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

Институт космических и информационных технологий
институт
Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ О.В. Непомнящий
подпись

«____»_____2020 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Методы управления многоадресной передачей данных
тема

09.04.01 – Информатика и вычислительная техника
код и наименование направления

09.04.01.05 – Сети ЭВМ и телекоммуникации
код и наименование магистерской программы

Научный руководитель	_____ *подпись, дата*	канд. физ.-мат. наук.	К.В. Коршун *инициалы, фамилия*
Выпускник	_____ *подпись, дата*		В.И. Туркевич *инициалы, фамилия*
Рецензент	_____ *подпись, дата*	директор ЗАО «Интертакс»	М.В. Алексеев *инициалы, фамилия*
Нормоконтролер	_____ *подпись, дата*	канд. физ.-мат. наук.	К.В. Коршун *инициалы, фамилия*

Красноярск 2020

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

Институт космических и информационных технологий

институт

Вычислительная техника

кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ О.В. Непомнящий
подпись

«____»_____2019 г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

в форме магистерской диссертации

Студенту Туркевичу Владиславу Ивановичу.

Группа: КИ18-01-5м Направление (специальность): 09.04.01 Информатика и вычислительная техника.

Тема выпускной квалификационной работы: «Методы управления многоадресной передачей данных».

Утверждена приказом по университету № 2096/с от 14.02.2019

Руководитель ВКР: Коршун К.В., кандидат физ.-мат. наук, доц., доцент НУЛ телекоммуникационных систем кафедры ВТ ИКИТ СФУ.

Исходные данные для ВКР: Исследование методов многоадресной передачи данных для разработки алгоритма функционирования надежного многоадресного транспортного протокола.

Перечень разделов ВКР: Реферат, Введение, Обзор предметной области, Разработка надежного многоадресного транспортного протокола, Реализация протокола, Заключение, Список используемых источников.

Перечень графического материала: презентация.

Руководитель ВКР _____ К.В. Коршун
подпись *инициалы, фамилия*

Задание принял к исполнению _____ В.И. Туркевич
подпись *инициалы, фамилия*

«____»_____2019 г

РЕФЕРАТ

Выпускная квалификационная работа магистра по теме «Методы управления многоадресной передачей данных» содержит в себе 49 страниц текстового документа, 11 использованных источников, 4 таблицы и 14 иллюстраций.

MULTICAST, ACK, НАДЕЖНАЯ ДОСТАВКА ДАННЫХ, PGM, M/TCP, TREE-BASED ACK, МНОГОАДРЕСНОЕ ДЕРЕВО.

Цель работы: разработка алгоритма функционирования надежного многоадресного транспортного протокола.

Во введении раскрывается актуальность работы, ставятся цели и задачи.

В первой главе изложены результаты обзора существующих многоадресных транспортных протоколов, и определены критерии, предъявляемые к новому протоколу.

Во второй главе приведены результаты разработки алгоритма функционирования надежного многоадресного транспортного протокола, соответствующего критериям, определенным в первой главе.

В третьей главе описана демонстрационная реализация протокола, описан виртуальный стенд, на котором производится тестирование реализации, а также приведены результаты тестирования.

В результате работы разработан алгоритм функционирования надежного многоадресного транспортного протокола и произведена проверка его работоспособности на виртуальном стенде.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность исследования. Задача гарантированной передачи данных группе получателей часто возникает в различных приложениях, таких, как текстовые и мультимедийные чаты, распространение файлов (например, обновление программного обеспечения), аудио-видео трансляции, и так далее. Для ее решения в настоящее время применяются три основных подхода.

Клиент-серверный подход предполагает создание выделенного сервера, который пересылает данные от отправителя всем получателям. Недостатками данного подхода являются наличие единой точки отказа, а также высокие требования к аппаратному обеспечению и каналу связи [1].

Децентрализованный подход (peer-to-peer) предполагает создание наложенной сети непосредственно между получателями. Недостатком данного метода является зависимость скорости передачи информации от количества клиентов, находящихся онлайн, а также проблема поиска узлов наложенной сети («пиров») [3].

Третий подход – использование многоадресной передачи данных на сетевом уровне (multicast). Преимуществом подхода является максимально эффективное использование каналов связи. Недостатками является необходимость поддержки на сетевом уровне, а также несовместимость транспортного протокола TCP с multicast. В силу указанных недостатков данный подход применяется в основном для аудио-видео трансляций в пределах корпоративной или провайдерской сети [4].

Решением мог бы стать транспортный протокол с гарантированной доставкой потока данных, использующий многоадресную рассылку. В

настоящее время не существует универсального протокола такого типа. Целью работы является исследование методов передачи данных и создание протокола, соответствующего ряду, определенных в результате сравнения существующих реализаций, критериев.

Целью данной работы является разработка алгоритма функционирования надежного многоадресного транспортного протокола.

Задачи:

1. Исследование основных подходов к управлению передачей данных;
2. Исследование и сравнительный анализ существующих многоадресных транспортных протоколов;
3. Определение требований, предъявляемых к новому транспортному протоколу;
4. Разработка алгоритма функционирования нового транспортного протокола;
5. Разработка реализации алгоритма и тестирование ее работоспособности на лабораторном стенде.

1 Обзор предметной области

На данный момент не существует транспортного протокола общего назначения с гарантированной доставкой, использующего технологию многоадресной передачи данных (Reliable Multicast protocol, RM). Задача его реализации является крайне трудоемкой.

Различные приложения имеют разные требования к транспортному протоколу этот факт ограничивает пространство разработки. Так, например, к двум приложениям, имеющим различные требования, не может быть

применено одно решение. Однако, существуют успешные узкопрофильные реализации, предназначенные для приложений определенного типа.

1.1 Основные подходы к управлению передачей данных

1.1.1 Ограничения со стороны приложения

Требования к приложениям для надежной многоадресной передачи столь же широки и разнообразны, как и сами приложения. Однако существует ряд требований, которые существенно влияют на разработку протокола с технологией RM. Краткий список включает в себя:

- Необходима ли приложению информация о том, что все приемники получили данные?
- Необходимо ли приложению работать с большим числом приемников?
- Необходимо ли приложению обеспечивать полностью надежную доставку данных?
- Необходимо ли приложению осуществлять ограниченную по времени доставку данных?

Подтверждение доставки данных

Во многих приложениях логически определенные единицы данных должны доставляться нескольким клиентам, например, файл или набор файлов, пакет программного обеспечения и т.д. Блок данных приложения (ADU) определяется как логически отделимый блок данных, который полезен для приложения. В некоторых случаях единица данных приложения может быть достаточно короткой, чтобы помещаться в один пакет, тогда как в других случаях может быть намного длиннее пакета [7].

Протокол может гарантировать надежную доставку с помощью механизма подтверждений, информирующего отправителя о факте доставки данных. Существует два типа подтверждений: на уровне блока данных приложения и на уровне пакета. Первый используется чтобы, например, определить, когда необходимо прекратить отправку пакетов конкретного блока данных приложения. Второй используется, например, для определения момента, когда можно освободить буферное пространство, используемое для хранения пакетов, доставка которых была подтверждена [7].

Некоторым приложениям строго необходимо получать подтверждения доставки ADU от всех приемников, или же информацию о том, какие конкретно приемники не получили ADU.

Масштабируемость

Масштабирование набора получателей является одним из наиболее важных ограничений при выборе механизма подтверждения доставки. Так, например, протокол использующий механизм, при котором каждый отправитель отправляет источнику пакет ACK после каждого полученного пакета данных плохо масштабируется, так как источнику необходимо обрабатывать ACK пакеты от каждого получателя в следствии этого, при огромном количестве получателей могут возникать перегрузки. Для решения данной проблемы могут быть использованы различные механизмы подтверждения доставки. Данные механизмы описаны в пункте 1.1.3.

Полностью надежная доставка

Многие приложения требуют полностью надежной доставки ADU. В таких приложениях потеря какого-либо сегмента ADU ведет к тому, что остальные сегменты данного блока перестают быть полезными. Например, приложения для передачи файлов требуют полностью надежной доставки.

Однако некоторые приложения не нуждаются в полной надежности доставки. Примером является трансляция аудио, где отсутствующие пакеты лишь снижают качество принимаемого аудио.

Ограниченная по времени доставка данных

Некоторые приложения имеют жесткие ограничения по времени доставки - если данные не поступают к получателю к определенному времени, нет смысла их доставлять вообще [7]. Такая ограниченность может возникать при передаче данных в реальном времени, например при передаче потокового аудио или видео. В данном случае чрезвычайно важно, чтобы приложение имело контроль на тем, что оно передает в конкретный момент времени.

1.1.2 Ограничения со стороны сети

Свойства сети, в которой развертывается приложение, так же могут налагать определенные ограничения на разработку протокола.

Однонаправленные каналы

В сетях определенных типов может отсутствовать какая-либо форма обратной связи. Основным примером является спутниковое вещание, где обратный канал может быть очень узким или вообще отсутствовать. Для таких сетей пространство решений очень ограничено так как пропадает возможность реализовать большинство механизмов подтверждения доставки. Единственным решением является механизм на основе прямой коррекции ошибок (англ. Forward Error Correction, FEC).

Уровни поддержки со стороны сети

Транспортный протокол надежной многоадресной рассылки предполагает механизмы, работающие на конечных узлах и маршрутизаторы, которые пересылают многоадресные пакеты. Однако, возможны реализации прибегающие к некоторой дополнительной степени поддержки со стороны узлов сети. Все возможные реализации можно разбить на четыре класса в зависимости от степени поддержки со стороны сети [8].

