МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВПО «КубГУ»)**

**Кафедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ МАРШРУТИЗАЦИИ В AD-HOC СЕТЯХ**

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc467494529)

[1 ОСНОВНЫЕ ПОНЯТИЯ ОПРЕДЕЛЕНИЯ AD-HOC СЕТЕЙ 4](#_Toc467494530)

[2 ПРОТОКОЛЫ МАРШРУТИЗАЦИИ В AD-HOC СЕТЯХ 5](#_Toc467494531)

[2.1 КЛАССИФИКАЦИЯ ПРОТОКОЛОВ МАРШРУТИЗАЦИИ 6](#_Toc467494532)

[2.2 ОПИСАНИЕ РАБОТЫ ОСНОВНЫХ ПРОТОКОЛОВ МАРШРУТИЗАЦИИ 6](#_Toc467494533)

[3 СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОТОКОЛОВ МАРШРУТИЗАЦИИ 14](#_Toc467494534)

[4 МОДЕЛИРОВАНИЕ AD-HOC СЕТЕЙ 15](#_Toc467494535)

# 

# ВВЕДЕНИЕ

Одной из перспективных современных технологий передачи данных являются беспроводные сети типа «ad-hoc». Отличительной особенностью беспроводных сетей ad-hoc или самоорганизующихся динамических сетей является то, что узлы сети соединяются «на лету».

С момента начала работы над стандартом появилось множество новых задач, связанных с развертыванием таких беспроводных сетей, как:

* домашние, офисные и заводские сети;
* сети университетских городков;
* муниципальные и коммерческие публичные сети;
* сети транспортных узлов (автомобилей, а в последние несколько лет и самолетов);
* сети, развертываемые в зонах чрезвычайных ситуаций;
* сети военного назначения.

Эти задачи являются исключительно актуальными в современном динамично развивающемся мире, поэтому понимание и исследование принципов работы ad-hoc сетей, моделирование их поведения является безусловно актуальным.

# ОСНОВНЫЕ ПОНЯТИЯ ОПРЕДЕЛЕНИЯ AD-HOC СЕТЕЙ

Для организации самоорганизующихся сетей используют стандарты (протоколы):

1. IEEE 802.11 - WiFi,
2. IEEE 802.15.4 (ZigBee),
3. IEEE 802.15.1 (Bluetooth).

ZigBee – спецификация протоколов сетевого уровня, использующих сервисы нижних уровней — канального и физического. Основаны на стандарте IEEE 802.15.4, описывающем беспроводные персональные вычислительные сети (WPAN). Технология ZigBee позволяет создавать самоорганизующиеся и самовосстанавливающиеся беспроводные сети с автоматической ретрансляцией сообщений, с поддержкой батарейных и мобильных узлов. Сети, созданные при помощи данной технологии, потребляют очень мало энергии — индивидуальные устройства согласно данным сертификации ZigBee позволяют энергобатареям работать два года.

Одна из основных областей применения технологии ZigBee – это беспроводные самоорганизующиеся сети (ad-hoc сети).

Ad-hoc сети – радиосети со случайными стационарными абонентами, реализующие полностью децентрализованное управление при отсутствии базовых станций или опорных узлов. Топология – фиксированная со случайным соединением узлов.

Данные сети обладают некоторыми особенностями, в отличии от проводных сетей, а именно: большое количество устройств (более 10000), возможность перемещения узлов сети, отсутствие централизованного управления, изменение условий распространения радиосигнала, и также, как и для любых беспроводных систем, ограничение полосы пропускания и зоны радиовидимости. В связи с этим традиционные протоколы маршрутизации в данных сетях использоваться не могут.

# ПРОТОКОЛЫ МАРШРУТИЗАЦИИ В AD-HOC СЕТЯХ

Для успешного применения в Ad-hoc сетях протоколы маршрутизации должны обладать следующими качествами:

1. Быть распределенными. Все узлы в сети должны быть способны осуществлять маршрутизацию и не иметь жестко закрепленных за собой функций.

2. Обеспечивать надежную доставку пакетов в условиях постоянно изменяющейся топологии сети.

3. Обеспечивать малое время построения маршрута в условиях постоянно изменяющейся топологии сети.

4. Обеспечивать оперативное обнаружение нарушения маршрута и его восстановление.

5. Не допускать образования петель в маршрутах.

6. Рассылать при функционировании как можно меньший объем служебной информации.

7. Обеспечивать высокую производительность сети при различных ее размерах.

