СЪДЪРЖАНИЕ

Увод

В последните десетина години сме свидетели на непрекъснат прогрес в областта на безжичните комуникации. Налице е огромен напредък в съществуващата мрежова инфраструктура, броя на достъпните приложения както и появата на множество устройства използващи безжична комуникация: портативни и джобни компютри, преносими компютри, клетъчни телефони, персонални асистенти (PDAs). Тези устройства непрекъснато увеличават своята мощност и възможности и започват да играят все по-важна роля в нашия живот. Показателен е факта, че освен, че стават все по-малки, по-евтини, по-удобни и по-мощни, те вече изпълняват все повече приложения и мрежови услуги.

В днешни дни когато говорим за търговското използване на безжичната решения, почти винаги ги свързваме със съществуването на скъпа безжична инфраструктура, разчитаща на наличието на фиксирана, централизирана и предварително настроена апаратура. Въпреки, че това води до някои очевидни предимства се изисква време за предварителна настройка, закупуване на апаратура която може да е доста скъпа, а също така съществуват ситуации в които липсва инфраструктура, не може да бъде инсталирана на нужното място или за нужното време. Осигуряването на така необходимата свързаност и достъп до мрежовите услуги в тези ситуации изискват изграждането на мобилни разпределени мрежи. Мобилните разпределени мрежи (Mobile Ad-hoc Networks - MANETs) се състоят от устройства, който са в състояние самостоятелно да се организират в безжични мрежи. Тъй като една разпределена мрежа се изчерпва с устройствата от който е изградена имаме възможност за лесно разгръщане на една динамична, само-организираща се и сравнително евтина мрежа. Отделните мобилни станции имат възможност да се движат напълно произволно като по този начин топологията може да се променя доста бързо и непредсказуемо. Този тип мрежи могат да съществуват като самостоятелни или да бъдат свързвани към по-големи мрежи и Интернет. Тези предимства на разпределените мрежи имат все пак и своята цена, която е за сметка на по-сложни технологични решения, които се изискват на всичките слоеве на използвания комуникационен модел. Поради всички тези причини, мобилните разпределени мрежи (MANETs) са една от най-новите и предизвикателни области в класа на безжичните мрежи, която обещава да има все по-голямо влияние върху живота на отделния човек. На тях се залага да са следващата ключовата стъпка в еволюцията на безжичните мрежи. Както и останалите безжични мрежи и разпределените такива се изправят пред традиционните проблеми на безжичната и на мобилната комуникации като оптимално използване на честотната лента, контрол върху изразходваната мощност, подобрения върху качеството на предаване. В добавка характеристики като липсата на фиксирана инфраструктура и факта, че мобилните станции освен като крайни устройства действат и като маршрутизатори, препращайки пакетите на останалите станции по пътя им към крайния получател (т.нар. multihop routing) са уникални за този тип мрежи и крият нови проблеми за изследване като конфигурация на мрежата, откриване на съседи, поддържане на топологията, адресиране, маршрутизиране.

Появили се първоначално за да изпълняват изцяло военно-тактически задачи днес пред мобилните разпределени мрежи съществуват все по-голямо разнообразие от възможности за тяхното приложение. Това е особено вярно в последните няколко години когато рязко нараснали интерес към тяхното изследване, води до засилване на интереса от страна на индустрията и комитетите за стандартизация. Появата на нови технологии като Bluetooth, IEEE 802.11 и Hyperlan значително улесняват използването на разпределените мрежи извън военния сектор. Някои от новопоявилите се приложения на мобилните разпределени мрежи включват: изграждане на мрежи от сензори, използването им при спешни спасителни операции, изграждане на виртуални класни стаи, комуникация по време на конференции, събирания или лекционни часове, роботизирани домашни любимци, достъп до Интернет извън границите на дома или офиса, изграждане на WLAN и PAN мрежи в дома и офиса, електронна търговия, мобилни офиси, транспортни услуги: разпространение на новини, на информация за времето, коопериране с останалите автомобили на пътя за предварително известяване за пътни събития и инциденти.

Целите на настоящата дипломна работа е да се разгледат възможностите и проблемите пред използването на мобилните разпределени мрежи за разпространяване на транспортна информация (задръстване, инцидент, мокър и тъмен участък и други) между автомобили участващи в пътното движение.

1 ОБЗОР НА СЪЩЕСТВУВАЩИТЕ АЛГОРИТМИ И ПРОТОКОЛИ ЗА МОБИЛНИ РАЗПРЕДЕЛЕНИ МРЕЖИ (MANETS И VANETS)

Мобилните разпределените мрежи елиминират необходимостта от съществуваща инфраструктура, те позволяват на устройствата да създават и да се присъединяват към мрежи образувани динамично когато се появи нужда за това, без значение на времето и мястото, за голям обхват от възможни приложения. Поради голямата им гъвкавост, която позволява тяхното разгръщане без да е необходимо предварително настройване и администриране, както и голяма свобода в движението на станциите, интереса към тях е особено голям в последно време. Повечето от съществуващите протоколи и алгоритми датират от последните десетина години. Въпреки това обаче концепцията за разпределените мрежи не е нова, тя датира от далечната 1969 година.

Исторически първите мобилни разпределени мрежи също както и други ключови технологии имат за свой първоизточник Министерството на отбраната на Съединените Щати (DoD). Целите им били обвързани с чисто тактически задачи като подобряване на комуникацията на бойното поле. Поради голямата динамичност на военни операции не винаги може да се разчита на предварително изградена инфраструктура на бойното поле, а чистата безжична комуникация има ограничението сигналите рядко да могат да се разпространяват на много големи разстояния в следствие на интерференция. Именно тук се е появила необходимостта от мрежа която да е безжична, да не изисква наличието на предварително инсталирана инфраструктура, да е само-конфигурираща се и да позволява междинно препредаване на информацията така че да се повиши обсега на действието й. Първите приложения датират от 1972 година и са свързани с проекта DARPA Packet Radio Network (PRNet). Проекта бил вдъхновен от успеха на технологията за комутиране на пакети (packet switching), а именно споделяне на пропускателната способност и store-and-forward маршрутизиране и възможностите за използването им в мобилните разпределени мрежи. В резултат от проекта била предложена една разпределена архитектура състояща се от мрежи от мобилни станции с радиочестотни приемо-предаватели комуникацията в която ставала с разпръскване на радио сигнали. Тази архитектура разчитала на минимален централизиран контрол и комбиниран между ALOHA и CSMA достъп до преносната среда позволяващ динамичното и споделяне. В допълнение чрез използване на многостъпково ‘store-and-forward’ маршрутизиране се е избегнал проблема с разстоянието. По този начин е станало възможно комуникацията между повече на брой потребителя разположени в голяма географска област.

През 1983 година се появява проекта Survivable Radio Networks (SURANs), който е създаден от DARPA с идеята да разреши някои останали отворени проблеми, а именно мащабируемостта и лесното разширяване на мрежата за по голям брой станции, сигурността, повишаване на изчислителните способности и по-ефективно управление на енергията на радио излъчвателите. Най-главните цели пред проекта били създаването на мрежов алгоритъм който да работи за мрежи включващи десетки хиляди станции, да устоява на атаки на сигурността, както и да се използват малки по размер, с ниска цена и ниска консумация радио приемопредаватели, които обаче да поддържат по-усложнените протоколи за обмен на пакети. В резултат на тези усилия през 1987 година се появява технологията Low-cost Packet Radio (LPR). Новото било въвеждането на широчинен спектър, а също така и разделянето на клъстери с което се решавали проблемите с мащабируемостта.

Периода от края на 80-те и началото на 90-те е белязан с бума на развитие на микрокомпютрите и Интернет което създава благоприятна среда за развитието и на безжичните мрежи. През 1994 година с цел да засили развитието на безжичните мобилни мрежи DARPA стартира информационната програма Global Mobile (GloMo) целта на която е разработване на Ethernet подобна мултимедийна свързаност навсякъде и по всяко време между безжични устройства. В резултат се появяват няколко мрежови модела: Wireless Internet Gateways (WINGs) използващ плоска peer-to-peer архитектура и Multimedia Mobile Wireless Network (MMWN) използваща йерархична мрежова архитектура разделена на базата на клъстерите.

Един от последните проекти на военните на САЩ в областта на мобилните разпределени мрежи Littoral Battle-space Advanced Concept Technology Demonstration (ELB ACTD) е от 1999 година и е свързан с изграждането на връзка между кораби в открито море и морския щаб на сушата посредством препредаване от въздушни станции.

Въпреки, че първоначалните задачи пред мобилните разпределени мрежи са били изцяло военни и тактически, в по-скорошно време са открити много нови възможности за прилагането им извън военния сектор, върху които се фокусира тяхното развитие в последните няколко години. В следващите редове са дадени настоящи и бъдещи приложения на мобилните разпределени мрежи както и примери за услуги които те осигуряват:

- Мрежа от сензори: сбор от вградени устройства със сензори за събиране на информация в реално време за автоматизиране на всекидневните функции. Примери за използването им са за следене на климатичните условия, на земетръсната активност, различни сензори за производствена апаратура и други;
- Спешни спасителни операции: търсене на загубени хора или бързо извличане и транспортиране на информация за пациент от и до болнично заведение;
- Алтернатива на фиксираната инфраструктура при отпадането и в случай на земетресение, ураган, пожар, наводнение или друго бедствие.
- В образованието: за изграждане на виртуални класни стаи или за комуникация по време на конференции, събирания или лекции;
- За забавление: многопотребителски игри, роботизирани домашни любимци, достъп до Интернет извън границите на дома или офиса;
- За изграждане на WLAN и PAN мрежи в дома и офиса

- В търговския сектор: електронна търговия – електронно разплащане от всякъде, мобилни офиси, за автомобилни услуги като разпространение на новини, на информация за времето, коопериране с останалите автомобили на пътя за предварително известяване за пътни събития и инциденти.

Динамичната природа на мобилните разпределени мрежи води до честа и непредсказуема промяна на мрежовата топология което затруднява и усложнява маршрутизирането между мобилните станции в мрежата. Тези предизвикателства, заедно с особената важност на маршрутизиращите протоколи за комуникацията между отделните станции прави маршрутизирането най-активно изследваната област от MANET мрежите. В последните няколко години са предложени голям брой маршрутизиращи протоколи и алгоритми, чиито предимства се оценяват и сравняват на базата на различни конфигурации на мрежата и на моделите на движение на мобилните станции. Основни изисквания пред новите маршрутизиращи протоколи са коректно и ефикасно установяване на пътища по които да бъдат разпространявани съобщения надеждно и без особено забавяне, установяването трябва да става чрез минимално количество контролна информация и минимално използване на пропускателната способност. Традиционните маршрутизиращи протоколи са неприложими за мобилните разпределени мрежи поради характерните особености на последните.

Съществуващите маршрутизиращи протоколи за мобилни разпределени мрежи най-общо могат да бъдат разделени в три категории: *проактивни*, *реактивни* и *хибридни*. Проактивните маршрутизиращи протоколи поддържат информация за пътищата между кои да е две станции в мрежата. Тъй като тази информация обикновено се пази в таблици те се наричат още table-driven протоколи. Реактивните маршрутизиращи протоколи от друга страна изграждат път между две станции само когато възникне нужда от това. Обикновено посредством процедура за откриване на нов път инициирана от източника на информацията. Веднъж открит, маршрута се пази докато е наличен и докато се използва. След определен период без да бъде използван той се отстранява и при следваща необходимост се търси наново. Предимството на проактивните маршрутизиращи протоколи е времето за използване на маршрута, тъй като той е наличен може да се използва веднага, докато при реактивните е необходимо време за неговото откриване което довежда до забавяне на предаването на първия пакет. Въпреки това проактивните протоколи са свързани с поддържане на таблици, т.е. необходимо е мобилните устройства да имат по-голям капацитет от памет, а също така при тях изразходването на пропускателната способност за целите на маршрутизирането е много по-голямо. Третия тип протоколи, хибридните имат за цел да спечелят от предимствата на другите двата използвайки както реактивен така и проактивен подход. Около всяка станция се формира зона, вътре в зоната се използва проактивно маршрутизиране, а за всеки пакет към станция извън зоната на източника се прилага реактивно маршрутизиране.

Към проактивните протоколи принадлежат Destination-Sequenced Distance-Vector (DSDV), Wireless Routing Protocol (WRP), Global State Routing (GSR), Fisheye State Routing (FSR), Optimized Link State Routing (OLSR), Cluster-head Gateway Switch

Routing (CGSR), Hierarchical State Routing (HSR), Topology broadcast reverse path forwarding (TBRPF), Distance routing effect algorithm for mobility (DREAM).

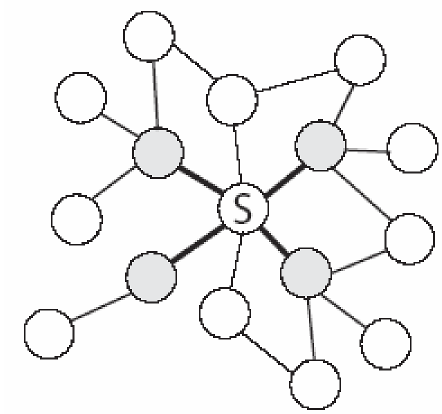
Протоколът DSDV намира единствен път между всеки две мобилни устройства, който гарантирано не съдържа цикъл и е с минимална дължина. За намиране на най-къс път се използва Белман-Форд алгоритъма. Избягването на цикли се постига с последователностни номера. Всяка станция поддържа последователно нарастващ номер за себе си и списък от номерата на всички останали станции. С цел намаляване на големината на служебната информация се използват два типа съобщения за опресняване на пътищата: пълни (изпраща се цялата маршрутизираща таблица) и частични (изпращат се само настъпилите промените). Въпреки това DSDV все още използва значително количество от пропускателната способност на мрежата и не може да бъде използван за мрежи съдържащи голям брой станции.

Протоколът WRP също гарантира, че намерените пътища не съдържат цикли. Той също както DSDV принадлежи към класа на distance-vector маршрутизиращите протоколи. Недостатък на този протокол е, че поддържа четири маршрутизиращи таблици което консумира значителна част от ресурсите памет на мобилните станции, която нараства с нарастването на размерите на мрежата. Друг недостатък е, че разчита на надеждно и в правилен ред получаване на маршрутизиращите съобщения. За станциите с ограничена мощност недостатък е периодичното обменяне на т.нар. hello съобщения между съседите когато няма предаване на пакети, което не позволява на станциите да преминат в спящ режим и да намалят консумацията си.

Работата на GSR протокола се основава на класическите link-state маршрутизиращи протоколи с тази оптимизация, че маршрутизиращите съобщения се обменят само между съседни станции което значително намалява броя на необходимите контролните съобщения. Обмяната на тези съобщения става през определен период от време. Големината на тези съобщения обаче е значителна и при нарастване на мрежата нараства още което повишава и използването на пропускателната способност от маршрутизиращия протокол.

Протокола FSR е подобрение на GSR. Подобрението идва от факта, че информацията за по-близките (по брой стъпки) станции се разпространява по-често от тази за по-отдалечените. Това значително подобрява работата на протокола като отчитаме факта, че обикновено по отдалечените пътища пропадат по-често в следствие на мобилността на станциите. Това подобрение обаче е за сметка на намалена акуратност на пътищата до по-отдалечените станции.

Протоколът OLSR е оптимизирана версия на класическия link-state маршрутизиращ протокол какъвто е OSPF. Оптимизирането идва от използване на концепцията за Multipoint Relays (MPRs) (фиг.1), това е множество от съседни станции които ще извършват разпръскването на маршрутизиращите съобщения до станциите отдалечени на две и повече стъпки от източника. По този начин значително се намалява броя на broadcast съобщенията в мрежата. Маршрутизиращите съобщения се обменят през определен период от време, като определянето на този период е критично за работата на протокола. Протоколът дава значително подобрение спрямо традиционните link-state протоколи за мрежи в които мобилните станции са разположени нагъсто.



Фиг. 1: Многоточкови релета

Протоколът CGSR принадлежи към класа на йерархичните маршрутизиращи протоколи. Отделните станции са групирани в клъстери, но вътре в клъстера не се налага да се поддържа йерархичност. За всеки клъстер се избира една главна станция, т.нар. cluster-head която управлява останалите. Всяка комуникация между клъстерите задължително минава през главната станция за съответния клъстер. Останалите станции трябва да пазят информация само за пътя до своята главна станция. Станциите които попадат в два клъстера се наричат gateway станции и чрез тях се осъществява връзката между отделните клъстери. Въпреки, че се явява сериозно подобрение спрямо протоколите използващи механизма на наводнението (flooding) количеството на допълнителната контролна информация необходима за поддържането на клъстерите е значително. Причината е, че всяка станция периодично разпръсква своята таблица със съседствата в клъстера и я обновява с новополучената информация от останалите.

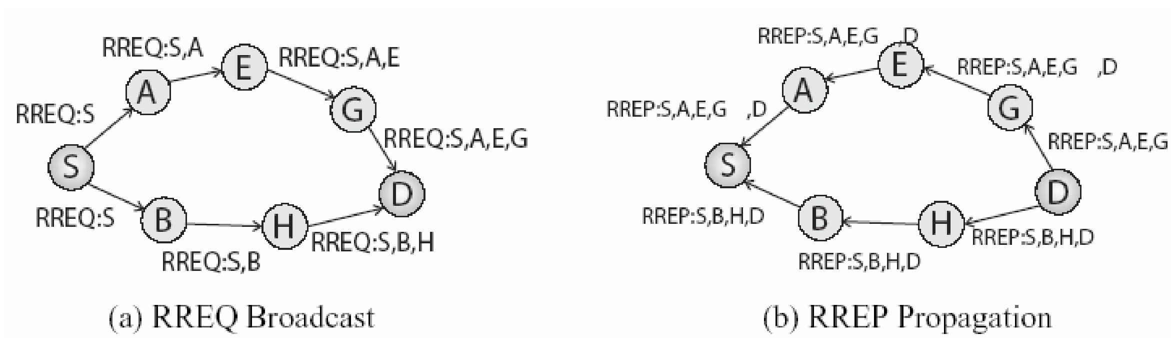
Протоколът TBRPF е още един представител на link-state маршрутизиращите протоколи. При него се използва концепцията на Reverse-Path Forwarding (RPF) за разпространение на промените в топологията по обратния път на покриващото дърво. За изчисляването на най-кратките пътища се използва модифицирана версия на алгоритъма на Дийкстра за частична информация за топологията, която има всяка станция. Чрез използването на частични hello съобщения, в които са включени само промените в състоянието на съседните станции, което прави hello съобщенията значително по-малки.

DREAM протоколът е най-подходящ от разгледаните проактивни маршрутизиращи протоколи за мрежи включващи голям брой мобилни станции. Причината за това е, че вместо да се обменя информация за състоянието на връзките се обменят местоположенията на станциите, което е много по-малко като количество информация. Всяка станция е способна да определи географското си местоположение посредством GPS устройство. В допълнение честотата на обменяните маршрутизиращи съобщения зависи от мобилността на отделните станции и ефекта на разстоянието, това означава, че статичните станции няма нужда да изпращат маршрутизираща информация.

В заключение проактивните маршрутизиращи протоколи не са подходящи за мрежи състоящи се от по-голям брой станции поради повишаването на количеството маршрутизираща информация което би попречило на обменянето на полезна информация. Някои от разгледаните протоколи биха работили добре и при по-голям брой станции, но относително ниска мобилност. Използването им би било уместно само когато се цели преследване на качество на услугата (QoS) което не може да бъде осигурено от реактивните маршрутизиращи протоколи.

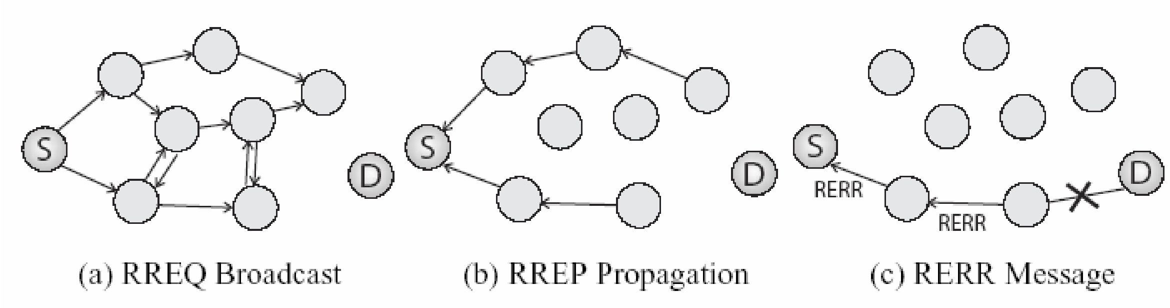
Реактивните маршрутизиращи протоколи, наричани още *on-demand* са проектирани така, че да намалят допълнителната контролна информация която се обменя при проактивните, като се поддържа информация само за активните пътища. Това означава, че пътища се изграждат и поддържат само за станции които се нуждаят да изпратят данни до определен получател. Откриването на пътища в този случай обикновено се осъществява по механизма на наводняването (flooding). Когато бъде достигнат получателя той потвърждава пътя използвайки междинните станции в обратен ред ако връзките са двупосочни или отново с наводнение ако съществуват еднопосочни връзки. Реактивните маршрутизиращи протоколи се разделят от своя страна на две под-категории: *source routing* и *hop-by-hop routing.* При source routing реактивните маршрутизиращи протоколи заглавната част на всеки обменен пакет включва адресите на всички междинни станции между източника и получателя. По този начин се избягва необходимостта междинните станции да поддържат маршрутизираща информация за всеки активен път, който минава през тях за да могат да препратят информацията към крайната станция. Нещо повече, не е необходимо да се поддържа свързаност със съседните станции посредством периодични сигнализиращи съобщения. Този подход обаче прави използването им в големи мрежи компрометирано поради две причини; първо с нарастването на броя на междинните станции нараства и вероятността за пропадане на пътя и второ с нарастване на дължината на пътищата силно нараства и допълнителната служебна информация включвана в заглавната част на всеки пакет. При hop-by-hop маршрутизирането всеки пакет съдържа само адресите на крайната станция и на следващата станция. Всяка междинна станция проверява маршрутизиращата си таблица за да реши накъде да препрати пакета по пътя му към своя получател. Този подход позволява по-голяма адаптивност към динамичната среда на мобилните разпределени мрежи защото всяка станция може да опреснява маршрутизиращата си таблица с по-нова информация която получава и да препредава пакетите по нови и по-добри пътища. Недостатък е необходимостта междинните станции да поддържат маршрутизираща информация за всеки активен път през тях, както и да поддържат свързаност със своите съседи. Към реактивните протоколи принадлежат Ad-hoc On-demand Distance Vector (AODV), Dynamic Source Routing (DSR), Temporary Ordered Routing Algorithm (TORA), Associativity-based Routing (ABR), Location-aided Routing (LAR), Ant-colony-based Routing Algorithm (ARA), Flow Oriented Routing Protocol (FORP), Cluster-based Routing Protocol (CBRP).

Протоколът DSR принадлежи към source routing реактивните маршрутизиращи протоколи и за него са валидни недостатъците описани по-горе. За динамични и много големи мрежи значителна част от пропускателната способност се използва за обмяна на маршрутизираща информация затова не е обосновано използването му. Въпреки това той има значителни преимущества пред останалите реактивни протоколи и за малки до средни мрежи и такива с умерена мобилност работи значително добре. Когато станция се нуждае да изпрати пакет до друга за която не знае маршрут тя наводнява мрежата с *route request* съобщения. Всяка станция която получи тези съобщения ги препраща, включвайки себе си в заглавната част на съобщението (фиг. 2), освен ако не е крайната станция или няма информация за път до нея, като в този случай отговаря с *route reply* съобщение. Отговорът се изпраща до източника по същия механизъм. Ако има промяна някъде по пътя се изпраща *route error* съобщение и отпадналата предишна информацията се изтрива. Недостатък е недостатъчно ефективното отстраняване на остарели пътища което освен, че е свързано с повишено използване на пропускателната способност е свързано и с препълване с невярна информация на маршрутизиращата памет (route cache) на междинните станции. Предимството на този протокол пред останалите е откриването на повече от един маршрут, които се пазят временно и при отпадане на текущо използвания той може веднага да бъде заменен което може да бъде много ефективно при ниска мобилност на станциите. Друго предимство е, че не е необходимо периодичното обменяне на hello съобщения и станции могат да влязат в спящ режим, като намалят консумацията на енергия и пропускателна способност.



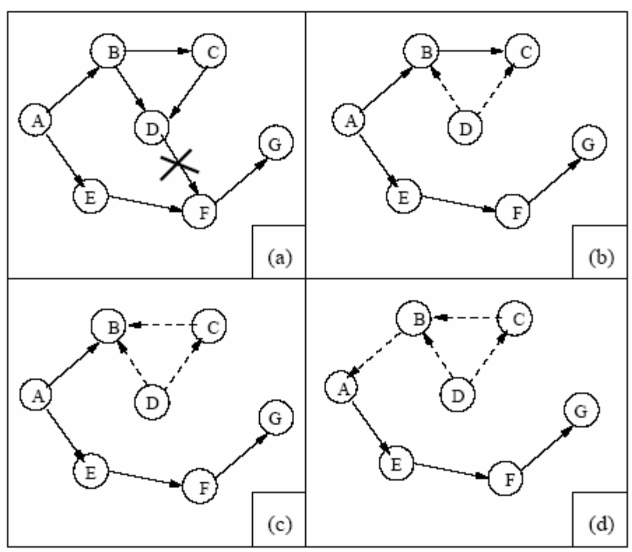
Фиг. 2: Механизма на откриване на маршрут при DSR

Протоколът AODV е друг пример за реактивен маршрутизиращ протокол, който обаче използва подхода hop-by-hop routing. За насочване на съобщенията се използва информацията в маршрутизиращата таблица. За предотвратяване на цикли и разпознаване на по-нови пътища се използва идеята за последователните номера на DSDV протокола. Тези номера се включват във всяко маршрутизиращо съобщение, при получаване на съобщение с по-нов номер старият път се изтрива от маршрутизиращата таблица независимо, че пътя може все още да е активен. Освен тях се използват и таймери след изтичането на които информацията отново се изтрива без значение дали пътя е активен или не. Протоколът поддържа използването на най-много един път до дестинация. Въпреки, че се адаптира добре и за по-големи мрежи откриването на нов маршрут може да е свързано с голямо закъснение, а отпадането му довежда до допълнително закъснение и консумира повече пропускателна способност. Модификация на протокола наречена Ad-hoc On-demand Multipath Distance Vector (AOMDV) позволява използването на алтернативни пътища за намаляване на броя на процедурите за откриване на нов път, което води до значителни подобрения. На фигура 3 е показан механизма на откриване и поддържане на пътища.



