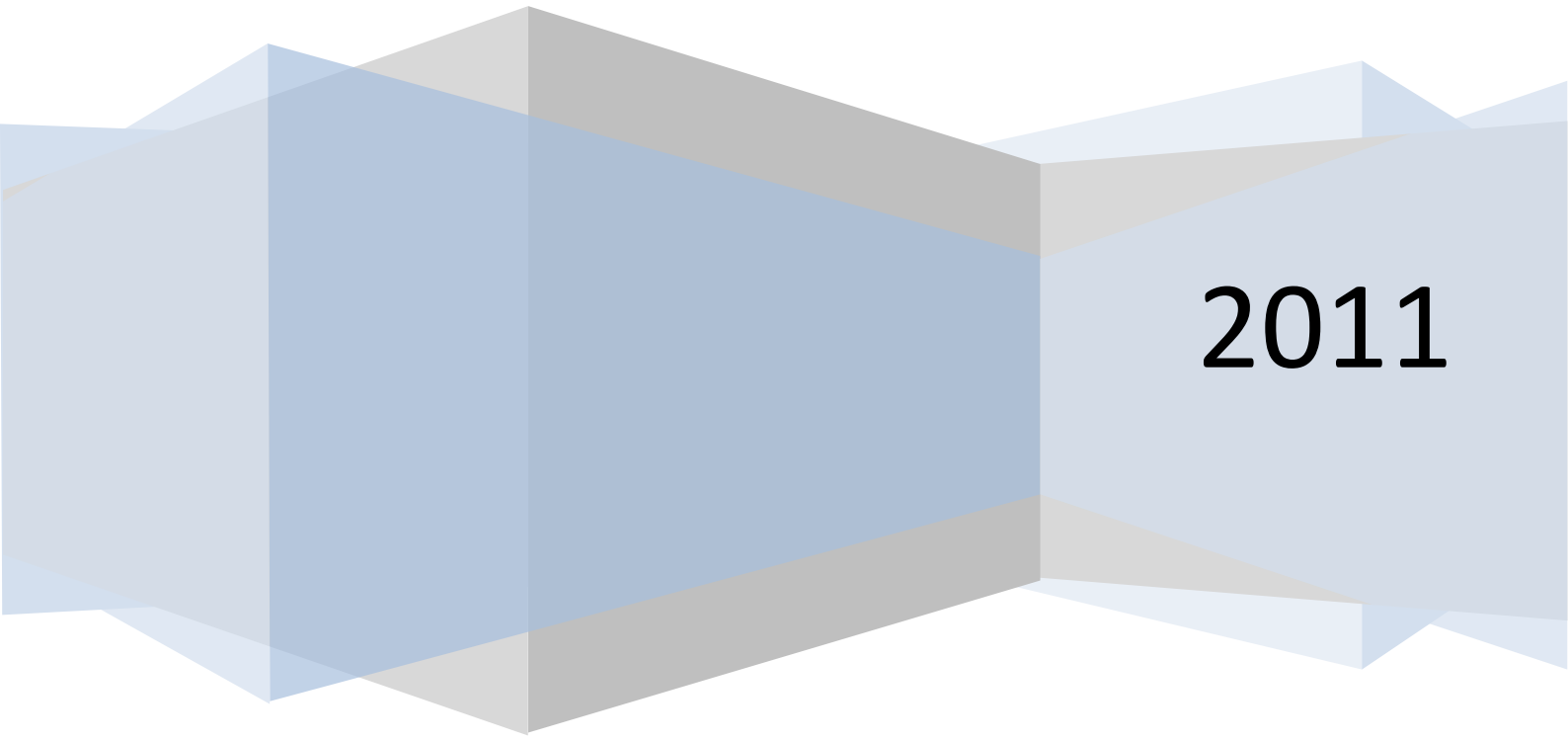
Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
Новгородский государственный университет
имени Ярослава Мудрого

Сетевые информационные технологии

Курс лекций



2011

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

Новгородский государственный университет
имени Ярослава Мудрого
Институт электронных и информационных систем

Кафедра радиосистем

Леонтьев А. Г.

Сетевые информационные технологии

Учебное пособие

Великий Новгород
2011

УДК 621.38
ББК

Леонтьев А. Г. Сетевые информационные технологии: Учебное пособие / ФГБОУ «Новгородский государственный университет им. Ярослава Мудрого», Великий Новгород, 2011 г. - 95 с.

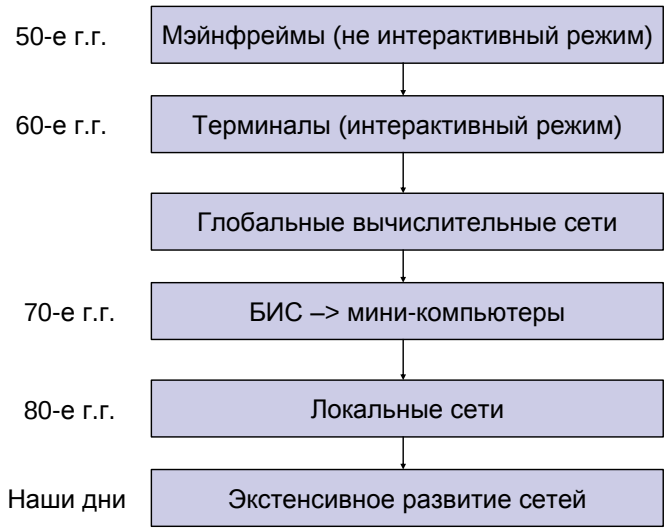
В учебном пособии рассмотрена основы передачи дискретных данных, вопросы управления сетевым трафиком и основные методы сжатия данных.

Учебное пособие отвечает новым образовательным стандартам и предназначено для подготовки специалистов по специальности 210302.65 "Радиотехника".

Учебное пособие одобрено советом института электронных и информационных систем Новгородского государственного университета имени Ярослава Мудрого.

Оглавление

Эволюция компьютерных сетей



Эволюция компьютерных сетей началась в 50х годах прошлого века. Первые компьютеры были огромными монстрами и занимали целые здания. Предназначались они лишь для небольшого числа избранных пользователей.

Работа строилась по системе пакетной обработки данных на базе одного большого компьютера – мэйнфрейма. Пользователи делали перфокарты, содержащие данные для программ и непосредственно сами программы. Операторы в вычислительном центре, где находился мэйнфрейм, вводили данные с перфокарт в компьютер, а распечатанные результаты пользователи получали на след. день. Следовательно, одна сделанная ошибка в стопке перфокарт означала задержку на сутки (как минимум).

Интерактивный режим – это когда пользователь оперативно руководит процессом обработки своих данных и, естественно, он был бы гораздо удобнее. Но интересами программистов и инженеров на заре развития компьютеров пренебрегали по причине огромной стоимости вычислительных машин. Во главу угла ставилась именно эффективность работы компьютера.

В связи с удешевлением процессоров в начале 60х годов появился новый способ организации вычислительного процесса, предлагающий большую интерактивность. Так называемый терминальный режим позволял пользователю работать с мэйнфреймом, не замечая задержки в работе, т.е. время реакции машины было мало. Компьютер работал в режиме разделения времени, отдавая каждому терминалу какой-то квант машинного времени. Вычислительная мощность – централизована, ввод-вывод данных – распределен. Пользователь мог получить доступ к общим ресурсам, что создавало иллюзию работы в сети и единоличного владения компьютером. Именно централизованный характер обработки данных отличает терминальный режим доступа от локальной сети. С другой стороны – предприятия не были готовы к созданию сетей, т.к. т.н. «закон Гроша» очень хорошо отражал действительность того времени.

Закон Гроша: «мощность компьютера возрастает как квадратная функция от инвестированных затрат», другими словами «производительность компьютера пропорциональна квадрату его стоимости». За одну и ту же сумму выгоднее купить один более мощный компьютер, чем два менее мощных (их суммарная мощность намного меньше, чем у одного за те же деньги).

Создание глобальных вычислительных сетей началось с возникновением потребности доступа к компьютеру с терминалов, удаленных на сотни и тысячи километров. Терминалы соединялись с компьютером с помощью модемов через телефонные линии. Затем стали появляться системы, где вместо терминалов использовались компьютеры. Таким образом, компьютеры получили возможность обмениваться данными в автоматическом режиме, что является базовым механизмом любой вычислительной сети. Стали реализовываться механизмы обмена файлами, синхронизации баз данных, электронной почты.

Таким образом, вопреки популярному мнению, первыми в мире появились именно глобальные вычислительные сети, на которых и были отработаны ставшие стандартами современные концепции построения сетей, такие как многоуровневое построение коммуникационных протоколов, коммутация и маршрутизация пакетов.

В начале 70х произошел технологический прорыв в области производства компонентов – появились БИС (большие интегральные схемы), что привело к созданию мини-компьютеров. Закон Гроша перестал работать, т.к. несколько мини-компьютеров обеспечивали производительность большую, чем у мэйнфрейма, а стоили дешевле.

Предприятия получили возможность покупать компьютеры для задач управления производством, складом и т.д., но тем не менее, работали они автономно.

С ростом потребностей пользователей, появилась необходимость обмена данными между близко расположенными компьютерами. В ответ на эту потребность организации стали соединять компьютеры между собой и разрабатывать программное обеспечение для их взаимодействия. Так появились первые локальные вычислительные сети. На первых порах компьютеры использовали сложные и разнообразные нестандартные устройства сопряжения, каждое со своим способом передачи данных и типом кабелей. Это создавало, например, большое поле для студенческих работ. Большинство курсовых и дипломных работ назывались «Устройство сопряжения……».

В середине 80х годов утвердились стандартные технологии объединения компьютеров в сеть (Ethernet, Token Ring, Arcnet….). Толчком для развития этих стандартов стало массовое появление персональных компьютеров, которые стали идеальными элементами для построения сетей, т.к. были достаточно мощными для работы сетевого ПО, а с другой стороны нуждались в объединении для решения сложных вычислительных задач, и, что немаловажно, для разделения доступа к периферийным устройствам, которые били достаточно дорогими (принтер, графопостроитель) либо дисковых массивов хранения данных. Можно сказать, что на этом закончилась эра массового применения мэйнфреймов и терминалов.

В дальнейшем, развитие локальных сетей шло более экстенсивно. Увеличивались скорости сетей. Модифицировались протоколы. С появлением большого количества непрофессиональных пользователей развивались методы прозрачной работы в сети и доступа к разделяемым ресурсам.

Глобальные сети развивались и развиваются немного по другому пути. Скорости даже в 10 Мбит/с считаются во многих сетях непозволительной роскошью. Приходится пользоваться существующими низкоскоростными каналами связи, например, телефонными линиями. Поэтому, основным критерием часто является экономное расходование ресурсов канала связи. Поэтому, процедуры прозрачного доступа долгое время считались непозволительной роскошью.

Что изменилось в наши дни:

- Появляются высокоскоростные территориальные линии связи, что нивелирует разницу между локальными и глобальными сетями. Появляются множество удобных и прозрачных служб доступа к ресурсам (Internet).
- Локальные сети. Вместо соединяющего компьютеры пассивного кабеля в большом количестве появляется коммуникационное оборудование – коммутаторы, маршрутизаторы, шлюзы. Благодаря им, строятся большие корпоративные сети, насчитывающие тысячи компьютеров и имеющие сложную топологию.
- Сами компьютеры становятся крупнее (мощнее). После эйфории от небольших персональных компьютеров, возродился интерес к мощным серверам-мэйнфреймам. Оказывается, их намного проще обслуживать, чем сотни PC, объединенных в сеть.
- Очень важно. Появилась тенденция, несвойственная раньше компьютерам. Возникли новые виды трафика – голос, видео…, что требует внесения изменений в известные ранее протоколы и разработку новых. Этот вид информации чувствителен к задержкам при передаче (рассинхронизация и в конечном итоге искажение информации). Это называется трафик реального времени (в отличие от например, передачи файлов и электронной почты). Эти проблемы решаются различными способами, например, при помощи специально для этого разработанной технологии ATM. В конечном итоге это должно привести к слиянию не только локальных и глобальных сетей, но и информационных сетей (телефония, телевидение). Это должно случиться с массовым переходом от методов коммутации каналов (используемых в

телефонии) к методу коммутации пакетов. Наш курс будет посвящен, в том числе и изучению этих методов и соответствующих протоколов.

Существует много типов распределенных вычислительных систем. Их основным признаком является наличие нескольких центров обработки данных.

1. Мультипроцессорные компьютеры. Общая операционная система, распределяющая вычислительную нагрузку между процессорами. Взаимодействие – через общую оперативную память (самый простой метод).
2. Многомашинная система – несколько компьютеров + программные и аппаратные средства. Эффективность снижается, если связь по данным в распараллеливаемых задачах сильная. Обмен – через общие многовходовые периферийные устройства (дисковые массивы).
3. Вычислительные сети – еще большая автономность обрабатываемых данных и слабее программно-аппаратные связи. Связь – сетевые адаптеры и стандартные протоколы. Протяженность линий связи – большая. Основная цель создания ЛВС – разделение локальных ресурсов каждого пользователя сети между всеми.

Весь комплекс программно – аппаратных средств сети может быть описан многослойной моделью. В основе – аппаратный слой стандартизированных компьютерных платформ (от PC до супер ЭВМ, в соответствии с решаемыми задачами).

Второй слой это коммуникационное оборудование. Играет не менее важную роль. Включает мосты, повторители, коммутаторы, маршрутизаторы. Превращаются в основные наряду с компьютерами как по значимости и влиянию на характеристики сети, так и по цене. Могут иметь свое программное обеспечение и нуждаться в администрировании (напр., маршрутизаторы Cisco).

Третий слой, образующий программную платформу сети – ОС. От того, какие принципы сетевой организации положены в основу ОС, зависит общее функционирование сети (насколько она обеспечивает безопасность и защищенность данных, число пользователей, переносимость, масштабируемость и т.д.).

Самый верхний слой – различные сетевые приложения (БД, почтовые системы, системы автоматизации коллективной работы…). Здесь важно знать предоставляемый программой возможности, а также совместимость с другими сетевыми приложениями и ОС.

Преимущества использования сетей.

- Способность выполнять параллельные вычисления. Концептуальное преимущество. За счет этого может быть достигнута производительность любого сколь угодно мощного компьютера.
- Лучшее соотношение производительность-стоимость.
- Более высокая отказоустойчивость. Отказоустойчивость – это способность системы выполнять свои функции при отказе отдельных элементов аппаратуры и неполной доступности данных. Основа отказоустойчивости – избыточность. При выходе из строя узла, приписанные ему задачи переназначаются. Для этого используются процедуры динамической или статической реконфигурации.
- Адаптация к территориально распределенному характеру прикладных задач. Например, при автоматизации производства, на некоторой территории есть сотрудники, отделы и подразделения, автономно решающие свои задачи и поэтому рациональнее предоставлять им собственные вычислительные средства, но в тоже время, т.к. их задачи взаимосвязаны в рамках

общего производственного процесса, вычислительные средства нужно объединять в одну систему. Как раз в такой ситуации адекватное решение – вычислительная сеть.

- Возможность совместного использования данных и устройств. Речь идет об относительно дорогостоящих периферийных устройствах (дисковые массивы, принтеры, графопостроители…).
- Еще один побудительный мотив к построению сетей (гораздо более важный, чем экономия средств за счет разделения дорогой аппаратуры) – обеспечение оперативного доступа к корпоративной информации. В условиях современной жесткой конкуренции очень важна, например, качественная поддержка клиента. А при большой номенклатуре выпускаемых изделий, либо предоставляемых услуг оперативный доступ к справочной базе и надежные связи в корпоративной сети – это залог хорошей работы (пример, мобильная компания).

Выводы:

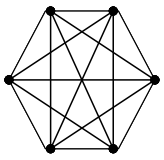
- Вычислительные сети явились результатом эволюции компьютерных технологий.
- Вычислительная сеть – это совокупность компьютеров, соединенных линиями связи. Линии связи образованы кабелями, сетевыми адаптерами и другими коммуникационными устройствами.
- Основная цель сети – обеспечить пользователям возможность совместного использования ресурсов всех компьютеров.
- Вычислительная сеть – это одна из разновидностей распределенных систем, достоинством которых является возможность распараллеливания вычислений и, следовательно, повышения производительности и отказоустойчивости системы.
- Важнейший этап в развитии сетей – появление стандартных сетевых технологий типа Ethernet, позволяющих быстро и эффективно объединять компьютеры различных типов.
- Использование вычислительных сетей дает следующие возможности:
    - Разделение дорогостоящих ресурсов.
    - Совершенствование коммуникаций.
    - Улучшение доступа к информации.
    - Свобода в территориальном размещении компьютеров.

Проблемы объединения нескольких компьютеров

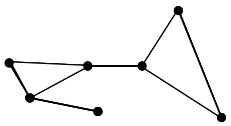
Топология определяет способ взаимосвязи узлов сети (компьютеров, концентраторов и т.п.). Физическая и логическая топология сети могут сильно различаться.

Полносвязанная топология

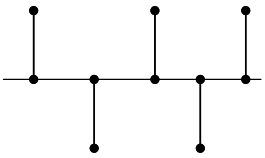
- Очень дорогой вариант
- Используется только при небольшом количестве узлов.



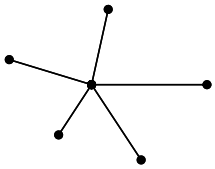
Полносвязная топология



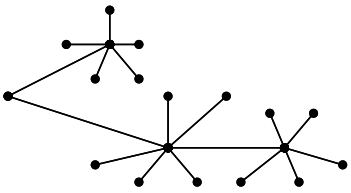
Ячеистая топология



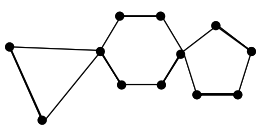
Общая шина



Звезда



Древовидная (иерархическая)звезда



Кольцо

Ячеистая топология

- Непосредственные связи имеют только узлы с напряженным трафиком.
- Связь между другими узлами производится через транзитные промежуточные узлы.
- Допускает объединение большого количества компьютеров и используется в глобальных сетях.

Общая шина

- Соединение компьютеров по «монтажному ИЛИ»
- Простота и экономичность
- Низкая надежность
- Невысокая производительность
- Большая распространенность.

Звезда

- Надежность
- Производительность
- Более высокая сложность из-за наличия концентратора
- Возможность фильтрации сообщений в концентраторе.

Древовидная (иерархическая звезда)

- Более высокая стоимость сетевого оборудования
- Более сложное управление и программное обеспечение.
- Наибольшая распространенность

Кольцо

- необходимость резервирования
- возможность контроля доставки сообщения по петле обратной связи

Смешанные топологии

Используются при построении больших сетей.

Совместное использование линий связи – основа упрощения и удешевления сети. Почти все топологии используют общие линии в той или иной мере. При этом возникают проблемы электрической и логической совместимости. Большие задержки и количество компьютеров - это основные препятствия к совместному использованию линий связи в глобальных сетях.

Адресация компьютеров в сети

Требования к адресу:

- Уникальность в сети любого размера
- Автоматизированная схема назначения адреса
- Иерархическая структура
- Удобство для пользователей, т.е. символьное представление.
- Компактность

Используется одновременно 3 схемы адресации и протоколы преобразования адресов:

- **Аппаратный адрес** (например, MAC-адрес сетевого адаптера) – только для небольшой сети (например 5A-8B-17-B1-92):
  - Компактный
  - Не требует ручного назначения
  - Отсутствует иерархия
  - Неудобства при замене и установке нескольких сетевых карт на один компьютер
- **Символьный адрес** или имя (например: ivan_danilov.radiosystem.novsu.ac.ru)
  - Применим в сети любого размера
  - Удобен для запоминания
  - Иерархическая структура позволяет использовать краткие символьные имена при работе в локальной сети
- **Числовой составной адрес** (IP, IPX)– для передачи по сети
  - Двухуровневая иерархия :старшая часть-номер сети, младшая-номер узла (например 193.233.109.xxx). В протоколе IPv6 – трехуровневая иерархия

Пользователи применяют символьные адреса, числовые номера при передаче по глобальной сети и аппаратные – внутри сети получателя сообщения.

Распределение адресов выполняется централизованно службой DNS (Domain Name System). В сети выделяется сервер, где хранятся все имена узлов, символьные и числовые.

Возможен распределенный подход к установлению соответствия между именами, когда перед началом передачи данных производится широковещательный опрос компьютеров сети для опознания символьного или числового адреса. Ответ содержит аппаратный адрес компьютера.

Структуризация как средство построения больших сетей

В сетях с небольшим количеством компьютеров (10…30) обычно применяется одна из типовых топологий – общая шина, кольцо, звезда, т.е. однородные топологии (когда все компьютеры в сети имеют одинаковые права по отношению друг к другу). Какие в этом плюсы:

- Простота наращивания числа компьютеров.
- Простое обслуживание и эксплуатация сети.

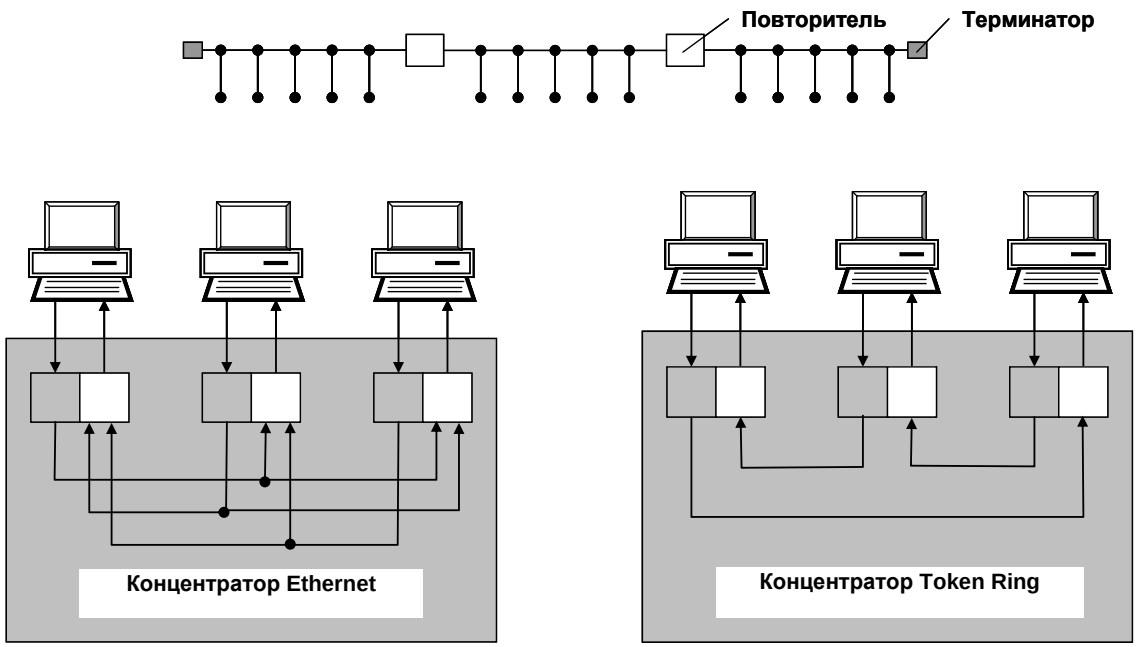
При построении больших сетей однородные топологии превращаются из преимуществ в недостатки:

- Ограничения на длину связей между узлами.
- Ограничения на количество узлов в сети.
- Ограничения на количество трафика, порождаемого узлом сети.

Например, технология Ethernet позволяет использовать коаксиальный кабель длиной не более 185м., к которому можно подключить не более 30 компьютеров. Однако при интенсивном обмене приходится снижать количество компьютеров до 10…20.

Для снятия подобных ограничений используются специальные методы структуризации сети и специальное структурообразующее оборудование, которое еще называют коммуникационным.

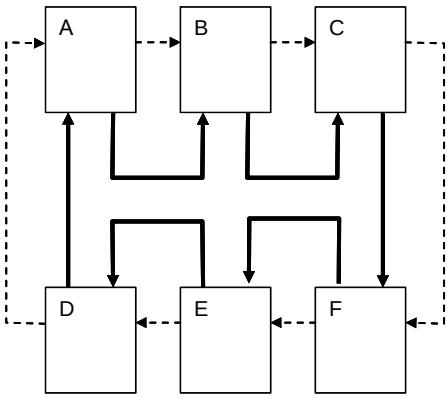
Физическая структуризация сети



Простейшее из коммуникационных устройств – **повторитель**. Используется для увеличения общей длины сети. Преодолеваются ограничения на длину связи за счет улучшения качества передаваемого сигнала (восстановление мощности и амплитуды, улучшение фронтов).

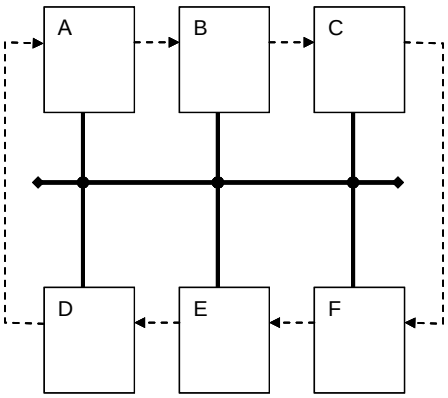
Повторитель, который имеет несколько портов и соединяет несколько физических сегментов, называется **концентратором** или **хабом** (hub – основа, центр деятельности). Концентраторы повторяют сигнал, пришедший по одному из портов, на других портах. Концентраторы характерны почти для всех базовых технологий локальных сетей. Разница в том, на каких именно портах повторяются входные сигналы. Хабы для сетей Ethernet повторяют входной сигнал на всех портах, кроме того, с которого они поступают. А, например, хабы для сетей Token Ring повторяют входной сигнал только на одном порту – к которому подключен следующий в кольце компьютер.

Очень важно: концентратор всегда изменяет физическую топологию сети, но при этом не изменяет логическую топологию.



Физическое кольцо

Логическое кольцо



Физическая общая шина

Логическое кольцо

Физическая топология – конфигурация связей, образованных кабелем, **логическая топология** – конфигурация информационных потоков между компьютерами.

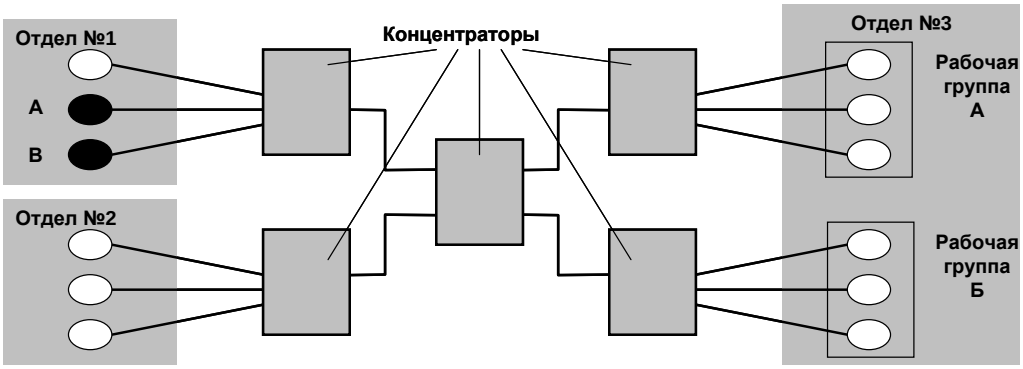
Во многих случаях физическая и логическая топологии совпадают. Например, сеть, представленная на картинке слева, имеет физическую топологию кольцо. Компьютеры получают доступ к сети за счет передачи друг другу специального кадра – маркера, причем маркер передается от компьютера к компьютеру в том же, порядке, в каком компьютеры образуют физическое кольцо.

Сеть, изображенная справа – пример несовпадения топологий. Фактически компьютеры соединены по технологии «общая шина», но доступ к шине происходит не по алгоритму случайного доступа, а путем передачи маркера в кольцевом порядке: от комп. А – комп. В – затем комп. С и т.д. Здесь порядок передачи маркера определяется не физическими связями в сети, а логическим конфигурированием драйверов сетевых адаптеров. Ничто не мешает настроить драйверы иначе, при этом физическая топология не изменится. Другой пример несовпадения – сеть на рисунке слева. Хаб образует физическую топологию звезда, однако логическая топология осталась без изменения – это общая шина. Логика доступа к сети не меняется – определение занятости среды, получение доступа, обработка коллизий – все остается в силе.

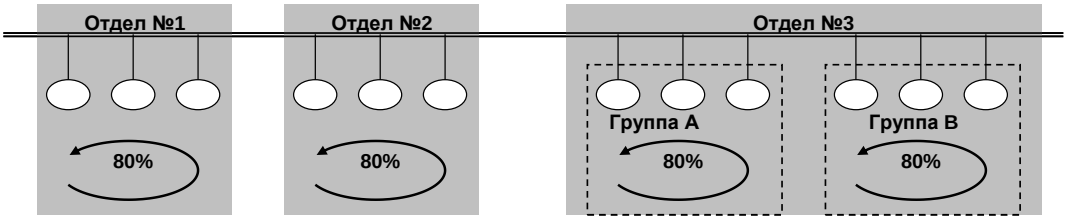
Физическая структуризация с помощью хабов полезна не только для увеличения расстояния между узлами, но и для повышения надежности сети. Например, если концентратор обнаруживает, что узел долго монопольно использует порт – просто отключит его.

Логическая структуризация сети

Сеть, в которой физические сегменты рассматриваются в качестве одной разделяемой среды, оказывается неадекватна структуре информационных потоков.



Физическая структуризация с помощью концентраторов



Логическая структура сети и распределение информационных потоков

Например, в сети с общей шиной взаимодействие любой пары компьютеров занимает ее на все время обмена, поэтому при увеличении числа узлов сети шина становится узким местом. Этот случай иллюстрирует верхний рисунок. Например, компьютер А, находясь в одной подсети с компьютером В, посылает ему данные, хабы распространяют их по всем сегментам и, хотя они и не нужны отделам 2 и 3, они вынуждены простаивать, принимая эти данные тоже.

Такая ситуация возникает из за того, что логическая структура сети осталась однородной и решение проблемы состоит состоит в логической структуризации сети.

Логическая структуризация сети *– это процесс разбиения сети на сегменты с локализованным трафиком.*

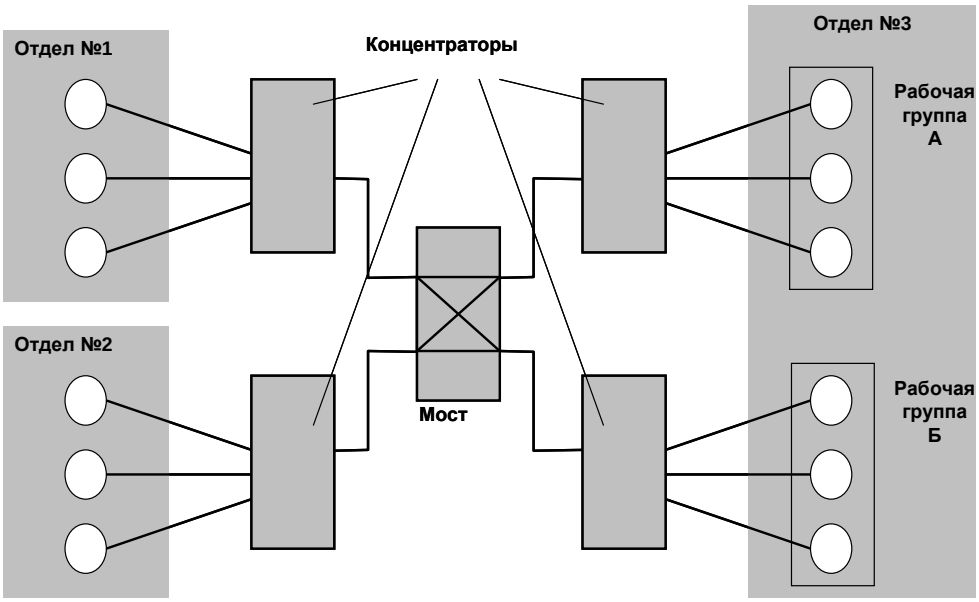
В рассмотренном выше примере нужно сделать так, чтобы кадры, передаваемые компьютерами отдела 1, не выходили за рамки подсети этого отдела, а в сети других отделов попадали только кадры, адресованные им.

Существует эмпирический закон, в соответствии с которым в логическом сегменте 80% трафика является внутренним, и 20% - внешним.

Для логической структуризации сети используют мосты, коммутаторы, маршрутизаторы и шлюзы.

Мост делит разделяемую среду передачи на части (логические сегменты). Фильтрация происходит по аппаратным адресам адаптеров.

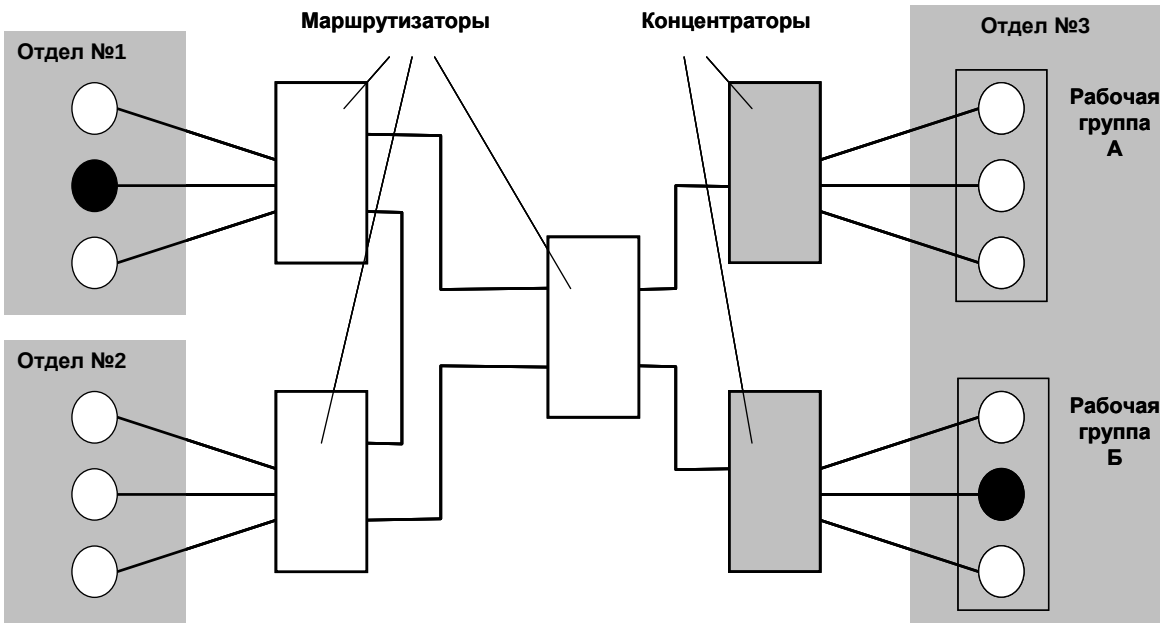
На рисунке показана сеть, полученная из сети с центральным хабом путем замены его на мост. Точной топологии связей мост не знает и поэтому мост достаточно примитивно представляет связи между узлами – он запоминает на какой порт поступил кадр данных от каждого компьютера и в дальнейшем передает кадры, предназначенные для этого компьютера, только ему в этот порт. Из-за этого применение мостов приводит к ограничению: сегменты должны быть соединены так, чтобы не образовывались замкнутые контуры.



Логическая структуризация сети с помощью моста

Коммутатор или свитч ничем не отличается от моста по принципу обработки кадров. Различие в том, что каждый его порт оснащен процессором, работающим независимо от других, за счет чего повышается производительность. Можно сказать, что коммутатор – это мост, работающий в параллельном режиме.

Ограничения по топологии связей в сетях с мостами и коммутаторами привели к тому, что появились более эффективные коммуникационные устройства – **маршрутизаторы** (англ. - **router**).



Логическая структуризация сети с помощью маршрутизаторов

Маршрутизаторы образуют логические сегменты посредством явной адресации, используя не плоские аппаратные адреса, а составные адреса – IP, где имеются поля номера сети. Все компьютеры, у которых значение этого поля одинаково, принадлежат к одному сегменты, который в этом случае называется подсетью. Маршрутизаторы могут работать в сетях со сколь угодно сложной топологией и в т.ч. с замкнутыми контурами, при этом ои осуществляют выбор наиболее рационального маршрута из нескольких возможных. Еще одна важная особенность маршрутизаторов – они могут связывать в единую сеть подсети, построенные с использованием разных технологий, например, Ethernet и X.25.

Кроме перечисленных устройств, отдельные части сети может соединять **шлюз (gateway)**. Они нужны не для локализации трафика, для объединения сетей с разными типами системного и прикладного программного обеспечения.

Многоуровневый подход. Протокол и интерфейс

Многоуровневый подход – это один из методов решения задачи декомпозиции. ***Декомпозиция*** – *это разбиение одной сложной задачи на несколько более простых задач-модулей*. Все множество модулей разбивают на уровни, которые сформированы следующим образом: для выполнения своих задач модуль обращается с запросом только к модулям нижележащего уровня. А результаты работы могут быть переданы только модулю соседнего вышележащего уровня. Набор функций, который нижележащий уровень предоставляет вышележащему – это **интерфейс**.

Чтобы пояснить понятия **протокол-интерфейс** рассмотрим простой пример.

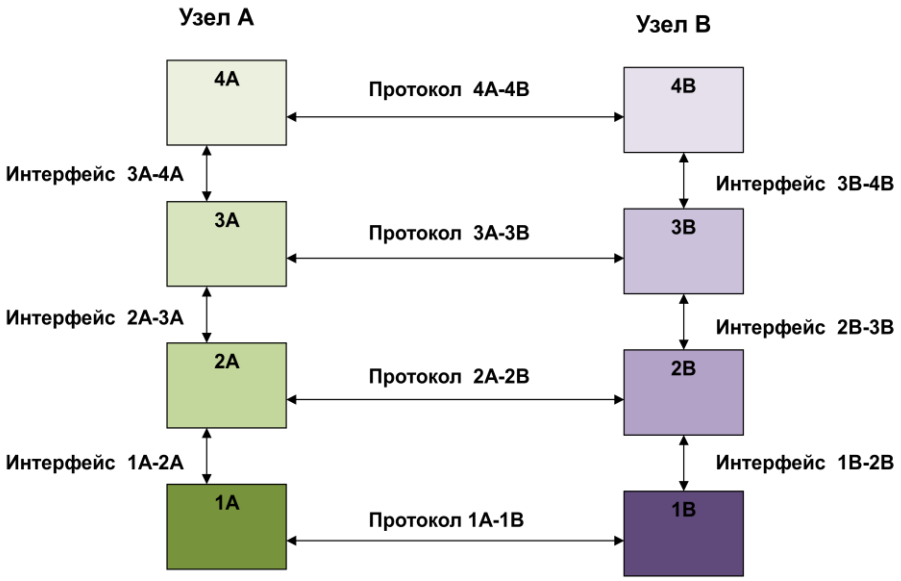
Пример многоуровневого взаимодействия

Есть 2 предприятия А и В, связанные каким-либо деловым сотрудничеством. Между предприятиями существуют многочисленные договоренности и соглашения, например регулярные поставки продукции одного предприятия другому. Начальник предприятия А регулярно (каждый месяц) посылает официальное сообщение начальнику предприятия В о том, сколько и чего доступно на складе. В ответ, начальник В посылает заявку сколько и чего нужно. Это и есть установленный порядок взаимодействия, который называется **протокол**. В данном случае – протокол уровня начальников. Начальники посылают свои заявки не сами, а через своих секретарей. Порядок взаимодействия начальника и секретара –это межуровневый **интерфейс**. Например, на предприятии А обмен идет по электронной почте, а на предприятии В начальник общается с секретарем по телефону. Т.е., интерфейсы начальник-секретарь отличаются. После того, как сообщения переданы секретарям, начальников не волнует, как эти сообщения будут переданы дальше. Это может быть почта, факс, курьер и т.д. Выбор способа передачи – это компетенция секретарей, они решают этот вопрос не уведомляя начальников и связываясь только между собой. Это протокол взаимодействия секретарей. При решении других вопросов начальники могут общаться по другим протоколам-правилам и это не повлияет на работу секретарей, для которых не важно, что отправлять, а важно, чтобы сообщения дошли до адресата.

