Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет инфокоммуникаций

Кафедра инфокоммуникационных технологий

Дисциплина: Системы коммутации каналов и пакетов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

**Разработка приложения «Студенты»**

Студент гр.963101 Д.А. Коржовник

Руководитель Н.Н. Ловчий

Минск 2021

**СОДЕРЖАНИЕ**

В[ведение 5](#_Toc86937424)

[1.](#_Toc86937425) Удаленная база данных Firebase [7](#_Toc86937425)

[1.1 Краткое сведение Firebase 7](#_Toc86937426)

[1.2 Настройка Firebase для среды разработки Android Studio 8](#_Toc86937427)

[2 Приложения «Студенты» 10](#_Toc86937428)

[2.1 Концепция приложения 12](#_Toc86937429)

[2.2 Описание работы приложения 12](#_Toc86937429)

[Заключение 28](#_Toc39696851)

[Приложение А (обязательное) Блок-схема алгоритма работы приложения 30](#_Toc39696853)

[Приложение Б (обязательное) Листинг исходного кода приложения 32](#_Toc39696855)

# **ВВЕДЕНИЕ**

Компьютерной сетью называют совокупность узлов (терминалов, компьютеров, периферийных устройств), имеющих возможность информационного взаимодействия друг с другом с помощью специального коммуникационного оборудования и программного обеспечения. Размеры таких сетей варьируются в широких пределах – от пары соединенных между собой компьютеров, стоящих в соседних помещениях, до миллионов компьютеров, разбросанных по всему миру (часть которых может находиться и на космических объектах). По широте охвата принято деление сете на несколько категорий.

1. Local-Area Network (Локальные вычислительные сите).

2. Campus-Area Network (кампусная сеть).

3. Metropolitan-Area Network (сеть городского масштаба).

4. Wide-Area Network (широкомасштабная сеть).

5. Global-Area Network (глобальная сеть).

Локальные вычислительные сети позволяют объединять компьютеры, расположенные в ограниченном пространстве. Для локальных сетей, как правило, прокладываться специализированная кабельная система, и положение возможных точек подключения абонентов, ограниченного этой кабельной системой.

Локальные сети можно объединить в более крупномасштабные образования – CAN (объединяющая локальные сети близко расположенных зданий), MAN, WAN, GAN.

В сетях применяться различные сетевые технологии, из которых в локальных сетях наиболее распространены Ethernet, Token Ring, 100VG-AnyLAN, ARCnct, FDDI.

В любой сети применяется тот или иной метод управления обменом (он же метод доступа, он же метод арбитража), разрешающий или предотвращающий конфликты между абонентами. От эффективно выбранного метода зависит очень многое: скорость обмена информацией между компьютерами, нагрузочная способность сети, время реакции сети на внешние события и так далее. Метод управления – это один из важнейших параметров сети. Тим метода управления обменом во многом определяется особенностями топологии сети, но в то же время он и не привязан жестко к топологии.

Методы доступа к среде передачи делятся на вероятностные и детерминированные.

При вероятностном (probabilistic) методе доступа узел, желающий послать кадр в сеть, прослушивает линию. Если линия занята или обнаружена коллизия (столкновение сигналов от двух передатчиков), попытка передачи откладывается на некоторое время. Основные разновидности:

1.#CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) – множественный доступ с прослушивание несущей и избеганием коллизий.

2.#CSMA/CD (Carrier Sense Multiple Access/Collision Detect) – множественный доступ с прослушиванием несущей и обнаружением коллизий.

Сети с большой нагрузкой требуют более эффективные методы доступа. Один из способов повышения эффективности – перенос управления доступом от узлов в кабельный центр. При этом узел посылает кадр в коммуникационное устройство. Задача этого устройства – обеспечить прохождение кадра к адресату с оптимизацией общей производительности сети и обеспечением уровня качества обслуживания, требуемого конкретным приложением.

Целью курсовой работы является изучение принципов функционирования коммутируемых и виртуальных сетей, подробное рассмотрение сетей Ethernet и Fast Ethernet, принципов коммутации сегментов и узлов сети, а также виртуальные локальные сети, возможное применение коммутаторов.

**1.СЕТИ ETHERNET И FAST** [**Enternet**7](#_Toc86937425)

## **1.1Возникновение сети и описание**

За время, прошедшее с появлением первой локальной сетей, было разработано несколько сотен самых разнообразных сетей разных сетевых технологий, однако заметное распространение получили всего несколько сетей, что связано, прежде всего, с поддержкой этих сетей известными фирмами и с высоким уровнем стандартизации принципов их организации. Далеко не всегда стандартные сети имеют рекордные характеристики, обеспечивающие наиболее оптимальные режимы обмена, но большее объемы выпуска их аппаратуры и, следовательно, ее невысокая стоимость обеспечивают им огромное преимущество. Немаловажно и то, что производители программных средств также в первую очередь ориентируются на самые распространенные сети. Поэтому пользователь, выбирающий стандартные сети, имеет полную гарантии совместимости аппарата и программ.

Наибольшее распространение среди стандартных сетей получила сеть Ethernet. Принято считать, что Ethernet был изобретен 22 мая 1973 года, когда Роберт Меткалф составил докладную записку для главы PARC о потенциале технологии Ethernet. Однако законное право на технологи было получить только через несколько лет. В 1976 году он и его ассистент Девил Боггс издали брошюру «Ethernet: Distributed Packet-Switching For Local Computer Networks». Меткалф ушел из Xerox в 1979 году и основал компанию. 3Com для продвижения компьютеров и локальных вычислительных сетей. Ему удалось убедить DEC, Intel и Xerox работать совместно и разработать стандарт Ethernet (DIX).

