Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра «Вычислительная техника»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.И. Легалов
подпись инициалы, фамилия
« _____» _____ 20 ___ г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01– Информатика и вычислительная техника

Моделирование протокола маршрутизации на основе роевого интеллекта для Mesh-сетей

Пояснительная записка

Руководитель _____ доцент, к.т.н. Ф.А. Казаков
подпись, дата

Выпускник _____ К.Р. Васерчук
подпись, дата

Нормоконтролер _____ В.И. Иванов
подпись, дата

Красноярск 2016

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра «Вычислительная техника»

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.И. Легалов
подпись инициалы, фамилия
« ___ » _____ 20 ___ г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
в форме бакалаврской работы

Студенту Васерчук Кириллу Романовичу
Группа КИ12-09Б Направление (специальность) 09.03.01 Информатика и вычислительная техника
Тема выпускной квалификационной работы Моделирование протокола маршрутизации на основе роевого интеллекта для Mesh-сетей
Утверждена приказом по университету № _____ от _____
Руководитель ВКР Ф.А. Казаков, доцент, к.т.н., ИКИТ СФУ
Исходные данные для ВКР _____
_____
_____
Перечень разделов ВКР_____
_____
_____
_____
Перечень графического материала _____
_____
_____
_____

Руководитель ВКР _____ Ф.А. Казаков
подпись

Задание принял к исполнению _____ К.Р. Васерчук
подпись

« ___ » _____ 20__ г.

РЕФЕРАТ

Выпускная квалификационная работа по теме «Моделирование протокола маршрутизации на основе роевого интеллекта для Mesh-сетей» содержит 53 страницы текстового документа, 12 использованных источников, 34 иллюстрации.

ПРОТОКОЛ, СЕТЬ, МАРШРУТИЗАТОР, РОУТЕР, УЗЕЛ, МУРАВЕЙ, ПУТЬ, КАНАЛ, ИНТЕРФЕЙС, МЕТРИКА, ПРОПУСКАНАЯ СПОСОБНОСТЬ, ТАБЛИЦА МАРШРУТИЗАЦИИ.

Объект, подлежащий моделированию – протокол AntNet.

Цели работы:

- рассмотреть и промоделировать действующий протокол динамической маршрутизации – протокол OSPF;

- оценить достоинства и недостатки действующего протокола;

- разработать способ решения выявленных недостатков;

- произвести моделирование выявленного решения и оценить полученные результаты.

В результате рассмотрения и последующего моделирования действующего протокола динамической маршрутизации был рассмотрен алгоритм его работы, а так же была выявлена ситуация, в которой сеть работает неэффективно. Был получен график, который наглядно отражает суть выявленной проблемы.

В итоге был рассмотрен и промоделирован протокол маршрутизации, алгоритмы работы которого предусматривают решение для выявленного неэффективного случая работы. В основу алгоритма работы нового протокола лег «Муравьиный алгоритм», являющийся алгоритмом роевого интеллекта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Маршрутизацию можно описать как процесс передачи пакетов между соединенными сетями. В TCP/IP-сетях маршрутизация является частью протокола IP (Internet Protocol) и используется в сочетании с другими службами сетевых протоколов для обеспечения передачи данных между узлами, расположенными в разных сегментах более крупной TCP/IP-сети [2].

IP − это своего рода «почтовая система» стека TCP/IP, выполняющая сортировку и доставку IP-данных. Каждый входящий или исходящий пакет называется IP-датаграммой. Датаграмма IP содержит два IP-адреса, их трактовка: адрес источника (отправляющего узла) и адрес назначения (принимающего узла). В отличие от аппаратных адресов, IP-адреса в датаграмме в процессе передачи ее по TCP/IP-сети остаются постоянными [2].

Маршрутизация является основной функцией IP. Обмен IP-датаграммами и их обработка на каждом узле выполняются протоколом IP, работающим на межсетевом уровне [2].

Открытый протокол, базирующийся на алгоритме поиска наикратчайшего пути (Open Shortest Path Fisrt − OSPF) является протоколом маршрутизации, разработанным для сетей IP рабочей группой Internet Engineering Task Force (IETF), занимающейся разработкой протоколов для внутрисистемных роутеров (interior gateway protocol − IGP) [3].

OSPF имеет две основных характеристики. Первая из них − это то, что протокол является открытым, т.е. его спецификация является общественным достоянием. Второй его главной характеристикой является то, что он базируется на алгоритме SPF. Алгоритм SPF иногда называют алгоритмом «Дейкстра» по имени автора, который его разработал [3].

Являясь алгоритмом с объявлением состояния канала, OSPF отличается от RIP и IGRP, которые являются протоколами маршрутизации с вектором расстояния. Роутеры, использующие алгоритм вектора расстояния, отправляют всю или часть своей таблицы маршрутизации в сообщения о корректировке

маршрутизации, но только своим соседям. А роутеры, использующие алгоритмы маршрутизации по состоянию канала требуют отправки сообщений во все роутеры, которые находятся в пределах одной и той же иерархической области. В объявления о состоянии канала протокол OSPF включает информацию о подключенных интерфейсах, об использованных показателях и о других переменных. По мере накопления роутерами OSPF информации о состоянии канала, они используют алгоритм SPF для расчета наикратчайшего пути к каждому узлу [3].

Протокол OSPF решает большинство проблем, связанных с маршрутизацией настолько эффективно, что его работа кажется идеальной. Но существует серьезная проблема, с которой OSPF не может справиться [4].

Протокол OSPF работает на основании алгоритма Дейкстры, т.е. строя маршрут от адреса отправки пакета до адреса доставки учитывается метрика каналов, в итоге получается самый короткий путь. Однако не исключено, что на одном из каналов, включенных в маршрут следования пакета, могут передаваться большие объемы данных. Это приведет к тому, что пакет, доставленный до этого загруженного канала, будет поставлен в хвост длинной очереди на передачу сначала на одном конце канала, а потом на другом. В сложившейся ситуации маршрут, имеющий большую метрику, но на котором отсутствует какая либо нагрузка (кроме передачи служебных сообщений, в случае с протоколом OSPF), мог бы оказаться более предпочтительным, но протокол OSPF не позволяют обнаружить этот факт.

Существующие алгоритмы оптимизации потоков с учетом перегрузок каналов достаточно сложны, поэтому поиски более простых и эффективных методов решения этой задачи ведутся до сих пор.

В качестве решения проблемы передачи трафика по загруженным каналам был рассмотрен протокол AntNet, основанный на роевом интеллекте. Этот протокол маршрутизации основывается на алгоритмах оптимизации трафика сети с помощью подражания муравьиному алгоритму.

В произвольные моменты времени роутер сети, настроенной с помощью AntNet, генерирует сообщения, которые отсылаются до определенного принимающего узла, но каждое сообщение следует по своему маршруту. С помощью такого алгоритма замеряется время следования сообщения на каждом маршруте, и затем путь с меньшим временем следования сменяет прежний маршрут и становится новым маршрутом по умолчанию до данного узла назначения.

1 Обзор проблемы

1.1 Обзор протокола

Протокол OSPF (Open Shortest Path First − выбор кратчайшего пути первым) является протоколом, основанным на алгоритме состояния связей, и обладает многими особенностями, ориентированными на применение в больших гетерогенных сетях. Протокол был принят в 1991 году [5, с. 582].

OSPF разбивает процедуру построения таблицы маршрутизации на два этапа, к первому относится построение и поддержание базы данных о состоянии связей сети, ко второму − нахождение оптимальных маршрутов и генерация таблицы маршрутизации [5, с. 583].

Связи сети могут быть представлены в виде графа, в котором вершинами графа являются маршрутизаторы и подсети, а ребрами − связи между ними (рисунок 1).

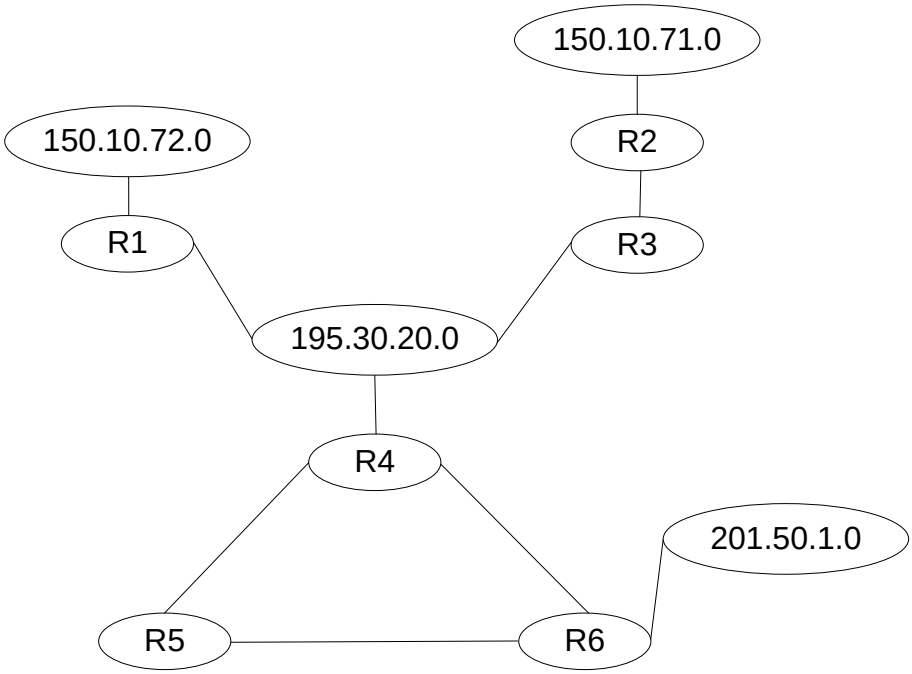

Рисунок 1 – Граф сети, построенный протоколом OSPF

Каждый маршрутизатор обменивается со своими соседями той информацией о графе сети, которой он располагает к данному моменту. Этот процесс похож на процесс распространения векторов расстояний до сетей в протоколе RIP, однако сама информация качественно иная − это информация о топологии сети. Сообщения, с помощью которых распространяется топологическая информация, называются объявлениями о состоянии связей (Link State Advertisement, LSA) сети. При транзитной передаче объявлений LSA маршрутизаторы не модифицируют информацию, как это происходит в дистанционно-векторных протоколах, в частности в RIP, а передают ее в неизменном виде. В результате все маршрутизаторы сети сохраняют в своей памяти идентичные сведения о текущей конфигурации графа связей сети [5, с. 583]. Операция синхронизации баз данных между соседними маршрутизаторами происходит и при восстановлении ранее разорванного соединения, поскольку в образовавшихся после аварии двух изолированных подсистемах базы данных развивались независимо друг от друга.

Сначала маршрутизаторы обмениваются только описаниями своих баз данных (Database Description), содержащими идентификаторы записей и номера их версий, это позволяет избежать пересылки всего содержимого базы данных.

Во время этого обмена каждый маршрутизатор формирует список записей, содержимое которых он должен запросить (то есть эти записи в его базе данных устарели либо отсутствуют), и соответственно отправляет пакеты запросов о состоянии связей (Link State Request). В ответ он получает содержимое последних версий нужных ему записей в пакетах типа "Обновление состояния связей (Link State Update)".

После синхронизации баз данных производится построение маршрутов.

Для контроля состояния связей и соседних маршрутизаторов OSPF-маршрутизаторы передают друг другу особые сообщения HELLO каждые 10 секунд. Небольшой объем этих сообщений делает возможным частое тестирование состояния соседей и связей с ними. В том случае, когда сообщения HELLO перестают поступать от какого-либо непосредственного

соседа, маршрутизатор делает вывод о том, что состояние связи изменилось с работоспособного на неработоспособное. После этой процедуры маршрутизатор вносит соответствующие коррективы в свою топологическую базу данных. Обрыв связи может быть также обнаружен и с помощью протокола канального уровня, который просигнализирует о недоступности канала. Одновременно с изменением своей топологической базы данных маршрутизатор отсылает всем непосредственным соседям объявление LSA об этом изменении, те также вносят исправления в свои базы данных и, в свою очередь, рассылают данное объявление LSA своим непосредственным соседям [5, с. 583].

Нахождение оптимальных маршрутов и генерация таблицы маршрутизации. Задача нахождения оптимального пути на графе является достаточно сложной и трудоемкой. В протоколе OSPF для ее решения используется итеративный алгоритм Дейкстры. Каждый маршрутизатор сети, действуя в соответствии с этим алгоритмом, ищет оптимальные маршруты от своих интерфейсов до всех известных ему подсетей. В каждом найденном таким образом маршруте запоминается только один шаг − до следующего маршрутизатора. Данные об этом шаге и попадают в таблицу маршрутизации [5, с. 584].