Мы имеем дело с двумя уровнями – уровнем начальников и уровнем секретарей и каждый из них имеет собственный протокол, который может быть изменен независимо от протокола другого уровня. Эта независимость протоколов и есть основное достоинство многоуровневого подхода.

Протокол – формализованные правила, определяющие последовательность и формат сообщений, которыми обмениваются сетевые компоненты одного уровня в разных узлах.

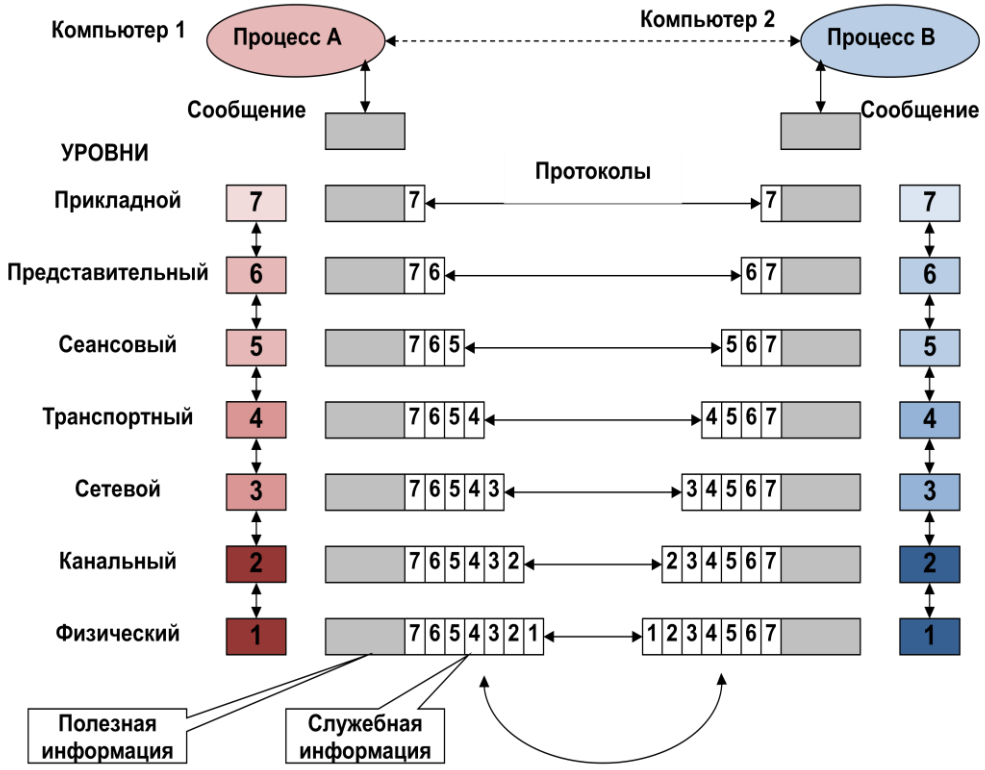
Интерфейс – правила взаимодействия сетевых компонентов соседних уровней одного узла, реализуемые с помощью стандартизованных сообщений.



Взаимодействие двух узлов

Иерархический набор протоколов, достаточный для организации взаимодействия узлов в сети, называется **стеком коммуникационных протоколов**.

Модель OSI



Семиуровневая модель взаимодействия открытых систем

В начале 80-х международная организация по стандартизации ISO разработала эталонную модель архитектуры компьютерных протоколов. Разработчики модели OSI предполагали, что эта модель и протоколы, разрабатываемые в ее рамках, будут доминировать в компьютерной связи и в конце концов вытеснят конкурирующие модели, такие как TCP/IP. Этого не произошло. Хотя в контексте OSI было создано много полезных протоколов, сама модель не получила всеобщего признания. Напротив, доминирующей стала именно TCP/IP. Причина была в том, что на момент разработки OSI аналогичные

протоколы TCP/IP были работоспособными и отлаженными. Тем не менее, рассмотрение этой модели важно для понимания общих принципов многоуровневого подхода.

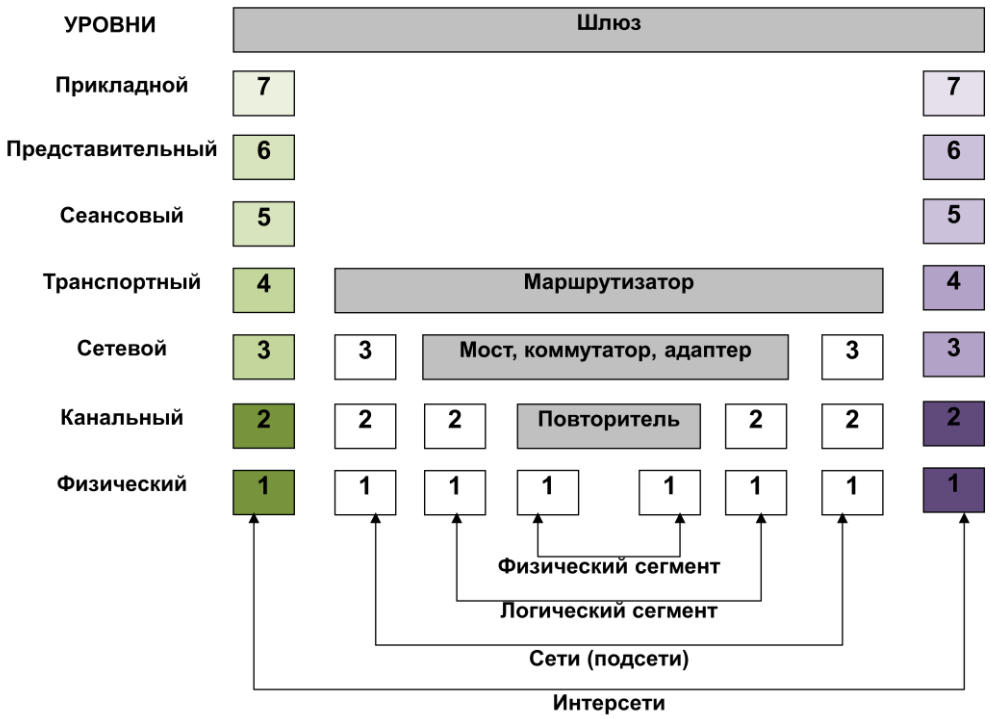
Полное описание модели OSI занимает примерно 1000 страниц текста. Вкратце: в модели OSI семь уровней взаимодействия: прикладной, представительный, сеансовый, транспортный, сетевой, канальный и физический.

Итак, пусть приложение обращается с запросом к прикладному уровню, например, к файловой службе. На основании этого запроса программное обеспечение прикладного уровня формирует сообщение стандартного формата. Обычно сообщение состоит из заголовка и поля данных. В заголовке – служебная информация, которую нужно передать прикладному уровню адресата, чтобы сообщить ему, какую работу нужно выполнить (например, о месте нахождения файла и о типе операции, которую необходимо над ним выполнить). Поле данных может быть пустым или содержать данные, например, те, которые нужно записать в удаленный файл. Но для того, чтобы доставить эту информацию по назначению, предстоит решить еще много задач, ответственность за которые лежит на нижних уровнях.

После формирования сообщения прикладной уровень направляет его вниз по стеку. Протокол представительного уровня добавляет собственную служебную информацию, в котором содержатся указания для представительного протокола машины-адресата. Получившееся сообщение передается вниз сеансовому уровню, который, в свою очередь, добавляет свой заголовок и т.д. Некоторые протоколы помещают служебную информацию не только в начале сообщения, но и в виде т.н. «концевика». Наконец, сообщение достигает нижнего, физического уровня, который собственно и передает его по линиям связи машине-адресату. К этому моменту сообщение «обрастает» заголовками всех уровней.

Когда сообщение по сети поступает на машину-адресат, оно принимается ее физическим уровнем и последовательно перемещается вверх с уровня на уровень. Каждый уровень анализирует и обрабатывает заголовок своего уровня, выполняет свои функции, удаляет свой заголовок и передает оставшееся сообщение (которое для этого уровня трактуется как поле данных) протоколу наверху.

Уровни модели OSI.



Соответствие функций коммуникационных устройств уровням модели OSI

Физический уровень имеет дело с передачей битов по физическим каналам связи, таким, например, как коаксиальный кабель, витая пара, оптоволоконный кабель. К этому уровню имеют отношение характеристики физических сред передачи данных, такие как полоса пропускания, помехозащищенность, волновое сопротивление и другие. На этом же уровне определяются характеристики электрических сигналов, передающих дискретную информацию, например, крутизна фронтов импульсов, уровни напряжения или тока передаваемого сигнала, тип кодирования, скорость передачи сигналов. Кроме этого, здесь стандартизуются типы разъемов и назначение каждого контакта.

Функции физического уровня реализуются во всех устройствах, подключенных к сети. Со стороны компьютера функции физического уровня выполняются сетевым адаптером или последовательным портом.

Примером протокола физического уровня может служить спецификация 10Base-T технологии Ethernet, которая определяет в качестве используемого кабеля неэкранированную витую пару категории 3 с волновым сопротивлением 100 Ом, разъем RJ-45, максимальную длину физического сегмента 100 метров и манчестерский код для представления данных в кабеле.

Канальный уровень

На физическом уровне просто пересылаются биты. При этом не учитывается, что в некоторых сетях, в которых линии связи разделяются несколькими парами взаимодействующих компьютеров, физическая среда передачи может быть занята. Поэтому одна из задач канального уровня - проверка доступности среды передачи. Другая задача канального уровня - обнаружение и коррекция ошибок. Для этого на канальном уровне биты группируются в наборы, называемые **кадрами**. Канальный уровень обеспечивает корректность передачи каждого кадра, помещая специальную последовательность бит в начало и конец каждого кадра, для его выделения, а также вычисляет контрольную сумму, обрабатывая все байты кадра определенным способом и добавляя контрольную сумму к кадру. Канальный уровень может не только обнаруживать ошибки, но и исправлять их за счет повторной передачи поврежденных кадров. Необходимо отметить, что функция исправления ошибок не является обязательной для канального уровня, поэтому в некоторых протоколах этого уровня она отсутствует, например, в Ethernet.

Хотя канальный уровень и обеспечивает доставку кадра между любыми двумя узлами локальной сети, он это делает только в сети с совершенно определенной топологией связей, именно той топологией, для которой он был разработан. К таким типовым топологиям, поддерживаемым протоколами канального уровня локальных сетей, относятся общая шина, кольцо и звезда, а также структуры, полученные из них с помощью мостов и коммутаторов.

В локальных сетях протоколы канального уровня используются компьютерами, мостами, коммутаторами и маршрутизаторами. В компьютерах функции канального уровня реализуются совместными усилиями сетевых адаптеров и их драйверов.

В глобальных сетях, которые редко обладают регулярной топологией, канальный уровень часто обеспечивает обмен сообщениями только между двумя соседними компьютерами, соединенными индивидуальной линией связи. Примером протокола «точка-точка» (как часто называют такие протоколы) могут служить широко распространенный протокол PPP.

Тем не менее для обеспечения качественной транспортировки сообщений в сетях любых топологий и технологий функций канального уровня часто оказывается недостаточно, поэтому в модели OSI решение этой задачи возлагается на два следующих уровня — сетевой и транспортный.

Сетевой уровень.

Функции сетевого уровня достаточно разнообразны. Начнем их рассмотрение на примере объединения локальных сетей.

Протоколы канального уровня локальных сетей обеспечивают доставку данных между любыми узлами только в сети с соответствующей типовой топологией, например топологией иерархической звезды. Это очень жесткое ограничение, которое не позволяет строить сети с развитой структурой, например, сети, объединяющие несколько сетей предприятия в единую сеть, или высоконадежные сети, в которых существуют избыточные связи между узлами. Можно было бы усложнять протоколы

канального уровня, но принцип разделения обязанностей между уровнями приводит к другому решению. Чтобы с одной стороны сохранить простоту процедур передачи данных для типовых топологий, а с другой допустить использование произвольных топологий, вводится дополнительный сетевой уровень.

На сетевом уровне сам термин сеть наделяют специфическим значением. В данном случае под сетью понимается совокупность компьютеров, соединенных между собой в соответствии с одной из стандартных типовых топологий и использующих для передачи данных один из протоколов канального уровня, определенный для этой топологии.

Внутри сети доставка данных обеспечивается соответствующим канальным уровнем, а вот доставкой данных между сетями занимается сетевой уровень, который и поддерживает возможность правильного выбора маршрута передачи. Сети соединяются между собой специальными устройствами, называемыми маршрутизаторами.

Маршрутизатор — это устройство, которое собирает информацию о топологии межсетевых соединений и на ее основании пересылает пакеты сетевого уровня в сеть назначения. Чтобы передать сообщение от отправителя, находящегося в одной сети, получателю, находящемуся в другой сети, нужно совершить некоторое количество транзитных передач между сетями, или хопов (от hop — прыжок), каждый раз выбирая подходящий маршрут. Таким образом, маршрут представляет собой последовательность маршрутизаторов, через которые проходит пакет.

Проблема выбора наилучшего пути называется маршрутизацией, и ее решение является одной из главных задач сетевого уровня. Эта проблема осложняется тем, что самый короткий путь не всегда самый лучший. Часто критерием при выборе маршрута является время передачи данных по этому маршруту; оно зависит от пропускной способности каналов связи и интенсивности трафика, которая может изменяться с течением времени. Некоторые алгоритмы маршрутизации пытаются приспособиться к изменению нагрузки, в то время как другие принимают решения на основе средних показателей за длительное время. Выбор маршрута может осуществляться и по другим критериям, например надежности передачи.

Сетевой уровень решает также задачи согласования разных технологий, упрощения адресации в крупных сетях и создания надежных и гибких барьеров на пути нежелательного трафика между сетями.

Сообщения сетевого уровня принято называть **пакетами.** При организации доставки пакетов на сетевом уровне используется понятие «номер сети». В этом случае адрес получателя состоит из старшей части — номера сети и младшей — номера узла в этой сети (номера хоста). Все узлы одной сети должны иметь одну и ту же старшую часть адреса, поэтому термину «сеть» на сетевом уровне можно дать и другое, более формальное определение: сеть — это совокупность узлов, сетевой адрес которых содержит один и тот же номер сети.

На сетевом уровне определяются два вида протоколов. Первый вид — сетевые протоколы (routed protocols) — реализуют продвижение пакетов через сеть. Именно эти протоколы обычно имеют в виду, когда говорят о протоколах сетевого уровня. Однако часто к сетевому уровню относят и другой вид протоколов, называемых протоколами обмена маршрутной информацией или просто протоколами маршрутизации (routing protocols). С помощью этих протоколов маршрутизаторы собирают информацию о топологии межсетевых соединений. Протоколы сетевого уровня реализуются программными модулями операционной системы, а также программными и аппаратными средствами маршрутизаторов.

На сетевом уровне работают протоколы еще одного типа, которые отвечают за отображение адреса узла, используемого на сетевом уровне, в локальный адрес сети. Такие протоколы часто называют протоколами разрешения адресов — Address Resolution Protocol, ARP. Иногда их относят не к сетевому уровню, а к канальному, хотя тонкости классификации не изменяют их сути.

Примерами протоколов сетевого уровня являются протокол межсетевого взаимодействия IP стека TCP/IP и протокол межсетевого обмена пакетами IPX стека Novell.

Транспортный уровень.

На пути от отправителя к получателю пакеты могут быть искажены или утеряны. Хотя некоторые приложения имеют собственные средства обработки ошибок, существуют и такие, которые предпочитают сразу иметь дело с надежным соединением. Транспортный уровень обеспечивает приложениям или верхним уровням стека — прикладному и сеансовому — передачу данных с той степенью надежности, которая им требуется.

Например, модель OSI определяет пять классов сервиса, предоставляемых транспортным уровнем. Эти виды сервиса отличаются качеством предоставляемых услуг: срочностью, возможностью восстановления прерванной связи, наличием средств мультиплексирования нескольких соединений между различными прикладными протоколами через общий транспортный протокол, а главное — способностью к обнаружению и исправлению ошибок передачи, таких как искажение, потеря и дублирование пакетов.

Как правило, все протоколы, начиная с транспортного уровня и выше, реализуются программными средствами конечных узлов сети — компонентами их сетевых операционных систем. В качестве примера транспортных протоколов можно привести протоколы TCP и UDP стека TCP/IP и протокол SPX стека Novell.

Протоколы нижних четырех уровней обобщенно называют сетевым транспортом или транспортной подсистемой, так как они полностью решают задачу транспортировки сообщений с заданным уровнем качества в составных сетях с произвольной топологией и различными технологиями. Остальные три верхних уровня решают задачи предоставления прикладных сервисов на основании имеющейся транспортной подсистемы.

Сеансовый уровень обеспечивает управление диалогом: фиксирует, какая из сторон является активной в настоящий момент, предоставляет средства синхронизации, которые позволяют вставлять контрольные точки в длинные передачи, чтобы в случае отказа можно было вернуться назад к последней контрольной точке, а не начинать все с начала. На практике немногие приложения используют сеансовый уровень, и он редко реализуется в виде отдельных протоколов, хотя функции этого уровня часто объединяют с функциями прикладного уровня и реализуют в одном протоколе.

Представительный уровень имеет дело с формой представления передаваемой по сети информации, не меняя при этом ее содержания. За счет уровня представления информация, передаваемая прикладным уровнем одной системы, всегда понятна прикладному уровню другой системы. С помощью средств данного уровня протоколы прикладных уровней могут преодолеть синтаксические различия в представлении данных или же различия в кодах символов. На этом уровне может выполняться шифрование и дешифрование данных, благодаря которому секретность обмена данными обеспечивается сразу для всех прикладных служб. Примером такого протокола является протокол Secure Socket Layer (SSL), который обеспечивает секретный обмен сообщениями для протоколов прикладного уровня стека TCP/IP.

Прикладной уровень — это в действительности просто набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам, таким как файлы, принтеры или гипертекстовые Web-страницы, а также организуют свою совместную работу, например, с помощью протокола электронной почты. Единица данных, которой оперирует прикладной уровень, обычно называется **сообщением (message).**

Три нижних уровня — физический, канальный и сетевой — являются сетезависимыми, то есть протоколы этих уровней тесно связаны с технической реализацией сети и используемым коммуникационным оборудованием.

Три верхних уровня — прикладной, представительный и сеансовый — ориентированы на приложения и мало зависят от технических особенностей построения сети.

Транспортный уровень является промежуточным, он скрывает все детали функционирования нижних уровней от верхних. Это позволяет разрабатывать приложения, не зависящие от технических средств непосредственной транспортировки сообщений.

Физический уровень
- обеспечивает передачу неструктурированного потока битов по физическому носителю
- имеет дело с механическими, электрическими, функциональными и процедурными характеристиками доступа к физической линии связи

Канальный уровень или уровень передачи данных
- обеспечивает надежную передачу данных по физической линии
- посылает блоки (кадры) с необходимой синхронизацией, контролем ошибок и управлением потоком

Сетевой уровень
- обеспечивает независимость верхних уровней от технологий передачи данных и коммутации
- отвечает за установку и разрыв соединений и за управление соединениями

Транспортный уровень
- обеспечивает надежный и прозрачный перенос данных между конечными узлами
- обеспечивает сквозное восстановление после ошибок и управление потоком

Сеансовый уровень
- предоставляет структуру управления для взаимодействия приложений
- устанавливает и разрывает сеансы между приложениями

Представительный уровень
- обеспечивает прикладному процессу независимость от синтаксиса представления данных

Прикладной уровень
- обеспечивает доступ пользователей
- предоставляет распределенные информационные службы

Различия локальных и глобальных сетей

В последнее время эти отличия становятся все менее заметные, тем не менее они существуют.

Протяженность, качество и способ прокладки линий связи. Локальные сети по определению отличаются от глобальных небольшим расстоянием между узлами сети. Это в принципе делает возможным использование в локальных сетях качественных линий связи: коаксиального кабеля, витой пары, оптоволоконного кабеля, которые не всегда доступны (из-за экономических ограничений) на больших расстояниях, в глобальных сетях. В глобальных сетях часто применяются уже существующие линии связи (например, телефонные).

Сложность методов передачи и оборудования. В условиях низкой надежности физических каналов в глобальных сетях требуются более сложные, чем в локальных сетях, методы передачи данных и соответствующее оборудование. Так, в глобальных сетях широко применяются модуляция, асинхронные методы, сложные методы контрольного суммирования, квитирование и повторные передачи искаженных кадров. С другой стороны, качественные линии связи в локальных сетях позволили упростить процедуры передачи данных за счет применения немодулированных сигналов и отказа от обязательного подтверждения получения пакета (квитирования).

Скорость обмена данными. Одно из главных отличий локальных сетей от глобальных - высокоскоростные каналы обмена данными между компьютерами, скорость которых (10, 100 Мбит/с) сравнима со скоростями работы устройств и узлов компьютера — дисков, внутренних шин обмена данными и т. п. За счет этого у пользователя локальной сети, подключенного к удаленному разделяемому ресурсу (например, диску сервера), складывается впечатление, что он пользуется этим

диском, как «своим». Для глобальных сетей типичны гораздо более низкие скорости передачи данных — 28800, 33600 бит/с, 56 и 64 Кбит/с и только на магистральных каналах — до 2 Мбит/с.

Разнообразие услуг. Локальные сети предоставляют, как правило, широкий набор услуг — это различные виды услуг файловой службы, услуги печати, услуги службы передачи факсимильных сообщений, услуги баз данных, электронная почта и другие, в то время как глобальные сети в основном предоставляют почтовые услуги и иногда файловые услуги с ограниченными возможностями — передачу файлов из публичных архивов удаленных серверов без предварительного просмотра их содержания.

Оперативность выполнения запросов. Время прохождения пакета через локальную сеть обычно составляет несколько миллисекунд, время же его передачи через глобальную сеть может достигать нескольких секунд. Низкая скорость передачи данных в глобальных сетях затрудняет реализацию служб для режима on-line, который является обычным для локальных сетей.

Разделение каналов. В локальных сетях каналы связи используются, как правило, совместно сразу несколькими узлами сети, а в глобальных сетях — индивидуально.

Использование метода коммутации пакетов. Важной особенностью локальных сетей является неравномерное распределение нагрузки. Отношение пиковой нагрузки к средней может составлять 100:1 и даже выше. Такой трафик обычно называют пульсирующим. Из-за этой особенности трафика в локальных сетях для связи узлов применяется метод коммутации пакетов, который для пульсирующего трафика оказывается гораздо более эффективным, чем традиционный для глобальных сетей метод коммутации каналов. Эффективность метода коммутации пакетов состоит в том, что сеть в целом передает в единицу времени больше данных своих абонентов.

Масштабируемость. «Классические» локальные сети обладают плохой масштабируемостью из-за жесткости базовых топологий, определяющих способ подключения станций и длину линии. При использовании многих базовых топологий характеристики сети резко ухудшаются при достижении определенного предела по количеству узлов или протяженности линий связи. Глобальным же сетям присуща хорошая масштабируемость, так как они изначально разрабатывались в расчете на работу с произвольными топологиями.

Главное требование, предъявляемое к сетям – это предоставлять пользователям потенциальную возможность доступа к разделяемым ресурсам всех компьютеров, объединенных в сеть.

Все остальные требования – связаны с качеством выполнения этой задачи. Хотя все эти требования весьма важны, часто понятие «качество обслуживания» трактуется более узко – в него включаются только две самые важные характеристики сети – производительность и надежность.

Производительность характеризуется следующими параметрами:

- Время реакции сети – интегральная характеристика. Именно она имеется в виду, когда пользователь говорит «Сегодня сеть работает медленно». В общем случае определяется как интервал времени между возникновением запроса к сетевой службе и получением ответа на этот запрос. Значение этого показателя зависит и от типа службы, к которой идет обращение и от текущего состояния элементов сети, загруженность коммутаторов, маршрутизаторов и т.д. Поэтому имеет смысл использовать средневзвешенную оценку загруженности, усредняя этот показатель по пользователям, серверам, времени суток и т.д.
- Пропускная способность – объем данных, переданный в единицу времени.
    - Средняя пропускная способность. Отношение общего объема переданных данных к времени передачи (час, день, неделя).
    - Мгновенная пропускная способность (10 мс –1 сек)
    - Максимальная пропускная способность – наибольшая мгновенная за интервал измерения.
    - Общая пропускная способность сети – количество информации, переданное между всеми узлами сети за время наблюдения.

- Задержка передачи – это параметр по смыслу близок ко времени реакции, но характеризует только сетевые этапы обработки данных, без задержек обработки компьютерами сети. Т.е., это задержка между временем поступления пакета на вход сетевого устройства и временем появления на выходе.

Пропускная способность и задержка – величины независимые. Сеть может иметь высокую ПС, но вносить большие задержки при передаче каждого пакета. Пример – спутниковая связь. Пропускная способность – 2 Mbit/s, а задержка передачи не менее 0,24 с, что определяется длиной канала в 72000 км.

Надежность и безопасность сети характеризуется:

- Коэффициентом готовности – это доля времени, в течение которого система может быть использована. Готовность может быть улучшена путем введения избыточности в структуру системы. Ключевые элементы должны существовать в нескольких экземплярах. Чтобы систему можно было отнести к высоконадежным, она должна как минимум обладать высокой готовностью, но этого недостаточно. Необходимо обеспечить
- Сохранность данных и защиту их от искажений. Кроме того, должна поддерживаться согласованность данных, например при резервировании данных на нескольких файловых серверах (ресурсах) должна постоянно обеспечиваться их идентичность.
- Вероятность доставки пакета без искажений либо другие показатели, напр. Вероятность потери пакета
- Безопасность от несанкционированного доступа. Это другой аспект надежности.
- Отказоустойчивость. Способность системы скрыть от пользователя выход из строя отдельных элементов. В отказоустойчивой системе отказ одного из элементов приводит к некоторому снижению качества работы, а не к полному останову. Так при отказе одного из файловых серверов увеличивается только время доступа к базе данных из-за уменьшения степени распараллеливания запросов, но в целом система будет выполнять свои функции.

Расширяемость и масштабируемость.

- Расширяемость означает возможность добавления отдельных элементов сети (пользователей, компьютеров, приложений, служб), замены аппаратуры, наращивания сегментов сети. Однако расширение всегда ограничивается некоторыми пределами. Например, локальная сеть Ethernet, построенная на коаксиальном кабеле обладает хорошей расширяемостью, в смысле что позволяет легко подключить новые станции. Однако такая сеть имеет ограничения – при числе компьютеров 30-40 штук она перестает корректно работать. Наличие такого ограничения – признак плохой масштабируемость системы при хорошей расширяемости.
- Масштабируемость говорит о том, что сеть позволяет наращивать количество узлов и протяженности связей без ухудшения производительности. Для обеспечения масштабируемости чаще всего приходится применять дополнительное коммуникационное оборудование.

Прозрачность сети достигается в том случае, когда сеть представляется пользователю не как множество отдельных компьютеров, связанной сложной системой оборудования, а как единая вычислительная машина. Известный лозунг компании Sun Microsystems: «Сеть – это компьютер».

Прозрачность может быть достигнута на двух различных уровнях — на уровне пользователя и на уровне программиста. На уровне пользователя прозрачность означает, что для работы с удаленными ресурсами он использует те же команды и привычные ему процедуры, что и для работы с локальными ресурсами. На программном уровне прозрачность заключается в том, что приложению для доступа к удаленным ресурсам требуются те же вызовы, что и для доступа к локальным ресурсам. Прозрачность на уровне пользователя достигается проще, так как все особенности процедур, связанные с распределенным характером системы, маскируются от пользователя программистом, который создает приложение. Прозрачность на уровне приложения требует сокрытия всех деталей распределенности средствами сетевой операционной системы.

Сеть должна скрывать все особенности операционных систем и различия в типах компьютеров. Пользователь компьютера Macintosh должен иметь возможность обращаться к ресурсам, поддерживаемым UNIX-системой, а пользователь UNIX должен иметь возможность разделять информацию с пользователями Windows. Подавляющее число пользователей ничего не хочет знать о внутренних форматах файлов или о синтаксисе команд UNIX.

В настоящее время нельзя сказать, что свойство прозрачности в полной мере присуще многим вычислительным сетям, это скорее цель, к которой стремятся разработчики современных сетей.

Поддержка разных видов трафика.

Главной особенностью трафика, образующегося при динамической передаче голоса или изображения, является наличие жестких требований к синхронности передаваемых сообщений. При запаздывании сообщений будут наблюдаться искажения. В то же время трафик компьютерных данных характеризуется крайне неравномерной интенсивностью поступления сообщений в сеть при отсутствии жестких требований к синхронности доставки этих сообщений. Например, доступ пользователя, работающего с текстом на удаленном диске, порождает случайный поток сообщений между удаленным и локальным компьютерами, зависящий от действий пользователя по редактированию текста, причем задержки при доставке в определенных (и достаточно широких с компьютерной точки зрения) пределах мало влияют на качество обслуживания пользователя сети. Все алгоритмы компьютерной связи, соответствующие протоколы и коммуникационное оборудование были рассчитаны именно на такой «пульсирующий» характер трафика, поэтому необходимость передавать мультимедийный трафик требует внесения принципиальных изменений как в протоколы, так и оборудование.

Особую сложность представляет совмещение в одной сети традиционного компьютерного и мультимедийного трафика. Передача исключительно мультимедийного трафика компьютерной сетью хотя и связана с определенными сложностями, но вызывает меньшие трудности. А вот случай сосуществования двух типов трафика с противоположными требованиями к качеству обслуживания является намного более сложной задачей. Обычно протоколы и оборудование компьютерных сетей относят мультимедийный трафик к факультативному, поэтому качество его обслуживания оставляет желать лучшего. Сегодня затрачиваются большие усилия по созданию сетей, которые не ущемляют интересы одного из типов трафика. Наиболее близки к этой цели сети на основе технологии ATM.

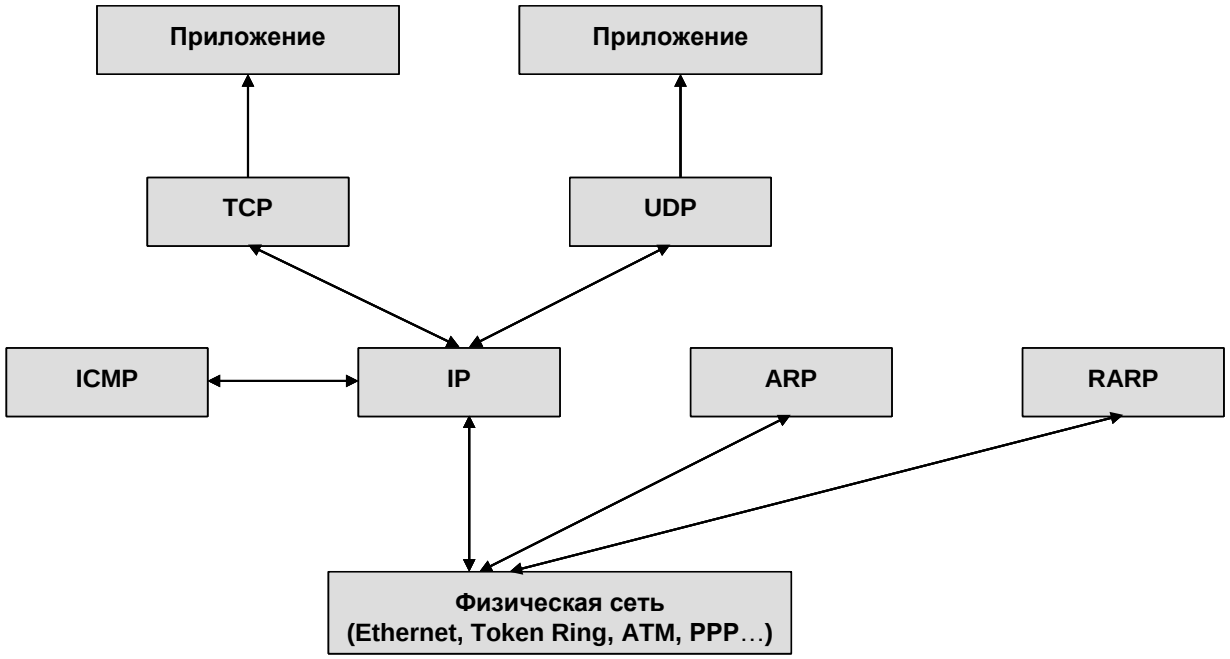
Управляемость.

Управляемость сети подразумевает возможность централизованно контролировать состояние основных элементов сети, выявлять и разрешать проблемы, возникающие при работе сети, выполнять анализ производительности и планировать развитие сети.

Полезность системы управления особенно ярко проявляется в больших сетях: корпоративных или публичных глобальных. Большинство существующих средств вовсе не управляют сетью, а всего лишь осуществляют наблюдение за ее работой. Они следят за сетью, но не выполняют активных действий, если с сетью что-то произошло или может произойти.

Совместимость.

Совместимость или интегрируемость означает, что сеть способна включать в себя самое разнообразное программное и аппаратное обеспечение, то есть в ней могут сосуществовать различные операционные системы, поддерживающие разные стеки протоколов, и работать аппаратные средства и приложения от разных производителей. Сеть, состоящая из разнотипных элементов, называется неоднородной или гетерогенной, а если гетерогенная сеть работает без проблем, то она является интегрированной. Основной путь построения интегрированных сетей — использование модулей, выполненных в соответствии с открытыми стандартами и спецификациями.



Состав стека протоколов TCP/IP

Архитектура сетей TCP/IP.

В состав стека протоколов TCP/IP, кроме давших название этим сетям протоколов Transmission Control Protocol и Internet Protocol, входят: User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP), ARP (Address Resolution Protocol), RARP (Reverse Address Resolution Protocol) и ряд протоколов прикладного уровня, в частности TELNET, FTP, SMTP, SNMP и HTTP.

В отличие от 7-ми уровневой модели OSI протокольный стек TCP/IP разбит на 4 уровня. Сетевой уровень (IP) обеспечивает передачу информации через произвольную комбинацию сетей, использующих этот же набор протоколов.

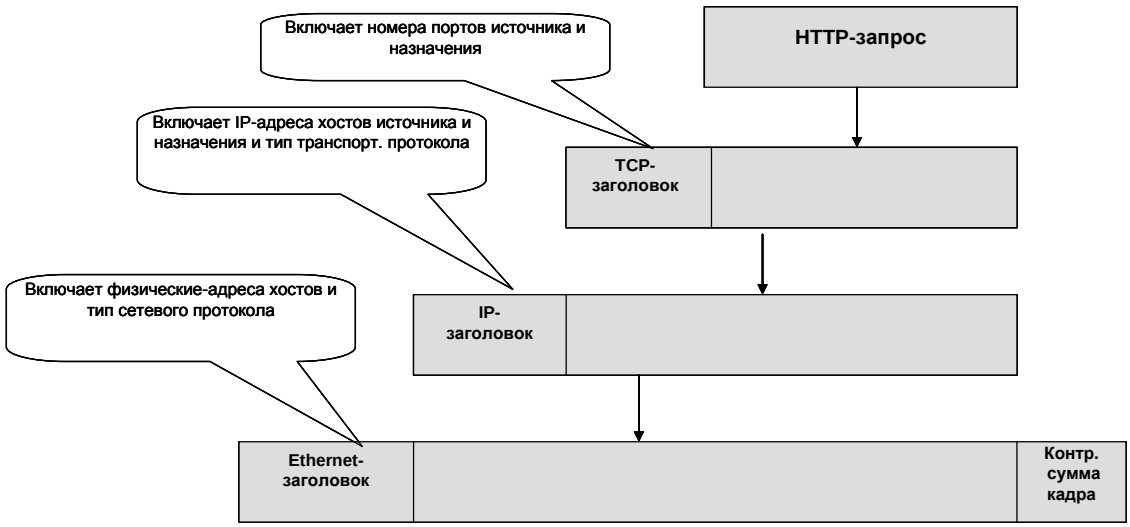
IP-протокол предоставляет лишь один вид сервиса – передачу пакетов без предварительного установления соединения и настолько хорошо, насколько получится (best-effort connectionless packet transfer). Пакеты пересылаются между узлами коммутации без предварительного установления соединения; они маршрутизируются независимо, и пакеты одного приложения могут доставляться по разным маршрутам. Узлы коммутации, соединяющие сети, могут испытывать перегрузки и уничтожать пакеты. Ответственность за восстановление утерянных пакетов и надлежащий порядок их передачи приложению лежит на транспортном уровне, который представлен протоколами TCP и UDP.

Разнообразие требований сетевых приложений обусловило необходимость двух протоколов транспортного уровня. Так, например, приложения передачи файлов и web (FTP, HTTP) для пересылки своих сообщений используют TCP, в то время как приложения управления сетевыми устройствами, служба имен (DNS), потоковые приложения реального времени используют в качестве транспортного протокола UDP. Далее будем называть протокольные блоки TCP **сегментами**, а блоки UDP – **дейтограммами**.

Сетевой уровень (протокол IP) мультиплексирует протокольные блоки транспортного уровня в IP-потоки; при этом сегменты транспортного уровня могут фрагментироваться (если они превышают максимально допустимый размер). Протокольные блоки IP обычно называют **пакетами**.

После вычисления маршрута передачи пакета, посредством протокола ARP определяется физический адрес следующего на маршруте хоста и пакет направляется на физический уровень сети. Иногда бывает необходимо решить обратную задачу, то есть по заданному физическому адресу определить логический сетевой адрес устройства. В частности, эта проблема актуальна для процедуры загрузки бездисковых станций, когда такая станция рассылает в широковещательном режиме запрос, содержащий ее физический адрес и «просьбу» сообщить ей логический сетевой адрес. Этот запрос обрабатывается сервером RARP, который и передает пославшей его станции требуемую информацию.