8. Поддерживать требуемый уровень качества обслуживания (QoS)

## КЛАССИФИКАЦИЯ ПРОТОКОЛОВ МАРШРУТИЗАЦИИ

Существует множество вариантов классифицировать протоколы, на рис.1 приведен один из них.

В настоящее время протоколы маршрутизации можно классифицировать следующим образом:

Классификация протоколов маршрутизации в ad-hoc сетях

По критерию определения маршрута

По механизму формирования таблицы маршрутизации

Протоколы географической маршрутизации

Протоколы со сложной метрикой

Протоколы вектора расстояния

Рективные протоколы

Гибридные протоколы

Проактивные протоколы

Рис. 1 Классификация протоколов маршрутизации в ad-hoc сетях

*Проактивные протоколы.* Каждый узел сети строит маршруты на основе служебных сообщений с информацией обо всех изменения топологии сети, и сохраняет их в таблицу маршрутизации.

*Реактивные протоколы.* Составляют маршруты при необходимости. Узел-инициатор рассылает широковещательный запрос. В ответ узел-адресат, высылает сообщение, содержащее необходимый маршрут, который узел-инициатор записывает в свою таблицу маршрутизации.

*Гибридные протоколы.* Данные протоколы комбинируют механизмы работы проактивных и реактивных протоколов. Обычно используются для сетей с большим количеством устройств. Они разбивают сеть на подсети, в которых работает проактивный протокол, а между собой подсети функционирует реактивными методами.

*Протоколы вектора расстояния.* Маршрут считается оптимальным, если он содержит наименьшее число хопов (переходов) между отправителем и адресатом.

*Протоколы со сложной метрикой маршрутов*. Используют комплексную оценку маршрута по нескольким параметрам.

*Протоколы географической маршрутизации*. Узел-отправитель посылает сообщение на географическое местоположения узла-адресата, вместо использования сетевого адреса. Существует различные подходы: однопутевой, многопутевой, и потоковый (flooding-based). Однопутевой подход имеет два метода: «жадной» пересылки и «лицевой» маршрутизации (face routing).

## ОПИСАНИЕ РАБОТЫ ОСНОВНЫХ ПРОТОКОЛОВ МАРШРУТИЗАЦИИ

Наиболее активно используемыми протоколами являются:

* AODV;
* DSDV;
* OLSR;
* DSR.

**Протокол AODV**

AODV (AdhocOn-DemandDistanceVector) – протокол динамической маршрутизации, использующий вектор расстояния. Работа этого протокола выглядит следующим образом: узлу-инициатору требуется передать данные узлу-адресату, если соответствующий маршрут неизвестен, то узел-инициатор посылает широковещательный RREQ запрос для поиска маршрута.

Поля запроса RREQ:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Тип | J | R | G | D | U | Зарезервировано | Кол-во переходов |
| Идентификационный номер сообщения RREQ | | | | | | | |
| IP-адрес узла-адресата | | | | | | | |
| Порядковый номер узла-адресата | | | | | | | |
| IP-адрес узла-инициатора | | | | | | | |
| Порядковый номер узла-инициатора | | | | | | | |

Рис. 2 Формат кадра-запроса для поиска маршрута AODV

Флаг J – устанавливается для широковещательной рассылки.

Флаг R – устанавливается для восстановления маршрута.

Флаг G - устанавливается, если необходимо установить обратный маршрут от узла-адресата до узла-инициатора.

Флаг D – устанавливается, если сообщение необходимо доставить именно узлу-адресату.

Флаг U – устанавливается, если неизвестен порядковый номер узла-адресата.

Запрос передается всем соседям данного узла, которые также отправляют запрос своим соседям. Через некоторое время запрос достигает либо узла-адресата, либо некого узла, которому известен необходимый маршрут. Узлу-инициатору отправляется RREP ответ, передающийся по построенному при рассылке запроса маршруту.

Поля ответа RREP:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип | R | A | Зарезервировано | Префикс размера | Кол-во переходов |
| IP-адрес узла-адресата | | | | | |
| Порядковый номер узла-адресата | | | | | |
| IP-адрес узла-инициатора | | | | | |
| Срок жизни (Lifetime) | | | | | |

Рис. 3 Формат кадра-ответа построения маршрута AODV

Флаг R – устанавливается для восстановления маршрута

Флаг A – устанавливается, если требуется подтверждение

Если флаг А установлен, то в ответ на RREP сообщение, узел-инициатор отправляет подтверждение узлу-адресату - сообщение RREP-ACK, подтверждающее что узел получил RREP сообщение.