Если состояние связей в сети изменилось, и произошла корректировка графа сети, каждый маршрутизатор заново ищет оптимальные маршруты и корректирует свою таблицу маршрутизации. Аналогичный процесс происходит и в том случае, когда в сети появляется новая связь или новый сосед, объявляющий о себе с помощью своих сообщений HELLO. При работе протокола OSPF конвергенция таблиц маршрутизации к новому согласованному состоянию происходит достаточно быстро, быстрее, чем в сетях, в которых работают дистанционно-векторные протоколы. Это время состоит из времени распространения по сети объявления LSA и времени работы алгоритма Дейкстры, который обладает быстрой сходимостью. Однако

вычислительная сложность этого алгоритма предъявляет высокие требования к мощности процессора маршрутизатора [5, с. 584].

Когда состояние сети не меняется, то объявления о связях не генерируются, топологические базы данных и таблицы маршрутизации не корректируются, что экономит пропускную способность сети и вычислительные ресурсы маршрутизаторов. Однако у этого правила есть исключение: каждые 30 минут OSPF-маршрутизаторы обмениваются всеми записями базы данных топологической информации, то есть синхронизируют их для более надежной работы сети. Так как этот период достаточно большой, то данное исключение незначительно сказывается на загрузке сети [5, с. 584].

1.2 Учет метрики в протоколе OSPF

При поиске оптимальных маршрутов протокол OSPF по умолчанию использует метрику, учитывающую пропускную способность каналов связи. Кроме того, допускается применение двух других метрик, учитывающих задержки и надежность передачи пакетов каналами связи. Для каждой из метрик протокол OSPF строит отдельную таблицу маршрутизации. Выбор нужной таблицы происходит в зависимости от значений битов TOS в заголовке пришедшего IP-пакета. Если в пакете бит D (Delay − задержка) установлен в 1, то для этого пакета маршрут должен выбираться из таблицы, в которой содержатся маршруты, минимизирующие задержку. Аналогично, пакет с установленным битом T (Throughput − пропускная способность) должен маршрутизироваться по таблице, построенной с учетом пропускной способности каналов, а установленный в единицу бит R (Reliability − надежность) указывает на то, что должна использоваться таблица, для построения которой критерием оптимизации служит надежность доставки [5, с. 584].

Протокол OSPF поддерживает стандартные для многих протоколов (например, для протокола покрывающего дерева) значения расстояний для

метрики, отражающей пропускную способность: так, для сети Ethernet она равна 10, для Fast Ethernet − 1, для канала T-11 (цифровой канал технологии PDH), обладающего пропускной способностью 1,544 Мбит/с − 65, для канала с пропускной способностью 56 Кбит/с − 1785. При наличии высокоскоростных каналов, таких как Gigabit Ethernet или STM-16/64, администратору нужно задать другую шкалу скоростей, назначив единичное расстояние наиболее скоростному каналу [5, с. 584].

При выборе оптимального пути на графе с каждым ребром графа связывается метрика, которая добавляется к пути, если данное ребро в него входит. Рассмотрим рисунок 2.

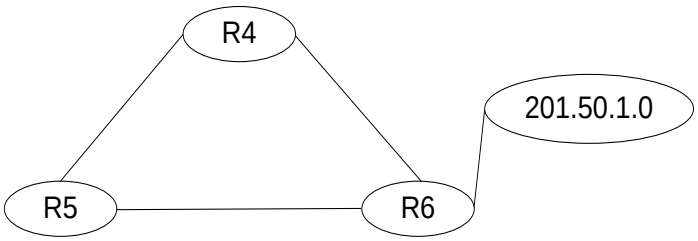

Рисунок 2 – Участок сети, изображенной на рисунке 1

Пусть маршрутизатор R4 связан с маршрутизаторами R5 и R6 каналами T-11, а маршрутизаторы R5 и R6 связаны между собой каналом 56 Кбит/с. Тогда R5 определит оптимальный маршрут до сети 201.50.1.0 как составной, проходящий сначала через R4, а затем через R6, поскольку у этого маршрута метрика будет равна 65 + 65 = 130 единиц. Непосредственный маршрут через R6 не будет оптимальным, так как его метрика равна 1785 [5, с. 585].

Протокол OSPF разрешает хранить в таблице маршрутизации несколько маршрутов к одной сети, если они обладают равными метриками. В таких случаях маршрутизатор может работать в режиме баланса загрузки маршрутов, отправляя пакеты попеременно по каждому из маршрутов [5, с. 585].

Вычислительная сложность протокола OSPF быстро растет с увеличением размера сети. Для преодоления этого недостатка в протоколе

OSPF вводится понятие области сети. Маршрутизаторы, принадлежащие некоторой области, строят граф связей только для этой области, что упрощает задачу. Между областями информация о связях не передается, а пограничные для областей маршрутизаторы обмениваются только информацией об адресах сетей, имеющихся в каждой из областей, и расстоянием от пограничного маршрутизатора до каждой сети. При передаче пакетов между областями выбирается один из пограничных маршрутизаторов области, а именно тот, у которого расстояние до нужной сети меньше [5, с. 585].

Протокол OSPF имеет много преимуществ. Например, OSPF имеет высокую скорость сходимости по сравнению с дистанционно-векторными протоколами маршрутизации. В OSPF присутствует поддержка сетевых масок переменной длины (variable length subnet mask, VLSM). Поддержка VLSM позволяет гибко управлять пространством IP-адресов, не используя жесткие рамки классовой адресации. Использование этого метода позволяет экономно использовать ограниченный ресурс IP-адресов, поскольку возможно применение различных масок подсетей к различным подсетям.

Среди прочих преимуществ OSPF можно выделить оптимальное использование пропускной способности с построением дерева кратчайших путей. Но основным преимуществом OSPF является его эффективность: даже в очень больших сетях OSPF мало загружает сеть своим трафиком. Однако OSPF является сложным: он требует грамотного планирования и более сложен в настройке и администрировании.

1.3 Недостаток протокола OSPF

Протокол OSPF, используя алгоритм состояния связей, эффективно устраняет недостатки многих протоколов динамической маршрутизации. Но имеется серьезная проблема, с которой OSPF не может справиться.

На иллюстрации ниже изображен фрагмент сети (рисунок 3).

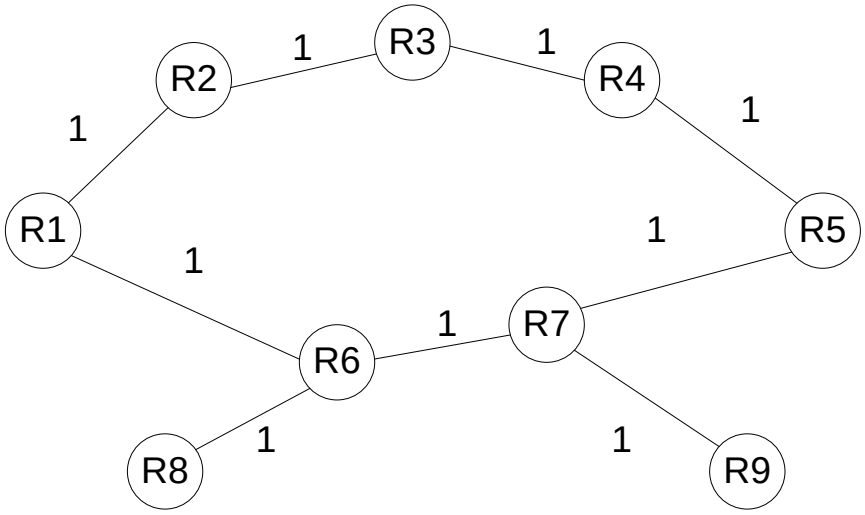

Рисунок 3 – Фрагмент сети

Кружками на рисунке обозначены маршрутизаторы (R1-R9), а линиями обозначены линии связи (каналы). Рядом с каждой линией связи указана ее метрика. Все каналы связи маршрутизаторов являются идентичными и имеют одинаковую пропускную способность. Конкретные значения метрики выбраны одинаковыми с той целью, чтобы проблему, заключенную в протоколе OSPF, было проще понять. Формируя маршрут от роутера R1 к роутеру R5 OSPF выберет следующий путь: R1-R6-R7-R5. Маршрут через узлы R2, R3 и R4 будет отвергнут как более длинный. В определенный момент времени с узла R8 на узел R9 начинают пересылаться большие объемы данных. Это приведет к тому, что пакет, переданный с R1 и адресованный узлу R5, будет поставлен в хвост длинной очереди на передачу сначала на узле R6, затем на R7. В сложившейся ситуации маршрут R1-R2-R3-R4-R5 мог бы оказаться более предпочтительным, но используемый в настоящее время протокол OSPF версии 2 не позволяет обнаружить этот факт.

При исправности канала R1-R6 переключение на канал R1-R2 может быть исключено еще и потому что протокол OSPF работает так, что он использование альтернативных маршрутов в нем исключено, так как он не вносит их в таблицы маршрутов (т.е. маршрутизатор считает, что их просто не существует). Использование альтернативных маршрутов в OSPF

предусмотрено только для случаев, когда от одного маршрутизатора до другого существует более одного маршрута с одинаковой метрикой. В этом случае пакеты будут попеременно слаться по данным путям, реализуя тем самым балансировку нагрузки, которая происходит в протоколе OSPF таким образом.

И это был рассмотрен случай, когда сеть состоит менее чем из 10-ти маршрутизаторов. В случае же рассмотрения большой сети падение скорости может быть более заметным, так как на пути следования пакета может встретиться не один канал с низкой пропускной способностью.

В заголовке IP-пакета имеется поле, которое называется «Тип обслуживания» (Type of Service, TOS). Данный байт отвечает за набор критериев, определяющих тип обслуживания пакетов. В данном поле содержится 3 бита DTR (Delay, Throughput, Reliability), которые определяют требования ко времени задержки передачи сегментов, требование к надежности передачи, и требование к пропускной способности маршрута. Но даже особое выставление битов DTR в поле «Тип обслуживания» пакета IPv4 не может гарантировать, что скорость передачи пакетов останется по-прежнему высокой. В лучшем случае для передаваемого пакета будет выбран другой маршрут, без нагрузки на каналах, либо с минимальной нагрузкой, но в худшем случае, выбранный маршрут передачи останется по-прежнему неоптимальным.

Существующие алгоритмы оптимизации потоков с учетом перегрузок каналов достаточно сложны, поэтому поиски более простых и эффективных методов решения этой задачи ведутся до сих пор. Одному из методов и посвящена данная работа [4].

2 Модификация для решения проблемы

2.1 Решение при помощи муравьев

Для решения задачи по учету пропускной способности интерфейсов маршрутизаторов следует обратиться к исследованиям биологов, а именно к экспериментам по наблюдению за поведением колонии муравьев.

Муравьиные алгоритмы или Ant Colony Optimization (ACO) получили свое распространение в последние два десятилетия. ACO активно используются исследователями для решения оптимизационных задач (задача о коммивояжере, раскраска графов, нахождение кратчайших путей и др.). Данный тип алгоритмов основывается на поведении социальных насекомых в природе. К социальным насекомым относятся все виды муравьев, термитов, некоторые виды пчел и ос. Поведение всех перечисленных насекомых основано на роевом интеллекте (Swarm Intelligence). В поведении социальных насекомых различают два вида взаимодействия между особями:

- прямое взаимодействие (обмен пищей, визуальный контакт, химический контакт и др.);

- косвенное взаимодействие или стигмержи (stigmergy) − две особи взаимодействуют косвенно, когда одна из особей модифицирует окружающую среду, а другая, со временем, реагирует на это изменение.

В природе косвенное взаимодействие осуществляется через феромон (pheromone) − специальный, довольно стойкий секрет, оставляемый как след при перемещении насекомого. Чем больше концентрация феромона на тропе, тем больше муравьев будет по ней двигаться. Со временем феромон испаряется, что позволяет муравьям адаптировать свое поведение под изменения внешней среды [7].

Рассмотрим случай, когда на пути от муравейника к источнику пищи встречается развилка и муравей должен самостоятельно выбрать один из двух путей (рисунок 4).