Физический уровень TCP/IP сети может использовать любую технологию канального уровня – Ethernet, Token Ring, ATM, PPP и т.д.



Инкапсуляция протокольных блоков в TCP/IP стеке

Протокольные блоки вышележащих уровней инкапсулируются в протокольные блоки нижележащих уровней. При этом, блок каждого уровня содержит специфическую информацию, позволяющую точно адресовать его. Так, сегмент TCP (UDP-дейтограмма) содержит в своем заголовке номер **порта**, однозначно определяющий приложение, которому он принадлежит. IP-пакет адресуется **логическим адресом** хоста, также однозначно определяющим его в распределенной сети. Кадр канального уровня включает в себя **физический адрес** ближайшего на маршруте доставки хоста, которому необходимо передать пакет.

IP-протокол

Основной протокол стека TCP/IP, - Internet Protocol (IP), - по своим функциям соответствует сетевому уровню модели OSI. Механизмы протокола обеспечивают ненадежную доставку пакетов данных между сетевыми устройствами (устройствами, имеющими сетевой адрес) в режиме без предварительного установления соединения (дейтограммный сервис). Этот тип сервиса часто называют сервисом «настолько хорошо, насколько возможно» (best effort service), что отражает отсутствие в протоколе процедур контроля доставки пакетов. Решение задачи надежности доставки возлагается на протоколы верхних уровней, главным образом на TCP. Основными функциями протокола IP являются:

- формирование пакетов из сегментов транспортного уровня, с их предварительной фрагментацией (если необходимо);
- обеспечение логической адресации сетевых устройств
- поддержка процесса маршрутизации
- продвижение пакетов от одного узла коммутации до другого.

Структура IP-пакета.

Функции IP-протокола реализуются посредством специальной структуры заголовка пакета.

Заголовок содержит поля фиксированной длины (первые 20 байт) и поле переменной длины (поле опций), размер которого может достигать 40 байт. Для выравнивания этого поля по 32 битной границе

предусмотрено поле «Выравнивание» (Padding) которое заполняется нулями. Рассмотрим назначение полей заголовка.

Формат заголовка IP-пакета

Версия (Version) – поле определяет номер версии протокола. В настоящее время используется версия 4 и ведется активная подготовка к переходу на версию 6. Версия 5 описывает протокол ST2, разработанный для передачи данных потоковых приложений реального времени. Поле проверяется перед обработкой пакета и пакеты несогласующейся с протокольным стеком приемника версией, отбрасываются. Одновременно, включение версии в каждую дейтограмму позволяет использовать разные, но согласующиеся, версии на разных хостах.

Длина заголовка (Internet Header Length, IHL) - Поле определяет длину заголовка, измеренную в 32-битных словах. Корректный заголовок имеет длину не менее 5 слов. Длина поля опций (в 32-разрядных словах) может быть определена как значение этого поля минус 5.

Тип сервиса (Type of service, ToS) – определяет тип требуемого обслуживания пакета. Первые три бита задают уровень приоритета обслуживания (0-7), 3-5 биты определяют требования к задержке (какая получится, низкая), уровень пропускной способности (обычный, высокий) и надежность доставки (какая получится, высокая). Практически, большая часть маршрутизаторов игнорирует данные этого поля. Однако в настоящее время в связи с разработкой механизмов обеспечения в IP-сетях служб с гаранированным качеством обслуживания делаются попытки использования значений этого поля.

Общая длина (Total length) - поле содержит общую длину пакета, размер которого не может превышать 65535 байт. Практически пакеты такой длины никогда не используются, поскольку технологии канального уровня накладывают свои ограничения. Так, Ethernet не допускает кадров с длиной более 1500 байт, FDDI – 4096 байт и т.д. В этой связи, протокол IP выполняет фрагментацию сегментов данных, поступающих к нему от TCP и UDP протоколов. Следует отметить, что маршрутизатор не выполняет сборку пакетов, даже если следующая сеть имеет параметр MTU (Maximum Transmission Unit), допускающий более крупные пакеты. Сборка пакетов в исходный сегмент производится на месте назначения.

Поля «**Идентификатор**», **«Флаги**» и «**Смещение фрагмента**» управляют процессом сборки сегмента.

Время жизни (Time to live, TTL) - поле, определяющее максимальное время, которое пакет может существовать в сети. Значение этого поля (в секундах) устанавливается при отправке пакета и уменьшается на единицу по мере прохождения им маршрутизаторов. При достижении нулевого значения этого поля пакет уничтожается. Максимальное значение поля – 255 секунд. Этот механизм помогает избежать перегрузок сети при возникновении ошибок в таблицах маршруизации, приводящих к образованию петель.

Протокол (Protocol) – поле указывает модулю какого протокола (TCP, UDP, ICMP) передать полученный IP-пакет.

Контрольная сумма (Header checksum) – поле содержит значение контрольной суммы**,** рассчитанной только по заголовку. Поскольку значения некоторых полей заголовка изменяются по мере прохождения пакета по маршруту (поле TTL, например), то значения рассматриваемого поля проверяются и пересчитываются на каждом маршрутизаторе. Этот механизм является единственным средством обеспечения достоверности передачи, содержащимся в протоколе IP.

Адрес отправителя (Source IP address) и **Адрес получателя (Destination IP address)** – поля одинаковой длины (32 бита), содержащие соответствующие адреса. Правила адресации в IP-сетях будут рассмотрены далее.

Опции (Options) – необязательное поле, используемое при отладке сетей и для запроса определенных специфических процедур обработки. В настоящее время используется крайне редко. В связи с разработкой новых протоколов, обеспечивающих большую гибкость в обработке IP-трафика, возможность использования этих полей вновь стала предметом обсуждения комитетов по стандартизации.

При получении пакета маршрутизатор вычисляет контрольную сумму заголовка пакета и, если она не совпадает со значением поля «Контрольная сумма», то пакет отбрасывается. При положительном результате проверки, производится изменение некоторых полей и рассчитывается новое значение поля «Контрольная сумма». Затем по таблице маршрутизации определяется адрес следующего маршрутизатора, на который должен быть направлен этот пакет, и он передается на соответствующий интерфейс.

Адресация в сетях IP. Разбиение на подсети

Классификация IP адресов

Для идентификации каждого компьютера в IP-сети необходима система их адресации. При этом учитывается, что сетевые устройства (компьютер, маршрутизатор и т.д.) могут иметь несколько сетевых интерфейсов, и каждый из них должен иметь уникальный адрес. Если обратиться к аналогии обычной адресной системы (улица, дом, квартира), то становится ясной целесообразность построения системы сетевой адресации по иерархии «сеть-интерфейс». Принятая в сетях IP система адресации описана в документах RFC 990 и RFC 997. Каждое сетевое устройство имеет адреса трех типов:

1. Физический адрес узла, определяемый используемой технологией канального уровня. Для Ethernet – это MAC-адрес его сетевой карты, назначаемый фирмой-производителем. Он представляет собой шести-байтовое число, первые три байта которого однозначно определяют фирму-производителя, а последние три байта – уникальны для каждой карточки, произведенной в рамках данной фирмы.
2. IP-адрес, состоящий из 4-х байтов, и также являющийся совершенно уникальным.
3. Символьный идентификатор – имя, назначаемое по определенным правилам и являющееся полным эквивалентом IP-адреса.

IP-адрес строится по двухуровневой иерархии, т.е. он объединяет в себе адрес сети и адрес хоста. Разделение сетевого адреса на 2 части имеет большой практический смысл, ибо позволяет магистральным маршрутизаторам существенно сократить размер своих таблиц коммутации, формируя их на основании только сетевой части адреса назначения. Для удовлетворения потребностей адресации сетей различного масштаба были введены несколько классов сетей, отличающиеся размером полей, отводимых для указания номера сети и номера хоста. При этом, размер поля полного адреса всегда равен 32 битам. Структура адресов сетей разных классов приведена на рисунке.

IP-адрес обычно записывается в форме 4-х трехразрядных десятичных чисел, разделенных точкой. Каждое из этих десятичных чисел соответствует одному байту двоичного представления адреса. Так, например, адрес (в десятичном представлении) имеющий вид 128.135.68.5., принадлежит сети класса **B,** т.к. в этом случае первые два бита адреса – 10, и, следовательно, левые 16 бит являются адресом сети, а правые16 бит – адресом хоста.

Некоторые адреса являются зарезервированными и не могут присваиваться хостам. Так, адрес 127.x.x.x (x – означает любое число, обычно 0) зарезервирован для обратной связи, используемой при тестировании взаимодействия процессов на одной сетевой станции. Когда приложение использует этот адрес в качестве адреса назначения, стек TCP/IP данного хоста возвращает данные приложению, ничего не передавая на физический интерфейс. Поэтому адреса, начинающиеся на 127, запрещается присваивать сетевым устройствам. Другим зарезервированным адресом является, так называемый, широковещательный адрес, содержащий 1, или 0, во всех своих битах. Пакет с адресом назначения 255.255.255.255 или 1.1.1.1 будет доставлен всем устройствам сети, к которой принадлежит узел-отправитель, но маршрутизаторы такие пакеты не обрабатывают. Существует и направленное широковещание – способ адресации, при котором один пакет, отосланный в определенную сеть, будет доставлен всем ее хостам. Такой пакет должен содержать корректный адрес сети и иметь все биты адреса хоста равными 1. Так, например, пакет с адресом 184.90.255.255 будет доставлен всем станциям сети класса **B**, имеющей адрес 184.90.

Значения первых четырех битов адреса однозначно определяют обе границы его сетевой части. Максимальное число сетей класса **A** равно $2^6+2^5+ \ldots.+2^0 -1 =126$ (сетевой префикс, состоящий из одних нулей недопустим). Каждая сеть класса **A** может содержать $2^{24}-2 = 16\ 777\ 214$ устройств. В целом, адресный блок сетей класса **A** занимает около 50% общего адресного пространства. В таблице содержатся аналогичные показатели для сетей класса **B** и **C**.

Класс сетей	Значение первого байта адреса	Количество сетей	Количество хостов в сети класса	Удельный вес класса в адресном пространстве, %
A	001 – 126	126	16 777 214	50
B	128 – 191	16 384	65 534	25
C	192 - 223	2 097 152	254	12,5

Рассмотренная выше двухуровневая схема адресации оказалась недостаточно гибкой и в 1985 году была определена трехуровневая схема адресации (RFC 950). Необходимость перехода к такой системе адресации можно проиллюстрировать следующим примером. Организация получила сеть класса **B**, позволяющую адресовать 65000 хостов. Пока в сети работали относительно немного станций – она функционировала благополучно. Но с ростом числа активных хостов неизбежно рос и широковещательный трафик, поскольку этот тип пакетов активно используется в служебных целях многими протоколами. Это обстоятельство неизбежно вело к снижению эффективной производительности сети. Кроме того, управление несколькими десятками тысяч подключений из единого административного центра представляет собой очень трудную задачу. К этим проблем следует добавить и то, что любая организация развивается в направлении роста самостоятельности своих подразделений, что также неизбежно должно отражаться и в структуре сети. Возможность же структурирования сети организации за счет получения дополнительных номеров сетей была быстро исчерпана и резкая нехватка адресного пространства стала реальностью еще во второй половине 80-х годов. Кроме того, рост числа используемых сетей вел к росту таблиц маршрутизации магистральных (межсетевых) маршрутизаторов и, следовательно, к снижению их производительности.

Перечисленные проблемы в значительной степени связаны с проблемой масштабируемости сети, и они были разрешены введением в иерархию адресации третьего уровня. Этот уровень, получивший название «уровень подсети», был выделен в пространстве адресов хоста.

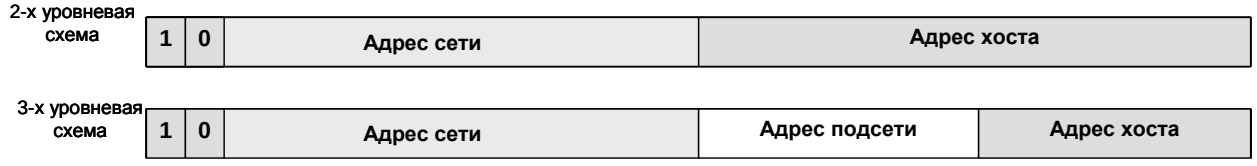


Схема адресации с введением подсетей

	Подсеть	Маска	Эквивалентный адрес подсети	Число хостов в подсети
Исходная сеть	**193.10.1.0**	**255.255.255.0**	**193.10.1.0/24**	**254**
Подсеть 1	**193.10.1.0**	**255.255.255.128**	**193.10.1.0/25**	**126**
Подсеть 2	**193.10.1.128**	**255.255.255.192**	**193.10.1.128/26**	**62**
Подсеть 3	**193.10.1.192**	**255.255.255.224**	**193.10.1.192/27**	**30**
Подсеть 4	**193.10.1.224**	**255.255.255.224**	**193.10.1.224/27**	**30**

При этом поля адреса сети и адреса подсети в совокупности называют расширенным сетевым префиксом. В рассмотренном выше примере, выделение в пространстве адреса хоста 6 разрядов для адреса подсети позволяет организовать 62 подсети, содержащих до 1022 хостов каждая. Такое разбиение на подсети не требует согласования со специальным служебным органом Интернет, регулирующим распределение адресного пространства.

Отрицательным следствием введения подсетей стало усложнение процедуры определения адреса хоста. Действительно, сетевые устройства используют старшие биты сетевого адреса для определения класса сети, после чего, в случае двухуровневой иерархии, легко находится граница между битами сетевого адреса и адреса хоста. В трехуровневой системе адресации этот прием работать не сможет. Для определения адреса подсети и адреса хоста потребовалось ввести **сетевую маску –** 32-битный адрес, в котором все биты, соответствующие битам расширенного сетевого префикса, установлены в 1, а соответствующие битам адреса хоста – в 0. Так, например, пусть необходимо в сети класса **B** 132.10 выделить подсеть, содержащую не более 100 хостов. Для адресации такого числа сетевых устройств достаточно располагать 7 битами (2^7 =128), которые и отводятся для адреса хоста; оставшиеся 9 бит отводятся для номера подсети (их можно организовать 2^9 -2=510) а старшие 16 бит – это адрес сети.

Таким образом, маска подсети, в которой находятся хосты с адресами:

132.10.12.129, 132.10.12.130,….,132.10.12.254

будет иметь вид

11111111 11111111 11111111 10000000 (255.255.255.128).

Если маршрутизатор получит пакет с адресом назначения

10000100 00001010 00001100 10110000 (132.10.12.176)

и выполнит по отношению к нему и сетевой маске операцию логического «И», то результат будет содержать номер подсети. В данном случае получаем:

10000100 00001010 00001100 10000000,

что соответствует адресу подсети 132.10.12.128. Далее, по таблице маршрутизации будет найден следующий узел на пути к этой подсети, и пакет отправится на соответствующий интерфейс.

Необходимо заметить, что информация о маске подсети IP-пакетом не переносится (хост-источник о структуре сети, в которой находится получатель, ничего не знает). Передача этой информации является задачей протоколов маршрутизации, или она задается статически при конфигурировании маршрутизатора.

В стандартах, описывающих современные протоколы маршрутизации, часто делается ссылка на длину расширенного префикса сети, а не на маску подсети. В такой записи адрес устройства в рассмотренном выше примере имеет вид 132.10.12.176/25. В качестве примера нумерации подсетей в таблице приведено разбиение сети класса **C** на подсети. Указанное в таблице число хостов в каждой из подсетей на два меньше возможного числа адресов, поскольку адреса, в которых все биты поля адреса хоста установлены в единицу и в ноль, являются зарезервированными и используются как широковещательные для данной подсети. Так при подсетях, приведенных в таблице., адрес 193.10.1.0 является широковещательным лишь для подсети 193.10.1.0/24, а не для всех подсетей. Аналогично, адрес 193.10.1.255 является широковещательным только для подсети 193.10.1.224/27.

Введение подсетей, решив проблемы масштабирования адресного пространства, потребовало определенного усложнения протоколов маршрутизации, которые должны обрабатывать (и переносить)

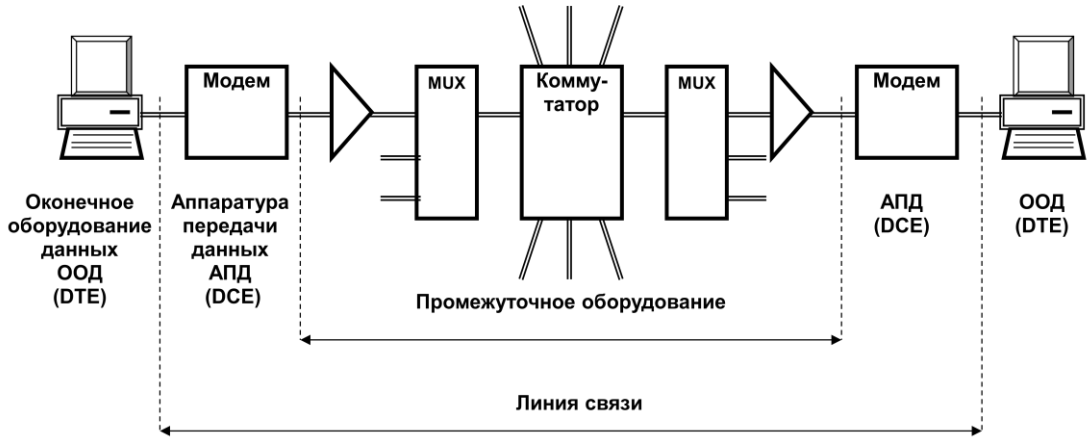
не только адрес сетевого устройства, но и его маску. В настоящее время все широко используемые протоколы маршрутизации (RIP-2, IS-IS, OSPF) переносят эту информацию.

Основы передачи дискретных данных

Все сетевые технологии при всех их отличиях базируются на общих принципах передачи дискретных данных. Эти принципы находят свое воплощение в:

- методах представления двоичных сигналов (т.е. нулей и единиц) с помощью импульсных или синусоидальных сигналов в линиях связи различной физической природы,
- методах обнаружения и коррекции ошибок,
- методах компрессии и
- методах коммутации.

Линии связи и их характеристики



Состав линии связи

Линия или канал связи состоит в общем случае из:

- физической среды, по которой передаются электрические сигналы,
- аппаратуры передачи данных,
- промежуточной аппаратуры.

В зависимости от среды линии делятся на:

- проводные или воздушные,
- кабельные (медные и волоконно – оптические),
- радиоканалы наземной и спутниковой связи.

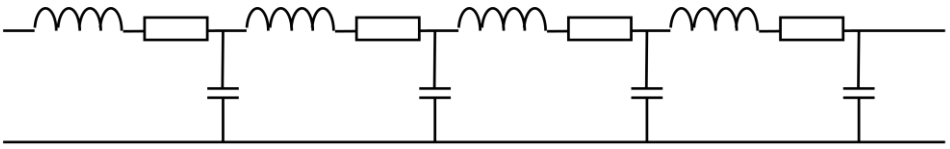
Кабельные линии - проводники, заключенных в несколько слоев изоляции: электрической, электромагнитной, механической, а также, возможно, климатической. Три основных типа кабеля:

- кабели на основе скрученных пар медных проводов,
- коаксиальные кабели с медной жилой,
- волоконно-оптические кабели.

Аппаратура передачи данных (DCE) непосредственно связывает компьютеры или локальные сети пользователя с линией связи и является **пограничным** оборудованием. Пример DCE – модемы.

Оконечное оборудование данных (DTE). Примеры DTE - компьютеры.

Аппаратура передачи данных по аналоговым и цифровым линиям связи отличается, так как в первом случае линия связи предназначена для передачи сигналов произвольной формы и не предъявляет никаких требований к способу представления единиц и нулей, а во втором — все параметры передаваемых линией импульсов стандартизованы. Другими словами, на цифровых линиях связи протокол физического уровня определен, а на аналоговых линиях — нет.

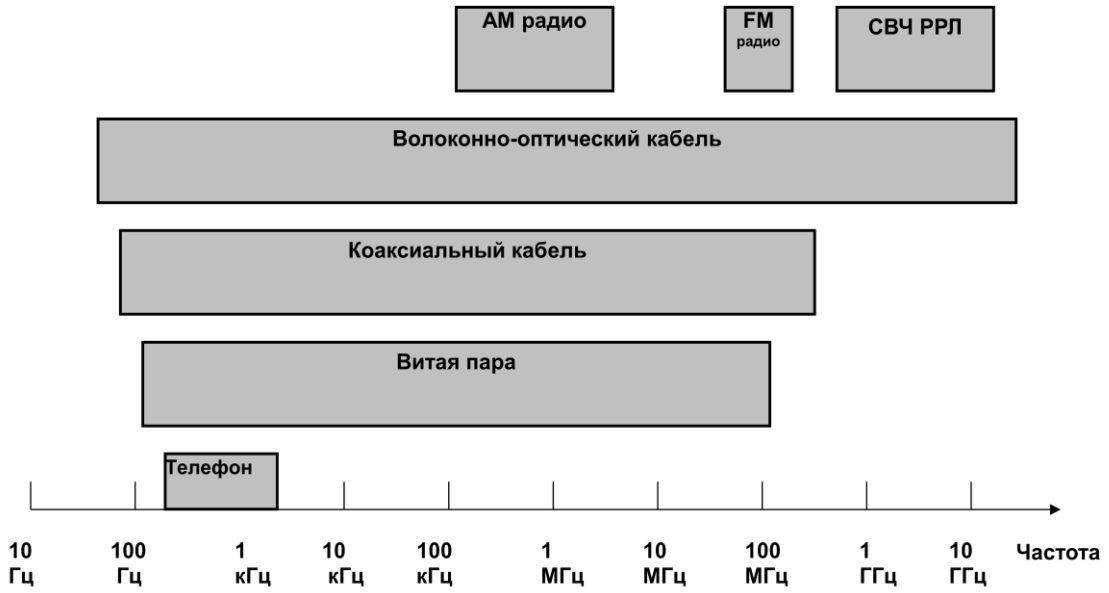


Представление линии связи как распределенной индуктивно-емкостной нагрузки

Линия связи искажает передаваемые данные. Линия связи представляет собой некую распределенную комбинацию активного сопротивления, индуктивной и емкостной нагрузки.

Основные характеристики линий связи:

- амплитудно-частотная характеристика;
- полоса пропускания;
- затухание;
- помехоустойчивость;
- перекрестные наводки на ближнем конце линии;
- пропускная способность;
- достоверность передачи данных;
- удельная стоимость.



Полосы пропускания линий связи

Полоса пропускания зависит от типа линии и ее протяженности. На рисунке показаны полосы пропускания линий связи различных типов, а также наиболее часто используемые в технике связи частотные диапазоны.

Пропускная способность линии характеризует максимально возможную скорость передачи данных по линии связи. Измеряется в битах в секунду.

Теория информации говорит, что любое различимое и непредсказуемое изменение принимаемого сигнала несет в себе информацию. Т.е. прием синусоиды, у которой амплитуда, фаза и частота остаются неизменными, информации не несет, так как изменение сигнала хотя и происходит, но является хорошо предсказуемым. Аналогично, не несут в себе информации тактовые имульсы.

Если сигнал изменяется так, что можно различить только два его состояния, то любое его изменение будет соответствовать наименьшей единице информации — биту. Если же сигнал может иметь более двух различимых состояний, то любое его изменение будет нести несколько бит информации.

Количество изменений информационного параметра несущего периодического сигнала в секунду измеряется в бодах.

Пропускная способность линии в битах в секунду в общем случае не совпадает с числом бод. Она может быть как выше, так и ниже числа бод, и это соотношение зависит от способа кодирования.

Если сигнал имеет более двух различимых состояний, то пропускная способность в битах в секунду будет выше, чем число бод.

При использовании сигналов с двумя различимыми состояниями может наблюдаться обратная картина, когда каждый бит в последовательности кодируется с помощью нескольких изменений информационного параметра несущего сигнала. Например, при кодировании единичного значения бита импульсом положительной полярности, а нулевого значения бита — импульсом отрицательной полярности физический сигнал дважды изменяет свое состояние при передаче каждого бита. При таком кодировании пропускная способность линии в два раза ниже, чем число бод, передаваемое по линии.

Связь между полосой пропускания линии и ее максимально возможной пропускной способностью, вне зависимости от принятого способа физического кодирования, установил Клод Шеннон:

$$C=F \log_2(1+Pc/Pш)$$

где C — максимальная пропускная способность линии в битах в секунду,
F — ширина полосы пропускания линии в герцах,
Pc — мощность сигнала,
Pш — мощность шума.

Из этого соотношения видно, что хотя теоретического предела пропускной способности линии с фиксированной полосой пропускания не существует, на практике такой предел имеется. Действительно, повысить пропускную способность линии можно за счет увеличения мощности передатчика или же уменьшения мощности шума (помех) на линии связи.

Близким по сути к формуле Шеннона является следующее соотношение, полученное Найквистом, которое также определяет максимально возможную пропускную способность линии связи, но без учета шума на линии:

$$C=2F \log_2(M)$$

где M — количество различимых состояний информационного параметра.

Если сигнал имеет 2 различимых состояния, то пропускная способность равна удвоенному значению ширины полосы пропускания линии связи. Если же передатчик использует более чем 2 устойчивых состояния сигнала для кодирования данных, то пропускная способность линии повышается, так как за один такт работы передатчик передает несколько бит исходных данных, например 2 бита при наличии четырех различимых состояний сигнала.

Хотя формула Найквиста явно не учитывает наличие шума, косвенно его влияние отражается в выборе количества состояний информационного сигнала. Для повышения пропускной способности канала хотелось бы увеличить это количество до значительных величин, но на практике мы не можем этого сделать из-за шума на линии.

Помехоустойчивость и достоверность

Перекрестные наводки на ближнем конце (Near End Cross Talk — NEXT) определяют помехоустойчивость кабеля к внутренним источникам помех, когда сигнал, передаваемый выходом передатчика по одной паре проводников, наводит на другую пару проводников сигнал помехи. Если ко второй паре будет подключен приемник, то он может принять наведенную внутреннюю помеху за полезный сигнал. Показатель NEXT, выраженный в децибелах, равен

$$NEXT= 10 \lg(Pвых/Pнав)$$

где Pвых — мощность выходного сигнала, Pнав — мощность наведенного сигнала.

Чем меньше значение NEXT, тем лучше кабель. Так, для витой пары категории 5 показатель NEXT должен быть меньше -27 дБ на частоте 100 МГц.

Методы передачи дискретных данных на физическом уровне

1. Модуляция.
2. Цифровое кодирование.

Цифровое кодирование.

Применяют потенциальные и импульсные коды. Потенциальные для представления сигнала используют уровень, импульсные – импульс либо перепад.

Требования к методам цифрового кодирования:

- Битовая синхронизация приемника и передатчика
- Распознавание ошибочных символов
- Отсутствие постоянной составляющей для обеспечения гальванической развязки
- Экономичное использование частотного спектра

Синхронизация.

- отдельной тактирующей линии связи (на небольших расстояниях)
- самосинхронизирующиеся коды, сигналы которых несут для передатчика указания о том, в какой момент времени нужно осуществлять распознавание очередного бита (или нескольких бит, если код ориентирован более чем на два состояния сигнала). Любой резкий перепад сигнала — так называемый фронт — может служить хорошим указанием для синхронизации приемника с передатчиком.

Потенциальный код без возвращения к нулю

Плюсы

- прост в реализации,
- обладает хорошей распознаваемостью ошибок (из-за двух резко отличающихся потенциалов),

Минусы

- не обладает свойством самосинхронизации.
- наличие низкочастотной составляющей, которая приближается к нулю при передаче длинных последовательностей единиц или нулей.

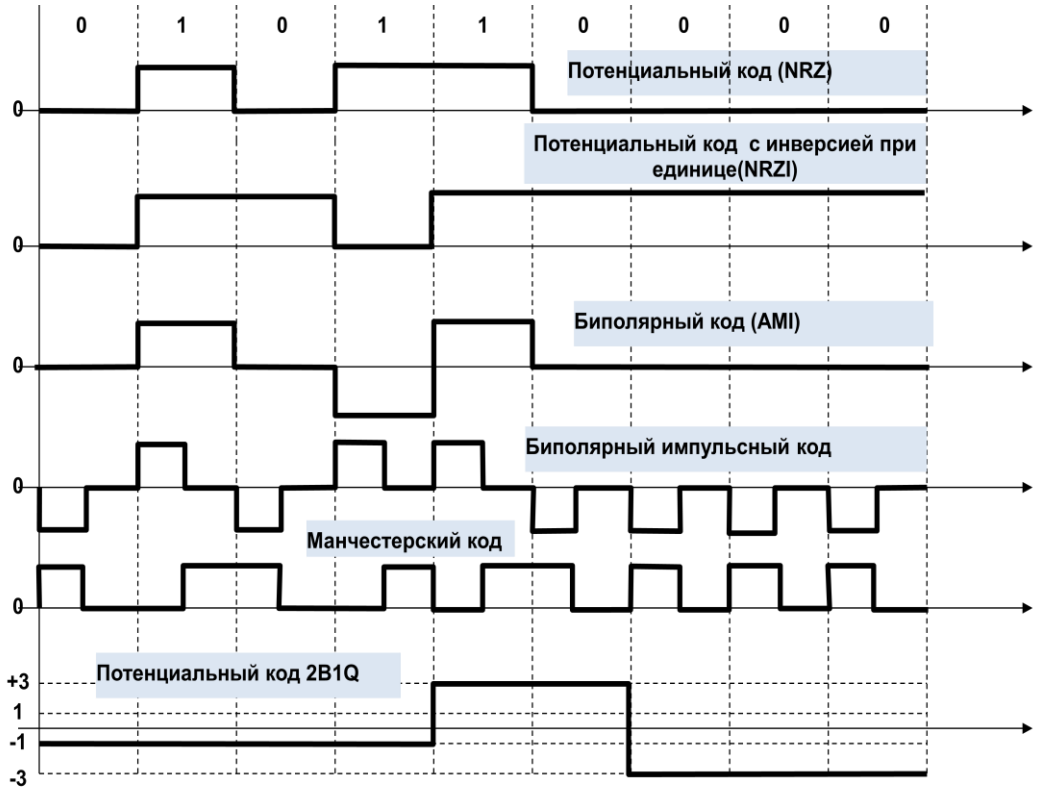
Метод биполярного кодирования с альтернативной инверсией

Для кодирования логического нуля используется нулевой потенциал, а логическая единица кодируется либо положительным потенциалом, либо отрицательным, при этом потенциал каждой новой единицы противоположен потенциалу предыдущей.

- частично ликвидирует проблемы постоянной составляющей и отсутствия самосинхронизации, присущие коду NRZ.
- для различных комбинаций бит на линии использование кода AMI приводит к более узкому спектру сигнала, чем для кода NRZ, а значит, и к более высокой пропускной способности линии.
- в коде AMI используются не два, а три уровня сигнала на линии. Дополнительный уровень требует увеличение мощности передатчика.

Потенциальный код с инверсией при единице

Существует код, похожий на AMI, но только с двумя уровнями сигнала. При передаче нуля он передает потенциал, который был установлен в предыдущем такте (то есть не меняет его), а при передаче единицы потенциал инвертируется на противоположный. Этот код называется потенциальным кодом с инверсией при единице



Для улучшения потенциальных кодов, подобных AMI и NRZI, используются два метода. Первый метод основан на добавлении в исходный код избыточных бит, содержащих логические единицы. Очевидно, что в этом случае длинные последовательности нулей прерываются и код становится самосинхронизирующимся для любых передаваемых данных. Исчезает также постоянная составляющая, а значит, еще более сужается спектр сигнала. Но этот метод снижает полезную пропускную способность линии, так как избыточные единицы информации не несут.

Другой метод основан на предварительном «перемешивании» исходной информации таким образом, чтобы вероятность появления единиц и нулей на линии становилась близкой. Устройства, или блоки, выполняющие такую операцию, называются скрэмблерами (scramble — свалка, беспорядочная сборка). При скремблировании используется известный алгоритм, поэтому приемник, получив двоичные данные, передает их на дескрэмблер, который восстанавливает исходную последовательность бит.

Биполярный импульсный код

- хорошие самосинхронизирующие свойства
- постоянная составляющая при передаче длинной последовательности единиц или нулей
- спектр шире, чем у потенциальных кодов.

Манчестерский код

Самый распространенный метод кодирования.

- для кодирования единиц и нулей используется перепад потенциала, то есть фронт импульса
- каждый такт делится на две части, а информация кодируется перепадами потенциала, происходящими в середине каждого такта.
- так как сигнал изменяется по крайней мере один раз за такт передачи одного бита данных, то манчестерский код обладает хорошими самосинхронизирующими свойствами
- полоса пропускания манчестерского кода уже, чем у биполярного импульсного
- нет постоянной составляющей.

Потенциальный код 2B1Q

Показан потенциальный код с четырьмя уровнями сигнала для кодирования данных. Это код 2B1Q название которого отражает его суть — каждые два бита (2B) передаются за один такт сигналом, имеющим четыре состояния (1Q).

Логическое кодирование должно заменять длинные последовательности бит, приводящие к постоянному потенциалу, вкраплениями единиц.

Для логического кодирования характерны два метода — избыточные коды и скрэмблирование.

Избыточные коды основаны на разбиении исходной последовательности бит на порции, которые часто называют символами. Затем каждый исходный символ заменяется на новый, который имеет большее количество бит, чем исходный.

Исходный код	Результирующий код	Исходный код	Результирующий код
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

Например, логический код 4B/5B меняет исходные символы длиной в 4 бита на символы длиной в 5 бит. Результирующие символы могут содержать 32 битовых комбинации, в то время как исходные символы — только 16. Поэтому в результирующем коде можно отобрать 16 таких комбинаций, которые не содержат большого количества нулей, а остальные считать запрещенными кодами (code violation). Символы кода 4B/5B длиной 5 бит гарантируют, что при любом их сочетании на линии не могут встретиться более трех нулей подряд.

Скрэмблирование

Методы скрэмблирования заключаются в побитном вычислении результирующего кода на основании бит исходного кода и полученных в предыдущих тактах бит результирующего кода. Например, скрэмблер может реализовывать следующее соотношение:

$$B_i = A_i + B_{i-3} + B_{i-5},$$

где Bi — двоичная цифра результирующего кода, полученная на i-м такте работы скрэмблера, Ai - двоичная цифра исходного кода, поступающая на i-м такте на вход скрэмблера, Bi-з и Bi-5 — двоичные цифры результирующего кода, полученные на предыдущих тактах работы скрэмблера, соответственно на 3 и на 5 тактов ране» текущего такта, + операция исключающего ИЛИ (сложение по модулю 2).

Например, для исходной последовательности 110110000001 скрэмблер даст с дующий результирующий код:

B1 = A2 = 1 (первые три цифры результирующего кода будут совпадать с исходным, так как еще нет нужных предыдущих цифр)

B2 = A2 = 1

B3 = A3 = 0

B4= A4 + B1 = 1 + 1 = 0

B5 = A5 + B2 = 1 + 1 = 0

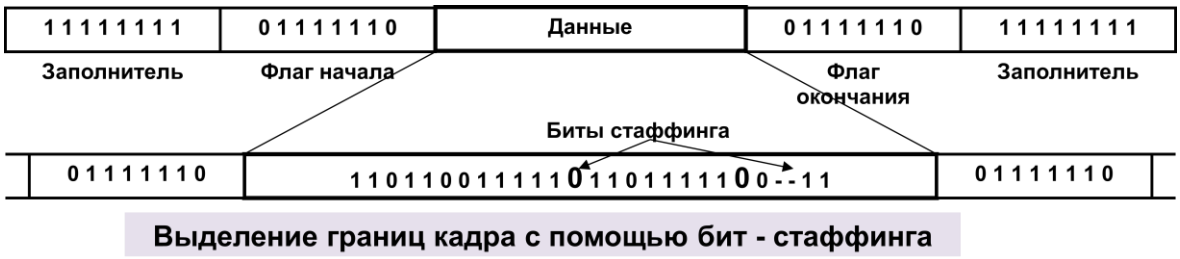
B6 = A6 + B3 + B1 = 0 + 0 + 1 = 1

B7 = A7 + B4 + B2 = 0 + 0 + 1 = 1

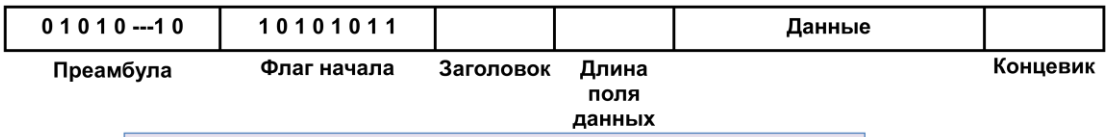
B8 = A8 + B5 + B3 = 0 + 0 + 0 = 0
B9 = A9 + B6 + B4 = 0 + 1 + 0 = 1
B10 = A10 + B7 + B5 = 0 + 1 + 0 = 1
B11 = A11 + B8 + B6 = 0 + 0 + 1 = 1
Bl2 = A12 + B9 + B7 = 1 + 1 + 1 = 1

Таким образом, на выходе скрэмблера появится последовательность 110001101111, в которой нет последовательности из шести нулей, присутствовавшей в исходном коде.

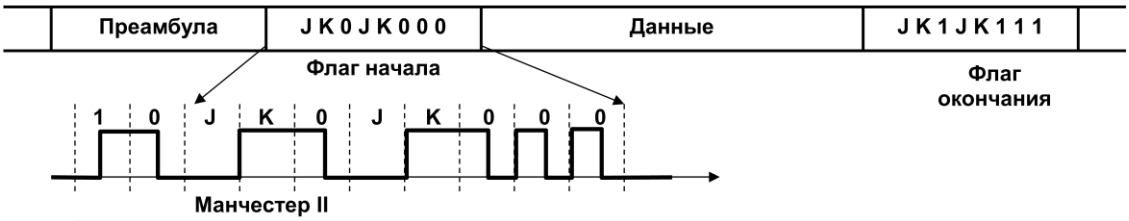
Методы передачи данных канального уровня



Выделение границ кадра с помощью бит - стаффинга



Формат кадра с указанием длины поля данных



Выделение границ кадра с запрещенными символами линейного кода

На рисунке показаны различные схемы бит-ориентированной передачи. Они отличаются способом обозначения начала и конца каждого кадра.

Первая схема. Начало и конец каждого кадра отмечается одной и той же 8-битовой последовательностью — 01111110, называемой флагом. Термин «бит-ориентированный» используется потому, что принимаемый поток бит сканируется на побитовой основе для обнаружения стартового флага, а затем во время приема для обнаружения стопового флага. Поэтому длина кадра в этом случае не обязательно кратна 8 бит.