Впервые этот стандарт был опубликован 30 сентября 1980 года. Он начал соперничество с двумя крупными запатентованными технологиями: token ring и ARCNET, – которые вскоре были раздавлены под накатывающимися волнами продукции Ethernet. Стандарт получил название IEEE 802.3. Он определяет множественный доступ к моноканалу типа «шина» с обнаружение конфликтов и контролем передачи, то есть с методом доступа CSNA/CD. Метод CSNA/CD, метод доступа к среде передачи данных, называемый метод коллективного доступа с опознаванием несущей и обнаружением коллизий. Этому стандарту удовлетворяют и некоторые другие сети, а также все виды стандартов Ethernet, так как он не очень сильно детализирован. В результате сети стандарта IEEE 802.3 нередко не совместимы между собой как по конструктивным, так и по электрическим характеристикам. Для передачи двоичной информации по кабелю для всех вариантов физического уровня технологии Ethernet обеспечивающих пропускную способность 10 Мбит/с, используется манчестерский код.

## **1.2 СТАНДАРТЫ ETHERNET И ИХ ХАРАКТЕРИСТИКА**

В исходной версии Ethernet предусматривалось применение коаксиального кабеля двух видов – «толстого» и «тонкого» (стандарты 10Base-5 и 10Base-2, соответственно). Однако в начале 90-ч годов также появились спецификации для построения сетей Ethernet с использованием витой пары (10Base-T) и оптоволокна(10Base-FL). Позже в 1995 был опубликован стандарт архитектуры Fast Ethernet (IEEE 802.3z и 802.3ab), а в 2002 году стандарт 10 Gigabit Ethernet (IEEE 802.3 ae).

Сравнение различных характеристик Ethernet приведено в таблице 1.

Таблица 1- Характеристики различных стандартов Ethernet

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Реализация | Скорость передачи данных, Мбит/c | Топология | Среда передачи | Максимальная длина кабеля, м |
| Ethernet |  |  |  |  |
| 10Base-5 | 10 | «шина» | Толстый коаксиальный кабель | 500 |
| 10Base-2 | 10 | «шина» | Тонкий коаксиальный кабель | 185; реально до 300 |
| 10Base-T | 10 | «звезда» | Вытая пара | 100 |
| 10Base-FL | 10 | «звезда» | оптоволокно | 500(станция-концентратор);  2000(между концентраторами) |
| Fast Ethernet |  |  |  |  |
| 100Base-TX | 100 | «звезда» | Витая пара категории 5 (используется две пары) | 100 |
| 100Bade-T4 | 100 | «звезда» | Витая пар категории 3, 4 или 5 (используется четыре пары) | 100 |

Продолжение таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100Base-FX | 100 | «звезда» | Многомодовое или  одномодовое оптоволокно | 2000  (многомодовое);  15000 (одномодовое);  Реально до 40 км |
| Gigabit Ethernet |  |  |  |  |
| 1000Base-T | 1000 | «звезда» | Витая пара категории 5 и выше | 100 |
| 1000Base-CX | 1000 | «звезда» | Специальный кабель типа STP | 25 |
| 1000Base-SX | 1000 | «звезда» | Оптоволокно | 220-550 (многомодовый), в зависимости от типа |
| 1000Base-LX | 1000 | «звезда» | оптоволокно | 550 (многомодовый);  5000 (одномодовый);  Реально до 80 км |
| 10Gigabit Ethernet |  |  |  |  |
| 10GBase-x  (x–набор стандартов) | 10000 | «звезда» | оптоволокно | 300-40000(в зависимости от типа кабеля и длинны волны лазера) |

В современных версиях Ethernet использование физической топологии «шина» уже не предусмотрена, а также сети, построенные на коаксиальных кабелях, практически отсутствуют.

Сеть Ethernet наиболее популярна в мире (более 80 % рынка). Этому поспособствовало то, что с самого начала все характеристики, параметры, протоколы сети были открыты для всех, в результате чего огромное число производителей во всем мире стало выпускать аппаратуру Ethernet, полностью совместимою между собой.

**2 ПЕРЕДАЧА ИНФОРМАЦИИ В СЕТИ**

**2.1 Назначение пакетов и их структура**

Информация в локальных сетях как правило передается отдельными порциями кусками, называемыми в различных источниках пакетами кадрами блоками. Использование пакетов связано с тем, что в сети как правило одновременно можно производить несколько сеансов связи (при топологии «шина» и «кольцо») то есть в течении одного и того же времени могут идти два и более процесса передачи данных между различными парами абонентов. Пакеты позволяют разделить во времени сеть между передающими информацию абонентами уравнять в правах всех абонентов, а также примерно уравнять время доступа к сети и интегральную скорость передачи информации для всех абонентов.

Важно также и то, что при передачи больших массивов информации становиться довольно высокая вероятность ошибки из-за помех и сбоев. Например, при характерной для локальной сети величине вероятность одиночной ошибки в 10~8 пакетах длиной 10 Кбит будет искажен с вероятностью 10~4, а массив длиной 10 Мбит с вероятностью 10~1. К тому же обнаружить ошибку в массиве из нескольких мегабайт немного сложнее чем в пакете из нескольких килобайт. При обнаружении ошибки придется повторить передачу всего массива что гораздо сложнее чем повторно передать небольшой пакет данных. Но при повторной передаче большого массива снова высока вероятность ошибки и процесс этот при слишком большом массиве может повторяться до бесконечности.