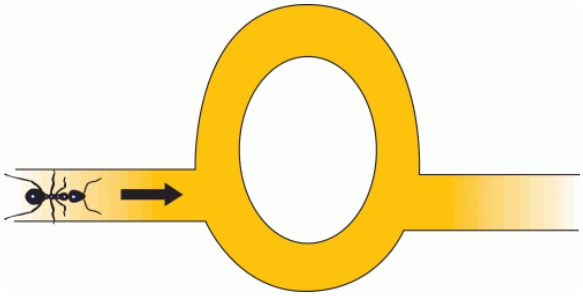

Рисунок 4 – Муравей встречает на пути следования развилку [4]

Один из путей намного длиннее, но муравей, находящийся на развилке, не имеет об этом никакого представления. Через некоторое время оказывается, что так называемый «муравьиный трафик» направлен именно по короткому пути. Как это происходит, будет рассмотрено далее [4].

В худшем случае первый муравей, дойдя до развилки, выбирает неоптимальный, длинный путь. Муравей на пути своего следования помечает пройденную дорогу, выделяя вещество под названием феромон. Феромоны, помогают муравью найти обратный путь, выполняя, таким образом, роль зарубок, которые охотник оставляет на деревьях в лесу. Следующий муравей, дойдя до развилки дорог на пути домой, чувствует запах феромонов, оставленных первым муравьем. Чаще всего муравей выбирает путь, уже пройденный его сородичем. Но в муравьиной колонии есть особи, которые действуют противоположно большинству представителей колонии, и выбирают путь, на котором запах феромона является слабее или вообще отсутствует. Не исключено, что такой первопроходец решится пойти по неизведанному пути, который является непредпочтительным для большинства, и этот муравей повернет туда, где ранее еще никто не ходил. У следующего муравья выбор будет сложнее: следовать по длинной дороге, по которой проследовали уже многие, либо выбрать короткий путь, по которому прошел только один муравей и запах феромона является пока еще относительно слабым. Даже если новый муравей снова выберет традиционный маршрут – это не будет являться

проблемой. Вероятность использования альтернативного пути все равно постепенно будет увеличиваться, так как феромоны постепенно испаряются, а муравьи, которые следует по короткому пути, успевают обновлять метки на коротком пути.

Каждый муравей, который выбирает короткий путь на развилке, постепенно повышает шансы, что поэтому короткому пути пойдут и его последователи. Таким образом, происходит срабатывание механизма положительной обратной связи, что в итоге приводит к тому, что основной поток муравьев постепенно переключается с длинного пути на короткий. Традиции муравьиной колонии, таким образом, делают благое дело, сокращая тем самым расстояние от дома до источника пищи, а отдельные «исследователи», не признающие догм колонии, выбирающие дорогу, по которой никто не ходит, либо добираются до источника пищи дольше остальных муравьев, либо вовсе остаются ни с чем.

Эксперименты и наблюдения за природой живых организмов часто приводят к необычным и интересным техническим решениям. Так произошло и с такой областью исследований как «маршрутизация»: именно муравьиный алгоритм работы подсказал ученым Гианни Ди Каро (Gianni Di Caro) и Марко Дориго (Marco Dorigo) идею создания системы маршрутизации, основанную на базе мобильных агентов. Данная система была описана в 1998 году в совместной работе Ди Каро и Доринго и получила название AntNet.

Создание системы мобильных агентов AntNet было не первой попыткой использования муравьиного алгоритма к задачам маршрутизации. Шондерверд (Schoonderwoerd) и его коллеги в 1996-1997 годах начали рассматривать маршрутизацию как область исследований, для которой можно применить идею муравьиного алгоритма. Ими был придуман подход к управлению сетью при помощи муравьев (ABC − ant-based control). Этот подход используется в телефонных сетях при маршрутизации данных, и данный подход во многом отличается от алгоритма работы протокола AntNet. Кроме того, ученые Субраманиан (Subramanian), Чен (Chen) и Друшель (Druschel) в 1997 году

предложили использовать свой алгоритм, в основу которого легло поведение муравьиной колонии в природе. Этот алгоритм работы был применен к сетям с коммутацией пакетов и представлял собой расширение ABC-алгоритма, идея которого принадлежала Шондерверду.

2.2 Идея мобильных агентов

Под программным агентом в компьютерных науках понимается программа, которая вступает в состояние посредничества с пользователем или другой целевой программой. Слово «агент» происходит от латинского слова agere (делать) и означает соглашение выполнять какие-то действия от имени кого-либо. Такие «действия от имени» означают право решать, какие действия (если они требуются) являются целесообразными для применения в системе. Основная идея состоит в том, что агенты запускаются не непосредственно для решения определенной задачи, а активизируются самостоятельно в какие-то моменты времени. В свою очередь понятие «мобильные агенты» подразумевает агентов, способных переместить своё выполнение на другие процессоры для последующего выполнения там.

Главная особенность, отличающая мобильные агенты от других типов программ, – это их способность перемещаться по сети, причем перемещение происходит по их собственной инициативе. По причине этого следует, что мобильными агентами нельзя считать ни сценарии в составе HTML-страниц, ни апплеты, и даже сериализуемые объекты, допускающие копирование по сети не могут называться мобильными агентами. На определенном этапе своего выполнения агент принимает решение, что для него будет целесообразным сменить «место жительства». Для этого мобильный агент инициирует свое перемещение, и на узле назначения агент продолжает свою работу в том же состоянии, в котором он находился перед началом перемещения.

Для создания мобильных агентов подходят язык программирования, обеспечивающие реальную переносимость кода. Предпринималось множество

попыток по реализации мобильных агентов на интерпретируемых языках, в частности на языке Tcl, но чаще всего попытки реализовать мобильный агент делаются с помощью языка Java. Но каким бы удачным не был выбор языка реализации агента, какие бы хорошие решения не предпринимал разработчик, мобильный агент не будет работать на том узле, где не осуществляется его поддержка. Иначе бы вся сеть очень скоро была бы наводнена вирусами, созданными по такому принципу. Перемещение мобильного агента возможно лишь на то устройство, где есть вся необходимая инфраструктура для поддержки агентов данного конкретного типа, которая иногда может быть реализована через достаточно сложные программы.

2.3 Протокол AntNet

AntNet − это адаптивный, распределенный, базирующийся на агентах алгоритм маршрутизации. Элементы AntNet, включают непременный атрибут любого маршрутизатора – таблицу маршрутизации.

Работа предполагаемого роутера могла бы быть представлена следующим образом. На начальном этапе работы могли бы устанавливаться исходные маршруты, которые необязательно должны быть оптимальными. Эти маршруты нужны лишь с той целью, чтобы начать работу в сети. Затем маршрутизатор приступает к началу передачи полезных данных – IP-пакетов. После того, как счетчик переданных пакетов становится кратным 1000 единиц, маршрутизатор создает сообщение «ant-ping», так называемого «муравья», который используется в протоколе для определения RTT маршрута (round-trip time). RTT является ключевым параметром в определении оптимального маршрута. После рассылки сообщений ant-ping маршрутизатор продолжает передачу полезных IP-пакетов по исходному маршруту, который был установлен ранее в таблице маршрутизации.

Муравей ant-ping рассылается по всем интерфейсам маршрутизатора, то есть с помощью широковещательной передачи. Пакет будет следовать до узла

назначения не всегда оптимальными путями, но узлы сети, которые не ожидают прихода данного сообщения ant-ping, будут просто отбрасывать такой пакет. К узлам, отбрасывающим определенный пакет ant-ping, относятся узлы, которые уже получали данный пакет. Это будет возможно в виду того, что в процессе путешествия в ant-ping формируется особый стек, где фиксируется информация о посещенных узлах.

Дойдя до конечного узла, муравей проделает весь путь в обратном порядке по пройденному ранее маршруту. Так как ant-ping хранит в своем стеке пройденные узлы, то узлы, на которых он не был, будут просто отбрасывать такие пакеты.

Исходный маршрутизатор, дождавшись муравья, получает необходимую информацию о времени прохождения маршрута. Имеющиеся данные можно интерпретировать как реальную метрику пути, учитывающую текущую конфигурацию и загрузку сети. Сведения, поступившие от различных муравьев, позволяют достаточно точно сравнить стоимость прохождения пакета по разным маршрутам и, соответственно, оптимизировать таблицу маршрутизации.

Граф состояний протокола маршрутизации AntNet может быть представлен следующим образом (рисунок 5).

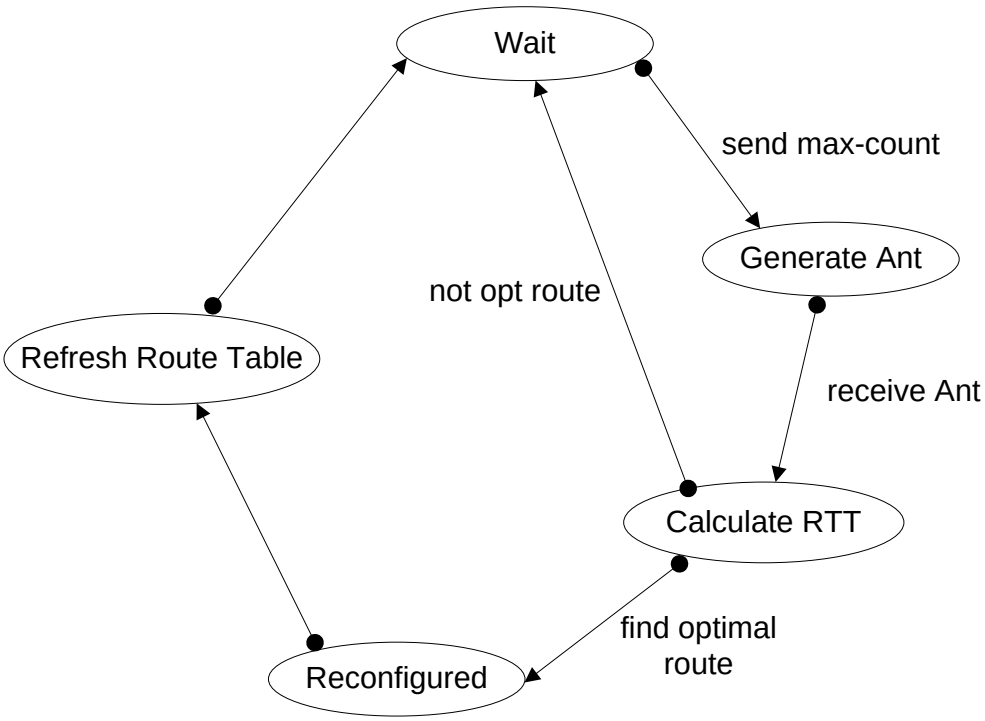

Рисунок 5 – Граф состояний протокола AntNet

Работа протокола начинается в состоянии «Wait», то есть в состоянии ожидания. В состоянии «Wait» протокол может находиться бесконечно долго.

После некоторых внешних воздействий протокол переходит в состояние «Generate Ant», в котором происходит рассылка муравьев по всем интерфейсам узла. Муравьи рассылаются по сети, замеряя параметр RTT. После сбора всех муравьев протокол переходит в состояние «Calculate RTT», в котором происходит подсчет RTT муравьев со всех маршрутов. В случае если оптимального маршрута не было выявлено, протокол возвращается в состояние «Wait». Если же оптимальный маршрут был найден, то протокол переходит в состояние «Reconfigured». В этом состоянии происходит изменение существующей информации о маршрутах, выбираются наиболее оптимальные маршруты следования. После протокол переходит в состояние «Refresh Route Table» и обновляет таблицу маршрутизации, после чего маршрут по умолчанию до целевого узла меняется, пакеты начинают передаваться по оптимальному маршруту, а протокол возвращается в начальное состояние «Wait».

Принцип определения маршрутов, применяемый в системе AntNet, обеспечивает движение пакетов по различным направлениям с учетом реальной загрузки линий. Данный алгоритм функционирования протокола AntNet порождает определенные трудности, так как следует учитывать резкое увеличение накладных расходов, связанных со сбором информации о сети. Это связано с тем, что приходится передавать исполняемый код агента. Да и объем данных о маршрутах увеличивается – ведь агент движется к узлу назначения не по кратчайшему пути. Таким образом, AntNet создает нагрузку на сеть в 20–30, а в некоторых случаях – и в 40 раз большую, чем маршрутизаторы, поддерживающие протокол OSPF. Но несмотря на многократное повышение накладных расходов, данные, передаваемые AntNet, занимают приблизительно 0,3% пропускной способности линий связи, в особенности если учесть положительный эффект от использования системы [4].

Что касается протокола по состоянию канала связи – OSPF, то написание для него мобильного агента без серьезных изменений его алгоритмов работы является очень сложной задачей. Все дело в том, что в своих таблицах, содержащих информацию о топологии сети, OSPF не содержит информации об альтернативных маршрутах, что делает отправку мобильных агентов с целью исследования сети и сбора служебной информации задачей труднореализуемой.

