuristiken är ett relation mellan en godtycklig nod och destinationsnoden. Detta kan till exempel handla om avståndet till destinationen. A* grundar sig på att minimera $f(N) = g(N) + h(N)$. Nästa nod som besöks är alltså den nod som ger minst värde på $f(N)$. Från startnoden S i Figur 5 har vi två alternativ: besöka nod B eller nod C. $f(B) = 3 + 3 = 6$, och $f(C) = 4 + 1 = 5$. $f(C)$ är mindre än $f(B)$, så nod C besöks. Från nod C kan vi besöka destinationsnoden direkt med $f(G) = 6 + 0 = 6$, eller så kan vi besöka nod D med kostnad $f(D) = 4 + 2 = 6$. Eftersom värdet av $f(G)$ i detta fallet var mindre än eller lika med än alla andra alternativ så kan det garanteras att S, C, G är den billigaste vägen från S till G.

Guidade sökningar är generellt sett mer effektiva eftersom antalet besökta noder i regel är färre. Vilken heuristik som presterar bäst i detta fall är ännu okänt och måste undersökas. Det finns exempelvis information tillgänglig som tillåter oss att beräkna det geografiska avståndet mellan två godtyckliga noder i nätverket. Denna information kan användas som en heuristik. Det går även att använda förbindelsekvaliteten mellan två noder som en heuristik i sökningen. Ett tredje alternativ som kan vara intressant att un-



Figur 5: Vad en A*-algoritm ser när den traverserar en graf. De heldragna linjerna är de bågar som kan traverseras av algoritmen medan de streckade linjerna symboliserar heuristiken $h(N)$ mellan noderna och destinationsnoden som guidar sökningen. Värdena över respektive båge representerar bågarnas kostnad.

dersöka är möjligheten att kombinera olika heuristiker till en enda heuristik alternativt att ha en dynamisk heuristik som ändras beroende på grafens struktur.

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 leta efter vägar baserat på andra kriterier än kortaste vägen.

3.4 Multivariata egenskaper i förbindelserna

Vissa av förbindelsens egenskaper mellan två noder i nätverket beror på hela sträckan från startnoden. Dessa egenskaper, som hädanefter kallas multivariata egenskaper, kan vara exempelvis paketförlust. Paketförlust kan ske över alla förbindelser som traverseras. Den resulterande paketförlusten Pf_e i en väg X genom grafen $G = (V(G), E(G))$, beräknas därför:

$$1 - \prod_{e \in E(X)} (1 - Pf_e) \tag{3}$$

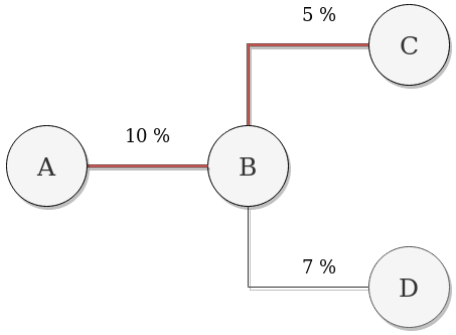
Figur 6 visar ett nätverk med paketförlusten över respektive förbindelse. Då vi vandrat från nod A till nod B så är paketförlusten 10%, och vid nod C så är förlusten 5%. Den slutgiltiga paketförlusten mellan nod A och nod C i figuren beräknas:

$$f_{ac} = 1 - (1 - 0.10) \times (1 - 0.05) = 0.145 = 14.5\% \tag{4}$$

Samma princip gäller för bandbredden mellan nod A och nod B, Bb_{ab}. Samtliga bandbredder i vägen mellan nod A och nod B måste beaktas. Den resulterande bandbredden

mellan nod A och nod B är den minsta av förbindelsernas bandbredder:

$$Bb_{AB} = min(Bb_{AC}, Bb_{CD}, \ldots, Bb_{YB}) \tag{5}$$



Figur 6: Paketförlust över förbindelserna i en graf. Den totala paketförlusten i kommunikationen mellan A och C blir 14.5%.

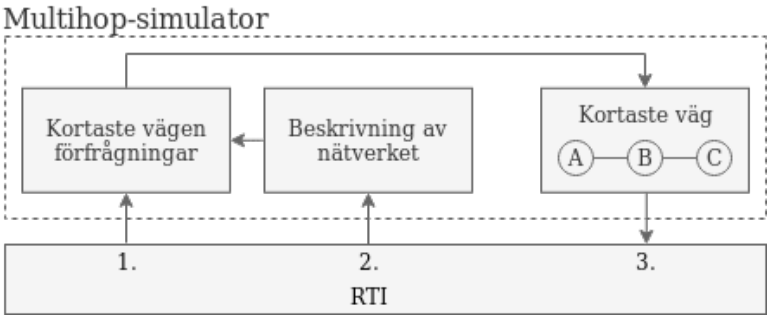
3.5 Data Distribution Management

Många distribuerade interaktiva simuleringssystem (DIS-system) sänder alla uppdateringar till alla federat vilket innebär att antalet uppdateringar som sänds över RTI:n med N federat blir N^2. Även bandbredden som krävs för att skicka uppdateringarna ökar proportionellt mot N^2. Många försök har gjorts för att lösa detta problem, och i stort sett alla har försökt minimera komplexiteten genom att bara skicka uppdateringar till de simulatorer som begär en uppdatering. En studie av detta gjordes av Morse et al. [11]. Tekniken kallas för data distribution management (DDM) [17] och kan användas av den uttömmande sökningen som förklarades i Avsnitt 3.2. Istället för att alla potentiellt sett 39800 kortaste vägar skickas till alla noder så skickas istället endast de vägar med n1 som startnod till n1, och endast de vägar med n2 som startnod till n2, och så vidare.