С другой стороны, пакеты имеют преимущество и перед побайтовой (8 бит) и пословной (16 бит или 32 бита) передачей информации так как увеличиваться полезная загрузка сети за счет уменьшения требуемого количества служебной информации. Это же относиться и к маленьким пакетам длиной в несколько байт. Ведь каждый передаваемый по сети пакет обязательно содержит в себе биты, относящиеся непосредственно к обмену в сети (стартовый бит, биты адресации, биты типа и номера пакетов и так далее). Для маленьких пакетов доля этой служебной информации будет непозволительно высокой что приведет к снижению интегральной (средней) скорости обмена информации между абонентами сети.

Существует некоторое оптимальная длина пакета (или оптимальный диапазон длин пакета) при которой средняя скорость обмена информацией по мети будет максимальна. Эта длина не является неизменной величиной она зависит и от уровня помех, и от метода управления обменом и от количества абонентов сети, и от характера передаваемой информации и от многих других факторов.

Структура пакета определяется прежде всего аппаратурными особенностями данной сети выбранной топологии и типом передачи информации, а также существенно зависит от использованного протокола (порядка обмена информации).

Для передачи информации в сети Ethernet применяется стандартный код Манчестер-П. при этом один уровень сигнала нулевой, а другой – отрицательный, то есть постоянная составляющая сигнала не равна нулю. При отсутствии передачи потенциал в сети нулевой. Гальваническая развязка осуществляется аппаратурой адаптеров репитеров и концентраторов. При этом приемопередатчик сети гальванически развязан от остальной аппаратуры с помощь. Трансформаторов и изолированного источника питания, а с кабелем сети соединен напрямую.

Доступ к сети Ethernet осуществляется по методу CSMA/CD, обеспечивающим полное равноправие абонентов. В сети используются пакеты переменной длины со структурой, представленной на рисунке 1. Длина кадра Ethernet, то есть пакета без преамбулы) должна быт не менее 512 битовых интервалов или 51,2 мкс (именно такова предельная величина двойного времени прохождения в сети).

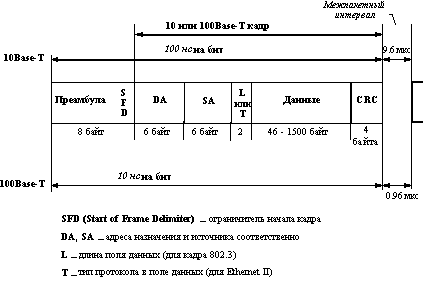


Рис. 1- Структура пакета сети Ethernet/

В пакет Ethernet входит следующие поля:

1. Преамбула состоит из 8 байт первые чес из которых представляют собой кол 10101010 а последний восьмой – код 10101011.В стандарте IEEE 802.3 этот последний байт называется признаком начала кадра (SFD – Start of Frame Delimiter) и образует отдельное поле пакета.

2. Адрес получателя (приемника\_ и адрес отправителя(передатчика) включают по 6 байт. Эти адресные поля обрабатывается аппаратурой абонентов.

3. Поля управления (L/T – Length/Type) содержат информацию о длине поля данных. Оно может также определить тип используемого протокола. Принято считать, что если значение этого поля не больше 1500, то оно определяет длину поля данных если ее его значение больше 1500, то оно определяет тип кадра. Поле управления обрабатывается программно.

4. Поле данных должно включать в себя от 46 до1500 байт данных. Если пакт должен содержать менее 46 байт данных, то поле данных дополняется байтами заполнения. Согласно стандарту IEEE 802.3 в структуре пакета выделяется специальное поле заполнения (pad data – незначащие данные) которое может иметь нулевую длину, когда данных достаточно (более 46 байт).

5. Поле контрольной суммы (FCS – Frame Check Sequence) содержит 32-разрядную циклическую контрольную сумму пакета (CRC) и служит для проверки правильности передачи пакета.

Таким образом минимальная длина кадра составляет 64 байта (512 бит). Именно эта величина определяет максимально допустимую задержку распространения сигнала в 512 битовых интервалов (51,2 мкс для Ethernet и 5,12 мкс для Fast Ethernet). Стандарт предполагает, что преамбула может уменьшатся при прохождении пакета через различные сетевые устройства поэтому она не учитываться. Максимальная длина кадра равна 1518 байтам (12144 бито то ест 1241,4 мкс для Ethernet и 121,44 мкс для Fast Ethernet). Это важно для выбора размера буферной памяти сетевого оборудования и для оценки общей загруженности сети.

**2.2 Адресация пакетов**

Каждый абонент (узел) локальной сети должен иметь свой уникальный адрес (он же идентификатор, MAC- адрес) чтобы ему можно было адресовать пакеты. Существуют две основные системы присвоения адресов абонентам сети (точнее сетевым адаптерам этих абонентов).

Первый подход заключается в том, что при установке сети каждому абоненту присваивается свой адрес (программно или с помощью переключателей на плите адаптера). При этом требуемое количество разрядов адреса определяется из простого уравнения:

,

где n – количество разрядов адреса, а максимально возможное количество абонентов в сети.

Например, восьми разрядов адреса достаточно для сети из 255 абонентов один адрес (обычно 1111…11) отводиться для широковещательной передачи, то есть используется для пакетов, адресованных всем абонентам одновременно.

Достоинства данного подхода -простота и малый объем служебной информации в пакете, а также простота аппаратуры адаптера, распознающей адрес пакета.