3 Моделирование

3.1 Обзор среды моделирования OMNeT++

OMNeT++ − это расширяемый, модульный фреймворк, на основе компонентов библиотеки C++, используемый для построения моделей сети, представляет собой симулятор дискретных событий. В системе OMNeT++ заложена детальная реализация протоколов, начиная от сетевого уровня, возможность написания и подключения собственных модулей, развитый графический интерфейс [8].

Изменение состояния моделируемой системы происходит в дискретные моменты времени по списку будущих событий (future eventlist), отсортированных по времени. Событием может быть: начало передачи пакета, тайм-аут и т. п. События происходят на основе выполнения простых модулей (simple module). У такого модуля есть функции инициализации, обработки сообщения, действия и завершения работы [8].

Система INET Framework – это комплект модулей с открытым исходным кодом, позволяющих реалистично моделировать узлы и протоколы проводных и беспроводных сетей. Он включает модели различных протоколов Интернета: IP, IPv6, TCP, UDP, 802.11, Ethernet, PPP, MPLS с LDP и RSVP-TE signalling, OSPF версии 2 и ряд других. В комплект также входят различные реалистичные примеры использования этих протоколов [8].

Наивысший уровень абстракции в моделировании IP в INET Framework – это сеть, состоящая из IP-узлов. Узел может быть маршрутизатором или хостом. IP-узел отвечает компьютерному представлению стека протоколов Интернета. Модули, из которых он состоит, организованы так, как происходит обработка IP-дейтаграммы в операционных системах. Обязательным является модуль, отвечающий за сетевой уровень (реализующий обработку IP) и модуль “сетевой интерфейс”. Дополнительно подключаются модули, реализующие протоколы транспортного уровня [8].

Помимо модуля INET, для OMNeT++ существует еще несколько модулей, которые не будут использованы в данной работе, так как в этом нет необходимости. Наименование этих модулей:

- MiXiN − является фреймворком OMNeT++. MiXiM создан для моделирования мобильных и беспроводных сетей (беспроводных сенсорных сетей, узкоспециализированных сетей, автомобильных сетей и т.д.). MiXiM концентрируется на нижних уровнях стека протоколов, а также предлагает детальные модели распространения радиоволн, оценки помех, встроенный радиопередатчик энергопотреблением и беспроводных протоколов MAC [9];

- Castalia − является симулятором работы беспроводных сенсорных сетей (Wireless Sensor Network, WSN), сетей BAN (Body Area Networks) и в целом сетей маломощных встраиваемых устройств [9].

Сборка проектов, их пересборка, а так же запуск моделирования и удаление проектов возможно с помощью командной строки OMNeT++, которая расположена в корне программы и называется «mingwenv.cmd» (рисунок 6).

Рисунок 6 – Командная строка OMNeT++

На иллюстрации, представленной далее, изображено окно визуализации OMNeT++ (рисунок 7).

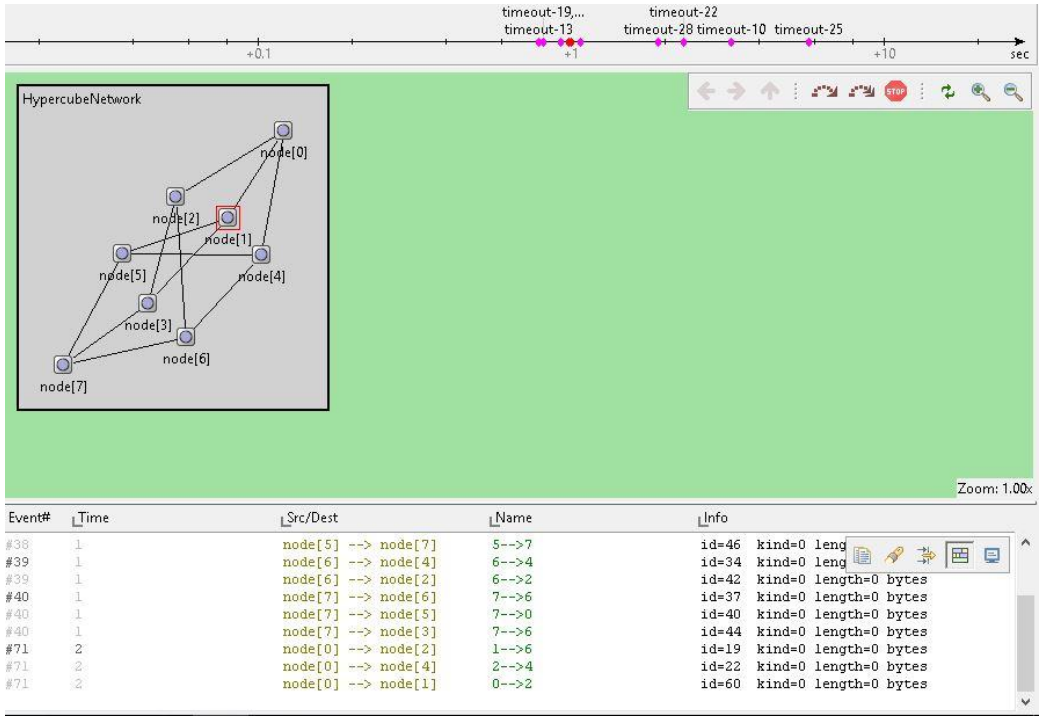

Рисунок 7 – Окно визуализации OMNeT++, часть 1

В верхнем окне, отображается компьютерная сеть для проведения моделирования. Она представляет собой набор клиентских станций, сервер и маршрутизатор, соединенных каналами связи. Каждый узел может нести различную функциональность в зависимости от параметров или набора внутренних модулей. Внутренние модули отвечают за работу протоколов и приложений на различных уровнях модели OSI. Узлы сети соединяются между собой каналами связи, параметры которых можно изменять. На рисунке есть и другое окно, нижнее, в нем происходит запись лога всех происходящих процессов.

Помимо описанных окон есть окна, на которых отображается внутренняя структура сервера и маршрутизатора (протоколы сетевого, транспортного уровней модели OSI, таблица маршрутизации и т.д.) (рисунок 8).

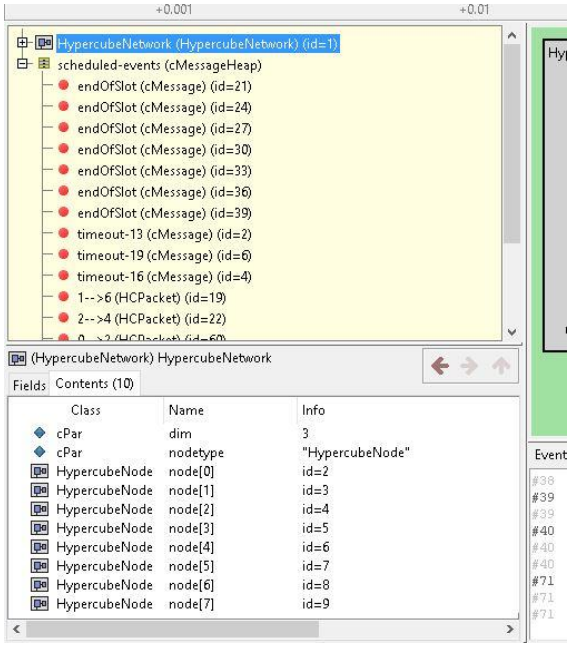

Рисунок 8 – Окно визуализации OMNeT++, часть 2

Все вышеупомянутые свойства доступны в любое время протекания процесса. Тем самым, пользователь может отследить передвижение интересующего пакета по сети [8].

Далее следует рассмотреть интегрированную среду разработки OMNeT++ (IDE). На рисунке 9 изображено окно IDE.

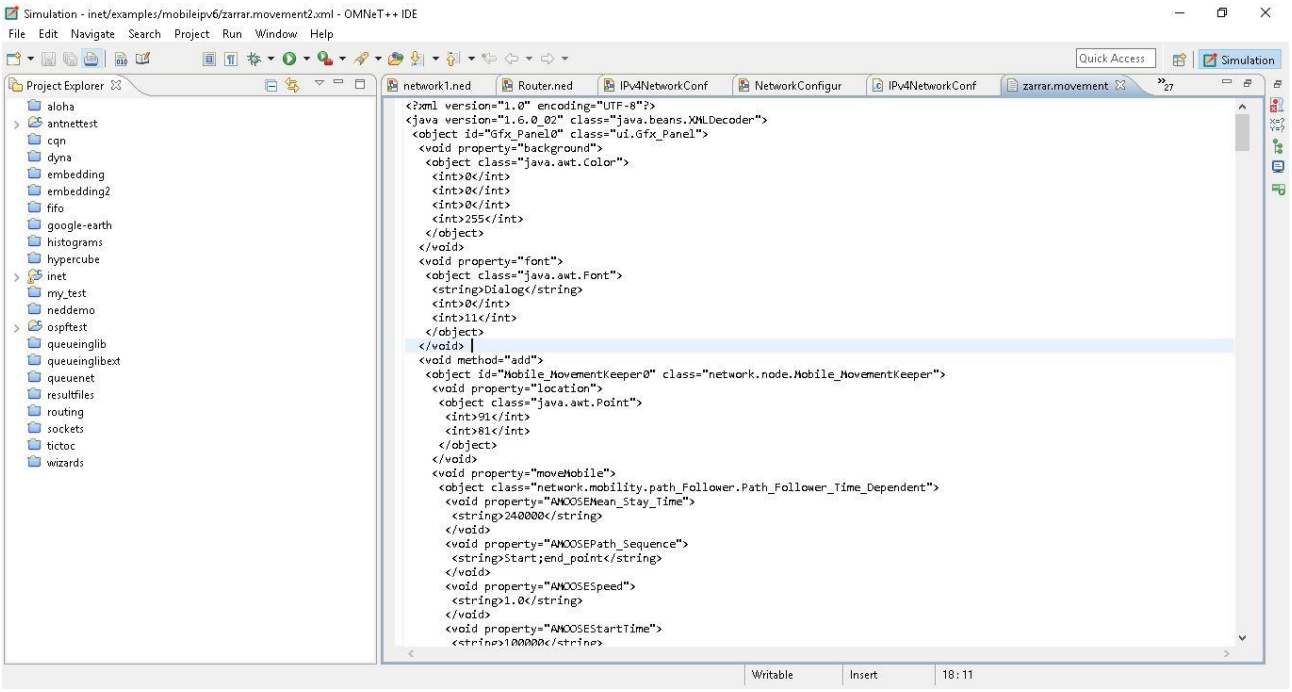

Рисунок 9 – Окно OMNeT++ IDE

В окне IDE слева можно увидеть поле, в котором располагается список всех проектов, которые могут быть рассмотрены и промоделированы. Справа располагается рабочее поле программы, где можно изучать содержимое файлов проектов, изменять их.

3.2 Моделирование протокола OSPF

3.2.1 Описание исходной модели

В качестве модели сети была выбрана и построена сеть, представленная на рисунке 10.

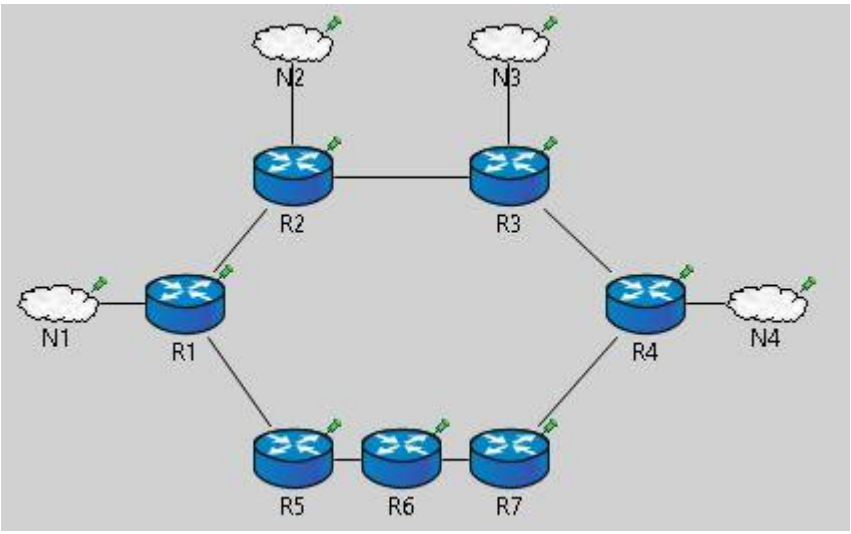

Рисунок 10 – Модель сети, построенная в OMNeT++

Для ее создания и функционирования потребовалось в папке проекта создать соответствующие файлы:

- network.ned;

- omnetpp.ini;

- ASConfig.xml.