4 Metod

Detta kapitel består av en implementationsdel och en experimentbeskrivning. I implementationsdelen beskrivs vilka val som gjordes med avseende på simulatorns funktionalitet och hur implementationen gick till. I experimentdelen beskrivs hur en federation skapades i syfte att testa och optimera multihop-simulatorns inställningar.

4.1 Implementation



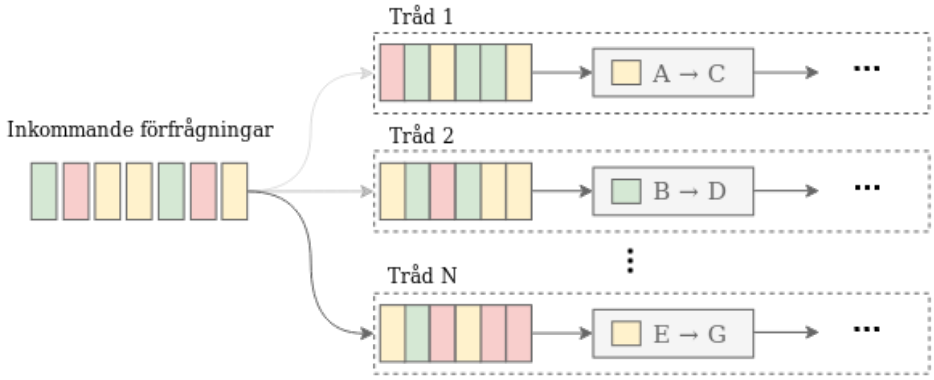
Figur 7: En modell över hur kommunikationens ingångar, 1. och 2., samt utgångar, 3., ser ut från multihop-simulatorn till RTI:n.

Multihop-simulatorn implementerades med en centraliserad routingalgoritm. Centraliserad routing är smidigt att implementera i fallet som studeras eftersom det inte kräver någon tillgång till, eller installation på, andra maskiner i federationen. Detta är speciellt önskvärt eftersom komponenterna som kör simuleringarna inte alltid är lättillgängliga. De kan till exempel stå i ett annat land eller kan av andra skäl inte kunna uppdateras med ny mjukvara. Det kan inte heller förutsättas att alla deltagande federat har tillräckligt bra hårdvara för att göra alla de tunga beräkningar som krävs.

Routingalgoritmen bygger på en uttömmande sökning men har även stöd för att fungera enligt request-response. Enligt Figur 7 kommer beskrivningen av nätverket in genom ingång 2, varpå det i den uttömmande sökningen genereras kortaste-vägen förfrågningar av alla typer. Det innebär att federaten som vill veta de kortaste vägarna inte nödvändigtvis behöver konfigureras till att skicka förfrågningar. Om det i framtiden skulle visa sig att antalet noder eller antalet typer av kommunikation som existerar i federationen blir många fler, och den uttömmande sökningen som följd blir ineffektiv, så kan den uttömmande sökningen slås av. Ingång 1 används då istället för att ta emot förfrågningar istället för att de genereras. Oavsett om förfrågningarna genereras eller ej, så hanteras de förfrågningar som finns av multihop-simulatorn som sedan publicerar resultaten till RTI:n. När den uttömmande sökningen har hanterat alla förfrågningar och nätverket har uppdaterats så utförs alla beräkningar på nytt förutsatt att nätverket har uppdaterats sedan senaste gången beräkningarna gjordes.

För att effektivisera lösningen parallelliseras den över flera trådar. Varje tråd har en egen kö med förfrågningar. Förfrågningarna som kommer in till multihop-simulatorn fördelas lika över dessa köer. Den första förfrågningen som kommer in ges till tråd 1, den andra till tråd 2, och så vidare. Efter tråd N tilldelats en förfrågning återupprepas pro-

cessen och nästkommande förfrågning ges till tråd 1. Genom att inte ha en gemensam kö som behandlas av alla trådar så blockeras inte trådarna av varandra i en synkronisering.