Никакой дополнительной поддержки. Маршрутизаторы просто пересылают пакеты, протокол работает только на конечных узлах сети.

Многоуровневый подход. Данный подход предполагает, что данные разбиты на несколько групп многоадресной рассылки, а получатели присоединяются к соответствующим группам, чтобы получать только необходимый им трафик. Реализация данного подхода требует дополнительной поддержки со стороны маршрутизаторов.

Серверный подход. В данном подходе применяются дополнительные узлы, которые берут на себя часть функций по доставке данных или агрегированию информации о доставке данных.

Подход на основе поддержки со стороны маршрутизаторов. При использовании данного подхода маршрутизаторы берут на себя весь функционал по доставке данных получателям, который включает в себя и механизмы контроля за доставкой и возможное агрегирование информации о доставке данных. Поскольку маршрутизаторы могут напрямую влиять на многоадресную маршрутизацию, они имеют больший контроль над тем, какой трафик направляется членам группы, чем узлы на основе серверного подхода. Однако маршрутизаторы обычно не имеют большого объема свободной памяти или вычислительной мощности, что ограничивает функционал, который возможно реализовать в них.

1.1.3. Основные механизмы подтверждения доставки

Две основные проблемы, возникающие при разработке протокола с технологией RM это, проблема контроля перегрузки и обеспечения хорошей пропускной способности. Потеря пакетов играет главную роль в отношении этих проблем и является основным препятствием, которое необходимо преодолеть для достижения хорошей пропускной способности и для обеспечения работы сети без перегрузок. Таким образом, регистрация потери пакета и реагирование на нее имеет решающее значение для решения этих проблем. Механизмы контроля за доставкой могут использовать один или несколько следующих методов [7]:

- Подтверждение пакета данных.
- Отрицательное подтверждение отсутствующих пакетов данных.
- Избыточность, позволяющая принимать не все пакеты.

Механизм на основе подтверждения пакета данных (англ. ACK-based mechanism). Данный механизм является наиболее простым. Каждый получатель отправляет отправителю пакет с подтверждением после каждого принятого пакета данных. Если подтверждение не приходит, то отправитель совершает повторную отправку пакетов. Такие механизмы ограничены очень маленьким числом получателей ввиду неспособности отправителя обрабатывать большое количество подтверждений.

Механизм на основе дерева подтверждений (англ. Tree-based ACK Mechanisms). Данный механизм организован по структуре связного ациклического графа , в котором получатели являются конечными вершинами, а корнем является отправитель. Получатели генерируют ACK-пакет для родительского узла, который, в свою очередь, агрегирует эти ACK-пакеты и отправляет своему родительскому узлу, таким образом все подтверждения доходят до отправителя, этот механизм относится только к передаче подтверждений о доставке, в то время как данные передаются как

обычно от отправителя к получателю. Данный механизм является хорошо масштабируемым и обладает хорошей отказоустойчивостью, но требует определённой степени поддержки со стороны сети.

Механизм на основе негативных подтверждений (англ. NACK-based mechanisms). Вместо отправки подтверждений для каждого полученного пакета данных получатели отправляют так называемые отрицательные подтверждение (NACK) для каждого пакета данных, который они не получили. Это имеет ряд преимуществ перед механизмами на основе обычных подтверждений (ACK):

— Отправителю больше не нужно знать точное количество получателей. Это устраняет этап построения топологии, необходимый для алгоритмов на основе ACK.

— Повышение отказоустойчивости.

— Требуется только одно NACK от любого из получателей, для того чтобы донести до отправителя информацию о том какой пакет потерян и у какого числа получателей.

Недостатками является то, что отправителю труднее определить тот момент, когда можно освободить буфер передачи, и что необходимы дополнительные механизмы что бы определить действительно ли все пакеты данных дошли до получателя.

Механизм на основе прямого исправления ошибок. Прямое исправление ошибок (англ. Forward Error Correction) — это хорошо известный метод защиты данных от повреждений. Самая простая форма FEC на уровне пакета состоит в том, чтобы применить к группе пакетов битовую операцию поразрядного деления (XOR), в результате этой операции сформировывается новый пакет, который отправляется вместе с остальными. Если, например, было отправлено три исходных пакета плюс пакет XOR, то, если получатель пропустил какой-либо один из исходных пакетов данных, но

получил пакет XOR, он может воспроизвести отсутствующий исходный пакет.

1.2 Обзор существующих методов передачи данных

1.2.1 Протокол Pragmatic General Multicast (PGM)

Pragmatic General Multicast (PGM) - это надежный многоадресный транспортный. PGM работает с IP-multicast. PGM гарантирует, что получатель в группе либо получит все пакеты данных, либо сможет обнаружить невосстановимую потерю пакетов данных. Масштабируемость достигается за счет построения иерархии и использования следующих механизмов: FEC (прямое исправление ошибок), NACK (отрицательное подтверждение) и подавление NACK. PGM является экспериментальным протоколом [11].

Использование данного протокола предполагает наличие в сети узлов, поддерживающих PGM. Тем не менее, он так же предназначен для работы, хотя и с меньшей эффективностью, в сетях, где некоторые или все узлы не поддерживают PGM.

PGM позволяет приемникам присоединяться и отсоединяться в любое время, обеспечивая надежность только в пределах текущего окна передачи. Следовательно, он лучше всего подходит для приложений, которые способны восстановить данные на своем уровне, в случае невосстановимой

потери на уровне протокола, таким образом PGM предпочтителен для приложений, которым важно получать данные в реальном времени.

Примеры приложений — это распространение котировок акций и создание образа диска. В случае котировок акций надежность очень важна, но, если к моменту отмены котировки передача не состоялось, то от повторной передачи следует отказаться. При создании образа диска более эффективно продолжать передачу новых данных, а не замедлять работу в угоду нескольких «медленных» получателей, которые позже могут извлечь недостающие данные с помощью стандартных методов клиент-сервер.

Архитектура

Дерево PGM строится с использованием узлов, поддерживающих PGM в существующем многоадресном дереве. Отправитель многоадресно передает последовательные пакеты данных (ODATA) получателям. Когда получатели обнаруживают отсутствующие пакеты, они направляют NAK своему родителю. Родитель подтверждает прием NAK, многоадресно отправляя в ответ подтверждение получения NAK (NCF). Пакет восстановления, (RDATA) генерируется либо хостом-источником, либо локальным узлом, выделенным для процедур восстановления (DLR) в ответ на NAK. RDATA — это, в зависимости от параметров сеанса, либо повторно отправленный потерянный пакет, либо пакет FEC. Перед отправкой NAK, получатели взводят определенный таймер, и если до истечения времени будет получен соответствующий NCF, то отправка NAK отменяется, таким образом обеспечивается получение только одной копии потерянного пакета для группы получателей [11]. Более наглядно этот процесс описан на рисунке 1.

Иерархия

PGM хранит и поддерживает в актуальном состоянии дерево пути от каждого отправителя multicast-сообщений до слушателей. Каждый узел

дерева знает, откуда ему пришел пакет и в случае, если лист дерева (слушатель multicast-сообщений) обнаружил, что пакет не получен, подтверждение потерянного пакета может подняться вверх от листа к корню дерева (отправитель multicast-сообщений). На каждом узле дерева выставляется состояние, означающее, что в подсети потерян пакет. При повторной отправке потерянный пакет может игнорироваться на узлах, где это состояние не выставлено и дойдет только до тех подсетей, где произошла потеря пакета. Поддержание дерева пути достигается благодаря сообщениям пути (path messages) которые посылаются в multicast-группы через равные промежутки времени каждым отправителя [11].

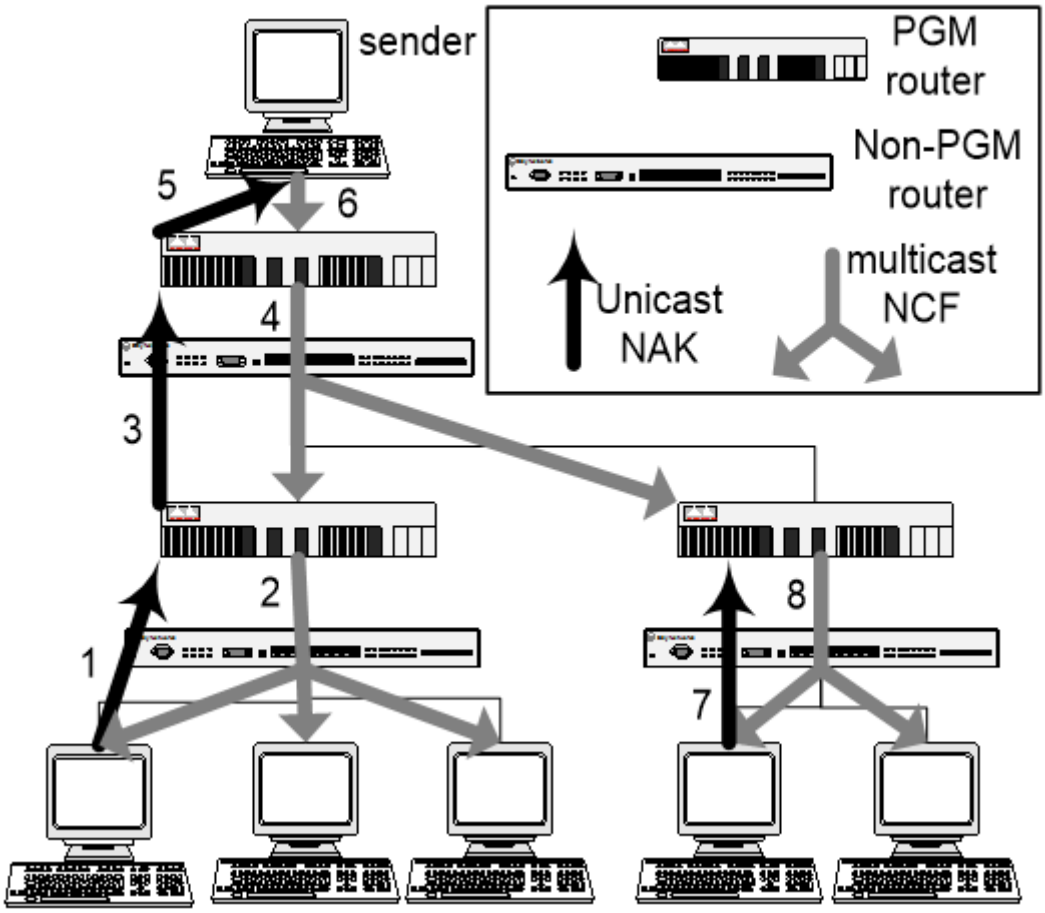


Рисунок 1 – Сценарий NAK/NCF.