Поля сообщения RREP-ACK:

|  |  |
| --- | --- |
| Тип | Зарезервировано |

В результате выстраивается необходимый маршрут. Если ответ отправляет промежуточный узел, то он отправляет еще один ответ узлу-адресату для построения обратного маршрута от узла-адресата до узла-инициатора. Таблицы маршрутизации узлов могут содержать наборы информации различной давности. Для того чтобы определить, как давно произошло обновление таблицы введены порядковые номера узлов, так как для построения маршрута следует использовать более новую информацию. Эта идея заимствована из протокола DSDV. Идея следующая: у каждого узла есть свой порядковый номер, этот номер хранится в таблицах маршрутизации вместе с информацией о маршрутах до данного узла. При каком-либо изменении связей с узлами-соседями, порядковый номер узла увеличивается. Таким образом, при формировании маршрута выбираются узлы с наибольшими номерами, так как их таблицы маршрутизации содержат более новую информацию.

Если узел обнаруживает нарушение маршрута, он широковещательно рассылает RERR сообщение об ошибке.

Поля сообщения RERR:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | N | Зарезервировано | DestCount |
| IP-адрес недостижимого узла-адресата | | | |
| Порядковый номер недостижимого узла-адресата | | | |
| Additional Unreachable Destination IP Addresses (if needed) | | | |
| |Additional Unreachable Destination Sequence Numbers (if needed) | | | |

Рис. 4. Формат кадра-сообщения RERR при нарушении маршрута

**Особенности протокола AODV**

При формировании маршрута всегда выбирается более новый маршрут, т.е. узлы с большими порядковыми номерами.

Узел-адресат может увеличить свой порядковый номер только на единицу.

Протокол работает на прикладном уровне, в качестве транспортного протокола используется UDP.

Узел может получить ответ, даже если он его не запрашивал и должен его обработать.

Пример работы протокола AODV:

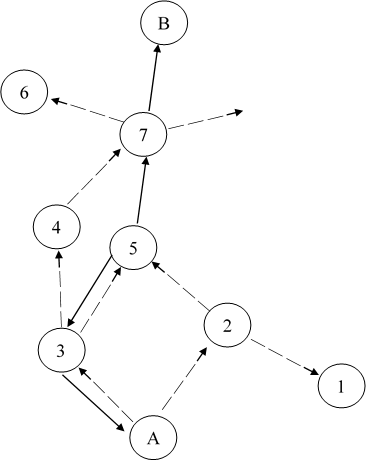


Рис.5 Формирование маршрута от А до В

Пунктирными стрелками на рис.5 изображена отправка REEQ запроса узлом А, которому необходимо узнать маршрут до узла В. У Промежуточного узла 5 есть информация о нужном маршруте, и он отправляет узлу А RREP ответ (на рисунке изображен сплошными стрелками), а также ответ узлу B для построения обратного маршрута.

**Протокол OLSR**

OLSR (Optimized Link State Routing Protocol) – проактивный протокол. Заполнение таблиц маршрутизации производится с помощью широковещательной рассылки HELLO-сообщений. В результате обработки этих сообщений каждый узел узнает об изменениях топологии сети. В сообщениях содержится собственный адрес узла, отправившего данное оповещение, а также адреса всех его доступных соседей, их адреса. Все узлы периодически с заданным интервалом широковещательно рассылают своим соседям HELLO-сообщения. Если в течении некоторого времени узел не получает сообщения от своего соседа, то связь считается разорванной. HELLO-сообщения распространяются не по всей сети, с их помощью узел может узнать информацию только о своих одно- и двухшаговых соседях.

Поля HELLO-сообщения:

|  |  |  |  |
| --- | --- | --- | --- |
| Reserved | | Htime | Willingness |
| Link Code | Reserved | Link Message Size | |
| Neighbor Interface Address | | | |
| Neighbor Interface Address | | | |
| … | | | |
| Link Code | Reserved | Link Message Size | |
| Neighbor Interface Address | | | |
| Neighbor Interface Address | | | |

Рис.7 Формат HELLO-сообщения OLSR

Htime – время, через которое будет отправлено следующее HELLO-сообщение.

Willingness – готовность узла осуществить переадресацию сообщения на другие узлы.

Link Code – в этом поле указывается информация о связи между интерфейсом отправителя и следующим в списке соседей интерфейсом. Также здесь указывается информация о состоянии соседа.