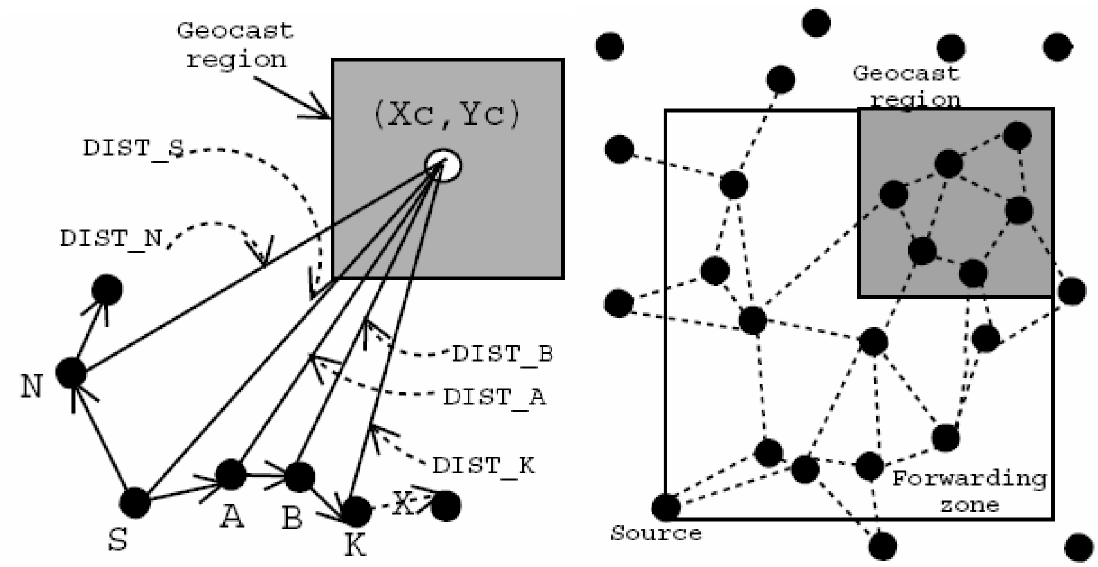
Фиг. 3: Откриване и поддържане на маршрути при AODV

При протокола TORA процедурата по откриване на нови маршрути изчислява множество пътища до крайната станция всички те несъдържащи цикли. За целта се използва т.нар. *Destination-oriented directed acyclic graph* (DAG) граф. Въпреки, че разпределените мрежи се представят от ненасочени графи, в концепцията на TORA се използва логическа насоченост на ребрата на графа по посока от източника до получателя. При отпадане на всички връзки в графа водещи до крайната станция започва процедура по обръщане на посоките на ребрата (връзките), т.нар. *link reversal* процедура (фиг. 4), докато отново се стигне до състояние при което графа отново е насочен към крайната станция. Протоколът TORA използва модифицирана версия на алгоритъма за обръщане на посоките на ребрата, който дава възможност за откриване на несвързани сегменти в графа, което е полезна черта, липсваща в голяма част от другите протоколи. По-нови пътища се откриват едва когато отпаднат всички стари, което е аналогично на AODV протокола. Недостатък на TORA е, че разчита на надеждно и в правилен ред получаване на маршрутизиращите съобщения. Също така използвайки процедурата за обръщане на посоките на връзките прави много трудно използването на метрики за определяне на по-предпочитани пътища. Последното почти елиминира предимството на множеството маршрути до крайната станция. Въпреки това TORA остава предпочитан избор за мрежи в които се изисква множество станции да имат маршрут до една крайна станция.



Фиг. 4: Фази по откриване и поддържане на маршрут при TORA

Следващия алгоритъм, LAR, използва техниката на наводняването за откриване на маршрути. Авторите на алгоритъма предлагат два подхода за ограничаване на района на разпространение на съобщенията (фигура 5). Първият е с дефинирането на зона на наводнението. При него станция ще препредаде получено съобщение само ако се намира в зоната на наводнение. За целта се разчита на това всяка станция да е способна да определи своето географско местоположение посредством GPS устройство. При втория подход всяка станция препредава получено съобщение само ако се намира по-близо да получателя от текущата. По този начин LAR открива също така най-кратките пътища. Съществуват модификации които да се справят с разпокъсани мрежи в които няма път между станциите в зоната на наводнение, но съществува такъв извън нея. Недостатък на алгоритъма е, че при силно мобилни мрежи производителността спада до тази на останалите протоколи използващи метода на наводнението, като AODV и DSR.



Фиг. 5: Две схеми за избор на зона на наводнение за LAR подхода

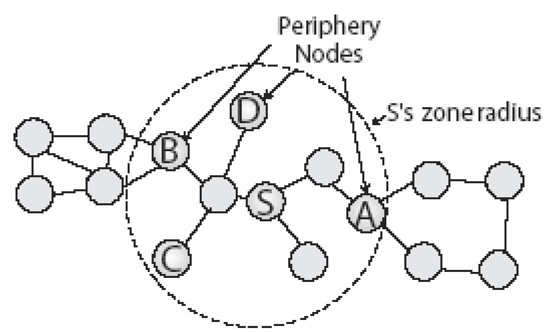
Идеята залегнала зад ARA е да се използва поведението на мравките при търсене на храна за намаляване на допълнителната служебна информация свързана с маршрутизирането. Когато мравката излезе да търси храна тя тръгва от гнездото по посока на храната като остава след себе си следа наречена феромон. Останалите мравки са способни да проследят следата. Подобно на AODV и DSR тук също имаме две фази: откриване на маршрут и поддържане на маршрут. При първата фаза се разпространява Forward ANT (FANT) съобщение по посока на крайната станция. На всяка стъпка междинната станция изчислява “силата на феромона” по броя стъпки за достигането до нея. След това препраща съобщението до своите съседи. Когато бъде достигната крайната станция тя изпраща Backward ANT (BANT) съобщение по посока на източника. Предимството на този подход е, че FANT и BANT съобщенията са малки по размер. Когато BANT съобщението достигне източника започва обмяната на данновите пакетите. За запазване на следата с всеки обменен пакет се увеличава стойността на феромона, а с изтичане на определено време той се намалява. Когато дадена връзка се прекъсне (следата изчезне), тя започва да се търси първо при съседните станции. За разпространение на маршрутизиращите съобщения отново се използва механизма а наводнението, което може да доведе до проблеми с пропускателната способност при увеличаване на размерите на мрежата.

Основната идея на протокола FORP е да намали броя на пропадналите маршрути в следствие на мобилността на станциите като се опитва да предскаже кога даден път ще пропадне. По този начин започва да се използва алтернативен маршрут още преди текущия да пропадне. Механизма за предсказване е следния: когато има нужда от изграждане на нов път между две станции се инициират Flow_REQ съобщения, които се разпращат до всички подобно на DSR протокола. Когато междинна станция получи такова съобщение тя изчислява времето на валидност на връзката с предходната станция, т.нар. *Link Expiration Time* (LET) посредством данните от GPS устройство и добавя тази стойност към съобщението преди да го препрати нататък. Когато съобщението стигне до крайната станция се определя минималното от LET времената което се нарича *Route Expiration Time* (RET). По-късно по време на обменянето на данновите пакети междинните станции отново включват LET времената за да помогнат на крайната станция да предскажи пропадането на маршрута. Ако бъде предсказано пропадане се стартира процедура подобна на първоначалното откриване чрез изпращането на Flow_HANDOFF съобщения. Предимството на този подход е минималното смущаване на трафика между двете станции в резултат на пропадането на връзката. Въпреки това, поради факта, че протокола разчита на алгоритъма на чистото наводнение на съобщенията (pure flooding) използването му може да е свързано с проблеми при увеличаване размерите на мрежата.

Протоколът CBRP е първият от йерархичните протокол от групата на реактивни маршрутизиращи протоколи. Станциите са организирани йерархично в клъстери. За всеки клъстер се избира главна станция т.нар. *cluster-head* чрез който се извършва комуникацията в клъстера и между отделните клъстери. Предимството му е свързано с намаляването на броя на обменяните маршрутизиращи съобщение, но както и другите йерархични протоколи е свързан с допълнителна комуникация за изграждане и поддържане на клъстерите. При голяма мобилност производителността му спада значително.

В най-лошия сценарий когато път между източника и получателя не е известен почти всички от реактивните протоколи имат еднакво лоша производителност поради характера на търсене на нов път. Това е обикновено случая след първоначалното включване на станцията в мрежата. По-късно когато станцията се е задържала вече известно време тя има възможност да опознае част от околната топология и някои от протоколите се възползват от това с цел оптимизирането на последващото търсене на пътища. Сравнението между проактивните и реактивните протоколи не дава ясно предимство на нито единия за по-общи приложения, тъй като всеки от тях има преимущества в една или друга ситуация. Въпреки това изследванията са показали, че реактивните протоколи се представят ако не по-добре, то поне със същата производителност както проактивните за по-общи приложения. Не е било трудно да се предположи, че ако двата подхода се комбинират ще се получат по-добри резултати отколкото поотделното им прилагане. Така се ражда идеята за хибридните протоколи. Повечето от съществуващите хибридни протоколи се базират на понятието за зона, т.е. от гледна точка на отделните станции мрежата е разделена на зони, които може да се препокриват или не. Най-известните от хибридните маршрутизиращи протоколи са: Zone Routing Protocol (ZRP) и Zone-based Hierarchical Link State (ZHLS).

Протоколът ZRP е типичен пример за хибриден маршрутизиращ протокол. Всяка станция представлява център на зона, като по този начин мрежата е разделена на множество зони, които се застъпват. Всяка зона е с радиус ρ (фиг. 6) като радиуса не оказва някакви физически граници, а е цяло число даващо броя на стъпките до станциите по ръба на зоната. Стойността на радиуса контролира размерите на проактивната и реактивната част на мрежата. Станциите в зоната са разделени на вътрешни и периферни. Периферни са всички станции отдалечени на ρ на брой стъпки от централната станция. ZRP протокола се състои от три различни компонента. Локалната проактивна компонента се нарича Intra-zone Routing Protocol (IARP), а глобалната реактивна компонента съответно Inter-zone Routing Protocol (IERP). Трябва да се подчертае, че IARP и IERP не са конкретни маршрутизиращи протоколи, сбор от проактивни, съответно реактивни маршрутизиращи протоколи. Поради факта, че топологията вътре в зоните е известна може да бъде използвано при откриване на глобални пътища. Това се осъществява чрез концепцията на т.нар. *bordercasting*. Тя се изразява в насочването на информацията за топологията предоставена от IARP към граничните станции на зоната. Компонентата която се занимава с тази задача се нарича Bordercast Resolution Protocol (BRP). За откриването на нови и отпаднали стари съседи ZRP разчита на протокол на каналния слой наречен Network Discovery Protocol (NDP). Протоколът ZRP е атрактивен с това, че може да се трансформира в изцяло проактивен, в изцяло реактивен или в нещо между тях само с променянето на един единствен параметър - радиуса на зоната. При класическата версия на протокола радиуса е предварително фиксиран. Версията на протокола Adaptive ZRP предлага алгоритъм за неговото динамично променяне.



Фиг. 6: Маршрутизираща зона на ZRP с радиус 2

Протоколът ZRP има плоска структура. За разлика от него ZHLS е йерархичен протокол. При него мрежата е разделена на зони които не се препокриват. Всяка станция има два идентификационни номера: *node ID* и *zone ID*, втория от които се изчислява чрез информация от GPS. Йерархичната структура е на две нива: топология на ниво станция и топология на ниво зона. За разлика обаче от останалите йерархични протоколи тук липсва усложненията за избор на главна станция и поддържане на клъстери. При ZHLS е намален размера на служебната информация понеже когато се търси нов път се изпращат съобщения на ниво зона. Намерените пътища са устойчиви на движение на станциите, стига да е в рамките на зоната. Недостатък на ZHLS е изискването всички станции да имат предварително зададени статични карти на зоните, което не винаги е приложимо.

Освен на проактивни, реактивни и хибридни, маршрутизиращите протоколи за мобилните разпределени мрежи могат да бъдат класифицирани по това колко са получателите на информацията на: *unicast* маршрутизиращи протоколи (получателя е един единствен и точно определен), *multicast* (получателите са група от станции, групирането се извършва на базата логическа а не физическа принадлежност), *geocast* (подобно на multicast, но групите се определят на базата на географско разположение на станциите), *anycast* (най-скоро появилия се, при него получател на информацията е коя да е станция от дадена група станции) и *broadcast* (информацията е предназначена до всички станции).

По голяма част от протоколите които разгледахме до тук спадат към класа на unicast маршрутизиращите протоколи. Концепцията за multicasting се появява наскоро поради наличието на все по-голям брой приложения изискващи изпращането на едни и същи данни от един хост до група от хостове. Тъй като unicast предаване до всички би било крайно неефективно и би заемало значителна част от пропускателната способност и мрежовите ресурси се разработват протоколи и устройства които да се справят с проблема посредством multicasting. Когато говорим за прилагането на multicast маршрутизиращите протоколи в мобилните разпределени мрежи проблемите които изникват са още по големи поради често сменящата се топология. Този тип протоколи разчитат на изграждането на дървовидна структура, в корена на която е разположен източника, а листата са всички станции до които трябва да достигне дадената информация. Колкото по оптимално е дървото толкова по оптимално е и самото доставяне на информацията. При фиксираните мрежи междинните възли на дървото представляват маршрутизатори, докато при разпределените самите станции са маршрутизатори. Дори при малка мобилност на междинните станции е възможно разпадане на дървото съпроводено от последващото му изграждане което не е ефективна задача. Печалбата обаче от този тип маршрутизиране е значителна тъй като пропускателната способност при разпределените мобилни мрежи е още по критичен ресурс.

По-нагоре вече споменахме един мултикаст протокол – MAODV. Други популярни мултикаст маршрутизиращи протоколи са: Core-Assisted Mesh Protocol (CAMP), On-Demand Multicast Routing Protocol (ODMRP), Multicast Core-Extraction Distributed Ad hoc Routing (MCEDAR), Differential Destination Multicast (DDM), Ad-hoc Multicast Routing Protocol (AMRoute). Поради специфичните особености на мобилните разпределени мрежи не винаги е най-ефикасно използването на дървовидна структура, при повишаване на броя на станциите по ефикасно се явява използването на мрежеста (mesh) структура, поради по-голямата гъстота на станциите и по-малкото контролна информация нужна за поддържането на последната. За оптимизирането на маршрутизацията се използва информация за местоположението на станциите (GPS информация) там където е възможно.

Информацията за местоположението на станциите е силно застъпена при geocast маршрутизиращите протоколи. Целта на този тип протоколи е да доставят пакети до група от станции разположени в дадена географска област. При тях се използват два подхода за доставяне на информацията до географския регион. При единия се използва метода на наводняването (най-често негова модификация), а при другия се изгражда пътища до областта и чак когато информацията достигне до нея се наводнява. Първият протокол който ще разгледаме е Location Based Multicast (LBM). Този протокол е базиран на вече разгледания по-горе Location-aided routing (LAR). За разлика от него обаче получател на информацията не е една единствена станция, а група от станции разположени в т.нар. geocast регион. Алгоритъма за разпространение на информацията е същия.

Протоколът Voronoi diagram based geocasting има за цел да повиши процента на успешно доставените пакети и да понижи броя на стъпките в маршрута. Мрежата се разделя на отделни региони посредством използването на диаграмите на Вороной. По този начин всички съседи на източника на информация попадат в даден регион. Решението кои станции да разпространяват маршрутизираща информация се взима на базата на това дали региона в който се намират пресича geocast областа или не.

Протоколът GeoGRID също както LBM се базира на модификация на unicast маршрутизиращ протокол, а именно GRID. При него мрежата се разделя на правоъгълна решетка. За всяка клетка от решетката се избира една станция наречена шлюз, която ще извършва препращането на съобщенията по посока на geocast областта. Самата област се състой от една или няколко съседни клетки. Дефинира се още т.нар. forwarding зона която представлява минималния правоъгълник включващ едновременно geocast региона и източника на информацията. Избора на шлюз е съществен за работата на протокола. За да е максимално ефективно маршрутизирането е добре това да е станцията която е най-близо до центъра на клетката. При наличие на по-близка станция в следствие на мобилност се избира нов шлюз.

Следващия протокол GeoTORA има за цел да подобри използваемостта на пропускателната способност като избегне механизма на наводнението. За маршрутизиране до geocast областта се използва разгледания по-горе unicast протокол TORA. Когато пакета стигне до станция намираща се в geocast областта тя започва да наводнява областта. Тук се използва концепцията на т.нар. anycasting. При нея е достатъчно пакета да се достави до поне една, без значение коя, станция от областта, която след това го разпространява до останалите. Недостатък на този подход е, че ако нямаме свързаност между станциите в областта не се гарантира доставяне на пакета до всяка станция.

Протоколът Mesh-based Geocast Routing Protocol има а цел да повиши надеждното предаване на информация до Geocast областта посредством изграждане на резервни връзки до нея. При него източника наводнява forwarding зоната с JOIN-DEMAND пакети. Когато такъв пакет стигне до станция от Geocast областта тя го връща по обратния път до източника. По този начин станциите по края на областта стават част от мрежата. Колкото по-голяма forwarding зона се избере толкова по-голяма мрежа се получава в резултат. При всичките разгледани Geocast протоколи съществува компромис между надеждност и използване на пропускателната способност.

Използването на broadcast предаване, т.е. изпращане на информация предназначена до всички станции, е изключително важна функция особено когато говорим за мобилни разпределени мрежи. Въпреки, че от всички методи за предаване този е свързан с най-голямо използване на пропускателната способност използването му не може да бъде избегнато изцяло и голяма част от маршрутизиращите протоколи разчитат на него. Затова се полагат усилия за разработване на стратегии за намаляване на излишното препредаване на информация и на нивото на възникване на колизии, като същевременно си остава условието информацията да достигне до всички станции. На приложението на broadcast пакетите може да се гледа по два различни начина. От една страна то е свързано с нарастване на състезанието за преносната среда и броя на колизиите. От друга това е най-ефективния начин да предадем една информация до всички посредством изпращането на един единствен пакет. При безжичните мрежи обаче, ако всяка станция препредаде получения broadcast пакет веднага това би довело до възникването на т.нар. *broadcast storm.* Поради тази причина съществуват различни модификации, които малко или много се справят с проблема. Най-общо broadcast протоколите се класифицират в четири групи:

- Използващи чистото наводнение: всяка станция във мрежата препраща пакета точно веднъж. Процесът продължава докато всички станции получат пакета. Използва се за мрежи с ниска средна гъстота и висока мобилност на станциите;
- Използващи коефициент на вероятност: всяка станция препраща пакета с определена вероятност. По този начин се намалява броя на дублираните пакети за гъсти мрежи. Подобрение на този метод е да се отложи препредаването с даден период и да се извърши само ако броя на получените му дубликати е по-малък от даден предварително зададен;
- Следващия клас се базира на района на покритие. Междинните станции препращат пакета само ако се намират близо до границата на района на покритие на тази от която са го получили. Тук се цели оптимизация от това че отделните зони на покритие се застъпват силно;
- Друга оптимизация е да се използва знанието на станциите за своите съседи. Всяка станция преглежда списъка от съседите включен в заглавната част на пакета и го сверява със своя, ако не може да достигне нови станции не препредава пакета.

Следващата важна стъпка в изграждането на приложения за мобилни разпределени мрежи е проектирането на транспортен протокол. С цел по-голяма универсалност, тук няма толкова голямо разнообразие от протоколи. Двата протокола на транспортния слой които се използват най-често са TCP и UDP. TCP е надеждния, връзко-ориентиран протокол, които се използва в 90% от приложенията в Интернет и следователно не може да бъде подминат и при безжичните мрежи. За да отговори на уникалните им изисквания обаче съществуват промени в дизайна му които да отразят специфичните характеристики на мобилните разпределени мрежи.

За разлика от него UDP е ненадеждния, безвръзко-ориентиран протокол, които обаче със своята простота и лекота се използва за все по-голям брой приложения за мобилни разпределени мрежи и в редица направени изследвания дава по-добра производителност от TCP. Тъй като UDP е по-малко изследвания от двата протокола в последните години има засилен интерес към него.

Поради специфичните задачи пред настоящата дипломна работа внимание е отделено само върху протоколите от приложния слой за разпространение на информация между абонати разположени върху автомобили. Това е класа на VANET (Vehicular Ad-hoc Networks) мрежите. Този тип мрежи внасят допълнителни ограничения, които правят неприложими стандартните протоколи за мобилни разпределени мрежи. Например повечето от съществуващите протоколи не отчитат факта, че в условията на автомобилно движение имаме висока степен на мобилност (до 30-40 м/сек) ограничена в рамките на съществуващата пътна инфраструктура. Тук също така отпадат ограничения като ограничена мощност и изчислителни възможности.

Идеите за приложение на мобилните разпределени мрежи в осигуряване на контрол и сигурност на пътя датират доста назад във времето, но именно скорошното широко развитие на този вид мрежи и прогреса в оборудването на автомобилите с необходимите комуникационни и изчислителни устройства дава възможност за тяхното реализиране. Затова и не е чудно, че повечето от протоколите се появяват съвсем скоро във времето. Проектът CarTALK 2000 е един от първите проекти в Европа, който се фокусира върху изграждането на нова система за подпомагане на водача базирана на комуникация между автомобилите. Сред основните целите на проекта са ранното известяване за настъпили инциденти и затруднени пътни условия за да може водачите на автомобилите по-назад да реагират своевременно избягвайки верижни катастрофи. За контрол при влизане и излизане от магистралите и на кръстовищата в рамките на градовете. За известяване за спрели автомобили при отсъствие на пряка видимост вследствие на виражи, препятствия или влошени климатични условия.

Освен с конкретни задачи се появяват и протоколи които разглеждат механизми за адаптиране на мобилността на автомобилите за целите на разпространение на информация. Пример за такъв протокол е Mobile-centric Data Dissemination algorithm for VANETs (MDDV). Както го определят авторите му, той представлява повече подход за комуникация между автомобилите отколкото конкретен протокол. При него се отчита проблема с разпокъсаността на този тип мрежи и надеждното доставяне на информация. Конкретните приложения за които би могъл да бъде използван са съответно доставяне на multicast и unicast информация до станции/група от станции с точно или приблизително местоположение. Друго приложение би могло да бъде сканирането на мрежата за определена услуга (примерно шлюз за връзка към Интернет и други). Тъй като този тип мрежи обикновено са силно разпокъсани се препоръчва използването на *opportunistic* методи. При тях междинните станции играят не само ролята на маршрутизатори, но и съхраняват пакетите в локални буфери когато няма на кога да ги предадат и ги обменят когато се появи възможност за това. Това разбира се е свързано с известно забавяне на доставянето на информацията, но е единствения възможен начин когато няма наличен път между източника и получателя. Пакетите се разпространяват от станциите най-близко по посока на получателите. За повишаване на надеждността се поддържа втора вълна от станции които са способни да продължат разпространението на информацията при отклонение или отпадане на някоя от предните. Разработени са механизми за справяне с проблема с остаряването на информацията. Всеки пакет има характерна заглавна част по която се разпознава по-актуална информация и тя заменя старата.

Протоколът Received Message Dependent Protocol (RMDP) е още един пример за приложението на мобилните разпределени мрежи за разпространение на информация между автомобилите. За определяне на местоположението се използва GPS. Подхода тук е всеки автомобил да разпръсква локална събирана информация заедно с такава научена от съседните станции за определен период от време. По този начин тази информацията се разпространява до отдалечени автомобили които евентуално биха имали полза от нея. Като следствие се получава информация за пътно събитие да достигне до автомобилите които се приближават към него посредством тези които движейки се в противоположната лента събират информация за него и я разпространяват. Не се извършва агрегиране на информацията, а всяка станция генерира информацията с фиксирана дължина. Тези късчета информация се капсулират в един UDP пакет и се разпространяват през определен период от време до текущите съседи на автомобила, които с времето се променят доста динамично поради характера на този тип мрежи. Протоколът RMDP се явява наследник на Speed Dependent Random Protocol (SDRP) при който интервалът на разпространение се е определял в зависимост от скоростта на движение (по-малък при по-висока скорост и обратно). При RMDP се цели намаляване на броя на възникналите колизии в състояния на тежък трафик, като тук интервалът на предаване се определя в зависимост от броя на получените съобщения (по-малък за повече получени съобщения и обратно). И двата подхода сами по-себе си имат преимущества и недостатъци. Евентуално тяхно комбиниране би следвало да разреши част от проблемите и да доведе до по-добри резултати.

В заключение, може да се каже, че вече съществуват достатъчен брой разработени маршрутизиращи протоколи. Въпреки, че те далеч не са най-оптималните и работата по проектирането на нови маршрутизиращи протоколи трябва да продължи, вече има база за изграждане на достатъчно добри приложения и в последно време се забелязват все повече усилия насочени натам. Мобилните разпределени мрежи състоящи се от оборудвани автомобили (VANETs) са един нов и перспективен клас, който набира популярност. Сред причините за това е появата на все по-голям брой автомобили снабдени с борд-компютри, GPS и устройства за безжична комуникация. Прилагането на разпределените мрежи в тези условия е обещаващо поради високата цена за изграждане и поддържане на съществуващите системите за контролиране и улесняване на пътното движение (Intelligent Transportation Services - ITS). Работата по изграждане на ефективни приложения за този тип мрежи обаче тепърва предстой.

ЦЕЛИ И ЗАДАЧИ

Целите и задачите на настоящата дипломна работа са както следват: проектиране и разработване на протокол за генериране и разпространение на информация за задръстване в условията на разпределена система; езика за разработване на протокола да бъде Java поради платформената независимост която осигурява; симулационно изследване на протокола с използването на реални пътни данни за град Нотингам, Англия; оценка и сравнение на резултатите с други вече съществуващи решения.

2 ПРОЕКТИРАНЕ НА АЛГОРИТЪМ ЗА РАЗПРОСТРАНЕНИЕ НА ТРАНСПОРТНА ИНФОРМАЦИЯ

Възможните приложения на разпределените мобилни мрежи (VANETs) са изключително много, общото между всички тях е, че те неминуемо трябва да се занимават с генериране на дадена информация и доставянето и до определени автомобили, като при това спазват някои ограничения, характерни за този тип мрежи. Задачите пред настоящата дипломна работа са разработване на протокол, който да разпознава настъпването на транспортни събития (задръстване, инцидент, мокър и тъмен участък и други) и да достави максимално ефективно (от гледна точка на време и използвана пропускателна способност) информацията за тези събития до автомобилите които биха имали полза от нея. В случая това са автомобилите които се движат по посока на събитието. Ако водачите получат достатъчно рано тази информация те биха имали възможността да реагират по подходящ начин (да сменят маршрута в случай на задръстване, да намалят скоростта в случай на инцидент, да пуснат фаровете или чистачките в случай на тъмен или мокър участък и други).

Един добър алгоритъм за разпространение на информация трябва да отговаря на специфичните особености на мрежата в която ще бъде използван. Затова нека да разгледаме характеристиките на VANET мрежите. При VANET мрежите автомобилите обменят информация със своите съседи (автомобилите намиращи се в рамките на техния радио обхват), за разпространение на информацията до останалите автомобили се използват маршрутизиращи протоколи за разпределени мрежи. Съществуват няколко важни характеристики които отличават този тип мрежи от останалите разпределени мрежи. Те включват:

- Характерна висока мобилност: тъй като тя води до доста динамична топология се смята за недостатък, но също така може да бъде използвана за оптимизиране на протоколите за разпространение на информация;
- Ограничено и силно предсказуемо движение в следствие както на статичната топология на пътищата така и от правилата за движение по тях;
- Комуникацията обикновено е локална: между автомобили разположени географски близко един до друг.
- Разделеност на мрежата: за разлика от останалите разпределени мрежи, тук вероятността да няма път между две станции е много голяма. Разпределени мрежи се образуват и разпадат много бързо. Това трябва да бъде отчетено от протокола за разпространение на информацията;
- Обикновено този тип мрежи имат много по-голям размер и включват много по-голям брой станции, все неща с които повечето маршрутизиращи протоколи срещат трудности;
- Липсва ограничението за мощността присъщо например за мрежите от сензори и останалите в които устройствата са на батерии;

- И не на последно място автомобилите обикновено са наясно с местоположението си посредством GPS устройства и предварително зададени цифрови пътни карти.

Поради тези си особености проектирането на протокол за този тип мрежи е изправено пред различни изисквания. Поради разделеността и високата мобилност например, не е подходящо използването на големи логически структури като поддържане на дървета и графи. По-ефективно се явява комуникацията само със съседните автомобили. От друга страна фактори като ненадеждни преносни канали, ненадеждност на самите автомобили, голямата мобилност и разпокъсаност внасят несигурност в предаването на информацията и затова се налага репликиране на информацията на няколко станции с цел подобряване на производителността и повишаване на надеждното предаване на информацията.