Файлы с расширением .ned являются файлами, которые отвечают за создание всех компонентов сети. В данном случае, на рисунке выше видно, что в сеть входят 7 ед. маршрутизаторов.

Рассмотрим создание роутера R1 (рисунок 11).

```
network OspfNet
{
    submodules:
        R1: OSPFRouter {
            parameters:
                @display("p=70,198");
            gates:
                ethg[3];
        }
```

Рисунок 11 – Создание маршрутизатора R1

Создание всех видимых компонентов сети происходит в секции submodules, которая принадлежит секции network с именем OspfNet. Создавая маршрутизатор R1, ему присваивается готовый класс OSPFRouter, который уже включен в OMNeT++, а именно в модуль INET, где прописана вся структура роутера, принадлежащего к данному классу. Затем, в секции parameters указывается значение для @display, которая отвечает за точку размещения роутера на картинке сети. После секции parameters следует секция gates, где необходимо обозначить количество интерфейсов маршрутизатора.

Создание остальных маршрутизаторов имеет такой же вид, с той лишь разницей, что у каждого маршрутизатора свое значение для @display и свое количество задействованных интерфейсов.

Создание сетевых хабов N1-N4 имеют несколько другую структуру (рисунок 12).

```
N1: OspfLan {
    parameters:
        h = 2;
        @display("p=123,83");
}
```

Рисунок 12 – Создание сетевого хаба N2

Как и в случае с маршрутизатором, необходимо присвоить значение для @display, но помимо этого, для данного компонента сети в секции parameters необходимо обозначить количество хостов через переменную h. Класс OspfLan, к которому принадлежат сетевые хабы, является классом, который был создан в ходе выполнения работы, и в список готовых классов OMNeT++ он не входит (рисунок 13).

```
module OspfLan
{
    parameters:
        int h; // number of hosts on the hub
        @display("i=old/cloud");
    gates:
        inout ethg[];
    submodules:
        hub: EtherHub {
            @display("is=s");
        }
        host[h]: StandardHost {
            @display("is=s");
        }
    connections:
        for i=0..sizeof(ethg)-1 {
            hub.ethg++ <--> ethg[i];
        }
        for i=0..h-1 {
            hub.ethg++ <--> C <--> host[i].ethg++;
        }
}
```

Рисунок 13 – Создание класса OspfLan

Создание класса OspfLan начинается с создания секции module, которой присваивается соответствующее имя. В секции parameters создаем переменную h, отвечающую за количество хостов в хабе, и присваиваем значение для @display, которое отвечает за изображение данного компонента в сети. В секции submodules создаем сетевой коммутатор и хостов, которым присваиваются готовые классы EtherHub и StandardHost соответственно, а затем присваиваются значения для @display. В секции connections с помощью циклов с предусловием производится автоматическая трассировка соединений внутри хаба.

Для создания каналов передачи данных между маршрутизаторами нужно прописать необходимые данные в секции connections, которая принадлежит секции network OspfNet (рисунок 14).

```
connections:
    R1.ethg[0] <--> C <--> R2.ethg[0];
    R2.ethg[1] <--> C <--> R3.ethg[0];
    R3.ethg[1] <--> C <--> R4.ethg[0];

    R1.ethg[1] <--> C <--> R5.ethg[0];
    R5.ethg[1] <--> C <--> R6.ethg[0];
    R6.ethg[1] <--> C <--> R7.ethg[0];
    R7.ethg[1] <--> C <--> R4.ethg[1];

    N2.ethg++ <--> C <--> R2.ethg[2];
    N3.ethg++ <--> C <--> R3.ethg[2];

    N1.ethg++ <--> C <--> R1.ethg[2];
    N4.ethg++ <--> C <--> R4.ethg[2];
```

Рисунок 14 – Секция создания каналов связи

Между двумя компонентами устанавливается соответствие. При установлении соответствия необходимо указать тип канала, который соединяет два узла, в данном случае «C». C – это созданный класс для канала связи, который представлен далее (рисунок 15).

```
channel C extends ThruputMeteringChannel
{
    delay = 0.1us;
    datarate = 100Mbps;
    thruputDisplayFormat = "#N";
}
```

Рисунок 15 – Созданный класс для канала связи

Создавая канал C через extends указывается класс наследования ThruputMeteringChannel, у которого будут унаследованы необходимые переменные, и присвоены выбранные значения. Переменная delay отвечает за задержку, datarate отражает максимальную пропускную способность канала связи, а thruputDisplayFormat отвечает за отображение названия канала в процессе симуляции.

Для того, чтобы все вышеупомянутые готовые классы можно было использовать, их необходимо подключить в файле (рисунок 16)

```
import inet.common.misc.ThruputMeteringChannel;
import inet.linklayer.ethernet.EtherHub;
import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.node.inet.StandardHost;
import inet.node.ospfv2.OSPFRouter;
```

Рисунок 16 – Подключение файлов

На рисунке 17 изображена первая часть кода файла omnetpp.ini.

```
[General]
network = OspfNet
tkenv-plugin-path = ../../../etc/plugins
sim-time-limit = 75s
**.ospf.ospfConfig = xmldoc("ASConfig.xml")
**.eth[*].numOutputHooks = 1
**.eth[*].outputHook[0].typename = "ThruputMeter"
```

Рисунок 17 – Часть кода файла omnetpp.ini

В файле omnetpp.ini было указано следующее. В качестве названия сети для симуляции было задано название OspfNet. Затем был указан путь для плагинов, необходимых для симуляции. В качестве времени симуляции было выбрано время, которое равно 75 сек. Такой временной промежуток был выбран в виду того, что на нем можно увидеть цикл работы протокола OSPF и отследить решаемую с помощью AntNet проблему.

Затем в omnetpp.ini был подключен файл ASConfig.xml, необходимый для начальной настройки сети протоколом OSPF, а так же для его последующей работы. В следующей строке инициализируется параметр outputHook, который предусмотрен в OMNeT++ с целью проведения специальных замеров. Для сети был создан один такой параметр, который был настроен на расчет пропускной способности каналов между маршрутизаторами.

Далее следует изображение, отражающее процесс инициализации TCP соединения (рисунок 18).

```
**.N1.host[0].numTcpApps = 1
**.N1.host[0].tcpApp[0].typename = "TCPSessionApp"
**.N1.host[0].tcpApp[0].sendBytes = 2 MiB
**.N1.host[0].tcpApp[0].active = true
**.N1.host[0].tcpApp[0].connectAddress = "N4.host[0]"
**.N1.host[0].tcpApp[0].connectPort = 10021
**.N1.host[0].tcpApp[0].tOpen = 60s
**.N1.host[0].tcpApp[0].tSend = 60.05s
**.N1.host[0].tcpApp[0].tClose = 60.5s

**.N4.host[0].numTcpApps = 1
**.N4.host[0].tcpApp[0].typename = "TCPEchoApp"
**.N4.host[0].tcpApp[0].localPort = 10021
```

Рисунок 18 – Создание TCP соединения

Создается одно TCP приложение. В качестве типа приложения TCP было выбрано простое приложение, которое устанавливает TCP соединение согласно своему алгоритму, с передачей всех флагов, а затем происходит передача сегментов и закрытие TCP соединения. Такой тип приложения называется «TCPSessionApp». Объем передаваемых данных составляет 2 Мбита. Порт установки соединения – 10021. tOpen – время открытия соединения равно 60 сек. tSend и tClose – это время начала передачи и время закрытия соединения соответственно. Следующие (последние) три строчки кода отображают создание одного приложения типа «TCPEchoApp», которое прослушивает порт 10021, и при любом полученном на этом порту сообщении отправляет его «зеркальную» копию назад источнику. Копия является зеркальной в том плане, что назад она проследует по тому же маршруту, по которому и была доставлена до целевого роутера.

Данные типы TCP приложений создаются для случаев передачи пакетов между сетевыми хабами N1 и N4, а так же между хабами N2 и N3.

Файл ASConfig.xml нужен для указания параметров, необходимых для начальной настройки и последующей работы OSPF протокола в процессе моделирования в среде OMNeT++, так как в нем содержатся необходимые для этого параметры работы. Данные XML-файла считываются, парсятся и отправляются на дальнейшую обработку с целью построения топологии OSPF-

сети. Например, в XML-файле ASConfig указываются отношения соседства, параметры для выбора DR и BDR роутера, создаются зоны (рисунок 19, 20).

```
<?xml version="1.0"?>
<OSPFASConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="OSPF.xsd">

  <!-- Areas -->
  <Area id="0.0.0.0">
    <AddressRange address="R2>N2" mask="R2>N2" status="Advertise" />
    <AddressRange address="R3>N3" mask="R3>N3" status="Advertise" />

    <AddressRange address="R1>N1" mask="R1>N1" status="Advertise" />
    <AddressRange address="R4>N4" mask="R4>N4" status="Advertise" />

    <AddressRange address="R1>R2" mask="R1>R2" status="Advertise" />
    <AddressRange address="R2>R1" mask="R2>R1" status="Advertise" />
```

Рисунок 19 – Файл ASonfig.xml, часть 1

```
    <AddressRange address="R6>R7" mask="R6>R7" status="Advertise" />
    <AddressRange address="R7>R6" mask="R7>R6" status="Advertise" />

    <AddressRange address="R7>R4" mask="R7>R4" status="Advertise" />
    <AddressRange address="R4>R7" mask="R4>R7" status="Advertise" />
  </Area>

  <Router name="R1" RFC1583Compatible="true">
    <BroadcastInterface toward="N1" areaID="0.0.0.0" interfaceOutputCost="1" routerPriority="1" />
    <PointToPointInterface toward="R2" areaID="0.0.0.0" interfaceOutputCost="1" />
    <PointToPointInterface toward="R5" areaID="0.0.0.0" interfaceOutputCost="1" />
  </Router>

  <Router name="R2" RFC1583Compatible="true">
    <BroadcastInterface toward="N2" areaID="0.0.0.0" interfaceOutputCost="1" routerPriority="1" />
    <PointToPointInterface toward="R1" areaID="0.0.0.0" interfaceOutputCost="1" />
    <PointToPointInterface toward="R3" areaID="0.0.0.0" interfaceOutputCost="1" />
  </Router>
```

Рисунок 20 – Файл ASConfig.xml, часть 2

После написания и подключения всех необходимых для моделирования файлов дизайн NED-файла выглядит следующим образом (рисунок 21).

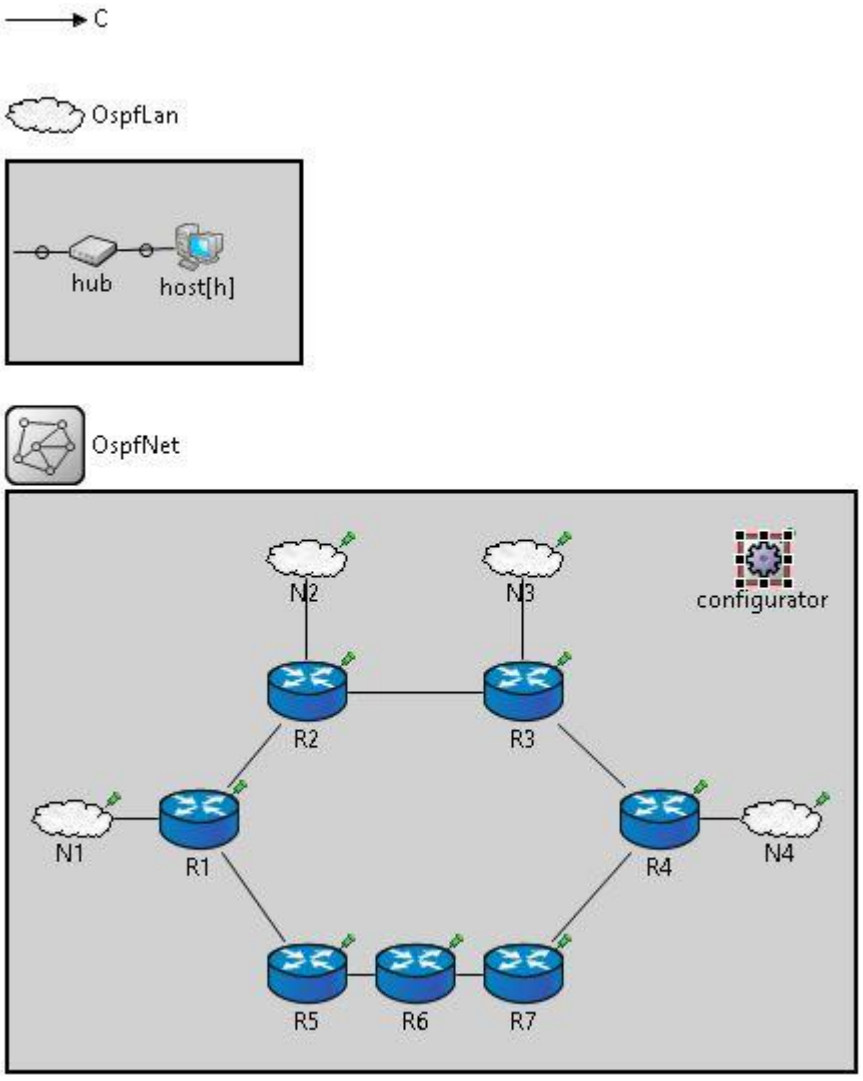

Рисунок 21 – Полученная NED-модель файла network.ned

На иллюстрации изображены все компоненты, входящие в моделируемую OSPF-сеть. Вверху изображен канал, названный «C» в network.ned файле. Изображение канала представлено в виде стрелки. Под изображением канала находится изображение сетевого хаба, названного OspfLan. Сетевой хаб OspfLan в рамках всей моделируемой сети OspfNet будет изображен в виде облака. В случае рассмотрения структуры хаба OspfLan, он будет представлен структурой в виде сетевого коммутатора и заданного числа компьютеров (хостов). Коммутатор будет соединен со всеми компьютерами с одной стороны, а с другой у него будет интерфейс для выхода во внешнюю среду хаба.