Недостаток – трудоемкость задания адресов и возможность ошибки (например, двум абонентам сети может быть присвоен один и тот же адрес).

Второй подход к адресации был разработан международной организацией IEEE, занимающейся стандартизацией сетей. Именно он используется в большинстве сетей и рекомендован для всех новых разработок. Идея состоит в том, чтобы присваивать уникальный сетевой адрес каждому адаптеру сети еще на этапе его изготовления. Если количество возможных адресов будет достаточно большим, то можно быть уверенным что в любой сети не будет абонентов с одинаковыми адресами. Для реализации этого подхода был выбран 48-битнфй формат адреса что соответствует примерно 280 триллионам различных адресов.

Чтобы распределить возможные диапазоны адресов между многочисленными изготовителями сетевых адаптеров была предложена следующая структура48-битного стандартного адреса:

1.Младшие 24 разряда кода адреса называются OUA (Organizationally Unique Address) – организационно уникальный адрес. Именно их присваивает производитель сетевого адаптера. Может быть свыше 16 миллионов комбинаций.

2.Следующие 22 разряда кода называются OUI (Organizationally Unique Identifier) – организационно уникальный идентификатор. IEEE присваивает один или несколько ОШ (опросный штепсель) каждому производителю сетевых адаптеров. Это позволяет исключить совпадение адресов адаптеров от различных производителей. Возможно, свыше 4 миллионов различных OUI.

3.Два старших разряда адреса являются управляющими и определяют тип адреса способ интерпретации остальных 46 разрядов. Старший бит I/G (Individual/Group) определяет индивидуальный это адрес или групповой. Если он установлен на 0, то это индивидуальный адрес если установлен на 1, то групповой (многопунктовый или функциональный) адрес. Пакеты с групповыми адресами получают все имеющие сетевые адаптеры причем групповой адрес определяется всеми 46 младшими разрядами. Второй управляющий бит U/L (Universal/Local) называется флажком универсального/местного управления и определяет как был присвоен адрес данному сетевому адаптеру. Обычно он установлен на 0. Установка бита U/L на 1 означает что адрес задан не производителем сетевого адаптера, а организацией, использующей данную сеть. Это довольно редкая ситуация.

Для широковещательной передачи используется специально выделенный сетевой адрес, все 48 битов которого установлены в единицу. Его принимают все абоненты сети независимо от их индивидуальных и групповых адресов.

Данной системы придерживаются многие сети, такие как Ethernet, Fast Ethernet, Tokin-Ring, FDDI, 100VG-AnyLAN. Ее недостатки- высокая сложность аппаратуры сетевых адаптеров, а также большая доля служебной информации в передаваемом пакете (адрес источника и адрес приемника требуют 96 битов пакета, или 12 байт).

Во многих сетевых адаптерах предусмотрен режим, который называется циркулярным. В этом режиме адаптер принимает все пакеты приходящие у нему независимо от значения поля адреса приемника. Этот режим используется, например для проведения диагностик сети измерения ее производительности контроля за ошибками передачи. В этом случае один компьютер принимает и контролирует все пакеты, приходящие по сети, но сам ничего не передает. В этом же режиме работает сетевые адаптеры мостов и коммутаторы, которые должны обрабатывать перед ретрансляцией все приходящие у ним пакеты.

**3.ОСНОВЫ ТЕХНОЛОГИИ ОРГАНИЗАЦИИ КАБЕЛЬНЫХ СИСТЕМ СЕТИ**

Разработанный в 1973 году стандарт Ethernet сегодня является одной из самых использованным среди технологий ЛВС.

По мере расширения сети доступная пользователю полоса (средняя скорость передачи) сужается за счет того, что канал 10 Mbps делится между всеми узлами сети. Повышение производительности компьютеров и использование приложений с интенсивным сетевым трафиком требует расширения полосы для полной реализации возможностей программ и оборудования. Расширение сетей и повышение производительности компьютеров требует расширения доступной пользователям полосы обеспечиваемой сетевой средой передачи.

Существуют два способа расширения полосы доступной каждому пользователю. Технология Fast Ethernet базируется на расширении полосы разделяемой среды до 100 Mbps обеспечивая рост скорости в 10 раз. Другим способом является снижение числа узлов сети, имеющих доступ к разделяемой среде и, следовательно, расширение доступной оставшимся узлам полосы. В предельном случае вся полоса канала передачи может быть предоставлена одному пользователю.

Процесс снижения числа узлов в сети называется сегментацией и осуществляется за счет деления большой сети на несколько меньших. Поскольку пользователям может требоваться доступ к ресурсам других сегментов нужен механизм обеспечения такого доступа обеспечивающий межсегментный обмен с достаточно высокой скоростью. Новый тип устройств называется коммутаторами Ethernet обеспечивает требуемые возможности.

**3.1 Основы организации сети**

Повторители

В начале 80-х годов сети Ethernet организовывались на базе шинной топологии с использованием сегментов на основе коаксиального кабеля длиной до 500 метров. Увеличение размеров сетей поставило задачу преодоления 50-метрового барьера. Для решения этой задачи использовались повторители (repeater):



Рисунок 2- топология повторителя.

Повторитель просто копирует (пересылает) все пакеты Ethernet из одного сегмента во все другие подключенные к нему. Основной задачей повторителя является восстановление электрических сигналов для передачи их в другой сегмент. За счет усиления и восстановления формы электрических сигналов повторителем становиться возможным расширение сетей, построенных на основе коаксиального кабеля и увеличение общего числа пользователей сети.