С цел по-голяма простото и яснота, в разискванията ще бъде разгледан само примера с разпространението на информация за задръстване. Останалите примери лесно могат да бъдат сведени до този с леки модификации, запазвайки основните принципи. При възникване на задръстване съществуват няколко ключови момента свързани с разпространението на информацията. Първия такъв който ще разгледаме е как да става регистрирането на събитие и с каква достоверност ще е взетото решение. Тук са възможни два различни подхода:

При първия подход колите които са попаднали в задръстване образуват група която е относително постоянна и с ниска мобилност. В този случай те започват да се откриват един друг и формират мрежа в която избират главна станция - автомобилът, който ще се нагърби с регистрирането на събитието и генериране на съответното съобщение. Избора на този автомобил не е еднозначен. От една страна изборът му трябва да е сравнително лесен и еднозначен, от друга това трябва и да е автомобила който ще се задържи максимално дълго в задръстването за да не се налага честа негова смяна което ще е свързано с наводнение на мрежата с пакети. Ако искаме избора на главна станция да е максимално прост може да се дефинира условие автомобила с най-малък (или най-голям) ID номер (примерно MAC адрес), да се избира за главен. В този случай обаче местоположението му вътре в задръстването е неизвестно и случайно и в някои случай това би водило до честа смяна на главната станция. Другия вариант е да се използва информация за местоположението на всяка станция и да се избира тази която е най-близко до центъра на задръстването. От една страна това усложнява значително алгоритъма за определянето на главната станция, от друга е свързано с нарушаване на конфиденциалността на местоположението на всеки автомобил. След като се избере такава станция тя решава на базата на брой автомобили, колко голямо е задръстването и генерира съобщение.

При другия подход ще използваме колите които се движат в насрещната лента за главни станции. Всяка кола локално определя дали е попаднала в задръстване на базата на следене на определени параметри като изминат път за дадено време например. Тъй като това не е много надеждно, решението може да не е винаги вярно. Когато в насрещната лента се появи автомобил той прави запитване на което отговарят само автомобилите които са регистрирали дадено събитие. На база на получената информация отсрещната кола взима решение дали да повярва, че е възникнало задръстване. Ако реши, че е възникнало на база на брой получени отговори и брой ленти в срещуположното платно се определя степента на задръстването и се генерира съобщение.

Детерминираността при избора на автомобил който да генерира съобщението прави втория подход по-обещаващ. Поради тази причина именно той е избран като подход за генериране на съобщението в настоящата дипломна работа. В самия подход е заложено агрегирането на информацията без да са необходими допълнителни усилия за това.

Другия ключов момент е определяна на автомобилите, които биха проявили интерес към генерираната информация. Когато говорим за задръстване, несъмнено това са автомобилите които се движат по посока на задръстването и евентуално възнамеряват да преминат през него. Това обаче съвсем не решава въпроса с избора на зона на разпространение. Ако избраната зона е много малка тогава може да се окаже, че информацията се получава единствено от автомобили които нямат алтернативи маршрути вече и следователно за тях тя ще е повече информативна отколкото полезна. Ако разпространяваме обаче информацията в твърде голяма област, тогава несъмнено тя ще достигне и до голям брой автомобили, които въобще нямат намерение да преминават през задръстването. По този начин разпространението ще е безполезно и ще сме отнели излишно от ресурсите на мрежата. Идеалния вариант е информацията да се разпространява до няколко (примерно 2,3) равностойни кръстовища назад така че да съществува възможност за промяна на маршрута. Поради разнообразния характер на пътните мрежи това не може да бъде фиксирано нито като разстояние, нито време. Затова подхода които е избран тук е към всяко съобщение да се обвържи число, наречено ‘време на живот’ (TTL, различно от TTL полето в IP хедъра) което да се намалява на всяко кръстовище в зависимост от това колко равностойни маршрута предлага. Когато времето на живот стане равно на нула, съобщението спира да се разпространява. Първоначалната стойност на TTL полето се задава от станцията която генерира съобщението за събитието. Тази стойност може да варира в зависимост от степента на задръстването, по-малките задръствания могат да се разпространяват до по-кратки разстояния, а по-големите до по-далеч. По този начин определихме една динамична област обхващаща автомобилите които ще получат съобщението, размерите на която зависят от различни фактори и отразяват максимално близко автомобилите които биха проявили интерес към информацията.

Следващата стъпка е определяне на станциите които ще извършват самото разпространението на генерираните съобщения. Тук възможностите отново са две. Едната възможност която съществува, донякъде е обвързана с първия разгледан подход за генериране на съобщенията. Колите участващи в задръстването генерират съобщение за задръстване и го препращат назад използвайки някой от съществуващите маршрутизиращи протоколи. Това е може би най-бързия вариант за разпространение на съобщенията. Пред него обаче съществува един значителен проблем, който е свързан с една от характеристиките на VANET мрежите, а именно факта, че те са силно разпокъсани и често няма връзка между крайните станции. В такива ситуации единственото възможно решение е да се използват т.нар. *opportunistic forwarding* алгоритми. При този вид алгоритми когато дадена станция не може да продължи разпространението на съобщение поради липса на следваща станция в обхвата й тя съхранява съобщението и го препраща едва когато се открие възможност за това. Правени са редица изследвания който са доказали ефективността на този метод при липса на свързаност между крайните станции. За ситуацията която разглеждаме, едно подобно съхранение би довело до връщане на съобщението обратно назад спрямо посоката на разпространение в следствие на движението на автомобилите. В този случай много по-удачно ще е да използваме автомобилите в насрещната лента за транспортиране на информацията. По този начин дори когато няма възможност за предаването й напред и се налага даден автомобил да съхрани информацията до появяването на подходяща възможност информацията ще продължи да се разпространява благодарение на движението на автомобила в същата посока. Това е особено ефективно като отчитаме факта, че автомобилите се характеризират с висока скорост на движение.

Генерирането и разпространяването на информация чрез автомобилите от насрещната лента, позволява също така да се поддържа статус на събитието, тъй като всеки следващ автомобил ще разнася по-нова информация. Това предимство обаче може да бъде недостатък ако всяка станция идваща насреща генерира и разпространява информация. В този случай, ще е често срещано явление по-нова информация да застигне и изпревари по-стара. Тогава възниква проблем с актуалността на информацията и необходимост от мерки за предотвратяването на този проблем. Първото решение е когато автомобила от насрещната лента направи запитване, всички останали негови съседи от същата лента, които са чули запитването да преминат в пасивен режим и да не изпращат и те запитвания в същия момент. По този начин се решава проблема когато отсреща идват група от автомобили а не един единствен. Поради ограничения обхват на радиопредавателите обаче е възможно много скоро да се получи повторно запитване от автомобил в насрещната лента. Тъй като той няма как да разбере за предхождащия го, тук сме избрали следния механизъм за ограничаване на генерирането на нови съобщения. Когато един автомобил получи запитване и отговори на него, той стартира таймер до изтичането на който няма да отговаря на нови запитвания. По този начин когато се получи второто запитване, малко след първото няма да се получат отговори и станцията ще реши, че няма възникнало събитие и няма да генерира съобщение.

Следващия проблем който трябва да бъде разрешен е свързан с надеждното предаване на информацията. Тъй като автомобилите сами по себе си не са особено надеждни е необходимо да репликираме съобщението на няколко станции. В разглеждания случай това би могло да стане без излишни усилия. Характерно за VANET мрежите е, че автомобилите се движат на групи вследствие на правилата за движение (примерно на светофарите имаме събиране на автомобили до светване на зелена светлина). По този начин когато един автомобил направи запитване, той доста често ще има около себе си съседи движещи се в същата посока. Когато станцията генерира съобщение, именно тези съседи биха могли да разпространяват негово копие. По този начин получаваме и едно допълнително предимство, което обаче е изключително важно. Кръстовищата обикновено са места където автомобилите се разделят, част от тях поемат в една посока, друга част поемат в друга. Предимството да имаме копирано съобщението на всеки от автомобилите от групата е че след всяко кръстовище съобщението ще поема по различни пътища и ще се увеличава обхвата на района на разпространение.

Аналогично изниква въпроса как да се реализира протокола така, че от една страна да имаме ефективно разпространение от гледна точка на ресурсите на мрежата като пропускателна способност, брой възникнали колизии изискващи препредаване, разумно използване на изчислителните възможности на автомобилите и на ресурсите памет. От друга страна от протокола се изисква максимално бързо и надеждно доставяне на информацията. Това е свързано с копиране на съобщението на по-голям брой автомобили и по-често предаване. От трета страна е проблема с актуалността на информацията, който може да се окаже доста съществен и да повлияе на цялостната представяне на протокола и да компрометира останалите показатели. Очевидно трябва да се приеме известен компромис между всички тези изисквания. За целите на маршрутизирането (разпространението) на съобщенията се използва подхода предложен от авторите на MDDV протокола. Идеята е за всяко съобщение да се поддържа статус на станцията която го съхранява. Това поле за статуса ще се променя динамично и ще определя кога една станция активно ще разпространява дадено съобщение и кога само ще го съхранява с цел повишаване на надеждността. Съответно това поле ще приема две стойности – ACTIVE и PASSIVE.

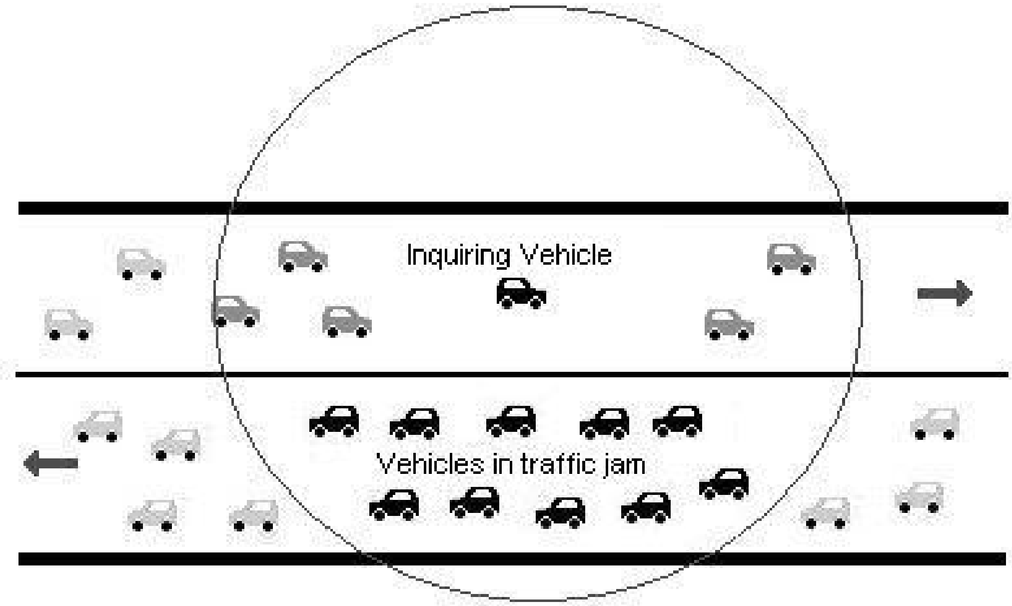
След като бяха разгледани задачите, които трябва да се решат от новия протокол, разгледани бяха подробно проблемите свързани с различните етапи от работата му както и конкретните възможности за решението им остава да се разгледа цялостно работата на предложения протокол. Преди това обаче са разгледани няколко допускания направени при проектирането му:

- Приема се, че всеки автомобил е снабден с GPS устройство. Това допускане е изключително важно за работата на предложения протокол;
- Допуска се, че съществуват предварително инсталирани цифрови карти. Посредством тях и GPS устройството, всеки автомобил е в състояние да определи местоположението си на картата и всички последващи оптимизации произтичащи от това;
- Допуска се, че всеки автомобил е снабден с устройство за безжична комуникация в ограничен обсег. В тази дипломна работа сме избрали то да е по стандарта IEEE 802.11, с радиус на действие 200-250 метра.
- Приема се, че пътищата са двупосочни. Предишните допускания са характерни и за останалите протоколи за VANET мрежи, докато последното е специфично за разглеждания. Причината за това допускане е важната роля на автомобилите в насрещната лента за генерирането и разпространението на съобщенията.

Както вече беше споменато по-горе, всеки автомобил, локално и независимо от останалите ще проверява за възникването на дадено събитие. Когато такова бъде регистрирано, информация за него ще се пази в локалните буфери и ще бъде докладвано само при поискване. В следващите редове ще разгледаме принципите за откриване на различните видове събития.

Един сигурен показател за възникването на задръстване е относително малкото изминато разстояние за определен интервал от време. Нека снемаме данните за изминатото разстояние през интервали от 10 секунди (Δt_1=10s). Тук времето от 10 секунди е избрано компромисно между точност на измерването на изминато разстояние и използването на изчислителните ресурси на станциите. Решението за това дали е възникнало задръстване ще взимаме, използвайки изминатото разстояние за последните 3 минути. При първоначалното потегляне на автомобила, ще задаваме някакво средно разстояние. Това няма да доведе до голяма неакуратност понеже рядко автомобилите веднага след потеглянето си се включват в задръстване. Ще приемаме, че е възникнало задръстване ако разстоянието изминато за този период е по-малко от дадено разстояние $d_{1,}$ където d_1 е равно на 15% от разстоянието което автомобила би изминал ако се движи със средна скорост. Тъй като средната скорост е различна за различните държави и класове пътища, то и избора на d_1 ще зависи от тях. Периода от 3 минути е избран така, че да се отчете възможността за спиране на няколко светофара и това да не се възприеме погрешно за възникнало задръстване. За изчислението на изминатото разстояние се използват данните които се получават от GPS устройството. Местоположението се отчита всяка секунда и по теоремата на Питагор се изчислява изминатото разстояние (т.нар. Евклидово разстояние), което ще се натрупва. Периода от една секунда е подбран, защото при високи скорости автомобилите се движат предимно праволинейно и тогава грешката от измерването на разстоянието ще е минимална, докато при по-ниски скорости интервалът от секунда ще е достатъчно малък за да даде отново добра акуратност на измерването на изминатото разстояние. Мокри и хлъзгави участъци се разпознават по буксуване на гумите. Това може да се засече лесно по включването на ABS системата на автомобилите. Дъждовни и тъмни участъци ще бъдат засичани по пуснати чистачки, съответно фарове. Инциденти ще се засичат при отваряне на въздушните възглавници например. Различните типове събития ще се съхраняват различен период от време след като вече няма показание че са валидни. Примерно, информацията за задръстване ще се изтрива веднага след като вече не е валидно условието за това, докато при другите събития има смисъл да продължим да ги пазим известен период и след това.

За илюстрация на работата на протокола ще използваме фигура 7. Приема се, че е възникнало задръстване и автомобилите попаднали в него са разбрали за наличието му. На фигурата това са автомобилите в тъмно с посока на движение наляво. По отсрещната лента се движат автомобили и един от тях, чиито ред е дошъл изпраща запитващо съобщение. В съобщението той включва своето местоположение и посока на движение. Съобщението се получава от всички автомобили в обсега му на излъчване. Автомобилите които се движат в същата посока както запитващия автомобил няма да излъчват запитващи съобщения за определен период. Автомобилите от отсрещната лента които имат регистрирани събития започват да отговарят на запитването един по един. В разглеждания случай автомобилите са попаднали в задръстване и следователно запитващата кола ще получи съобщения за задръстване. След това тя ще вземе решение за достоверността на възникналото събитие, като отчита броя на получените отговори и броя на лентите в насрещното платно. В зависимост от това тя генерира или не съобщение за възникване на задръстване. Тук се извършва филтрирането на грешната информация, която може да се е появила в следствие на не напълно надеждния метод използван от всеки автомобил.



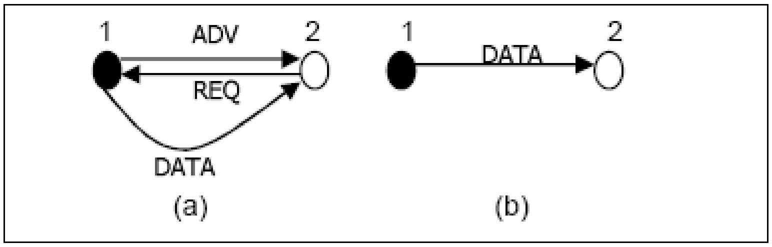
фиг. 7: Процес на запитване за регистрирани събития

Ако се прецени, че задръстване наистина съществува се генерира съобщение със следната структура:

- Текущата позиция на автомобила пренасящ съобщението
- Времето на регистрация на събитието
- Местоположението на задръстването (сегмент от пътя)
- Посоката в която е възникнало
- Статус поле (Активен, Пасивен)
- Код на събитието
- Време на живот на съобщението (TTL)
- Други данни

Имената на повечето от полетата говорят сами за предназначението им. Текущата позиция е нужна за да може даден автомобил да прецени дали се намира по-напред от този от когото е получил съобщението и следователно да започне да го разпространява. По следващите три полета може да бъде открита по-актуална информация за едно и също събитие при което тя да измести старата. Полетата за статус и време на живот са разгледани по-подробно по-надолу. Кодът на събитието служи за разграничаване на задръстването от останалите възможни събития. Тъй като конкретно е разгледан само случая на задръстване, а също и за бъдещи промени е оставена възможност за добавяне на още полета.

Съществуват два метода за обмен на съобщения, които са в пряка зависимост от вида на съобщението затова ще ги разгледаме тук. При първия (фиг. 8 (а)) станцията която иска да предава, първо рекламира данните изпращайки само част от съобщението. Ако получи искане чак тогава изпраща цялото съобщение. При втория подход (б) направо се изпраща цялото съобщение. Първият подход е по-подходящ ако имаме големи съобщения с относително малки мета данни. Тъй като в конкретния случай това не е така е избран втория подход.



фиг. 8: Методи за обмен на съобщенията

За разглеждания случай се приема, че е регистрирано задръстване. Запитващата кола генерира съобщение за задръстване, в което наред с останалите полета маркира себе си като активна в полето за статуса. След което тя разпраща така генерираното съобщение за да може останалите автомобили в нейната лента, които бяха преустановили запитванията да копират съобщението, маркирайки себе си като пасивни по отношение на него и да продължат нормалният процес на запитване на отсрещните автомобили.

Времето през което всеки автомобил прави запитване е зададено като Δt_2 и е системен параметър. Определянето на подходяща стойност за Δt_2 е свързано с компромис между често заемане на честотната лента и ранното откриване на задръстване.

След като бъде генерирано ново съобщение, бъдат определени активния и пасивните автомобили, които ще го разпространяват остава да се разгледа как ще става разпространението и доставянето на съобщението да останалите автомобили. Полето статус, както вече беше споменато по-горе отразява състоянието на автомобила за конкретното съобщение. Един автомобил може да бъде активен за едно съобщение и пасивен за друго. Съобщението се разпространява само от активните за него автомобили. Процесът на разпространение се осъществява чрез периодично разпръскване на съобщението, използвайки broadcast адрес. Периода на разпространение е Δt_3, където Δt_3 е системен параметър и изборът му е компромис между това съобщението да се получава повече от веднъж от някои автомобили (при по-малко Δt_3) и да съществуват автомобили които не го получават изобщо (при по-голямо Δt_3).

Причината да има пасивни автомобили е за да се повиши надеждното предаване на съобщението в случай на отпадане на активната станция и за да се разшири района на автомобилите получили съобщението. Същевременно тъй като пасивните автомобили не разпространяват съобщението имаме по-малко количество обменяни съобщения. В следващите редове е разгледано кога един автомобил се маркира като активен, кога преминава в пасивно състояние и кога изтрива съобщението.

Възможностите един автомобил да премине в активно състояние за дадено съобщение са три. Първата вече беше спомената, автомобила който генерира съобщението маркира себе си като активен. Втората е когато автомобил, който допреди това е бил в пасивно състояние престане да ‘чува‘ съобщението от активния за период по-голям от периода на разпространение Δt_3. Тогава се предполага, че е отпаднал и тогава станцията променя стойността на статус полето на активен и започва да разпространява съобщението. Именно по този начин когато след кръстовище част от автомобилите поемат по различни пътища ще разширят района на разпространение на съобщението. И третия възможен сценарии един автомобил да стане активен за дадено съобщение е когато получи съобщение от автомобил който се намира по-назад по посоката на движение. Тогава той копира съобщението и маркира себе си като активен по отношение на него.

Възможностите един автомобил да се маркира като пасивен за дадено съобщение са две. Първата е в следствие на това, че автомобила е бил в обсега на запитващата, копирал е генерираното съобщение и се е маркирал като пасивен. Втория случай е когато активен автомобил получи същото съобщение от автомобил който се намира пред него по посоката на движение. В този случай той преминава в пасивно състояние като преди това уведомява своите пасивни които изтриват съобщението. Тук може да се стигне но нестабилно състояние водещо до честа промяна на статуса на втория автомобил, поради тази причина една възможна оптимизация би била втория автомобил да си остане активен докато разстоянието с предния не стане по малко от 70% от максимума на обхвата.

И накрая е разгледано кога съобщенията се изтриват. Едно съобщение се унищожава първо когато TTL полето му стане равно на нула. По този начин се контролира региона на разпространение на съобщението. Втората причина да се изтрие едно съобщение е когато една пасивна станция получи нареждане за това от активната. И третата причина е когато се получи съобщение с по актуална информация, т.е. по-нови съобщения за същото сьбитие ‘догонят’ по старите. В този случай е излишно да продължаваме да разпространяваме старото съобщение.

За реализиране на предложения алгоритъм е избран програмния език JAVA. Една от причините е платформената независимост на приложенията реализирани на него, а другата е леката и интуитивна работа с нишки. За работата на нашия алгоритъм последното е от особено значение, тъй като вече разгледахме, че той се състои от няколко отделни задачи, които се изпълняват паралелно. Някои от задачите ще имат по-висок приоритет, съответно и нишката която ги реализира също ще е с по-висок приоритет, докато други ще имат нормален приоритет. Името което сме дали на протокола е Counter-flow Initiated Generation and Dissemination Protocol (CIGDP), което име отразява основните характеристики на протокола. В следващите редове е разгледано функционално описание на отделните алгоритми по които работи CIGDP протокола. След което на фигура 9 е показана функционална схема на работата и отделните модули на протокола.

Алгоритъм за изчисление на разстоянието:

```
Location_new = get_location() from GPS;
distance = calculate_euclidean_distance( location_new , location_old );
Push distance to distances' array
Location_old = location_new;
Sleep ( 1 second );
```

Алгоритъм за откриване на задръстване

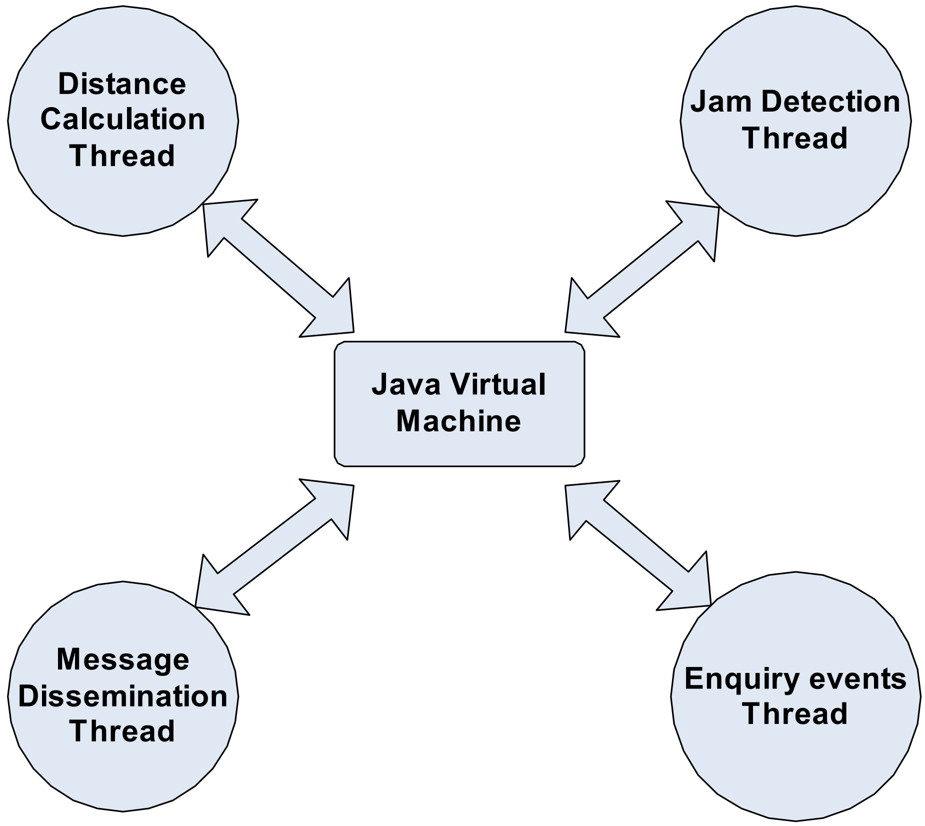
```
for i = 0 to 180 do
   d = d + distances[ i ];
If d <= d1 then jam = true;
else jam = false;
Sleep ( Δt1 seconds);
```

Алгоритъм за отправяне на запитвания

```
send (enquiry message) to all neighbors;
while receiving responses do
   responses_count++;;
If responses_count > (max * Number of lanes)  then
   generate_message (jam);
Sleep ( Δt2 seconds);
```

Алгоритъм за разпространение на съобщенията

```
If status of message m is Active then
   If new message m1 is received and m1 is new version of m then
      Erase (m);
      m.status = passive;
   Else
      send (m) to all neighbors;
      Sleep ( Δt3 seconds;)
Else if new message m2 is received and m2.status = active then
         Sleep ( Δt3 + random offset );
      Else
         m.status = active ;
         Sleep (Δt3 seconds);
End if
```



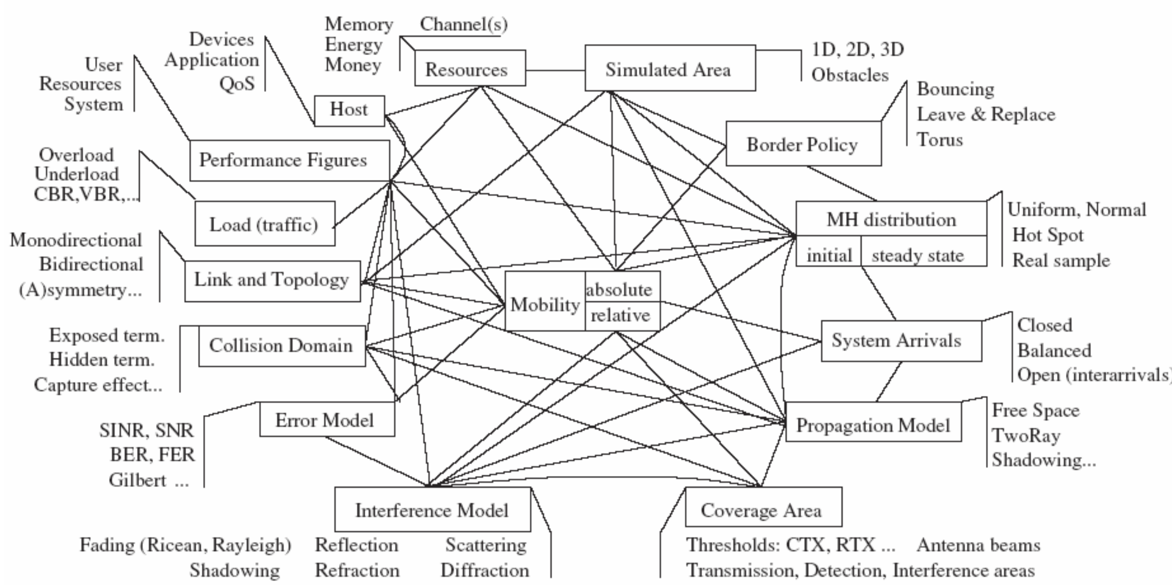
фиг. 9: Функционално описание на съвместната работа на отделните модули на CIGDP протокола. Паралелизирането се осъществява от виртуалната машина

3 ИЗБОР НА СЦЕНАРИЙ ЗА ВЕРИФИКАЦИЯ НА ПРОТОКОЛА

Когато се говори за верификация на нови системи и протоколи или подобрения в дизайна на вече съществуващи решения за компютърните мрежи съществуват най-общо три възможни сценария да се направи това. Всеки от тях има своите недостатъци и предимства и достоверността на получените резултати е различна и градира колкото по-близо до реалните ситуации се доближава. Първият сценарий е чрез използването на математическо моделиране и анализ. При този метод се създават математически модели и аналитично се определят характеристиките и очакваните резултати. Този метод е най-лесен за реализиране, дава някаква яснота върху работата на протоколите, но от трите метода той е с най -далечни от реалните резултати. В повечето случай резултатите получени по този начин се отнасят за конкретния модел и не дават възможност за голяма детайлизация което прави трудно оценката и сравнението между две мобилни безжични системи и услугите които те предлагат. Именно възможността за сравнение е един от основните инструменти на изследователите на нови протоколи и концепции, който дава база за подобрения и оценка на получената производителност. Това е особено вярно когато се отнася за мобилните разпределени мрежи, при които по-голямата детайлизация е от изключително значение за оценката на работата. Поради тези причини моделирането в повечето случай не е особено популярен вариант за оценка на този тип системи. Понякога се прилага като най-първата фаза от оценката на нов протокол или модел. Следващия възможен сценарий използван за верификация е т.нар. *real-world deployment*. При него системите и протоколите се тестват в реални житейски ситуации. Той е особено възможен благодарение на развитието на мобилните изчислителни станции и достъпността на устройствата за безжична комуникация (IEEE 802.11). Разбира се тъй като е най-близко до реалността, този метод ще дава и най-достоверни резултати. Той обаче няма предимствата на третия метод (симулирането), който го превъзхожда и от трите има най-голяма популярност в изследванията на нови протоколи. Въпреки това, изпитването в реални житейски ситуации си остава основния способ за определяне на ефективността и работата на разпределените мобилни мрежи и обикновено е последната фаза на тестване на протоколите доказали се като успешни след верификация с останалите методи. Последния сценарии е верификация на протоколи и архитектури чрез методите на симулирането. Симулацията се е наложила като най-ефективния инструмент на изследователската общност за изпитване, оценка и извличане на критични за работата параметри и характеристики. Това което симулацията дава като предимство пред реалните житейски изследвания са бързина на подготовка на изследването, повторяемостта на изследванията при еднакви входни параметри, чрез което се постига изолиране на даден параметър и оценяване на представянето на изследвания протокол за различни негови стойности. Осъществяването на два еднакви експеримента е много трудно и практически невъзможно при изследванията в реални житейски ситуации. Друго предимство е възможността чрез симулация да се проведе експеримента с голямо разнообразие от сценарии и параметри за да се получи по-пълна оценка на изследвания модел. Несъмнено предимство на симулацията пред житейските изследвания е възможността да се включат стотици мобилни станции в експеримента, които да се движат в голям географски район, нещо което е непосилно за реално изследване. Въпреки своите предимства съществуват и редица недостатъци пред симулационното изследване които са свързани повече с несъвършенството на съществуващите симулационни инструменти, които са налични за използване. Една детайлна симулация в повечето случай има повишени изисквания към изчислителната мощ на машината на която се провежда експеримента. Например често срещан е случая едно симулационно изследване да изисква часове и дори дни за да се проведе. Изискванията към ресурсите памет също са големи. За целта се разработват оптимизации към съществуващите симулатори, както и се разработват нови които да ускорят процеса на симулация. Много от симулаторите вече поддържат разпределена паралелна работа между няколко машини свързани в мрежа за подобряване на времето и увеличаване на мащаба на симулирането, включвайки по-голям брой станции движещи се върху по-големи райони. От друга страна достоверността на резултатите получени от симулация е в голяма зависимост от това колко добре симулационната среда се доближава до действителността. В най-голяма степен това касае моделите на движение на станциите и моделите за разпространение на сигналите между тях. Няколко проведени експеримента доказват необходимостта от достоверен модел на движение.