Далее на рисунке представлено изображение сети OspfNet в виде модуля, а ниже представлена подробная структура полученной сети.

3.2.2 Эксперимент над исходной моделью

При запуске моделирования проекта ospftest происходит сборка всех входящих в него файлов и их компиляция. После открывается окно моделирования. Иллюстрация полученного окна моделирования представлена на рисунке 22.

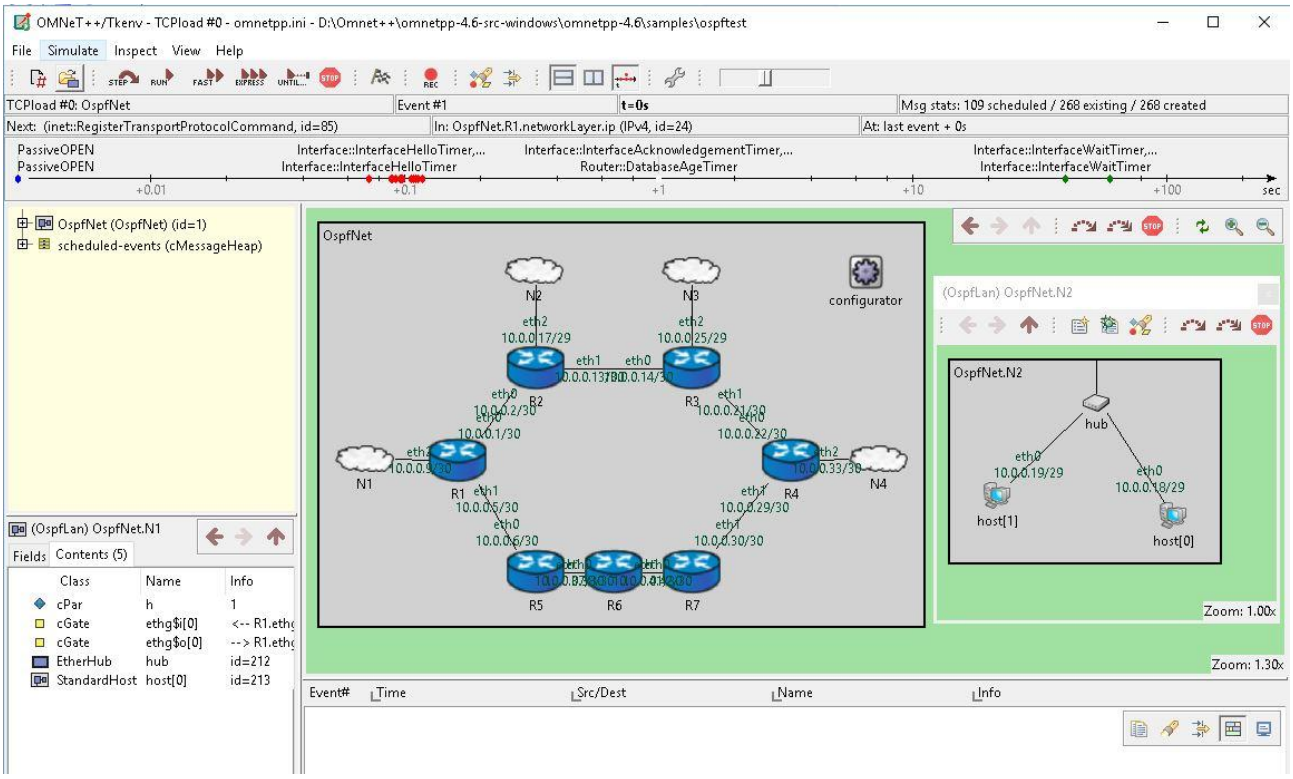

Рисунок 22 – Окно моделирования сети OspfNet

В центре окна изображена моделируемая сеть OspfNet. В сети присутствует элемент с названием «configurator», который располагается в правом верхнему углу сети. Данный элемент используется для выдачи IP-адресов элементам сети. Справа от изображения структуры сети располагается

структура сетевого хаба N2, которая включает в себя сетевой коммутатор и два компьютера (хоста), а так же линии связи, соединяющие элементы сети N2.

Каждый канал связи, соединяющий элементы сети, имеет надписи с каждой стороны соединения. Надписи состоят из названия интерфейса, к которому подведена линия связи, а так же IP-адреса интерфейсов и маски подсети. Маска подсети представлена в виде символа «/» и числа, обозначающего длину префикса.

Запустив через горизонтальное меню симуляцию работы сети, можно наблюдать процесс настройки сети и ее работы. С течением времени окно со списком событий будет постепенно пополняться новыми событиями и информацией о них. Данное окно располагается внизу окна моделирования, под окном с отображением топологии сети.

Над окном с топологией сети располагается временная шкала, занимающая всю ширину окна программы. На временной шкале отображаются события, которые произойдут в течение следующих 100 секунд.

При включении самой низкой скорости симуляции протекание каждого события сопровождается анимацией в окне моделирования. Можно наблюдать инициализацию маршрутизаторов в начале симуляции, отправку Hello-пакетов и установление TCP-соединения с последующей передачей сегментов данных и закрытием TCP-соединения.

По окончанию симуляции в папке проекта был создан файл TCPload.anf, в который были помещены все данные, собранные программой в ходе моделирования. Собранные данные могут быть использованы для получения графиков работы сети. В зависимости от выбора, график может быть выведен в виде скаляра или вектора после выбора необходимых параметров.

Для создаваемого графика был выбран вариант вывода данных в виде вектора. В качестве параметров вывода было выбрано, что на графике должна быть отображена пропускная способность на всех интерфейсах роутера R2. В итоге был получен график, наглядно отображающий проблему, которую протокол OSPF неспособен эффективно решать, а именно: падение пропускной

способности на маршрутизаторах вследствие появления посторонней нагрузки на роутерах, включенных в маршрут передачи сообщений. Полученный график представлен на рисунке 23.

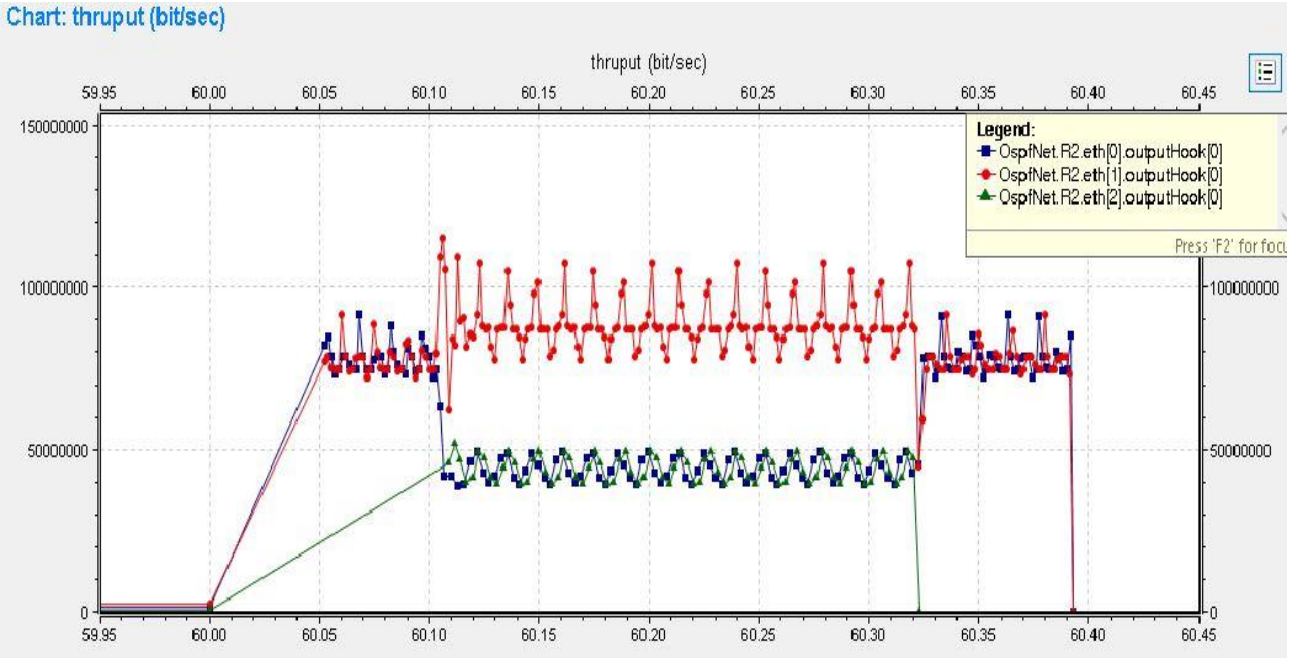

Рисунок 23 – График, полученный в результате моделирования

По оси абсцисс отложено время симуляции, а по оси ординат – величина пропускной способности.

В момент времени 60 сек. происходит инициализация и начало передачи полезных сообщений с сетевого хаба N1 на хаб N4. Согласно структуре сети, OSPF выберет маршрут N1-R1-R2-R3-R4-N4, как маршрут с самой малой метрикой. В момент времени 50.1 происходит инициализация паразитной нагрузки на маршрутизаторах R2 и R3, и начинается передача сообщений большого объема с хаба N2 на N3 по маршруту N2-R2-R3-N3. Вследствие повышения нагрузки на маршрутизаторах R2 и R3 скорость передачи пакетов с R1 и R4 упадет.

3.3 Моделирование протокола AntNet

3.3.1 Описание полученной модели

В качестве сети для моделирования была взята та же сеть, что и в случае моделирования протокола маршрутизации OSPF. Но конфигурация сети претерпела значительные изменения. Состав основных файлов, необходимых для моделирования работы протокола AntNet:

- AntModule.cc ;
- AntPing.msg;
- AntPing_m.cc;
- AntPing_m.h;
- CommandAnt.msg;
- CommandAnt_m.cc;
- CommandAnt_m.h;
- Makefile;
- Config.xml;
- network.ned;
- omnetpp.ini.

Файл AntModule.cc является файлом, необходимым для создания модуля «AntMod» в роутере, так как алгоритмы работы модуля были полностью прописаны в этом файле. Данный модуль будет играть ключевую роль в моделировании протокола маршрутизации на основе роевого интеллекта, так как он будет выполнять функции модуля, который реализует алгоритм работы протокола AntNet. Файл AntModule.cc был написан на языке программирования C++.