(1) Получатели отправляют NAK о потерянном пакете. (2) Родительский маршрутизатор осуществляет многоадресную передачу NCF. (3) Маршрутизатор направляет NAK своему родителю, который (4) осуществляет многоадресную передачу NCF. (5) NAK передается отправителю, который отвечает многоадресным NCF (6). Позднее другой получатель обнаруживает потерю того же пакета и одноадресно передает NAK (7) своему родителю, который многоадресно передает NCF (8). Родитель не отправляет восходящий NAK, поскольку он уже отметил прием соответствующего NCF [11].

1.2.2 M/TCP (Multicast-extension to TCP)

M/TCP разработан на основе инициированной отправителем многоадресной передачи (SIM) и явной многоадресной рассылки (Xcast)[10].

M/TCP может применяться к тем приложениям, в которых отправитель инициирует передачу данных. Например FTP, SMTP. Для приложений, работающих с данными протоколами, требуется только реализация на стороне отправителя. Получатели принимают пакеты как одноадресные, поэтому нет необходимости изменять как стеки протоколов на стороне получателя, так и приложения.

Данный протокол разработан на основе концепции явной многоадресной рассылки. M/TCP предоставляет возможность надежной многоадресной передачи данных и поддерживает существующие приложения работающие на основе обычного TCP. Схема передачи данных в M/TCP представлена на рисунке 2.

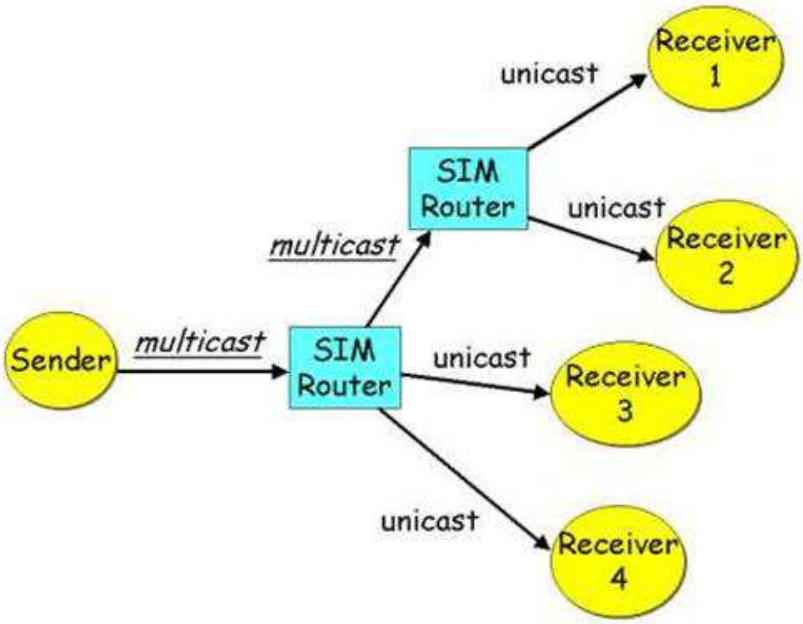


Рисунок 2 – Передача данных в M/TCP.

Основными недостатками M/TCP являются необходимость поддержки со стороны сети, возможность работы только с небольшим количеством приемников и возможность работы только с определенными типами приложений.

M/TCP предназначен для поддержки приложений, имеющих следующие характеристики [10]:

Высокая надежность. Так как M/TCP является разновидностью TCP он обеспечивает высокую надежность передачи данных.

Асимметричная передача данных. По существу, M/TCP предназначен для многоадресных обменов один-ко-многим. M/TCP применим в основном для асимметричного приложения, где трафик от отправителя намного больше, чем трафик от получателей, передача инициируется со стороны отправителя.

Не в режиме реального времени. Механизмы управления скоростью передачи использующиеся в TCP не подходят для передачи мультимедийных данных в реальном времени.

При передаче данных M/TCP берет в расчет возможности каждого приемника. Когда поднабор получателей в группе теряет пакет, потерянный пакет повторно передается в другом потоке TCP. Таким образом скорость передачи не зависит от самого медленного приемника.

При потере пакета выполняется следующий алгоритм [10]:

— Если два или более получателя потеряли один и тот же пакет, повторная передача будет выполняться многоадресной передачей.

— В случае, когда только один получатель потерял пакет, передача будет выполнена многоадресно.

— После завершения повторной передачи новые пакеты будут передаваться отдельно созданной группе получателей, у которых наблюдалась потеря пакетов.

— Данная группа будет вновь объединена с основной, когда достигнет одинаковой скорости с исходным потоком передачи.

1.3 Сравнительный анализ методов передачи данных

По результатам обзора существующих RM протоколов можно сделать вывод что приложения, работающие с протоколом PGM, не получают информацию о том получили ли все приемники данные, PGM за счет механизма NACK и своей иерархии является отлично масштабируемым, обеспечивает полностью надежную доставку данных, в нем отсутствует возможность работы с односторонними каналами и с прерывистыми потоками данных, также рекомендуема частичная поддержка со стороны сети. PGM лучше всего подходит для передачи файлов.

M/TCP же удовлетворяет потребность приложений в получении информации о том, что все приемники получили данные за счет использования механизма ACK, но способен работать с очень маленьким количеством приемников, также данный протокол способен работать с прерывистыми потоками данных. M/TCP необходима полная поддержка на сетевых узлах. M/TCP лучше всего подходит для текстовых чатов или приложений передающих Push-уведомления.

В таблице 1 приведено сравнение вышеперечисленных протоколов и протокола UDP:

1-Информация о том, что все приемники получили данные

2-Масштабируемость

3-Полностью надежная доставка данных

4-Ограниченная по времени доставка данных

5-Поддержка однонаправленных каналов

6-Требование поддержки со стороны сети

Таблица 1 – Сравнение различных транспортных протоколов

Протокол	Требования приложений				Ограничения со стороны сети		Типичные приложения
	1	2	3	4	5	6	
UDP	-	+	-	+	+	-	Аудио-видео трансляции, IPTV
PGM	-	+	+	-	-	+/-	Передача файлов
M/TCP	+	-	+	-	-	+	Текстовые чаты, передача Push-уведомлений
*	+	+	+	-	+	-	Передача управляющих команд в реальном времени

* - Несуществующий, на данный момент, протокол.

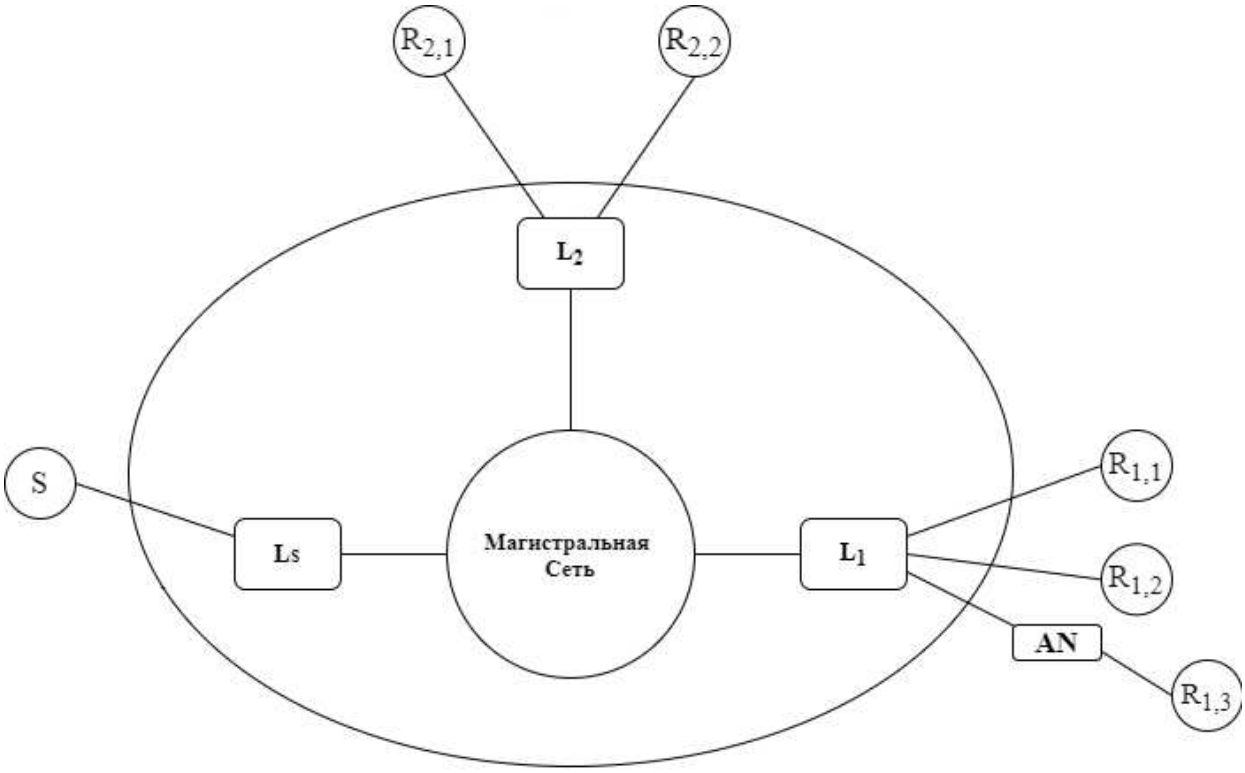
1.4 Выводы

Исходя из анализа в п. 1.3 можно сделать вывод что существующие реализации решают узкий круг задач. На данный момент нет протокола, способного работать с прерывистым потоком данных, не требующего поддержки со стороны сети и хорошо масштабируемого. Примером использования такого протокола является приложение, передающее управляющие команды огромному числу приемников.