Link Message Size – размер сообщения, измеряется в байтах. Считается с начала поля “Link Code” и до следующего поля “Link Code” (если оно отсутствует, то до конца сообщения).

Neighbor Interface Address – адрес интерфейса соседнего узла.

Для распространения информации о связи между соседями по всей сети, узлы периодически с некоторым интервалом широковещательно рассылают TOPOLOGY-CONTROL сообщения.

Поля TOPOLOGY-CONTROL сообщения:

|  |  |
| --- | --- |
| ANSN | Reserved |
| Advertised Neighbor Main Address | |
| Advertised Neighbor Main Address | |
| … | |

Рис.8 Формат TOPOLOGY-CONTROL сообщения OLSR

Advertised Neighbor Sequence Number (ANSN) – порядковый номер узла. Каждый раз, когда узел обнаруживает изменение в соединение с соседями, он увеличивает свой порядковый номер. Это помогает выбирать наиболее новую информацию при построении маршрута. (аналогичный метод используется и в протоколе AODV).

Advertised Neighbor Main Address – поле содержит основной адрес узла-соседа. Отправитель сообщения помещает все известные ему адреса соседей. Если достигнут максимальный размер сообщения (разрешенный сетью), но не все адреса были помещены в сообщение, то узел формирует еще одно сообщение.

С помощью информации, полученной из HELLO и TOPOLOGY-CONTROL сообщений, каждый узел сети строит ориентированный граф, являющийся представлением соединений между узлами беспроводной сети. В полученном графе, формируются кратчайшие маршруты к каждому узлу сети. Маршрут представляет собой цепочку промежуточных узлов-ретрансляторов. В таблицу маршрутизации адрес записывается в виде адреса узла-адресата и первого ретранслятора. При необходимости передать информацию, узел-инициатор находит в своей таблице маршрутизации запись с адресом узла-адресата, и пересылает данные указанному в этой записи ретранслятору. Получив, информацию ретранслятор производит аналогичные действия. Так информация передается пока не достигнет узла-адресата или не будет отброшена при нарушении в сети.

Главной особенностью протокола OLSR является использование Multipoints Relays.

**Протокол DSR**

Dynamic Source Routing (DSR) (Динамическая маршрутизация от источника) — реактивный протокол. Схож c AODV в том, что также формирует маршрут по требованию, с помощью передачи широковещательного запроса. Отличается от AODV, он использует явную маршрутизацию от узла-источника, не полагаясь на маршруты, известные промежуточным узлам.

В DSR было внесено DSR-Flow (гибрид явной маршрутизации и маршрутизации по таблицам).

Явное задание маршрута требует накопления адресов каждого узла между источником и адресатом во время его формирования. Информация о накопленном пути пополняется узлами, получающими широковещательные запросы от источника. В результате, маршрутизируемые пакеты содержат адрес каждого узла, через который они прошли и, соответственно увеличиваются заголовки пакетов, это может привести к избыточности служебный данных для длинных путей или больших адресов (например, IPv6). Для таких случаев в DSR-Flow определена функция «flow id», которая позволяет пакетам быть отправленными в соответствии с таблицами маршрутизации.

Явное задание маршрутов помогает избежать периодическую проверку маршрутов (например, как в AODV), так как вся информация обновляется узлами пока через них проходят данные.

Маршрут сформируется только в том случае, если сообщение с запросом дошло до узла-адресата. Чтобы послать ответ на запрос, узел-адресат должен знать маршрут к источнику. Если маршрут есть в кэше, то он и используется. Иначе, маршрут будет сформирован при помощи накопленных в запросе от источника адресов узлов, но для этого необходимо чтобы соединения в сети были симметричными.

Основное отличие протокола DSR от других реактивных протоколов в том, что в нем не требуется периодическая передача каждым узлом сообщений своим соседям, чтобы оповестить о своем присутствие.

Работа этого протокола осуществляется следующим образом.

Узлу-инициатору требуется установить маршрут до узла-адресата. Он рассылается широковещательный запрос RouteRequest. Каждый узел сети, если он не является адресатом и его TTL (время жизни) не превышено ретранслирует запрос дальше по сети. Также, у запроса RouteRequest есть порядковый номер, который формируется узлами, через которые он прошел. Каждый раз, когда узел получает запрос, он проверяет его порядковый номер. Запрос отправляется только в том случае, если он не дублируется. Этот метод помогает предотвратить формирование петель в путях. Узел-адресат, получая RouteRequest запрос отправляет источнику ответ, по тому же пути, по которому запрос шел до адресата.