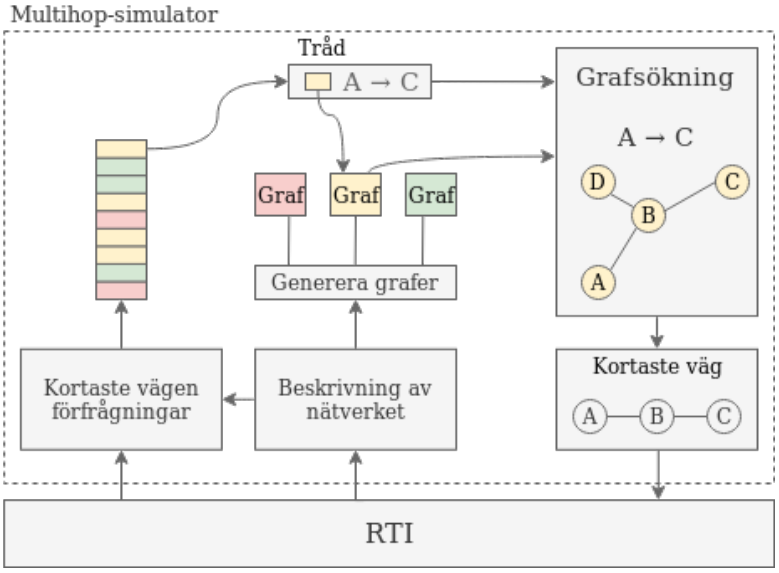
Figur 8: Struktur för att hantera förfrågningar. Alla inkommande förfrågningar placeras i en gemensam kö men behandlas separat i varje tråd.

Då en tråd hämtat en förfrågning ur kön, så betraktas förfrågningens kommunikationstyp för att avgöra vilken graf det är som ska sökas. Om en förfrågning kommer in med kommunikationstyp A, så behöver det alltså finnas en graf för kommunikationstyp A som består av de förbindelser i nätverket som kan traverseras av kommunikationstypen i fråga. Därför genereras en graf för varje kommunikationstyp av en modul i simulatorn så att de finns tillgångliga direkt då de behövs av trådarna. Så fort en ny nätverksbeskrivning kommer in så motsvarar graferna som byggts upp inte längre verkligheten, och alltså måste graferna genereras om så fort en ny beskrivning av nätverket tillhandahålls från RTI:n. Figur 9 visar programflödet från en enskild tråds perspektiv. När tråden har funnit ett svar, om ett sådant finns, så skickas den kortaste vägen ut till RTI:n, varpå en ny förfrågning hämtas från kön och process återupprepas. När kön är tom så väntar trådarna på att nya förfrågningar ska komma in till kön.

Endast graferna är delade resurser i arkitekturen, och eftersom de inte modifieras av trådarna så behöver de inte synkroniseras.

4.1.1 Val av heuristiker

En undersökning gjordes för att jämföra hur en A*-sökning med olika heuristiker presterade. Hur denna jämförelse genomfördes finns presenterat i Avsnitt 4.2.2. För att få fram heuristiker så söktes Pitch befintliga simulator igenom efter information som skulle kunna användas för att definiera heuristikerna. Det första som hittades var varje nods position. Genom positionsvariabeln och avståndsformeln kan ett avstånd erhållas mellan alla nodpar. Det betyder att en möjlig heuristik är baserad på avståndet till destinationsnoden. Den andra som hittades var förbindelsekvaliteten mellan varje par av noder i nätverket som ett flyttalsvärde mellan 0 och 11, där ett ökande värde innebär sämre förbindelsekvalitet, och ett värde över 7 kan tolkas som ingen framkomlighet alls. Förbindelsekvaliteten utgör den andra heuristiken som jämförs.



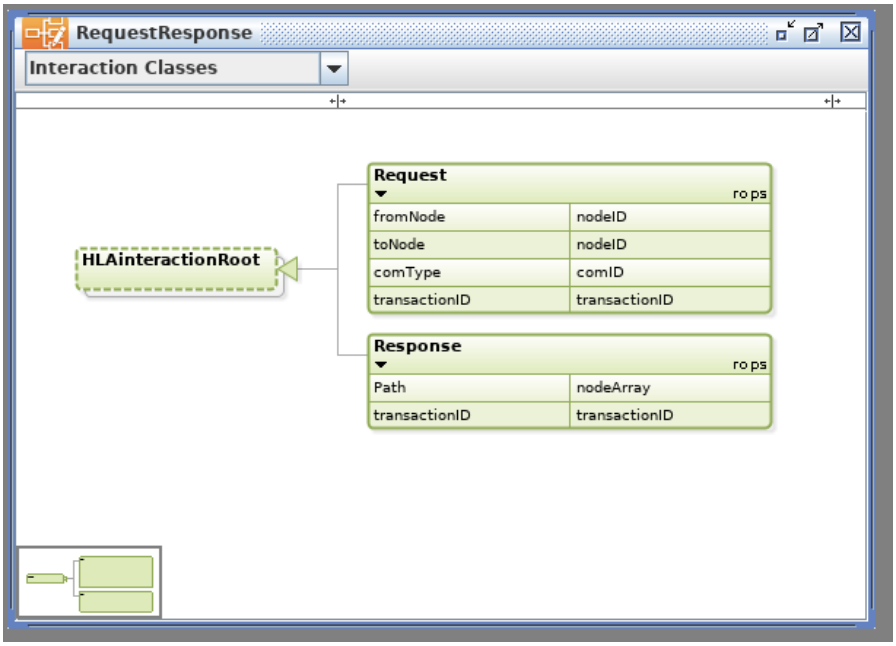
Figur 9: Multihop-simulatorns arkitektur där en tråd har plockat ut en förfrågning ur sin kö. Vilken graf som ska sökas är beroende av vilken kommunikationstyp förfrågningen har.