Header-файл модуля AntMod представлен на рисунке 24.

```
#include "inet/common/INETDefs.h"
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "AntPing_m.h"
#include "inet/linklayer/ethernet/ComandAnt_m.h"
#include "inet/networklayer/ipv4/IPv4RoutingTable.h"
#include <vector>

namespace inet {

namespace AntMod {

class AntModule : public cModule
{
  public:
    AntModule();
    virtual ~AntModule();

  private:
    bool routeDown;
    int *bufRTT, numRTTmsg, countMessage, indexSlowRoute;

  protected:
    virtual AntPing *generateMessage();
    virtual void forwardMessage(AntPing *msg);
    virtual void handleMessage(cMessage *msg);
    virtual void inspectRTT(cMessage *msg);
    virtual void rebuildRoutingTable(int indexSlowRoute);
};
```

Рисунок 24 – Описание файла AntModule.h

В файле AntModule.h представлены все переменные и методы, используемые в процессе работы модуля AntMod.

В публичной части класса AntMod находятся конструктор и деструктор класса. Конструктор класса вызывается при создании объекта класса для инициализации элементов класса некоторыми начальными значениями. Деструктор служит для уничтожения элементов класса и срабатывает тогда, когда завершается работа программы.

В приватную часть класса занесены все используемые классом переменные. Переменные занесены в данную секцию с целью закрытия доступа к ним из других классов.

В секцию protected были вынесены все используемые в классе функции. Функция generateMessage используется для заполнения полей служебных сообщений ant-ping (рисунок 25).

```
AntPing *AntModule::generateMessage()
{
    char *src, *dest;
    double routeTT;
    src = getParentModule()->getFullName();
    if (src == "R1")
        dest = "R7";
    else dest = "R1";

    char msgname[20];
    sprintf(msgname, "%s-%s-%d", src, dest, countMessage);
    countMessage++;

    AntPing *msg = new AntPing(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    msg->setBufRoute(getParentModule()->getName());
    msg->setCountRoute(0);
    if (!routeDown)
    {
        setTimestamp(routeTT);
        msg->setRTT(routeTT);
    }
    return msg;
}
```

Рисунок 25 – Функция generateMessage

Сначала в функции происходит создание переменных, которые будут использованы для снятия необходимых значения. Затем эти переменные используются для инициализации полей сообщения ant-ping. В поле RTT сообщения ant-ping записывается время начала движения сообщения. В функции предусмотрен отказ от записи RTT в соответствующее поле сообщения при создании «зеркального» ant-ping, то есть того сообщения, которое проделает путь до исходного узла в обратном порядке.

Функция forwardMessage используется в классе AntMod для отправки сообщения ant-ping по всем интерфейсам маршрутизатора (рисунок 26).

```
void AntModule::forwardMessage(AntPing *msg)
{
    for ( int i = 0; i < 3; i++ )
    {
        countRoute++;
        send(msg, "ethg[i]");
    }
}
```

Рисунок 26 – Функция forwardMessage

В функции forwardMessage происходит инкремент переменной countRoute и отправка сообщения ant-ping по определенному интерфейсу. Переменная countRoute используется как счетчик отправленных сообщений ant-ping, что бы в дальнейшем определить, все ли отправленные муравьи были получены.

Функция handleMessage создана с целью сбора служебных сообщений ant-ping и принятия решения об их дальнейшей обработке и передаче или удалении. Как только муравей достигает узла, являющегося пунктом назначения в маршруте следования, в окне симуляции OMNeT++ отображающем топологию сети происходит вывод сообщения о том, что муравей достиг пункта назначения.

Метод inspectRTT служит с целью обработки параметра RTT в муравьях (рисунок 27).

```
void inspectRTT(cMessage *msg)
{
    bufRTT[numRTTmsg]=msg->getRTT();
    numRTTmsg++;

    if ( msg->bufRoute[0] == 2)
        indexSlowRoute = 5;

    if ( numRTTmsg == countMessage)
    {
        double bestRTT = bufRTT[indexSlowRoute];
        bool findNewRoute = false;

        for ( int i = 0; i < numRTTmsg; i++ )
        {
            if ( bestRTT > bufRTT[i])
            {
                bestRTT = bufRTT[i];
                indexSlowRoute = i;
                findNewRoute = true;
            }
        }
        if ( findNewRoute )
            rebuildRoutingTable(indexSlowRoute);
    }
}
```

Рисунок 27 – Функция inspectRTT

Обработка RTT заключается сборе значений RTT в массив bufRoute с каждого прибывшего муравья, проследовавшего до пункта назначения и обратно. После сбора всех значений RTT происходит сравнение RTT каждого маршрута и в случае нахождения пути с RTT меньшим, чем на маршруте по умолчанию происходит изменение таблицы маршрутизации. Функция изменения таблицы маршрутизации называется rebuildRoutingTable и представлена на рисунке 28.

```
void Router::rebuildRoutingTable(int indexSlowRoute)
{
    EV_INFO << "Rebuilding routing table:\n";

    IPv4Route *entry = rt->getRoute(indexSlowRoute);
    rt->deleteRoute(entry);

    IPv4Route *newRoute;
    newRoute->push_back("10.0.0.0");
    newRoute->push_back("*");
    newRoute->push_back("255.255.255.252");
    newRoute->push_back("20");
    newRoute->push_back("10.0.0.1");
    newRoute->push_back("DIRECT_MANUAL");
    rt->addRoute(newRoute);

    EV_INFO << "Routing table was rebuilt.\n"
            << "Results:\n";

    unsigned long routeCount = routingTable.size();
    for ( int i = 0; i < routeCount; i++) {
        EV_INFO << *routingTable[i]
                << "\n";
    }
}
```

Рисунок 28 – Функция rebuildRoutingTable

При вызове функции изменения таблицы маршрутизации происходит передача в данную функцию индекса маршрута, который является маршрутом по умолчанию в таблице маршрутизации и который следует сменить. Смена маршрута происходит через удаление этого маршрута из таблицы маршрутизации rt (Routing Table). После создается новый маршрут newRoute, который затем заносится в таблицу маршрутизации. В окне OMNeT++ для вывода сообщений происходит вывод переопределенной таблицы маршрутизации.

Файл AntPing.msg необходим для определения структуры сообщений, которые будут использоваться модулем AntMod для определения маршрутов с наименьшим значением RTT. Эти сообщения будут называться «ant-ping». Код данного файла представлен на рисунке 29.

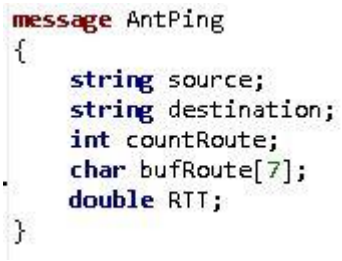

Рисунок 29 – Структура файла AntPing.msg

В файле AntPing.msg создается класс «message», который используется в OMNeT++ специально для сообщений, и данному классу присваивается название AntPing. Далее для данного класса создаются поля сообщения. Для выполнения задач протокола AntNet необходимо создать пять полей для сообщения. В поле «source» будет храниться название модуля, источника, породившего сообщение ant-ping. Поле «destination» будет хранить текстовое значение, которое будет отражать пункт назначения сообщения ant-ping. Поле «countRoute» отражает количество пройденных маршрутизаторов. В поле «bufRoute» будут храниться названия пройденных маршрутизаторов. А в поле «RTT» будет храниться замеренное значение времени, которое будет отражать время прохождения сообщением пути до узла назначения и обратно.

Класс message AntPing будет являться спецификацией сообщения ant-ping. На основании данного класса будут сгенерированы файлы, содержащие базовые методы для работы с сообщениями ant-ping.

Файл CommandAnt.msg, как и AntPing.msg, создается с целью определения структуры служебных сообщений. Приход сообщения CommandAnt инициализирует начало рассылки муравьев с модуля AntMod. Файл CommandAnt.msg имеет схожее с файлом AntPing.msg описание, отличие составляют только используемые поля сообщения. В сообщениях CommandAnt используются поля source, destination и переменная startPing, имеющая тип bool и используемая для обозначения необходимости начала рассылки сообщений ant-ping.

Класс «message», созданный для файлов AntPing.msg и CommandAnt.msg будет наследоваться от базового класса «cMessage», включенный в Header-файл omnetpp.h.

Для генерации файлов, необходимых для работы с сообщениями ant-ping и command-ant, будет использоваться файл «Makefile», который включен в проект AntProject , созданный для моделирования протокола AntNet. Makefile настроен таким образом, что в процессе работы компилятора вызывается opp_msgc и генерирует AntPing_m.h, AntPing_m.cc, CommandAnt_m.h и CommandAnt_m.cc из декларации сообщения. Они будут содержать сгенерированный класс AntPing и CommandAnt, являющиеся подклассом от cMessage. Вследствие этого классы AntPing и CommandAnt будут иметь методы для работы с соответствующими сообщениями, в числе которых методы получения и установки каждого поля сообщения.

Основными методами, использованными в процессе построения алгоритма работы модуля AntMod с сообщениями являются так называемые методы сеттеры и геттеры. Геттером называется специализированный метод, используемый в объектно-ориентированном программировании с целью получения данных, получение доступа к которым напрямую ограничено из-за использованных модификаторов доступа. Сеттер выполняет функцию противоположную геттеру, он устанавливает значения для полей, доступ к которым напрямую так же ограничен. Список геттеров и сеттеров для работы с сообщениями ant-ping представлен на рисунке 30.

```
virtual const char * getSource() const;
virtual void setSource(const char * source);
virtual const char * getDestination() const;
virtual void setDestination(const char * destination);
virtual double getRTT() const;
virtual void setRTT(double RTT);
```

Рисунок 30 – Список основных использованных методов файла AntPing_m.h

Файл network.ned, принадлежащий проекту моделирования протокола AntNet, претерпел некоторые изменения в сравнении с таким же фалом для проекта моделирования сети OSPF. Для маршрутизаторов был присвоен готовый класс Router, который по умолчанию не поддерживает динамическую маршрутизацию. То есть без объявления протокола маршрутизации, маршрутизация на роутерах настроена не будет. В секции объявления подключаемых файлов была вставлена строчка кода, в которой класс Router подключается к проекту antProject в файле network.ned.

Для построения статической маршрутизации на маршрутизаторах моделируемой сети, в OMNeT++ предусмотрен конфигуратор, который называется IPv4NetworkConfigurator. Все параметры, которые конфигуратор выдаст маршрутизаторам для работы в TCP/IP-сети, должны быть прописаны в специально созданном файле с расширением XML. Такой XML-файл был создан в папке проекта AntProject , и этому файлу было присвоено название Config.xml.

Работа с файлом Config.xml начинается с написания дескриптора «config». Написание данного дескриптора означает, что после его объявления в теле документа следует содержимое, которое ориентировано на работу с модулем IPv4NetworkConfigurator. Содержимое будет считываться, парситься и отправляться на дальнейшую обработку с целью присвоения целевым компонентам сети, на работу с которыми рассчитан конфигуратор. После дескриптора «config» следует объявление элементов и их атрибутов в виде пар «имя-значение». Завершается разметка таких XML-документов завершающим тегом </config>.

Рассмотрим файл Config.xml (рисунок 31)

```
<config>
   <interface hosts="N1" towards="R2" address="172.0.1.1" netmask="255.255.255.0" mtu="1500"/>
   <interface hosts="N2" towards="R3" address="172.0.2.1" netmask="255.255.255.0" mtu="1500"/>

   <interface hosts="R1" towards="N3" address="172.0.3.2" netmask="255.255.255.0" mtu="1500"/>
   <interface hosts="R7" towards="N4" address="172.0.4.2" netmask="255.255.255.0" mtu="1500"/>

   <interface hosts="R1" towards="R2" address="172.0.0.1" netmask="255.255.255.0" mtu="1500"/>
   <interface hosts="R2" towards="R1" address="172.0.0.2" netmask="255.255.255.0" mtu="1500"/>
```

Рисунок 31 – Объявление интерфейсов в файле Config.xml

Сначала список элементов документа представлен объявлением интерфейсов на маршрутизаторах и хабах. Это реализуется с помощью тегов <interface> и последующего перечисления атрибутов в данном теге. Атрибут «hosts» означает объявление имени компонента сети. Атрибут «towards» имеет значение имени компонента, с которым соединяется интерфейс компонента, объявленного в «hosts». Значение атрибута «address» будет присвоено объявляемому интерфейсу в качестве IP-адреса. У объявляемого адреса должна быть маска, которая объявляется в сущности «netmask». Максимальный размер полезного блока данных одного пакета (maximum transmission unit) представляется значением атрибута «mtu» и означает максимальный размер блока пакета, который может быть передан целевым протоколом без фрагментации.