2 Разработка надежного многоадресного транспортного протокола

Архитектура сети

Пусть отправители и получатели будут подключены к магистральной сети через локальные коммутаторы доступа, прямо или косвенно через узлы доступа (рисунок 3).



S - Отправитель
Ls - Локальный коммутатор доступа для отправителя
Li - Локальный коммутатор доступа для i-й области
Ri,j - j-й получатель для i-й локальной области
AN - Узел доступа

Рисунок 3 – Модель сети.

Приемники могут быть сгруппированы в локальные области на основе их близости в сети.

Многоадресное дерево, корнем которого является отправитель, а вершинами являются все получатели, определяется на сетевом уровне и называется глобальным. Также существуют локальные многоадресные

деревья, которые являются частями глобального. Глобальное многоадресное дерево показано сплошными линиями на рисунке 4. Получатели в локальной области, обслуживаемые Li, обозначаются как Ri,j. Локальные коммутаторы доступа не являются получателями.

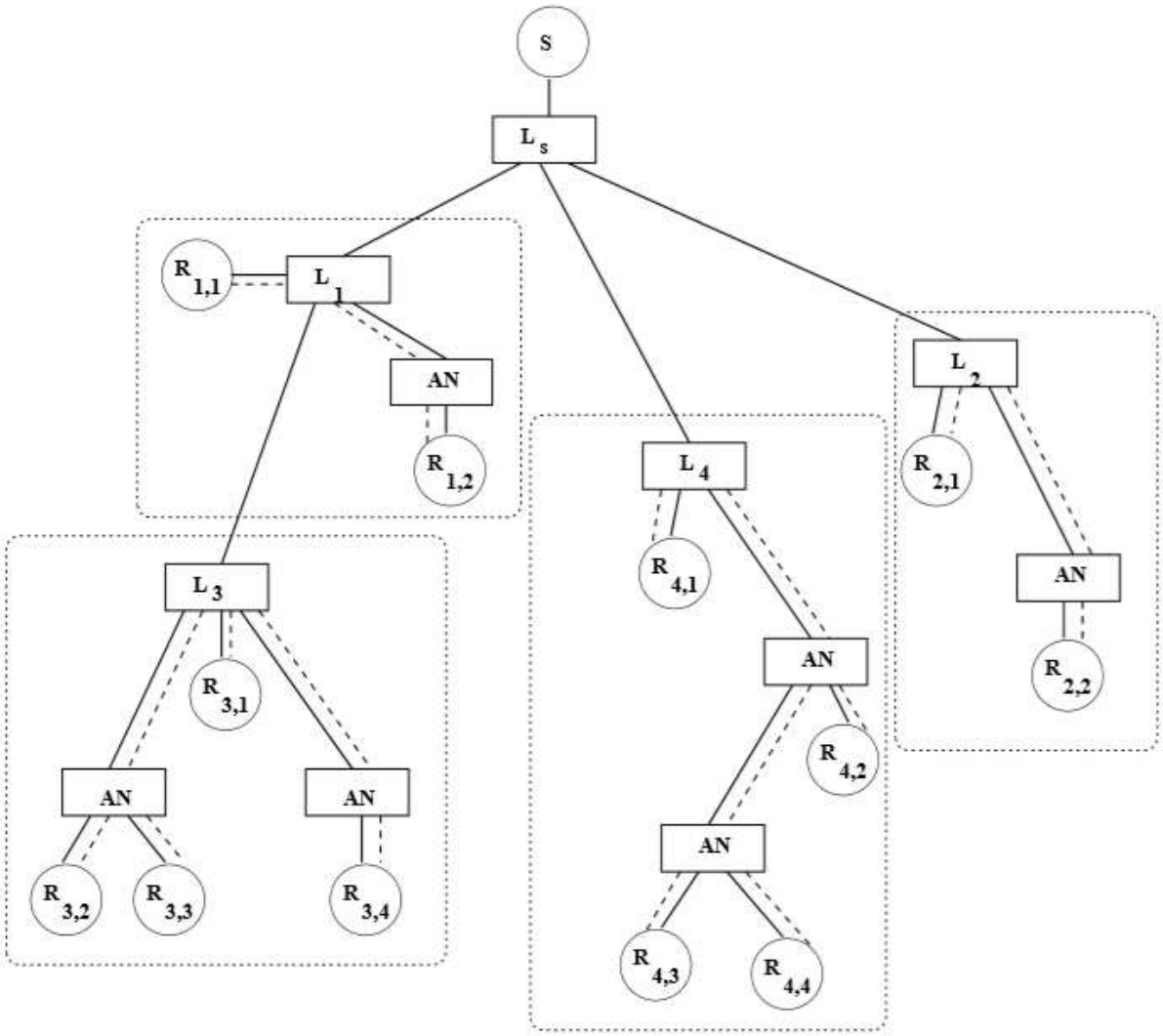


Рисунок 4 – Глобальное многоадресное дерево

2.1 Описание протокола

Надежный многоадресный транспортный протокол обеспечивает последовательную доставку без потерь от одного отправителя группе получателей. Отправитель обеспечивает надежную доставку путем выборочной повторной передачи потерянных пакетов в ответ на запрос повторной передачи от получателей.

Главной особенностью данного протокола является группировка приемников в локальные области и использование выделенных приемников (Designated Receiver) в каждой локальной области. Несмотря на то, что отправитель многоадресно передает каждый пакет всем получателям, находящимся в глобальном многоадресном дереве, только выделенные приемники отправляют сообщения отправителю с указанием, какие пакеты они получили и какие пакеты они не получили. Получатели в локальной области передают их в соответствующий DR. Стоит отметить, что DR не консолидирует сообщения о состоянии приемников в своей локальной области, но использует эти сообщения для выполнения локальных повторных передач получателям, значительно уменьшая сквозную задержку. Таким образом, отправитель видит только DR, а DR видит только получателей в своей локальной области. Обработка сообщений о состоянии доставки данных происходит исключительно между отправителем и выделенными приемниками, что позволяет избежать проблемы имплозий ACK.

На рисунке 2 приемники $R_{i,1}$ выбраны в качестве DR для групп $R_{i,j}$ в своих локальных областях. Локальные многоадресные деревья обозначены пунктирными линиями на рисунке 4.

2.2 Алгоритм работы

1. Отправитель осуществляет многоадресную передачу пакетов данных всем получателям в глобальном многоадресном дереве. Эта многоадресная передача называется глобальной многоадресной рассылкой.

2. Каждый выделенный приемник отправляет сообщение с информацией о доставке в форме пакетов состояния с определенной периодичностью. Каждый пакет содержит информацию о том какие пакеты данных были успешно приняты выделенным приемников. На основании этих сообщение отправитель определяет какие пакеты должны быть переданы повторно. Если число выделенных приемников, запрашивающих повторную передачу пакета, превышает определенный порог, повторная передача выполняется многоадресно, в противном случае приемник повторно передает пакеты только запрашивающим выделенным приемникам. Далее каждый выделенный приемник многоадресно рассылает данные в пределах своей локальной области.

3. Каждый приемник не являющейся выделенным отправляет свой статус своему RD через равные промежутки времени. Выделенный приемник осуществляет повторную локальную многоадресную передачу пакета, если число приемников в его области, запрашивающих повторную передачу, превышает пороговое значение; в противном случае пакет передается одноадресно для тех приемников, которые запросили его повторную передачу.

4. Отправитель многоадресно отправляет новые пакеты, если в окне есть свободное место.

Отправитель делит данные, подлежащие передаче, на пакеты данных фиксированного размера, за исключением последнего. Пакет данных идентифицируется типом DATA, а тип DATA EOF идентифицирует

последний пакет данных. Отправитель назначает каждому пакету данных порядковый номер, начиная с 0. Получатель периодически отправляет пакеты ACK отправителю или DR. Пакет ACK содержит нижний конец окна приема (L) и битовый вектор фиксированной длины для получения размера окна, указывающего, какие пакеты получены и какие пакеты потеряны. В таблице 1 перечислены типы пакетов, используемые в данном протоколе.

Таблица 2 - Типы пакетов

ACK	Пакет подтверждения доставки
ACK TXNOW	Пакет требования немедленной передачи
DATA	Пакет данных
DATA EOF	Последный пакет данных
RESET	Пакет для разрыва соединения
RTT MEASURE	Пакет для измерения времени приема-передачи
RTT ACK	ACK для пакета RTT MEASURE
SND ACK TOME	Пакет для выбора AP

Таблица 3 - Параметры соединения.