4.1.2 Program till federat

För att programmet som utvecklats enligt Avsnitt 4.1.1 ska bli ett federat i en federationen så krävs ytterligare åtgärder. Speciellt krävs tre steg:

1. Starta en RTI genom att köra programmet Pitch pRTI.
2. Utveckla en FOM-modul i Pitch Visual OMT.
3. Generera mallkod för ett federat i Pitch Developer Studio och integrera implementationen i denna mall.

FOM-modulen tillåter kommunikation med resten av federationen genom att definiera datautbytet som skickas till och från multihop-simulatorn. Som figur 7 visade så finns tre in- och utgångar vilka alla behöver vara definierade. Därför skapades först en FOM-modul för en förfrågning som tas emot genom ingång 1. I samma modul kan även definieras hur ett svar ser ut som skickas genom utgång 3. Figur 10 visar en skärmbild från programmet Pitch Visual OMT. I figuren illustreras hur en *Request* respektive en *Response* definieras. I en *Request* specificeras en ursprungsnod *fromNode*, en destinationsnod *toNode*, en kommunikationstyp *ComType*, samt ett transaktionsid *transactionID* - ett unikt värde som identifierar förfrågan och som skickas tillbaka i svaret. På så sätt kan avsändaren veta att meddelandet är till för just denne. En *Response* består av transaktionsid:et såväl som en lista av noder som utgör den kortaste vägen mellan ursprungsnod och destinationsnod om en sådan finns. Om ingen väg hittas genom grafen mellan de två noderna så är fältet *nodeArray* istället tomt. Beskrivningen av nätverket, som är en ingång till multihop-simulatorn, är redan definierad av det federat som publicerade beskrivningen, och behöver alltså inte omdefinieras.



Figur 10: Skärmbild från Pitch Visual OMT. I programmet är det möjligt att skapa nya FOM-moduler som definierar datautbytet som ska ske mellan federaten.

När alla FOM-moduler som behövs av federatet är skapade så importeras modulerna i Pitch Developer Studio som används för att generera mallkod för ett federat. Mallkoden som genereras utgör själva federatet som har stöd för att koppla upp sig mot en RTI samt ta emot och skicka information till RTI:n. Federatet har däremot ännu inget faktiskt beteende. Först efter att det utvecklade programmet har implementerats och integrerats i denna mallkod så är federatet färdigt och redo för att köras i federationen.

4.2 Experimentbeskrivning

Detta avsnitt redogör för hur resultaten i Kapitel 5 erhölls genom en beskrivning av hur testerna av federatet utfördes. Först beskrivs en experimentell kontext där hårdvara och mjukvara som använts presenteras. I samma avsnitt beskrivs även den federationsarkitektur som användes i testningen. Därefter kommer ett avsnitt om hur heuristikerna jämfördes följt av ett avsnitt om hur trådningen undersöktes.

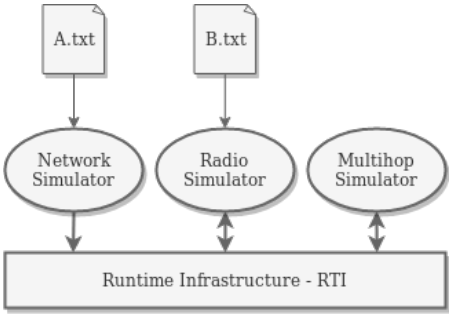
4.2.1 Experimentell kontext

För att testa multihopsimulatorn utvecklades en testmiljö som reflekterade verkliga scenarion där multihopsimulatorns prestanda kunde undersökas. Specifikationer för hård- och mjukvara som användes i testmiljön är redovisat i Tabell 1. För detta behövdes speciellt två andra federat skapas. Ett första federat publicerar information om nätverkets utseende, vilket inkluderar vilka noder som finns, deras positioner, och förbindelsekvaliteten mellan varje par av noder. Ett andra federat publicerar förfrågningar om kortaste vägen. Arkitekturen illustreras i Figur 11. Informationen som de två federaten publicerar styrs med hjälp av textfilerna A.txt och B.txt. Detta gjorde det enkelt att designa olika nätverksstrukturer och bestämma precis vilka förfrågningar som skulle skickas till multihopsimulatorn. Textfilen A.txt innehöll för testerna som gjordes i detta avsnitt en beskrivning av ett nätverk med 200 noder. Vidare så aktiverades den uttömmande

Hårdvara		Mjukvara	
CPUs:	8	Java version:	13.0.2
Threads per core:	2	Java SE JRE:	13.0.2+8 (64-bit)
Cores per socket:	4	OS:	Debian GNU/Linux 10
Sockets:	1		
Model:	Intel Core i7-4770		
CPU op-mode:	64 bit		

Tabell 1: Hård- och mjukvaruspecifikationer för maskinen som testerna kördes på.

sökningen. Detta resulterar i att då denna federation körs igång så kommer multihopsimulatorn att generera 39800 förfrågningar om vägar genom grafen - en väg mellan varje par av noder - så fort den har fått nätverksbeskrivningen av nätverkssimulatorn. På så sätt kan olika konfigurationer testas i multihopsimulatorn med samma nätverk och samma förfrågningar för varje körning.