**Протокол HWMP**

HWMP (HybridWirelessMeshProtocol) – гибридный протокол, базируется на протоколе AODV. Однако в HWMP механизмы маршрутизации работают на канальном уровне, где доступна информация о соседних узлах и условиях беспроводной передачи, что делает алгоритмы маршрутизации более эффективными. Реактивный и проактивный механизмы могут быть использованы как одновременно в одной сети, так и отдельно. Использование данного протокола помогает сократить размер таблиц маршрутизации и объем рассылаемой информации для ее формирования. Является базовым для сетей, построенных согласно технологии IEEE 802.11s.

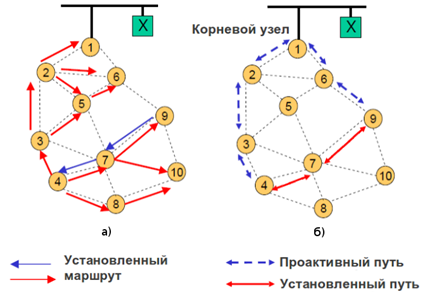


Рис. 9. Пример построения маршрута  от узла 4 до узла 9 в реактивном (а) и проактивном (б) режимах.

В реактивном режиме, маршрутные таблицы создаются в mesh-узлах непосредственно перед передачей (по запросу RREQ). Перед началом передачи, узел-отправитель формирует широковещательный запрос Path Request (RREQ) ко всем соседним узлам, которые, в свою очередь вносят необходимые изменение в поле метрики, и посылают данный запрос далее. Узел-получатель принимает пакет RREQ с уже сформировавшейся информацией о метрике всего пути, и формирует пакет подтверждения Path  Reply  (RREP),  который отсылает отправителю. Отправитель, приняв  пакет  RREP, считывает информацию о  метрики  пути и  принимает решение  о  начале  процесса  передачи  (рисунок  9а).

В отличие от реактивного режима, где таблица маршрутизации создаётся в момент необходимости передачи информации, проактивный режим предусматривает наличие корневого mesh-узла. Данный корневой узел рассылает широковещательные RREQ запросы, тем самым формируя дерево путей  с  вершиной  в  корневом  узле.  Таким образом, на момент необходимости  передачи информации, mesh-узлы  уже  имеют таблицу  маршрутизации,  что  позволяет  быстрее  установить  соединение  с  узлом  назначения (рисунок  9б).

# СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОТОКОЛОВ МАРШРУТИЗАЦИИ

Каждый протокол маршрутизации имеет свои преимущества и недостатки.

Достоинство протокола AODV состоит в том, что он не создаёт дополнительного трафика при передаче данных по уже известному маршруту. Также алгоритм векторной маршрутизации довольно прост и не требует больших объёмов памяти и машинного времени. Однако протоколу AODV требуется больше времени на установку маршрута, чем во многих других протоколах.

Протокол DSR не использует таблицы маршрутизации, поэтому сеть не нагружается периодически отправляемым сообщения об изменениях в таблице, как в большинстве протоколов. Недостаток данного протокола в отсутствии механизма, восстанавливающего разорванные соединения. В кэше маршрута может содержаться устаревшая информация, что приведет к нарушениям во время формирования маршрута. И, также как AODV, протоколу DSR требуется много времени на установку маршрута.

Протокол HWMP прост и хранит минимум информации. Ему известен только один путь до каждого из узлов. Каждое PREQ-сообщение считается пришедшим по единственно верному пути, если его порядковый номер больше предыдущего или метрика лучше, но если PREQ-сообщение, отправленное по более короткому пути, было потеряно (что для широковещательных рассылок возможно), то путь становится длиннее, чем он есть на самом деле.

Первоначальное определение OLSR не включает каких-либо положений для распознавания качества канала связи; оно просто предполагает, что связь установлена, если недавно было получено некоторое количество HELLO-сообщений. Это предполагает, что связь является бимодальной (либо установлена, либо нет), что не всегда является верным для беспроводных сетей, где линии передачи часто демонстрируют средние показатели потери пакетов данных.

Будучи проактивным протоколом, OLSR использует энергетические и сетевые ресурсы для распространения данных о возможных неиспользуемых маршрутах. Хотя это не является проблемой для проводных сетей, это делает использование OLSR непригодным для сенсорных сетей, которые большую часть времени находятся в режиме ожидания.

Будучи протоколом канального уровня, OLSR требует достаточно большой пропускной способности и мощности процессора для вычисления оптимальных путей в сети. В типичных сетях, где используется OLSR (которые редко превышают несколько сотен узлов), это не является проблемой.