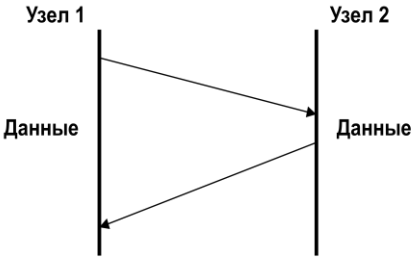
Чтобы обеспечить синхронизацию приемника, передатчик посылает последовательность байтов простоя (каждый состоит из 11111111), предшествующую стартовому флагу.

Для достижения прозрачности данных в этой схеме необходимо, чтобы флаг не присутствовал в поле данных кадра. Это достигается с помощью приема, известного как вставка 0 бита, — бит-стаффинга. Схема вставки бита работает на передающей стороне во время передачи поля данных кадра. Если эта схема обнаруживает, что подряд передано пять 1, то она автоматически вставляет дополнительный 0 (даже если после этих пяти 1 шел 0). Поэтому последовательность 01111110 никогда не появится в поле данных кадра. Аналогичная схема работает в приемнике и выполняет обратную функцию. Когда после пяти 1 обнаруживается 0, он автоматически удаляется из поля данных.

Во второй схеме для обозначения начала кадра имеется только стартовый флаг, а для определения конца кадра используется поле длины кадра, которое при фиксированных размерах заголовка и концевика чаще всего имеет смысл длины поля данных. Для обозначения факта незанятости среды в исходном состоянии по среде вообще не передается никаких символов. Чтобы все остальные станции вошли в битовую синхронизацию, посылающая станция предваряет, содержимое кадра последовательностью бит, известной как преамбула, которая состоит из чередования единиц и нулей 101010... Войдя в битовую синхронизацию, приемник исследует входной поток на побитовой основе, пока не обнаружит байт начала кадра 10101011. За этим байтом следует заголовок кадра, в котором в определенном месте находится поле длины поля данных. Таким образом, в этой схеме приемник просто отсчитывает заданное количество байт, чтобы определить окончание кадра.

Третья схема (внизу) использует для обозначения начала и конца кадра флаги, которые включают запрещенные для данного кода сигналы (V). Например, при манчестерском кодировании вместо обязательного изменения полярности сигнала в середине тактового интервала уровень сигнала остается неизменным и низким (запрещенный сигнал J) или неизменным и высоким (запрещенный сигнал К). Начало кадра отмечается последовательностью JK0JK000, а конец — последовательностью JK1JK100. Этот способ очень экономичен, так как не требует ни бит-стаффинга, ни поля длины, но его недостаток заключается в зависимости от принятого метода физического кодирования. При использовании избыточных кодов (например 4B/5B) роль сигналов J и К играют запрещенные символы.

Каждая из трех схем имеет свои преимущества и недостатки. Флаги позволяют отказаться от специального дополнительного поля, но требуют специальных мер: либо бит-стаффинга, либо запрещенных сигналов, что делает эту схему зависимой от способа кодирования.



Протокол без установления соединения

При передаче данных на канальном уровне используются как дейтаграмные процедуры, работающие без установления соединения, так и процедуры с предварительным установлением логического соединения.

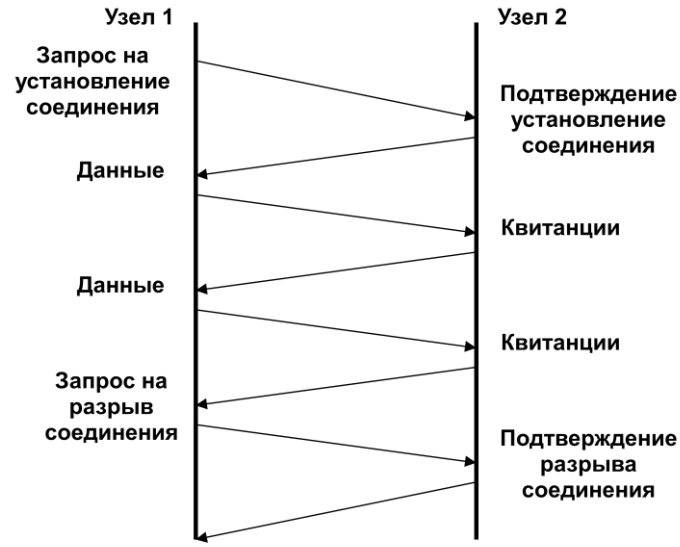
При дейтаграммной передаче кадр посылается в сеть «без предупреждения», и никакой ответственности за его утерю протокол не несет. Предполагается, что сеть всегда готова принять кадр.

+ работает быстро, так как никаких предварительных действий перед отправкой данных не выполняется;

- трудно организовать отслеживание факта доставки.

Передача с установлением соединения более надежна, но требует больше времени для передачи данных и вычислительных затрат от конечных узлов.

В этом случае узлу-получателю отправляется служебный кадр специального формата с предложением установить соединение. Если узел-получатель согласен с этим, то он посылает в ответ другой служебный кадр, подверждающий установление соединения и предлагающий для данного логиче-кого соединения некоторые параметры, например идентификатор соединения, максимальное значение поля данных кадров, которые будут использоваться в рамках данного соединения, и т. п. Узел-инициатор соединения может завершить процесс установления соединения отправкой третьего служебного кадра, в котором сообщит, что предложенные параметры ему подходят. На этом логи-



Протокол с установлением соединения

ческое соединение считается установленным, и в его рамках можно передавать информационные кадры с пользовательскими данными.

После передачи некоторого законченного набора данных, например определенного файла, узел инициирует разрыв данного логического соединения, посылая соответствующий служебный кадр.

Обнаружение и коррекция ошибок

Большая часть протоколов канального уровня выполняет только одну задачу — обнаружение ошибок, считая, что корректировать ошибки, то есть повторно передавать данные, должны протоколы верхних уровней.

Все **методы обнаружения** основаны на передаче служебной избыточной информации, по которой можно судить о достоверности принятых данных. Эту служебную информацию принято называть контрольной суммой. Контрольная сумма вычисляется как функция от основной информации, причем необязательно только путем суммирования. Принимающая сторона повторно вычисляет контрольную сумму кадра по известному алгоритму и в случае ее совпадения делает вывод о том, что данные были переданы корректно. Существует несколько распространенных алгоритмов вычисления контрольной суммы, отличающихся вычислительной сложностью и способностью обнаруживать ошибки в данных.

Контроль по паритету – наиболее простой и наименее мощный метод контроля. С его помощью можно обнаружить только одиночные ошибки в проверяемых данных. Метод заключается в суммировании по модулю 2 всех бит контролируемой информации. Результат суммирования также представляет собой один бит данных. Метод редко применяется в вычислительных сетях из-за его большой избыточности и невысоких диагностических способностей.

Вертикальный и горизонтальный контроль по паритету представляет собой модификацию описанного выше метода.Исходные данные рассматриваются в виде матрицы, строки которой составляют байты данных. Контрольный разряд подсчитывается отдельно для каждой строки и для каждого столбца матрицы. Этот метод обнаруживает большую часть двойных ошибок, однако обладает еще большей избыточностью. На практике сейчас также почти не применяется.

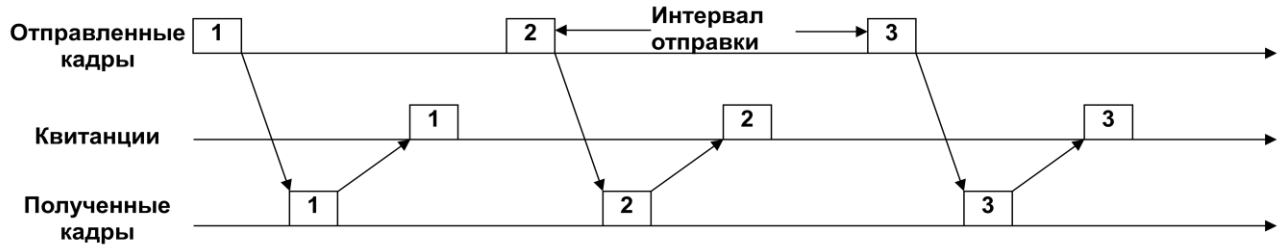
Циклический избыточный контроль (CRC) является в настоящее время наиболее популярным методом контроля в вычислительных сетях (и не только в сетях, например, этот метод широко применяется при записи данных на диски и дискеты). Метод основан на рассмотрении исходных данных в виде одного многоразрядного двоичного числа. Например, кадр стандарта Ethernet, состоящий из 1024 байт, будет рассматриваться как одно число, состоящее из 8192 бит. В качестве контрольной информации рассматривается остаток от деления этого числа на известный делитель R. Обычно в качестве делителя выбирается семнадцати- или тридцати трехразрядное число, чтобы остаток от деления имел длину 16 разрядов (2 байт) или 32 разряда (4 байт). При получении кадра данных снова вычисляется остаток от деления на тот же делитель R, но при этом к данным кадра добавляется и содержащаяся в нем контрольная сумма. Если остаток от деления на R равен нулю, то делается вывод об отсутствии ошибок в полученном кадре, в противном случае кадр считается искаженным.

Этот метод обладает более высокой вычислительной сложностью, но его диагностические возможности гораздо выше, чем у методов контроля, по паритету. Метод CRC обнаруживает все одиночные ошибки, двойные ошибки и ошибки в нечетном числе бит. Метод обладает также невысокой степенью избыточности. Например, для кадра Ethernet размером в 1024 байт контрольная информация длиной в 4 байт составляет только 0,4 %.

Методы коррекции ошибок в вычислительных сетях основаны на повторной передаче кадра данных. Чтобы убедиться в необходимости повторной передачи данных,

- отправитель нумерует отправляемые кадры и для каждого кадра ожидает от приемника квитанции — служебного кадра, извещающего о том, что исходный кадр был получен и данные в нем оказались корректными.
- время этого ожидания ограничено — при отправке каждого кадра передатчик запускает таймер, и, если по его истечении положительная квитанция на получена, кадр считается утерянным.
- приемник в случае получения кадра с искаженными данными может отправить отрицательную квитанцию — явное указание на то, что данный кадр нужно передать повторно.

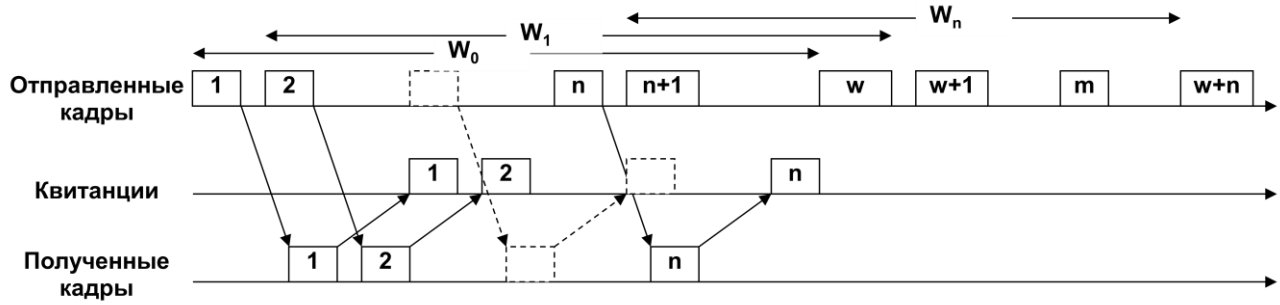
Существуют два подхода к организации процесса обмена квитанциями: с простоями и с организацией «окна».



Метод с простоями

Метод с простоями требует, чтобы источник, пославший кадр, ожидал получения квитанции (положительной или отрицательной) от приемника и только после этого посылал следующий кадр (или повторял искаженный). Если же квитанция не приходит в течение тайм-аута, то кадр (или квитанция) считается утерянным и его передача повторяется. Видно, что в этом случае производительность обмена данными существенно снижается, — хотя передатчик и мог бы послать следующий кадр сразу же после отправки предыдущего, он обязан ждать прихода квитанции.

Второй метод называется методом **«скользящего окна»**. В этом методе для повышения коэффициента использования линии источнику разрешается передать некоторое количество кадров в непрерывном режиме, то есть в максимально возможном для источника темпе, без получения на эти кадры положительных ответных квитанций. Количество кадров, которые разрешается передавать таким образом, называется размером окна. Рисунок иллюстрирует данный метод для окна размером в W кадров.



Метод скользящего окна

В начальный момент, когда еще не послано ни одного кадра, окно определяет диапазон кадров с номерами от 1 до W включительно. Источник начинает передавать кадры и получать в ответ квитанции. Для простоты предположим, что квитанции поступают в той же последовательности, что и кадры, которым они соответствуют. При получении первой квитанции окно сдвигается на одну позицию, определяя новый диапазон от 2 до (W+1).

Процессы отправки кадров и получения квитанций идут достаточно независимо друг от друга. Рассмотрим произвольный момент времени tn, когда источник получил квитанцию на кадр с номером n. Окно сдвинулось вправо и определило диапазон разрешенных к передаче кадров от (n+1) до (W+n). Все множество кадров, выходящих из источника, можно разделить на несколько групп:

- Кадры с номерами от1 до n уже были отправлены и квитанции на них получены, то есть они находятся за пределами окна слева.
- Кадры, начиная с номера (n+1) и кончая номером (W+n), находятся в пределах окна и потому могут быть отправлены не дожидаясь прихода какой-либо квитанции. Этот диапазон может быть разделен еще на два поддиапазона:
    - кадры с номерами от (n+1) до m которые уже отправлены, но квитанции на них еще не получены;
    - кадры с номерами от m до (W+n), которые пока не отправлены, хотя запрета на это нет.
- Все кадры с номерами, большими или равными (W+n+1), находятся за пределами окна справа и поэтому пока не могут быть отправлены.

Хотя в данном примере размер окна в процессе передачи остается постоянным, в реальных протоколах (например, TCP) можно встретить варианты данного алгоритма с изменяющимся размером окна.

Итак, при отправке кадра с номером n источнику разрешается передать еще W-1 кадров до получения квитанции на кадр n, так что в сеть последним уйдет кадр с номером (W+n-1). Если же за это время квитанция на кадр n так и не пришла, то процесс передачи приостанавливается, и по истечении некоторого тайм-аута кадр n (или квитанция на него) считается утерянным, и он передается снова.

Если же поток квитанций поступает более-менее регулярно, в пределах допуска в W кадров, то скорость обмена достигает максимально возможной величины для данного канала и принятого протокола.

Метод с простоями является частным случаем метода скользящего окна, когда размер окна равен единице. Метод скользящего окна имеет два параметра, которые могут заметно влиять на эффективность передачи данных между передатчиком и приемником — размер окна и величина тайм-аута ожидания квитанции. Выбор тайм-аута зависит не от надежности сети, а от задержек передачи кадров сетью.

Во многих реализациях метода скользящего окна величина окна и тайм-аут выбираются адаптивно, в зависимости от текущего состояния сети.

Управление сетевым трафиком

Чтобы почувствовать актуальность темы – простой пример (Столлингс, «Современные компьютерные сети», глава 8). Веб-сервер обслуживает индивидуальные запросы в среднем за 1мс. Чтобы упростить задачу – ровно за 1мс. Если скорость поступления запросов – 1 запрос в миллисекунду – то кажется, можно утверждать, что сервер справится с такой нагрузкой.

1. Идеальный случай – постоянный интервал прихода запросов, сервер обрабатывает запрос сказу как он приходит. Как только завершается обработка – сразу переходит к следующему.
2. Более реалистичный вариант. Средняя скорость – такая-же, но интервал прихода запросов не постоянный. В течение одного интервала может не поступить ни одного запроса, а в другое время – большое количество. Но средняя скорость – постоянна. В период высокой нагрузки сервер хранит запросы в буфере, т.е. они стоят в очереди на обработку. Самый важный вопрос в данном случае – насколько большим должен быть буфер.

Есть таблицы, показывающие поведение такой системы:

Первая таблица – предполагается, что в систему поступает в среднем 500 запросов в секунду, что соответствует половине производительности сервера. Записи в таблице показывают количество запросов, прибывающих каждую секунду, количество запросов обработанных в течение этой секунды и количество избыточных запросов, поставленных в очередь. Данные в таблице содержатся за 50 секунд.

0	0	0	0
1	88	88	0
2	796	796	0
3	1627	1000	627
4	51	678	0
5	34	34	0
6	966	966	0
7	714	714	0
8	1276	1000	276
9	494	769	0
10	933	933	0
11	107	107	0
12	241	241	0
13	16	16	0
14	671	671	0
15	643	643	0
16	812	812	0
17	262	262	0
18	218	218	0
19	1378	1000	378
20	507	885	0
21	15	15	0
22	820	820	0
23	1253	1000	253
24	307	559	0
25	540	540	0

26	190	190	0
27	500	500	0
28	96	96	0
29	943	943	0
30	105	105	0
31	183	183	0
32	447	447	0
33	542	542	0
34	166	166	0
35	165	165	0
36	490	490	0
37	510	510	0
38	877	877	0
39	37	37	0
40	163	163	0
41	104	104	0
42	42	42	0
43	291	291	0
44	645	645	0
45	363	363	0
46	134	134	0
47	920	920	0
48	1507	1000	507
49	598	1000	105
50	172	277	0
Среднее	**499**	**499**	**43**

Среднее значение за 50 секунд – 43 запроса в очереди. Пиковое – 600.
Второй случай. 95% производительности. В среднем – 950 запросов в секунду.

0	0	0	0
1	167	167	0
2	1512	1000	512
3	3091	1000	2604
4	97	1000	1701
5	65	1000	765
6	1835	1000	1601
7	1357	1000	1957
8	2424	1000	3382
9	939	1000	3320
10	1773	1000	4093
11	203	1000	3269
12	458	1000	2754
13	30	1000	1784
14	1275	1000	2059
15	1222	1000	2281
16	1543	1000	2824
17	498	1000	2322
18	414	1000	1736
19	2618	1000	3354
20	963	1000	3317
21	29	1000	2346
22	1558	1000	2904
23	2381	1000	4285
24	583	1000	3868
25	1026	1000	3894

26	361	1000	3255
27	950	1000	3205
28	182	1000	2387
29	1792	1000	3179
30	200	1000	2378
31	348	1000	1726
32	849	1000	1575
33	1030	1000	1605
34	315	1000	921
35	314	1000	234
36	931	1000	165
37	969	1000	134
38	1666	1000	800
39	70	871	0
40	310	310	0
41	198	198	0
42	80	80	0
43	553	553	0
44	1226	1000	226
45	690	915	0
46	255	255	0
47	1748	1000	748
48	2863	1000	2611
49	1136	1000	2748
50	327	1000	2074
Среднее	**948**	**907**	**1859**

Среднее количество ожидающих в буфере запросов – 1859. Это может показаться удивительным. Скорость поступления запросов увеличилась менее чем в 2 раза, а средняя длина очереди – в 40 раз.

Третья таблица. Скорость постуления запросов – 99% производительности. Среднее количество запросов – 2583. Незначительное увеличение скорости заросов приводит к 40%-му росту очереди.

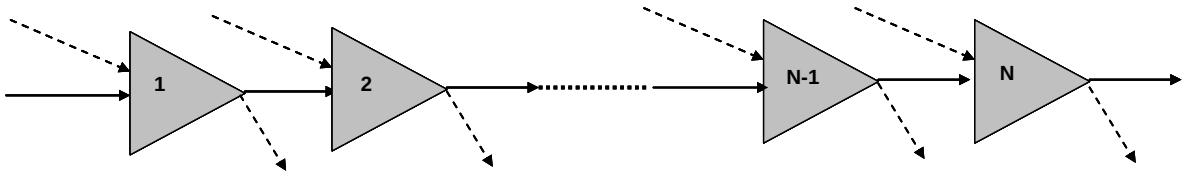
0	0	0	0
1	174	174	0
2	1576	1000	576
3	3221	1000	2798
4	101	1000	1899
5	67	1000	966
6	1913	1000	1879
7	1414	1000	2292
8	2526	1000	3819
9	978	1000	3797
10	1847	1000	4644
11	212	1000	3856
12	477	1000	3333
13	32	1000	2365
14	1329	1000	2693
15	1273	1000	2967
16	1608	1000	3574
17	519	1000	3093
18	432	1000	2525
19	2728	1000	4253
20	1004	1000	4257
21	30	1000	3287
22	1624	1000	3910
23	2481	1000	5391
24	608	1000	4999
25	1069	1000	5068
26	376	1000	4445
27	990	1000	4435
28	190	1000	3625
29	1867	1000	4492
30	208	1000	3700
31	362	1000	3062
32	885	1000	2947
33	1073	1000	3020
34	329	1000	2349
35	327	1000	1676
36	970	1000	1646
37	1010	1000	1656
38	1736	1000	2392
39	73	1000	1465
40	323	1000	788
41	206	994	0
42	83	83	0
43	576	576	0
44	1277	1000	277
45	719	996	0
46	265	265	0
47	1822	1000	822
48	2984	1000	2805
49	1184	1000	2990
50	341	1000	2330
Среднее	**988**	**942**	**2583**

Этот грубый пример наводит на мысль о том, что поведение системы с очередью может не согласовываться с нашей интуицией.

Функции маршрутизаторов (коммутаторов):

- демультиплексирование входного потока пакетов,
- определение для каждого пакета выходного порта и
- формирование посредством мультиплексирования выходного потока пакетов.

Таким образом, **ресурсами сети** являются буферная память и производительность процессора маршрутизатора, а также - пропускная способность линий передачи. Полный маршрут потока пакетов, в рамках конкретной прикладной задачи, или пакетов одного конкретного соединения через сеть, может быть представлен цепочкой мультиплексоров и линий передачи.



Модель полного пути пакета в форме цепочки мультиплексоров и линий передачи

Пунктирными стрелками на этом рисунке представлены другие входные потоки. Эти потоки конкурируют между собой за ресурсы сети – емкость буферов, очередность обслуживания, и пропускную способность линий передачи.

Результаты этого соперничества

- крайне неравномерное распределение ресурсов сети

- снижение качества обслуживания одного, или всех потоков
- возникновение перегрузок на отдельных участках или даже коллапс сети (нулевая производительность).

Предупреждение нежелательных состояний сети требует мер управления сетевым трафиком, **целью** которых являются:

- предупреждение резкого падения производительности сети и
- обеспечение требуемого приложением уровня качества обслуживания (Quality of Service, QoS) генерируемого им потока.

Обеспечение качества обслуживания – комплексная задача и в ее решении принимают участие все уровневые протоколы, в том числе и сетевые. Качество обслуживания описывается системой параметров, специфической для каждого уровня. Параметры качества обслуживания для уровня сети:

- полное время доставки,
- неравномерность задержки доставки пакетов,
- уровень потерь пакетов.

Ясно, что значения параметров качества обслуживания потока в сети зависит от их значений, реализуемых в каждом узле коммутации. Так, полное время доставки (end-to-end delay) пакета определяется задержками, испытываемыми им на каждом коммутационном узле. Следовательно, и среднеяя задержка доставки – это сумма средних задержек. Очевидно, что если последние ограничены сверху, то максимальная полная задержка доставки пакетов данного потока не будет превосходить суммы верхних границ задержек на кажком узле коммутации.

Важной характеристикой качества обслуживания является джиттер (jitter-дрожание) – т.е. величина неравномерности времени доставки пакетов на всем маршруте. Ее максимальное значение может вычисляться как разность верхней и нижней границ задержки доставки пакета.

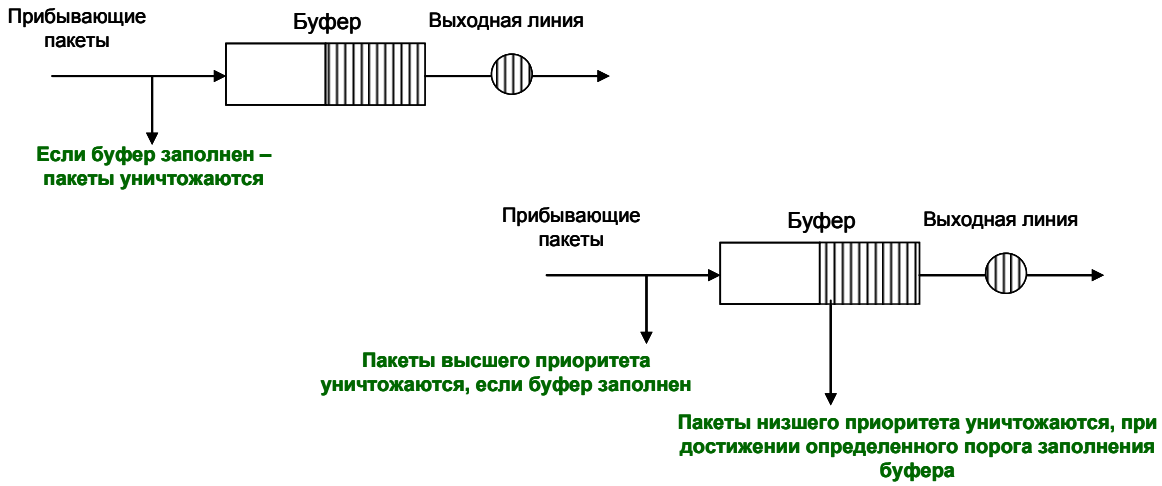
Если пакет приходит в коммутационное устройство в момент, когда его буферная память полностью занята, то он просто уничтожается. Поэтому, причинами потерь пакетов в коммуникационных устройствах являются также резкие скачки числа пакетов на входном порту и внезапное уменьшение пропускной способности выходной линии, или ее перегрузка. Поскольку подобные явления имеют случайный характер, то уровень потерь пакетов на маршруте от источника до приемника определяется суммой вероятностей их потерь на любом из коммутаторов, входящих в маршрут доставки.

Таким образом видно, что управление сетевым трафиком требует наличия:

- механизмов регулирования нагрузки, которые могут применяться к линиям и коммутационным устройствам в сети
- механизмов формирования (профилирования) потоков на входе в сеть
- механизмов справедливого распределения ресурсов сети, выделяемых для обслуживания разных потоков
- средств реализации сетевых политик (набора административных правил доступа к ресурсам сети).

Базовыми элементами реализации перечисленных механизмов являются

- алгоритмы обслуживания очередей пакетов и
- алгоритмы управления средней и пиковой скоростью передачи потоков.



Модель очереди FIFO без приоритезации (а) и с приоритезацией (б)

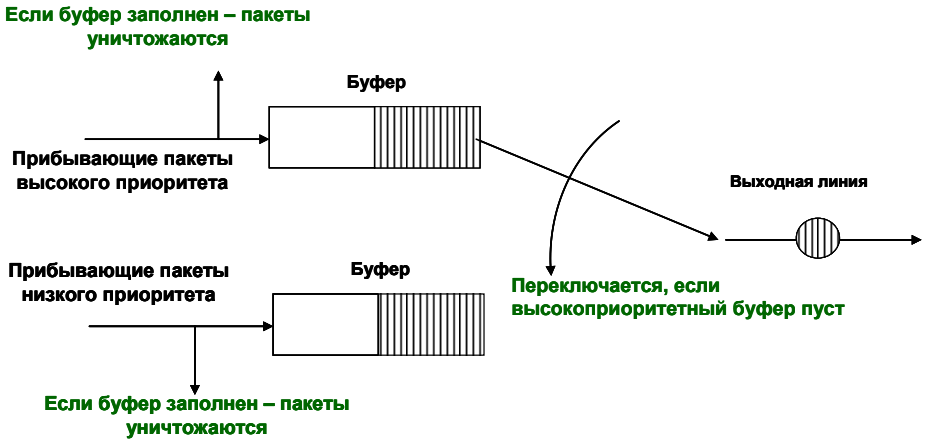
Механизм планирования обслуживания очередей (планировщик очередей) определяет порядок обслуживания пакетов и, следовательно, самым непосредственным образом влияет на распределение ресурсов узла коммутации. Планировщик очередей определяет сколько пакетов из каждого информационного потока будет обслужено в единицу времени.

Простейшим механизмом управления потоком пакетов является очередь с дисциплиной «Первый пришел – первый вышел» (First-In, First-Out, FIFO). В этом случае все прибывающие пакеты помещаются в одну общую очередь и обрабатываются последовательно в порядке прибытия

Пакеты, приходящие в моменты полного заполнения буферной памяти, уничтожаются. Задержка обработки и уровень потерь пакетов в FIFO-буфере зависят от скорости их поступления и длины; поэтому с ростом скорости поступления пакетов и увеличением неравномерности их размера качество обслуживания пакетов в такой очереди будет ухудшаться. Очевидно также, что в такой очереди невозможно обеспечить разные уровни обслуживания пакетов, принадлежащих разным инфрмационнвм потокам. С дисциплиной FIFO связана еще одна проблема – монополизация ресурсов маршрутизатора потоком высокой интенсивности, который заполняет входной буфер и делает недоступными ресурсы маршрутизатора для всех остальных потоков.

Очередь FIFO может быть модифицирована так, что она сможет обеспечивать различный уровень обслуживания разным информационным потокам, т.е. очередь становится приоритетной. Этот механизм предполагает разделение входного трафика на классы (может быть, по типу протокола, по адресу источника, по типу приложения и т.п.) и тогда при достижении определенного порога загруженности буфера пакеты более низкого приоритета будут уничтожаться, сохраняя тем самым возможность обработки пакетов более высокого приоритета. Последние будут обрабатываться до момента полной загруженности буфера.

Другим вариантом решения задачи различения классов обслуживания пакетов является организация нескольких входных очередей (по одной для каждого класса обслуживания), конкурирующих за доступ к ресурсам (CPU и выходная линия) маршрутизатора. Выходная линия в любой момент является доступной для пакетов из буфера, соответствующего самому высокому классу обслуживания (например, пакетам потока, чувствительного к величине задержек) и лишь когда он пуст – обслуживается буфер низшего приоритета. Размер буферов в такой архитектуре также может быть различным, что позволяет удовлетворить разные требования к уровню вероятности потери пакетов.



Модель очереди FIFO с приоритезацией на основе входных буферов

Ресурсы коммутатора всегда ограниченны и ситуация, когда разные информационные потоки вынуждены конкурировать за доступ к ним, является типичной. При этом, рассмотренные выше варианты обеспечения разных классов обслуживания не решают проблему «справедливого» распределения ресурсов маршрутизатора. Действительно, при достаточно высоком трафике высшего приоритета потоки низшего приоритета могут вообще не получить доступа к ресурсам маршрутизатора; не решается также проблема распределения ресурсов между потоками одного класса сервиса, что приводит к монополизации ресурсов более «жадными» приложениями.

Справедливая очередь

«Справедливая очередь» нацелена на преодоление отмеченной выше ассиметрии распределения ресурсов коммутатора. Пусть каждый информационный поток обслуживается своей логической очередью. В идеальной системе (поток входных данных рассматривается как некая непрерывная и всегда существующая вещь) пропускная способность выходной линии (C) может быть разделена поровну между всеми очередями посредством их циклического подключения к выходной линии на строго определенные и равные промежутки времени. При наличии *n* очередей, каждая из них получает возможность передавать свои пакеты в выходную линию со скоростью *C/n* бит/сек.

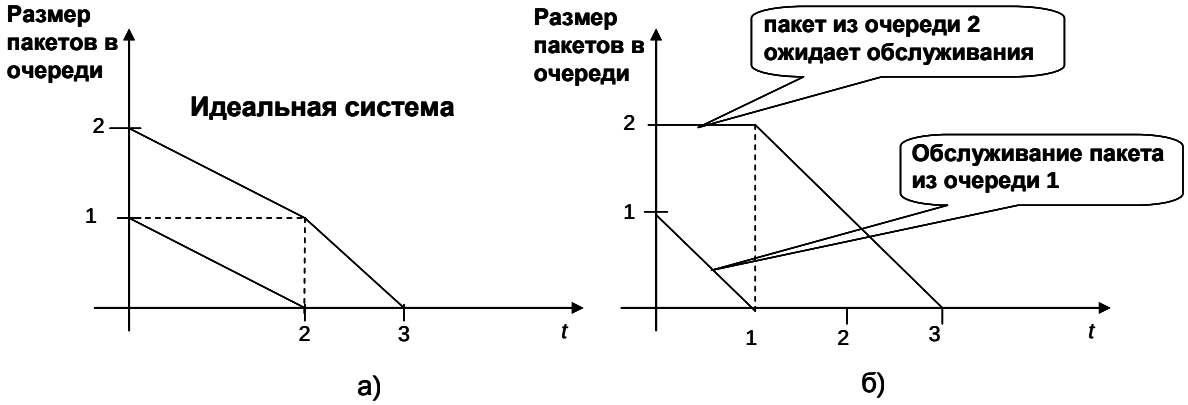
Эффективная скорость передачи может быть и выше, поскольку некоторые очереди становятся время от времени пустыми, и в это время другие из них могут передавать данные. Такая организация очереди предупреждает монополизацию ресурсов выходной линии «жадными» приложениями. Размер буфера для каждого потока может быть выбран таким, чтобы удовлетворялись специфические требования к уровню потерь конкретного потока.

Однако, рассматриваемая очередь является «справедливой» когда мы считаем входные потоки идеальными (непрерывными). В этом случае, пропускная способность передающей линии действительно равномерно распределяется между всеми непустыми очередями. Однако, в силу дискретной природы потока и различия размеров его пакетов, «справедливость» не обеспечивается.

Каждая из входных очередей циклически получает доступ к выходной линии на строго определенные и равные промежутки времени, и за этот интервал может быть передано строго определенное количество бит информации. Если длина пакета превосходит это значение, то он не будет передан; если она меньше – то часть времени доступа к линии этой очередью будет потрачена непроизводитеьно.

Для уменьшения таких непроизводительных потерь можно дефрагментировать приходящие пакеты на относительно короткие составляющие, что, очевидно, будет вести к усложнению процедуры коммутации.

Посмотрим на различия в обслуживании «идеального» потока и потока пакетов конечной длины в системе с двумя очередями.



Обслуживание «справедливой» очереди в идеальной системе и в системе с попакетной дисциплиной при разных длинах пакетов

Каждая из двух очередей содержит только по одному пакету размером *L* бит; при этом пропускная способность выходной линии составляет *L* бит/сек = 1 пакет/сек. В идеальной системе можно обеспечить для каждой очереди возможность передавать свои данные на скорости ½ пакета в секунду.

Если обслуживание очередей организовано побитно (т.е. из каждой очереди равномерно передается по одному биту), то время цикла передачи составляет *2* секунды. Весь пакет из первой очереди будет передан за время *2-1/L* секунд, а передача пакета из второй очереди завершится ровно через 2 секунды (уже видна некоторая «несправедливость» в обслуживании).

Если же обслуживание очередей организовано на «пакетном» принципе (рис. б), то пакет из первой очереди будет обслужен за 1 секунду, а из второй – через 2 секунды («несправедливость» стала еще большей). Естественно, усредненная «несправедливость» будет тем меньше, чем больше пакетов будет обслужено из каждой очереди.

Однако, в случае неравенства длин пакетов распределение пропускной способности выходной линиии не будет «справедливым». Например, если длины пакетов первой очереди вдвое больше длин пакетов второй очереди, то при достаточной длине очередей и обслуживании их на «попакетной» основе эффективная полоса пропускания на данном коммутаторе для первого потока будет вдвое больше, чем для второго. Поэтому, разумным решением является предоставление каждому потоку такой доли ресурса выходной линии, чтобы время завершения обработки пакетов каждого из них оказалось близким к значению, которое было бы получено в идеальной системе с непрерывным потоком входных данных. При попакетной дисциплине обслуживания единственной возможностью достижения этого остается выбор очередности обслуживания буферов.

Для этого каждый пакет, находящийся в очереди, следует снабдить *меткой времени окончания его обслуживания*, вычисленной исходя из модели непрерывного потока.

Очередность же обслуживания входных буферов в каждом цикле определяется значениями этих меток по принципу «Пакет с меньшим значением метки обслуживается первым». Такую дисциплину обслуживания можно назвать **«попакетная справедливая» очередь**. Рассмотрим процедуру вычисления метки.

Пусть существуют ***n*** потоков, каждый из которых обслуживается отдельной очередью со скоростью 1 бит/сек (модель, максимально близкая к непрерывному потоку). Циклом обслуживания назовем период времени, в течении которого каждая очередь обслуживается по одному разу и ***R(t)*** – число циклов обслуживания всех этих ***n*** очередей, реализованных за время ***t***. Будем считать, что ***k***-*й* пакет ***i***-*го* потока прибывает в пустой буфер своей очереди в момент t_k^i и длина этого пакета равна ***P(i,k)*** бит. Этот пакет на побитовой основе будет обработан за ***P(i,k)*** циклов, т.е в момент t^*, когда

$$R(t^*) = F(i,k) = R(t_k^i) + P(i,k).$$

Если пакет прибывает не в пустой буфер, то $F(i,k) = F(i,k-1) + P(i,k)$.

Величина *F(i,k)* и будет использована как финишная метка обслуживания пакета.

Величина *F(i,k)* не определяет действительное (реальное) время завершения обработки *k-го* пакета; она служит лишь признаком очередности, в которой будут обслуживаться буферы в пределах данного цикла.

Для примера рассмотрим систему из двух очередей, первая из которых имеет один пакет единичной длины, а вторая – один пакет удвоенной длины.

В идеальной системе обслуживание очередей производится со скоростью пакет в ½ сек в течении времени пока обе они имеют данные (т.е. на интервале от 0 до 2 единиц времени) и со скоростью пакет в 1 сек - на интервале от 2 до 3 единиц времени.

В «попакетной справедливой» очереди значения финишних меток будут *F(1,1)= R(0)+ 1=1* и *F(2,1)= R(0)+ 2=2.* Поскольку *F(1,1)< F(2,1),* то обслуживание начинается с первой очереди.

Таким образом, если в идеальной системе поток с короткими пакетами передавался с **эффективной** скоростью ½ сек$^{-1}$., а поток с более длинным пакетом – со скоростью 2/3 сек$^{-1}$, то обслуживание их в «справедливой» очереди реализовывалось с эффективными скоростями равными 1сек$^{-1}$ и 2/3 сек$^{-1}$ соответственно. Если бы обслуживание началось с очереди, имеющей более длинный пакет, то эти скорости имели бы значения 1сек$^{-1}$ для «длинной» очереди и 1/3 сек$^{-1}$ для короткой. Последние значения отличаются от идеальных в большей степени.

Взвешенная справедливая очередь

Определение взвешенной справедливости обобщает механизм «справедливого» распределения ограниченного ресурса на случай когда потоки «платят» разную стоимость за обладание единицей ресурса, что соответствует **разной значимости потоков**.