Figur 11: Testmiljön som används för att testa multihopsimulatorns beteende i skarpt läge. Textfilen A.txt beskriver nätverket och textfilen B.txt specificerar vilka förfrågningar som ska skickas. Nätverkssimulatorn läser A.txt och publicerar beskrivningen. Radiosimulatorn läser B.txt och skickar förfrågningar.

4.2.2 Heuristiker

Federationen som utvecklades i Avsnitt 4.2.1 kördes 100 gånger för varje heuristik. Den genomsnittliga exekveringstiden mättes för dessa 100 körningar där varje exekveringstid syftar på den tid som körningarna spenderade i grafsökningen. För att få ett bättre perspektiv över hur distansheuristiken och signalkvalitetesheuristiken jämförde sig med varandra så utvecklades en tredje heuristik som valde väg genom grafen baserat på slumpen. Distansheuristikens exekveringstid utgör ett referensvärde för vad som är en låg exekveringstid, och exekveringstiden med heuristiken som väljer väg baserat på slumpen utgör referensvärde för vad som är en hög exekveringstid. Exekveringstiden av signalkvalitetsheuristiken kan på så sätt placeras på en skala och jämföras med vad som i kontexten anses vara höga och låga exekveringstider.

4.2.3 Trådning

För effektivisera programmet ytterligare undersöktes hur presentandan kunde förbättras genom multitrådning. Federationen som utvecklades i Avsnitt 4.2.1 kördes 100 gånger per test, och varje test använde mellan 1 och 10 trådar. En A*-algoritm med distansheuristik användes för samtliga tester.

5 Resultat

Detta kapitel består av två avsnitt som redogör för hur heuristikerna presterar i jämförelse med varandra samt hur trådning kunde användas för att förbättra federatets prestanda.

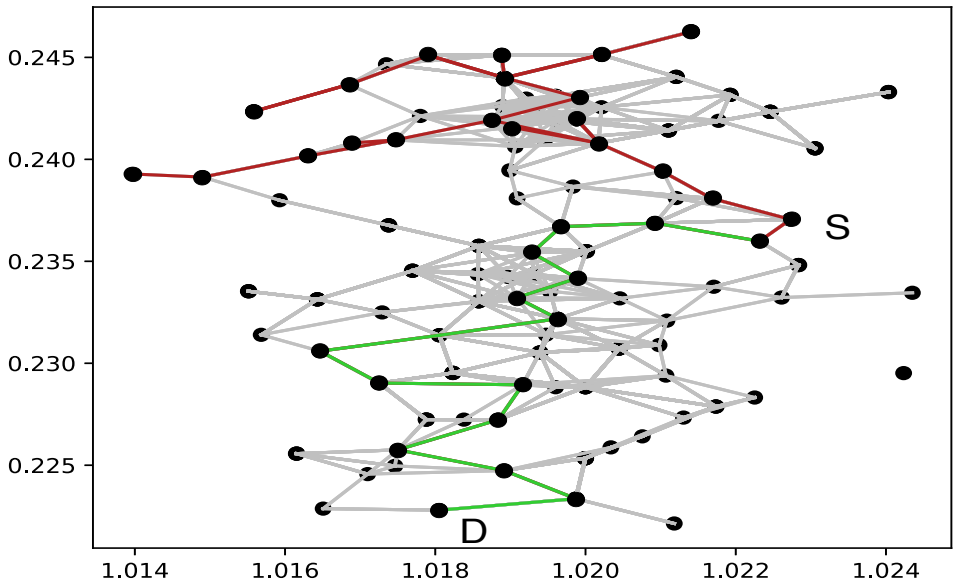
5.1 Heuristiker

Tabell 2 visar hur olika heuristiker jämför sig med varandra. Exekveringstiden syftar på den tid som programmet spenderade enbart på grafsökningarna. Den blinda heuristiken valde nästa nod att besöka baserat på slumpen, och syftar till att användas som en referens att jämföra de andra heuristikerna med. I jämförelsen tilläts ett obegränsat antal hopp från ursprungsnoden istället för maximalt fyra för att exponera deras olikheter. Den procentuella differensen är beräknad med distansen som referensvärde.

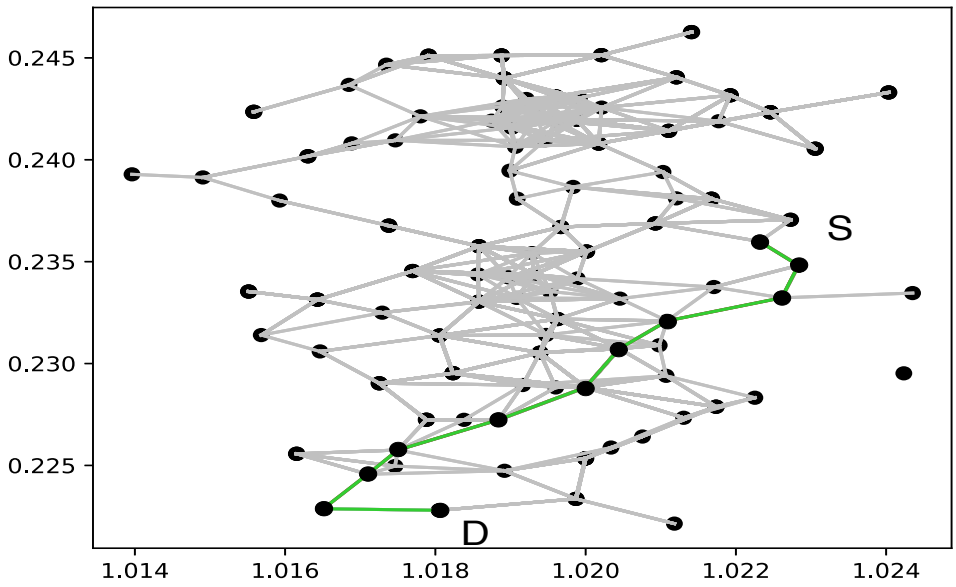
Heuristik	Medelvärde (ms)	Median (ms)	Standardavvikelse (ms)	Differens (%)
Distans:	27.91	27.0	3.98	+0.00
Signal:	30.66	30.0	3.72	+9.85
Blind:	33.24	32.0	6.7	+19.09