От разгледаните възможности за верификация на предложения протокол, отчитайки специфичните им особености е избран варианта със симулиране. Тъй като мобилните станции в конкретната мрежа представляват автомобили, които се характеризират с висока мобилност, голям географски обхват и специфичен модел на движение този вариант е най-подходящия за оценка на работата на протокола. Въпреки голямото разнообразие от симулатори, които са налични за използване съществуват някои общи характеристики, които са валидни за получаването на детайлни и по-близки до реалността данни. При симулационното изследване на разпределените мобилни мрежи всеки експеримент трябва да отговаря на някои фактори и условия на моделирането. Тези условия трябва да бъдат ясно и детайлно дефинирани и може да се включват в различните нива на симулацията. Пример за такива условия са обхвата на предаване, ниво на мощността, отслабване на сигнала, източниците на трафик, големината на буферите, движение на мобилните станции и ограничение в топологията, разпространение на сигнала, препятствия, грешки в предаването и много други. За всяко условие се създава отделен модел, като отделните модели комуникират и зависят един от друг. На фигура 10 е показана една сравнително пълна картина на основните модели и взаимодействието помежду им. От фигурата се вижда, че централно място при моделирането има движението на мобилните станции. Всеки симулационен инструмент трябва да разполага с решение на отделните нива и колкото по-ефективни са решенията, толкова по ефективно е и самото симулиране.

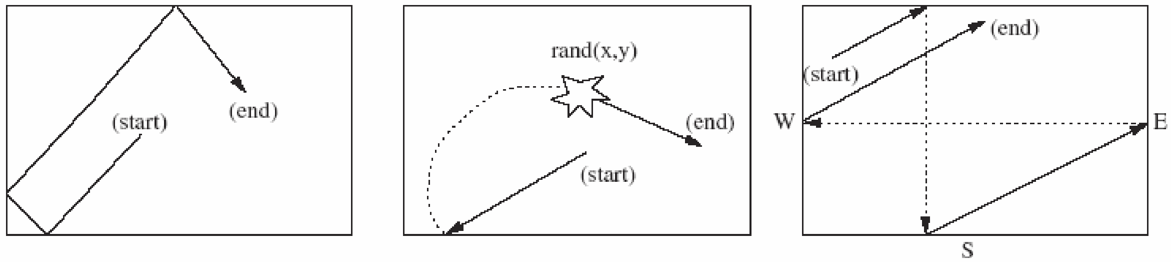
В следващите редове ще разгледаме по-важните модели, как са моделирани и какви предимства и недостатъци имат. След това ще разгледаме няколко съществуващи инструмента за симулиране за безжични мрежи, ще ги сравним и оценим и ще изберем един от тях с помощта на който ще реализираме симулирането на CIGDP протокола.



фиг. 10: Карта на моделите

Първият проблем които ще разгледаме е за избора на симулационна област. Тя винаги е с фиксирана големина и има някакви предварително зададени граници. Тя може да бъде в едно или няколко измерения (1-D, 2-D, и т.н.) и обикновено се представя в картезиански координати. Когато говорим за VANET мрежите данните обикновено са зададени в географски координати и следователно след като бъде избрана областта е необходимо да се преобразуват до картезиански. Областта освен граници може да съдържа различни препятствия които възпрепятстват както движението така и разпространението на сигналите. При движението на мобилните точки е възможно траекторията им да се пресече с границите на областта. В този случай съществуват няколко разработени правила за решение на проблема:

- Метода на отражението (фигура 11а): при този метод когато една мобилна станция достигне границата на симулационната област траекторията и се изменя както лъч отразен от повърхността на огледало. Този метод е подходящ когато е необходимо броя на станциите да остане постоянен;
- Метода на напускането със замяна (фигура 11б): при него когато една станция достигне границата тя се изключва от симулацията и на нейно място се появява нова със случайна траектория и разположение вътре в симулационната област. Този метод има за цел да разреши проблема със скупчването на мобилните станции предимно в центъра на полето при горния случай;
- При тороидалния метод (фигура 11в) станцията се включва отново в симулационната област така сякаш областта е затворен тороид и срещуположните и граници съвпадат. Тук отново се решава проблема със съсредоточаване на станции в центъра, но понеже не създаваме нова станция, то старата си запазва цялата история.



фиг. 11: Правила за пресичане на границите на симулационното поле

Следващия проблем който трябва да бъде решен от симулиращата програма е с обхвата и разпространението на сигналите при безжичните мобилни станции. Мобилните станции излъчват посредством *omni-directional* антени, които излъчват равномерно във всички посоки. Обхвата обикновено е фиксиран, но когато имаме източници на батерии е добре да се създаде модел на намаляващ обхват в зависимост от мощността на станцията. Друг важен момент е наличието на препятствия, един максимално близък до реалността модел трябва да отчете наличието им като намали размера на обхвата. Това води до т.нар. ‘тунелен ефект, при който станция разположена между две сгради има нормален обхват напред и назад, но в страни, обхвата и е силно намален. За постигане на по-голяма достоверност трябва да съществува модел отчитащ възникването на интерференция и грешки при предаването вследствие на мобилността на станцията.

Изграждането на връзки между станциите е друг ключов момент при моделирането на симулацията. Това е така поради мобилността, която може да бъде както абсолютна така и относителна. Нека вземем две станции A и B, тогава са възможни три сценария:

- A и B са извън обхвата на другия, тогава за комуникация между тях трябва да се търси връзка чрез други станции посредством обръщение към маршрутизиращ протокол.
- A е в обхвата на B, но не и обратното. Дължи се обикновено на различни запаси от енергия и в този случай топологията е под формата а насочен граф.
- A и B са в обхвата на другия. Това е най-предпочитания случай, понеже се реализира най-лесно. Много симулатори предполагат, че винаги е изпълнено условието двете станции да имат един и същ обхват. За случая с VANET мрежите може да се приеме, че е изпълнено това условие понеже нямаме ограничения в мощността.

Моделът на движение, както вече споменахме по-горе заема централно място при симулацията на мобилни разпределени мрежи. Това е така защото последствията от мобилността засягат всички останали слоеве от комуникационния модел. Няколко направени изследвания показват, че без близък до реалността модел на движение не можем да се надяваме на достоверност на получените резултати. Един добър модел на движение трябва да отчете голям брой детайли и особености от по-ниско ниво. Най-общо моделите на движение се разделят на два вида: с файл и синтетични.

Методите с файл, т.нар. *trace files* имат голяма достоверност и акуратност, въпреки това техен недостатък е големината на файловете, която зависи от броя на станциите и от избрания период на дискретизация. Друг проблем е, че поради малкия брой работещи системи за разпределени мобилни мрежи е трудно да се съберат данни за движението на станциите в тях.

Синтетичните методи от своя странна имат за цел да пресъздадат реално движение на станциите. Едни от първите синтетични методи са използвали математическо представяне на движението. Самото движение е било далеч от реалността, но тези модели се срещат в някои симулации и до днес поради възможността за точното повторение на движението при няколко последователни експеримента. Другата голяма група синтетични методи са тези на случайното движение. Те имат за цел да изпитат максимално даден протокол създавайки условията на най-лош сценарии. Наскоро се появяват и разширения на тези методи, в които чрез добавянето на ефекта на взаимосвързаността, ограниченията в движението, груповото поведение с цел повишаване на реалността в симулациите. Различаваме три варианта на методите със случайно движение:

- Модели позволяващи на мобилните станции движение в цялата симулационна област без ограничения. При тях се използват псевдослучайни алгоритми за избор на скорост и посока;
- Модели внасящи ограничение в движението като улици и стени, но все още позволяващи псевдослучаен избор на посока и скорост на всяко кръстовище;
- Модели при които имаме предварително зададени пътища. Най-детерминираните от трите варианта.

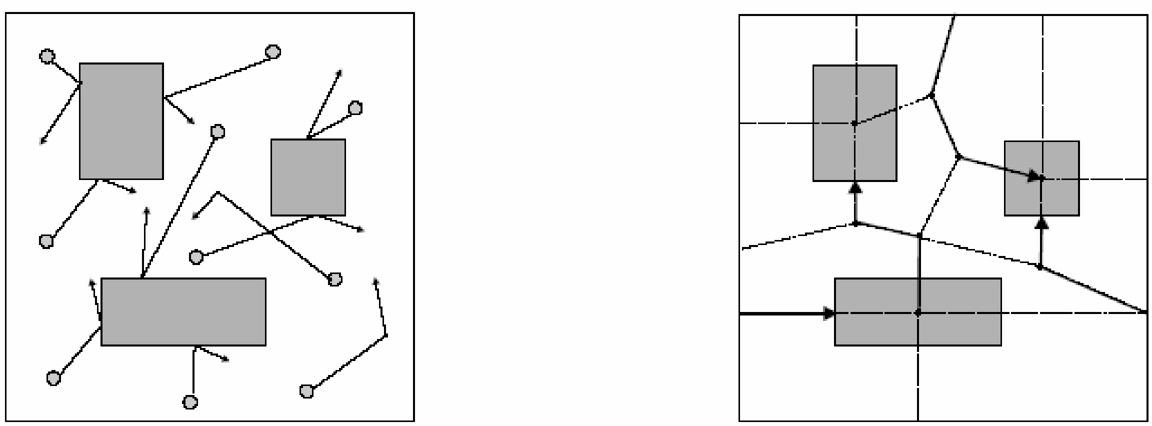
В следващите редове са разгледани някои от най-често използваните модели на движение в симулациите. В последно време се забелязва прогрес в моделите на движение с цел по-голям реализъм.

Модел на случайно движение (Random Mobility Model): този модел се явява дискретизиран вариант на известното Брауново движение. Движението е напълно непредсказуемо без наличието на никаква взаимосвързаност. Всяка мобилна точка си избира произволна посока, скорост която се променя през определен интервал. При това няма връзка между предишно избраните стойности на параметрите. Това е много нереалистичен модел и се използва като лесен за реализиране и за изпитване на протоколите при най-лошия случай (worst-case). Най-често се прилага за двуизмерни пространства (2-D). На база на този модел съществуват различни разновидности които да допринесат за повишаване на реализма на движението. Един от тях е следващия разгледан.

Моделът Random Waypoint Model доразвива горния модел с добавянето на периоди на движение и паузи. Всяка станция избира произволни скорост и дестинация. При достигането и се прави малка пауза преди да се изберат нови случайни стойности на параметрите на движение. Тук не са нужни правила за границите на симулационната област, защото станцията може само да ги достигне но не и пресече. Този метод е доста предпочитан за симулация но като предшественика си страда от проблемите със скупчване на станциите в центъра на симулационната област, липса на взаимосвързаност и зависимост между избираните параметри.

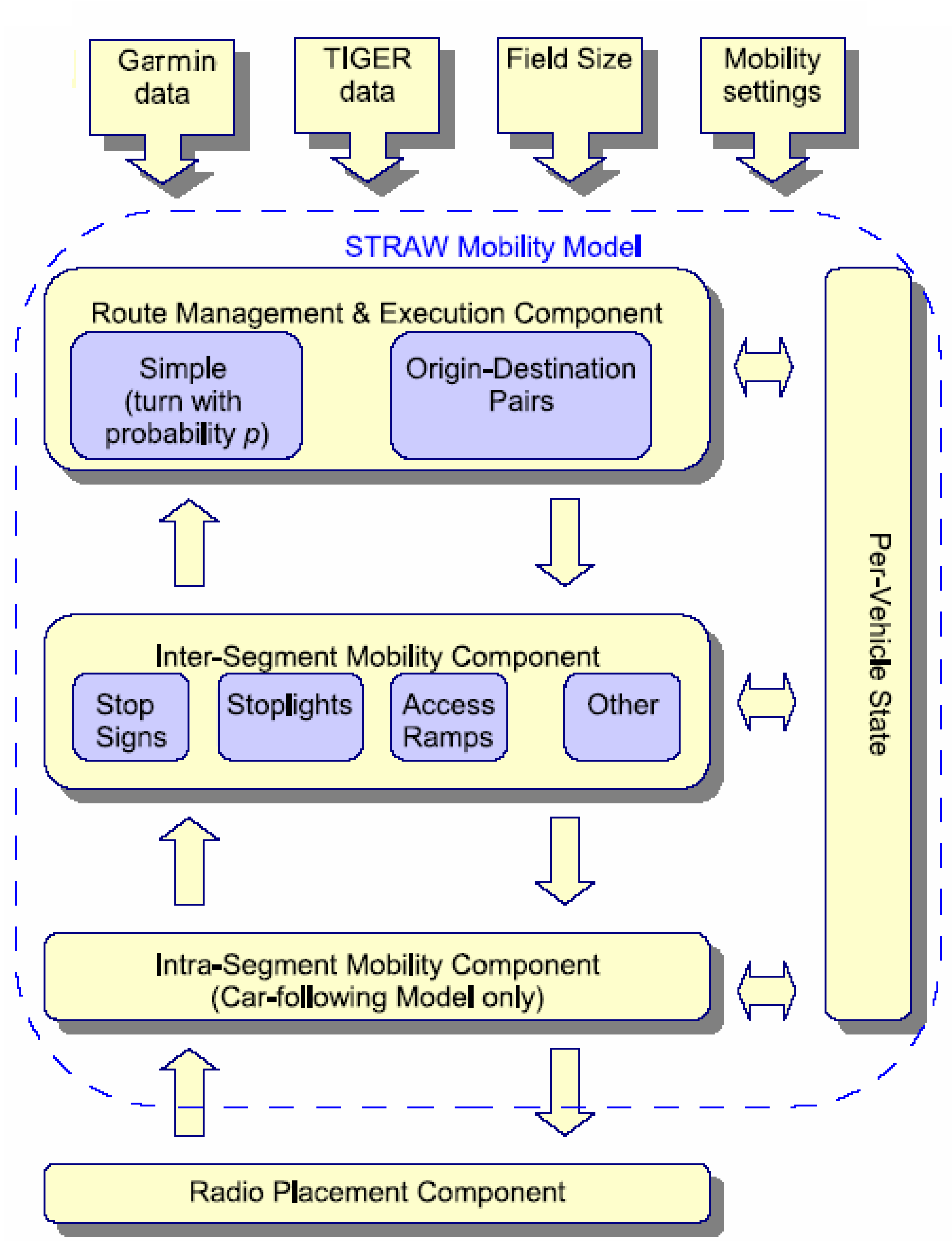
Моделът City Section Mobility Model има за цел да обедини предходния модел със моделът Манхатан. Тук имаме наличието на ограничения типични за градски условия като пътища кръстовища и стени. Станцията си избира произволна дестинация и се предвижва до там следвайки най-линейния маршрут. Когато пристигне, прави малка пауза и избира нова дестинация. Разновидност на този модел е Graph-based Mobility Model при който симулационната област е разгледана като граф, върховете на който представляват кръстовища, а ребрата пътищата между тях. На всяка стъпка се избира произволен връх и се изчислява най-краткия път до него.

Следващия модел включва възможност за поставяне на препятствия. При Obstacles Mobility Model първоначално се поставят сградите. Те както е и в действителността ограничават движението на мобилните станции и разпространението на сигналите. За целта първо трябва да имаме зададено географското разположение на сградите. На база на тях се изчисляват пътищата посредством диаграмите на Вороной. Те се поставят на равни разстояния от две съседни сгради. В самите сгради се поставят ‘врати’ позволяващи влизането и преминаването през тях. За моделиране на маршрутите се избират начална и крайна точка и се изчислява най-краткия по разстояние път. Този модел е добър за симулиране на предвижването на хора с мобилни устройства между сградите и дава добра достоверност на получените резултати. На фигура 12 са показани моделите на движение с и без предварително определени пътища.



Фиг. 12: Схеми на модела на движение с препятствия

Следващия модел на движение има за цел да реализира максимално близко до реалността движението на автомобилите в пътната мрежа. Моделът Street Random Waypoint (STRAW) постига това чрез използването на реални пътни данни и включването на голям брой действителни правила за движение. Движението на отделните автомобили освен ограничено в рамките на улиците и пътните знаци е зависимо от останалите участници. Използва се популярният модел на ‘следвай предходния автомобил’. При този модел при наличието на автомобил отпред се ускорява до достигането му след което скоростите постепенно се изравняват. Отразява се възможността пътищата да имат няколко ленти за движение във всяка посока. При пресичане на кръстовища се спазват правилата за предимство или сигнализацията на светофара ако има такъв. Моделът включва два варианта за избор на маршрут на движение. При единия на всяко кръстовище се избира произволна посока на движение. Възможно е индивидуално задаване на вероятност автомобила да завие в една или друга посока. Вторият вариант осигурява по-голяма детерминираност на маршрутите. При него се задават двойка местоположения начало-край (origin-destination OD). Пътищата се изчисляват предварително посредством алгоритъм, който може да бъде оптимизиран за време или разстояние. Фигура 13 илюстрира отделните компоненти на STRAW и връзката между тях.



фиг. 13: Обща блок-диаграма на моделът на движение STRAW

В следващите редове са разгледани трите основни метода на симулиране и разликата между тях: аналогов, дискретен и хибриден. Разгледани са и някои от най-популярните инструменти за симулиране.

При аналоговите методи, симулационното време и промените на състоянието протичат плавно и непрекъснато във времето. Тези методи обикновено разчитат на сложни диференциални формули за представяне на различните аспекти на системата. За разлика от тях при дискретните методи промяната в състоянието става само през точно определени интервали от време. С това те са по-близки до работата на компютрите отколкото реалното протичане на събитията. Съществуват и трети тип модели които обединяват горните два – хибридните. Те притежават характеристики както на аналоговите така и на дискретните. Изборът кой от трите метода да използваме зависи много от конкретните задачи пред изследването.

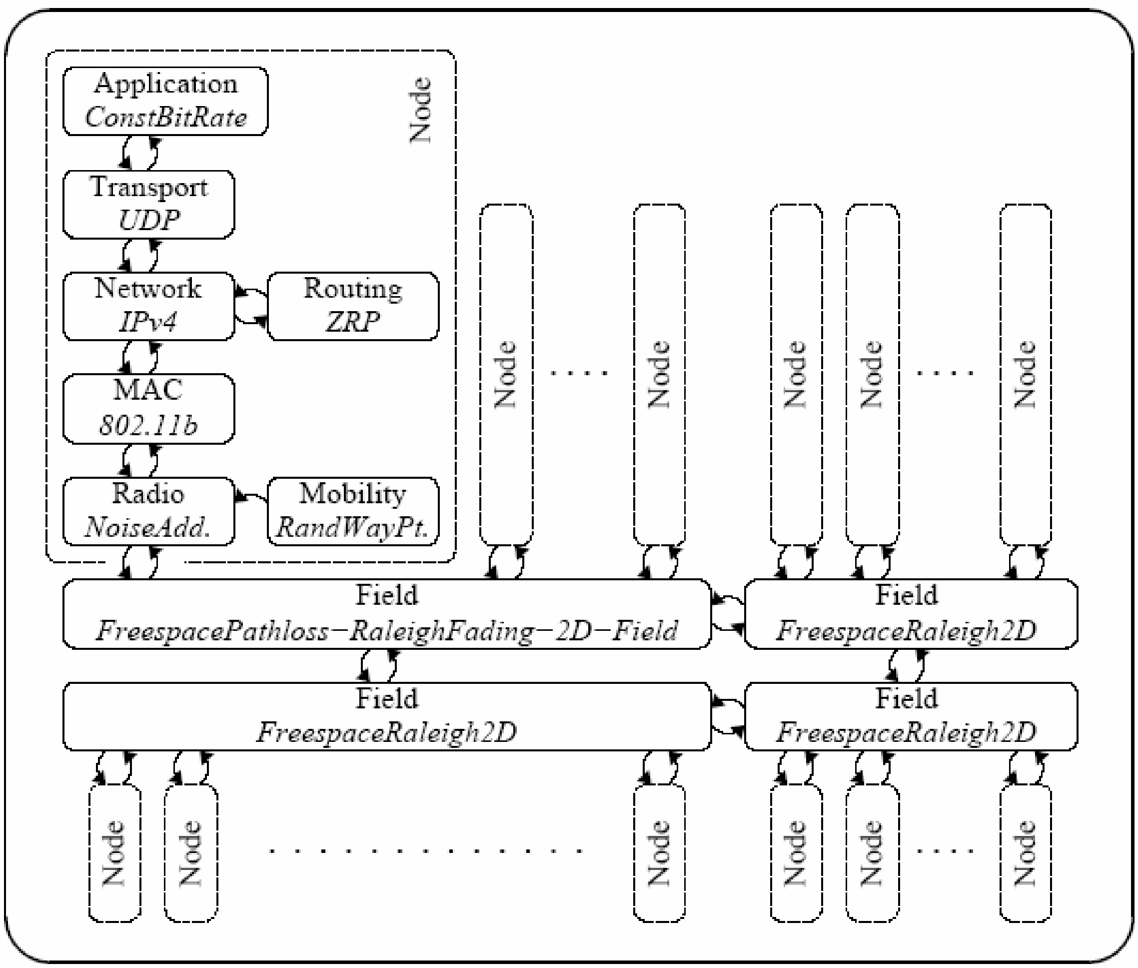
Първия мрежов симулатор, който е разгледан е Network simulator 2 (ns-2). Той спада към класа на дискретните или още т.нар. discrete-event симулатори. Първоначалната му поява е в далечната 1989 година с проекта REAL. От тогава той търпи голямо развитие, добавени се възможности за симулиране на безжични, на разпределени, а в най-скоро време и на мобилни разпределени мрежи. В изследователските среди той се радва с най-голяма популярност. Разработен е като монолитен, с последователно изпълнение симулатор, който използва библиотечния подход. Сред предимствата му са използването на т.нар. *split objects*. Те позволяват конфигуриране чрез TCL базирани скриптове на имплементираните чрез C обекти мрежови протоколи. Този подход го прави много удобен за ползване от потребителите му. Въпреки, че е широко използван и един от най-ранните симулатори той не е завършен симулатор. Непрекъснато се правят подобрения и включване на нови функционалностти от изследователи и разработчици. Това се дължи на факта, че симулатора е широко достъпен и с отворен код. Сред недостатъците на ns-2 са голямата консумация на памет, което прави трудно и дори невъзможно симулирането на мрежи с голям брой станции. Затруднено е и проектирането на нов протокол поради сложните и уникални концепции заложени в кода му, за разучаването на които изисква време. От друга страна самата симулация отнема много голямо време за протичане поради последователния характер на изпълнение на дискретните събития. Разработени са вариантите на ns-2 за паралелно изпълнение на няколко машини, които донякъде се справят с част от проблемите. Залагането на модел на движение става основно посредством *trace* файлове.
Следващият често използван симулатор е GloMoSim (Global Mobile Information System Simulator). Той е сравнително нов симулатор написан на PARSEC, език който представлява силно оптимизиран за целите на симулацията вариант на езика C. Също както и ns-2 е базиран на библиотеки, като самия симулатор е изграден като отделни модули от библиотеки всяка включваща протоколи от конкретен слой от мрежовия модел. По този начин GloMoSim може да бъде разширяван лесно с добавяне на нови библиотеки. Голяма част от предимствата на GloMoSim се дължат на езика PARSEC (Parallel Simulation Environment for Complex Systems). При него се използва базиран на съобщения подход за симулиране на дискретните събития. Физическите процеси се моделират от симулационни обекти наречени *entity*. Съществува в два варианта: с последователно изпълнение (достъпна безплатно за академичните среди) и паралелна

версия която се предлага комерсиално. Също както ns-2 симулатора безплатната му версия е с отворен код. При симулация на много станции страда поради големите изисквания към памет на отделните entity блокове. Авторите му са въвели като решение групиране на няколко станции в един entity обект, но това довежда до усложнение в кода и програмирането. Въпреки това GloMoSim е способен да симулира много по-големи мрежи с до 10 000 станции на големи специализирани мулти-процесорни машини.