Wr	размер окна приема в пакетах
Ws	размер окна отправки в пакетах
Tdally	задержка после отправки последнего пакета
Tretx	задержка после отправки последнего пакета
Trtt	интервал времени для измерения времени приема-передачи
Tsap	интервал времени для отправки SND ACK TOME
Tsend	интервал времени для отправки пакетов данных
Tack	интервал времени для отправки пакетов состояния
Packet	Размер пакета данных в октетах
Cache	Размер пакета данных в октетах
MCASTthresh	порог многоадресной повторной передачи

Установка соединения

Соединение идентифицируется парой конечных точек: конечной точкой источника и конечной точкой назначения. Конечная точка источника состоит из сетевого адреса отправителя и номера порта; конечная точка назначения состоит из адреса многоадресной группы и номера порта. Каждое соединение имеет набор связанных параметров соединения (см. Таблицу 3). Протокол предполагает наличие диспетчера сеансов, который отвечает за предоставление отправителю и получателю(ям) соответствующих параметров соединения. Протокол использует значения по умолчанию для любого параметра соединения, если он не указан явно.

Как только диспетчер сеансов предоставил отправителю и получателям информацию о сеансе, получатели инициализируют блок управления соединением и остаются в неподключенном состоянии; тем временем отправитель начинает передачу данных. Получив пакет данных от отправителя, получатель переходит из неподключенного состояния в подключенное состояние. В подключенном состоянии приемники периодически генерируют ACK, поддерживая соединение активным.

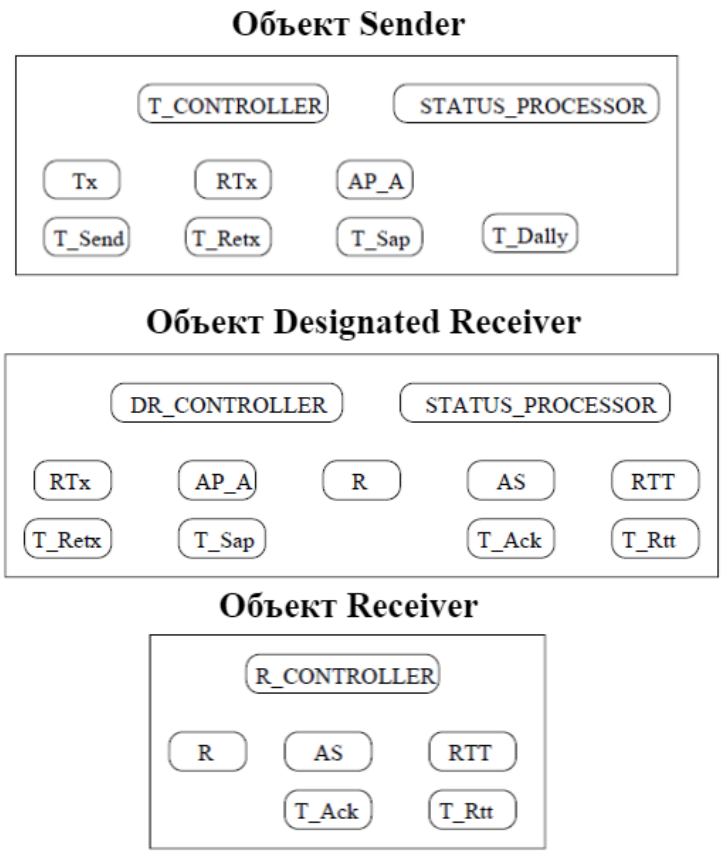
Во время работы протокола отправитель не имеет информации о получателях. Получатели могут вступать в multicast-группу или покидать ее без информирования отправителя. Следовательно, цель данного протокола - обеспечить надежную доставку текущим участникам сеанса многоадресной передачи данных. Поскольку отправитель не имеет информации о получателях, завершение сеанса основано на таймере. После того как отправитель передает последний пакет данных, запускается таймер, который истекает через T_{dally} секунд. (Выделенный приёмник также запускает таймер, после корректного получения всех пакетов данных.) После истечения таймера, отправитель удаляет всю информацию о состоянии соединения. Сообщение ACK от получателя сбрасывает таймер к начальному значению. Каждый получатель удаляет свой блок управления соединением и

прекращает отправку ACK после корректного получения всех пакетов данных. DR ведет себя как обычный получатель за исключением того, что он удаляет свой блок управления соединением только после истечения времени таймера.

Поскольку период времени между передачей последовательных ACK от получателей намного меньше, чем T_{dally}, отправитель предполагает, что либо все приемники корректно приняли весь объём информации, либо произошла внештатная ситуация. Возможные примеры внештатных ситуаций: выход из строя сетевых устройств, выход приемников из группы. Во время возникновения данных ситуаций происходит разрыв соединения путем отправки пакета RESET. Например, когда протокол определяет, что отправляющее приложение завершило работу до того, как передача данных была успешно завершена, он использует пакет RESET для того, чтобы информировать все приемники о закрытии соединения.

Объекты протокола

Надежный многоадресный транспортный протокол имеет три основных объекта: (1) Отправитель, (2) Получатель и (3) Назначенный



получатель (см. рисунок 5).

Рисунок 5 – Объекты протокола.

Объект Sender имеет метод T_CONTROLLER, который решает, должен ли отправитель передавать новые пакеты (используя метод Tx), проводить повторную передачу потерянных пакетов (используя метод RTx) или рассылать с информацией о том, что данный отправитель является обработчиком сообщений ACK (используя метод AP_A). Метод STATUS_PROCESSOR обрабатывает ACK сообщения от получателей и обновляет соответствующие структуры данных.

Также объект Sender имеет таймеры T_Sent, T_Retx и T_Sap регулирующие работу методов Tx, RTx и AP_A соответственно. Таймер T_Dally используется для разрыва соединения.

Метод R_CONTROLLER объекта Receiver отвечает за доставку данных принимающему приложению (используя метод R), за отправку сообщение ACK (используя метод AS) или за отправку пакетов измерения времени приема-передачи (используя метод RTT) для динамического вычисления круговой задержки между приемником и его обработчиком сообщений ACK.

Следует отметить, что объект Receiver имеет два таймера: T_Ack и T_Rtt, данный таймеры контролируют работу методов AS и RTT соответственно. Метод R не управляется таймером, а активируется в тот момент, когда принимающее приложение запрашивает пакеты.

Фактически обьект DR является комбинацией обьектов Receiver и Sender.

Передача данных

Отправитель многоадресно отправляет пакеты данных с интервалом определенным параметром T_{send}. Число пакетов, переданных за каждый интервал, обычно зависит от доступного пространства в окне отправки. Отправитель в течении времени T_{send} может отправить число пакетов не превышающее число W_s. Таким образом максимальную скорость передачи MaxTR можно рассчитать по формуле 1.

$$MaxTR = \frac{W_s * Packet\_Size}{T_{send}} \quad (1)$$

Подтверждение доставки

Приемники при помощи метода AS шлют пакеты ACK с определенной периодичностью, указывая статус окна приема. Приемники используют битовый массив размером в Wr бит (размер окна приема) для записи

информации о корректно принятых пакетах, хранящихся в буфере (см. рисунок 6).

Рисунок 6 – Пример окна приема.

Каждый бит соответствует одному слоту пакета в буфере приемника. Значение «1» означает что пакет принят корректно. Например, на рисунке 6 показано окно приема из восьми пакетов; пакеты 16, 17 и 19 получены правильно и хранится в буфере. Когда получатель отправляет ACK отправителю или выделенному приемнику, он включает в себя значение левого края окна приема L и битовый вектор. Следует отметить, что приемник передает пакеты приложению в последовательности. Например, если приемник получает пакет 15 от отправителя и не получает пакет 18, он может доставить пакеты 15, 16 и 17 приложению и увеличить значение L до 18.

Измерение круговой задержки и расчет T_{ack}

Приемники периодически отправляют сообщения ACK. Если данные сообщения отправляются слишком часто, обработчик сообщений ACK может прекратить повторную передачу одного и того же пакета, не зная, был ли тот принят получателями правильно. Дабы предотвратить избыточные

повторные передачи, каждый приемник динамически измеряет время приема-передачи между собой и своим обработчиком сообщений ACK, используя пакет RTT_MEASURE. На основе полученных данных, каждый приемник вычисляет T_{ack}, интервал между передачей последовательных сообщений ACK.

Приемник отправляет первый пакет RTT_MEASURE сразу после установления соединения. Последующие пакеты RTT_MEASURE отправляются с фиксированным интервалом, T_{rtt}. Чтобы измерить время приема-передачи, приемник помещает локальную временную метку в пакет RTT_MEASURE и отправляет пакет своему обработчику сообщений ACK. Когда обработчик получает пакет RTT_MEASURE, он немедленно меняет тип пакета на RTT_ACK и отправляет пакет обратно приемнику. После получения пакета RTT_ACK приемник рассчитывает время приема-передачи как разницу между временем, в которое пакет RTT_ACK был принят и временной меткой, хранящейся в нем.