Tabell 2: Olika resultat för en A*-algoritm med tre olika heuristiker att beräkna en väg mellan alla par av noder i nätverket med 200 noder.

Figur 12 och 13 illustrerar hur respektive heuristik traverserar ett nätverk med 200 noder i ett koordinatsystem. De svarta punkterna representerar noder, de gråa bågarna representerar alla förbindelser i nätverket, de röda bågarna representerar förbindelser som heuristiken traverserade i sökandet efter destinationsnoden, och de gröna bågarna representerar vägen som heuristiken hittade till destinationsnoden. Det egentliga maximala antalet hopp på fyra från ursprungsnoden hävdes då dessa figurer genererades för att framhäva deras olika beteenden.



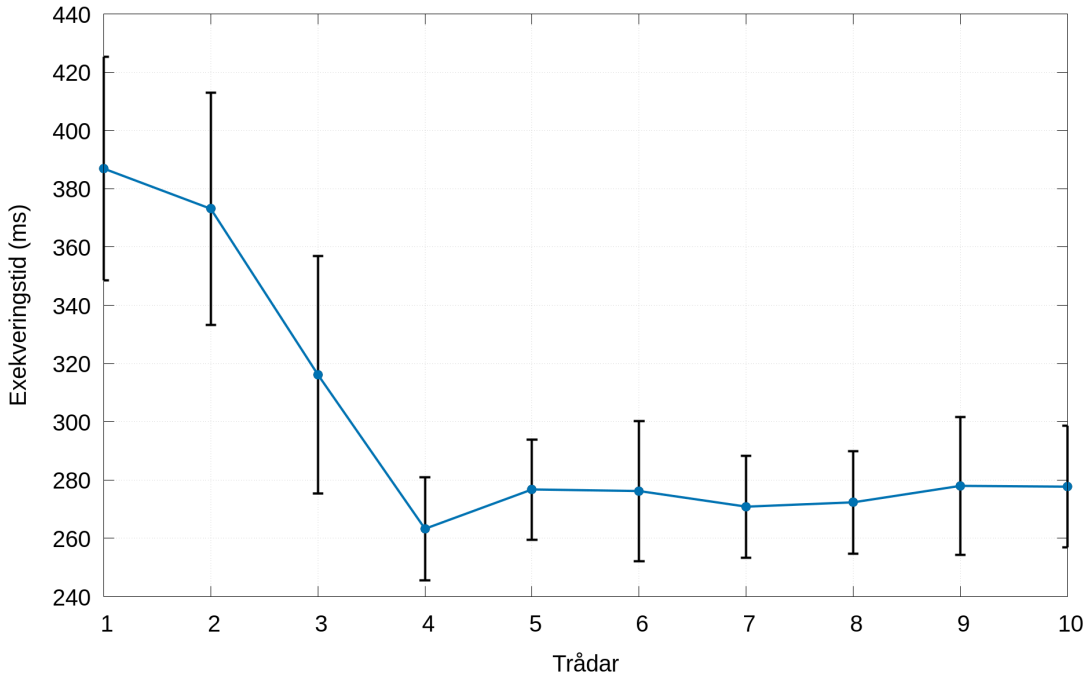
Figur 12: Vägen som traverseras från nod S till nod D av signalkvalitetsheuristiken.



Figur 13: Vägen som traverseras från nod S till nod D av distansheuristiken.

5.2 Trådar

Figur 14 visar hur antalet trådar påverkade programmets exekveringstid då heuristik baserad på distans användes. Med fyra trådar uppnåddes en genomsnittlig exekveringstid på 263 millisekunder för att behandla 39800 förfrågningar. Standardavvikelsen var 21.5 millisekunder. De vertikala linjerna i figuren representerar $\mu \pm \sigma$, där μ är medelvärdet av exekveringstiderna och σ är en standardavvikelse. Tiderna är mätta från det att hela testnätverket lästs in och alla köer har fyllts med förfrågningar tills det att alla köer med förfrågningar är tomma och alla trådar har behandlat sin sista förfrågning.



Figur 14: Genomsnittlig exekveringstid över 100 körningar för programmet att med distansheuristiken behandla 39800 förfrågningar om kortaste vägen som funktion av antal trådar som användes.