Проанализировав преимущества и недостатки рассматриваемых протоколов, можно сделать выводы:

Проактивные протоколы обладают явным преимуществом во времени построения маршрута. Когда узлу необходимо отправить сообщение какому-либо другому узлу, потребуется считать маршрут из уже составленной таблицы, а реактивным протоколам необходимо разослать широковещательный запрос и дождаться подтверждения от адресата. Однако проактивным протоколам необходимо постоянно осуществлять широковещательные рассылки, на что может расходоваться значительная доля пропускной способности сети, особенно в условиях крупных сетей.

Проведем сравнительный анализ сложности рассмотренных алгоритмов маршрутизации:

Время работы протокола OLSR – O(D\*I), память – O(N2).

AODV: время работы - O(2D), память – O(2N).

DSR: время работы - O(2D), память - O(2N).

Где, D – диаметр сети, I – среднее время обновления, N – количество узлов сети.

# МОДЕЛИРОВАНИЕ AD-HOC СЕТЕЙ

Для сравнения параметров рассмотренных протоколов следует промоделировать их работу, что поможет на практике рассмотреть их особенности, преимущества и недостатки.

В настоящее время среда NS-2 является оптимальным средством для исследования эффективности в различных условиях сетей. Программа NS-2 (Network Simulator 2) предназначена для моделирования сетей, в том числе и ad-hoc сетей.

NS2 является объектно-ориентированным ПО, ядро которого реализовано на языке С++. В качестве интерпретатора используется язык скриптов OTcl (Object oriented Tool Command Language), который является объектно-ориентированным расширением языка Tcl. В NS2 реализованы практически все известные протоколы сетей связи, в том числе и протоколы беспроводных сетей. Также симулятор содержит средство анимации результатов моделирования nam (Network Animator), которое предоставляет графическое воспроизведение проведенного эксперимента: отображение топологи сети, анимация пакетов, узлов. И различные возможности анализа данных. В качестве входных данных для nam используются файлы, записанные в процессе моделирования сети.

Промоделируем в NS-2 два протокола маршрутизации: AODV и DSR.

Разберем построение сети, состоящей из двух подвижных узлов.

Перед началом симуляции необходимо определить следующие параметры: тип канала, очереди, MAC уровень, беспроводной канал, тип антенны, модель радиораспространения, протокол маршрутизации, и т.д.

Узлы движутся по площади 500х500 м2.Узлы начинают свое движение с противоположных концов. В первой половине симуляции они движутся навстречу друг другу, а во второй в противоположном направлении. Между ними устанавливается TCP соединение. Пакеты пересылаются между узлами, когда они находятся в зоне радиопокрытия. Когда они расходятся на большое расстояние, пакеты начинают теряться.

Для начала необходимо задать параметры узла, а также другие необходимые параметры: тип антенны, протокол маршрутизации, канал, очередь, MAC уровень и т.д.

set val(chan) Channel/WirelessChannel ;# channel type

set val(prop) Propagation/TwoRayGround ;# radio-propagation mode

set val(ant) Antenna/OmniAntenna ;# Antenna type

set val(ll) LL ;# Link layer type

set val(ifq) Queue/DropTail/PriQueue ;# Interface queue type

set val(ifqlen) 50 ;# max packet in ifq

set val(netif) Phy/WirelessPhy ;# network interface type

set val(mac) Mac/802\_11 ;# MAC type

set val(rp) AODV ;# ad-hoc routing protocol

set val(nn) 2 ;# number of mobilenodes

Далее нужно создать новый объект класса Simulator, в котором содержатся все методы необходимые для дальнейшего описания модели:

set ns [ new Simulator ]

Затем создадим файл трассировки и используем для него функцию trace-all:

set tracefd [open simple.tr w]

$ns\_ trace-all $tracefd

Затем создадим объект топологии и его координаты, который хранит пути передвижения подвижных узлов внутри заданной границы:

set topo [new Topography]

$topo load\_flatgrid 500 500

Далее создадим объект God (хранит информацию о состоянии среды, которая не должна быть известна каждому участнику симуляции):

create-god $val(nn)

Теперь создаем подвижные узлы. Для начала, необходимо сконфигурировать узлы:

$ns node-config -adhocRouting $val(rp) \

-llType $val(ll) \

-macType $val(mac) \

-ifqType $val(ifq) \

-ifqLen $val(ifqlen) \

-antType $val(ant) \

-propType $val(prop) \

-phyType $val(netif) \

-channelType $val(chan) \

-topoInstance $topo \

-agentTrace ON \

-routerTrace ON \

-macTrace OFF \

-movementTrace ON

Далее создаем узлы:

for {set i 0} {$i < $val(nn) } {incr i} {

set n($i) [$ns node ]

$n($i) random-motion 0 ;# disable random motion

}

Далее необходимо задать начальные координаты узлов, скорость и направление передвижения, так как случайное передвижение отключено:

$n(0) set X\_ 6.0

$n(0) set Y\_ 3 .0

$n(0) set Z\_ 0.0

$n(1) set X\_ 400.0

$n(1) set Y\_ 390.0

$n(1) set Z\_ 0.0

Узел 0 начинает движение из точки (6,3), а узел 1 из точки (400,390).

$ns at 50.0 "$n(1) setdest 25.0 20.0 15.0"

Означает что узел 1, при времени симуляции 50.0 мс начнет двигаться в точку (25,20) со скоростью 15 м/с.

$ns at 10.0 "$n(0) setdest 20.0 18.0 1.0"

$ns at 100.0 "$n(1) setdest 490.0 480.0 15.0"

Далее установим поток трафика между узлами:

set tcp [new Agent/TCP]

$tcp set class\_ 2

set sink [new Agent/TCPSink]

$ns attach-agent $n(0) $tcp

$ns attach-agent $n(1) $sink

$ns connect $tcp $sink

set ftp [new Application/FTP]

$ftp attach-agent $tcp

$ns at 10.0 "$ftp start"

Далее необходимо определить время окончания симуляции и перезагрузить узлы:

for {set i 0} {$i < $val(nn) } {incr i} {

$ns at 150.0 "$n($i) reset";

}

$ns at 150.0001 "stop"

$ns at 150.0002 "puts \"NS EXITING...\" ; $ns halt"

proc stop {} {

global ns tracefd

close $tracefd

}

Симуляции остановится на 150 секунде, и узлы перезагрузятся.

И осталось только добавить команду запуска симуляции:

puts "Starting Simulation..."

$ns\_ run

Результатом симуляции такой модели будет создание файла трассировки simple.tr. В этом файле мы можем увидеть, что на 100 секунде, узлы обмениваются маршрутной информации и устанавливается TCP соединение:

s 100.000000000 \_0\_ AGT --- 4 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0

r 100.000000000 \_0\_ RTR --- 4 tcp 40 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0

s 100.000000000 \_0\_ RTR --- 0 AODV 48 [0 0 0 0] ------- [0:255 -1:255 30 0] [0x2 1 9 [1 0] [0 20]] (REQUEST)

r 100.000900018 \_1\_ RTR --- 0 AODV 48 [0 ffffffff 0 800] ------- [0:255 -1:255 30 0] [0x2 1 9 [1 0] [0 20]] (REQUEST)

s 100.000900018 \_1\_ RTR --- 0 AODV 44 [0 0 0 0] ------- [1:255 0:255 30 0] [0x4 1 [1 2] 10.000000] (REPLY)

r 100.005134145 \_0\_ RTR --- 0 AODV 44 [13a 0 1 800] ------- [1:255 0:255 30 0] [0x4 1 [1 2] 10.000000] (REPLY)

s 100.005134145 \_0\_ RTR --- 4 tcp 60 [0 0 0 0] ------- [0:0 1:0 30 1] [0 0] 0 0

r 100.007310199 \_1\_ AGT --- 4 tcp 60 [13a 1 0 800] ------- [0:0 1:0 30 1] [0 0] 1 0

s 100.007310199 \_1\_ AGT --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0

r 100.007310199 \_1\_ RTR --- 5 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0

s 100.007310199 \_1\_ RTR --- 5 ack 60 [0 0 0 0] ------- [1:0 0:0 30 0] [0 0] 0 0

r 100.009786254 \_0\_ AGT --- 5 ack 60 [13a 0 1 800] ------- [1:0 0:0 30 0] [0 0] 1 0

На 116 секунде, когда узлы отдаляются друг от друга, соединение разрывается:

s 116.394526043 \_1\_ RTR --- 0 AODV 32 [0 0 0 0] ------- [1:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)

D 116.402377879 \_0\_ RTR CBK 2652 tcp 1060 [13a 1 0 800] ------- [0:0 1:0 30 1] [1332 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2653 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1333 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2654 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1334 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2655 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1335 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2659 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1336 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2660 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1337 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2663 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1338 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2664 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1339 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2666 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1340 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2668 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1341 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2669 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1342 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2672 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1343 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2676 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1344 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2679 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1345 0] 0 0