На рисунке 32 представлено объявление другой части элементов в файле Config.xml.

```
  <route hosts="R5" destination="R3" netmask="/32" metric="20" interface="eth1"/>
  <route hosts="R5" destination="R6" netmask="/32" metric="20" interface="eth1"/>
  <route hosts="R5" destination="R7" netmask="/32" metric="20" interface="eth1"/>
  <route hosts="R5" destination="N2.*" netmask="/30" metric="20" interface="eth1"/>
  <route hosts="R5" destination="N4.*" netmask="/30" metric="20" interface="eth1"/>
  <route hosts="R5" destination="*" netmask="/0" metric="20" interface="eth0"/>
</config>
```

Рисунок 32 − Объявление маршрутов в файле Config.xml

В качестве объявляемых в файле Config.xml элементов представлено объявление статических маршрутов, которые будут занесены в таблицу маршрутизации (Routing Table) роутера моделируемой сети. Объявление статического маршрута в качестве элемента начинается с дескриптора <route>. После объявления дескриптора происходит перечисление атрибутов. Атрибут «hosts» означает определение узла, являющегося источником маршрута. В значении атрибута «destination» указывается целевой пункт назначения, в который должен проделать свой путь отправляемый пакет. Сущность, представленная атрибутом «netmask», означает маску подсети, которая используется для отделения IP-адреса сети от IP-адреса сетевого узла в этой сети. На рисунке значение имени «netmask» представлено в виде префикса, значение которого отражает количество двоичных единиц в маске подсети. Значение атрибута «metric» показывает цену описываемого в теге <route> маршрута. Атрибут «interfaces» используется для обозначения в сетевом узле имени интерфейса, через который следует отправлять пакеты для описанного маршрута.

Заканчивается файл Config.xml завершающим тегом </config>.

Файл omnetpp.ini по сравнению с таким же файлом в проекте сети, построенной для протокола OSPF, почти не претерпел изменений. Было изменено название моделируемой сети на «AntNetwork».

3.3.2 Эксперимент над полученной моделью

Полученная модель выглядит таким же образом, как и на рисунке для сети моделирования протокола OSPF, но алгоритм работы для полученной модели другой.

Внутренняя структура роутера, использующего в качестве протокола маршрутизации протокол AntNet, представлена на рисунке 33.

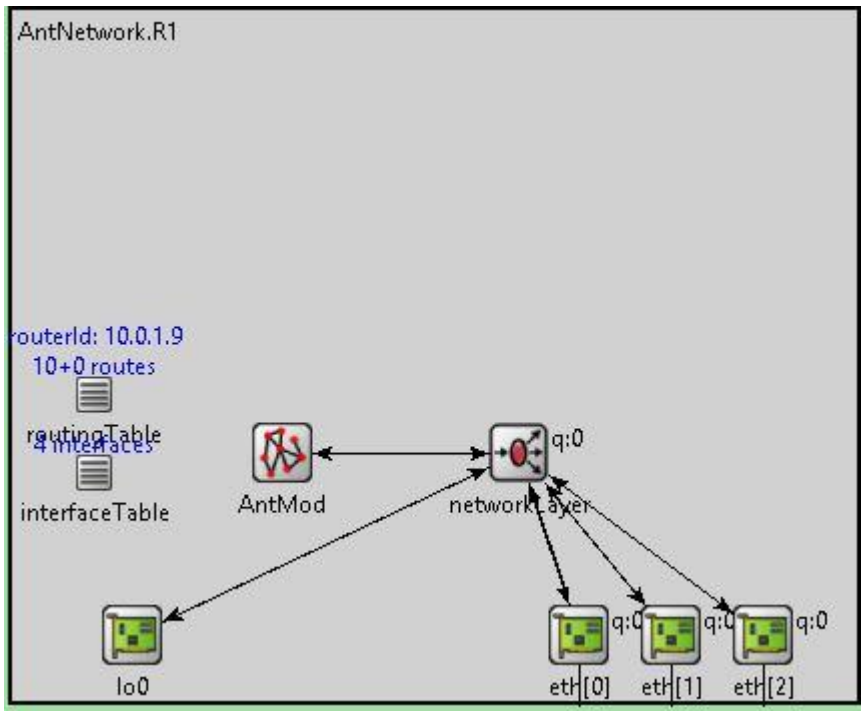

Рисунок 33 – Структура роутера R1

Из представленной структуры видно, что роутер R1 состоит их нескольких сетевых интерфейсов, таблицы маршрутизации и модулей, среди которых есть модуль AntMod, реализующий работу протокола AntNet на данном роутере. Таблица маршрутизации включает 10 записей. Таблица интерфейсов включает интерфейс loopback и 3 интерфейса, соединяющих роутер R1 с другими элементами сети.

По окончанию процесса симуляции работы сети в папке results проекта AntProject был получен файл TCPload-0.vec. Данный файл был использован для получения графика, на котором показан вектор пропускной способности роутеров R1 и R2. График отображен на рисунке 34.

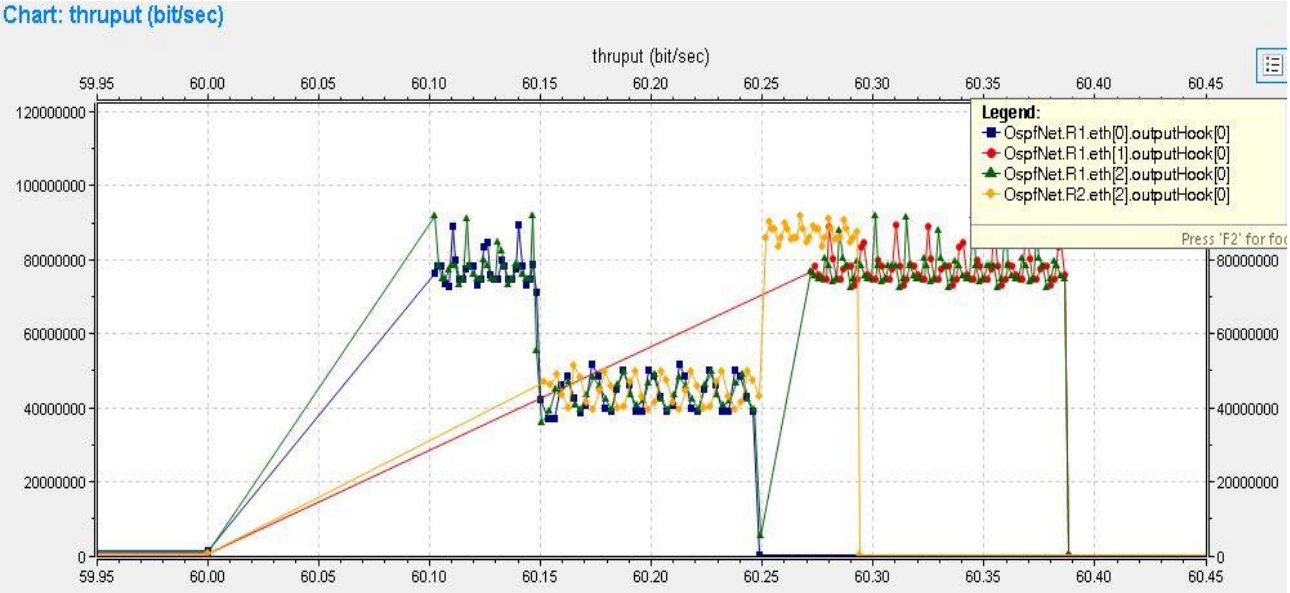

Рисунок 34 – График, полученный в результате моделирования AntNet

По оси абсцисс отложено время симуляции, а по оси ординат – величина пропускной способности.

В момент времени 60 сек. происходит инициализация и начало передачи полезных сообщений с сетевого хаба N1 на хаб N4. Согласно структуре сети, маршрут от сетевого хаба N1 на N4 будет проходить через роутеры R1-R2-R3-R4, так как данное правило было заведено через файл Config.xml данного проекта. В момент времени 60.15 происходит инициализация паразитной нагрузки между маршрутизаторами R2 и R3, и начинается передача сообщений большого объема с хаба N2 на N3 по маршруту N2-R2-R3-N3. Вследствие повышения нагрузки на маршрутизаторах R2 и R3 скорость передачи данных с N1 на N4 упадет примерно в 2 раза. Модуль AntMod маршрутизатора R1 в очередной раз отправит муравьев и тем самым обнаружит, что RTT маршрута по умолчанию является больше, чем на маршруте с большей метрикой, но отсутствующей нагрузкой на каналах. После чего будет изменена таблица маршрутизации и пакеты пойдут по маршруту N1-R1-R5-R6-R7-R4-N4. На графике в момент времени 60,2476 происходит обрыв связи вследствие удаления действующего маршрута и добавления нового. В момент времени

50.2763 завершается процесс обновления таблицы маршрутизации на роутере R1, вследствие чего данные на маршрутизаторе R1 начинают передаваться через шлюз eth[1], который соединен с роутером R5. Пропускная способность каналов роутера R1вновь используется в полной величине. Из-за перестроения таблицы маршрутизации на R1 передача данных с хаба N2 на N3 продолжилась с полным использованием всей пропускной способности канала R2-R3.

ЗАКЛЮЧЕНИЕ

В данной работы был рассмотрен протокол маршрутизации на основе роевого интеллекта – протокол AntNet. Данный протокол был промоделирован в среде моделирования OMNeT++. Основанием для моделирования послужила проблема, выявленная в результате моделирования протокола по состоянию канала связи – OSPF. Результаты показали, что OSPF неспособен эффективно решать проблему отслеживания загруженности каналов связи и производить последующую перестройку таблицы маршрутизации.

В процессе разработки AntNet был создан граф состояний протокола, кратко описывающий алгоритм работы протокола AntNet.

С помощью библиотеки INET Framework для OMNeT++ был реализован модуль, обеспечивающий работу протокола AntNet.

Моделирование нового протокола было выполнено на той же тестовой сети. На основании полученных результатов было увидено, что протокол AntNet в определенный момент времени отследил, что на используемом маршруте передачи данных время следования пакетов больше чем на другом маршруте. Таким образом, протокол отследил большую нагрузку на используемом канале, после чего таблица маршрутизации была перестроена, и данные стали передаваться по другому маршруту.

В процессе моделирования AntNet было увидено, что протокол может быть использован в исследованиях по повышению производительности сети при больших нагрузках за счет использования алгоритмов, с помощью которых замеряется время следования пакетов по сети и происходит перенаправление трафика по маршруту с меньшим показателем RTT.

В результате выполнения данной работы была рассмотрена работоспособность сети, использующей в качестве протокола маршрутизации протокол AntNet. Для достижения эффективности в работе протокола AntNet требуются дополнительные исследования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Система менеджмента качества. Общие требования к построению, изложению и оформлению документов учебной деятельности. СТО 4.2-07-2014. Красноярск.: ИПЦ СФУ, 2014. – 60 с.

2 Сайт Microsoft [Электронный ресурс] – Режим доступа: https://msdn.microsoft.com/ru-ru/library/cc785246(v=ws.10).aspx

3 Сайт о программировании Realcoding.Net [Электронный ресурс] – Режим доступа: http://www.realcoding.net/articles/vvedenie-v-protokol-ospf.html

4 Электронный журнал ITC [Электронный ресурс] – Режим доступа: http://itc.ua/articles/razgovor_o_marshrutizacii_ne_okonchen_25136/

5 Компьютерные сети, Олифер В., Олифер Н., изд.: Питер , 2010. 943 с.

6 Электронная энциклопедия [Электронный ресурс] – Режим доступа: http://dic.academic.ru/dic.nsf/ruwiki/1605981

7 Динамический алгоритм маршрутизации на основе агентных технологий, Солдатова В.А. Кафедра ПМИ ДонНТУ [Электронный ресурс] – Режим доступа: http://masters.donntu.org/2005/fvti/soldatova/library/soldatova.htm

8 Моделирование беспроводных сетей с помощью специализированных инструментов, Солнов А.И, автореф. дис. [Электронный ресурс] – 5с., Режим доступа:http://conf.mirea.ru/CD2013/pdf/p5/5.pdf

9 Официальный сайт OMNeT++ [Электронный ресурс] – Режим доступа:https://omnetpp.org/models

10 OMNeT++ simulator [Электронный ресурс] – Режим доступа: http://omnetsimulator.com

11 Ant Colony Optimization and its Application to Adaptive Routing in Telecommunication Networks : дис. д-ра физ.-мат. наук / Gianni Di Caro. – Bruxelles, 2004 – 121 с.

12 AntNet: Distributed Stigmergetic Control for Communications Networks / Gianni Di Caro, Marco Dorigo // Journal of Articial Intelligence Research 9 : сб. науч. ст. / IRIDIA, Universite Libre de Bruxelles, 1998. − С. 317–365.