6 Diskussion

Syftet med detta kapitel är att diskutera, kommentera, och kritisera resultat och metod. Kapitlet består av en resultatdel, en metoddel, och en del med arbetet i ett vidare sammanhang.

6.1 Resultat

Signalkvalitetsheurestiken visade sig vara prestandamässigt likvärdig distansheuristiken. Knappa nio procent skiljde dessa två åt i exekveringstid och sannolikt hade differensen varit än mindre om antalet hopp från ursprungsnoden satts till maximalt fyra. Den lilla skillnaden i exekveringstid kan ha att göra med att distansheuristiken är mer komplex att beräkna än heuristiken baserad på signalkvalitet, likaså är den blinda heuristiken mindre komplex än signalheuristiken. Eftersom simulatorn tillfredsställde 39800 kortaste-vägen förfrågningar med fyra trådar på 263 millisekunder i snitt, så är differensen av simulatorns totala exekveringstid mellan distansheuristiken och signalheuristiken endast 1 procent.

Det kan sägas att distansheuristiken är den lämpliga heuristiken att använda om syftet är att få fram den kortaste vägen. Signalheuristiken är däremot ett alternativ om intresset istället ligger i att ta fram den väg med bäst signalkvalitet.

Det visade sig att fyra trådar gav den minsta genomsnittliga exekveringstiden 263 millisekunder såväl som den lägsta individuella exekveringstiden på 233 millisekunder. Exekveringen av programmet gjordes på en dator med 4 kärnor och 2 trådar per kärna. Dessa resultat kan vara unika för hårdvaran som testerna kördes på, och andra datorer med olika många CPU:er lär mycket väl generera andra resultat. Det visade sig även att standardavvikelsen vid mätningarna var så höga som 40 millisekunder. Den lägsta standardavvikelsen på 17 millisekunder erhölls med fem trådar. Differensen mellan att köra endast en tråd i jämförelse med att köra fyra trådar är i snitt 123 millisekunder, vilket motsvarar en procentuell minskning i tidsåtgång med 32 %, på 39800 förfrågningar. Att använda fler än 4 trådar visar sig inte påverka exekveringstiden särskilt mycket.

6.2 Metod

Simulatorn använder sig av en uttömmande sökning för att beräkna alla kortaste vägar. Det medför nackdelen att om antalet noder, N, i grafen ökar med ett så ökar antalet kortaste vägar som behöver beräknas med $2 \times N$, och om antalet kommunikationstyper i grafen ökar med ett så ökar antalet kortaste vägar som behöver beräknas med $N \times (N - 1)$. Detta även om det inte är någon nod som någonsin behöver veta den kortaste vägen. Om det är känt att det finns vissa noder som aldrig kommer att behöva den kortaste vägen till och/eller från sig, så hade programmet inte behövt beräkna de vägarna. Den ökande komplexiteten hos simulatorn i samband med ett ökat antal noder hade på så vis kunnat minimeras, men någon sådan funktion implementerades ej.

Federatet som utvecklats i detta arbete för att beräkna hur kommunikationen ska dirigeras genom nätverket är kompatibelt med vilket nätverk som helst. Förutsättningen är att federatet förses med information om nodernas geografiska position såväl som nodernas parvisa uppkopplingskvalitet i enlighet med HLA. Sökalgoritmen som federatet

bygger på är komplett, det vill säga att en väg mellan två noder alltid kommer hittas om en sådan finns och den vägen består av mindre än fyra bågar, enligt avgränsning 1 i Avsnitt 1.4. Om distansheuristiken används så kommer den kortaste vägen alltid hittas mellan två noder om en väg finns, och om heuristiken baserad på signalkvalitet används så kommer den väg med bäst uppkopplingskvalitet alltid hittas mellan två noder om en väg finns.

Utifrån metoden som tillämpas i arbetet går inte att dra några slutsatser om lösningens effektivitet i nätverk generellt. Det nätverk som lösningen testades på innehöll precis 200 noder, och förgreningsfaktorn var ungefär sju. Det innebär att resultatens validitet är begränsade till nätverk med samma egenskaper som det som användes i detta arbete. Givetvis är även resultaten beroende på vilka hårdvaruresurser som federatet allokeras under körning och på relevanta mjukvaruspecifikationer. Validiteten måste även utvärderas med hänsyn till hur många mätningar som resultaten baseras på. I metoden som tillämpades i detta arbete baseras varje individuellt mätresultat i Kapitel 5 på genomsnittliga resultat över 100 programkörningar.

Resultaten som ges från metoden uppnår viss grad av reliabilitet: standardavvikelsen för exekveringstiden då fyra trådar kördes i 14 var 21.5 millisekunder. I övrigt är resultaten endast beroende av nätverkets struktur. Alla steg beskrivs utförligt i Kapitel 4, i Tabell 1 presenteras också hårdvaran. Replikation och därmed det vetenskapliga kravet på falsifierbarhet kan därför uppfyllas[2].