Мосты и маршруты.

При использовании повторителей максимальная протяженность сети составляет 2500 метров. Для преодоления этого ограничения требуются другие устройства, называемые мостами (bridge). Мосты имеют много отличий от повторителей. Повторители передают все пакеты, а мосты только те, которые необходимы в данный момент. Если пакет не нужно передавать в другой сегмент он фильтруется. Для мостов существуют многочисленные алгоритмы (правила) передачи и фильтрации пакетов минимальным требованиям является фильтрация пакетов по адресу получателя.

Другим важным отличием мостов от повторителей является то, что сегменты, подключенные к повторителю, образуют свою среду с полосой 10 Mbps. При использовании моста пользователи одного сегмента распределяют полосу, а пользователи разных сегментов используют независимые среды. Следовательно, мост обеспечивает преимущества как с точки зрения расширения сети, так и обеспечения большей полосы для каждого пользователя.

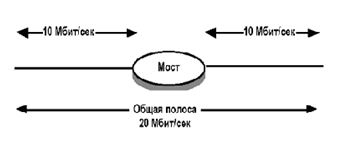


Рисунок 3 – схематичное представление моста

Поначалу в сетях Ethernet использовалась шинная топология на основе коаксиального кабеля, а для расширения сетей применялись 2-х портовые повторители или мосты. Однако в конце 80-х годов началось широкое распространение сетей на основе кабеля со скрученными парами проводников (витая пара). Новая технология 10Base-T стала очень популярной и привела к трансформации топологии сетей от шинной магистрали к организации соединения типа “звезда”.

Требования к повторителям и мостам для таких сетей существенно изменились по сравнению с простыми двухпортовыми устройствами для сетей с шинной топологией – современные мосты и повторители представляет собой сложные многопортовые устройства. Мосты позволяют сегментировать сети на меньшие части, в которых общую среду разделяет на небольшое число пользователей.

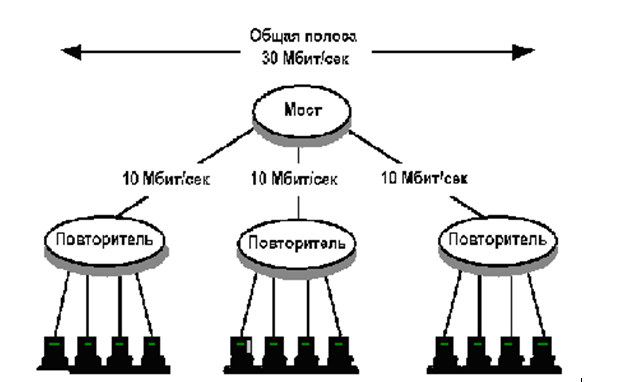
Маршрутизаторы подобны мостам также позволяют сегментировать сети Ethernet. Маршрутизаторы фильтруют и пересылают сетевой трафик на основе алгоритмов и правил существенно отличающихся от тех, что используются мостами. Такой способ сегментирования сетей более дорог.

Рисунок 4- схема организации маршрутизатора

Переключение портов

Нынешние модульные концентраторы (повторители) часто позволяют организовать несколько сегментов каждый из которых предоставляет пользователям отдельную раздельную полосу 10 Mbps. Некоторые концентраторы позволяют программным путем разделять порты устройства на независимые сегменты такая возможность называется переключением портов. Концентратор, к примеру может содержать три различных сегмента Ethernet организованных внутренними средствами хаба. Переключение портов обеспечивает администратору сети высокую гибкость организации сегментов позволяя переносить порты из одного сегмента в другой программными средствами. Эта возможность полезна при распределении нагрузки между сегментами Ethernet и снижение расходов связанные с подобными операциями. Переключение портов статическое связывание портов с различными сегментами Ethernet – сильно отличается от коммутации Ethernet.

**3.2 Принципы коммутации сегментов и узлов локальной сети**

Технология коммутации сегментов Ethernet была предложена фирмой Kalpana в 199- году в ответ на растущие потребности в повышении пропускной способности связей высокопроизводительных серверов с сегментами рабочей станции. Эта технология основана на отказе от использования разделяемых линий связи между всеми узлами сегмента и использовании коммутаторов позволяющих одновременно передавать пакеты между всеми его парами портов.

Функционально многопортовый коммутатор работает как многопортовый мост, то есть работает на канальном уровне анализирует заголовки кадров автоматически строит адресную таблицу и на основании этой таблицы перенаправляет кадр в один из своих выходных портов или фильтрует его, удаляя из буфера. Новшество заключалось в параллельной обработке поступающих кадров, в то время как мост обрабатывает кадр за кадром. Коммутатор же обычно имеет несколько внутренних процессоров обработки кадров каждый из которых может выполнить алгоритм моста. Таким образом можно считать, что коммутатор – это мультипроцессорный мост, имеющий за счет внутреннего параллелизма высокую производительность.