Дизайнерите на повечето симулатори са избрали за реализация C или базиран на C език за програмиране. Следващия симулатор прави изключение защото е написан на JAVA. Симулатора за безжични мрежи SWANS (Scalable Wireless Ad hoc Network Simulator) е построен върху JiST (Java in Simulation Time) платформата, която представлява дискретно симулационно ядро за общи цели. Сред основните му предимствата са възможността за симулиране на мрежи с много голям брой станции, интегрирането на съществуващи JAVA приложения и лекотата на описване на нови протоколи. Симулаторът SWANS е организиран под формата на отделни компоненти които могат да бъдат подходящо комбинирани за изграждане на различни сценарии на симулация. Разгледания по-горе модел на движение STRAW е създаден именно като компонент на SWANS симулатора. Възможностите на SWANS са подобни на другите два популярни симулатора ns-2 и GloMoSim, но изследванията показват, че SWANS използва значително по-малко ресурси памет и работи по-добре от тях. Поради специфичните задачи пред настоящата дипломна работа именно SWANS симулатора е избран за верификация на предложения CIGDP протокол. В следващата глава са разгледани по-подробно специфичните концепции и характеристики на SWANS симулатора. Разгледани са фазите по изграждането на нов протокол и подготвянето на симулацията му, използвайки предложения в настоящата работа CIGDP протокол като пример.

4 СИМУЛАЦИОННО ИЗСЛЕДВАНЕ НА ПРЕДЛОЖЕНИЯ ПРОТОКОЛ

За симулация на CIGDP протокола е избран SWANS симулатора. Както вече беше споменато в предходната глава симулаторът се състой от независими компоненти, които могат да бъдат комбинирани за цялостна симулация на дадена безжична мрежа. Разработени са и продължават да се разработват нови компоненти за различните слоеве на мрежовия модел. Фигура 14 илюстрира примерно съчетание на отделни компоненти.

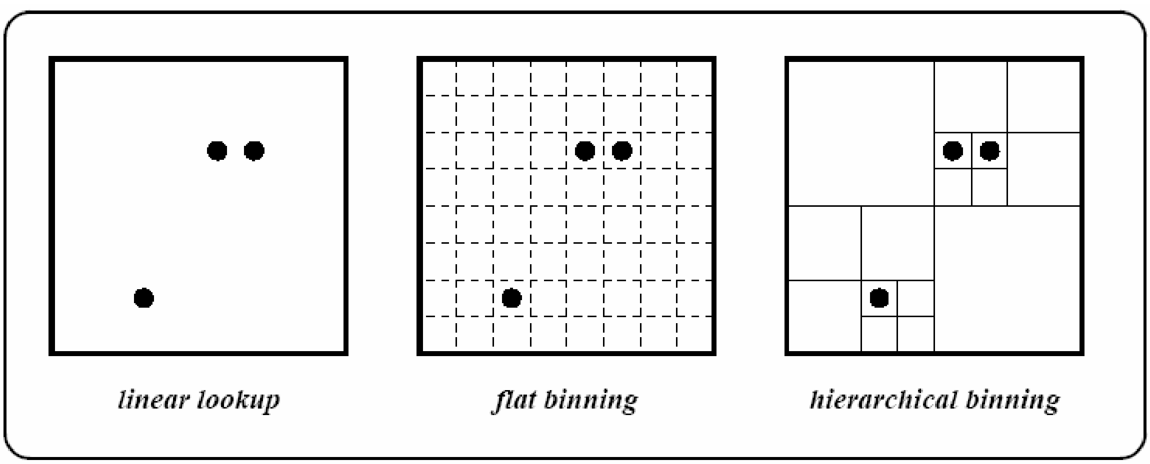


Фиг. 14: SWANS симулатора се състой от отделни компоненти, които комбинирани изграждат цялостна симулация. Показана е примерна тяхна конфигурация и инициализация, както и взаимодействието между отделните станции.

Всяка компонента на симулатора е реализирана като отделна капсула (т.нар. *JiST entity*). Всяка капсула си има свое собствено вътрешно състояние и комуникира с останалите компоненти посредством открит събитийно-базиран интерфейс. Към всеки компонент е привързан стек, чрез който се осъществява обмяната на съобщения. Това улеснява разработването на нови протоколи свеждайки го до описанието на малки, събитийно базирани компоненти. Голямо предимство на SWANS пред останалите симулатори е комуникацията между слоевете, реализирана много ефективно благодарение на JiST концепцията. Тук няма сериализиране, копиране или превключване на контекста между капсулите, понеже Java обектите се обменят по указател (reference). Например едно broadcast съобщение може да бъде споделено между всички станции и същия обект , който е пуснат от приложния слой на изпращача ще се получи от приложния слой на приемащите станции. Този дизайн спестява консумацията на памет, позволявайки симулирането на по-големи мрежи. Работата по освобождаването на паметта заета от отделните пакети е изцяло оставена на събирача на боклук в Java (garbage collector), което улеснява проектирането.

Сред предимствата на SWANS симулатора е възможността за внедряване на стандартни приложения без каквито и да е промени по тях в симулацията. Това уникално за съществуващите симулатори качество допълнително облекчава симулирането давайки възможност за използване на вече съществуващи приложения като web servers, peer-to-peer и мултикаст приложения.

Друго голямо предимство на SWANS симулатора е моделирането на разпространението на сигналите. Когато една станция излъчи сигнал той ще се получи от всички станции в обхвата й, както и ще доведе до интерференция с малката част от станциите които се намират преди прага на чувствителност. При останалите симулатори се използват основно два метода на разпространение на сигналите: с линейно търсене измежду всички станции и с търсене с помощта на решетка. В допълнение към тях SWANS използва и йерархично търсене което дава много по-голяма ефективност. Трите метода за разпространение на сигналите са разгледани на фигура 15.



Фиг. 15: Три алтернативни метода за разпространение на сигналите в SWANS. От изборът на метод зависи колко големи мрежи могат да бъдат симулирани

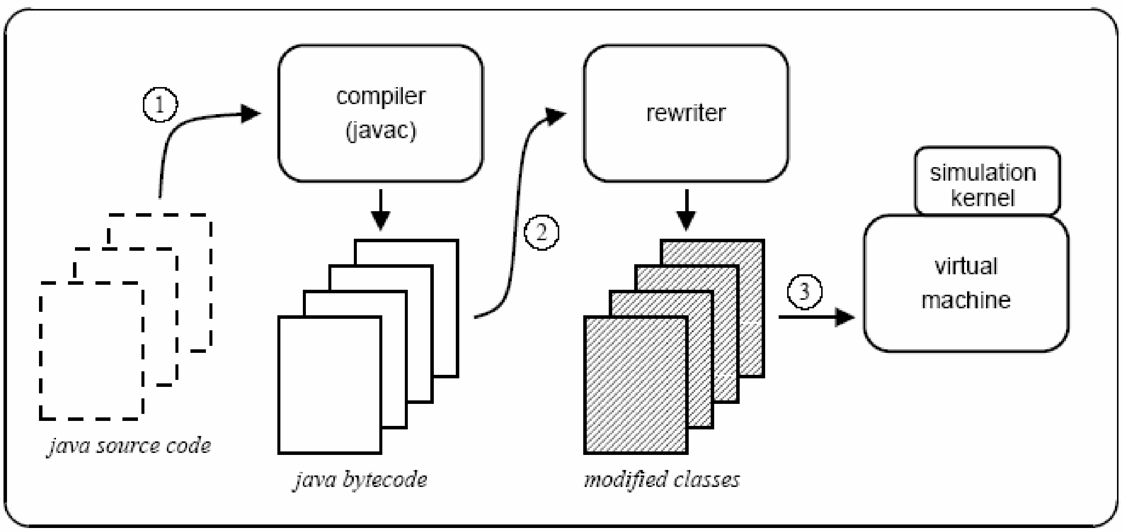
В следващите редове са разгледани накратко различните компоненти които са разработени до момента за SWANS симулаторът. Те са класифицирани по това на кой слой от мрежовия модел работят.

- Физически: компонентите на този слой са отговорни за моделиране на разпространението на сигналите между станциите, както и за модела на движение на станциите (групирани в *field entity*). Тук са още компонентите отнасящи се до радио излъчвателите и параметрите им на работа: дуплекс, честота, мощност на излъчвателите, чувствителност на приемателите, пропускателна способност и модел на грешките (*radio entity*);
- Канален: на този слой работи компонентата за достъп до преносната среда. Разработени са пълния DCF IEEE 802.11 протокол (*mac entity*) включващ препредаване, NAV и back-off функционалността. Както и един Dump протокол при който станциите излъчват само ако не са заети с друго предаване, разработен е и loopback интерфейс;
- Мрежови: на този слой е разработен единствено IPv4 протокола (*net entity*). Разработени са packet handlers за насочване на пакетите към един или друг транспортен протокол. Имплементирани са broadcast и loopback адресирането;
- Маршрутизиращ: Този слой получава от мрежовия слой пакетите за които е необходимо намиране на път. Връща му като резултат адреса на следващата станция от пътя. Имплементирани са ZRP, DSR и AODV протоколите (*route entity*);
- Транспортен: на транспортния слой работи компонентата *transport entity*, която имплементира TCP и UDP протоколите. Тя обикновено си взаимодейства с съответните сокети от приложния сой. По долу е разгледано как става разграничаването с реалните сокети.
- Приложен: на този слой са разработени еквиваленти на реалните Java сокети и потоци за вход/изход. Реализирано е и приложение за откриване на съседи (единствения компонент при който се прескача един слой при обмяната на съобщения и съобщението се предава директно на мрежовия)

Съществува един много важен интерфейс, който е общ за всички слоеве на SWANS и дава общата функционалност на системата. Това е интерфейса *message*, който представлява движението на пакетите през стековете на станциите. Обектите имплементиращи този интерфейс трябва да са постоянни и да не се променят във времето (т.нар. *timeless*). Всяка от компонентите на различните нива има своя имплементация на този интерфейс, определяща структурата на заглавната част на съобщението за съответния слой.

Както вече бе споменато, SWANS симулаторът работи върху JiST (Java in Simulation Time), симулационно ядро използващо дискретния, събитийно-базиран модел на симулация. За да може да се опише нов протокол, който да се симулира в SWANS трябва да сме запознати с архитектурата и начина на работа на JiST. Следващите редове разглеждат именно това.

Архитектурата на JiST, разгледана на фигура 16 се състой от четири ясно разграничими компонента: компилатор, преобразувател на байт код, симулационно ядро и виртуална машина. При създаването на нов протокол за симулиране, той се описва на чист, немодифициран Java код и се компилира на стандартен компилатор до байт код. Този байт код после се преработва така, че да може да работи върху симулационното ядро в т.нар. симулационно време което ще разгледаме по-късно. Симулираната програма, преобразувателя на байт код и симулационното ядро са написани всичките на чиста Java. Самия процес на симулиране също протича на стандартна немодифицирана Java виртуална машина (JVM). Предимствата на този подход са възможността за използване на съществуващ вече код, стандартни библиотеки и съществуващи вече компилатори. Възползването от всички предимства на стандартния Java език. От друга страна факта, че ядрото и самата симулация работят в един и същ процес намалява загубите от сериализиране и превключване на контекста.



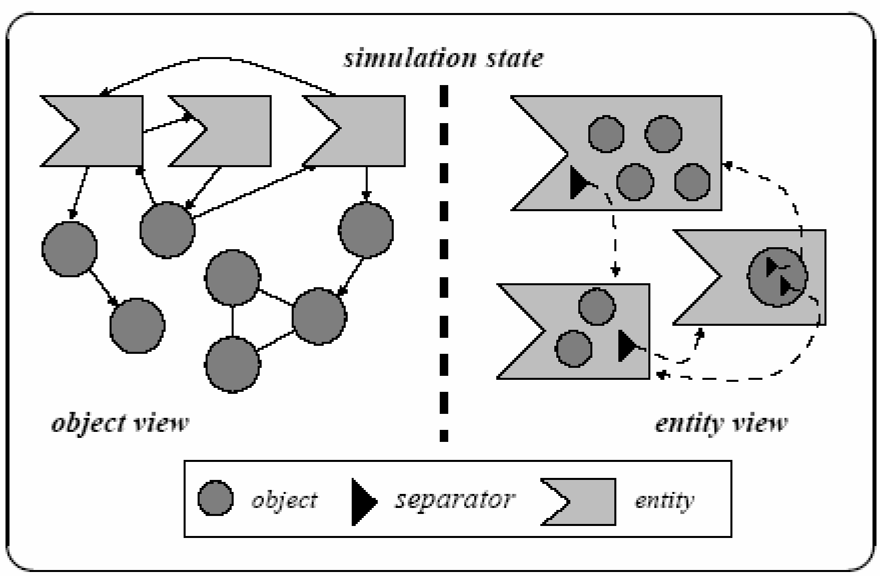
Фиг. 16: Системната архитектура на JiST: симулацията се компилира (1), подготвя се за изпълнение от преобразувателя на байт код (2) и накрая се изпълнява (3)

Изпълнението на модифицираните класове на виртуалната машина е както всяко друго стандартно приложение. Няма гаранции за времето на изпълнение, което зависи от скоростта на процесора и от непредсказуеми събития като прекъсвания и входно/изходни операции. Това време, което не е зависимо от прогреса на приложението, авторите на симулатора наричат актуално време (*actual time*). За да може да се направи връзка между протичането на изпълнението и времето, изследователите са въвели, т.нар. реално време (*real time*) при което се гарантира изпълнение на задачите за даден времеви интервал. Концепцията която се използва в Jist е коренно противоположна: скоростта на протичане на времето е зависима от прогреса на приложенията. Това е т.нар. симулационно време (*simulation time*). При него всички задачи се изпълняват за нула време последователно една след друга докато се изпълнят всички, след което се преминава към следващия квант време. Програмите могат да напредват във времето чрез извикването на функцията sleep(n). След извикването и програмата ще продължи изпълнението си след точно ‘n’ кванта време. Симулацията протича докато се изпълнят всички задачи или докато се стигне до предварително зададен квант от време при който се спира изпълнението. В таблица 1 са обобщени трите вида взаимодействие между прогреса на програмата и протичането на времето.

Актуално време	- Прогреса на програмата и времето са независими
Реално време	- Прогреса на програмата зависи от времето
Симулационно време	- Прогресът на времето зависи от програмата

Табл. 1: Връзката между протичането на програмата и времето

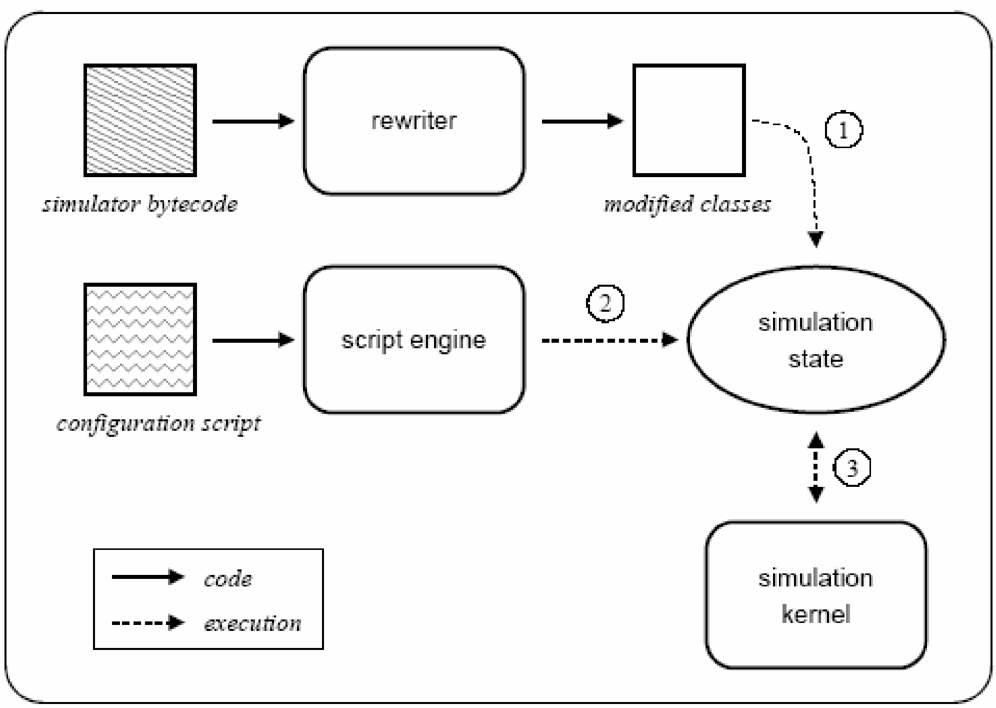
Симулационните програми за JiST се пишат на обектно-ориентирания език Java и както всяка друга програма писана на обектно-ориентиран език състоянието и по време на изпълнение се представлява от състоянието на отделни обекти. Тези обекти комуникират помежду си обменяйки съобщения, които представляват извикване на метод на обект. С цел улеснение на проектирането се въвежда логическото понятие *entity* (обекти наследяващи интерфейса JistAPI.Entity). За виртуалната машина това са си стандартни обекти, но те служат за логическо обединение на няколко обекта. Това означава, че за да се изпълни метод на обект принадлежащ на друга капсула (entity) трябва да и се прати съобщение. Всеки указател към обект трябва да произлиза от същата капсула. Указателите към капсулите обаче могат да произлизат от всякъде. Въвежда се понятието разделител (*separator*), чрез който става връзката между капсулите. Те се генерират по време на преобразуването на байт кода като всяка референция към обект от друга капсула се заменя от съответния сепаратор. Сепараторите и групирането на обекти в капсули са илюстрирани на фигура 17.



Фиг. 17: Логическо разделение на капсулите посредством сепаратори

Вече споменахме за т.нар. *timeless* обекти, когато разглеждахме интерфейса на съобщенията при SWANS. Това са обекти за който се знае, че са постоянни с времето. Знаейки това можем да ги обменяме между капсулите по референция вместо да се копират. За един обект се знае, че е timeless ако наследява интерфейса JistAPI.Timeless. Останалите обекти се предават по копие и не могат да бъдат използвани за обратна връзка посредством някое от полетата на обекта.

Едно важно качество на симулаторите е конфигурирането на постановката и желанието за повторно използване на кода за различни постановки. Възможностите са няколко, първата е на ниво код, параметрите са зададени в самия програмен код. При промяна е необходимо прекомпилиране, което прави този метод не много бърз. Друг метод е с конфигурационен файл, при който е необходима допълнителна програма наречена *driver* която да извърши разбора и инициализирането. Този метод ни спестява прекомпилирането, но не е много гъвкав поради наличието на фиксирана структура на конфигурационния файл. Това донякъде може да се избегне с използване на XML. Третия вариант е внедряването на скриптов език за целите на конфигурацията. Това е постигнато чрез TCL при ns-2, но с цената на усложняване и повишаване на използваната памет за симулацията. В Java използването на този метод идва без особено висока цена тъй като поради спецификата на езика не е необходим допълнителен код в симулираните компоненти за да се стартират посредством скрипт език. Това е благодарение на динамичната натура на езика Java, който има вградена поддръжка на т.нар. *reflections.* Това е илюстрирано на фигура 18, от която става ясно, че подобна конфигурация е толкова гъвкава колкото и предварително зададена в самия код конфигурация поради наличието на междинното ниво байт код.



Фиг. 18: Конфигуриране на симулацията посредством скрипт език

Изпращането на събития между компонентите се реализира чрез извикване на методи. Това крие редица предимства както и опростяване и намаляване на необходимия код без последствия за производителността. Сред предимствата са:

- Type safety: по-голяма изолираност от грешки понеже източника и получателя на съобщението се проверяват статично от компилатора;
- Event typing: определянето на типа на съобщението се извършва автоматично по време на изпълнението и не е необходимо да се извършва преобразуване на типа на съобщението понеже това се осъществява автоматично при извличането му от опашката на чакащите събития;
- Event structures: не е нужно да се организират сложни структури с единствената задача да се предадат параметри към съобщението. Тук това отново се реализира автоматично;
- Debugging: при повечето събитийно ориентирани симулатори, откриването на грешки е трудна задача защото събитията пристигат без контекст. Тук има възможност да се предават заедно с контекста си (мястото от където е пуснато и състоянието). По този начин събитията могат да се свържат във верига чрез която да се проследи проблема. Поради тежестта на тази операция обаче се прилага само в режим на debugging
- Execution: и не на последно място това позволява за прозрачно реализиране на паралелно и разпределено изпълнение, без необходимостта от допълнително модифициране на кода.

Дотук бяха разгледани принципите на работа и механизмите необходими за изграждане на нов протокол и цялостната му симулация на JiST/SWANS симулатора. В следващите редове са разгледани конкретните стъпки от подготовката и симулацията на CIGDP протокола. Той е реализиран като компонента от приложния слой. Освен връзка с долния транспортен слой е добавена и със компонентата за мобилността за получаване на данни за местоположението (т.е. компонентата за движение играе ролята на GPS устройството).

Протоколът CIGDP е реализиран в четири отделни модула, всеки от които реализиран като капсула (JiST Entity). Това са модула за изчисляване на изминатото разстояние, модула за локално откриване на задръстване, модула за запитване на отсрещните автомобили и модула за разпространение на генерираните съобщения. Стартирането на модулите по време на симулация обаче не става едновременно от зареждащата програма. Последователността на зареждане е следната: зареждащата програма създава инстанция на мобилният възел, инициира протоколите от по-долните нива, след което стартира модула за запитване. При стартирането си този модул зарежда модулът за откриване на задръстване, който от своя страна стартира модулът за измерване на изминатото разстояние. Пълният програмен код на CIGDP протоколът е включен в Приложение 3.

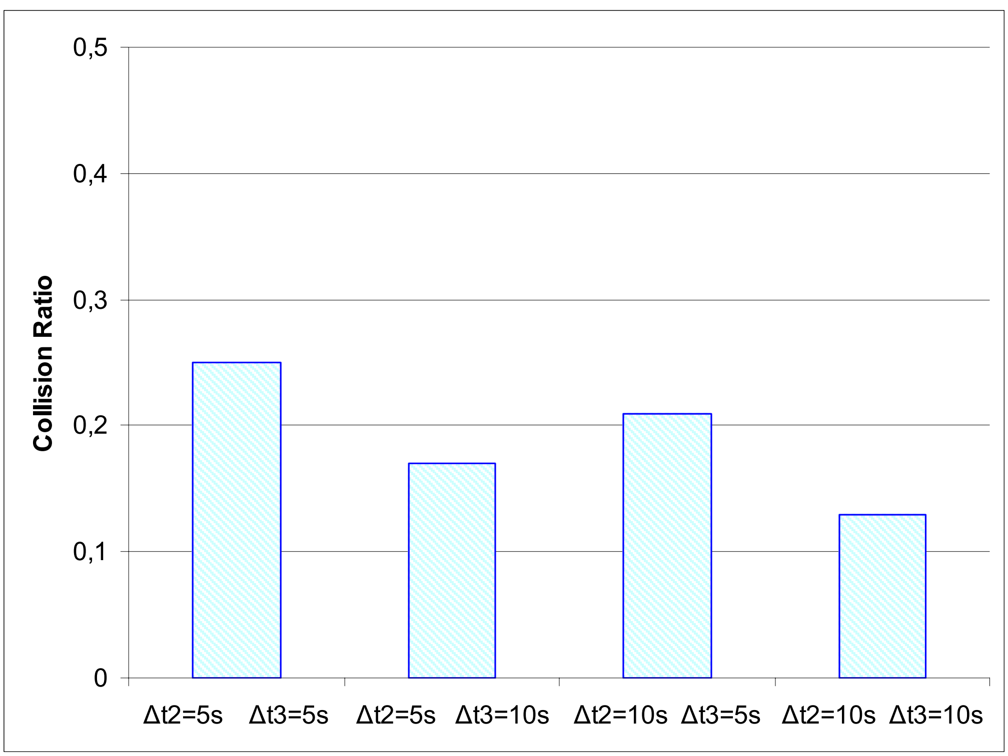
В следващите редове е разгледана подготовката и протичането на симулацията и получените резултати. Подготовката на симулацията е разгледана по слоеве. Първото решение което трябва да се вземе е изборът на модел на движение. От съществуващите модели разработени за конкретно избрания симулатор изборът е спрян на STRAW моделът поради най-голяма реалност на движението, която дава за VANET мрежите. Този модел вече бе разгледан по-горе. Той дава възможност за използване на реални пътни данни с което още повече доближава симулирането да реалността тъй като различните градове имат различна пътна инфраструктура и следователно и симулираните протоколи ще имат различни показатели за всеки от тях. Моделът STRAW прочита пътните данните от подходящо форматирани файлове. Съществуващите му възможности са използването на обществено-достъпните пътни данни за територията на САЩ (т.н. TIGER данни). За да може да се симулира работата на протоколът за реални пътни данни за град Нотингам, Англия е разработен допълнителен помощен инструмент - програма преобразуваща наличните пътни данни за град Нотингам, които са в Garmin формат (вид GPS данни) по подходящ начин за прочитане от STRAW софтуера. Кодът на тази програма е включен в приложение 3. Моделът STRAW предлага две възможности за определяне на маршрутите на автомобилите. При единия на всяко кръстовище автомобила с определена вероятност продължава в една или друга посока. При втория вариант се задават начало и край за всеки автомобил и се изчислява маршрута използвайки A* (произнася се A звезда) алгоритъма за намиране на най-краткия маршрут. Алгоритъмът е евристичен от класа на т.нар. *best-first search* алгоритми към които принадлежи и алгоритъма на Дийкстра. Като евристичен алгоритъм той не гарантира винаги намиране на път и обикновено ако не успее да намери такъв се стартира стандартния алгоритъм на Дийкстра. За целите на симулацията беше избран моделът с начални и крайни точки. Той бе предпочетен поради по-големия контрол над движението на автомобилите и възможността за предизвикване на задръстване. Което представлява основна задача на изследването.

Изборът на компонентите на останалите нива е следния: на каналния слой се използва реализацията на IEEE 802.11b DCF. Обхвата на действие е 250м; на мрежовия слой се използва протокола IPv4. Избраната реализация на адресиране е плоско адресиране (нямаме обособени мрежова и хост част). За IP адрес се използва поредния номер на инициализиране от зареждащата програма на конкретния възел; изборът на протокол от транспортния слой е спрян на UDP. Порта е избран да бъде 5001. Основната задача на симулирането е определянето на работоспособността на CIGDP протоколът, който е включен на приложния слой.

Следващата стъпка е инициализирането на симулацията. Поради по-лесното реализиране и понеже не гъвкавостта на стартиране на симулацията е целта пред настоящата дипломна работа е избран метода със статична инициализация. Симулацията се стартира от програма наречена драйвер, която прочита параметрите на симулацията от XML файл. Използват се стандартни публични програми за синтактичен разбор на XML файла. След което се генерира инстанция на симулатора, подават му се обработените параметри и се стартира симулацията. Следващата стъпка е зареждането на файловете с пътните данни. Обикновено пътните данни представят големи области или цели градове. Симулацията обаче в рамките на цялата област би отнела значително време и не е подходяща за началните изследвания поради забавянето между фазата на откриване на проблеми и отстраняването им. Поради тази причина сред входните параметри се задават и границите на областта в която ще се движат автомобилите, т.нар. симулационна област. Когато се зареждат данните се зареждат само тези части от картата които принадлежат на симулационната област.

Избраното време за симулиране на протоколът е 20 минути. Отчитането на избраните параметри, които ще следим започва след интервал от 60 секунди. Причината за това отместване е, че първоначално мобилните възли имат нулева скорост на движение при поставянето си върху картата и е необходимо време до достигане на реална среда на движение.