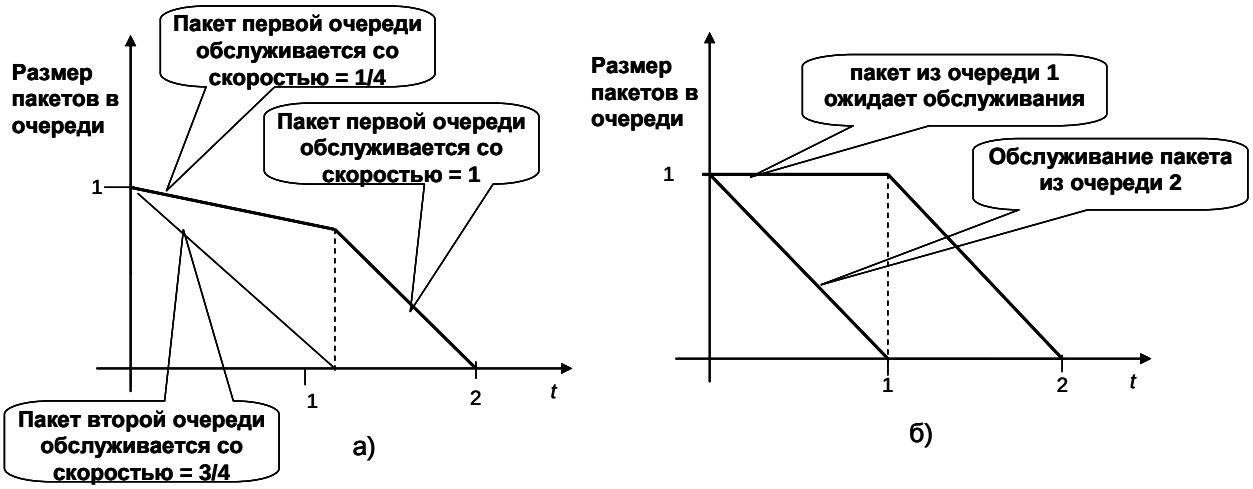
Мера значимости отражается показателем веса потока W_i . Пусть, как и выше, каждый информационный поток обслуживается отдельной очередью, но $W_1=1$, а $W_2=3$. Общая цена, которую готовы «заплатить» за ресурс маршрутизатора равна *4.* В этом случае «справедивость» заключается в том, что первый поток должен получить ¼ , а поток 2 – ¾ пропускной способности выходной линии. При обслуживании очередей на «побитовой» основе первая из них получает возможность в каждом цикле передать 1 бит, а вторая – 3 бита.

В системе с попакетной дисциплиной обслуживания «взвешенная справедливость» также может быть реализована. Пусть имеются ***i*** информационных потоков и «вес» каждого W_i. Значения финишных меток при этом вычисляются в соответствии с выражением

$$F(i,k) = \max\{F(i,k-1), R(t_k^i)\} + P(i,k)/W_i$$

Последнее слагаемое в этом выражении показывает, что чем выше вес потока, тем меньшее значение финишной метки имеют его пакеты и тем быстрее они будут поступать в выходной канал коммутационного устройства.

Рисунок иллюстрирует обработку потоков с обозначенными выше весами в очередях с взвешенной справедливостью при попакетной дисциплине их обслуживания.



Обслуживание «справедливой» взвешенной очереди в идеальной системе и в системе с попакетной дисциплиной при равных длинах пакетов

Очевидно, что $F(1,1) = R(0) + 1/1 = 1$ и $F(2,1) = R(0) + 1/3 = 1/3$. Следовательно, пакет из второй очереди будет обслужен первым.

Взвешенная справедливая очередь является одним из инструментов обеспечения заданного уровня качества обслуживания в сети. Например, в сети Интернет существуют три стратегии обслуживания потоков.

- Первая из них реализует идею «как получится» и для нее достаточно применения FIFO-очередей.
- Вторая базируется на модели дифференциального обслуживания и предполагает разделение всего трафика на несколько классов, качество обслуживания каждого из которых удовлетворяет определенным ограничениям. При этом используется дисциплина приоритетных очередей. Эта стратегия относительно проста в реализации, но не гарантирует строгого обеспечения значений параметров качества обслуживания.
- Третий подход базируется на модели интегрального обслуживания, которая предполагает резервирование ресурсов, необходимых для обслуживания заданного потока с заданным качеством, на каждом маршрутизаторе от источника до приемника. Именнно здесь дисциплина взвешенных справедливых очередей находит свое практическое воплощение.

1. **Интегрированные службы**. Предоставлют коллективные услуги в сети. Изучаются суммарные требования трафика и,
   - во-первых, трафик ограничивается объемами, соответствующими текущим возможностям сети,
   - во-вторых, резервируются ресурсы сети для предоставления определенного качества обслуживания в соответствии с требованиями.
2. **Дифференцированные службы**. Наоборот, не рассматриваются суммарные запросы трафика, а также не резервируются заранее сетевые ресурсы. Вместо этого трафик классифицируется по группам. Каждая группа соответствующим образом помечается, а предоставляемая услуга зависит от членства в той или иной группе.

Т.е., можно сказать, что интегрированные услуги (т.е. совмещенные услуги) – это работа всей сети, которая анализирует свое состояние и рассматривает суммарные запросы, а дифференцированные услуги (раздельные) – предоставляются на основании информации внутри трафика (приоритеты и т.д.)

Некоторые **функции архитектуры ISA**, ориентированные на борьбу с перегрузкой и предоставление транспортных услуг с различными уровнями качества:

- **Контроль допуска**. Для транспорта с гарантированным уровнем качества обслуживания (в отличие от транспорта, обслуживаемого по остаточному принципу) архитектура ISA требует резервирования ресурсов для нового потока. Если маршрутизаторы совместно придут к

соглашению о недостаточности ресурсов для гарантий запрашиваемого уровня качества обслуживания, тогда потоку отказывается в доступе.

- **Алгоритм маршрутизации**. Решение о выборе маршрута может приниматься на основе различных параметров качества обслуживания, а не только минимального значения задержки.
- **Дисциплина очередей**. Учитывает требования различных потоков.
- **Политика отбрасывания пакетов**. Политика очередей определяет, какой пакет будет передан следующим, если несколько пакетов стоят в очереди к одному и тому же выходному порту. Отдельным вопросом является выбор отбрасываемого пакета. Политика отбрасывания пакетов может быть важным элементом борьбы с перегрузкой и обеспечения гарантий качества обслуживания.

Службы ISA определены на двух уровнях. **Во-первых**, службы предоставляют ряд общих категорий обслуживания, каждая из которых обеспечивает определенный обобщенный тип гарантий обслуживания. **Во-вторых**, в каждой категории обслуживание конкретного потока определяется заданными параметрами. Вместе эти параметры называются **спецификацией трафика**.

Три категории обслуживания:

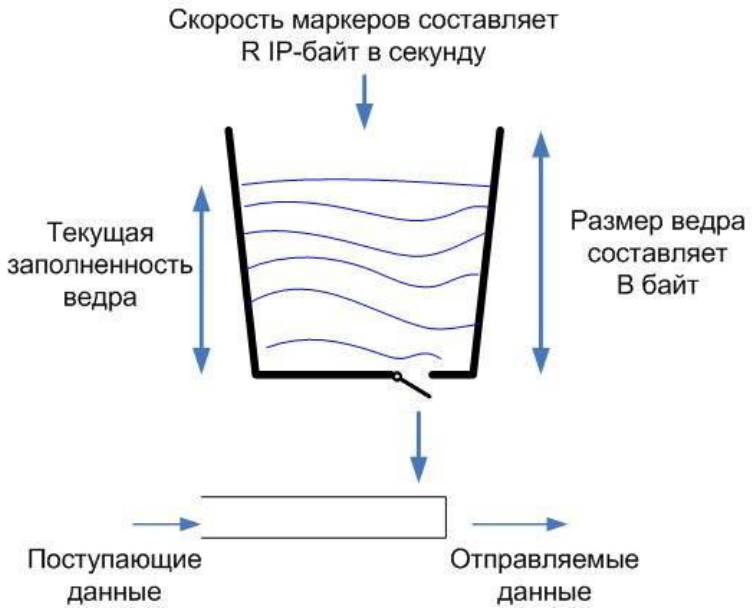
- гарантированное обслуживание;
- обслуживание с контролируемой нагрузкой;
- обслуживание с максимальными усилиями (то есть по остаточному принципу).

Алгоритм «маркерного ведра»

Приложение может потребовать зарезервировать для потока ресурсы с гарантированным или контролируемым уровнем качества. При этом точные параметры указываются в спецификации трафика. Если запрос на резервирование принимается, тогда спецификация трафика становится частью договора между потоком данных и службой. Служба соглашается предоставлять обслуживание с запрашиваемым уровнем качества до тех пор, пока поток данных соответствует спецификации трафика. Пакетам, превышающим параметры, указанные в спецификации, по умолчанию предоставляется обслуживание по остаточному принципу.

Далее следует определить одну общую концепцию: спецификацию трафика **маркерного ведра**. Этот способ описания трафика обладает несколькими преимуществами:

- Большое количество источников трафика могут быть просто и точно описаны с помощью схемы маркерного ведра.
- Схема маркерного ведра предоставляет лаконичное описание нагрузки, оказываемой потоком, позволяя службе легко определить требуемый объем ресурсов.



Спецификация трафика маркерного ведра включает два параметра:

- скорость пополнения маркеров R
- объем ведра B

Значение R определяет постоянную, установившуюся скорость передачи данных, то есть усредненную за относительно долгий период времени. Значение B определяет величину, на которую скорость передачи данных может превышать R в течение короткого интервала времени. В результате в течение любого интервала времени T количество переданных данных не может превышать RT+ B.

Термином «ведро» называют счетчик, указывающий количество байтов IP-данных, которые могут быть переданы в любой момент времени. Ведро заполняется байтовыми маркерами со скоростью R (то есть счетчик увеличивается на единицу R раз в секунду). Содержимое счетчика не может превышать определенного максимального значения, соответствующего объему ведра B. IP-пакеты прибывают и устанавливаются в очередь на обработку. IP-пакет может быть обработан, если в ведре есть достаточное количество маркеров (не менее размера IP-пакета). Если маркеров в ведре достаточно, пакет обрабатывается, а соответствующее количество маркеров вытекает из ведра. Если пакет прибывает, а в ведре нет достаточного количества маркеров, то это значит, что данный пакет превысил параметры спецификации трафика для данного потока. Что делать с таким пакетом, в документации архитектуры ISA не указывается. Обычно подобный пакет либо обслуживается по остаточному принципу (при наличии ресурсов), либо отбрасывается, либо особым образом помечается, чтобы его можно было отбросить позднее.

Средняя скорость передачи IP-данных, разрешенная маркерным ведром, равна R. Однако если возникает период простоя или период относительно низкой скорости поступления данных, ведро наполняется маркерами, что позволяет передать дополнительно до B байт сверх установленной скорости. Таким образом, объем ведра B представляет собой меру допустимой неравномерности потока данных.

Случайное раннее обнаружение

Подход к борьбе с перегрузкой представляет собой упреждающее отбрасывание пакетов, т.е. маршрутизатор отбрасывает один или несколько входящих пакетов прежде, чем переполнится выходной буфер.

Наиболее важный вариант схемы упреждающего отбрасывания пакетов называется случайное раннее обнаружение (Random Early Detection, RED). Сначала обсудим причины для применения схемы RED и цели ее разработки, после чего перейдем к деталям.

Когда в сети возникает перегрузка, буферы маршрутизаторов переполняются и маршрутизаторы начинают терять пакеты. Для TCP-трафика это является сигналом о необходимости перехода в т.н. «фазу затяжного пуска» для снижения нагрузки на сеть и устранения перегрузки. Это достаточно простая процедура, когда в начале передачи данных окно соединения устанавливается равным 1, а при получении каждого последующего подтверждения окно увеличивается на 1, пока не достигнет максимума.

У данного сценария есть два недостатка. Во-первых, потерянные пакеты необходимо передавать повторно, что увеличивает нагрузку на сеть и является причиной значительных задержек в TCP-потоках. Более серьезный феномен называется глобальной синхронизацией, при которой имеет место следующая ситуация. В случае резкого увеличения объемов трафика очереди переполняются, и множество пакетов теряется. Как правило, это затрагивает многие TCP-соединения, в результате они переходят в фазу затяжного пуска, суммарный объем сетевого трафика резко падает и в течение определенного периода сеть используется не в полной мере. Поскольку многие TCP-соединения переходят в фазу затяжного пуска примерно в одно и то же время, они и выходят из этого режима также приблизительно одновременно, что вызывает очередное резкое увеличение объемов трафика в сети и новый цикл «изобилия и голодания».

Одно из решений заключается в использовании на каждом маршрутизаторе буферов большего размера для снижения вероятности потери пакетов. У такого подхода есть недостаток. Когда эти большие буферы заполняются, задержки во всех соединениях возрастают во много раз.

Лучшее решение состоит в том, чтобы предвидеть наступление перегрузки и предлагать то одному, то другому TCP-сообщению снизить скорость передачи данных. Затем определяется эффект этого замедления, после чего, в случае необходимости, замедляется другое соединение. Подобным образом, когда начинается перегрузка, торможение трафика осуществляется постепенно, с минимальным воздействием на TCP-соединения и без глобальной синхронизации. Именно такое решение реализовано в методе RED.

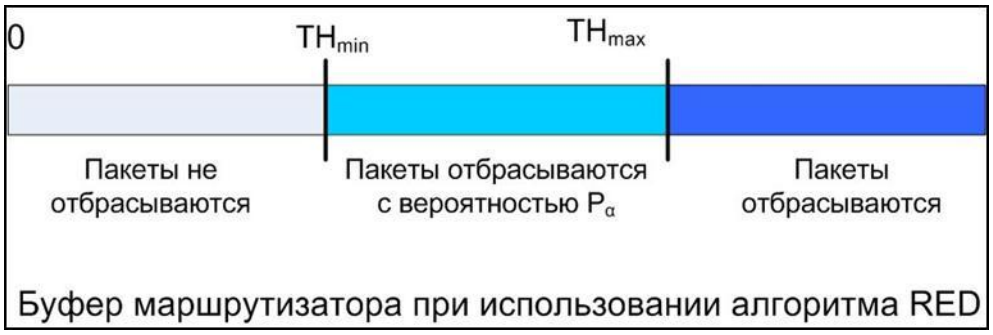
Цели разработки метода случайного раннего обнаружения

- Предупреждение перегрузки. Метод случайного раннего обнаружения предназначен не для реагирования на возникновение перегрузки, а для ее предотвращения.
- Предотвращение глобальной синхронизации. Когда маршрутизатор обнаруживает перегрузку, он должен решить, которому соединению (или соединениям) предложить снизить скорость передачи данных. В применяемом сегодня варианте этого метода данное уведомление является неявным и проявляется в потере пакетов. Обнаруживая перегрузку заранее и уведомляя о ней только те соединения, которые требуется, этот метод позволяет избежать глобальной синхронизации.
- Предотвращение дискриминации источников неравномерного трафика. С большой вероятностью перегрузка в сети начинается с поступления большого объема данных от одного или нескольких источников. Этот всплеск активности добавляется к текущей нагрузке на маршрутизатор. Если для отбрасывания выбираются только прибывающие пакеты, тогда велика вероятность того, что алгоритм отбрасывания пакетов будет направлен в основном против источников с непостоянной скоростью передачи данных.
- Ограничение средней длины очередей. Метод случайного раннего обнаружения должен уметь контролировать среднюю длину очередей и, следовательно, среднюю задержку.

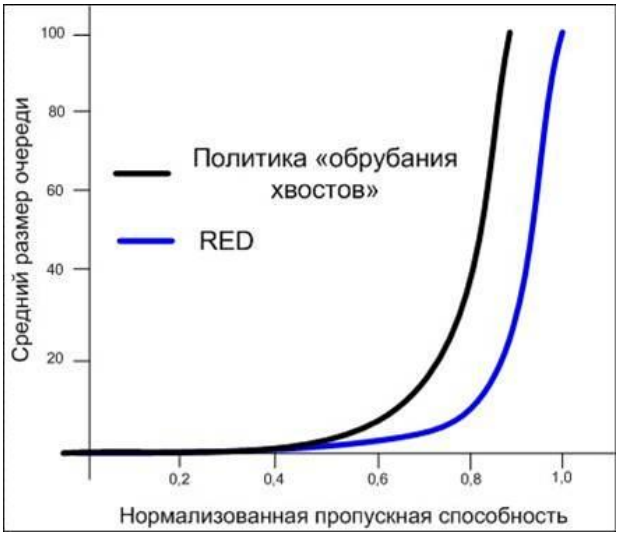
Каждый раз, когда новый пакет поступает в выходную очередь с дисциплиной FIFO, этот алгоритм выполняет две функции. Первый шаг заключается в вычислении средней длины очереди avg. Средняя длина очереди сравнивается с двумя уровнями. Если средняя длина очереди avg меньше нижнего предела THmin, то перегрузка считается минимальной или отсутствующей и пакет помещается в очередь. Если значение avg больше верхнего предела THmax, то перегрузка считается серьезной и пакет отбрасывается. Если значение avg находится между двумя предельными значениями, тогда мы попадаем в область контролируемой перегрузки. В этой области вычисляется вероятность выбрасывания пакета P_a, зависящая от точного значения средней длины очереди avg и увеличивающаяся при приближении значения avg к верхнему пределу. Когда средняя длина очереди находится в этой области, пакет отбрасывается с вероятностью P_a и ставится в очередь с вероятностью 1 - Pa.

```
вычислить среднюю длину очереди avg
        если avg < THmin
                установить пакет в очередь
        иначе если THmin < avg < THmax
                вычислить вероятность Pa
                с вероятностью Pa
                        отбросить пакет
                иначе с вероятностью 1 - Pa
                        установить пакет в очередь
        иначе если avg > THmax
                        отбросить пакет
```

По сути, первая часть алгоритма (вычисление средней длины очереди) определяет допустимый уровень неравномерности трафика, а вторая часть алгоритма — частоту отбрасывания пакетов при данном уровне перегрузки.



Буфер маршрутизатора при использовании алгоритма RED



На рисунке показан результат эмуляции, в которой алгоритм RED сравнивается с политикой обрубания хвостов (drop-tail policy), когда прибывающий пакет просто отбрасывается, если в очереди нет свободного места. При высоких уровнях перегрузки алгоритм RED заметно превосходит политику обрубания хвостов, обеспечивая более высокую пропускную способность.

Хотя архитектура ISA в целом является полезным инструментом, ее довольно трудно реализовать.

Архитектура дифференцированных служб (Differentiated Services, DS) – RFC 2475, предназначена для предоставления простого и легкого в реализации механизма поддержания сетевых служб, различающихся по производительности.

Ключевые характеристики дифференцированных служб:

- Для предоставления различных классов обслуживания IP-пакеты помечаются соответствующим образом, для этого используется поле типа службы протокола IPv4 или поле класса трафика IPv6. Таким образом, не требуется никаких изменений в протоколе IP.
- Перед использованием дифференцированной службы между поставщиком услуг и пользователем устанавливается соглашение об уровне обслуживания. Это позволяет избежать необходимости встраивания дифференцированных служб в приложения. Т.е., не нужно изменять существующие приложения.
- Дифференцированные службы предоставляют встроенный механизм агрегирования (многократного использования). Весь трафик с одинаковым полем DS обрабатывается сетевой службой одинаково. Например, несколько голосовых соединений не обрабатываются индивидуально, а обслуживаются вместе. Это обеспечивает хорошую масштабируемость для сетей большего размера и большей нагрузки.
- Дифференцированные службы реализуются на отдельных маршрутизаторах при установке пакетов в очередь и переправке их дальше в сеть на основе значения поля DS. Маршрутизаторы индивидуально обрабатывают каждый пакет, и у них нет необходимости сохранять информацию о состоянии потоков пакетов.

Теоретические основы сжатия данных

Прежде чем начать обсуждение проблем сжатия данных, освоим теоретический фундамент — теорию информации. Основы теории информации были заложены Клодом Шенноном (Claude Shannon) в процессе исследования пропускной способности информационного канала. Теория информации получила самые разнообразные применения. Для настоящего обсуждения важно то, что теория информации определяет предел, до которого для заданного потока данных можно сжимать информацию без потерь.

Основу теории информации составляют две математические концепции, названия которых могут ввести в заблуждение: информация и энтропия. Как правило, под *информацией* подразумевается нечто, относящееся к смыслу, а *энтропия* — термин из второго закона термодинамики. В теории информации информация имеет отношение к снижению неосведомленности о некоем событии, а энтропией называется усреднение информационных значений, подчиняющееся тем же математическим законам, что и термодинамическая энтропия. Рассмотрим это новое определение информации на примере.

Представим инвестора, которому требуется *информация* (совет) о состоянии определенных ценных бумаг. Этот инвестор советуется с брокером, обладающим специальной *информацией* (знанием) в данной области. Брокер *информирует* (сообщает) инвестора, что сегодня утром нагрянул федеральный инспектор, искавший *информацию* (свидетельства) о возможном мошенничестве, в котором замешана корпорация, выпустившая именно эти акции. В ответ на эту *информацию* (данные) инвестор решает продать свои акции, о чем и *информирует* (уведомляет) брокера. Другими словами, будучи *неуверенным* в вопросе о том, как распорядиться своим портфелем ценных бумаг, клиент консультируется с кем-то более *уверенным* в данном вопросе. Брокер уменьшает *неуверенность* клиента в этой области, рассказав ему о визите федерального инспектора, который пришел, чтобы разрешить собственную профессиональную *неуверенность*. Кульминацией возрастающей *уверенности* клиента о состоянии своих ценных бумаг становится устранение *неуверенности* брокера о намерении клиента продать эти ценные бумаги. Хотя термин *информация* может означать уведомление, знание или просто данные, в каждом случае получение информации эквивалентно уменьшению неуверенности. Таким образом, информация означает положительную разность между двумя уровнями неуверенности.

Информация

Чтобы рассматривать понятие информации с точки зрения математики, нам нужна определенная величина, годящаяся для измерения количества информации. Впервые эту проблему сформулировал и решил Хартли (Hartley) в 1928 г., изучая телеграфную связь. Хартли заметил, что если вероятность того, что некоторое событие произойдет, высока (близка к 1), неуверенность в том, что это событие произойдет, невелика. Если впоследствии мы узнаем, что это событие произошло, то количество полученной нами информации будет невелико. Таким образом, внушающей доверие мерой информации может служить величина, обратная вероятности некоего события, — *1/p.* Например, если происходит событие, вероятность которого равна 0,25, то мы получаем больше информации, чем если произойдет событие, вероятность которого равна 0,5. Если мера информации равна 1/p, тогда первое событие обладает информацией 4 (1/0,25), а количество информации о втором событии равно 2 (1/0,5). Но в использовании этой меры количества информации есть трудности:

Эта единица измерения, кажется, «не работает» с последовательностями событий. Рассмотрим двоичный источник, генерирующий поток из единиц и нулей с равной вероятностью значения каждого бита. Таким образом, каждый бит несет количество информации, равное 2 (1/0,5). Но если бит b_1 несет в себе 2 единицы информации, то сколько информации содержится в строке из двух битов $b_1 b_2$? Эта строка может содержать одну из четырех возможных комбинаций битов, причем каждое из четырех значений строка может принимать с вероятностью 0,25. Таким образом, при использовании в качестве меры количества информации величину 1/p два бита будут содержать четыре единицы информации. Продолжая данные рассуждения, количество информации, содержащееся в трех битах (b_1,b_2,b_3), равно восьми. Это означает, что первые два бита, b_1 и b_2, несут по две единицы информации, а третий бит (b_3) добавляет сразу четыре единицы информации. Следующий бит (b_4) добавит к последовательности уже восемь единиц информации и т. д. Это не кажется разумным.

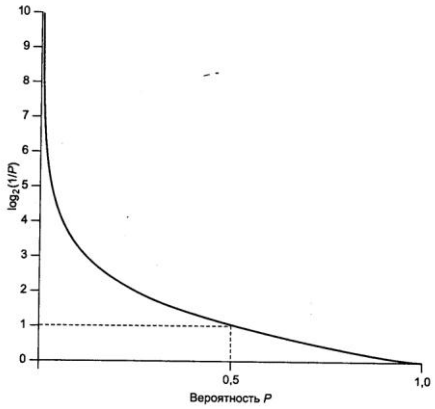
Рассмотрим событие, приводящее к изменению двух или более независимых переменных. Примером такого события может быть сигнал, кодируемый при помощи так называемого метода фазовой манипуляции, для которого используется четыре возможных фазы и две амплитуды. Один элемент такого сигнала содержит две единицы информации для амплитуды и четыре для фазы, итого шесть единиц. В то же время, каждый элемент сигнала может иметь восемь возможных значений, и поэтому должен соответствовать восьми единицам информации, если пользоваться принятой нами мерой.

Хартли разрешил данную проблему, прологарифмировав данную величину, то есть он предложил в качестве меры количества информации события *x* функцию log(1/*P(x)*), где *P(x)* означает вероятность события *x.* Формально можно написать:

$$I(x) = \log (1/P(x)) = - \log P(x)$$

Эта мера количества информации «работает» и позволяет получить множество полезных результатов.

Основание логарифма может быть взято произвольно, но удобнее всего логарифмировать по основанию 2, в этом случае единица информации называется битом.



На рисунке показан график зависимости количества информации единичного события от вероятности этого события. Когда вероятность события приближается к единице (событие происходит почти наверняка), количество информации, содержащееся в данном событии, приближается к нулю. И наоборот, когда вероятность события приближается к нулю (событие практически невероятно), количество информации, содержащееся в данном событии, приближается к бесконечности.

Энтропия

Другая важная концепция теории информации — *энтропия.* Эта концепция была предложена в 1948 г. основателем теории информации Шенноном. Шеннон определил энтропию как среднее количество информации, получаемое от значения случайной переменной. Предположим, что имеется случайная переменная *X,* способная принимать значения x_1 x_2,..., x_N, и что соответствующие вероятности каждого исхода равны $P(x_1)$, $P(x_2)$,..., $P(x_N)$. В последовательности из *K* переменных *X* результат X_j в среднем будет выбран $KP(X_j)$ раз. Таким образом, среднее количество информации, получаемое от *K* событий, равно (будем использовать обозначение P_j для $P(x_j)$):

$$KP_i \log (1/P1) + ... + KP_N \log (1/P_N).$$

Если разделить это выражение на *K,* то мы получим среднее количество информации для одного результата случайной переменной, называемое энтропией *X* и обозначаемое как *H(X):*

$$H(X) = \sum_{j=1}^{N} P_j \log(1/P_j) = -\sum_{j=1}^{N} P_j \log(P_j).$$

Функция *H* часто выражается как перечень вероятностей возможных результатов: $H(P_1, P_2, ..., P_N)$.

Для примера рассмотрим случайную переменную *X,* принимающую два возможных значения с соответствующими вероятностями *P* и 1 - *P.* В этом случае ассоциированная с *X* энтропия будет равна:

$$H(P, 1 - P) = - P \log(P) - (1 - P)\log(1 - P).$$

На рисунке ниже, показан график *H(X)* для данного случая как функция от *P.* По этому графику можно отметить несколько важных особенностей энтропии. Во-первых, если одно из двух событий является достоверным (P= 1 или P=0), тогда энтропия равна нулю. Одно из двух событий должно произойти, и никакой информации о том, что одно из них произошло, не содержится. Во-вторых, максимального значения функция *H(X)* достигает, когда два результата равновероятны. Это также обоснованно: когда два результата равновероятны, неуверенность в результате максимальна. Этот результат можно обобщить для случайной переменной с *N* результатами. Энтропия случайной переменной будет максимальна, когда все результаты равновероятны:

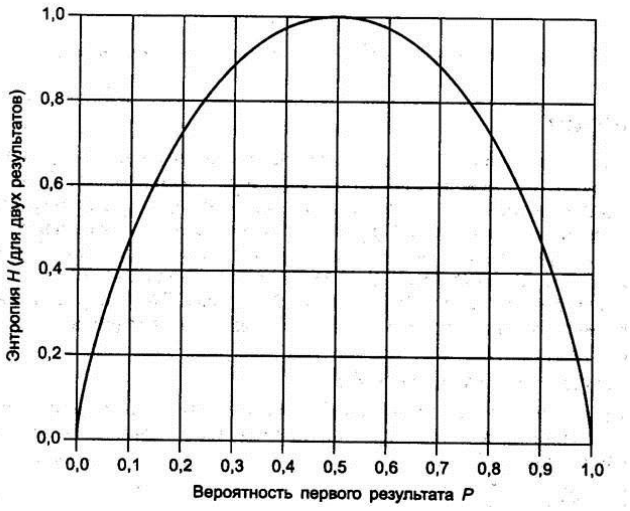
$$\max H(P_1, P_2, ..., P_N) = H(1/N, 1/N, ..., 1/N).$$

Например,

$$H(1/3, 1/3, 1/3) = 1/3 \log 3 + 1/3 \log 3 + 1/3 \log 3 \approx 1{,}585,$$

тогда как

$$H(1/2, 1/3, 1/6) = 1/2 \log 2 + 1/3 \log 3 + 1/6 \log 6 \approx 0{,}5 + 0{,}528 + 0{,}43 = 1{,}458.$$



Одно из важных приложений концепции энтропии заключается в том, что эта концепция помогает понять принципы работы алгоритмов сжатия данных. Энтропия случайной переменной или источника сообщений определяет количество битов, требуемых для представления результатов случайной переменной или альтернативных сообщений без потери информации. Таким образом, при разработке алгоритма сжатия данных энтропия представляет собой меру максимально возможного.

Код Хаффмана

Рассмотрим случайную переменную *X,* принимающую одно из восьми возможных значений $(x_1, x_2, \ldots, x_8)$ со следующими вероятностями:

Здесь P_i представляет собой вероятность выпадения результата x_i. Предположим, что сообщение состоит из реализаций случайной переменной *X,* и мы хотели бы закодировать сообщение в двоичном виде. Один очевидный вариант решения заключается в том, чтобы использовать 3-битовый код фиксированной длины, в котором каждое из восьми возможных значений случайной переменной *X* кодируется одним 3-битовым числом. Лучшая стратегия заключается в использовании кода переменной длины, в котором более длинные кодовые слова назначаются менее вероятным значениям *X,* а более вероятные значения *X* кодируются короткими кодовыми словами. Такой технический прием применяется в азбуке Морзе и коде Хаффмана. Предположим, что сообщения должны передаваться при помощи алфавита из N символов. Каждый символ должен уникальным образом кодироваться двоичной последовательностью. Нас интересует способ построения **оптимального кода**, то есть кода, дающего в результате минимальную среднюю длину кодируемого сообщения. Важно отметить, что мы не ищем минимальную длину кода для какого-либо конкретного сообщения или для всех сообщений (последнее найти невозможно), но минимальную длину кода, усредненную по всем возможным сообщениям.

Другой способ взглянуть на данные требования состоит в том, что мы получаем сообщение, уже закодированное путем назначения символам двоичных слов фиксированной длины.

Таким образом, если символов 8, каждый символ кодируется тремя битами. Если число символов в алфавите от 9 до 16, то каждый символ кодируется четырьмя битами и т. д. Такое кодирование не является оптимальным, если символы встречаются в сообщениях с разной вероятностью. В этом случае требование может быть сформулировано следующим образом.

Требуется разработать оптимальную схему кодирования с использованием кода переменной длины, позволяющую получить кодированное сообщение минимальной средней длины.

Рассмотрим двоичный код с кодовыми длинами $L_1, L_2 \ldots L_N$, поставленный в соответствие алфавиту из *N* символов с вероятностями $P_1, P_2, \ldots, P_N$. Для удобства предположим, что символы организованы в порядке убывания вероятности ($P_1 >= P_2 >= \ldots >= P_N$). Можно показать, что оптимальный код должен удовлетворять следующим требованиям:

- Никакие два разных сообщения не должны состоять из одинаковых последовательностей битов.
- Никакое кодовое слово не должно совпадать с префиксом другого кодового слова.
- Символы, встречающиеся с большей вероятностью, должны кодироваться более короткими кодовыми словами. То есть $L_1<=L_2<= ... <=L_N$.
- Два кодовых слова для символов с наименьшей вероятностью должны иметь одинаковую длину $(L_N = L_{N-1})$ и различаться только последней цифрой.

1. Первое требование гарантирует уникальность кодирования любого сообщения.
2. Второе требование определяет так называемый *мгновенный* код. Это требование не является строго обязательным, но оно требуется, чтобы гарантировать, что сообщение может быть декодировано шаг за шагом. При продвижении слева направо, как только последовательность битов совпадает с данным кодовым словом, декодирующий алгоритм может произвести соответствующее назначение. Вот пример кода, нарушающего первые два требования:

   x1 0 **x2** 010 **x3** 01 **x4** 10

   Двоичная последовательность 010 может соответствовать одному из трех сообщений: x_2, x_3x_1, x_1x_4.
3. Назначение третьего требования легко понять, если отметить, что при нарушении данного условия, поменяв два кода, вы сможете получить меньшее значение средней длины.
4. Чтобы понять суть последнего требования, предположим, что $L_N > L_{N-1}$. По второму требованию кодовое слово N - 1 не может быть префиксом кодового слова N. Поэтому первые $N-1$ цифры кодового слова N составляют уникальное кодовое слово, а остальные цифры можно отбросить. Если эти два кодовых слова различаются в каком-либо бите, кроме последнего, мы можем отбросить последний бит у каждого слова, получая таким образом лучший код.

На основании этих методов можно построить код, называемый кодом Хаффмана. Начнем с упорядочивания всех символов по убыванию вероятности, так что мы получим символы $(a_1, a_2,..., a_N)$ с вероятностями $P(a_1) >=P(a_2) >= ... >=P(a_N)$.

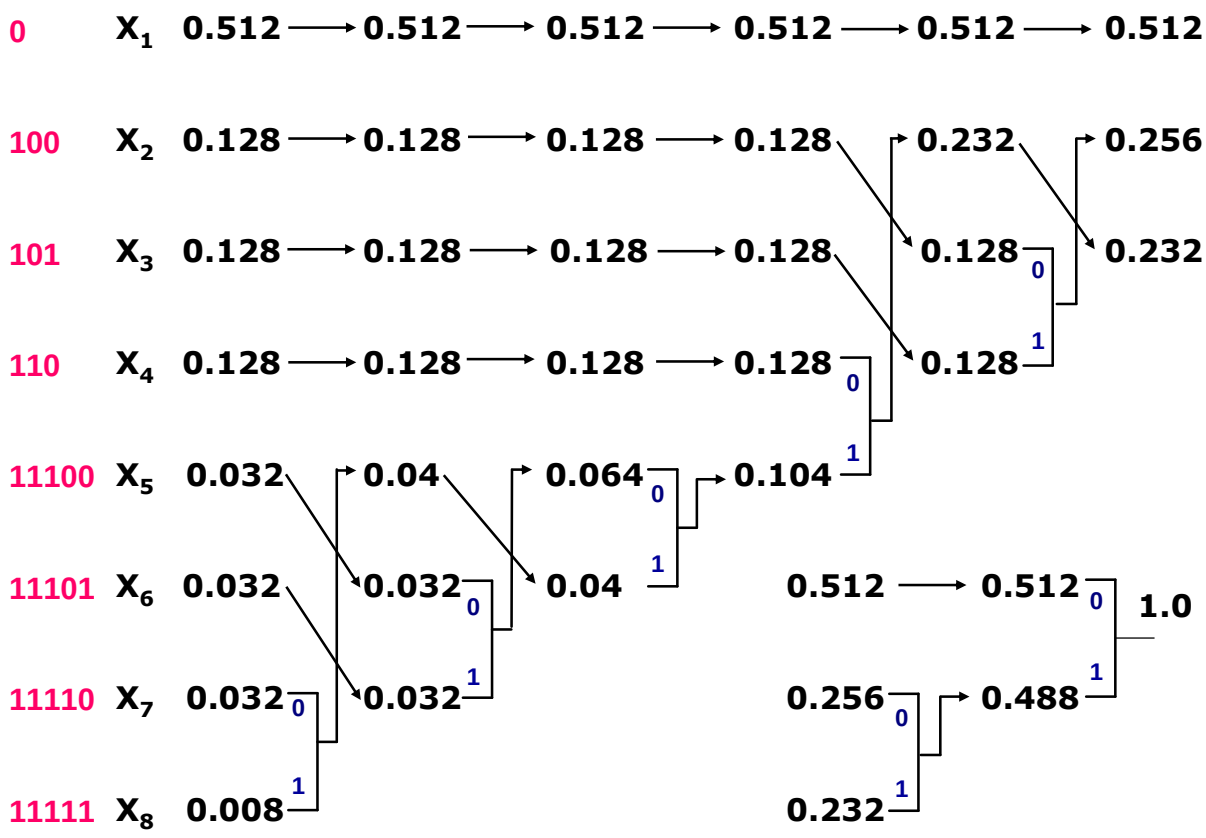
Затем объединим два последних символа в эквивалентный символ с вероятностью $P(a_{N-1}) + P(a_N)$. Коды этих двух символов будут одинаковыми, кроме последних двух цифр. Теперь у нас есть новый набор из N- 1 символов. При необходимости поменяем порядок следования символов таким образом, чтобы получить символы $(b_1, b_2, ... b_{N-1})$ с вероятностями $P(b_1) >= P(b_2)>= ... >= P(b_{N-1})$.

Теперь мы можем повторять этот процесс до тех пор, пока не останется всего два символа.

Т.е мы считаем каждый символ узлом создаваемого дерева и объедининяем два узла с минимальной вероятностью в узел, вероятность которого равна сумме двух соответствующих вероятностей. На каждом шаге будем повторять этот процесс до тех пор, пока не останется только один узел. В результате мы получим дерево, в котором у каждого узла, кроме корневого, одна ветвь отходит направо и две налево.

На каждом узле пометим две левые ветви символами 0 и 1 соответственно. Для каждого символа кодовое слово представляет собой строку меток от корневого узла к исходному символу.

Все получившиеся в результате кодовые слова показаны в левой части рисунка.



В таблице ниже приведены свойства кода Хаффмана для данного примера. Средняя длина кодового слова представляет собой вычисляемую следующим образом ожидаемую величину:

$$E[L] = \sum_{i=1}^{8} L_i P_i = 2.184$$

Символ	Код	P_i	L_i	$P_i L_{i\|}$	$\log(1/P_i)$	$P_i \log(1/P_i)$
X1	0	0.512	1	0.512	0.966	0.494
X2	100	0.128	3	0.384	2.966	0.38
X3	101	0.128	3	0.384	2.966	0.38
X4	110	0.128	3	0.384	2.966	0.38
X5	11100	0.032	5	0.16	4.966	0.159
X6	11101	0.032	5	0.16	4.966	0.159
X7	11110	0.032	5	0.16	4.966	0.159
X8	11111	0.008	5	0.04	6.966	0.056
Средняя длина = 2.184					**Энтропия = 2.176**	

Здесь L_i представляет собой длину i-го кодового слова. Таким образом, например, для сообщений, состоящих из 1000 символов, средняя длина кодированного сообщения равна 2184 бит. При простом кодировании каждого символа тремя битами длина этого сообщения будет составлять 3000 бит.