Обработка ACK и повторные передачи

Отправитель или выделенный приемник обрабатывает сообщения ACK от получателей в своей локальной области. На основе полученных сообщений ACK от приемников обработчик может идентифицировать потерянные пакеты, которые следует передать повторно. Один или несколько получателей могут потерять один и тот же пакет. Обработчик сообщений ACK определяет должен ли потерянный пакет быть повторно передан с использованием одноадресной или многоадресной рассылки. Для данной цели существуют два параметра: T_{retx} и $MCAST_{tresh}$, а также очередь повторной передачи. Если сообщение ACK содержит запрос на повторную передачу, номер последовательности запрашиваемого пакета добавляется в очередь повторной передачи. Очередь повторной передачи содержит: номер последовательности, пакеты который должны быть переданы повторно, счетчик C содержащий число равное количеству приемников, которые

должны повторно получить данный пакет, таблицу адресов приемников, запрашивающих данный пакет и указатель на следующий элемент очереди. По истечении интервала T_{retx}, обработчик сообщений ACK (при помощи метода RTx) помещает элемент в очередь повторной передачи. Если значение счетчика C превышает значение $MCAST_{thresh}$, отправитель или выделенный приемник рассылает потерянные пакеты многоадресно, в противном случае, приемник, используя таблицу адресов, отсылает пакеты одноадресно.

Отправитель использует три переменные: swin_lb, send_next, и avail_win для управления окном отправки. Переменная swin_lb хранит нижнюю границу окна отправки, send_next указывает на следующий номер последовательности, который будет использоваться во время отправки пакетов данных, avail_win это доступный размер окна для отправки данных. Отправитель увеличивает значение send_next и уменьшает значение avail_win после отправки данных. Когда сообщения ACK, подтверждающие получение пакетов с номером последовательности равным swin_lb, принимаются, swin_lb увеличивается так же, как и avail_win. Иллюстрацию работы механизма окна отправки можно наблюдать на рисунке 7.

Рисунок 7 – пример окна отправки.

Чтобы определить, сколько новых пакетов должно быть передано в следующем интервале отправки, отправитель вычисляет наименьшее значение L (L_{min}) среди значений L, принятых в сообщениях ACK в промежуток времени T_{send}. Если L_{min} больше, чем swin_lb, значение avail_win увеличивается на число равное L_{min} – swin_lb и swin_lb становится равно L_{min}. Значение swin_lb больше не уменьшается. Если приемник отправляет сообщения ACK со значением L, меньшим чем swin_lb, данные сообщения игнорируются.

Позднее подключение приемников

Надежный многоадресный транспортный протокол позволяет получателям присоединяться в любой момент текущего сеанса. Приемнику, который присоединяется после начала передачи данных, требуется передать те пакеты, которые он не получил. Кроме того, некоторые приемники могут «отставать» из-за различных причин, таких как перегрузка сети. Существует две функции, которые, работая в совокупности, позволяют «отставшим» приемникам получить недостающие данные:

— Запрос немедленной передачи.

— Кэширование данных в отправителе и в выделенных приемниках.

Запрос немедленной передачи. Когда приемник присоединяется поздно, он начинает получать многоадресные пакеты от отправителя, и, просматривая номер последовательности данных пакетов, он может сразу же обнаружить, что пропустил более ранние пакеты. В этот момент приемник отправляет пакет ACK_TXNOW, чтобы запросить у своего выделенного приемника или отправителя немедленную передачу предыдущих пакетов.

Пакет ACK_TXNOW отличается от пакета ACK только полем «тип пакета». Когда обработчик сообщений ACK принимает пакет ACK_TXNOW от приемника он проверяет битовый массив V и немедленно передает

пропущенные пакеты или пакет приемнику, используя одноадресную передачу.

Кэширование данных. Для обеспечения работоспособности вышеописанной функции отправитель и выделенные приемники обязаны буферизировать файл в течении всей сессии многоадресной передачи данных. Это позволяет приемникам запрашивать повторную передачу любых ранее отправленных данных у своих обработчиков сообщений ACK.

Выбор назначенных приемников и формирование локальных областей

Протокол предполагает, что имеется некоторая информация о приблизительном местонахождении получателей, и на основе этой информации определенные сервера или приемники с наименьшим ip-адресом в подсети выбираются в качестве назначенных приемников (DR).

Далее каждый DR и отправитель периодически отправляют пакет SEND_ACK_TOME приемникам, в котором для поля TTL задано заранее определенное значение. Приемник выбирает своего DR по пакету SEND_ACK_TOME с наименьшим значением TTL. Таким образом локальные области формируются вокруг каждого DR.

2.3 Вывод

В данной главе был разработан алгоритм работы протокола надежной многоадресной передачи данных, соответствующий требованиям, определенным в главе 1. Были описаны основные механизмы работы протокола.

3 Реализация протокола

3.1 Описание виртуального стенда

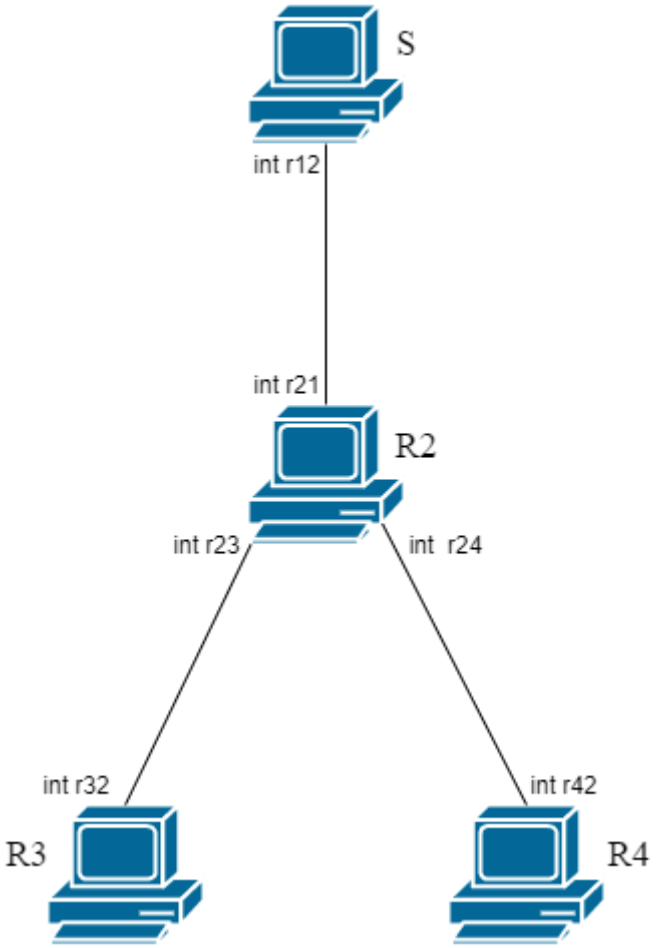


Рисунок 8 – Модель виртуального стенда

Виртуальный стенд представляет из себя набор из четырех узлов, соединенных так, как показано на рисунке 8. Каждый узел является виртуальной машиной с операционной системой Ubuntu 12.04.5 LTS. Узлы S, R3, R4 имеют по одному сетевому интерфейсу (см. рисунок 9).

```
r32       Link encap:Ethernet  HWaddr 16:15:76:c5:84:68
          inet6 addr: fe80::1415:76ff:fec5:8468/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:22 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1876 (1.8 KB)  TX bytes:1200 (1.2 KB)
```

Рисунок 9 – Информация о состоянии интерфейса r32 на узле R3

В качестве сети используется коммутируемый Ethernet. Узел R2 имеет три сетевых интерфейса объединённых в сетевой мост при помощи утилиты iproute2. Это необходимо т.к. R2 в данной модели является выделенным приемником и ему необходимо широковещательно передавать пакеты узлам R3 и R4.

```
my_bridge Link encap:Ethernet  HWaddr 4e:24:73:dc:7e:8b
          inet6 addr: fe80::4c24:73ff:fedc:7e8b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:1296 (1.2 KB)

r21       Link encap:Ethernet  HWaddr b6:ed:48:0a:6c:60
          inet6 addr: fe80::b4ed:48ff:fe0a:6c60/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16096 errors:0 dropped:1 overruns:0 frame:0
          TX packets:2736 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16454262 (16.4 MB)  TX bytes:48193 (48.1 KB)

r23       Link encap:Ethernet  HWaddr 56:8a:03:19:1a:3d
          inet6 addr: fe80::548a:3ff:fe19:1a3d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:800 (800.0 B)  TX bytes:1296 (1.2 KB)

r24       Link encap:Ethernet  HWaddr 4e:24:73:dc:7e:8b
          inet6 addr: fe80::4c24:73ff:fedc:7e8b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:648 (648.0 B)  TX bytes:1296 (1.2 KB)
```

Рисунок 9 – Информация о состоянии интерфейсов на узле R2

Так как реализация протокола выполнена на языке Python, на всех узлах установлен Python версии 2.3.7. Виртуальные машины работают в среде VMWare ESXi. Для связи используются виртуальные каналы GigabitEthernet.

3.2 Описание эксперимента

Реализация протокола была выполнена на языке Python. Для проверки работоспособности реализации проведен описанный ниже эксперимент. Для использования в реализации был выбран такой интерфейс программирования приложений как сырой сокет (англ. raw socket), эти сокеты позволяют реализовать в пользовательском пространстве новые протоколы стека Ipv4 и не выставляют каких-либо ограничений в формировании пакета на отправку.