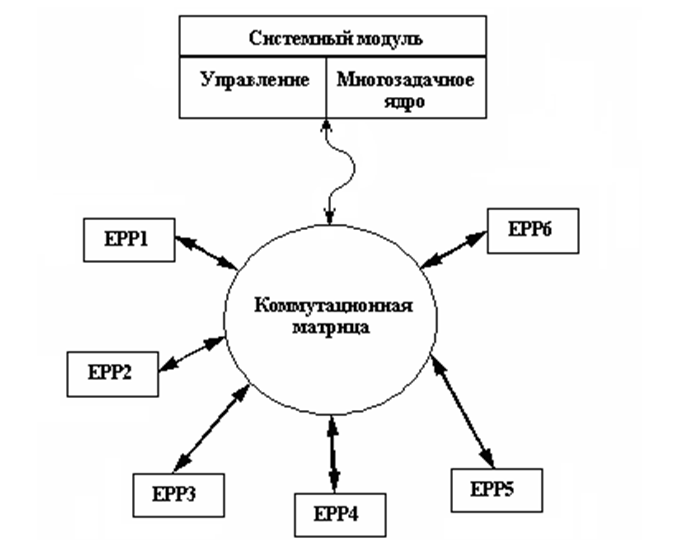
Структурная схема коммутатора EtherSwitch предложенного фирмой Kalpana представлена на рисунке 5.

Рисунок 5-

Каждый порт обслуживается одним процессором пакетов ethernet – EPP (Ethernet Packet Processor). Кроме того, коммутатор имеет системный модуль, который координирует работу всех процессоров EPP. Системный модуль ведет общую адресную таблицу коммутатора и обеспечивает управление коммутаторов по протоколу SNMP. Для передачи кадров между портами используется коммутационная матрица подобная тем, которые работают в телефонных коммутаторах или мультипроцессорных компьютерах соединяя несколько процессоров с несколькими модулями памяти.

При построении кадра в какой либо порт процессора EPP буферизуетс несколко первых байтов кадра для того чтобы прочитать адресс назначения.

Исходники, ЕЕЕЕЕЕЕЕЕЕ!

**MainActivity.kt**

package com.company.dima.studentsfirebase.activity

import android.os.Bundle

import androidx.appcompat.app.AppCompatActivity

import androidx.fragment.app.Fragment

import androidx.fragment.app.FragmentManager

import com.company.dima.studentsfirebase.R

import com.company.dima.studentsfirebase.fragments.StudentFragment

import com.company.dima.studentsfirebase.fragments.StudentListFragment

import com.company.dima.studentsfirebase.fragments.StudentSettingsFragment

import com.company.dima.studentsfirebase.inter\_face.Callbacks

class MainActivity : AppCompatActivity(), Callbacks {

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

val fm: FragmentManager = supportFragmentManager

val currentFragment = fm.findFragmentById(R.id.fragment\_container)

if (currentFragment == null) onMainScreen()

}

override fun onMainScreen(sortingMode: String) {

this.setTitle(R.string.app\_name)

startFragment(StudentListFragment.newInstance(sortingMode))

}

override fun onCreateNewStudent() {

this.setTitle(R.string.adding\_label)

startFragment(StudentFragment.newInstance())

}

override fun onStudentSelected(pathKey: String) {

this.setTitle(R.string.editing\_label)

startFragment(StudentFragment.newInstance(pathKey))

}

override fun onSettings() {

this.setTitle(R.string.setting\_label)

startFragment(StudentSettingsFragment.newInstance())

}

private fun startFragment(fragment: Fragment) {

supportFragmentManager

.beginTransaction()

.replace(R.id.fragment\_container, fragment)

.commit()

}

}

**StudentFirebase.kt**

package com.company.dima.studentsfirebase.database

import com.company.dima.studentsfirebase.model.Student

import com.google.firebase.database.FirebaseDatabase

object StudentFirebase {

private const val KEY = "STUDENT\_DB"

private val firebase = FirebaseDatabase.getInstance()

val reference = firebase.getReference(KEY)

val students = arrayListOf<Student>()

fun addStudent(student: Student) {

student.pathKey = firebase.reference.push().key ?: ""

firebase.reference.child(KEY).child(student.pathKey).setValue(student)

}

fun deleteStudent(student: Student) {

firebase.reference.child(KEY).child(student.pathKey).removeValue()

}

fun updateStudent(student: Student) {

firebase.reference.child(KEY).child(student.pathKey).setValue(student)

}

fun sortStudents(sortMode: String) {

when (sortMode) {

"name" -> students.sortBy { it.name }

"age" -> students.sortBy { it.age }

"rating" -> students.sortBy { 1.0 / it.rating }

else -> students.sortBy { it.pathKey }

}

}

}

**StudentFragment.kt**

package com.company.dima.studentsfirebase.fragments

import android.content.Context

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import android.widget.Button

import android.widget.EditText

import android.widget.Toast

import androidx.activity.addCallback

import androidx.core.content.res.ResourcesCompat

import androidx.fragment.app.Fragment

import com.company.dima.studentsfirebase.inter\_face.Callbacks

import com.company.dima.studentsfirebase.R

import com.company.dima.studentsfirebase.model.Student

import com.company.dima.studentsfirebase.database.StudentFirebase

class StudentFragment : Fragment() {

private lateinit var nameStudent: EditText

private lateinit var ageStudent: EditText

private lateinit var ratingStudent: EditText

private lateinit var functionalButton: Button

private lateinit var neutralButton: Button

private var student: Student = Student()

private var callbacks: Callbacks? = null

override fun onAttach(context: Context) {

super.onAttach(context)

callbacks = context as? Callbacks

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

requireActivity().onBackPressedDispatcher.addCallback(this) {

if (correctStudentData())

StudentFirebase.updateStudent(student)

snapBackToReality()

}

}

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View? {

val view = inflater.inflate(R.layout.fragment\_student, container, false)

nameStudent = view.findViewById(R.id.name\_student) as EditText

ageStudent = view.findViewById(R.id.age\_student) as EditText

ratingStudent = view.findViewById(R.id.rating\_student) as EditText

functionalButton = view.findViewById(R.id.add\_button) as Button

neutralButton = view.findViewById(R.id.neutral\_button) as Button

return view

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

val studentPathKey = arguments?.getSerializable(ARG\_STUDENT\_ID) as String?

if (studentPathKey != null) {

try {

student = StudentFirebase.students.find { it.pathKey == studentPathKey }!!