Энтропия и эффективность кодирования

Рассмотрим случайную переменную с множеством возможных значений ($x_1 x_2 \ldots x_N$), принимаемых с вероятностями ($P_1, P_2 \ldots P_N$), где P, означает вероятность результата x_i.

$$(x_1, x_2 \ldots x_N) \longrightarrow (P_1, P_2 \ldots P_N)$$

Определим нижнюю границу средней длины кода. Мы знаем, что мерой информации для x_i является $\log(1/P)$. Поэтому в идеальном случае мы сможем представить значение x_i- кодовым словом длины

$$L_i = \log\left(1/P_i\right) бит$$

Однако в большинстве случаев $log(l/P_i)$ не является целым числом, и лучшее, что мы можем сделать, — это выбрать ближайшее целое число L_i такое что

$$\log\left(\frac{1}{P_i}\right) \le L_i \le \log\left(\frac{1}{P_i}\right) + 1$$

Умножая на P_i и суммируя по всем кодовым словам, получаем:

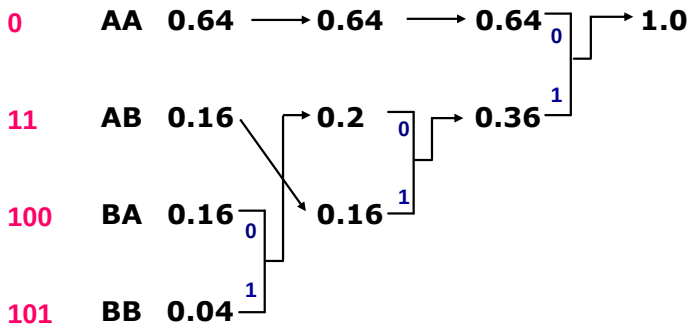
$$\sum_{i=1}^{N} P_i \log\left(\frac{1}{P_i}\right) \le \sum_{i=1}^{N} P_i L_i < \sum_{i=1}^{N} P_i \log\left(\frac{1}{P_i}\right) + \sum_{i=1}^{N} P_i$$

$$H(X) \le E[L] < H(X) + 1$$

Таким образом, оптимальный код позволяет получить среднюю длину кода, не более чем на 1 бит превосходящую энтропию оригинального набора символов. Таким образом, энтропия случайной переменной *X* может интерпретироваться как минимальное среднее число битов, необходимых для отображения одного значения *X.* Можно показать, что код Хаффмана удовлетворяет данному неравенству. Пример приводится в таблице. Средняя длина кода составляет 2,184, а энтропия равна 2,167. Обратите внимание на то, что не все отдельные кодовые слова удовлетворяют неравенству. Однако в среднем это неравенство выполняется.

Объединение символов. Переходные зависимости

Предположим, что у нас есть источник, генерирующий символы из алфавита *X,* состоящего всего из двух символов A и B, встречающихся с вероятностью 0,8 и 0,2. Энтропия при такой схеме составляет 0,8 log(l,25) + 0,2 log(5) = 0,722. Но лучшее, что можно сделать в данном случае, — это использовать по одному биту для каждого кодового слова (например, A = 0, B = 1). Если определить эффективность кода как отношение энтропии источника (среднее число битов информации в символе) к средней длине кодового слова (среднее число битов, используемых для кодирования одного символа), тогда эффективность этого кода будет равна 0,722.



Эффективность кода можно повысить, если объединить символы в блоки и кодировать эти блоки символов. Например, если мы будем кодировать по два символа одновременно, тогда можно рассматривать полученные в результате блоки символов как новый алфавит *Y,* состоящий из четырех символов (AA, AB, BA, BB). Если символы алфавита *X* следуют в сообщениях независимо друг от друга, тогда вероятности, с которыми в них встречаются символы алфавита *Y,* будут равны: $P_{AA} = 0,64$; $P_{AB} = 0,16$; $P_{BA} = 0,16$; $P_{BB} = 0,04$. На рисунке показан код Хаффмана для

Символ	Код	P_i	L_i	P_iL_i	$\log(1/P_i)$	$P_i\log(1/P_i)$
Независимые символы						
AA	0	0,64	1	0,64	0,644	0,412
AB	11	0,16	2	0,32	2,644	0,423
BA	100	0,16	3	0,48	2,644	0,423
BB	101	0,04	3	0,12	4,644	0,186
		Средняя длина = 1,56			Энтропия = 1,444	

такого варианта объединения символов в блоки, а в верхней части таблицы ниже статистика этого кода. В результате мы получаем код со средней длиной 1,56 при энтропии, равной 1,444, таким образом, эффективность кода составляет 0,926, что значительно лучше, чем при кодировании исходных символов без группировки. Обратите внимание на то, что информация источника не изменилась.

Энтропия случайной переменной *X* равна 0,722, а энтропия K равна 0,722 • 2 = 1,444, то есть все также по 0,722 бита на символ. Еще лучших результатов можно достичь, объединив символы по три. В данном случае эффективность кода составит 2,167/2,128 = 0,992. Энтропия все также будет составлять 0,722 бит на символ (2,167/3).

Можно показать, что эффективность оптимального кода для независимых последовательностей символов может быть повышена путем объединения символов, как это было показано ранее. Для блока из *K* символов мы получим следующее соотношение:

$$H(X) \le \frac{E[L]}{K} < H(X) + \frac{1}{K}$$

Таким образом, среднее количество битов на одно значение случайной переменной *X* можно сделать сколь угодно близким к энтропии, выбирая все большие размеры блоков.

В предыдущем пункте предполагалось, что следующий символ в последовательности не зависит от предыдущего символа. Однако если такая зависимость существует, тогда вероятности, связанные с блоками символов, меняются. Например, рассмотрим снова алфавит *X* из двух символов и предположим, что к нему применимы следующие переходные вероятности:

	A	B
A	0.9	0.1
B	0.4	0.6

Таким образом, вероятность того, что за символом A последует символ A, равна 0,9; вероятность того, что за символом B последует символ A, равна 0,1 и т. д. Несложно доказать непротиворечивость матрицы переходов при вероятностях появления отдельных символов A и B, равных 0,8 и 0,2 соответственно.

Теперь разработаем код для алфавита *Y,* составленного из комбинаций этих символов по два. Мы получим:

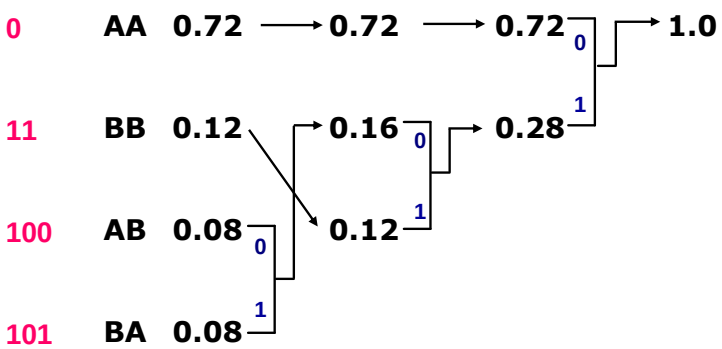
$P_{AA}=P_A*Pr(A|A)=0.8*0.9=0.72$

$P_{AB}=P_A*Pr(B|A)=0.8*0.1=0.08$

$P_{BA}=P_B*Pr(A|B)=0.2*0.4=0.08$

$P_{BB}=P_B*Pr(B|B)=0.2*0.6=0.12$

На рисунке показан код Хаффмана для данного случая, а в средней части таблицы — полученная в результате статистика. Обратите внимание на то, что энтропия Y меньше, чем в случае, когда следующие друг за другом символы независимы (1,292 против 1,444). Поскольку вероятности для отдельных символов остались теми же самыми (P_A = 0,8, P_B = 0,2), то чем это объясняется? Ответ состоит в том, что вероятности переходов добавляют информацию. Когда становится известен первый символ последовательности, мы получаем больше информации о том, какой символ появится следующим. Поэтому неопределенность уменьшается, и суммарная энтропия также снижается. Другими словами, в последовательностях появляется избыточность. Так, например, существует значительная избыточность в предложениях на английском языке.



Символ	Код	P_i	L_i	P_iL_i	$\log(1/P_i)$	$P_i\log(1/P_i)$
Зависимые символы						
AA	0	0,72	1	0,72	0,474	0,341
BB	11	0,12	2	0,24	3,059	0,367
AB	100	0,08	3	0,24	3,644	0,292
BA	101	0,08	3	0,24	3,644	0,292
		Средняя длина = 1,44			Энтропия = 1,292	
Зависимость символов не учитывается						
AA	0	0,72	1	0,72		
AB	11	0,08	2	0,16		
BA	100	0,08	3	0,24		
BB	101	0,12	3	0,36		
		Средняя длина = 1,48				

Для случая зависимых символов средняя длина кода равна 1,44, что дает эффективность 1,292/1,44 = 0,897. В нижней части таблицы показано, что произойдет, если проигнорировать переходные вероятности и предположить, что следующие друг за другом символы независимы даже в том случае, если они зависимы. В этом случае используются те же коды, что и для независимых символов (AA = 0, AB =11, BA= 100, BB =101). Однако теперь для второй по частоте комбинации BB используется самое длинное кодовое слово 101, поэтому средняя длина кодового слова оказывается равной 1,48, что больше средней длины, получавшейся при использовании оптимального кода Хаффмана.

Можно сделать вывод, что большей эффективности в сжатии данных можно достичь, кодируя данные большими блоками, а также учитывая зависимости данных.

Сжатие без потерь

Принцип сжатия данных довольно прост. Практически все формы представления данных (текст, числа, изображение, видео) содержат избыточные элементы. Данные могут быть сжаты с одной из двух целей: для экономии места при хранении или для снижения потребностей в пропускной способности. Также должна существовать возможность распаковать данные, восстановив их исходную форму. Для этого при сжатии данных удаляемые элементы данных должны замещаться другими таким образом, чтобы получатель смог восстановить исходную информацию.

Существует две основные разновидности сжатия данных: сжатие без потерь и сжатие с потерями. При *сжатии без потерь* (lossless compression) информация не теряется и распакованные данные идентичны исходным несжатым данным. Эффективность сжатия без потерь ограничена энтропией источника данных. В случае *сжатия с потерями* (lossy compression) распакованные данные могут быть приближением (с некоторой точностью) к исходным несжатым данным. Например, при сжатии изображений или видеоданных критерий может заключаться в том, что для человеческого глаза распакованное изображение неотличимо от оригинала.

Важным фактором при разработке и выборе алгоритмов сжатия данных является объем работ, требуемый для сжатия и распаковки данных. В общем случае существует компромисс между достижимой степенью сжатия и скоростью работы алгоритма сжатия.

Методы группового кодирования

Групповое кодирование (run-length encoding) представляет собой простой метод сжатия данных без потерь, довольно эффективный для сжатия текста. Этот метод также находит применение в факсимильном сжатии.

Подавление нулей. Групповое кодирование. Факсимильное сжатие

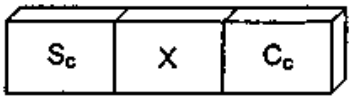
Один из старейших и простейших методов сжатия данных известен как подавление нулей, или подавление пробелов. Он до сих пор применяется в некоторых протоколах передачи данных. В тексте или потоке символов часто встречаются длинные строки пробелов или нулей. В методе подавления нулей передатчик сканирует данные в поиске строк пробелов и заменяет каждую такую строку двухсимвольным кодом. Код состоит из специального управляющего символа, за которым следует число пробелов в строке. Например, пусть имеется код с символами пробела, обозначенными знаком b: **XYZbbbbbQRX**

Эта строка заменяется следующей строкой, в которой S_c представляет собой специальный управляющий символ: **XYZSc5QRX**

Такая схема позволяет сделать короче все строки из трех и более пробелов. В методе подавления нулей приемник ищет в потоке входящих символов специальный символ, используемый для индикации удаленных пробелов. Получив такой символ, получатель понимает, что следующий символ содержит количество удаленных пробелов. По этой информации может быть восстановлен исходный поток данных.

При том что метод подавления нулей представляет собой крайне примитивную форму сжатия данных, его преимущество заключается в том, что он очень легко реализуется. Кроме того, выигрыш даже от применения такого простого метода может быть значительным.

Групповое кодирование представляет собой обобщение метода подавления нулей и применяется для сжатия повторяющихся данных любого типа. На рисунке иллюстрируется применение данного метода к символьным данным. Здесь используются следующие обозначения:



- S_c — специальный символ, указывающий на то, что за ним следуют сжатые данные;
- X — любой повторяющийся символ;
- C_c — счетчик символов, то есть количество повторений сжатого символа.

Примеры сжатия при групповом кодировании:

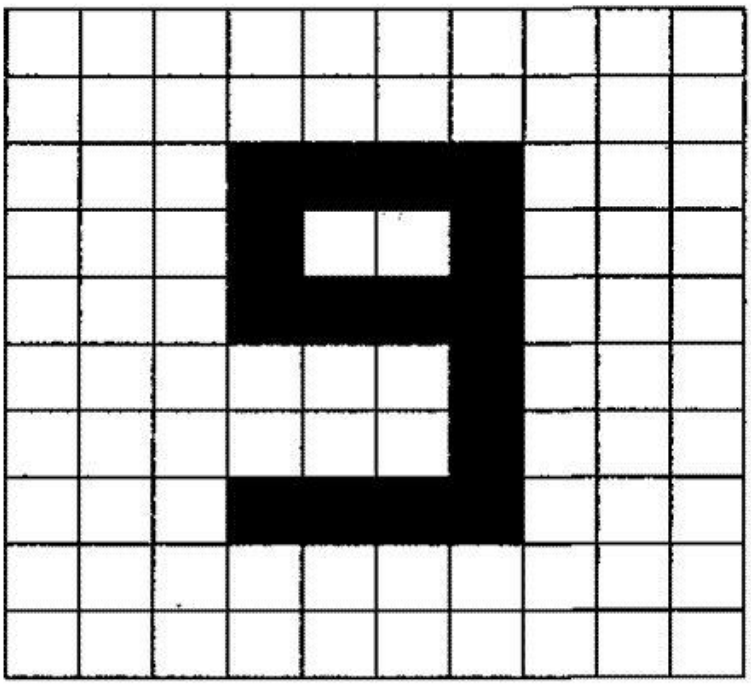
Исходная строка данных	Кодированная строка данных
$******55.72	S_c$*655.72
--------	S$_c$-9
GunsbbbbbbbbbButter	GunsS$_c$b9Butter

Как и в методе подавления нулей, передатчик ищет последовательности повторяющихся символов. В данном случае он заменяет их трехсимвольным кодом. Код состоит из специального индикатора сжатия, за которым следуют сам повторяющийся символ и число его повторений. Таким образом, данный метод позволяет сократить место, занимаемое любой последовательностью из четырех и более одинаковых символов.

Эффективность метода группового кодирования зависит от того, насколько часто в исходных данных встречаются последовательности повторяющихся символов, и от средней длины таких серий. Стандартной мерой эффективности сжатия является коэффициент сжатия, представляющий собой отношение длины несжатых данных к длине сжатых данных (включая символы кодирования).

Групповое кодирование было одним из первых методов сжатия факсимильных сообщений, но теперь оно более не используется для этой цели. Однако этот метод следует изучить, так как он применяется в более сложных методах сжатия изображений.

При использовании метода группового кодирования для изображений вместо отсканированной и оцифрованной линии передаются длины серий белых и черных элементов изображения. На рисунке показан пример применения метода группового кодирования к простому изображению размером 10x10 точек. Это может быть факсимильное изображение или отсканированное растровое компьютерное изображение. Изображение формата 10 x 10 легко преобразуется в 100-битовый код. В данном примере каждый пиксел представляется одним битом, обозначающим белый или черный цвет. Код длин серий состоит из длин чередующихся черных и белых последовательностей. Поскольку при таком кодировании изображения черный и белый цвета всегда чередуются, нет необходимости в использовании специального символа, указывающего цвет серии.



0000000000
0000000000
000**1111**000
000**1**00**1**000
000**1111**000
000000**1**000
000000**1**000
000**1111**000
0000000000
0000000000
Длина равна 100 бит

ДВОИЧНЫЙ КОД:
23W 4B 6W 1B 2W 1B 6W 4B 9W 1B 9W 1B 6W 4B 23W
или
23 4 6 1 2 1 6 4 9 1 9 1 6 4 23
Длина равна 15 символов или 120 бит

Таким образом, кодированный поток данных представляет собой строку чисел, обозначающих длины чередующихся серий черных и белых точек. Обратите внимание на то, что в данном простом примере кодированные данные занимают больше места, чем исходные. Однако в случае применения данного метода к типичной странице текста этот метод будет сжимать данные. Тем не менее, это далеко не самый эффективный способ сжатия изображений.

Методы сжатия данных очень важны для широкого применения цифровых факсимильных аппаратов. Для примера рассмотрим типичную страницу, отсканированную с разрешением в 200 пелов (белых или черных точек) на дюйм (что является приемлемой, но не высокой степенью разрешения). В результате такая страница содержит 3 740 000 бит (8,5 дюймов x 11 дюймов x 40 000 пелов на квадратный дюйм). При базовой скорости службы ISDN в 64 Кбит/с передача такой страницы займет около одной минуты. Обычно пользователи ожидают от систем работы со скоростью, близкой к скорости копирования, то есть несколько секунд на страницу. Для удовлетворения данного требования, если не прибегать к увеличению скорости передачи данных в линии, необходимо использовать метод сжатия данных.

Стандартизированы два метода сжатия данных без потерь для факсимильной связи: модифицированный код Хаффмана и модифицированный код READ (Relative Element Address Designate — относительное назначение адресов элементов). Модифицированный код Хаффмана используется по умолчанию в факсимильной аппаратуре группы 3, в которой модифицированный код READ может применяться факультативно. В аппаратуре группы 4 применяется модифицированный код READ. Кратко эти два стандарта (группы 3 и 4) характеризуются следующим образом:

- *Группа 3.* Первый цифровой факсимильный стандарт. Эта система обеспечивает только кодирование черных и белых значений с плотностью сканирования в 200 точек на дюйм по горизонтали и от 100 до 200 по вертикали. В аппаратуре группы 3 используются цифровая схема кодирования, а также средства уменьшения избыточной информации в сигнале документа перед модуляцией. Предполагается, что передача сигнала осуществляется через модем по аналоговой телефонной линии. Передача данных ускоряется в три и более раз по сравнению с группой 2.
- *Группа 4.* Также представляет собой черно-белый цифровой факсимильный стандарт. Эта группа предназначена для использования в цифровых сетях со скоростями до 64 Кбит/с при условии безошибочного приема. Стандартизированы разрешения от 200 до 400 точек на дюйм. Как и в группе 3, для снижения количества передаваемых битов применяются методы сжатия. В аппаратуре группы 4 время передачи одной страницы сокращено до нескольких секунд по сравнению с несколькими минутами в предыдущих стандартах.

Модифицированный код Хаффмана

В типичном документе черные и белые области имеют тенденцию к объединению. Если рассматривать документ как последовательность линий и обратить внимание на расположение участков белого и черного в линии, можно обнаружить длинные серии белых и черных точек. Благодаря этому свойству можно предположить, что сжатие на основе метода группового кодирования даст хороший результат. Состоящие из двух значений входные данные преобразуются в длины серий, которые затем кодируются для передачи. Кроме того, поскольку в общем случае длинные серии черных или белых точек менее вероятны, чем короткие, можно воспользоваться преимуществом кодирования последовательностей переменной длины. Для кодирования факсимильных документов может применяться код Хаффмана, который рассматривался на прошлой лекции. Этот метод можно применить к изображению построчно, кодируя последовательности черных и белых точек. Например, предположим, что при сканировании отдельной строки получается следующая последовательность черных и белых точек: W7, B7, W4, B8, W4, B7, W10 (здесь W означает белый, а B — черный). Если рассматривать каждый из этих элементов как символ алфавита источника, тогда для кодирования этих данных может быть использован метод кодирования Хаффмана. Однако поскольку стандарт требует по меньшей мере 1728 точек на линию, количество различных кодов, а следовательно, и средняя длина кода будет очень большой. Альтернативой является модифицированный метод кодирования Хаффмана. В этом методе длина серии *N* рассматривается как сумма двух слагаемых:

$$N = 64m + n;\ m = 0, 1, 2, \ldots 27;\ n = 0, 1, 2, \ldots 63$$

То есть длина каждой серии черных или белых точек считается величиной, кратной 64 с остатком.

Теперь каждая длина серии может быть представлена двумя значениями, одним для *m* и другим для *n,* и эти значения могут кодироваться при помощи метода Хаффмана. Например, строка из 200 черных точек подряд может быть выражена как 64 * 3 + 8. В стандарте определены восемь образцов документов и рассчитаны вероятности нахождения в документах серий различных длин. Поскольку для серий из черных и белых точек эти вероятности различаются, были сосчитаны два множества вероятностей. Длина серии делится на 64, после чего частное и остаток от деления кодируются двумя кодовыми словами. Если длина серии меньше 64 (частное от деления равно нулю), то такая длина серии кодируется только кодовым словом остатка. Серии длиной более 64 точек кодируются двумя кодами: кодом остатка (*n*) и кодом кратности *(m).* Каждая строка заканчивается уникальным кодовым словом EOL (End Of Line — конец строки). Это кодовое слово, никогда не встречающееся в строках данных, обеспечивает восстановление синхронизации в случае ошибок. Внутри строки должны чередоваться кодовые слова для белых и черных серий. Обратите внимание на то, что для белых и черных серий используются различные кодовые слова. Это обеспечивает дополнительную защиту от ошибок. Наконец, по соглашению каждая строка начинается с белой серии. Если первая точка в линии черная, то используется белая серия нулевой длины.

Арифметическое кодирование

Метод кодирования Хаффмана хотя и является простым в реализации и относительно эффективным, тем не менее, не позволяет достичь максимального коэффициента сжатия, если только все вероятности не представляют собой отрицательных степеней числа 2. Арифметическое кодирование предназначено для достижения более высоких коэффициентов сжатия за счет усложнения вычислений. При этом изображение обрабатывается таким образом, чтобы вероятности приближались к отрицательным степеням числа 2. Арифметическое кодирование используется в стандартах JPEG и MPEG. Мы начнем с концепции, лежащей в основе арифметического кодирования, а затем рассмотрим некоторые детали.

Основная концепция

Вспомним из обсуждения метода кодирования Хаффмана, что если отдельные кодируемые символы встречаются в исходном тексте с вероятностью P_i, то в лучшем случае мы можем использовать для каждого символа код длины L_i, удовлетворяющей следующему неравенству:

$$\log\left(\frac{1}{P_i}\right) \le L_i \le \log\left(\frac{1}{P_i}\right)+1$$

Здесь P_i представляет собой вероятность, с которой *i*-й символ встречается в исходных данных, а L_i — длину двоичного кода, которым кодируется данный символ. Исходя из данного неравенства, можно показать, что

$$H(X) \le E[L] < H(X)+1$$

Здесь *H(X)* представляет собой энтропию множества символов *X,* а *E[L]* является средней длиной кода для множества *X.* В схеме сжатия без потерь *H(X)* всегда будет нижней границей объема сжатых данных, так как *H(X)* определяет среднее количество информации, содержащейся в символьном наборе. Наконец, было показано, что эффективность алгоритма сжатия может быть повышена путем объединения символов и одновременного кодирования блоков по *K* символов каждый. В результате мы получаем следующее неравенство:

$$H(X) \le \frac{E[L]}{K} < H(X)+\frac{1}{K}$$

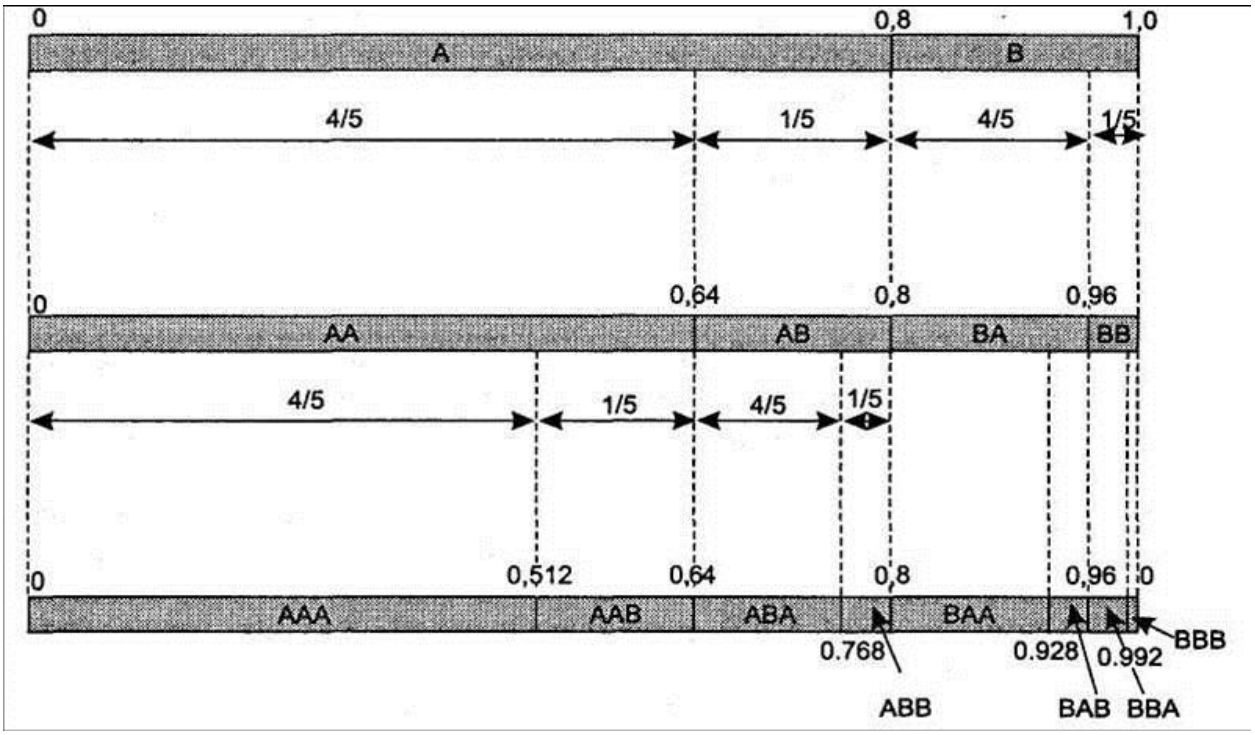
Таким образом, существует способ повышения эффективности алгоритма сжатия. Если нам нужно отправить сообщение длиной в *N* символов, мы можем использовать блок длиной *N.* Таким образом, мы обращаемся с каждым сообщением как с одним из M^N возможных результатов, где *M* представляет собой количество атомарных символов, а *N* — длину сообщения. В качестве примера рассмотрим случай, обсуждавшийся ранее, в котором имеются два символа, A и B, встречающиеся в исходном тексте с вероятностями 0,8 и 0,2. Если мы передаем сообщение длиной в 3 символа *(M* = 2, *N*= 3), тогда существуют 2^3 = 8 возможных результатов. Если появление в потоке данных каждого символа не зависит от предыдущих символов, тогда мы можем написать вероятности всех 8 результатов:

$P_{AAA} = 0{,}512,$	$P_{BAA} = 0{,}128,$
$P_{AAB} = 0{,}128,$	$P_{BAB} = 0{,}032,$
$P_{ABA} = 0{,}128,$	$P_{BBA} = 0{,}032,$
$P_{ABB} = 0{,}032,$	$P_{BBB} = 0{,}008.$

Один из возможных методов состоит в применении кода. Для сообщения с восемью возможными результатами это разумно, но при очень длинных сообщениях использование этого метода приведет к большим вычислительным затратам. Метод арифметического кодирования предоставляет остроумную альтернативу. Заметим, что сумма вероятностей всех возможных сообщений должна быть равна 1,0. Таким образом, можно разложить все результаты на интервале [0,1] в виде отрезков, длины которых равны их вероятностям.

На рисунке также показан простой метод генерирования интервалов по одному символу. Эта процедура выглядит следующим образом:

1. Алгоритм начинается с позиции первого символа в сообщении, а интервал делится в соответствии с вероятностями отдельных символов. В данном случае имеется два символа, A и B, встречающиеся в исходном тексте с вероятностями 0,8 и 0,2. Поэтому интервал [0, 1] разбивается на подынтервалы [0, 0,8) и [0,8, 1). Граница между интервалами может быть отнесена произвольно к любому из интервалов.
2. Каждый подынтервал разбивается тем же способом, который применялся на шаге 1. То есть каждый подынтервал дробится на доли в соотношении 4:1.
3. Этот процесс повторяется для *N* символьных позиций (сообщение длины *N).*



Теперь результату каждого сообщения поставлен в соответствие интервал, пропорциональный вероятности сообщения. Таким образом, каждый результат может быть представлен двоичным дробным числом, равным некоторой точке на интервале. Будем использовать для этого нижнюю границу каждого интервала.

Результаты такой операции показаны в пятой колонке таблицы ниже, при этом используются пять значащих цифр правее двоичной запятой. Такой точности достаточно для однозначной идентификации каждого интервала, а следовательно, и каждого результата. Поэтому мы можем использовать дробную часть каждого значения в качестве кода для соответствующего результата (например, ABA= 10100). Однако полученные коды можно усовершенствовать. Например, можно удалить из кода некоторые цифры, если только это не приведет к неразличимости кодов (то есть ни один код не должен совпадать с начальными цифрами другого кода). Результат этой операции показан в шестой колонке таблицы. Обратите внимание на то, что в общем случае более короткие интервалы идентифицируются большим числом битов. В этом есть смысл: нижняя граница короткого интервала близка к нижней границе следующего интервала. Поэтому двоичные дроби для этих двух границ будут совпадать в нескольких разрядах. Чем меньше интервал, тем больше одинаковых разрядов в соответствующих дробях.

Последова-тельность	P_i	Кумуля-тивная вероят-ность	Интервал	Двоичное представление нижней границы	Код	P_iL_i	$P_i\log(1/P_i)$
AAA	0.512	0.512	[0,0.512]	0.00000	0	0.512	0.494
AAB	0.128	0.64	[0.512,0.64]	0.10000	100	0.384	0.38
ABA	0.128	0.768	[0.64,0.768]	0.10100	101	0.384	0.38
ABB	0.032	0.8	[0.768,0.8]	0.11000	11000	0.16	0.159
BAA	0.128	0.928	[0.8,0.928]	0.11001	11001	0.64	0.38
BAB	0.032	0.96	[0.928,0.96]	0.11101	111	0.096	0.159
BBA	0.032	0.992	[0.96,0.992]	0.11110	11110	0.096	0.159
BBB	0.008	1.0	[0.992,1.0]	0.11111	11111	0.04	0.056
						2.408	2.167

Таким образом, мы разработали новый метод назначения кодов сообщениям. В последних двух колонках таблицы средняя длина нового кода сравнивается с энтропией множества сообщений. Несмотря на всю свою внешнюю привлекательность новый метод обладает двумя существенными недостатками:

- Если все возможные интервалы вычислять заранее, то объем необходимых вычислений будет огромен. Например, если длина сообщения равна 1000 символов и используется алфавит из 128 символов (например, набор символов ASCII), тогда число всех возможных сообщений будет составлять $128^{1000} = 10^{2107}$.
- Описанная схема применима только к сообщениям фиксированной длины. Метод арифметического кодирования позволяет устранить оба недостатка.

Чистое арифметическое кодирование

Мы начнем с простейшей формы алгоритма арифметического кодирования. Хотя этот алгоритм преодолевает упомянутые ранее недостатки, в нем сохраняются некоторые вычислительные сложности. Тем не менее, проще понять метод арифметического кодирования, рассмотрев сначала более простую форму. При арифметическом кодировании нет необходимости заранее генерировать интервалы для всех возможных сообщений. Вместо этого интервалы для отдельного сообщения вычисляются посимвольно. Таким образом, для передачи одного сообщения вычисляется только один интервал. Кроме того, поскольку процесс является посимвольным, нет необходимости фиксировать длину сообщения заранее.

Вычисления продолжаются до тех пор, пока не будет достигнут конец сообщения.

Базовый алгоритм

Каждый шаг алгоритма начинается с полуоткрытого интервала *[L, H),* вначале инициализируемого как [0,1):

1. Текущий интервал разбивается на подынтервалы по одному для каждого возможного символа. Каждый подынтервал пропорционален вероятности, с которой соответствующий символ встречается в тексте.
2. Выбирается подынтервал, соответствующий текущему символу. Этот подынтервал становится текущим интервалом.
3. Если обработано все сообщение, выполняется следующий шаг, если нет, возвращаемся к шагу 1.
4. Выводится количество битов, достаточное для того, чтобы можно было отличить данный интервал от двух соседних интервалов.
5. Выводится некий специальный код конца сообщения, чтобы уведомить получателя.

На первом шаге нет необходимости вычислять все подынтервалы. Вместо этого вычисляются только подынтервал, соответствующий текущему символу.

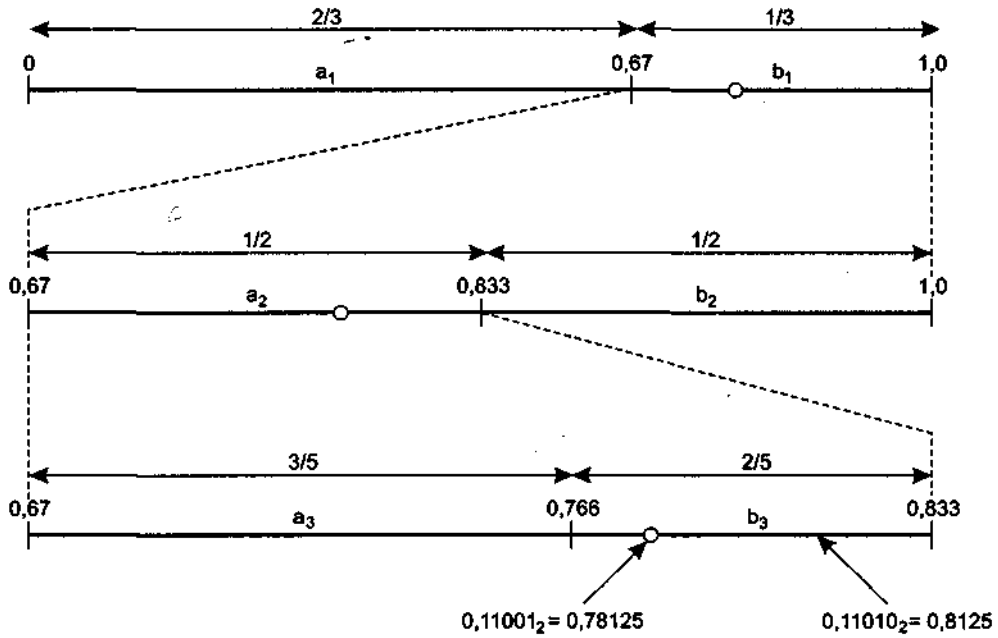
Пример

Пусть имеются два символа, a и b, вероятности которых зависят от положения в трехсимвольном общении:

$$\Pr(a_1) = 2/3, \quad \Pr(b_1) = 1/3,$$
$$\Pr(a_2) = 1/2, \quad \Pr(b_2) = 1/2,$$
$$\Pr(a_3) = 3/5, \quad \Pr(b_3) = 2/5.$$

Здесь $\Pr(a_i)$ представляет собой вероятность того, что символ a встретится в i-й позиции сообщения. В данном случае кодируется сообщение «bab».

На рисунке ниже показано арифметическое кодирование этого сообщения. Данному сообщению соответствует конечный интервал [23/30,5/6) = [0,766,0,833). В двоичном виде конечный интервал выглядит следующим образом: [0,110001..., 0,110101...). Обратите внимание на то, что все числа, начинающиеся с 0,110001, целиком попадают в интервал, но существуют некоторые числа, начинающиеся с 0,1100, не попадающие в этот интервал (например, 0,1100001 < 23/30). Поэтому кода 11001 достаточно, чтобы однозначно идентифицировать интервал и, следовательно, однозначно идентифицировать сообщение. В общем случае, чем меньше конечный интервал (соответствующий менее вероятному сообщению), тем больше битов кода требуется для уникальной идентификации интервала.



Декодирование

Процесс декодирования выполняется по той же общей поэтапной схеме. На этот раз обнаруживаются последовательные подынтервалы, приводящие к конечному интервалу и открытию соответствующих сообщений. Общие правила выглядят следующим образом. Опять же, начнем с интервала [0,1):

1. Разделим текущий интервал на подынтервалы, по одному для каждого символа. Длина каждого подынтервала пропорциональна вероятности появления соответствующего символа.
2. Выберем подынтервал, содержащий код, выраженный в виде двоичной дроби. Выведем символ, определяющий этот код.

Эти два шага повторяются до тех пор, пока не будет распаковано все сообщение. Существует несколько способов определить, что сообщение распаковано полностью. К исходному сообщению может быть добавлен символ конца сообщения. В этом случае процесс декодирования остановится при обнаружении этого символа. Другой способ состоит в том, чтобы при декодировании на каждом шаге проверять, все ли двоичные дроби, начинающиеся с кода, попадают в один интервал, и прекратить декодирование, когда это условие перестанет выполняться.

Процесс декодирования, как и процесс кодирования, иллюстрирует тем же рисунком. Численное значение конечного кода (0,78125 = 0,11001г) обозначается кружком на каждой линии процесса.

Инкрементное арифметическое кодирование

У только что описанного алгоритма чистого арифметического кодирования есть два недостатка. Во-первых, уменьшающиеся размеры текущего интервала требуют все более точных вычислений. Во-вторых, код не может быть послан, пока не обработано все сообщение.

Метод инкрементного арифметического кодирования позволяет решить обе проблемы. При использовании этой техники перед выполнением каждого следующего шага алгоритма текущий интервал всегда увеличивается, а кодовые биты генерируются на каждом шаге и могут передаваться или сохраняться после каждого шага.