D 116.402377879 \_0\_ RTR CBK 2680 tcp 1060 [0 1 0 800] ------- [0:0 1:0 30 1] [1346 0] 0 0

s 116.402377879 \_0\_ RTR --- 0 AODV 32 [0 0 0 0] ------- [0:255 -1:255 1 0] [0x8 1 [1 0] 0.000000] (ERROR)

Теперь промоделируем работу протокола DSR:

Заменим строку:

set val(rp) AODV ;# ad-hoc routing protocol

из преведущего скрипта на:

set val(rp) DSR ;# ad-hoc routing protocol

Остальное остается неизменным. И в результате получим аналогичный файл трассировки. В нем мы можем увидеть, что узлы обмениваются пакетной информации на 58 секунде и устанавливают соединение на секунде 70:

Ssb 58.04653 \_0\_ dropped 0 -> 1

D 58.046525416 \_0\_ RTR TOUT 661 tcp 60 [0 0 0 0] ------- [0:0 1:0 32 0] [0 0] 0 0

s 60.123981571 \_0\_ RTR --- 2179 DSR 32 [0 0 0 0] ------- [0:255 1:255 32 0] 1 [1 13] [0 13 0 0->0] [0 0 0 0->0]

s 60.171683750 \_0\_ RTR --- 2180 DSR 32 [0 0 0 0] ------- [0:255 1:255 32 0] 1 [1 14] [0 14 0 0->0] [0 0 0 0->0]

s 70.129750882 \_0\_ RTR --- 2400 DSR 32 [0 0 0 0] ------- [0:255 1:255 32 0] 1 [1 15] [0 15 0 0->0] [0 0 0 0->0]

r 70.130523613 \_1\_ RTR --- 2400 DSR 32 [0 ffffffff 0 800] ------- [0:255 1:255 32 0] 1 [1 15] [0 15 0 0->0] [0 0 0 0->0]

s 70.132439091 \_1\_ RTR --- 2401 DSR 44 [0 0 0 0] ------- [1:255 0:255 254 0] 2 [0 15] [1 15 2 0->1] [0 0 0 0->0]

r 70.136943204 \_0\_ RTR --- 2401 DSR 44 [13a 0 1 800] ------- [1:255 0:255 254 0] 2 [0 15] [1 15 2 0->1] [0 0 0 0->0]

SFESTs 70.136943204 \_0\_ 1864 [0 -> 1] 1(1) to 1 [0 |1 ]

s 70.136943204 \_0\_ RTR --- 1864 tcp 76 [0 0 0 0] ------- [0:0 1:0 32 1] [0 0] 0 0

r 70.139249395 \_1\_ RTR --- 1864 tcp 76 [13a 1 0 800] ------- [0:0 1:0 32 1] [0 0] 1 0

r 70.139249395 \_1\_ AGT --- 1864 tcp 40 [13a 1 0 800] ------- [0:0 1:0 32 1] [0 0] 1 0

s 70.139249395 \_1\_ AGT --- 2402 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0

r 70.139249395 \_1\_ RTR --- 2402 ack 40 [0 0 0 0] ------- [1:0 0:0 32 0] [0 0] 0 0

Разрывается соединение также на 116 секунде:

SSendFailure 116.408705414 \_0\_ 9933 0 0:0 1:0 0->1 0 0 0 2 [|0 1 ]

s 116.408829137 \_0\_ RTR --- 9969 DSR 32 [0 0 0 0] ------- [0:255 1:255 32 0] 1 [1 16] [0 16 0 0->0] [0 0 0 0->0]

s 116.458690046 \_0\_ RTR --- 9971 DSR 32 [0 0 0 0] ------- [0:255 1:255 32 0] 1 [1 17] [0 17 0 0->0] [0 0 0 0->0]

s 116.464890088 \_1\_ RTR --- 9972 DSR 32 [0 0 0 0] ------- [1:255 0:255 32 0] 1 [1 2] [0 2 0 0->0] [0 0 0 0->0]

s 117.329422733 \_0\_ AGT --- 10013 tcp 1040 [0 0 0 0] ------- [0:0 1:0 32 0] [3772 0] 0 0

r 117.329422733 \_0\_ RTR --- 10013 tcp 1040 [0 0 0 0] ------- [0:0 1:0 32 0] [3772 0] 0 0