} catch (e: Exception) {

}

functionalButton.text = resources.getString(R.string.delete)

functionalButton.background = ResourcesCompat

.getDrawable(resources, R.drawable.deleting\_button\_border, null)

nameStudent.setText(student.name)

ageStudent.setText(student.age.toString())

ratingStudent.setText(student.rating.toString())

functionalButton.setOnClickListener {

StudentFirebase.deleteStudent(student)

Toast.makeText(context, R.string.deleting\_notification, Toast.LENGTH\_SHORT).show()

snapBackToReality()

}

} else addStudent()

neutralButton.setOnClickListener {

if (correctStudentData() && studentPathKey != null)

StudentFirebase.updateStudent(student)

snapBackToReality()

}

}

private fun addStudent() {

functionalButton.setOnClickListener {

val name = nameStudent.text.toString()

val age = ageStudent.text.toString().toIntOrNull()

val rating = ratingStudent.text.toString().toFloatOrNull()

if (name.isNotEmpty() && age != null && rating != null) {

StudentFirebase.addStudent(Student(name, age, rating))

Toast.makeText(context, R.string.adding\_notification, Toast.LENGTH\_SHORT).show()

snapBackToReality()

} else Toast.makeText(

context,

R.string.input\_error\_notification, Toast.LENGTH\_SHORT

).show()

}

}

private fun correctStudentData(): Boolean {

val student = Student()

student.name = nameStudent.text.toString()

student.age = try {

ageStudent.text.toString().toInt()

} catch (e: Exception) {

student.age

}

student.rating = try {

ratingStudent.text.toString().toFloat()

} catch (e: Exception) {

student.rating

}

if (!(this.student.name == student.name &&

this.student.age == student.age &&

this.student.rating == student.rating)

) {

this.student.name = student.name

this.student.age = student.age

this.student.rating = student.rating

return true

}

return false

}

private fun snapBackToReality() {

callbacks?.onMainScreen()

}

override fun onDetach() {

super.onDetach()

callbacks = null

}

companion object {

private const val ARG\_STUDENT\_ID = "student\_id"

@JvmStatic

fun newInstance(pathKey: String): StudentFragment {

val args = Bundle().apply { putString(ARG\_STUDENT\_ID, pathKey) }

return StudentFragment().apply { arguments = args }

}

@JvmStatic

fun newInstance() = StudentFragment()

}

}

**StudentListFragment.kt**

package com.company.dima.studentsfirebase.fragments

import android.content.Context

import android.os.Bundle

import android.view.\*

import android.widget.TextView

import android.widget.Toast

import androidx.core.os.bundleOf

import androidx.fragment.app.Fragment

import androidx.recyclerview.widget.LinearLayoutManager

import androidx.recyclerview.widget.RecyclerView

import androidx.swiperefreshlayout.widget.SwipeRefreshLayout

import com.company.dima.studentsfirebase.inter\_face.Callbacks

import com.company.dima.studentsfirebase.R

import com.company.dima.studentsfirebase.model.Student

import com.company.dima.studentsfirebase.database.StudentFirebase

import com.google.android.material.floatingactionbutton.FloatingActionButton

import com.google.firebase.database.DataSnapshot

import com.google.firebase.database.DatabaseError

import com.google.firebase.database.ValueEventListener

class StudentListFragment : Fragment() {

private lateinit var studentRecyclerView: RecyclerView

private var adapter: StudentAdapter = StudentAdapter(emptyList())

private var callbacks: Callbacks? = null

private lateinit var menu: Menu

private var addingButton: FloatingActionButton? = null

override fun onAttach(context: Context) {

super.onAttach(context)

callbacks = context as? Callbacks

StudentFirebase.students.clear()

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setHasOptionsMenu(true)

}

override fun onCreateView(

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View? {

val view = inflater.inflate(R.layout.fragment\_student\_list, container, false)

studentRecyclerView = view.findViewById(R.id.student\_recycler\_view) as RecyclerView

studentRecyclerView.layoutManager = LinearLayoutManager(context)

studentRecyclerView.adapter = adapter

return view

}

override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {

super.onCreateOptionsMenu(menu, inflater)

inflater.inflate(R.menu.fragment\_student\_list, menu)

this.menu = menu

}

override fun onOptionsItemSelected(item: MenuItem): Boolean {

return when (item.itemId) {

R.id.sorting\_student -> {

callbacks?.onSettings()

true

}

else -> return super.onOptionsItemSelected(item)

}

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

val swipeRefreshLayout: SwipeRefreshLayout = view.findViewById(R.id.refresh)

swipeRefreshLayout.setOnRefreshListener {

callbacks?.onMainScreen()

}

StudentFirebase.reference.addValueEventListener(object : ValueEventListener {

override fun onDataChange(dataSnapshot: DataSnapshot) {

for (ds in dataSnapshot.children)

try {

StudentFirebase.students.add(requireNotNull(ds.getValue(Student::class.java)))

} catch (e: Exception) {

continue

}

val set = HashSet<Student>(StudentFirebase.students)

StudentFirebase.students.clear()

StudentFirebase.students.addAll(set)