6.3 Arbetet i ett vidare perspektiv

Det här arbetet kommer främst att vara till nytta för den svenska militären. Code of Ethics for Scientists [5] är en hederskodex som riktar sig mot forskning som angår bland annat militära hjälpmedel, och förklarar att det är osäkert om det är etiskt försvarbart att ta fram verktyg som kan bistå i ett militärt sammanhang. Enligt punkt tre i hederskodexen så har varje forskare därför ett ansvar att noggrant utvärdera arbetets möjliga konsekvenser. En konsekvens av detta arbete kan vara att ett av militärens utbildningsverktyg speglar verkligheten bättre. Det kommer att bli tydligare vilka enheter på ett stridsfält som har möjlighet att kommunicera över radio. Den informationen kan i sin tur användas av den svenska militären för att öva på, och förbättra, militära taktiker - inte bara radiokommunikation.

[2]Falsifierbarhet uttrycker, enligt filosofen Karl Popper, om en utsaga överhuvudtaget går att motbevisa. https://explorable.com/falsifiability, september 2020

7 Slutsatser

Detta kapitel innehåller dels svar på frågeställningarna i Avsnitt 1.3 och dels ett stycke om framtida arbete.

7.1 Frågeställningar

1. Är en centraliserad routingalgoritm tillräckligt effektiv för att utföra en uttömmande sökning av alla vägar, det vill säga vägen mellan alla par av noder, eller måste den arbeta på efterfrågan om en väg? Hur skalbar är en uttömmande sökning?

   En centraliserad routingalgoritm skalar dåligt men är enkel att implementera i fallet som studeras eftersom de andra federaten då inte behöver kunna ta emot nätverksbeskrivningar eller beräkna vägar på egen hand. Med en centraliserad routingalgoritm behöver endast ett federat göra detta, och sedan kan de andra federaten tillhandahålla vägarna från detta federat. Den centraliserade routingalgoritmen presterar tillräckligt bra för att beräkna 39800 kortaste vägar på 263 millisekunder. Detta ses som tillräckligt effektivt. Simulatorn har även stöd för att avaktivera den uttömmande sökningen och istället ta emot förfrågningar och skicka ut svar på dessa om det senare skulle behövas.

2. I vilken utsträckning kan trådning användas för att minska exekveringstiden av den centraliserade routingalgoritmen?

   Trådningen minskade den centraliserade routingalgoritmens exekveringstid från 388 millisekunder till 263 millisekunder, vilket motsvarar en procentuell minskning av 32 procent, på en dator med fyra kärnor och två trådar per kärna.

3. Vilken i kontexten lämplig heuristik bör användas i en A*-algoritm för att hitta vägen mellan två noder i nätverket som söker kommunikation med varandra? Först och främst söks den kortaste vägen mellan två noder. Om algoritmen är tillräckligt effektiv så kan det även finnas intresse av att hitta den väg som resulterar i bäst förbindelsekvalitet. Hur presterar algoritmerna som hittar dessa två typer av vägar i jämförelse med varandra?

   Både en kortaste väg och en väg med bäst förbindelsekvalitet kunde hittas mellan två noder i nätverket. Algoritmen som hittar de två vägarna hade likvärdig prestanda; alltså finns möjlighet att välja vilken algoritm som ska användas beroende på syfte.

7.2 Framtida arbete

Centraliserade routingalgoritmer är mindre skalbara än de distribuerade motsvarigheterna. Den mer kraftfulla distribuerade modellen bör implementeras om världen som ska simuleras består av många fler noder och/eller många fler kommunikationstyper. Till exempel så kan det vara intressant att även simulera internettrafik, telefontrafik och fordonstrafik i samma federation. Med den centraliserade routingalgoritmen som utvecklades i detta arbete hade detta inte varit möjligt eftersom antalet grafer och antalet noder hade varit för många. Det skulle vara möjligt att öka antalet förfrågningar som den distribuerade modellen kan hantera genom att parallellisera lösningen på flera

datorer och köra simuleringen på datorer med bättre hårdvara. Data Distribution Management som förklarades i Avsnitt 3.5 implementerades inte i arbetet, vilket innebär att alla kortaste vägarna i nuläget broadcastas till alla andra federat.

En Floyd-Warshall kan implementeras för att beräkna alla kortaste vägar i den uttömmande sökningen istället för A*-algoritmen som används i detta arbete, men en sådan lösning har inte stöd för att även hantera förfrågningar baserat på request-response. Istället lämnas den implementationen, och effektiviseringen av den uttömmande sökningen, till vidare arbete.

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] 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.

[4] 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.

[5] Ryden L. Tibell G. Gustafsson, B. and P. Wallensten. Code of ethics for scientists. 21(4):1, 1984.

[6] Stefan Hougardy. The floyd–warshall algorithm on graphs with negative cycles. *Information Processing Letters*, 110(8-9):279–281, 2010.

[7] 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.

[8] Ulrich Klein, Thomas Schulze, and Steffen Straßburger. Traffic simulation based on the high level architecture. In *1998 Winter Simulation Conference. Proceedings (Cat. No. 98CH36274)*, volume 2, pages 1095–1103. IEEE, 1998.

[9] 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.

[10] 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.

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

[12] 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.

[13] B Löfstrand Å Wihlborg S Eriksson M Johansson F Klasson P Svensson P Aktanius Möller, M Karlsson. The hla tutorial, 2012. available at http://pitchtechnologies.com/wp-content/uploads/2014/04/TheHLAtutorial.pdf.

[14] 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.

[15] 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.

[16] 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.

[17] 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.