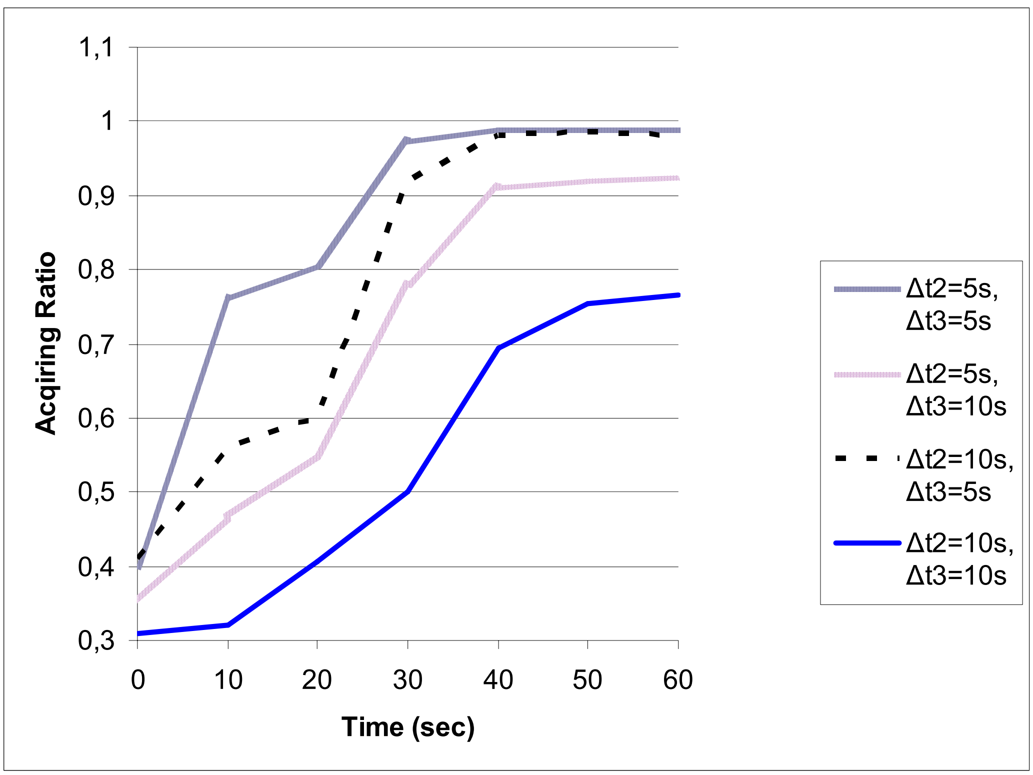
Параметрите, които представляват интерес са ниво на възникнали колизии и скорост на придобиване на информацията от всички автомобили, които преминават през задръстването. В резултат на няколко симулационни експеримента проведени със следните стойностти на системните параметри: Δt_1=10s, Δt_2=5,10s, Δt_3=5,10s и d_1=350m. Тук Δt_1 е кванта време през който всеки автомобил проверява параметрите си за възникнало задръстване, Δt_2 пък е времето през което се прави запитване към отсрещните автомобили за регистрирани събития, Δt_3 е интервала от време през който се разпръскват активните съобщения. Последния параметър d_1 дава критичното разстоянието под което се приема, че е възникнало задръстване. На фигура 20 са показани резултатите от процента на възникване на колизии за четири различни комбинации на системните параметри. Оценката на показаните резултати е разгледана в следващата глава.



Фиг. 20: Ниво на колизии за четири комбинации на системните параметри

Следващия важен показател за работоспособността на CIGDP протоколът е процентът на автомобилите придобили информацията за задръстване представен като функция на времето. На фигура 21 е представена графика с резултатите като отново са разгледани четири различни комбинации на системните параметри.

Тъй като самото разпространение на информацията за задръстване не би било много полезно ако не довеждаше до възможност за смяна маршрута с цел оптимизиране на времето за предвижване е направена оценка на времената за изминаване на разстоянието с и без промяна на маршрута. Избрани са такива пет автомобила, чиито маршрут пресича зоната на задръстване, но също така имат еквивалентни алтернативени маршрути. След получаването на съобщение за задръстване те преизчисляват маршрутите си с начало текущата си позиция и първоначалната си крайна точка. При отчитането на времената обаче не се отчитат никакви параметри на задръстването, а се предполага, че сегмента от пътя върху които се намира задръстването не е наличен и се избира втория най-оптимален. Резултатите са представени в таблица 2.



Фиг. 21: Придобиване на информацията

Таб. 2: Времена за изминаване на маршрутите с и без премаршрутизиране

Times for traversing OD distance (minutes)	Vehicle 1	Vehicle 2	Vehicle 3	Vehicle 4	Vehicle 5
Without Re-routing	5,23	13,1	2,7	4,74	13,01
With Re-routing	4,07	7,3	5,1	4,05	8,04

В тази глава бяха разгледани концепциите за симулиране на прокол на JiST/SWANS симулатора. Разгледани бяха етапите по подготовка и изпълнение на симулацията и изборът на компоненти на отделните слоеве. В самия край бяха предложени резултати от симулирането на CIGDP протокола. В следващата глава е предложена оценка и анализ на резултатите и разискване за производителността на предложения протокол, както и някои насоки за бъдещо развитие.

5 АНАЛИЗ И ОЦЕНКА НА РЕЗУЛТАТИТЕ

Тъй като не съществуват достатъчен брой разработени протоколи с подобни функции и действие, оценката на получените резултати представена в настоящата глава е повече аналитична отколкото сравнителна.

Първо са разгледани резултатите за нивото на колизии. От графиката представена на фигура 20 се забелязва, че процентът на възникнали колизии е в по-силна зависимост от интервалът на разпръскване на съобщенията Δt_3. Колкото по-малък е интервалът, т.е. по често се разпръскват активните съобщения, толкова е по-голям процентът на възникнали колизии. По-слаба е зависимостта от интервалът на запитванията и това е нормално, имайки предвид, че честотата на отправяне на запитвания е по-малка от тази на последвалото разпространение на генерираните съобщения. Сравнявайки тези параметри с параметрите за ниво на колизия при RMDP протокола наблудаваме значително по-добри стойностти, те обаче се дължат и на различните алгоритми представени зад двата протокола. При RMDP протоколът е характерно непрекъснато разпръскване на съобщения от всички автомобили, докато при CIGDP това е функция само на избрани автомобили.

Анализа на втората графика представена на фигура 21 показва следното. Първо съвсем логично графиките са нарастващи, като степентта на нарастване зависи най-силно отново от интервалът на разпространеие на съобщенията, докато интервалът на запитване има значение само за по-ранното откриване на задръствания. В графиката се наблюдават резки нараствания последвани от по-плавни участъци. Обяснението на този факт е включването на автомобил по-някои от другите пътища след всяко кръстовище по които преди това не е минавал друг пренасящ същата информация за задръстване. Друго възможно обяснение може да бъде типа на потоците автомобили. Обикновено те се събират на групи на червената светлина на светофарите и по този начин едно съобщение стига едновременно до няколко автомобила, след което следва закъснение до достигане на следващата група автомобили. Резултатите показват, че и при четирите стойности на системните параметри информацията достига до по-голям брой от автомобилите за около 35 секунди, след което графиките са сравнително линеини. Това е сравнително добър показател съизмерим с някои от резултатите за RMDP протокола.

Третия тип резултати касае изборът на нов маршрут след научаване на информацията за задръстване. Тъй като сред целите на настоящата дипломна работа не е изграждането на достатъчно добър алгоритъм за избиране на нов маршрут и преценяването кои автомобили да заобиколят задръстването и за кои ще е по-добре да преминат през него все пак, то на предложените резултати не може да се гледа като на абсолютно верни. Те по-скоро дават някаква приблизителна оценка на база на измерванията от пет предварително избрани автомобила. От таблицата се вижда, че все пак четири от автомобилите пристигат за по-кратко време след преизчисляване на маршрута, докато един от тях го взима за значително по-голямо време. Причината е, че алтернативния маршрут в този случай не е равностоен на този през задръстването.

В заключение може да се каже, че разгледаните резултати дават една добра оценка на работата на предложения протокол. Резултатите са по-добри или поне развностойни на RMDP протокола като същевременно имаме значително намаляване на броя обменени пакети, което съхранява така ценната пропускателна способност. С цел по-добро отчитане на работоспособността на предложения протокол е необходимо разработване на повече протоколи със подобни функции и сравнителна оценка за по-точно определяне на производителността. За да се оцени по-добре предимството на основната идея за генериране и разпространение на транспортната информация посредством автомобили в насрещната посока на движение е необходимо реализиране на варианта с разпространение от автомобилите участващи в събитието и сравнение на резултатите. На базата на получените резултати е възможно да се направи оценка на отделни части от работата на протоколът и да се приложат оптимизации. От получените резилтати става ясно, че изборът на стойности за параметрите Δt_1 и Δt_3 е свързан с компромис между повишаване на броя на колизиите и съкращаване на времето за достигане на информацията до автомобилите които имат нужда от нея. Сред възможностите за бъдеща работа са симулирането на работата на протоколът отчитайки факта, че е възможно не всички автомобили да са оборудвани с необходимите устройства. Също така е наложително разработване на стратегия за справяне с проблема на еднопосочни улици. В този случай трябва да съществува алтернативен алгоритъм за разпространение на информацията. Една добра оптимизация, би била динамично определяне на интервалът на разпръскване на съобщенията към скоростта на движение на автомобилите.

ЗАКЛЮЧЕНИЕ

Разработения протокол за генериране и разпространение на транспортна информация между абонати разположени върху автомобили използва последните достижения в областта на мобилните разпределени мрежи и оборудването на автомобилите с най-различни средства. Значимостта на проблеми като осигуряване на контрол и безопасност на пътя и улесняване на водачите в пътното движение е все по-голяма на фона на увеличаващия се брой автомобили, възникнали произшествия и брой задръствания. Сред предимствата на избрания децентрализиран подход са бързината на разгръщане на системата, ниската цена на изграждане и поддържане както и по-голямата универсалност.

Избраният програмен език Java осигурява платформена независимост, улеснено разработване и по-голяма универсалност. Тези предимства са особено важни за системи в който имаме толкова голямо разнообразие на производители.

Избраният подход автомобилите в насрещната посока на движение да генерират и разпространяват информацията е ключов за производителността на този тип мрежи, поради наличието на голямата разпокъсаност и възможността за използване на по-високата мобилност за оптимизиране на времето за доставяне на информацията.

Изборът за достъп до информация за местоположението е обоснован от гледна точка на факта, че все по-голяма част от съвременните автомобили вече притежават производствено поставени GPS устройства и инсталирани цифрови пътни карти. Наличието на подобна информация значително оптимизира работата на протокола.

Регистрирането на събитията, както и самото генериране и разпространение на съобщенията се извършва автоматично и остава прозрачно за водачите на автомобилите. По този начин се избягва излишното ангажиране на вниманието им за цели различни от управлението на автомобила. Това е особено важно за безопасността на движението.

Така предложения протокол може да бъде лесно трансформиран до готовност за изпълнение в реална обстановка и да се качи на мобилни станции в автомобили. Това е особено улеснено от реализирането му чрез немодифициран Java код. Основната задача обаче е да се продължи развитието и усъвършенстването му наред с оптимизиране на параметрите и сравнение с други протоколи чрез метода на симулирането. Примерни насоки за бъдещо развитие са адаптиране на протокола за ситуации като липса на насрещна лента или недостатъчен трафик по нея; отчитане на факта, че голяма част от използваните в момента автомобили не са оборудвани с необходимите устройства и следователно да се оцени представянето на протокола за различни проценти на екипираните автомобили.

Извършеното изследване е само за регистриране и разпространение на информация за задръстване. Следва да бъдат реализирани още приложения като известяване за инцидент, за мокър, тъмен и хлъзгав участък и други. Поради голямата прилика при разпространението на информацията за тези събития, това би следвало да се осъществи без значителни промени по структурата на протокола.

ЛИТЕРАТУРНИ ИЗТОЧНИЦИ

1. Tanenbaum, Andrew, **"Computer Networks, Fourth Edition"**, Prentice Hall 2003, ISBN: 0-13-066102-3

2. Basagni, S., Conti, M., Giordano, S., Stojmenovic, I. **"Mobile Ad Hoc Networking",** IEEE Press 2004, ISBN: 0-471-37313-3

3. Mohapatra, Krishnamurthy,S., **"Ad Hoc Networks: Technologies and Protocols",** Springer 2005, ISBN: 0-387-22689-3

4. Deitel, H., Deitel, P. **"JAVA: How To Program",** Prentice Hall 2004, ISBN: 0131483986

5. **http://jist.cornell.edu**
   JiST User Guide
   SWANS User Guide

6. Choffnes, D., Bustamante, F. "**An Integrated Mobility and Traffic Model for Vehicular Wireless Networks**, In *Proc. of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks (VANET)*, September 2005

7. Chakeres, I., Belding-Royer (2004) **"AODV Routing Protocol Implementation Design"**. *Proceedings of the International Workshop on Wireless Ad Hoc Networking (WWAN)*, 698-703

8. Dousse, O., Thiran, P., Hasler, M. (2002) **"Connectivity in ad-hoc and hybrid networks"**. *IEEE Infocom* 21, 1079-1088

9. Jardosh, A., Belding-Royer, E., Almeroth, K., Suri, S. (2003) **"Towards Realistic Mobility Models for Mobile Ad hoc Networks"**. *In Proceedings of ACM MOBICOM*, 217–229

10. Jiang, X., Camp, T. (2002) **"A Review of Geocasting Protocols for a Mobile Ad Hoc Network".** *Proceedings of the Grace Hopper Celebration (GHC '02)*

11. Royer, E., Toh, C., (1999) **"A Review of Current Routing Protocols for Ad-Hoc Mobile Wireless Networks"**. *IEEE Personal Communications Magazine*, 46-55

12. Saito, M., Tsukamoto, J. (2005) **“Evaluation of Inter-Vehicle Ad Hoc Communication Protocol”**. *19th International Conference on Advanced Information Networking and Applications (AINA’05)* 1, 78-83

13. Tian, J., Coletti, L. (2003) **“Routing approach in CarTALK 2000 project”**. *In Proceeding of the IST Mobile & Wireless Communications Summit 2*

14. Wu, H., Fujimoto, R. (2004) **“MDDV: a mobility-centric data dissemination algorithm for vehicular networks”**. *ACM Workshop on Vehicular Ad Hoc Networks (VANET)*

15. Xu, B., Ouksel, A., Wolfson **“Opportunistic Resource Exchange in Inter-Vehicle Ad-Hoc Networks”**. *IEEE International Conference on Mobile Data Management (MDM’04)*, 4-12

16. **"IEEE 802.11b Standard"**
http://standards.ieee.org/getieee802/download/**802.11b-1999.pdf**.

17. www.sun.com
j2sdk-1_4_2_09-windows-i586-p

18. www.eclipse.org
Eclipse SDK 3.1

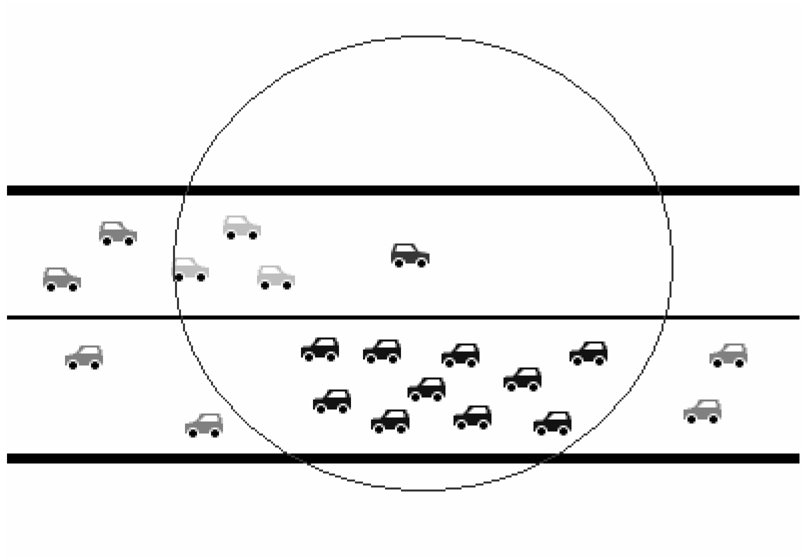
19. www.garmin.com
MapSource User's Guide *Rev. C, Jul, 2005*

20. **http://www.cgpsmapper.com/**
GPSmapper manual version 2.0

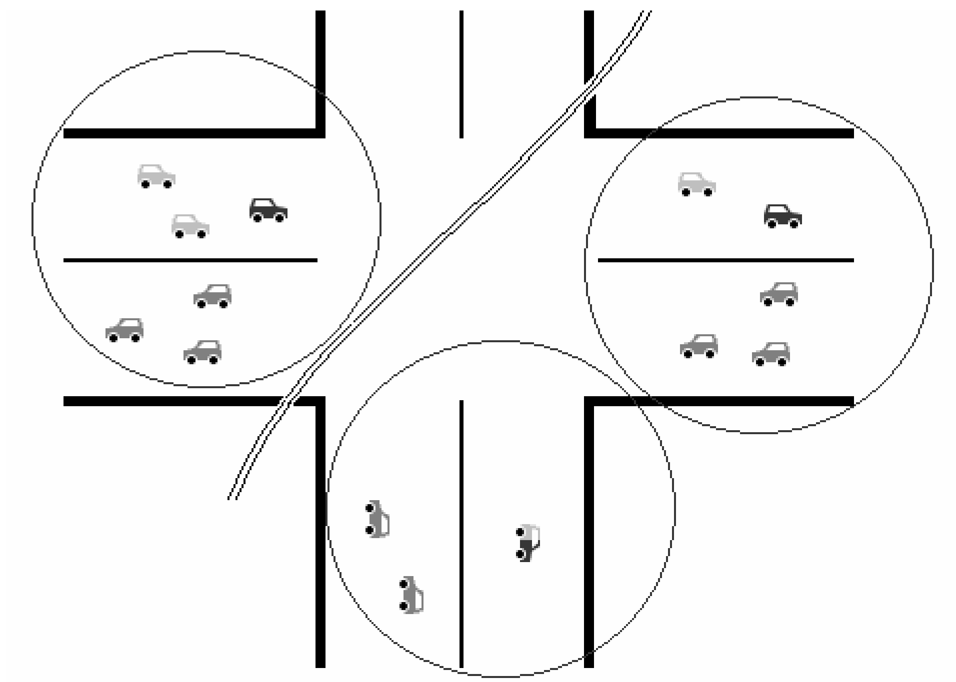
ПРИЛОЖЕНИЯ

- **Приложение 1: Функционална схема на мрежата VANET**
- **Приложение 2: Алгоритми на транспортния протокол**
- **Приложение 3: Листинг на програмните реализации**
    a. **Програмен код на модулите на CIGDP протокола**
    b. **Програма-драйвер за зареждане и стартиране на симулацията**
    c. **Програма за форматиране на Garmin GPS пътните данни във формат подходящ за използване от STRAW**

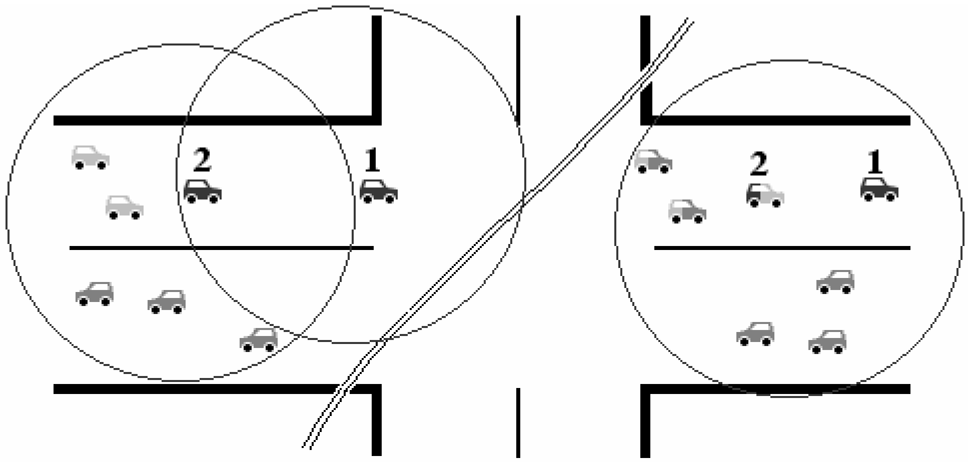
Приложение 1: Функционална схема на мрежата VANET



Фиг. 22: Схема на запитване и генериране на съобщение



Фиг. 23: Схема на разпространение (смяна на статуса от пасивен в активен)



Фиг. 24: Схема на разпространение (смяна на статуса от активен на пасивен)

Приложение 2: Алгоритми на транспортния протокол

Алгоритъм за изчисление на разстоянието:

```
Location_new = get_location() from GPS;
distance = calculate_euclidean_distance( location_new , location_old );
Push distance to distances' array
Location_old = location_new;
Sleep ( 1 second );
```

Алгоритъм за откриване на задръстване

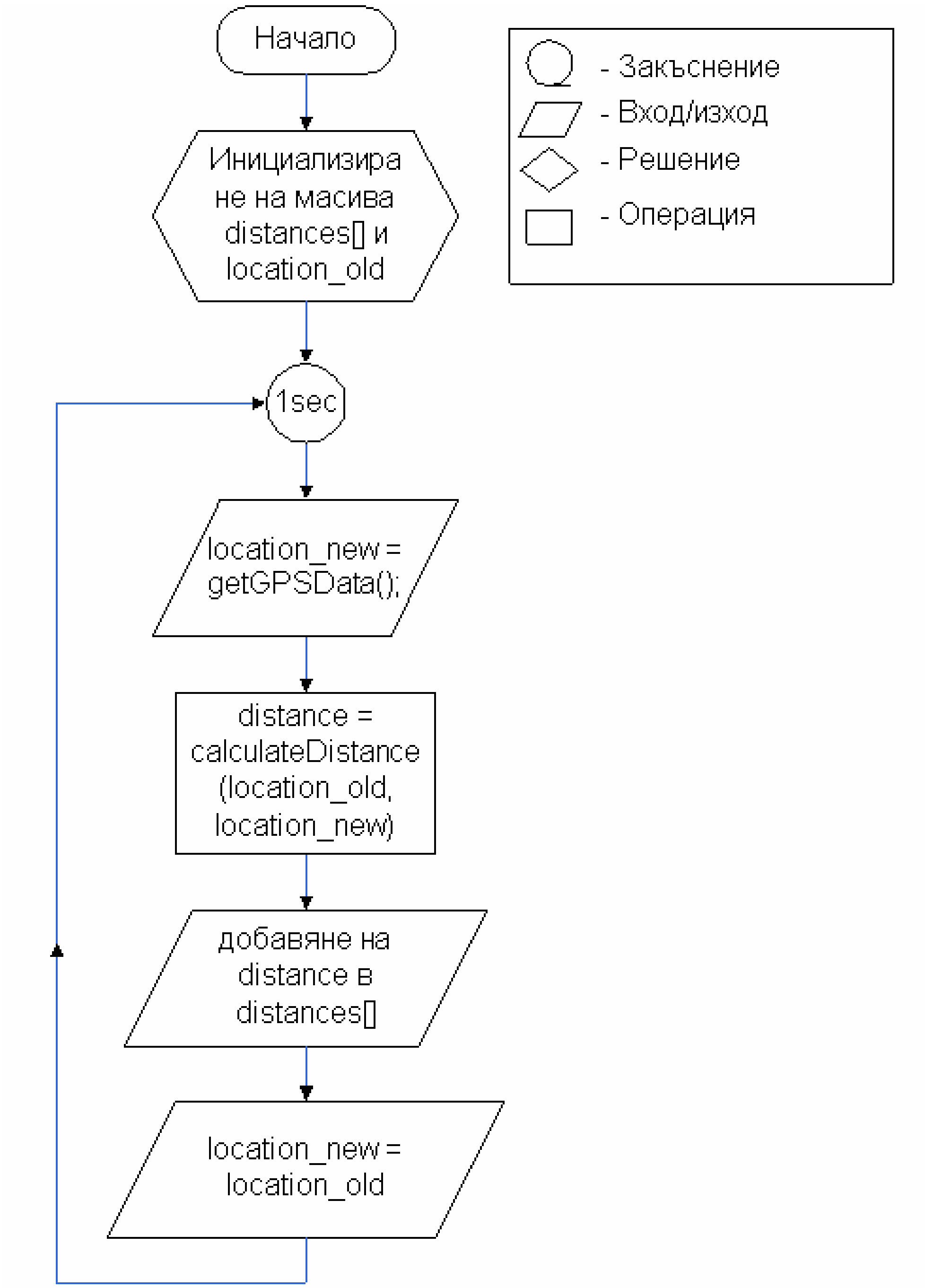
```
for i = 0 to 180 do
    d = d + distances[ i ];
If d <= d1 then jam = true;
else jam = false;
Sleep ( Δt1 seconds);
```

Алгоритъм за отправяне на запитвания

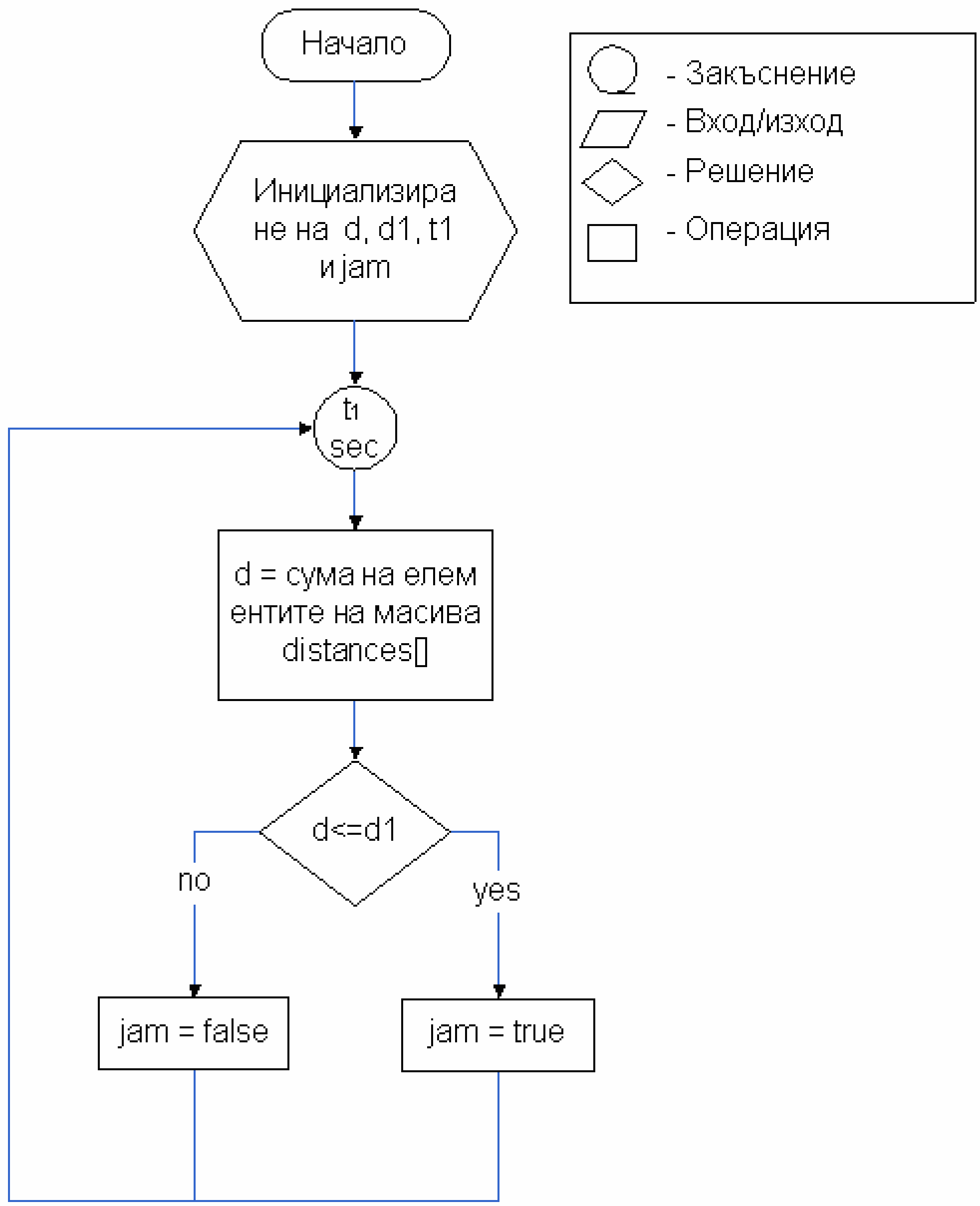
```
send (enquiry message) to all neighbors;
while receiving responses do
    responses_count++;;
If responses_count > (max * Number of lanes)  then
    generate_message (jam);
Sleep ( Δt2 seconds);
```