На всех узлах виртуального стенда инициализируется запуск программы из файла main.py. В зависимости от аргумента запуска узел работает в одном из трех режимов:

1. Отправитель (аргумент «-s»);
2. Приемник (аргумент «-r»);
3. Выделенный приемник (аргумент «-d»).

```
if __name__ == '__main__':
    interfaces = os.popen("ip a | egrep -o 'r[0-9]+'").read().split('\n')
    print(interfaces)
    interfaces.pop()

    sockets = [None] * len(interfaces)
    for i in range(0, len(interfaces)):
        sockets[i] = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(3))
        sockets[i].bind((interfaces[i], 0))
    print('Switching on ' + ' '.join(interfaces))

    if len(sys.argv) < 2:
        print("Need some arguments")
        sys.exit(1)
    if sys.argv[1] == "-s":
```

```
        filename = sys.argv[2]
        sender(sockets[0], filename)

    elif sys.argv[1] == "-r":
        if len(sys.argv) <= 2:
            print("Need input rd address second argument")
        else:
            print(sys.argv[2])
            receiver(sockets[0], sys.argv[2])

    elif sys.argv[1] == "-d":
        bridge_socket = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(3))
        bridge_socket.bind(('my_bridge', 0))
        receiver_d(bridge_socket)
```

В качестве теста работы алгоритма производится передача файла от отправителя S группе приемников R2-R4. В рамках данного эксперимента выделенным приемником является узел R2. На данном узле выполняется основной механизм, гарантирующий надежную доставку, а именно осуществляются агрегация подтверждений доставки от остальных приемников и повторная передача, в случае потери сегментов данных.

```
def receiver_d(socket_in):
    f = open("received.py", 'wb')
    sender_addr = ""
    src_addr = socket_in.getsockname()[4]
    print "My address :", bytes_addr_to_str(src_addr)
    local_receivers = [src_addr]
    num_receive_frame = 0
    num_err_frame = 0
    max_err_frame = 1
    while True:
        print "Wait frame..."
        raw_data, addr = socket_in.recvfrom(MAX_SIZE)
        dst_addr, src_addr, proto, frame_type, num_packet, is_last, data = parse_frame(raw_data)
        print('\nReceived Frame:')
        print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
        Packet: {}'.format(
            bytes_addr_to_str(dst_addr), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
            frame_type, num_packet, is_last))
```

```
if frame_type == TYPE_MSG_RD:
    print "New receiver in local area:", bytes_addr_to_str(src_addr)
    local_receivers.append(src_addr)
    max_err_frame = int(len(local_receivers) / 2)
    if max_err_frame == 0:
        max_err_frame = 1
    continue
if frame_type == TYPE_MSG_D:
    sender_addr = src_addr
    num_receive_frame += 1
    if data:
        f.write(data)
if frame_type == TYPE_MSG_ANSWER:
    num_receive_frame += 1
if frame_type == TYPE_MSG_ERR:
    num_err_frame += 1
if num_err_frame >= max_err_frame:
    num_err_frame = 0
    dst_addr = sender_addr
    src_addr = socket_in.getsockname()[4]
    print('\nSending Frame:')
    print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
    Packet: {}'.format(
        bytes_addr_to_str(dst_addr), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
       TYPE_MSG_REPEAT,
        num_packet, is_last))
    answer_frame = struct.pack(HEADER_FORMAT, dst_addr, src_addr, proto,
    TYPE_MSG_REPEAT, num_packet, is_last)
    socket_in.sendall(answer_frame)
    if is_last:
        break
if num_receive_frame == len(local_receivers):
    num_receive_frame = 0
    dst_addr = sender_addr
    src_addr = socket_in.getsockname()[4]
    print('\nSending Frame:')
    print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
    Packet: {}'.format(
```

```
            bytes_addr_to_str(dst_addr), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
            TYPE_MSG_ANSWER, num_packet, is_last))
        answer_frame = struct.pack(HEADER_FORMAT, dst_addr, src_addr, proto,
        TYPE_MSG_ANSWER, num_packet, is_last)
        socket_in.sendall(answer_frame)
        if is_last:
            break
    f.close()
    print('\nFile Downloaded')
```

По мимо всего прочего узел R2, так же, как и узлы R3 и R4 принимает пакеты данных от отправителя R1, отсылая ему сообщения о подтверждении доставки. В свою очередь узлы R3 и R4 работают в режиме обычного приемника, отсылая сообщения о успешной доставке своему выделенному приемнику R2.

```
def receiver(socket_in, rd_address):
    f = open("received.py", 'wb')
    is_last = False
    rd_address = str_addr_to_bytes(rd_address)
    src_addr = socket_in.getsockname()[4]
    proto = b'\x89\x99'
    print "My address :", bytes_addr_to_str(src_addr)
    print "RD address :", bytes_addr_to_str(rd_address)
    rd_frame = struct.pack(HEADER_FORMAT, rd_address, src_addr, proto, TYPE_MSG_RD, 0,
    False)
    socket_in.sendall(rd_frame)
    print('\nSending Frame to RD:')
    print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
Packet:
    {}'.format(
        bytes_addr_to_str(rd_address), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
        TYPE_MSG_RD, 0, is_last))
    while not is_last:
        raw_data, addr = socket_in.recvfrom(MAX_SIZE)
        dst_addr, src_addr, proto, frame_type, num_packet, is_last, data = parse_frame(raw_data)
        print('\nReceived Frame:')
```

```
        print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
        Packet: {}'.format(
            bytes_addr_to_str(dst_addr), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
            frame_type, num_packet, is_last))
        if frame_type != TYPE_MSG_D:
            continue
        if data:
            f.write(data)
        dst_addr = rd_address
        src_addr = socket_in.getsockname()[4]
        answer_frame = struct.pack(HEADER_FORMAT, dst_addr, src_addr, proto,
        TYPE_MSG_ANSWER, num_packet, is_last)
        socket_in.sendall(answer_frame)
        print('\nSending Frame:')
        print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
        Packet: {}'.format(
            bytes_addr_to_str(dst_addr), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
            TYPE_MSG_ANSWER, num_packet, is_last))
    f.close()
    print('\nFile Downloaded')
```

В рамках текущей реализации выделенный приемник выбирается вручную, поэтому узлы R3, R4 заранее обладая этой информацией, отправляют узлу R2 сообщения о том, что они находятся в его локальной области, R2 в свою очередь регистрирует их и начинает обрабатывать сообщения о подтверждении доставки.

Узел S работает в режиме отправителя, на данном узле происходит разбиение отправляемого файла на сегменты данных и последующая широковещательная отправка данных сегментов получателям, также узел получает сообщения о подтверждении доставки и в случае потери пакетов данных производит повторную передачу.

```
def sender(socket_out, filename):
    f = open(filename, "rb")
    data_size = MAX_SIZE - HEADER_SIZE
    src_addr = socket_out.getsockname()[4]
    print "My address :", bytes_addr_to_str(src_addr)
```

```
    dst_addr = b'\xff\xff\xff\xff\xff\xff' # FF:FF:FF:FF:FF:FF
    proto = b'\x89\x99'
    print("\nStart file sending...")
    data_parts = []
    data = f.read(data_size)
    while data != "":
        data_parts.append(data)
        data = f.read(data_size)
    if len(data_parts) == 0:
        print("len(data_parts) == 0")
        print("Stop file sending...")
        return
    num_packet = 0
    data = data_parts[num_packet]
    # Send first part to all clients
    header = struct.pack(HEADER_FORMAT, dst_addr, src_addr, proto, TYPE_MSG_D,
                         num_packet, num_packet == len(data_parts) - 1)
    socket_out.sendall(header + data)
    print('\nSending Frame:')
    print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
Packet:
  {}'.format(
        bytes_addr_to_str(dst_addr), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
        TYPE_MSG_D, num_packet, num_packet == len(data_parts) - 1))
    while True:
        raw_data, addr = socket_out.recvfrom(MAX_SIZE)
        dst_addr, src_addr, proto, frame_type, num_packet, is_last, data = parse_frame(raw_data)
        print('\nReceived Frame:')
        print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
        Packet: {}'.format(
            bytes_addr_to_str(dst_addr), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
            frame_type, num_packet, is_last))
        if frame_type != TYPE_MSG_REPEAT:
            num_packet += 1
        if num_packet >=len(data_parts):
            break
        data = data_parts[num_packet]
        dst_addr = b'\xff\xff\xff\xff\xff\xff'
        src_addr = socket_out.getsockname()[4]
        header = struct.pack(HEADER_FORMAT, dst_addr, src_addr, proto, TYPE_MSG_D,
                             num_packet, num_packet == len(data_parts) - 1)
        socket_out.sendall(header + data)
        print('\nSending Frame:')
        print('Destination:\t{} \nSource:\t\t{} \nProto: {} \nType: {} \nNum Frame: {} \nIs Last
        Packet: {}'.format(
            bytes_addr_to_str(dst_addr), bytes_addr_to_str(src_addr), bytes_addr_to_str(proto),
            TYPE_MSG_D, num_packet, num_packet == len(data_parts) - 1))
    f.close()
    print('\nFile Sent')
```

3.3 Результат эксперимента

Перед началом передачи файла узлы R3 и R4 отправили сообщения узлу R2, зарегистрировавшись в его локальной зоне (см. рисунок 10).

```
r2@switch:~$ sudo python main.py -d
['r21', 'r23', 'r24', '']
Switching on r21 r23 r24
My address : 4e 24 73 dc 7e 8b
Wait frame...

Received Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         16 15 76 c5 84 68
Proto: 89 99
Type: 3
Num Frame: 0
Is Last Packet: False
New receiver in local area: 16 15 76 c5 84 68
Wait frame...

Received Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         be 66 6e b6 ce d8
Proto: 89 99
Type: 3
Num Frame: 0
Is Last Packet: False
New receiver in local area: be 66 6e b6 ce d8
Wait frame...
```