Описание алгоритма

Каждый шаг алгоритма начинается с полуоткрытого интервала *[L, H),* который инициализируется значением [0,1). Кроме того, используется *счетчик следования* (follow counter), инициализируемый значением 0:

1. Текущий интервал разбивается на подынтервалы по одному для каждого возможного символа. Длина каждого подынтервала пропорциональна вероятности появления соответствующего символа.
2. Выбирается подынтервал, соответствующий текущему символу.
3. Следующие шаги выполняются в цикле до тех пор, пока не выполнится условие выхода из цикла:

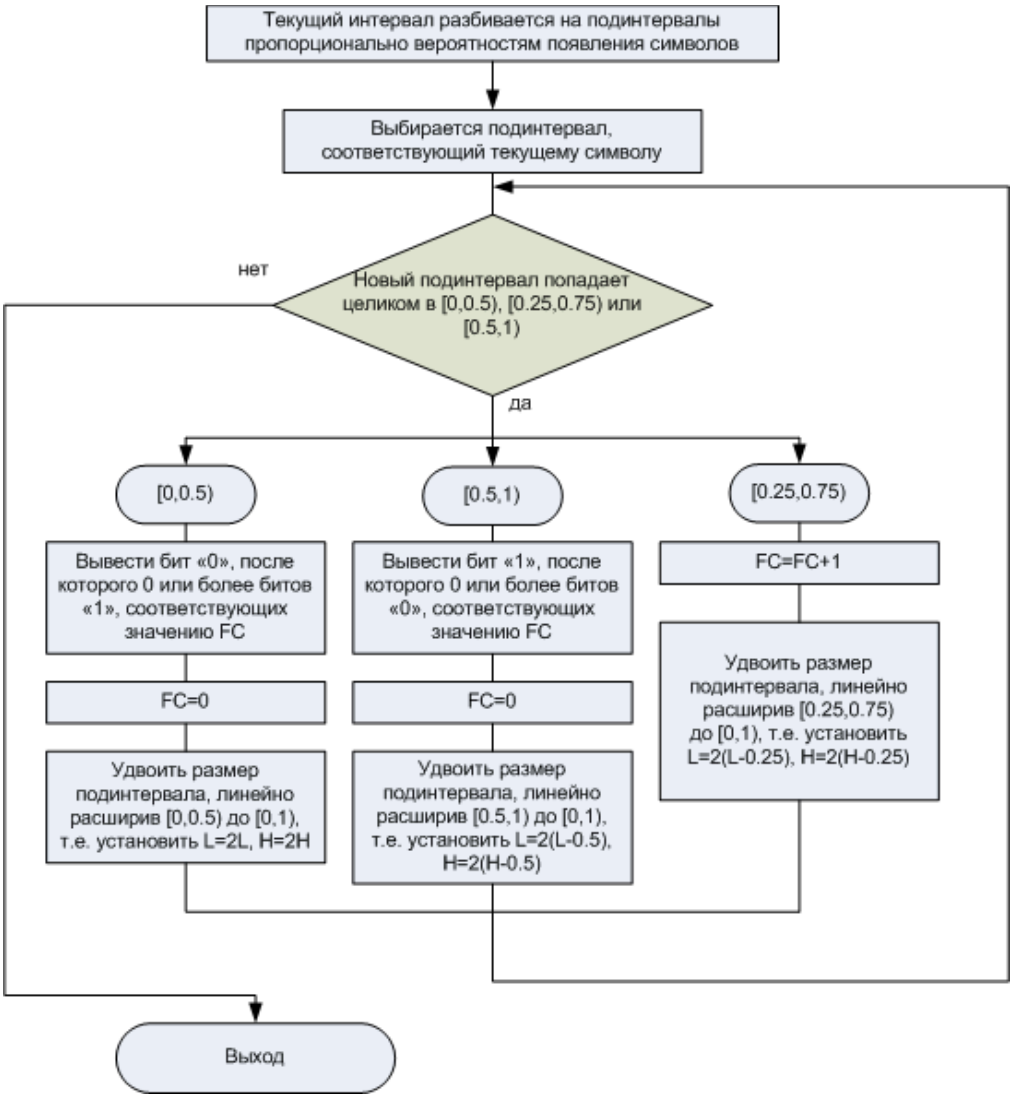
a. Если новый подынтервал не попадает целиком в один из следующих интервалов: [0, 0,5), [0,25, 0,75) или [0,5,1), — выйти из цикла.

b. Если новый подынтервал попадает в интервал [0,0,5), вывести бит 0, после которого ноль или более битов 1, соответствующих значению счетчика следования, после чего значение счетчика следования установить на ноль. Удвоить размер подынтервала, линейно расширив [0, 0,5) до [0,1). То есть установить $L = 2L$ и $H = 2H$.

c. Если новый подынтервал попадает в интервал [0,5, 1,0), вывести бит 1, после которого ноль или более битов 0, соответствующих значению счетчика следования, после чего значение счетчика следования установить на ноль. Удвоить размер подынтервала, линейно расширив [0,5,1) до [0,1). То есть установить $L = 2(1 - 0,5)$ и $H= 2(H- 0,5)$.

d. Если новый подынтервал попадает в интервал [0,25 0,75), увеличить на единицу значение счетчика следования. Удвоить размер подынтервала,, линейно расширив [0,25, 0,75) до [0,1). То есть установить $L = 2(L - 0,25)$ и $H=2(H-0,25)$.

Шаги с 1 по 3 повторяются до тех пор, пока не будет обработано все сообщение. Вот что, по сути, выполняет данный алгоритм. Если новый подынтервал целиком попадает в интервал [0,0,5) или [0,5,1,0), алгоритм формирует ведущий бит, указывающий, в какую половину интервала попадает новый подынтервал, после чего интервал удваивается, так что новый интервал отражает только неизвестную часть конечного интервала. Посмотрим, как это работает. Предположим, что на первом шаге формируется подынтервал в верхней половине единичного интервала Таким образом мы узнаем, что конечный подынтервал также должен находиться в верхней половине единичного интервала, и поэтому старший бит конечного кода должен быть равен 1. Когда этот бит становится известным, вторую половину единичною интервала можно проигнорировать, а подынтервал удвоить для определения следующего бита кода.

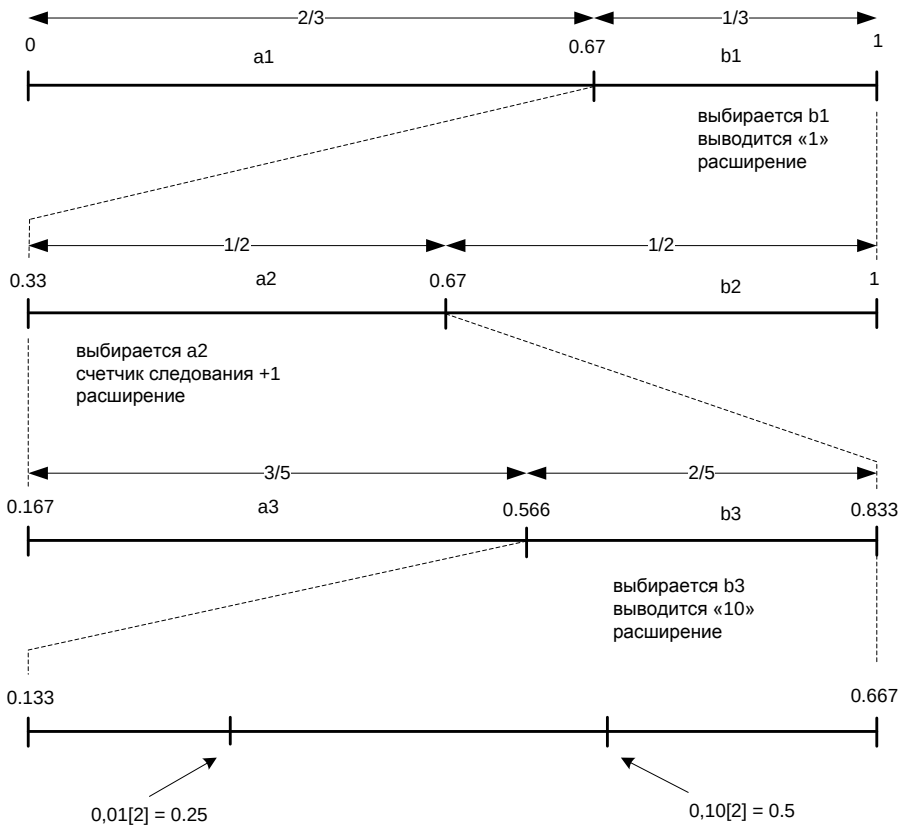


Если конечные точки текущего интервала близки к точке 0,5, но находятся по разные стороны от этой точки, тогда расположение следующего результата еще не определено. Однако каким бы он ни был, следующий бит будет иметь противоположное значение. Алгоритм следит за следующим битом и симметрично расширяет интервал вокруг значения 0,5.

На рисунке ниже показан процесс кодирования, в котором используется сообщение «bab».

В данном примере интервал расширяется один раз для каждого входного символа, но это не обязательно так. Может быть ноль, одно или несколько удвоений на символ. Когда последний символ обработан, результат представляет собой 110. К этому моменту конечный интервал равен [2/15…2/3). Поскольку все двоичные числа в этом интервале начинаются с 0,01, для однозначной идентификации этого диапазона достаточно кода 01.

Такие алгоритмы, как метод Хаффмана и арифметическое кодирование, основаны на знании оценки статистики сжимаемых данных. Эти алгоритмы не адаптируются к изменениям в представлении данных. Кроме того, ограничения памяти ставят предел их способности фиксировать взаимосвязи высокого порядка между словами и фразами текста. Другой подход, доказавший свою эффективность, используется в семействе методов, известных под названием алгоритмов совпадения строк.



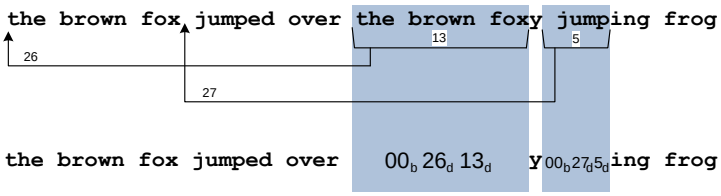
Алгоритмы совпадения строк

В то время как в методе Хаффмана и при арифметическом кодировании входные последовательности фиксированной длины преобразуются в коды переменной длины, алгоритмы совпадения строк преобразуют входные последовательности переменной длины в коды фиксированной длины. Кроме того, алгоритмы совпадения строк не выдвигают априорных предположений о статистике данных, а адаптируются к изменениям в характере входных данных по мере их обработки.

Все алгоритмы данной категории обязаны своим происхождением израильским исследователям Якову Зива (Jacob Zib) и Аврааму Лемпелю (Abraham Lempel). В 1977 г. они описали метод, основанный на буфере скользящего окна, в котором содержится только что обработанный текст. Этот алгоритм обычно называют LZ77. Разновидность этого алгоритма используется в популярной в Интернете схеме сжатия zip (PKZIP, gzip и т. д.). В следующем году те же авторы описали улучшенную версию этого алгоритма, основанного на структурированном в виде дерева словаре. Этот алгоритм называется LZ78. Затем алгоритм LZ78 был усовершенствован и назван LZW. Многие программы сжатия основаны на алгоритмах LZ78 и LZW, включая стандарт *V.42bis* предназначенный для голосовых модемов, популярный формат сжатия изображений GIF, а также программу сжатия данных в операционной системе UNIX. В алгоритмах LZ77, LZ78 и их вариантах используется тот факт, что слова и фразы в потоке текста (элементы изображения в случае GIF) повторяются с большой вероятностью. Когда встречается повторяющаяся последовательность, она может быть заменена коротким кодом. Программа сжатия ищет подобные повторяющиеся участки текста и «на лету» формирует коды, которыми заменяет их на выходе. Те же коды могут использоваться для обозначения новых последовательностей. Алгоритм должен быть определен таким образом, чтобы программа распаковки могла найти текущее соответствие между кодами и последовательностями исходных данных.

Другой способ повышения эффективности алгоритмов совпадения строк состоит в том, чтобы учитывать энтропию входных данных, рассматриваемых в виде последовательности строк. Чем выше уровень агрегации (объединения) входных символов, тем ниже энтропия. Поэтому следует ожидать, что, используя совпадения строк, можно добиться высоких коэффициентов сжатия.

Прежде чем перейти к изучению деталей алгоритма LZ77, рассмотрим простой пример. Пусть имеется следующая бессмысленная фраза:



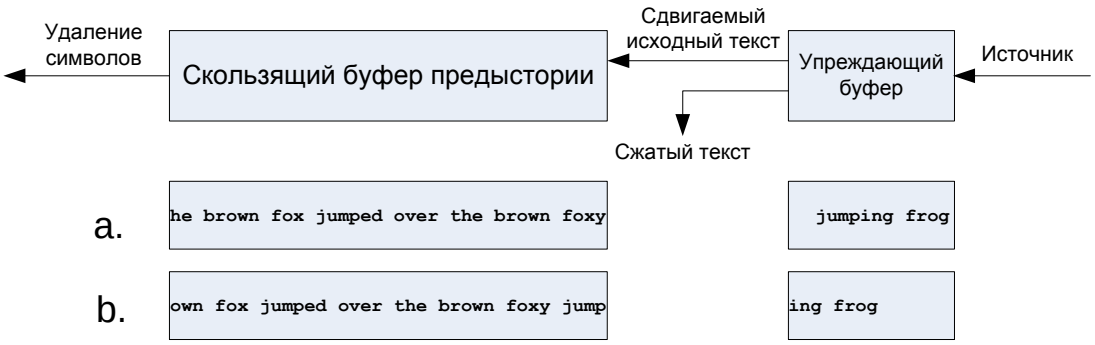
Алгоритм обрабатывает этот текст слева направо. Вначале каждый символ преобразуется в 9-битовый код, состоящий из двоичной единицы, за которой следует 8-битовое ASCII-представление символа. Во время обработки текста алгоритм ищет повторяющиеся последовательности. Когда встречается повторяющаяся последовательность, алгоритм ищет конец такой последовательности, то есть каждый раз алгоритм отыскивает как можно больше символов. Первой такой последовательностью является the brown fox. Повторяющаяся последовательность заменяется указателем на первый экземпляр последовательности и длиной этой последовательности. В данном случае последовательность the brown fox встречается на 26 позиций раньше и имеет длину 13 символов. Для данного примера рассмотрим два варианта кодирования: 8-битовый указатель и 4-битовая длина или 12-битовый указатель и 6-битовая длина. При этом 2-битовый заголовок указывает, который вариант выбран: 00 означает первый вариант, а 01 — второй вариант. Таким образом, второй экземпляр последовательности the brown fox кодируется как <00_b><26_d><13_d> или 00 00011010 1101.

Оставшиеся части сжатого сообщения представляют собой букву y, последовательность <00_b><27_d><5_d>, заменяющую последовательность, состоящую из пробела, за которым следует строка jump, и последовательность символов «ing frog».

Т.о., сжатое сообщение состоит из 35 9-битовых символов и двух кодов, то есть всего 343 бит. Таким образом, при 424 бит в несжатом сообщении коэффициент сжатия данного метода в этом примере составил 1,24.

В алгоритме сжатия LZ77 и его вариантах используются два буфера. В скользящем буфере предыстории хранятся *N* только что обработанных символов источника, а в упреждающем буфере содержатся следующие *L* символов, ожидающих обработки. Алгоритм пытается найти соответствие между двумя и более символами из начала упреждающего буфера и строкой в скользящем буфере предыстории. Если соответствие не обнаружено, первый символ упреждающего буфера выводится как 9-битовый символ, а также сдвигается в скользящее окно, при этом самый старый символ выбрасывается из скользящего окна. Если совпадение обнаруживается, алгоритм продолжает искать самое длинное совпадение. Затем строка, для которой найдено соответствие, выводится в виде триплета (индикатор, указатель, длина). Затем из скользящего окна выдвигаются столько символов, сколько содержится в кодированной триплетом последовательности, и столько же новых символов исходного текста помещаются в скользящее окно.

На нижней части рисунка показана работа данной схемы с последовательностью из нашего примера. В иллюстрации предполагается использование 39-символьного скользящего окна и 13-символьного упреждающего буфера. В верхней части примера первые 40 символов были обработаны, и несжатая версия последних 39 символов находится в скользящем окне. Остальная часть сообщения находится в упреждающем буфере. Алгоритм сжатия находит следующее совпадение, сдвигает 5 символов из упреждающего буфера в скользящее окно и формирует код для этой строки. Состояние буфера после этих операций показано в нижней части примера.



Несмотря на то что алгоритм LZ77 эффективен и адаптируется к природе текущего входного потока, он обладает рядом недостатков. Для поиска совпадений в предшествующем тексте этот алгоритм использует окно конечного размера. Если размер текста намного превышает размер окна, то многие потенциальные совпадения остаются необнаруженными. Размер окна может быть увеличен, но это приведет к двум нежелательным эффектам. Во-первых, время работы алгоритма возрастет. Во-вторых, придется увеличить размер поля указателя.

Процесс распаковки текста, сжатого при помощи алгоритма LZ77, прост. Алгоритм распаковки должен сохранять последние N символов распакованного результата. Когда встречается закодированная строка, алгоритм распаковки заменяет код с помощью долей указателя и длины.

Алгоритмы LZ78 и LZW

Алгоритмы LZ78 и LZW пытаются обойти ограничения схемы LZ77, вместо ограниченного окна создавая словари. Как и схема LZ77, алгоритм LZW (и LZ78) поддерживает словарь строк вместе с их кодами как для сжатия, так и для распаковки. Когда любая строка словаря появляется на входе алгоритма сжатия, эта строка заменяется кодом. Распаковщик, наоборот, заменяет коды соответствующими им строками. По мере работы алгоритма сжатия к словарю добавляются новые строки.

В любой момент времени словарь содержит все односимвольные строки. Механизм работает так, что любые ведущие подстроки присутствующей в словаре строки также могут использоваться в качестве элементов словаря. Например, если в словаре есть строка MEOW, снабженная своим уникальным кодовым словом, тогда строки MEO и ME также присутствуют в словаре и у каждой есть свое уникальное кодовое слово. Логически словарь может быть представлен в виде набора деревьев, при этом корень каждого дерева соответствует символу алфавита. Таким образом, по умолчанию имеется 256 деревьев (все возможные 8-битовые символы).

Рисунок ниже иллюстрирует работу алгоритма сжатия, для которого используется трехсимвольный алфавит. Вначале в словаре находятся только односимвольные строки (верхняя часть таблицы). Входные данные считываются слева направо. Поскольку в таблице нет совпадающих строк длиннее a, для этой строки выводится код 1, а к таблице добавляется новая строка ab, которой назначается код 4. Затем для начала новой строки используется символ b. Поскольку строки ba нет в словаре, она добавляется в словарь с кодом 5, а в выходной поток направляется символ b. Процесс продолжается подобным образом.

Интересный аспект работы семейства алгоритмов LZ78 заключается в том, что словарь не передается явно алгоритму распаковки. Вместо этого алгоритм распаковки создает идентичный словарь в процессе распаковки. Это возможно благодаря тому, что при распаковке строка, соответствующая коду, всегда встречается прежде самого кода.

Передатчик:

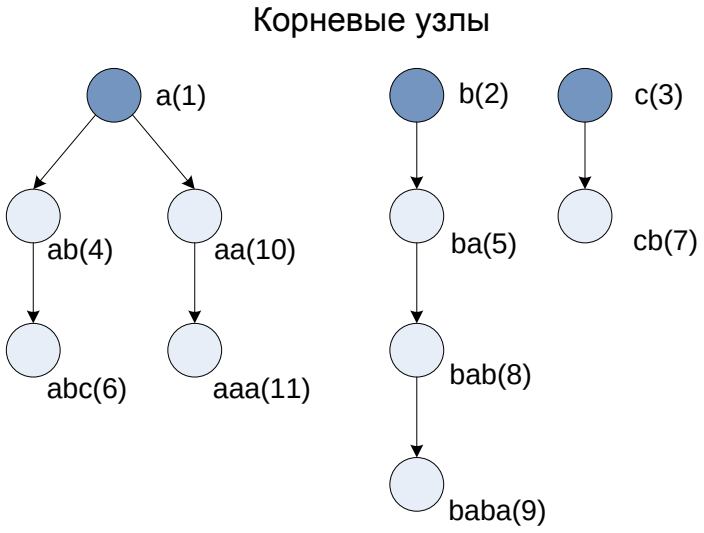
Входные символы:	a	b	a	b	c	b	a	b	a	b	a	a	a	a	a	a
Выходные коды:	1	2	4		3	5		8			1	10		11		
Новые строки, добавленные в таблицу:			4			6			8				10			
				5			7			9				11		

Приёмник:

Входные коды:	1	2	4	3	5	8	1	10	11
Выходные данные:	a	b	ab	c	ba	bab	a	aa	aaa
Новые строки, добавленные в таблицу:	4	5	6	7	8	9	10	11	

Нижняя часть рисунка, описывающего процесс кодирования иллюстрирует процесс распаковки. Каждый встреченный код преобразуется в соответствующую символьную строку. Между тем, выходной поток используется для создания новых элементов словаря по тем же правилам, что и в алгоритме сжатия.

Такое представление также предполагает структуру словаря, состоящую из нескольких деревьев, как показано на рисунке.



Еще один пример:

В нем мы ограничимся простым алфавитом — только заглавные буквы, без знаков препинания и пробелов. Сообщение, которое нужно сжать, выглядит следующим образом:

```
TOBEORNOTTOBEORTOBEORNOT#
```

Маркер **#** используется для обозначения конца сообщения. Тем самым, в нашем алфавите 27 символов (26 заглавных букв и #). Компьютер представляет это в виде групп бит, для представления каждого символа алфавита нам достаточно группы из 5-ти бит на символ. По мере роста словаря, размер групп должен расти, с тем чтобы учесть новые элементы. 5-битные группы дают $2^5 = 32$ возможных комбинации бит, поэтому, когда в словаре появится 33-е слово, алгоритм должен перейти к 6-битным группам. Поскольку используется группа из всех нолей 00000, то 33-я группа имеет код **32**. Начальный словарь будет выглядеть так:

```
# = 00000
A = 00001
B = 00010
C = 00011
.
.
.
Z = 11010
```

Без использования алгоритма LZW, при передаче сообщения как оно есть — 25 символов по 5 бит на каждый — оно займёт 125 бит. Сравним это с тем, что получается при использовании LZW:

```
Символ:  Битовый код:  Новая запись словаря:
         (на выходе)

T      20 =  10100
O      15 =  01111     28: TO
B       2 =  00010     29: OB
E       5 =  00101     30: BE
O      15 =  01111     31: EO
R      18 = 010010     32: OR <--- начинаем использовать 6-битные группы
N      14 = 001110     33: RN
O      15 = 001111     34: NO
T      20 = 010100     35: OT
TO     28 = 011100     36: TT
BE     30 = 011110     37: TOB
OR     32 = 100000     38: BEO
TOB    37 = 100101     39: ORT
EO     31 = 011111     40: TOBE
RN     33 = 100001     41: EOR
OT     35 = 100011     42: RNO
#       0 = 000000     43: OT#

Общая длина = 5*5 + 12*6 = 97 бит.
```

Таким образом, используя LZW мы сократили сообщение на 28 бит из 125 — это почти 22%. Если сообщение будет длиннее, то элементы словаря будут представлять всё более и более длинные части текста, благодаря чему повторяющиеся слова будут представлены очень компактно.

Мы получили закодированное сообщение, приведённое выше, и нам нужно его декодировать. Прежде всего, нам нужно знать начальный словарь, а последующие записи словаря мы можем реконструировать уже на ходу, поскольку они являются просто конкатенацией [1] предыдущих записей.

```
Данные:     На выходе:     Новая запись:
                          Полная:      Частичная:
10100  = 20     T                      28: T?
01111  = 15     O          28: TO      29: O?
00010  = 2      B          29: OB      30: B?
00101  = 5      E          30: BE      31: E?
01111  = 15     O          31: EO      32: O?  <--- начинаем использовать 6-битные группы
010010 = 18     R          32: OR      33: R?
001110 = 14     N          33: RN      34: N?
001111 = 15     O          34: NO      35: O?
010100 = 20     T          35: OT      36: T?
011100 = 28     TO         36: TT      37: TO? <--- для 36, добавляем только первый элемент
011110 = 30     BE         37: TOB     38: BE?      следующего слова словаря
100000 = 32     OR         38: BEO     39: OR?
100101 = 37     TOB        39: ORT     40: TOB?
011111 = 31     EO         40: TOBE    41: EO?
100001 = 33     RN         41: EOR     42: RN?
100011 = 35     OT         42: RNO     43: OT?
000000 = 0      #
```

Единственная небольшая трудность может возникнуть, если новое слово словаря пересылается немедленно. В приведённом выше примере декодирования, когда декодер встречает первый символ, **T**, он знает, что слово 28 начинается с T, но чем оно заканчивается? Проиллюстрируем проблему следующим примером.

```
Данные:     На выходе:     Новая запись:
                          Полная:      Частичная:
.
.
.
011101 = 29     AB         46: (word)   47: AB?
101111 = 47     AB?  <--- что нам с этим делать?
```

Мы декодируем сообщение **ABABA:**

На первый взгляд, для декодера это неразрешимая ситуация. Мы знаем наперёд, что словом 47 должно быть **ABA**, но как декодер узнает об этом? Заметим, что слово 47 состоит из слова 29 плюс символ идущий следующим. Таким образом, слово 47 заканчивается на «символ идущий следующим». Но, поскольку это слово посылается немедленно, то оно должно начинаться с «символа идущего следующим», и поэтому оно заканчивается тем же символом что и начинается, в данном случае — **A**. Этот трюк позволяет декодеру определить, что слово 47 это **ABA**.

В общем случае, такая ситуация появляется, когда кодируется последовательность вида *cScSc*, где *c* — это один символ, а *S* — строка, причём слово *cS* уже есть в словаре.

На момент своего появления алгоритм LZW давал лучший коэффициент сжатия, для большинства приложений, чем любой другой хорошо известный метод того времени. Он стал первым широко используемым на компьютерах методом сжатия данных.

[1] *Конкатенация - операция соединения (склеивания) нескольких строк символов в одну.*

Алгоритм был реализован в программе compress, которая стала более-менее стандартной утилитой Unix-систем приблизительно в 1986 году. Несколько других популярных утилит-архиваторов также используют этот метод или близкие к нему.

В 1987 году алгоритм стал частью стандарта на формат изображений GIF. Он также может (опционально) использоваться в формате TIFF.

В настоящее время, реализация алгоритма содержится в программе Adobe Acrobat.

Сжатие с потерями

Сжатие данных без потерь накладывает жесткую нижнюю границу на размер любого файла или сообщения. Этой границей является энтропия этого файла. Сжатие данных без потерь позволяет восстановить исходные несжатые данные с точностью до бита. Эта нужно для текстовых файлов, баз данных, двоичных файлов и т. д. Однако существует много типов файлов (данных), для которых не требуется абсолютно точного восстановления исходных данных. Например, аудио, видео, картинки. В этих случаях допускается некоторое количество ошибок при восстановлении исходных данных, и может применяться сжатие с потерями.

Ключевым вопросом сжатия с потерями является компромисс между коэффициентом сжатия и точностью восстановленных данных. Чем выше коэффициент сжатия для данного алгоритма сжатия, тем ниже точность восстановленных данных. Цель любого алгоритма сжатия с потерями заключается в достижении высоких коэффициентов сжатия за счет потери некоторых битов таким образом, чтобы восстановленные данные были достаточно близки к оригиналу.

Одномерное дискретное косинусное преобразование

Дискретное косинусное преобразование представляет собой преобразование массива пикселов в массив значений пространственной частоты. Это преобразование обратимо с точностью до ошибок округления. Дискретное косинусное преобразование не производит сжатия, но преобразует информацию об изображении в форму, более удобную для сжатия другими методами.

Предположим, что у нас есть одномерное изображение, представляющее собой линейный массив из N пикселов, каждый из которых обозначается некоторым значением яркости *p(x)* (0 < *x* < *N).* Таким образом, *p(x)* представляет собой пространственную функцию (а не функцию времени). Это изображение можно представить в виде суммы пространственно-частотных компонентов с частотами, изменяющимися

от 0 до *N*-1:

$$p(x) = \sqrt{\frac{2}{N}} \sum_{f=0}^{N-1} C(f) S(f) \cos\left[\frac{(2x+1)\pi f}{2N}\right] = \qquad (1) \qquad C(f) = \begin{cases} \frac{1}{\sqrt{2}}, f = 0, \\ 1, f > 0. \end{cases}$$

$$= \frac{S(0)}{\sqrt{N}} + \sqrt{\frac{2}{N}} \sum_{f=1}^{N-1} S(f) \cos\left[\frac{(2x+1)\pi f}{2N}\right]$$

Это напоминает представление непрерывной функции в виде ряда Фурье. В данном случае мы представляем дискретную функцию *p(x),* а частотные компоненты определены только для дискретных значений частот.

Чтобы получить формулу (1), нужно вычислить коэффициенты *{S(f), 0<f<N}.* Обратите внимание на то, что первое слагаемое формулы (1) представляет собой компонент с нулевой частотой, то есть постоянную составляющую (в нашем случае, постоянную составляющую яркости). Это слагаемое должно быть равно среднему значению *p(x).* Поэтому:

$$\frac{S(0)}{\sqrt{N}} = \frac{1}{N} \sum_{x-0}^{N-1} p(x)$$

$$S(0) = \frac{1}{\sqrt{N}} \sum_{x-0}^{N-1} p(x)$$

Общая формула для *S(f)* выглядит следующим образом:

$$S(f) = \sqrt{\frac{2}{N}} C(f) \sum_{x=0}^{N-1} p(x) \cos\left[\frac{(2x+1)\pi f}{2N}\right] \qquad (2)$$

Формула (2) называется одномерным дискретным косинусным преобразованием функции *p(x),* а формула (1) представляет собой обратное дискретное косинусное преобразование функции *S(f)*.

Рассмотрим пример, в котором блок из восьми пикселов одноцветного изображения подвергается дискретному косинусному преобразованию. Как правило (во многих стандартах сжатия), двумерное дискретное косинусное преобразование применяется к блоку размером 8x8 пикселов, так что данный пример представляет собой вполне реалистичную версию (только одномерную). В примере для обозначения яркости пикселов используются 8-битовые значения, так что 0 обозначает белый цвет, а 255 — черный. Это также типично для одноцветных изображений.

Исхоодное изображение

Вектор значений яркости

43	**138**	**148**	**183**	**208**	**254**	**248**	**148**

Яркость без постоянного уровня

-85	**10**	**20**	**55**	**80**	**126**	**120**	**20**
p(0)	p(1)	p(2)	p(3)	p(4)	p(5)	p(6)	p(7)

ДКП

122	**-129**	**-95**	**26**	**-73**	**4**	**-31**	**-11**
S(0)	S(1)	S(2)	S(3)	S(4)	S(5)	S(6)	S(7)

Обратное ДКП

-85	**10**	**20**	**55**	**80**	**126**	**120**	**20**

На рисунке показано исходное изображение, ниже — вектор значений яркости для этого изображения. Для удобства вычтем из каждого изображения 128 — такая операция называется *смещением уровня* — так, чтобы диапазон всех возможных значений яркости был симметричен относительно 0 (от -128 до 127), в результате получим третий рисунок. Затем производится дискретное косинусное преобразование:

$$S(f) = \frac{1}{2} C(f) \sum_{x=0}^{7} p(x) \cos\left[\frac{(2x+1)\pi f}{16}\right]$$

Эти вычисления соответствуют значениям, показанным на четвертом рисунке. Чтобы проверить полученные значения, сосчитаем обратное дискретное косинусное преобразование:

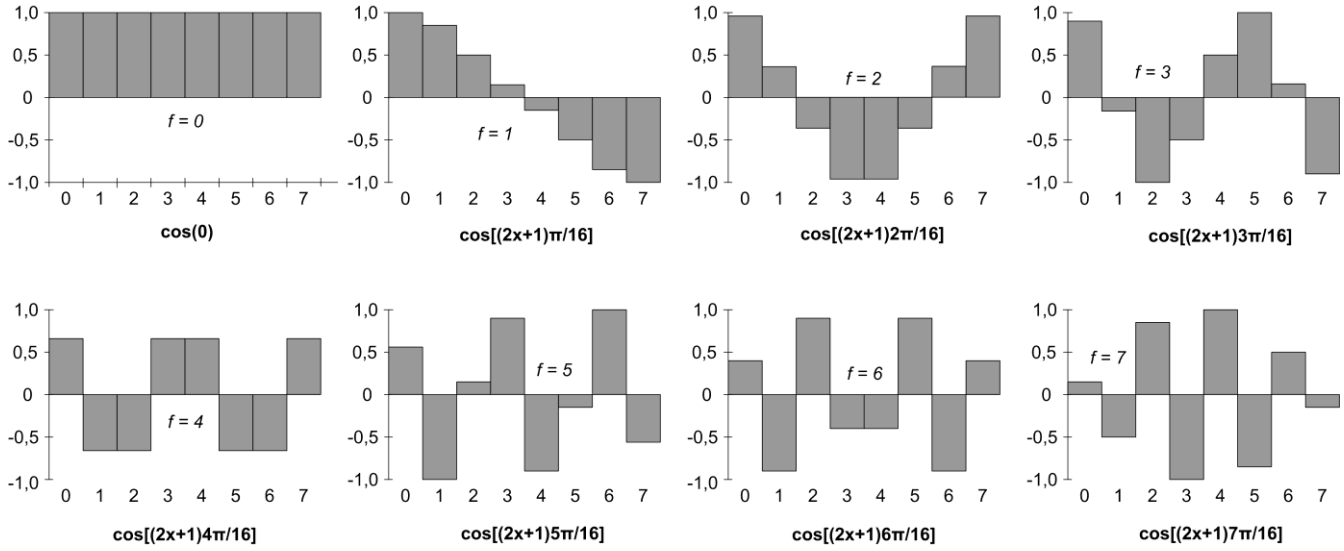
$$p(x) = \frac{1}{2} \sum_{x=0}^{7} C(f) S(f) \cos\left[\frac{(2x+1)\pi f}{16}\right]$$

В результате мы получим значения, показанные на рисунке внизу, которые точно совпадают с исходными значениями.

Стоит показать предыдущую формулу в развернутом виде, чтобы подчеркнуть природу дискретного косинусного преобразования:

$$p(x) = \frac{1}{\sqrt{8}} S(0)\cos[0] + \frac{S(1)}{2}\cos\left[\frac{(2x+1)\pi}{16}\right] +$$
$$+ \frac{S(2)}{2}\cos\left[\frac{(2x+1)2\pi}{16}\right] + \frac{S(3)}{2}\cos\left[\frac{(2x+1)3\pi}{16}\right] +$$
$$+ \frac{S(4)}{2}\cos\left[\frac{(2x+1)4\pi}{16}\right] + \frac{S(5)}{2}\cos\left[\frac{(2x+1)5\pi}{16}\right] +$$
$$+ \frac{S(6)}{2}\cos\left[\frac{(2x+1)6\pi}{16}\right] + \frac{S(7)}{2}\cos\left[\frac{(2x+1)7\pi}{16}\right]$$

Таким образом, функция *p(x)* может быть представлена в виде взвешенной суммы восьми косинусных функций. На рисунке показаны эти восемь функций, называемые базисными косинусными функциями. На каждом графике изображена косинусная функция дискретной пространственной переменной *x*. Каждая из этих дискретных функций представляет собой обычную косинусную функцию, дискретизированную по восьми отсчетам.



Этим функциям присущи несколько важных свойств.

- *Завершенность.* Взвешенная сумма этих восьми функций может быть вычислена для каждого из восьми значений пикселов.
- *Минимальность.* Ни одна из восьми волновых форм не может быть представлена взвешенной комбинацией других волновых функций, и для завершенности требуются все восемь волновых форм.
- *Уникальность.* Никакой другой набор косинусных функций, кроме масштабированных версий этих восьми волновых форм, не может использоваться для представления всех возможных последовательностей восьмизначений пикселов.

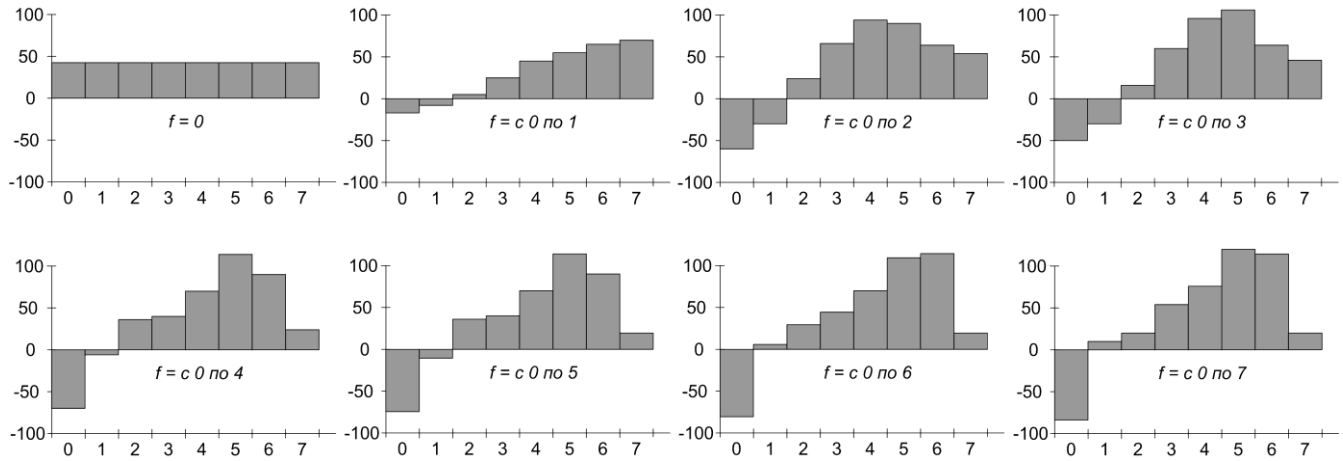
Эти свойства также применимы к общему случаю использования *N* косинусных функций для представления одномерного изображения из N пикселов.

Таким образом, любая дискретная функция $\{p(x), 0 \le x < N\}$ может быть представлена при помощи набора дискретных косинусных функций $\left\{\cos\left(\frac{(2x+1)\cdot \pi f}{2N}\right), 0 \le f < N\right\}$.

Обратите внимание на то, что в нашем примере большая часть амплитуд дискретного косинусного преобразования сконцентрирована на низких частотах, включая нулевую частоту (f=0). Это типично для изображений. Как правило, яркость изменяется постепенно и резкие переходы

встречаются нечасто. Соответственно, высокие пространственные частоты вносят в изображение незначительный вклад.

Важность низкочастотных составляющих иллюстрирует рисунок. На нем показана последовательность частичных сумм для нашего примера. На каждом графике к сумме добавляется еще одно слагаемое. На первом графике – всего одна составляющая с нулевой частотой, а на последнем – сумма всех восьми компонентов взвешенных частот. Уже после сложения трех или четырех составляющих общая форма конечной функции становится очевидной, а последние несколько слагаемых мало что меняют в общей картине. График в правом нижнем углу полностью представляет функцию p(x). Сама функция – внизу рисунка.



-85	10	20	55	80	126	120	20
p(0)	p(1)	p(2)	p(3)	p(4)	p(5)	p(6)	p(7)

Важность частотного представления изображения, выполненного при помощи ДКП заключается в следующем. Если вклад ВЧ компонент невелик, как это обычно и бывает, тогда можно удалить эти составляющие или представить их с меньшей точностью (меньшим количеством битов).

Двумерное дискретное косинусное преобразование

Двумерное дискретное косинусное преобразование является ключевым для стандартов JPEG и MPEG.