Алгоритъм за разпространение на съобщенията

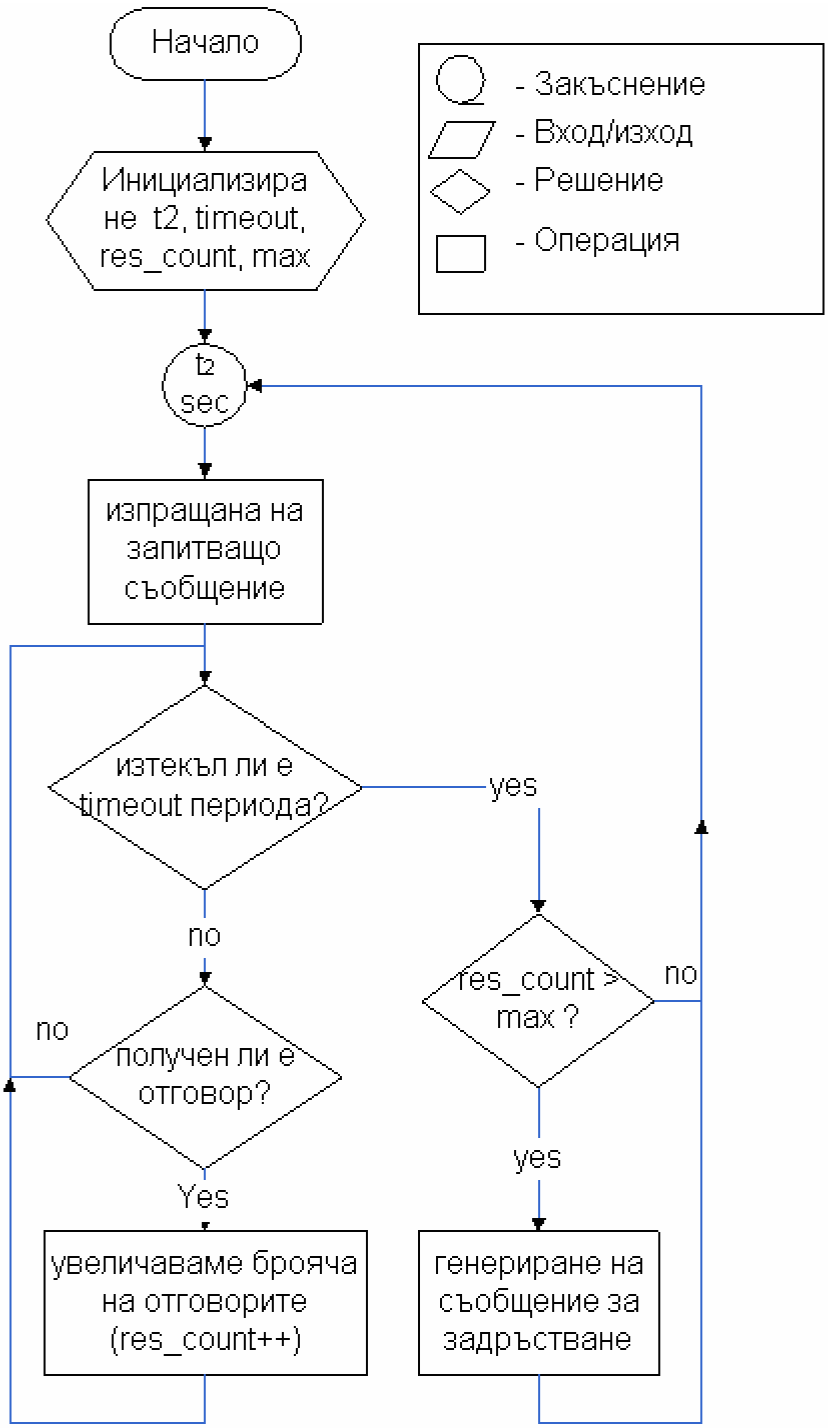
```
If status of message m is Active then
    If new message m1 is received and m1 is new version of m then
        Erase (m);
        m.status = passive;
    Else
        send (m) to all neighbors;
        Sleep ( Δt3 seconds;)
Else if new message m2 is received and m2.status = active then
            Sleep ( Δt3 + random offset );
        Else
            m.status = active ;
            Sleep (Δt3 seconds);
End if
```



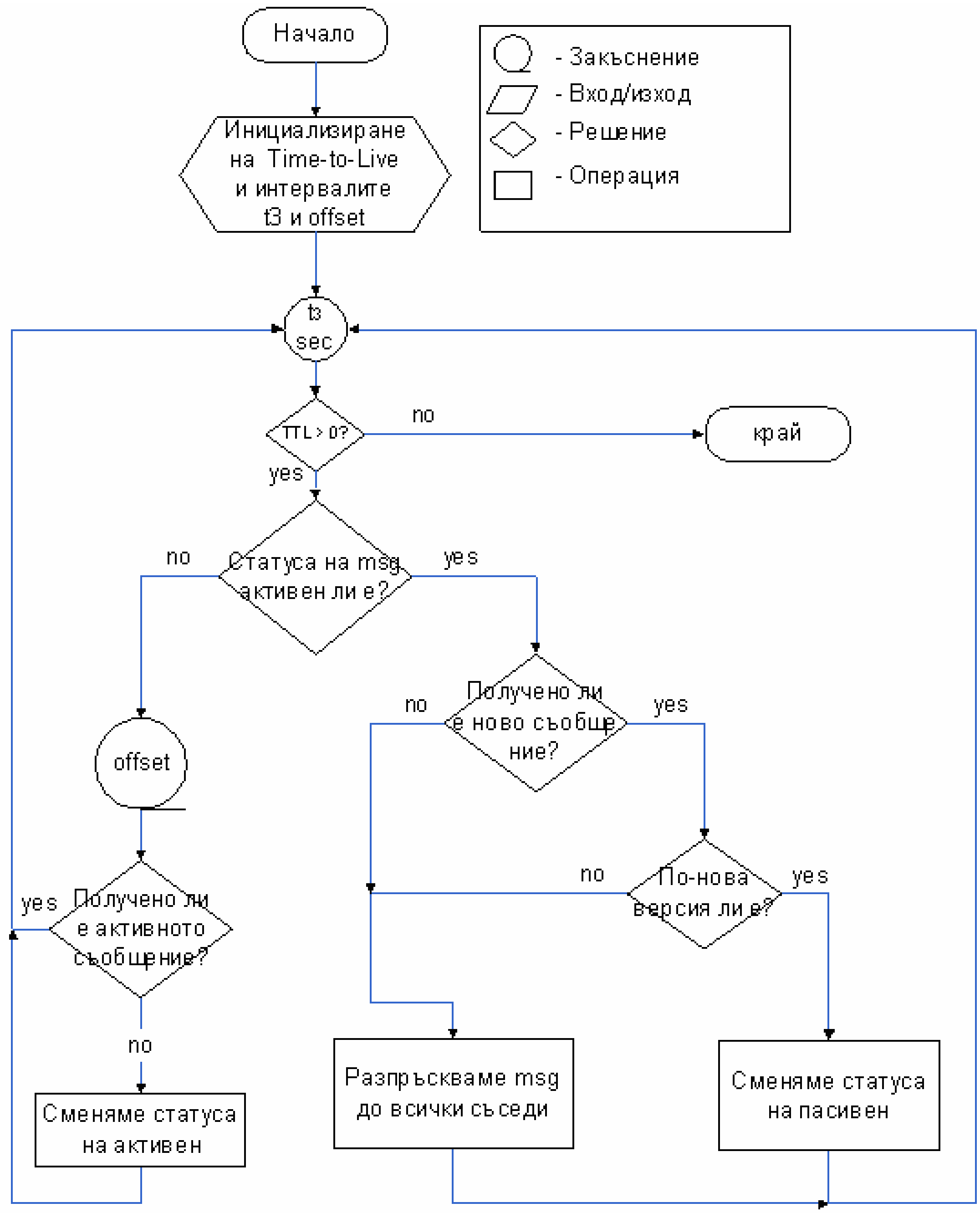
фиг. 25: Блок-схема на алгоритъма за изчисление на разстоянието.



Фиг. 26: Блок-схема на алгоритъма за откриване на задръстване



фиг. 27: Блок-схема на алгоритъма за запитване за регистрирани събития



Фиг. 28:Блок-схема на алгоритъма за разпространение на съобщението

Приложение 3: Листинг на програмните реализации

а) Програмен код на модулите на CIGDP протокола

```
package jist.swans.app.MyProtocol;

import jist.runtime.JistAPI;
import jist.swans.field.FieldInterface;

public interface AppCIGDPInterface extends JistAPI.Proxiable {
      public void setFieldEntity(FieldInterface field);
      public void start();
}

import jist.runtime.JistAPI;
import jist.swans.app.net.UdpSocket;
import jist.swans.field.FieldInterface;
import jist.swans.misc.Location;
import jist.swans.misc.Message;
import jist.swans.net.NetAddress;
import jist.swans.trans.TransInterface;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class AppCIGDP implements AppCIGDPInterface {

      /** Field entity. */
      private FieldInterface field;
      /** UDP entity. */
      private TransInterface.TransUdpInterface udp;
      /** self-referencing proxy entity. */
      private AppCIGDPInterface self;
      private JamDetection jam;
      private int id;
      private Enquiry enquiry;

      /** Udp socket for sending of udp messages */
      private UdpSocket sending_socket;

      /** Udp socket for listening for Jam Enquire messages */
      private UdpSocket socket1;
      /** Udp socket for sending of Jam Enquire messages */
      private UdpSocket socket2;
      /** Udp socket for listening for Jam Response messages */
      private UdpSocket socket3;
      /** Udp socket for sending for Jam Response messages */
      private UdpSocket socket4;

      public AppCIGDP() {
            self = (AppCIGDPInterface)JistAPI.proxy(this,
                    AppCIGDPInterface.class);
      }
      public void setFieldEntity(FieldInterface field) {
```

```
      if(!JistAPI.isEntity(field)) throw new
            IllegalArgumentException("expected entity");
      this.field = field;
}

public void setId(int id) { this.id = id; }

public void setUdpEntity(TransInterface.TransUdpInterface udp) {
      if(!JistAPI.isEntity(udp)) throw new
            IllegalArgumentException("expected entity");
      this.udp = udp;
}

public void start() {
      jam = new JamDetection(field,id);
      jam.getProxy().start();

      enquiry = new
            Enquiry(sending_socket,field,udp,jam.getProxy(),id);
      enquiry.getProxy().start();
      TransInterface.SocketHandler handler = new
            TransInterface.SocketHandler()
      {
            public void receive(Message msg, NetAddress src, int
                                                            srcPort)
          {
                  AppMessage.EnquiryMessage enquiry_msg =
                                    (AppMessage.EnquiryMessage)msg;
                  char msg_type = enquiry_msg.getType();
                  switch (msg_type)
                  {
                        case AppConstants.ENQUIRY_REQUEST:
System.out.println("Car ["+id.intValue()+"] received request ");
                        if (((JamDetection)jam).isJamDetected())
                  {
                              DatagramPacket dp = new
DatagramPacket(genResponseMsg(),AppConstants.ENQUIRY_MESSAGE_SIZE,
NetAddress.ANY.getIP(),AppConstants.ENQUIRY_PORT);
                              sending_socket.send(dp);
                        udp.send(response.toBytes(),
NetAddress.ANY.getIP(),AppConstants.ENQUIRY_PORT,
AppConstants.ENQUIRY_PORT, Constants.NET_PRIORITY_NORMAL);
                        }
                        break;
                        case AppConstants.ENQUIRY_RESPONSE:
// Response received, if time not expired count it, else discart it
                              if (JistAPI.getTime() <= lrsTime +
                                                REQUST_EXPIRE_TIME)
                                    addResponse(enquiry_msg);
                              break;
                        default: System.out.println("incorect
                  message type receive on enquiry request port");
                  return;
                  }
          }
      };
```

```
		udp.addSocketHandler(AppConstants.ENQUIRY_PORT, handler);
	}

	public AppCIGDPInterface getProxy() { return this.self; }
}

package jist.swans.app.MyProtocol;

import jist.runtime.JistAPI;
import jist.swans.field.Field;
import jist.swans.field.FieldInterface;

public interface DistanceInterface extends JistAPI.Proxiable {
	public void CalculateDistance();
	public void start();
}

import jist.runtime.JistAPI;
import jist.swans.misc.Location;
import jist.swans.Constants;

public class Distance implements DistanceInterface
{
	private final int NODE_ID = 1;
	private long distanceTable[];
	private Integer id;
	private Location locPrev;
	private Location locCurr;
	private FieldInterface field;
	private DistanceInterface self;

	public Distance(FieldInterface fieldEntity, int id) {
		//if(!JistAPI.isEntity(fieldEntity)) throw new
			IllegalArgumentException("entity expected");
	this.field = fieldEntity;
	this.id = new Integer(id);
	Location loc=((Field)field).getRadioData(this.id).getLocation();
	locPrev = new Location.Location2D(loc.getX(),loc.getY());
	locCurr = new Location.Location2D(loc.getX(),loc.getY());

	distanceTable = new long[AppConstants.DISTANCE_TABLE_SIZE];
	long initDist =
	       AppConstants.INIT_DISTANCE/AppConstants.DISTANCE_TABLE_SIZE;
		for (int i=0; i<AppConstants.DISTANCE_TABLE_SIZE; i++) {
			distanceTable[i] = initDist;
		}
	self=JistAPI.proxy(this,DistanceInterface.class);
	}
	public DistanceInterface getProxy() { return this.self; }

	public void setFieldEntity(FieldInterface fieldEntity){
		if(!JistAPI.isEntity(fieldEntity)) throw new
			 IllegalArgumentException("entity expected");
		this.field = fieldEntity;
	}
	public void setId(int id) { this.id = new Integer(id); }
```

```
	public void CalculateDistance() {
		JistAPI.sleep(AppConstants.DISTANCE_CALCULATE_INTERVAL);
		self.CalculateDistance();

		locPrev = locCurr;
		locCurr = ((Field)field).getRadioData(id).getLocation();

		JistAPI.getTime());
		for (int i=0; i<AppConstants.DISTANCE_TABLE_SIZE-1; i++)
			distanceTable[i] = distanceTable[i+1];
		distanceTable[AppConstants.DISTANCE_TABLE_SIZE-1] =
		Math.round(locCurr.distance(locPrev));
	}

	public void start() { self.CalculateDistance(); }

	public long getDistance() {
		long distance = 0;
		for(int i=0; i<AppConstants.DISTANCE_TABLE_SIZE; i++)
			distance+=distanceTable[i];
		return distance;
	}
}

package jist.swans.app.MyProtocol;

public interface EnquiryInterface extends JistAPI.Proxiable
{
	public void doEnquiry();
	public void start();
	public void doListen();
}

import java.net.DatagramPacket;
import java.net.InetAddress;
import jist.swans.misc.Location;
import jist.swans.misc.Message;
import jist.runtime.JistAPI;
import jist.swans.net.NetAddress;
import jist.swans.trans.TransInterface;
import jist.swans.field.*;
import jist.swans.Constants;

public class Enquiry implements EnquiryInterface {
	/** Switch whether to show debug info or not */
	private final boolean DEBUG_ENQUIRY = true;
	/** How much to wait for responses */
	private final long REQUST_EXPIRE_TIME = 3*Constants.SECOND;
	/** Number of cars in one lane above which a jam is considered */
	private final int CARS_COUNT_FOR_JAM = 10;

	/** self-referencing proxy entity. */
	private EnquiryInterface self;
	/** Udp socket for sending udp messages */
	private UdpSocket sending_socket;
```

```
	/** Referenc to Field entity */
	private FieldInterface field;
	/** Reference to UDP entity. */
	private TransInterface.TransUdpInterface udp;
	/** Reference to jam detection entity */
	private JamDetectionInterface jam;
	/** Self referencing id number */
	private Integer id;
	/** Last request send time */
	private long lrsTime = 0;
	/** count received responses */
	private int responses = 0;

	public Enquiry(UdpSocket sending_socket,FieldInterface field,
TransInterface.TransUdpInterface udp, JamDetectionInterface jam, int id)
	{
		this.sending_socket = sending_socket;
		this.self = JistAPI.proxy(this, EnquiryInterface.class);
		this.field = field;
		if(!JistAPI.isEntity(udp)) throw new
			IllegalArgumentException("expected entity");
		this.udp = udp;
		this.jam = jam;
		this.id = new Integer(id);
	}

	public EnquiryInterface getProxy() { return this.self; }

	public void genDissMsg() {
		JistAPI.sleep(REQUST_EXPIRE_TIME+1);
		if (JistAPI.getTime() == lrsTime + REQUST_EXPIRE_TIME+1) {
			if (responses > CARS_COUNT_FOR_JAM) genJamMsg();
		}
	}
	public void resetEnquiryParams() {
		lrsTime = JistAPI.getTime();
		responses = 0;
	}

	public byte[] genResponseMsg()
	{
		Location loc = ((Field)field).getRadioData(id).getLocation();
		Direction dir = new Direction();
		AppMessage.EnquiryMessage response =  new
	AppMessage.EnquiryMessage(AppConstants.ENQUIRY_RESPONSE,loc,dir,-1,
			AppConstants.JAM_EVENT_CODE, JistAPI.getTime());
		return response.toBytes();
	}

	public void addResponse(AppMessage.EnquiryMessage m){responses++;}

	public AppMessage.EnquiryMessage genRequestMsg() {
		Location loc = ((Field)field).getRadioData(id).getLocation();
		Direction dir = new Direction();
		AppMessage.EnquiryMessage response =  new
	AppMessage.EnquiryMessage(AppConstants.ENQUIRY_REQUEST,loc, dir,-1,
```

```
                          AppConstants.NO_EVENT_CODE, JistAPI.getTime());
            return response;
      }

      public void doEnquiry() {
            JistAPI.sleep(AppConstants.ENQUIRE_INTERVAL);
            self.doEnquiry();
            try {
              DatagramPacket dp = new DatagramPacket(
                  genRequestMsg(),AppConstants.ENQUIRY_MESSAGE_SIZE,
            NetAddress.ANY.getIP(),AppConstants.ENQUIRY_PORT);
                  sending_socket.send(dp);
                  resetEnquiryParams();
                  AppMessage.EnquiryMessage msg = genRequestMsg();
                  Message payload = new MessageBytes(msg.toBytes());
                  TransUdp.UdpMessage(AppConstants.ENQUIRY_PORT,
                             AppConstants.ENQUIRY_PORT, payload);

                  udp.send(payload,NetAddress.ANY, ENQUIRY_PORT,
                              ENQUIRY_PORT,NET_PRIORITY_NORMAL);
                  System.out.println("Sending message");
            }
            catch(Exception e) {e.printStackTrace();}
      }

      public void start() { self.doEnquiry();}

      public void doListen() {
          JistAPI.sleep(Constants.SECOND);
          AppMessage.EnquiryMessage msg=new AppMessage.EnquiryMessage();
          udp.receive(msg)
      }
}

package jist.swans.app.MyProtocol;

public interface JamDetectionInterface extends JistAPI.Proxiable
{
      public void start();
      public void JamDetect();
}
import jist.runtime.JistAPI;
import jist.swans.app.MyProtocol.JamDetectionInterface;
import jist.swans.field.FieldInterface;
import jist.swans.misc.Location;
import jist.swans.Constants;

public class JamDetection implements JamDetectionInterface {
      private boolean jam;
      private Distance distance;
      private JamDetectionInterface self;
      private int ind;

      public JamDetection(FieldInterface fieldEntity, int id){
            ind = id;
          this.jam = false;
```

```
            distance = new Distance(fieldEntity, id);
            self = (JamDetectionInterface)JistAPI.proxy(this,
                    JamDetectionInterface.class);
        }

        public JamDetectionInterface getProxy() { return self; }

        public void JamDetect() {
                JistAPI.sleep(AppConstants.JAMDETECTION_INTERVAL);
                self.JamDetect();
                JistAPI.getTime()/Constants.SECOND);
                if (distance.getDistance() < AppConstants.JAM_DISTANCE)
                        jam = true;
                else  jam = false;
        }

        public void start() {
            self.JamDetect();
            distance.getProxy().start();
        }

        public boolean isJamDetected() {return jam;}
}

package jist.swans.app.MyProtocol;
import java.nio.ByteBuffer;

import jist.swans.misc.Message;
import jist.swans.misc.Location;
import jist.runtime.JistAPI;

public abstract class AppMessage implements Message {
        public static class EnquiryMessage extends AppMessage       {
                final static int ENQUIRY_MESSAGE_SIZE = 30;

                private char type; // 0 -> request, not 0 -> response
                private Location loc = new Location.Location2D(0,0);
                private Direction dir = new Direction();
                private int segment;
                private int code;
                private long time;

                public EnquiryMessage() {
                        type = 0;
                        segment = -1;
                        code = AppConstants.INVALID_EVENT_CODE;
                        time = JistAPI.getTime();
                }

                public EnquiryMessage(char type, Location loc, Direction dir,
int segment, int code, long time) {
                        this.type = type;
                        this.loc = loc;
                        this.dir = dir;
                        this.segment = segment;
                        this.code = code;
```

```
            this.time = time;
      }

      public char getType() { return this.type; }

      public Location getLocation() {return this.loc; }

      public int getCode() {return this.code;}

      public int getSegment() { return this.segment;  }

      public long getTime() { return this.time; }

      public int getSize(){return ENQUIRY_MESSAGE_SIZE;}

      public byte[] toBytes() {
        ByteBuffer bb=ByteBuffer.allocate(ENQUIRY_MESSAGE_SIZE);
            bb.putChar(type);
            bb.putFloat(loc.getX());
            bb.putFloat(loc.getY());
            bb.putFloat(loc.getHeight());
            bb.putInt(segment);
            bb.putInt(code);
            bb.putLong(time);

            return bb.array();
      }
}

public static class DisseminationMessage extends AppMessage
{
      private static final int DISSEMINATION_MESSAGE_SIZE = 36;
      public static final char ACTIVE = 1;
      public static final char PASSIVE= 0;

      private char type; // only 0 is valid for now
      private Location cur_loc = new Location.Location2D(0,0);
      private long reg_time; // time of registration
      private int segment;
      private Direction dir = new Direction();
      private char status; // 0 -> Passive; not 0 -> Active
      private int code;
      private int ttl;

      public DisseminationMessage(char type, Location loc, long
time, int segment,Direction dir, char status, int code, int ttl) {
            this.type = type;
            this.cur_loc = loc;
            this.reg_time = time;
            this.segment = segment;
            this.dir = dir;
            this.status = status;
            this.code = code;
            this.ttl = ttl;
      }
```

```
            public void setLocation(Location newLoc) {cur_loc = newLoc;}

            public void decrementTtl(int dec) {
                  ttl -= dec;
                  if (ttl < 0) ttl = 0;
            }

            public void setStatus(char newStatus) {status = newStatus; }

            public int getSize(){return DISSEMINATION_MESSAGE_SIZE;}
      }
}

package jist.swans.app.MyProtocol;
import jist.swans.Constants;
public final class AppConstants
{
      public static final long INIT_DISTANCE = 900;
      public static final int  MINUTES_NUMBER = 3;
      public static final int  SECONDS_IN_MINUTE = 60;
      public static final int  DISTANCE_TABLE_SIZE =
                        MINUTES_NUMBER*SECONDS_IN_MINUTE;
      public static final long DISTANCE_CALCULATE_INTERVAL =
                        Constants.SECOND;
      public static final long JAMDETECTION_INTERVAL = 10 *
                        Constants.SECOND;
      public static final long JAM_DISTANCE = 300;
      public static final long ENQUIRE_INTERVAL = 20*Constants.SECOND;
      final static int ENQUIRY_MESSAGE_SIZE = 30;
      public static final int ENQUIRY_PORT = 5000;
      public static final int ENQUIRY_LISTEN_PORT = 5001;
      public final byte[] BROADCAST_ADDRESS = new byte[] {-1,-1,-1,-1};
      /** Event codes */
      public static final int NO_EVENT_CODE = 0;
      public static final int JAM_EVENT_CODE = 1;
      public static final int INVALID_EVENT_CODE = -1;
      /** Message types */
      public static final char INVALID_MESSAGE_TYPE = 0;
      public static final char ENQUIRY_REQUEST = 1;
      public static final char ENQUIRY_RESPONSE= 2;
      public static final char DISSEMINATION_MESSAGE = 3;

}
```