StudentFirebase.sortStudents(requireNotNull(arguments?.getString(SORTING\_MODE)))

adapter.students = StudentFirebase.students

studentRecyclerView.adapter = adapter

}

override fun onCancelled(databaseError: DatabaseError) {

Toast.makeText(context, "$databaseError", Toast.LENGTH\_SHORT).show()

}

})

addingButton = view.findViewById(R.id.floatingActionButton)

addingButton?.setOnClickListener {

callbacks?.onCreateNewStudent()

}

}

override fun onDetach() {

super.onDetach()

callbacks = null

}

private inner class StudentHolder(view: View) : RecyclerView.ViewHolder(view),

View.OnClickListener {

private lateinit var student: Student

private val nameTextView: TextView = itemView.findViewById(R.id.name\_student)

private val ageTextView: TextView = itemView.findViewById(R.id.age\_student)

private val ratingTextView: TextView = itemView.findViewById(R.id.rating\_student)

init {

itemView.setOnClickListener(this)

}

fun bind(student: Student) {

this.student = student

nameTextView.text = this.student.name

ageTextView.text = this.student.age.toString()

ratingTextView.text = this.student.rating.toString()

}

override fun onClick(v: View) {

callbacks?.onStudentSelected(student.pathKey)

}

}

private inner class StudentAdapter(var students: List<Student>) :

RecyclerView.Adapter<StudentHolder>() {

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): StudentHolder {

val layoutInflater = LayoutInflater.from(context)

val view = layoutInflater.inflate(R.layout.list\_item\_student, parent, false)

return StudentHolder(view)

}

override fun onBindViewHolder(holder: StudentHolder, position: Int) {

holder.bind(students[position])

}

override fun getItemCount() = students.size

}

companion object {

private const val SORTING\_MODE = "sorting mode"

@JvmStatic

fun newInstance(sortingMode: String = ""):

StudentListFragment = StudentListFragment().apply {

arguments = bundleOf(SORTING\_MODE to sortingMode)

}

}

}

**StudentSettingsFragment.kt**

package com.company.dima.studentsfirebase.fragments

import android.Manifest

import android.app.Activity

import android.content.Context

import android.content.Intent

import android.content.pm.PackageManager

import android.net.Uri

import android.os.Bundle

import android.view.LayoutInflater

import android.view.View

import android.view.ViewGroup

import android.widget.Button

import androidx.fragment.app.Fragment

import androidx.activity.addCallback

import androidx.core.app.ActivityCompat

import androidx.core.content.ContextCompat

import com.company.dima.studentsfirebase.inter\_face.Callbacks

import com.company.dima.studentsfirebase.R

import kotlin.system.exitProcess

class StudentSettingsFragment : Fragment() {

private var buttons: Array<Button?> = Array(3) { null }

private lateinit var contactButton: Button

private lateinit var backButton: Button

private lateinit var exitButton: Button

private var callbacks: Callbacks? = null

override fun onAttach(context: Context) {

super.onAttach(context)

callbacks = context as? Callbacks

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

requireActivity().onBackPressedDispatcher.addCallback(this) {

callbacks?.onMainScreen()

}

}

override fun onCreateView(

inflater: LayoutInflater, container: ViewGroup?,

savedInstanceState: Bundle?

): View? {

return inflater.inflate(R.layout.fragment\_student\_settings, container, false)

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

val idButtonArray = arrayOf(R.id.nameButton, R.id.ageButton, R.id.ratingButton)

for (i in idButtonArray.indices) {

buttons[i] = view.findViewById(idButtonArray[i])

buttons[i]?.setOnClickListener {

callbacks?.onMainScreen(switchSortParameter(i))

}

}

contactButton = view.findViewById(R.id.contact\_button)

contactButton.setOnClickListener {

if (isCallPermissionGranted()) callMe()

else requestCallPermission()

}

backButton = view.findViewById(R.id.neutral\_button)

backButton.setOnClickListener {

callbacks?.onMainScreen()

}

exitButton = view.findViewById(R.id.exit\_button)

exitButton.setOnClickListener {

exitProcess(0)

}

}

private fun switchSortParameter(number: Int): String =

when (number) {

0 -> "name"

1 -> "age"

2 -> "rating"

else -> ""

}

private fun isCallPermissionGranted(): Boolean =

context?.let {

ContextCompat.checkSelfPermission(

it,

Manifest.permission.CALL\_PHONE

)

} == PackageManager.PERMISSION\_GRANTED

private fun requestCallPermission() {

ActivityCompat.requestPermissions(

context as Activity,

arrayOf(Manifest.permission.CALL\_PHONE), REQUEST\_CODE

)

}

private fun callMe() {

val dial = "tel: +${resources.getString(R.string.phone)}"

val intent = Intent(Intent.ACTION\_DIAL, Uri.parse(dial))

startActivity(intent)

}

override fun onDetach() {

super.onDetach()

callbacks = null

}

companion object {

private const val REQUEST\_CODE = 100

@JvmStatic

fun newInstance() = StudentSettingsFragment()

}

}

**Callbacks.kt**

package com.company.dima.studentsfirebase.inter\_face

interface Callbacks {

fun onCreateNewStudent()

fun onStudentSelected(pathKey: String)

fun onSettings()

fun onMainScreen(sortingMode: String = "")

}

**Student.kt**

package com.company.dima.studentsfirebase.model

import android.os.Parcelable

import kotlinx.parcelize.Parcelize

@Parcelize

data class Student(

var name: String = "",

var age: Int = 0,

var rating: Float = 0.0f,

var pathKey: String = ""

) : Parcelable