Рисунок 10 – Получение регистрационных пакетов от узлов R3 и R4

После успешной регистрации узел S инициировал отправку пакета данных отправителям. Узел R2 производил пересылку полученного пакета приемникам R3 и R4, агрегируя полученные от них сообщения о доставке и в случае успешной передачи отправлял сообщение об успешной передаче отправителя, и запрашивал отправку следующего сегмента (см. рисунок 11, 12)

```
r1@switch:~$ sudo python main.py -s file
['r12', '']
Switching on r12
My address : 32 52 fe 32 75 d0

Start file sending...

Sending Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 0
Is Last Packet: False

Received Frame:
Destination:    32 52 fe 32 75 d0
Source:         4e 24 73 dc 7e 8b
Proto: 89 99
Type: 2
Num Frame: 0
Is Last Packet: False

Sending Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 1
Is Last Packet: False

Received Frame:
Destination:    32 52 fe 32 75 d0
Source:         4e 24 73 dc 7e 8b
Proto: 89 99
Type: 2
Num Frame: 1
Is Last Packet: False

Sending Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 2
Is Last Packet: True

Received Frame:
Destination:    32 52 fe 32 75 d0
Source:         4e 24 73 dc 7e 8b
Proto: 89 99
Type: 2
Num Frame: 2
Is Last Packet: True

File Sent
r1@switch:~$
```

Рисунок 11 - Вывод работы протокола с узла R1

```
Received Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 0
Is Last Packet: False
Wait frame...

Received Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         be 66 6e b6 ce d8
Proto: 89 99
Type: 2
Num Frame: 0
Is Last Packet: False
Wait frame...

Received Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         16 15 76 c5 84 68
Proto: 89 99
Type: 2
Num Frame: 0
Is Last Packet: False

Sending Frame:
Destination:    32 52 fe 32 75 d0
Source:         4e 24 73 dc 7e 8b
Proto: 89 99
Type: 2
Num Frame: 0
Is Last Packet: False
Wait frame...
```

Рисунок 12 – Фрагмент вывода работы протокола с узла R2

```
r3@switch:~$ sudo python main.py -r 4e:24:73:dc:7e:8b
['r32', '']
Switching on r32
4e:24:73:dc:7e:8b
My address : 16 15 76 c5 84 68
RD address : 4e 24 73 dc 7e 8b

Sending Frame to RD:
Destination:    4e 24 73 dc 7e 8b
Source:         16 15 76 c5 84 68
Proto: 89 99
Type: 3
Num Frame: 0
Is Last Packet: False

Received Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 0
Is Last Packet: False

Sending Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         16 15 76 c5 84 68
Proto: 89 99
Type: 2
Num Frame: 0
Is Last Packet: False

Received Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 1
Is Last Packet: False

Sending Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         16 15 76 c5 84 68
Proto: 89 99
Type: 2
Num Frame: 1
Is Last Packet: False

Received Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 2
Is Last Packet: True

Sending Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         16 15 76 c5 84 68
Proto: 89 99
Type: 2
Num Frame: 2
Is Last Packet: True

File Downloaded
r3@switch:~$
```

Рисунок 13 - Вывод работы протокола с узла R3

```
r4@switch:~$ sudo python main.py -r 4e:24:73:dc:7e:8b
['r42', '']
Switching on r42
4e:24:73:dc:7e:8b
My address : be 66 6e b6 ce d8
RD address : 4e 24 73 dc 7e 8b

Sending Frame to RD:
Destination:    4e 24 73 dc 7e 8b
Source:         be 66 6e b6 ce d8
Proto: 89 99
Type: 3
Num Frame: 0
Is Last Packet: False

Received Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 0
Is Last Packet: False

Sending Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         be 66 6e b6 ce d8
Proto: 89 99
Type: 2
Num Frame: 0
Is Last Packet: False

Received Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 1
Is Last Packet: False

Sending Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         be 66 6e b6 ce d8
Proto: 89 99
Type: 2
Num Frame: 1
Is Last Packet: False

Received Frame:
Destination:    ff ff ff ff ff ff
Source:         32 52 fe 32 75 d0
Proto: 89 99
Type: 1
Num Frame: 2
Is Last Packet: True

Sending Frame:
Destination:    4e 24 73 dc 7e 8b
Source:         be 66 6e b6 ce d8
Proto: 89 99
Type: 2
Num Frame: 2
Is Last Packet: True

File Downloaded
r4@switch:~$
```

Рисунок 14 - Вывод работы протокола с узла R4

В результате эксперимента от отправителя S был успешно передан файл, размером 3 килобайта узлам R2, R3 и R4. Ниже приведена таблица, отображающая количество пакетов, отправленных и принятых каждым узлом для передачи данного файла в сравнении с протоколом TCP.

Таблица 4 - Количество пакетов, отправленных и принятых каждым узлом

	TCP		Реализованный протокол	
Пакеты / Узел	Отправлено	Получено	Отправлено	Получено
S	15	12	3	3
R2	4	5	3	11
R3	4	5	4	3
R4	4	5	4	3

Исходя из этих данных можно сделать вывод что при использовании TCP количество трафика растет линейно в зависимости от числа приемников. При использовании надежного многоадресного транспортного протокола нагрузка на сеть распределяется равномерно, и количество трафика в сети сокращается и зависит от количества локальных зон и количества приемников в каждой локальной зоне.

3.4 Выводы

В данной главе был описан виртуальный стенд, на котором происходит проверка работы демонстрационной реализации описанного во второй главе алгоритма работы протокола. В результате анализа данных, полученных в ходе эксперимента, подтверждено функционирование основных механизмов протокола и проведено сравнение работы текущей реализации с работой протокола TCP.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были исследованы основные подходы к управлению многоадресной передачей информации, ограничения области применения надежных многоадресных протоколов, накладываемые приложениями и сетью. Также был проведен обзор и сравнительный анализ уже существующих решений в области RM протоколов, таких как протоколы PGM и M/TCP. На основе данного анализа был выявлен круг задач, которые не могут решить вышеперечисленные протоколы и определены требования предъявляемые к новому RM протоколу. Был разработан алгоритм нового протокола, соответствующий вышеуказанным критериям. На языке Python была создана тестовая реализация данного алгоритма и развернута на лабораторном стенде. В качестве проверки работы производилась отправка файла от отправителя группе получателей. В будущем планируется доработка реализации, добавление в нее механизмов, обеспечивающих наиболее стабильную работу алгоритма протокола.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Косюга О.С., Коляда В.В. Применение архитектур клиент сервер и файл сервер в информационных системах // Сборник материалов XI международного студенческого форума: «ИНФОРМАЦИОННОЕ ОБЩЕСТВО: СОВРЕМЕННОЕ СОСТОЯНИЕ И ПЕРСПЕКТИВЫ РАЗВИТИЯ» – Краснодар: КГУ, 2018. – 52-55 с.

2. Mehmet Ulema, Bin Wu. Next generation service overlay networks // IEEE Communications Magazine 50(1):52-53 · January 2012

3. Шитько А.М., Напей Н. В. Использование протокола peer-to-peer для защищенного обмена данными // Журнал: «Труды БГТУ. Серия 3: Физико-математические науки и информатика» – Москва: БГТУ, 2015. – 162-165 с.

4. Величко И.А. Многоадресная передача данных // Журнал: «НАУКА, ТЕХНИКА И ОБРАЗОВАНИЕ» – Иваново: КГУ, 2017. – 82-85 с.

5. Братчикова О.М. Преимущество многоадресной технологии IP Multicast по сравнению с традиционной технологией IP-Адресации (Unicast) // Сборник статей международной научно-практической конференции: «Актуальные проблемы авиации и космонавтики» – Красноярск: СибГАУ, 2015. – 525-526 с.

6. Красов А.В., Сахаров Д.В. Обеспечение безопасности передачи multicast-трафика в ip-сетях // Журнал: «Защита информации. Инсайд» – Санкт-Петербург: СПбГУТ, 2017. – 34-42 с.

7. M. Handley, S. Floyd. The Reliable Multicast Design Space for Bulk Data Transfer // Digital Fountain, Inc. August 2000

8. B. Whetten, L. Vicisano. Reliable Multicast Transport Building Blocks for One-to-Many Bulk-Data Transfer // Digital Fountain, Inc. January 2001

9. Giacomo Morabito, Sergio Palazzo. Modeling and Analysis of TCP-Like Multicast Congestion Control in Hybrid Terrestrial/Satellite IP Networks // IEEE Journal on Selected Areas in Communications 22(2):401-412 · February 2004

10. Vasaka Visoottiviseth, Takuya Mogami. M/TCP: The Multicast-extension to Transmission Control Protocol // The Graduate School of Information Science, Nara Institute of Science and Technology, Japan 2015.

11. Jim Gemmell, Todd Montgomery. The PGM Reliable Multicast Protocol // IEEE Journal on Selected Areas in Communications 17(1):16-22 · January 2003

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

Институт космических и информационных технологий
институт

Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____О.В. Непомнящий
подпись
«____»_____2020 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Методы управления многоадресной передачей данных
тема

09.04.01 – Информатика и вычислительная техника
код и наименование направления

09.04.01.05–Сети ЭВМ и телекоммуникации
код и наименование магистерской программы

Научный руководитель	_____ *подпись, дата*	канд. физ.-мат. наук	К.В. Коршун *инициалы, фамилия*
Выпускник	_____ *подпись, дата*		В.И. Туркевич *инициалы, фамилия*
Рецензент	_____ *подпись, дата*	директор ЗАО «Интертакс»	М.В. Алексеев *инициалы, фамилия*
Нормоконтролер	_____ *подпись, дата*	канд. физ.-мат. наук	К.В. Коршун *инициалы, фамилия*

Красноярск 2020