b) Програма-драйвер за зареждане на симулацията

```
package driver;

import java.io.IOException;
import jist.runtime.JistAPI;
import jist.swans.Constants;
import jist.swans.field.Fading;
import jist.swans.field.Field;
import jist.swans.field.Mobility;
import jist.swans.field.PathLoss;
import jist.swans.field.Placement;
import jist.swans.field.StreetMobilityOD;
import jist.swans.field.StreetPlacementRandom;
import jist.swans.mac.Mac802_11;
import jist.swans.mac.MacAddress;
import jist.swans.mac.MacInterface;
import jist.swans.misc.Location;
import jist.swans.misc.Mapper;
import jist.swans.misc.Message;
import jist.swans.misc.MessageBytes;
import jist.swans.net.NetAddress;
import jist.swans.net.NetIp;
import jist.swans.net.NetMessage;
import jist.swans.net.PacketLoss;
import jist.swans.radio.RadioInfo;
import jist.swans.radio.RadioNoise;
import jist.swans.radio.RadioNoiseAdditive;
import jist.swans.radio.RadioNoiseIndep;
import jist.swans.trans.TransUdp;
import jist.swans.app.MyProtocol.*;

/**
* Generic simulation configurator. Uses JistExperiment for settings.
*/
public class MyDriver {
    private static Vector memoryConsumption = new Vector();
    private final static boolean VERBOSE = false;
/**
* Add node to the field and start it.
*/
public static void addNode(JistExperiment je, int i, Vector nodes, Field
field, Placement place, RadioInfo.RadioInfoShared radioInfo, Mapper
protMap, PacketLoss inLoss, PacketLoss outLoss, Mobility mobility,
VisualizerInterface v) {
    RadioNoise radio;
    Location location;
    if (nodes!=null) {
         switch (je.radioNoiseType)
         {
            case Constants.RADIO_NOISE_INDEP:
                  radio = new RadioNoiseIndep(i, radioInfo);
                  break;
            case Constants.RADIO_NOISE_ADDITIVE:
                  radio = new RadioNoiseAdditive(i, radioInfo);
                  break;
```

```
        default:
              throw new RuntimeException("Invalid radio model!");
        }
      location = place.getNextLocation();
      field.addRadio(radio.getRadioInfo(), radio.getProxy(), location);

     field.startMobility(radio.getRadioInfo().getUnique().getID());
   }
   else  {
      radio = new RadioNoiseIndep(i, radioInfo);
      location = place.getNextLocation();
      field.addRadio(radio.getRadioInfo(), radio.getProxy(), location);
      field.startMobility(radio.getRadioInfo().getUnique().getID());
      return;
   }

   mac = new Mac802_11(new MacAddress(i),radio.getRadioInfo());
   MacInterface macProxy = null;
   switch (je.mac)
   {
           case Constants.MAC_802_11:
               mac = new Mac802_11(new MacAddress(i),
                       radio.getRadioInfo());
               macProxy = ((Mac802_11)mac).getProxy();
               break;
           case Constants.MAC_DUMB:
               mac = new Mac802_11(new MacAddress(i),
                                   radio.getRadioInfo());
                 macProxy = ((MacDumb)mac).getProxy();
                 break;
   }
   final NetAddress address = new NetAddress(i);
   NetIp net = new NetIp(address, protMap, inLoss, outLoss);
   TransUdp udp = new TransUdp();
   AppCIGDP app = new AppCIGDP();
   // node entity hookup
   radio.setFieldEntity(field.getProxy());
   radio.setMacEntity(macProxy);
   byte intId = net.addInterface(macProxy,new MessageQueue.
NoDropMessageQueue (Constants.NET_PRIORITY_NUM, NetIp.MAX_QUEUE_LENGTH));
   if (mac instanceof Mac802_11)    {
     ((Mac802_11)mac).setRadioEntity(radio.getProxy());
     ((Mac802_11)mac).setNetEntity(net.getProxy(), intId);
   }
   udp.setNetEntity(net.getProxy());
   net.setProtocolHandler(Constants.NET_PROTOCOL_UDP, udp.getProxy());
   app.setFieldEntity(field);
   app.setId(i);
   app.setUdpEntity(udp.getProxy());
   app.getProxy().start();
}  //method: addNode



/**
```

```
* Constructs field and nodes with given command-line options
*/
private static void buildField(JistExperiment je, final Vector nodes )
{
      Mobility mobility = null;
      Location.Location2D tr;
      Location.Location2D bl;
      if (je.seed == -1) r = new Random();
      else r = new Random(je.seed);
      // set random object for use in other simulation objects
      je.random = r;
      switch(je.mobility)
      {
            case Constants.MOBILITY_STATIC:
                  mobility = new Mobility.Static();
                  break;
            case Constants.MOBILITY_WAYPOINT:
                  mobility = new Mobility.RandomWaypoint(je.field,
je.pause_time,
                  je.granularity, je.max_speed, je.min_speed);
                  mobility = new Mobility.RandomWaypoint(je.field,
je.mobilityOpts);
                  break;
            case Constants.MOBILITY_WALK:
                  mobility = new Mobility.RandomWalk(je.field,
je.mobilityOpts);
                  break;
            case Constants.MOBILITY_STRAW_SIMPLE:
                  tr = new Location.Location2D(je.maxLong, je.maxLat);
                  bl = new Location.Location2D(je.minLong, je.minLat);
                  mobility = new StreetMobilityRandom(je.segmentFile,
je.streetFile,
                              je.shapeFile, je.degree, je.probability,
je.granularity, bl, tr, r);
                  break;
            case Constants.MOBILITY_STRAW_OD:
                  tr = new Location.Location2D(je.maxLong, je.maxLat);
                  bl = new Location.Location2D(je.minLong, je.minLat);
                  mobility = new StreetMobilityOD(je.segmentFile,
je.streetFile,
                              je.shapeFile, je.degree, bl, tr, r);
                  break;
            default:
            throw new RuntimeException("unknown node mobility model");
      }
      Spatial spatial = null;

      Location.Location2D corners[] = new Location.Location2D[4];
      if (je.mobility != Constants.MOBILITY_STRAW_SIMPLE && je.mobility
!= Constants.MOBILITY_STRAW_OD)
      {
            corners[0] = new Location.Location2D(0, 0);
            corners[1] = new Location.Location2D(je.field.getX(), 0);
            corners[2] = new Location.Location2D(0, je.field.getY());
            corners[3] = new Location.Location2D(je.field.getX(),
je.field.getY());
```

```
      }
      else {
            je.sm = (StreetMobility)mobility;
            StreetMobility smr = (StreetMobility)mobility;
            corners = (Location.Location2D[])smr.getBounds();
      }

      switch(je.spatial_mode)
      {
            case Constants.SPATIAL_LINEAR:
                  spatial = new Spatial.LinearList(corners[0],
corners[1], corners[2], corners[3]);
                  break;
            case Constants.SPATIAL_GRID:
                  spatial = new Spatial.Grid(corners[0], corners[1],
corners[2], corners[3], je.spatial_div);
                  break;
            case Constants.SPATIAL_HIER:
                  spatial = new Spatial.HierGrid(corners[0], corners[1],
corners[2], corners[3], je.spatial_div);
                  break;
            default:
            throw new RuntimeException("unknown spatial binning model");
      }
      if(je.wrapField) spatial = new Spatial.TiledWraparound(spatial);

      PathLoss pl;
      switch (je.pathloss)
      {
            case Constants.PATHLOSS_FREE_SPACE:
                  pl = new PathLoss.FreeSpace();
            case Constants.PATHLOSS_TWO_RAY:
                  pl = new PathLoss.TwoRay();
                  break;
            default:
            throw new RuntimeException("Unsupported pathloss model!");
      }
      Field field = new Field(spatial, new Fading.None(), pl,
                  mobility, Constants.PROPAGATION_LIMIT_DEFAULT);
      RadioInfo.RadioInfoShared radioInfo = RadioInfo.createShared(
                  je.frequency,
                  je.bandwidth,
                  je.transmit,
                  je.gain,
                  Util.fromDB(je.sensitivity),
                  Util.fromDB(je.threshold),
                  je.temperature,
                  je.temperature_factor,
                  je.ambiant_noise);

      RadioInfo.RadioInfoShared noRadio = RadioInfo.createShared(
                  je.frequency,
                  je.bandwidth,
                  0,
                  0,
                  Util.fromDB(10000),
```

```
                Util.fromDB(10000),
                je.temperature,
                je.temperature_factor,
                je.ambiant_noise);

    Mapper protMap = new Mapper(Constants.NET_PROTOCOL_UDP);
    protMap.mapToNext(Constants.NET_PROTOCOL_UDP);
    PacketLoss outLoss = new PacketLoss.Zero();
    PacketLoss inLoss = null;
    switch(je.loss){
          case Constants.NET_LOSS_NONE:
                inLoss = new PacketLoss.Zero();
                break;
          case Constants.NET_LOSS_UNIFORM:
                inLoss = new
PacketLoss.Uniform(Double.parseDouble(je.lossOpts));
                break;
          default:
          throw new RuntimeException("unknown packet loss model");
    }
    Placement place = null;
    switch(je.placement) {
          case Constants.PLACEMENT_RANDOM:
                place = new Placement.Random(je.field);
                break;
          case Constants.PLACEMENT_GRID:
                je.setPlacementOpts("");
                place = new Placement.Grid(je.field, je.placementOpts);
                break;
          case Constants.PLACEMENT_STREET_RANDOM:
                StreetMobility smr= (StreetMobility)mobility;
                Location.Location2D bounds[] =
(Location.Location2D[])smr.getBounds();
                place = new StreetPlacementRandom(bounds[0], bounds[3],
smr, je.driverStdDev);
                break;
          default:
          throw new RuntimeException("unknown node placement model");
    }
    int i;
    // create each mobile node
    for (i=1; i<=je.nodes; i++)   {
          if (je.penetrationRatio<1.0f &&
           (je.penetrationRatio*i)-Math.floor(je.penetrationRatio*i) >=
je.penetrationRatio){
addNode(je, i, null, field, place, noRadio, protMap, inLoss, outLoss,
                              mobility, v);
          }
          else // create nodes that will transmit
          {
                addNode(je, i, nodes, field, place, radioInfo, protMap,
inLoss, outLoss, mobility, v);
          }
    }
```

```
Field staticField = new Field(spatial, new Fading.None(), new
PathLoss.FreeSpace(),   new Mobility.Static(),
Constants.PROPAGATION_LIMIT_DEFAULT /* check */);

        RadioInfo.RadioInfoShared staticRadioInfo =
RadioInfo.createShared(
                  je.frequency,
                  je.bandwidth,
                  je.staticTransmit,
                  je.staticGain,
                  Util.fromDB(je.staticSensitivity), // TODO update these
with decent values
                  Util.fromDB(je.threshold),
                  je.temperature,
                  je.temperature_factor,
                  je.ambiant_noise);
        int max = je.staticNodes+i;
        for (int j=i; j <=max; j++)
        {
            addNode(je, j, nodes, staticField, staticPlace,
staticRadioInfo, protMap, inLoss,
                   outLoss, null, v);
         }
      }
      Vector sources = new Vector();
      int num_sources = 0;
      final int MAX_SOURCES = je.transmitters;
      boolean nodelist[] = new boolean[nodes.size()];

      Random myRandom = new Random(0);

      if (nodes.size() > 0){
            int index = myRandom.nextInt(nodes.size());

            while (num_sources < MAX_SOURCES){
                  sources.add(nodes.elementAt(index));
                  nodelist[index] = true;
                  do{
                        index = myRandom.nextInt(nodes.size());

                  } while(nodelist[index]== true && num_sources <
MAX_SOURCES-1);
            num_sources++;
            }
            JistAPI.sleep(je.startTime*Constants.SECOND);
            if (je.useCBR)
            {
                  //generateCBRTraffic(je, sources, nodes, myRandom);
throw new RuntimeException("CBR not supported :) Use CIGDP instead");
            }
            }
      } // end if nodes
      else // get memory usage every 5 %
      {
            int numTotalIters = 20;
```

```
            long delayInterval = (long)Math.ceil((double)je.duration *
(double)Constants.SECOND / (double)numTotalIters);
            for (int j = 0; j < numTotalIters; j++)
            {
                  memoryConsumption.add(new
Long(jist.runtime.Util.getUsedMemory()));
                  JistAPI.sleep(delayInterval);
            }
      }
} // buildField

/**
* Display statistics at end of simulation.
*/
public static void showStats(Vector nodes, final JistExperiment je,
            Date startTime, final long freeMemory, final String name){

      String output="";
      Date endTime = new Date();
      long elapsedTime = endTime.getTime() - startTime.getTime();

      output+=(float)elapsedTime/1000+"\t";

      output+=je.nodes+"\t";
      if (je.mobility==Constants.MOBILITY_STRAW_SIMPLE || je.mobility ==
Constants.MOBILITY_STRAW_OD)
            output+=je.sm.getArea()+"\t";
      else  { output+=je.getFieldX()+"\t"+je.getFieldY()+"\t"; }
      output+=je.transmitters+"\t";
      output+=je.exponent+"\t";
      output+=(je.useCBR?((60.0 *
je.cbrRate)/je.cbrPacketSize):je.sendRate)+"\t";
      if (je.mobility==Constants.MOBILITY_STRAW_SIMPLE)
output+=je.probability+"\t";
      else output+=je.pause_time+"\t";
      output+=je.protocol+"\t";
      if (je.penetrationRatio > 0)  {
            System.out.println("-------------");
            System.out.println("Packet stats:");
            System.out.println("-------------");
      }
      else {output+=printMemStats();}

      System.out.println("freemem:  "+Runtime.getRuntime().freeMemory());
      System.out.println("maxmem:   "+Runtime.getRuntime().maxMemory());
      System.out.println("totalmem:"+Runtime.getRuntime().totalMemory());
      long usedMem = Runtime.getRuntime().totalMemory()-
                        Runtime.getRuntime().freeMemory();
      System.out.println("used:     "+usedMem);
      System.out.println("start time  : "+startTime);
      System.out.println("end time    : "+endTime);
      System.out.println("elapsed time: "+elapsedTime);
      System.out.flush();

      if(je.mobility==Constants.MOBILITY_STRAW_SIMPLE || je.mobility ==
Constants.MOBILITY_STRAW_OD)  {
```

```
      output+=je.sm.printAverageSpeed(je.duration+je.startTime+je.resolut
ionTime, VERBOSE)+"\t";
            output+=je.penetrationRatio;
      }
      else {
            output+=je.fieldX+"\t";
            output+=je.fieldY+"\t";
      }

      String newname = name.split("\\.")[0]+".txt";
      try{ Util.writeResult(newname, output);}
      catch (IOException ioe) {ioe.printStackTrace();}
      //clear memory
      nodes = null;
}

private static String printMemStats() {
    String output="";
    double avg;
    long sum = 0;
    for (int i = 0; i < memoryConsumption.size(); i++) {
        sum+=((Long)memoryConsumption.get(i)).longValue();
    }
    avg = (double)sum/memoryConsumption.size();
    double stdSum = 0.0;
    //std deviation
    for (int i = 0; i < memoryConsumption.size(); i++)   {
        double diff = ((Long)memoryConsumption.get(i)).longValue() - avg;
        stdSum += diff*diff;
    }
    output+= (Math.sqrt(stdSum/memoryConsumption.size())) + "\t";
    return output;
}
public static void main(String[] args) {
      try {
            final JistExperiment je = (JistExperiment)(
                                          Util.readObject(args[0]));
            je.setField();
            final long freeMemory = Runtime.getRuntime().freeMemory();
            long endTime = je.startTime+je.duration+je.resolutionTime;
            if(endTime>0) {JistAPI.endAt(endTime*Constants.SECOND);}
            final Vector nodes = new Vector(je.nodes);
            final Date startTime = new Date();
            buildField(je, nodes);
            final String name = args[0];
            JistAPI.runAt(new Runnable(){
              public void run() {
                  showStats(nodes, je, startTime, freeMemory, name);
              }
            }, JistAPI.END);
      }

      catch(IOException e){System.out.println(e.getMessage());}
  }
} // END of MyDriver.class
```

c) Програмен код на Garmin Map Parser

```
/**
 * File:         Coordinates.java
 * Description:  Encapsulate geographical coordinates
*/

public class Coordinates {
  int Longitude;
  int Latitude;

  public Coordinates()  {
      this.Latitude = 0;
      this.Longitude = 0;
  }

  public Coordinates(int lat,int lon) {
      this.Latitude  = lat;
      this.Longitude = lon;
  }

  public void setCoordinates(int lat,int lon)  {
      this.Latitude = lat;
      this.Longitude = lon;
  }
}


/**
 * File:         StreetName.java
 * Description:  Encapsulate street's name
 * Author:       Mitko Shopov
 *                       Technical University of Plovdiv, Bulgaria
 *                       mshopov@tu-plovdiv.bg
*/

public class StreetName{
   char[] prefix = new char[2];
   char[] name = new char[30];
   char[] type = new char[4];
   char[] suffix = new char[2];

   public StreetName(String streetName)
   {
         //TODO: Prefix, sufix & type not implemented yet
         char[] sn= new char[streetName.length()];
         sn = streetName.toCharArray();
         for(int i=0; i<streetName.length(); i++) name[i] = sn[i];
         for(int i=streetName.length(); i < 30; i++) name[i] = ' ';
   }

   public void setStreetName(String streetName)
   {
         //TODO: Prefix, sufix & type not implemented yet
         char[] sn= new char[streetName.length()];
         sn = streetName.toCharArray();
```

```
        name = sn;
    }
}

/**
 * File:         ShapePoints.java
 * Description:  Encapsulate road's shape
*/

public class ShapePoints {
   int num_points;
   Coordinates[] points;

   public ShapePoints()   {
         this.num_points = 0;
         points = null;
   }

   public ShapePoints(int num_points, Coordinates[] points)   {
         this.num_points = num_points;
         if (this.num_points == 0) this.points = null;
         else this.points = new Coordinates[num_points];
         for (int i=0; i<this.num_points; i++)
                        this.points[i] = points[i];
   }

   public void setShapePoints(int num_points, Coordinates[] points) {
         this.num_points = num_points;
         if (this.num_points == 0) this.points = null;
         else this.points = new Coordinates[num_points];
         for (int i=0; i<this.num_points; i++)
                        this.points[i] = points[i];
   }

   public int getNumberOfPoints() {
         return this.num_points;
   }
}

 /**
 * File:         RoadSegment.java
 * Description:  Represent one road segment
*/

public class RoadSegment {
   int StartAddressLeft;
   int EndAddressLeft;
   int StartAddressRight;
   int EndAddressRight;
   //TODO: Addresses not implemented yet
   int StreetIndex;
   int ShapeIndex;
   Coordinates StartPoint = new Coordinates(0,0);
   Coordinates EndPoint = new Coordinates(0,0);
   char RoadClass;
```

```
   //TODO: Road class is not implemented yet. It's fixed as 14 to be a
valid RoadClass for STRAW

   public RoadSegment(int stri, int shi, Coordinates start, Coordinates
end, char rc) {
         this.StartAddressLeft = 0;
         this.EndAddressLeft = 0;
         this.StartAddressRight = 0;
         this.EndAddressRight = 0;
         this.StreetIndex = stri;
         this.ShapeIndex = shi;
         this.StartPoint = start;
         this.EndPoint = end;
         this.RoadClass = 14; //rc;
   }

   public void setRoadSegment(int stri, int shi, Coordinates start,
Coordinates end, char rc)   {
         this.StartAddressLeft = 0;
         this.EndAddressLeft = 0;
         this.StartAddressRight = 0;
         this.EndAddressRight = 0;
         this.StreetIndex = stri;
         this.ShapeIndex = shi;
         this.StartPoint = start;
         this.EndPoint = end;
         this.RoadClass = 14; //rc;
   }
}


/**
 * File:         Polygon.java
 * Description:  Encapsulate polygon's structure
*/

class Polygon {
  char type;
  char[] name = new char[30];
  int num_points;
  Coordinates[] points;

  public Polygon() {
         this.type = '0';
         for (int i=0; i<30; i++) this.name[i] = 0;
         this.num_points = 0;
  }

  public Polygon(char type, char[] name, int num_points, Coordinates[]
points) {
         this.type = type;
         this.name = name;
         this.num_points = num_points;
         this.points = new Coordinates[num_points];
         this.points = points;
  }
```

```
}

/**
 * File:         MapParser.java
 * Description:  Parse Garmin Maps data to format convenient for STRAW
 *     Currently this class generates four STRAW's input files:
 *                   chains.dat, names.dat, segments.dat, polygons.dat
*/

import java.io.*;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;

public class MapParser {
      private final static String SEGMENTFILE = "segments.dat";
      private final static String STREETFILE = "names.dat";
      private final static String SHAPESFILE = "chains.dat";
      private final static String POLYGONSFILE = "polygons.dat";
      private final static boolean DEBUG = false;

      final int ROAD_SEGMENT_SIZE = 44;
      final int MAX_ROAD_SEGMENTS = 20000; // big enough
      final int STREET_NAME_SIZE = 38;
      final int POLYGON_POINTS_NUM = 100;

      private LineNumberReader linereader = null;
      private FileReader reader = null;
      private String str;

      private RoadSegment segments[]= new RoadSegment[MAX_ROAD_SEGMENTS];
      private ShapePoints shapes[]  = new ShapePoints[MAX_ROAD_SEGMENTS];
      private StreetName  names[]    = new StreetName[MAX_ROAD_SEGMENTS];

      private int segments_ind = -1;
      private int shapes_ind   = -1;
      private int names_ind    = -1;

      /**
       * @param args name and path of file containing the GPS data
       */
      public static void main(String[] args) {
            if (args.length != 1){
                  System.out.println("main takes one parameter, the name
                                             and path of the input file");
                  return;
            }

            MapParser mp = new MapParser();
            mp.parseFile(args[0]);
      }

      private void writeSegments()  {
            try {
      FileOutputStream fos = new FileOutputStream(SEGMENTFILE,true);
                  DataOutputStream dos = new DataOutputStream(fos);
```

```
			for (int i=0; i<=segments_ind; i++)
				dos.write(serializeSegments(segments[i]).array());
			dos.close();
			fos.close();
		}
		catch (IOException exception)
		  System.out.println("MapParser::writeSegments: I/O Error ");
	}

	private void writeNames(){
		try{
		FileOutputStream fos = new FileOutputStream(STREETFILE,true);
		DataOutputStream dos = new DataOutputStream(fos);

		for (int i=0; i<names_ind+1; i++) {
				dos.write(serializeNames(names[i]).array());
		}
		dos.close();
		fos.close();
	}
	catch (IOException exception)
		System.out.println("MapParser::writeNames: I/O Error ");
	}

	private void writeShapes(){
		try{
		FileOutputStream fos = new FileOutputStream(SHAPESFILE,true);
		DataOutputStream dos = new DataOutputStream(fos);

			// Write number of shapes in the begining of file
			ByteBuffer bb = ByteBuffer.allocate(4);
			bb.order(ByteOrder.LITTLE_ENDIAN);
			bb.putInt(shapes_ind+1);
			dos.write(bb.array());
			for (int i=0; i<=shapes_ind; i++)
				dos.write(serializeShapes(shapes[i]).array());
			dos.close();
			fos.close();
		}
		catch (IOException exception)
		   System.out.println("MapParser::writeShapes: I/O Error ");
	}

	private void writePolygons()  {
		try{
FileOutputStream fosPolygons = new FileOutputStream(POLYGONSFILE,true);
DataOutputStream dosPolygons = new DataOutputStream(fosPolygons);

			//TODO: Not implemented yet :(
			dosPolygons.close();
			fosPolygons.close();
		}
		catch (IOException exception)
		  System.out.println("MapParser::writePolygons: I/O Error ");
}
```

```
private void parseFile(String input){
      try {
            reader =  new FileReader(input);
            linereader = new LineNumberReader(reader);

            while ((str = linereader.readLine()) !=null) {
      if (str.compareTo("[IMG ID]") == 0) parseImgID();
      else if (str.compareTo("[Countries]") == 0) parseCountries();
      else if (str.compareTo("[Regions]") == 0) parseRegions();
      else if (str.compareTo("[Cities]") == 0) parseCities();
      else if (str.compareTo("[ZipCodes]") == 0) parseZipCodes();
      else if (str.compareTo("[POI]") == 0) parsePoi();
      else if (str.compareTo("[POLYLINE]") == 0) parsePolyline();
      else if (str.compareTo("[POLYGON]") == 0) parsePolygon();
      }
      linereader.close();
      reader.close();

      writeSegments();
      writeNames();
      writeShapes();
      }
      catch (IOException exception)
            System.out.println("MapParser::parseFile: I/O Error ");
}

private ByteBuffer serializeNames(StreetName sn) {
      ByteBuffer bb = ByteBuffer.allocate(STREET_NAME_SIZE);
      bb.order(ByteOrder.LITTLE_ENDIAN);
      bb.put((byte)sn.prefix[0]);
      bb.put((byte)sn.prefix[1]);
      for(int i=0; i<30; i++) bb.put((byte)sn.name[i]);
      bb.put((byte)sn.type[0]);
      bb.put((byte)sn.type[1]);
      bb.put((byte)sn.type[2]);
      bb.put((byte)sn.type[3]);
      bb.put((byte)sn.suffix[0]);
      bb.put((byte)sn.suffix[1]);
      return bb;
}

private ByteBuffer serializeShapes(ShapePoints sp){
      ByteBuffer bb = ByteBuffer.allocate(sp.num_points*8+4);
      bb.order(ByteOrder.LITTLE_ENDIAN);
      bb.putInt(sp.num_points);
      for (int i=0; i< sp.num_points; i++) {
            bb.putInt(sp.points[i].Longitude);
            bb.putInt(sp.points[i].Latitude);
      }
      return bb;
}

private ByteBuffer serializeSegments(RoadSegment rs) {
      ByteBuffer bb = ByteBuffer.allocate(ROAD_SEGMENT_SIZE);
      bb.order(ByteOrder.LITTLE_ENDIAN);
      bb.putInt(rs.StartAddressLeft);
```

```
		bb.putInt(rs.EndAddressLeft);
		bb.putInt(rs.StartAddressRight);
		bb.putInt(rs.EndAddressRight);
		bb.putInt(rs.StreetIndex);
		bb.putInt(rs.ShapeIndex);
		bb.putInt(rs.StartPoint.Longitude);
		bb.putInt(rs.StartPoint.Latitude);
		bb.putInt(rs.EndPoint.Longitude);
		bb.putInt(rs.EndPoint.Latitude);
		bb.put((byte)rs.RoadClass);
		byte b = 0;
		bb.put(b); // Add this because STRAW expect 3 more bytes
		bb.put(b);
		bb.put(b);
		return bb;
	}

	private char parseType(String type) {
		type.trim();
		String hex = type.substring(7);
		return (char)Integer.parseInt(hex.trim(),16);
	}

	private String parseLabel(String label){
		label.trim();
		int ind = label.indexOf('~');
		if (ind == -1) return label.substring(6);
		else return label.substring(6,ind) + ' ' +
					label.substring(ind+7);
	}

	private Coordinates parseData(String data){
		// DATA has the following format: (xx.xxxxx,yy.yyyyy)
		data.trim();
		int dot1Ind = data.indexOf('.');
		int commaInd = data.indexOf(',');
		int dot2Ind = data.indexOf('.',commaInd);
		String strLat = data.substring(0,dot1Ind) +
			data.substring(dot1Ind+1,commaInd) + '0';
		String strLon = data.substring(commaInd+1,dot2Ind) +
			data.substring(dot2Ind+1) + '0';
		int lat = Integer.parseInt(strLat);
		int lon = Integer.parseInt(strLon);
		Coordinates point = new Coordinates(lat,lon);
		return point;
	}

	private void updateMinMax(Coordinates point){
	  if (point.Latitude < minLat ) minLat = point.Latitude;
	  if (point.Latitude > maxLat) maxLat = point.Latitude;
	  if (point.Longitude < minLon ) minLon = point.Longitude;
	  if (point.Longitude > maxLon) maxLon = point.Longitude;
	}

	private void parsePolyline()  {
		Coordinates startPoint = new Coordinates();
```

```
Coordinates endPoint = new Coordinates();
Coordinates point = new Coordinates();
Coordinates[] points = new Coordinates[MAX_ROAD_SEGMENTS];
String streetName = null;
char type   = 0;
int shp_num = 0;
int shp_ind = -1;
int nam_ind = -1;
int seg_ind = -1;

try {
      str = linereader.readLine().trim();
      while (str.compareTo("[END]")!=0){
            if ((str.substring(0,4).indexOf("Type"))==0)
                  type = parseType(str);
            else if str.substring(0,5).compareTo("Label"))==0
            {
            streetName = parseLabel(str);
            names[++names_ind] = new StreetName(streetName);
            nam_ind = names_ind;
            }
            else if
             ((str.substring(0,4).compareTo("Data"))==0) {
            seg_ind = 5;
            int closeBraketInd= 0;
            int openBraketInd = 0;
            int lastOpenBraket = 0;

            openBraketInd = str.indexOf('(');
            closeBraketInd = str.indexOf(')');
            lastOpenBraket = str.lastIndexOf('(');
            point =
parseData(str.substring(openBraketInd+1,closeBraketInd));
            startPoint = point;
            updateMinMax(point);
            openBraketInd = str.indexOf('(',closeBraketInd);
            closeBraketInd = str.indexOf(')',openBraketInd);

            while (openBraketInd < lastOpenBraket){
                  point =
parseData(str.substring(openBraketInd+1,closeBraketInd));
                  points[shp_num] = point; shp_num++;
                  updateMinMax(point);
                  openBraketInd =
                        str.indexOf('(',closeBraketInd);
                  closeBraketInd =
                        str.indexOf(')',openBraketInd);
                  }
                  point =
parseData(str.substring(openBraketInd+1,closeBraketInd));
                  endPoint = point;

                  updateMinMax(point);
            } // end parse Data
            str = linereader.readLine().trim();
      }
```

```
                if (shp_num > 0)  {
                shapes[++shapes_ind] = new ShapePoints(shp_num,points);
                shp_ind = shapes_ind;
                }
                if (nam_ind == -1) {
                      names[++names_ind] = new StreetName("No Name");
                      nam_ind = names_ind;
                }
                if (seg_ind != -1) segments[++segments_ind] = new
                RoadSegment(nam_ind,shp_ind,startPoint,endPoint,type);
          }
          catch (IOException exception)
            System.out.println("Error processing file: "+ exception);
    }

    private void parsePolygon(){
          try{
                str = linereader.readLine();
                while ((str.compareTo("[END]")!=0) &&
                      (str.compareTo("[END-Polygon]")!=0))
                      str = linereader.readLine();
          }
          catch (IOException exception)
            System.out.println("Error processing file: "+ exception);
    }

}
```