Так же как любой линейный массив из *N* пикселов может быть представлен в виде взвешенной суммы *N* косинусных функций с различными частотами, и матрица из *N x N* пикселов может быть представлена взвешенной суммой *N x N* косинусных функций. Формула выглядит следующим образом:

$$p(x,y) = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} C(u)C(v)S(u,v)\cos\left[\frac{(2x+1)\pi u}{2N}\right]\cos\left[\frac{(2y+1)\pi v}{2N}\right] \qquad (3)$$

где

$$C(f) = \begin{cases} \frac{1}{\sqrt{2}}, f = 0, \\ 1, f > 0 \end{cases} \quad для\ \ f = u\ или\ v.$$

Как следует из формулы (3), двумерный массив пикселов может быть представлен произведениями пространственно-частотных компонентов в горизонтальном и вертикальном измерениях. Отдельные косинусные сомножители те же самые, что и в одномерном случае. Таким образом, для дискретного косинусного преобразования 8 x 8 используются те же косинусные функции, что и на рисунке выше. Как и при одномерном преобразовании, слагаемое с нулевой частотой должно быть равно среднему значению функции *p(x).* В данном случае это — *S(0, 0)/N.* Поэтому

$$\frac{S(0,0)}{N} = \frac{1}{N^2}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} p(x,y),\ S(0,0) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} p(x,y)$$

Общая формула для S(f) выглядит следующим образом:

$$S(u,v) = \frac{2}{N}C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} p(x,y)\cos\left[\frac{(2x+1)\pi u}{2N}\right]\cos\left[\frac{(2y+1)\pi v}{2N}\right] \quad (4)$$

Формула (4) называется двумерным дискретным косинусным преобразованием функции *p(x, y).*

Мы рассмотрим пример в котором квадратный массив из 64 пикселов (8 x 8) одноцветного изображения подвергается дискретному косинусному преобразованию. В стандарте JPEG изображения обрабатываются блоками по 8x8. Этот размер был выбран по двум причинам. Во-первых, сложность вычислений быстро растет с увеличением размера обрабатываемого блока, поэтому блоки больших размеров оказывают чрезмерное давление на вычислительные ресурсы. Во-вторых, исследования различных изображений показали, что ограничения обрабатываемой области этими размерами не приводят к существенным потерям точности.

79	75	79	82	82	86	94	94
76	78	76	82	83	86	85	94
72	75	67	78	80	78	74	82
74	76	75	75	86	80	81	79
73	70	75	67	78	78	79	85
69	63	68	69	75	78	82	80
76	76	71	71	67	79	80	83
72	77	78	69	75	75	78	78

а.

619	-29	8	2	1	-3	0	1
22	-6	-4	0	7	0	-2	-3
11	0	5	-4	-3	4	0	-3
2	-10	5	0	0	7	3	2
6	2	-1	-1	-3	0	0	8
1	2	1	2	0	2	-2	-2
-8	-2	-4	1	2	1	-1	1
-3	1	5	-2	1	-1	1	-3

б.

На рисунке «а» показана матрица значений яркости после смещения уровня в диапазон от -128 до 127. Обратите внимание на то, что значения пикселов изменяются незначительно. Это типично для больших изображений: в любом блоке 8x8, скорее всего, изменения яркости будут небольшими. Затем производится преобразование:

$$S(u,v) = \frac{C(u)C(v)}{4}\sum_{x=0}^{7}\sum_{y=0}^{7} p(x,y)\cos\left[\frac{(2x+1)\pi u}{16}\right]\cos\left[\frac{(2y+1)\pi v}{16}\right] \quad (5)$$

В результате данных вычислений получается матрица, показанная на рисунке «б». Верхний левый элемент матрицы представляет собой значение S(0, 0), соответствующее среднему значению матрицы, умноженное на 8:

$$S(0,0) = \frac{1}{8}\sum_{x=0}^{7}\sum_{y=0}^{7} p(x,y) = 8\cdot\frac{\sum_{x=0}^{7}\sum_{y=0}^{7} p(x,y)}{64}$$

Значение этого элемента значительно превосходит значения всех остальных элементов, которые уменьшаются с ростом частоты (с удалением от левого верхнего элемента матрицы). Это типично для многих изображений. Другими словами, можно сказать, что большая часть информации в типичном

изображении находится в области низкочастотных составляющих. Другими словами, как правило, у изображений нет резких изменений яркости.

Для проверки этих значений можно осуществить обратное дискретное косинусное преобразование:

$$p(x,y)=\frac{1}{4}\sum_{x=0}^{7}\sum_{y=0}^{7}C(u)C(v)S(u,v)\cos\left[\frac{(2x+1)\pi u}{16}\right]\cos\left[\frac{(2y+1)\pi v}{16}\right] \qquad (6)$$

В результате подобных преобразований мы получим исходные входные данные, показанные на рис. *а.*

Вэйвлет – сжатие

В последние годы значительный интерес получило использование метода волнового сжатия (wavelet compression) изображений. Двумя заслуживающими внимания примерами использования данного метода являются система идентификации отпечатков пальцев ФБР и последняя версия широко применяемого стандарта JPEG —JPEG 2000. Привлекательность метода волнового сжатия заключается во впечатляющих коэффициентах сжатия при высоких скоростях. Метод волнового сжатия также обладает высокой гибкостью. Например, он позволяет представлять различные области изображения с различными степенями разрешения и точности. Кроме того, метод волнового сжатия хорошо подходит для прогрессивной передачи изображения, при которой сначала передается нечеткая версия изображения, а более мелкие детали пересылаются в ходе передачи позднее.

Хороший способ познакомиться с понятием *элементарной волны* (wavelet) — сравнение с представлением функции в виде ряда Фурье. Любая периодическая функция одной переменной *g(x)* может быть представлена в виде ряда Фурье:

$$g(x)=\frac{A_0}{2}+\sum_{n=1}^{\infty}\left[A_n\cos(2\pi nf_0x)+B_n\sin(2\pi nf_0x)\right]$$

Здесь f_0 представляет собой величину, обратную периоду функции *(f_0 = 1/T).* Частота f_0 называется *основной частотой* (fundamental frequency) или *основной гармоникой* (fundamental harmonic). Кратные f_0 частоты называют *гармониками* (harmonics). Таким образом, периодическая функция с периодом *T* может быть представлена как сумма синусоид с частотами, кратными частоте $f_0 = 1/T$. Как правило, разложение в ряд Фурье применяется к функции времени (то есть аргумент *x* соответствует времени), но эта операция также применима и к пространственной функции, что уместно при обработке изображений. По существу, ряды Фурье показывают, что любая периодическая функция может быть полностью описана суммой синусоид с частотами, кратными основной частоте. Говорят, что ряды Фурье представляют функцию в *области частот,* тогда как оригинальная функция определена во *временной области* или в *пространстве* в зависимости от того, относится аргумент *x* ко времени или к пространству.

Используемые при разложении в ряд Фурье коэффициенты вычисляются по следующим формулам:

$$A_0=\frac{2}{T}\int_0^T g(x)dx,\quad A_n=\frac{2}{T}\int_0^T g(x)\cos(2\pi nf_0x)dx,\quad B_n=\frac{2}{T}\int_0^T g(x)\sin(2\pi nf_0x)dx.$$

Таким образом, по представлению функции в виде ряда Фурье легко восстановить ее временную или пространственную форму. Непериодическая функция тоже может быть представлена в области частот при помощи преобразования Фурье. В этом случае вместо суммы дискретных гармоник функция представляется в непрерывном диапазоне частот в виде частотного спектра. Однако основные принципы здесь те же. Преобразование Фурье также обратимо, то есть можно по частотному спектру функции получить ее временное или пространственное представление.

Понятие представления Фурье можно распространить на двумерные функции и в таком виде использовать для обработки изображений.

Анализ Фурье представляет собой мощный инструмент, позволяющий решать многие проблемы, которые очень сложно решать другими методами. Коэффициенты Фурье можно вычислять, хранить, передавать и использовать для восстановления исходного сигнала или функции. При решении многих проблем значительно проще оперировать коэффициентами Фурье (в области частот), чем исходной функцией (во временной или пространственной области).

Однако анализ Фурье плохо подходит для работы с функциями, определенными на коротком интервале аргумента (в отличие от функций, заданных в диапазоне ($-\infty \le x \le \infty$), а также с функциями, значение которых

резко и значительно изменяется. Такими характеристиками обладают изображения, имеющие ограниченную протяженность в пространстве и, как правило, содержащие резкие края или области с резко изменяющимися характеристиками. Для обработки таких изображений, а также для многих других целей анализ элементарных волн подходит лучше, чем анализ Фурье.

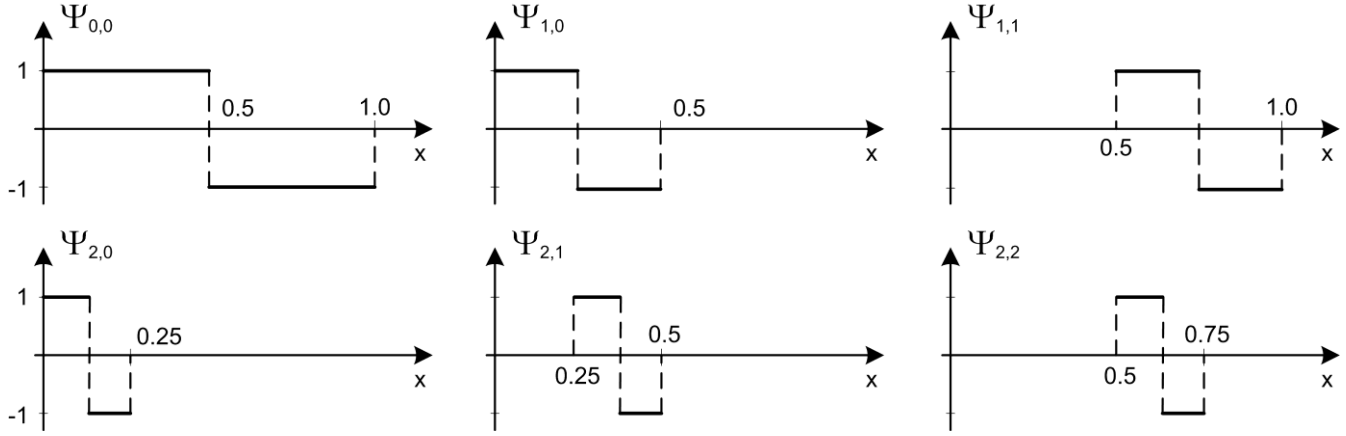
Как представление Фурье, волновое представление включает суммирование нескольких элементарных функций. В анализе Фурье в качестве элементарной функции используется синусоида, заданная на всей вещественной оси. В волновом анализе элементарной функцией является элементарная волна, представляющая собой функцию, отличную от нуля только на конечном отрезке значений аргумента и равную нулю для всех остальных значений аргумента. Функция представляется в виде суммы элементарных волн, каждая из которых, в свою очередь, представляет собой растянутую или суженную единичную элементарную волну, называемую *материнской элементарной волной* (mother wavelet). Путем анализа элементарных волн может быть осуществлено волновое преобразование сигнала, протяженного во времени, или изображения, в результате которого можно получить соответствующие коэффициенты. Как и в случае анализа Фурье, эти коэффициенты позволяют упростить обработку сигнала или изображения. Простейшая элементарная волна, применяемая в волновом анализе, называется элементарной волной Хаара (Haar wavelet) и определяется следующим образом:

$$\Psi_{0,0}(x)=\begin{cases}1, & 0\le x<\dfrac{1}{2}\\ -1, & \dfrac{1}{2}\le x<1\\ 0, & \textit{в остальных случаях}\end{cases}$$

Из этой базовой элементарной волны мы можем получить следующий набор функций:

$$\Psi_{j,k}(x)=\Psi_{0,0}(2^j x-k)=\begin{cases}1, & k2^{-j}\le x<\left(k+\dfrac{1}{2}\right)2^{-j}\\ -1, & \left(k+\dfrac{1}{2}\right)2^{-j}\le x<(k+1)2^{-j}\\ 0, & \textit{в остальных областях}\end{cases}\qquad \Psi$$

На рисунке показаны некоторые из этих элементарных волн Хаара. Обратите внимание на то, что с увеличением параметра *j* элементарная волна становится более короткой, то есть частота элементарной волны растет. Параметр *k* сдвигает элементарную волну по оси *x* (во времени или в пространстве).



Волновое представление позволяет использовать так называемый *анализ с переменным разрешением*. Сжатые версии базовой элементарной волны, сдвинутые в соответствующие области сигнала или изображения, могут представлять высокочастотные составляющие. Несжатые версии базовой элементарной волны соответствуют низкочастотным, медленно изменяющимся

компонентам. Добавляя все более сжатые элементарные волны, можно представлять сигнал или изображение с произвольной степенью разрешения или точности. Более того, подобное представление с переменным разрешением может найти очевидное применение для сжатия и прогрессивного преобразования:

- Изображение можно сжать, отбросив коэффициенты с относительно не большими значениями. В результате будут отброшены элементарные волны, вклад которых в формирование изображения невелик.
- Передачу изображения по сети можно начинать с низкочастотных компонентов. По мере приема более высокочастотных компонентов к изображению будет добавляться все больше деталей, а разрешение изображения будет увеличиваться, то есть изображение будет становиться четче.

Одномерное сжатие с помощью элементарных волн Хаара

Чтобы понять как можно использовать элементарные волны Хаара для сжатия изображения в одном направлении, воспользуемся примером:

Рассмотрим строку из восьми значений данных, которые могут быть рядом из матрицы 8x8 пикселов, значение каждого из которых соответствует яркости. Их значения – в верхней строке таблицы.

64	**48**	**16**	**32**	**56**	**56**	**48**	**24**
56	24	56	36	8	-8	0	12
40	46	16	10	8	-8	0	12
43	-3	16	10	8	-8	0	12

Мы будем преобразовывать изображение в три этапа, используя процесс, называемый усреднением и дифференцированием.

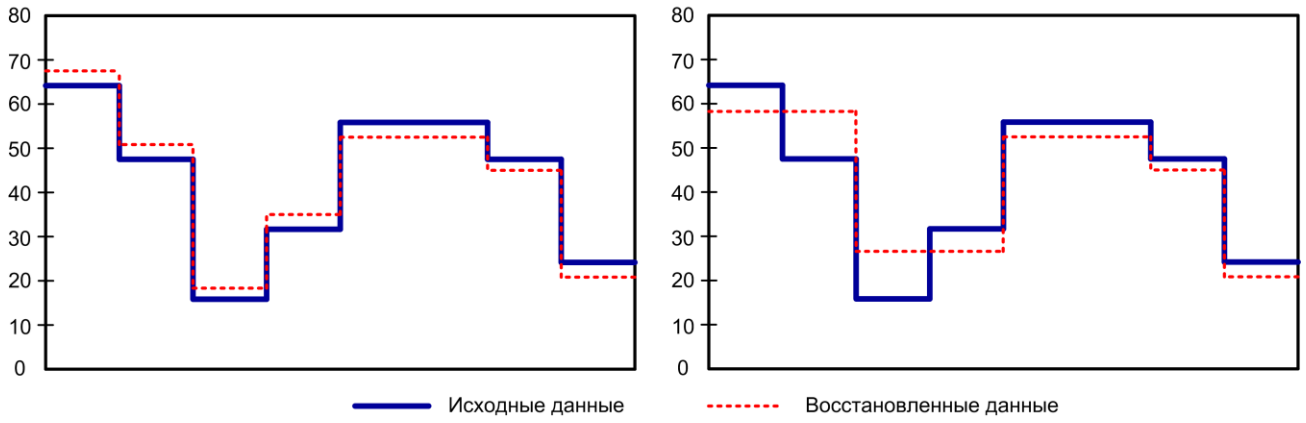
На первом шаге вычисляются средние значения последовательных пар чисел из исходной строки, в результате чего получаются первые четыре числа второй строки. Остальные четыре значения второй строки вычисляются как разности тех же пар чисел первого ряда. Таким образом, последние четыре числа второго ряда представляют собой уровень более мелкой детализации, отражающий изменения изображения. Первые четыре числа отражают фоновый уровень, к которому для воспроизведения оригинального изображения необходимо добавить более мелкие детали. На следующем шаге обрабатываются только первые четыре числа, которые снова попарно усредняются и дифференцируются. В результате мы получаем два числа, представляющие уровень меньшей степени детализации. На последнем шаге усредняются и дифференцируются два первых значения предыдущего ряда. Все разностные значения показаны полужирным шрифтом. Обратите внимание на то, что первое значение последнего ряда представляет собой среднее значение всех восьми чисел исходного изображения. Мы можем сделать несколько наблюдений:

- Этот процесс обратим. Путем соответствующих сложений и вычитаний мы можем снова получить из последнего ряда исходный ряд.
- Этот процесс можно обобщить применительно к строкам любой длины. Для троки из 2^k элементов требуются k этапов обработки. Любая строка, длина которой не равна степени 2, может быть дополнена нулями.
- Области, в которых исходные данные изменяются незначительно, после преобразования выглядят как последовательности небольших или нулевых значений.
- Преобразованное представление позволяет осуществить сжатие без потерь, так как обычно преобразованные данные содержат нулевые значения. В нашем примере седьмой элемент в преобразованном представлении равен нулю. Путем группового кодирования можно получить сжатую версию.
- Значительно большей степени сжатия, хотя и с потерями данных, можно добиться, игнорируя области малых изменений.

Чтобы продемонстрировать последнее утверждение, зададим некоторое пороговое значение ε и обнулим все элементы преобразованной строки, модуль которых не превышает ε. Например, при пороговом значении 4 из строки удаляется второй элемент, в результате чего получится строка (43, 0,16, 10, 8, -8, 0, 12). Поместим эту строку в последний ряд пустой таблицы и вычислим по ней исходные данные:

67	51	19	35	53	53	45	21
59	27	53	33	8	-8	0	12
43	43	16	10	8	-8	0	12
43	0	16	10	8	-8	0	12

59	59	27	27	53	53	45	21
59	27	53	33	0	0	0	12
43	43	16	10	0	0	0	12
43	0	16	10	0	0	0	12



На рисунке слева полученный результат сравнивается с исходными данными. Соответствие оказывается довольно близким. На черно-белом дисплее эти два изображения были бы практически идентичными. Теперь установим пороговое значение ε = 8. При этом мы устраним еще два элемента в сжатой строке и получим строку (43,0,16,10, 0,0,0,12). Распакуем эту строку: (59,59, 27, 27, 53, 53,45, 21). Этот результат сравнивается с исходными данными на рисунке справа. Данное приближение основывается только на пяти из исходных восьми значений, но приближение остается довольно неплохим.

Представление в виде матрицы

Для вычислений мы можем преобразовать только что описанные операции в простые конструкции линейной алгебры. Обратите внимание на то, что первый элемент в преобразованной строке представляет собой просто среднее значение всех точек исходных данных, то есть сумму всех значений, деленную на 8. Второй элемент представляет собой разность средних значений первых и последних четырех точек и т. д. Таким образом, мы можем представить данное преобразование в виде следующего произведения вектора и матрицы:

$$\begin{pmatrix}64 & 48 & 16 & 32 & 56 & 56 & 48 & 24\end{pmatrix}\times\frac{1}{8}\times\begin{pmatrix}1 & 1 & 2 & 0 & 4 & 0 & 0 & 0\\ 1 & 1 & 2 & 0 & -4 & 0 & 0 & 0\\ 1 & 1 & -2 & 0 & 0 & 4 & 0 & 0\\ 1 & 1 & -2 & 0 & 0 & -4 & 0 & 0\\ 1 & -1 & 0 & 2 & 0 & 0 & 4 & 0\\ 1 & -1 & 0 & 2 & 0 & 0 & -4 & 0\\ 1 & -1 & 0 & -2 & 0 & 0 & 0 & 4\\ 1 & -1 & 0 & -2 & 0 & 0 & 0 & -4\end{pmatrix}=$$

$$=\begin{pmatrix}43 & -3 & 16 & 10 & 8 & -8 & 0 & 12\end{pmatrix}$$

Определим два последних сомножителя в предыдущей формуле как матрицу W. Мы можем восстановить исходные данные из преобразованных данных с помощью обратной матрицы:

$$W^{-1} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

Убедиться в том, что матрица W^{-1} является обратной по отношению к W, можно, перемножив эти матрицы:

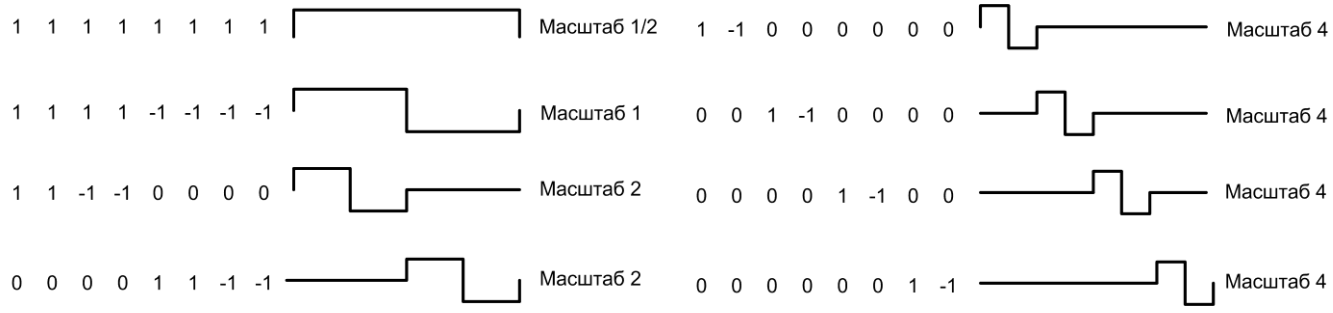
С помощью обратной матрицы W^{-1} мы можем восстановить исходные данные из преобразованных данных:

$$\begin{pmatrix} 43 & -3 & 16 & 10 & 8 & -8 & 0 & 12 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} =$$

$$= \begin{pmatrix} 64 & 48 & 16 & 32 & 56 & 56 & 48 & 24 \end{pmatrix}$$

Представление в виде элементарных волн Хаара

Предыдущая формула предлагает способ выражения преобразования в виде суммы элементарных волн Хаара. Вспомним, что нам необходимо сформировать семейство элементарных волн, сжимая базовую элементарную волну в разное число раз и смещая ее по оси абсцисс. Таким образом, мы получим элементарную волну полного размера, две элементарные волны половинного размера, четыре элементарные волны размера 1/4 и т. д. С помощью подобных уменьшенных и сдвинутых элементарных волн мы можем представить ряды обратной матрицы W^{-1}, как показано на рисунке ниже.



Прежде чем продолжить, определим новую функцию, называемую масштабирующей функцией Хаара:

$$\varphi(x) = \begin{cases} 1, \textit{если } 0 \le x < 1 \\ 0 \quad \textit{в противном случае} \end{cases}$$

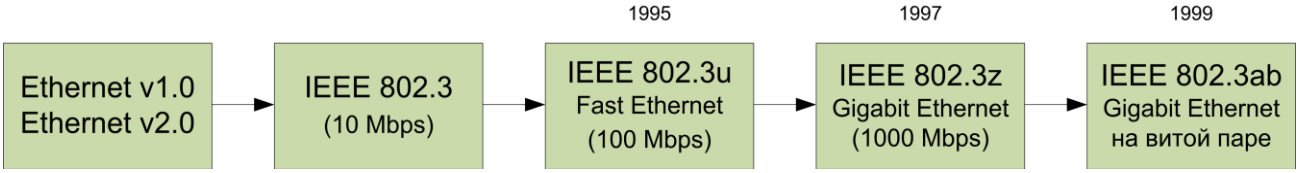
Теперь несложно выразить исходную строку данных в виде суммы элементарных волн Хаара и масштабирующей функции Хаара:

$$43\varphi(x) - 3\Psi_{0,0} + 16\Psi_{1,0} + 10\Psi_{1,1} + 8\Psi_{2,0} - 8\Psi_{2,1} + 0\Psi_{2,2} + 12\Psi_{2,3}$$

Таким образом, мы можем интерпретировать строку как сумму общего среднего значения и семи элементарных волн, умноженных на соответствующие коэффициенты.

IEEE802.3

Ветераном сетевых технологий (архитектур) является Ethernet.



Первые версии - Ethernet vl.O и Ethernet v2.0 предназначались только для коаксиального кабеля. Сейчас под названием Ethernet подразумевают стандарт IEEE 802.3 (скорость 10 Мбит/с). В 1995 году был принят стандарт IEEE 802.3u - Fast Ethernet со скоростью 100 Мбит/с, а в 1997 году IEEE 802.3z - Gigabit Ethernet (1000 Мбит/с). Осенью 1999 года принят стандарт IEEE 802.3ab - Gigabit Ethernet на витой паре категории 5. Популярные разновидности Ethernet обозначаются как 10Base2, 100BaseTX и др. Здесь первый элемент обозначает скорость передачи, Мбит/с. Второй элемент: Base - прямая (немодулированная) передача, Broad - использование широкополосного кабеля с частотным уплотнением каналов. Третий элемент: округленная длина кабеля в сотнях метров (10Base2 - 185 м, 10Base5 - 500 м, хотя в 1Base5 длина до 250 м) или среда передачи (T, TX, T2, T4 - витые пары, FX, FL, FB, SX и LX - оптоволокно, CX - твинаксиальный кабель для Gigabit Ethernet).

Технология Ethernet базируется на методе множественного доступа к среде передачи с прослушиванием несущей и обнаружением коллизий - CSMA/CD.

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:

CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) — множественный доступ с прослушиванием несущей и избежанием коллизий. Узел, готовый послать кадр, прослушивает линию.

При отсутствии несущей он посылает короткий сигнал запроса па передачу (RTS) и определенное время ожидает ответа (CTS) от адресата назначения. При отсутствии ответа (подразумевается возможность коллизии) попытка передачи откладывается, при получении ответа в линию посылается кадр. Метод не позволяет полностью избежать коллизий, но они обрабатываются на вышестоящих уровнях протокола.

Метод применяется в сети Apple LocalTalk, характерен простотой и низкой стоимостью.

CSMA/CD (Carrier Sense Multiple Access/Collision Detect) — множественный доступ с прослушиванием несущей и обнаружением коллизий. Это — полудуплексная архитектура, что означает возможность передавать информацию в тот или иной момент времени только для одной станции. Архитектуру CSMA/CD можно сравнить с общением людей, участвующих в селекторном совещании:

- Ни один из участников не знает, когда состоится выступление другого человека.
- Участник, который хочет сделать сообщение, должен ждать у телефона, пока телефонная линия не освободится, только тогда он сможет говорить.
- Когда телефонная линия освобождается, одновременно могут начать говорить два или несколько участников.
- Если двое участников говорят одновременно, слушателям трудно их понять, поэтому они должны замолчать и подождать, пока линия освободится, прежде чем вновь начать говорить.

Алгоритм работы: Узел, готовый послать кадр, прослушивает линию. При отсутствии несущей он начинает передачу кадра, одновременно контролируя состояние линии. При обнаружении коллизии передача прекращается и повторная попытка откладывается на случайное время.

Коллизии — нормальное, хотя и не очень частое явление для CSMA/CD. Их частота связана с количеством и активностью подключенных узлов. Нормально коллизии могут начинаться в определенном временном окне кадра, запоздалые коллизии сигнализируют об аппаратных неполадках в кабеле или узлах. Метод эффективнее, чем метод с избежанием, но требует более сложных и дорогих схем цепей доступа.

Применяется во многих сетевых архитектурах: Ethernet, EtherTalk (реализация Ethernet фирмы Apple), G-Net, IBM PC Network, AT&T StarLAN.

Общий недостаток вероятностных методов доступа — неопределенное время прохождения кадра, резко возрастающее при увеличении нагрузки на сеть, что ограничивает его применение в системах реального времени

При **детерминированном** (deterministic) методе узлы получают доступ к среде в предопределенном порядке. Последовательность определяется контроллером сети, который может быть централизованным (его функции может выполнять, например, сервер) или/и распределенным (функции выполняются оборудованием всех узлов).

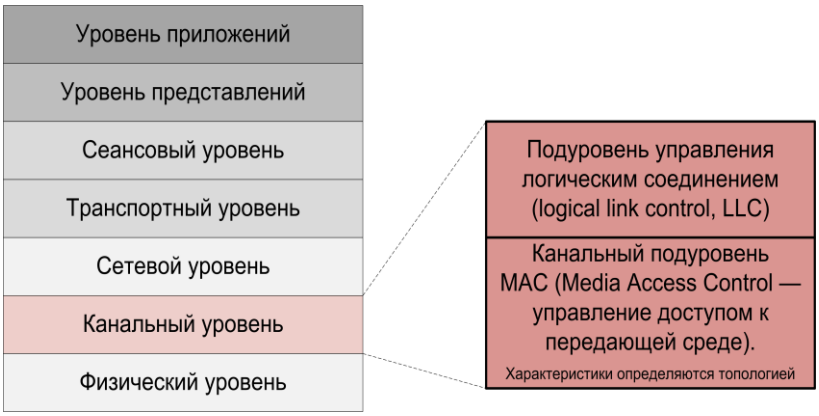
Основные типы:

доступ с передачей маркера (token passing), применяемый в сетях ARCnet, Token Ring, FDDI;

поллинг — опрос готовности, применяемый в больших машинах (mainframes) и технологии 100VG-AnyLAN, так же polling применяется в широкополосных беспроводных технологиях (WiMax). Основное преимущество метода — ограниченное время прохождения кадра, мало зависящее от нагрузки.

Ethernet 802.3 и модель OSI

Канальный уровень передачи данных имеет два подуровня.



Канальный подуровень (data link sublayer), также известный как уровень MAC (Media Access Control — управление доступом к передающей среде). Характеристики этого подуровня определяются топологией. Например, сети стандарта Token Ring 802.5 используют MAC, отличный от уровня MAC, используемого в сетях стандарта Ethernet 802.3

Подуровень управления логическим соединением (logical link control, LLC). Общий уровень для всех сетей, основанных на стандарте IEEE802. Предоставляет простой протокол передачи фреймов, который обеспечивает передачу фреймов без установления соединения. Механизм уведомления отправителя о том, что фрейм доставлен или не доставлен, отсутствует.

Фрейм (кадр) Ethernet состоит из следующих полей:

Преамбула. Представляет собой набор из семи октетов (октет содержит 8 битов), т.е. всего 56 бит, поочередно принимающих значение 0 и 1. Т.е., каждый октет представляет собой битовую комбинацию 10101010. Преамбула указывает станции-получателю, что передается фрейм.

Флаг начала фрейма (SFD). Представляет собой 8-битовое поле, содержащее битовую комбинацию, аналогичную заголовку, но оба последних разряда имеют значение 1 (10101011). Эта комбинация указывает станции-получателю, что вслед за данным полем будет передана содержательная часть фрейма.

MAC-адрес получателя. Поле адреса приемника имеет 48-разрядное значение, указывающее адрес станции-приемника, для которой предназначен фрейм.

Адрес отправителя. Поле адреса отправителя представляет собой 48-разрядное значение, указывающее адрес станции-отправителя.

Поле TLV – использует 16 разрядов, для того чтобы указать, какой тип протокола более высокого уровня инкапсулирован в поле данных или в поле содержимого пакета. Это поле также называют полем типа фрейма Ethernet; его значение указывает на режим работы Ethernet .

Содержимое или данные. Поле содержимого или данных содержит пакеты протокола более высокого уровня и должно иметь ширину не менее 46 байт и не более 1500 байт. Минимальное значение размера данных или содержимого обусловлено необходимостью предоставления шанса приема пакета всем станциям. Эту проблема рассмотрим далее. Если размер данных или содержимого менее 46 байт, передающая станция дополняет содержимое, чтобы размер поля составлял как минимум 46 байт.

Контрольная последовательность фрейма (FCS). Поле содержит значение циклического избыточного кода (CRC), вычисленное на основе битовой комбинации фрейма.

Адреса Ethernet (MAC) представляют собой 48-разрядные значения и отчасти назначаются в рамках глобальной системы идентификации (курируемой IEEE), отчасти — производителями оборудования. Организация назначает каждому поставщику 24-разрядный уникальный организационный идентификатор (OUI). Этот идентификатор включается в Ethernet-адрес в качестве первых 24-х разрядов.

Благодаря этому гарантируется уникальность Ethernet-адреса. Каждая станция может быть включена в любую сеть мира и быть однозначно идентифицирована.

Например, Ethernet-адрес маршрутизатора Cisco начинается на 00-03-6b, оставшиеся 24 бита назначаются устройству компанией Cisco.

Поле адреса назначения фрейма (DA - Destination Address) может содержать адрес одного из трех типов: Уникальный MAC-адрес единственного получателя кадра (unicast).

Широковещательный адрес (broadcast), указывающий на то, что данный кадр адресован всем увидевшим его абонентам сети.

Групповой адрес (multicast), являющийся признаком, по которому узлы могут обрабатывать интересующие их кадры. Тип адреса задается его первым байтом.

Уникальность адресации адаптеров, обеспечивается специальным соглашением, по которому каждому производителю аппаратуры выделяется свое значение (одно или несколько) – байты 2-3 (иногда к коду производителя относят и первый байт, имеющий нулевое значение).

Байты 4-6 заполняются изготовителем (эта информация может рассматриваться как серийный номер платы).

Случаются и конфузы, когда «подпольные» производители снабжают свои изделия одинаковыми адресами — больше одного такого устройства в.одной локальной сети работать не будет.

Ряд моделей адаптеров (в комплекте, с. драйверами) позволяет задавать MAC-адрес узла и произвольно, но в этом случае ответственность за уникальность адресации ложится на администратора. Признаком «ручного» задания адреса должна быть единица во втором справа разряде первого байта адреса (02-xx-xx-xx-xx-xx).

FF-FF-FF-FF-FF-FF — широковещательный адрес.

01-xx-xx-xx-xx-xx — групповой адрес. Идентификатором группы являются байты 2-6.

Широковещательные фреймы принимаются и обрабатываются всеми станциями домена. Станция, получающая "чужие" широковещательные фреймы, использует свой центральный процессор (ЦП) для их обработки, в то время как его должны были бы использовать для своих нужд другие ресурсы станции Процесс обработки таких фреймов может показаться простым делом, однако, как известно, широковещательная лавина может вызвать перегрузку в сети и подключенных к ней станций.

Многоадресатные (групповые) фреймы похожи на широковещательные в том смысле, что они позволяют отправителю направлять их сразу группе получателей, а не одному.

На многоадресатные фреймы должна быть "проведена подписка"; это означает, что станция-приемник должна изъявить желание получать их.

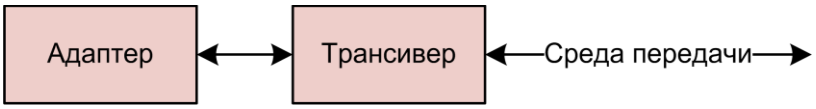
Многоадресатное IP-вещание - общепринятый механизм работы с содержимым потокового видео.

Одноадресатная рассылка представляет собой простейший и прямой способ передачи данных станции-получателю. Передающая станция направляет фрейм с адресом назначения, соответствующим Ethernet-адресу конкретной станции. Только эта приемная станция получает и обрабатывает фрейм и его содержимое.

Особенности реализации CSMA/CD в Ethernet

Каждый узел сети имеет сетевой адаптер — схему, реализующую метод CSMA/ CD на аппаратном (или микропрограммном) уровне.

Адаптер имеет приемопередатчик — трансивер, подключенный к общей (разделяемой) среде передачи.



Адаптер узла, нуждающийся в передаче информации, прослушивает линию и дожидается «тишины» — отсутствия сигнала (несущей). Далее он формирует кадр (фрейм), начинающийся с синхронизирующей преамбулы, за которой следует поток двоичных данных в самосинхронизирующемся (манчестерском) коде. Все остальные узлы принимают этот сигнал; синхронизируются по преамбуле и декодируют его в последовательность бит, помещаемую в свой приемный буфер. Окончание кадра определяется по пропаданию несущей, и по этому событию приемники анализируют принятый кадр.

Этот кадр контролируется на отсутствие ошибок (с помощью контрольной последовательности бит и по длине), после чего в «хорошем» кадре проверяется адресная информация.

В каждом кадре имеется заголовок с MAC-адресами узла-источника и узла его назначения. Если адрес назначения кадра соответствует MAC-адресу данного узла, то кадр поступает на дальнейшую обработку протоколами вышестоящих уровней. Кадры, не адресованные данному узлу, им игнорируются,на аппаратном уровне адаптера.

Теперь предположим, что два узла хотят передать данные почти одновременно: оба дождались «тишины» и стали передавать преамбулу.

Столкновение двух сигналов — коллизия— приведет к их искажению, которое обнаруживается передатчиком. Передающие узлы, обнаружив коллизию, прекращают передачу кадра, после чего повторную попытку передачи сделают через случайный интервал времени (каждый через свой) после

освобождения линии. Если повторная попытка также не удалась, делается следующая (и так до 16 раз), причем интервал увеличивается.

Приемник обнаруживает коллизию по ненормально короткой длине (в «хорошем» кадре она не может быть меньше 64 байт, не считая преамбулы) и такие кадры отбрасывает. Коллизии являются нормальным, хотя и нежелательным явлением. Метод CSMA/CD хорошо работает лишь при общей загрузке канала (среды передачи) до 30 %.

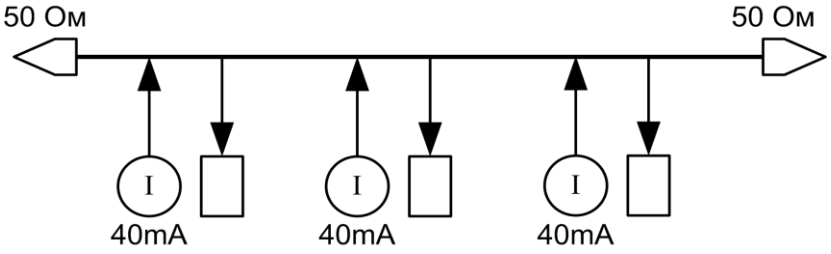
При большей загрузке коллизии приводят к прогрессирующей деградации производительности, что является слабым местом технологии Ethernet (любой технологии допускающей коллизии).

Несмотря на то, что в принципе Ethernet допускает наличие в одном сегменте сотен (даже тысяч) узлов, при их высокой активности разумный размер домена коллизий — группы узлов, связанных общей средой (кабелями и повторителями), — ограничен лишь несколькими десятками узлов.

Протяженность домена коллизий ограничивается временем распространения сигнала между самыми удаленными друг от друга узлами. В связи с различными способами физического кодирования сигнала в линии, временные соотношения удобно измерять в битовых интервалах bt (bit time).

Битовый интервал – время, необходимое для передачи одного бита, которое при скорости передачи 10 Мбит/с составляет 0,1 мкс. Смежные 8-битные труппы называют как байтами, так и октетами («связисты» более тяготеют к словосочетанию «октет», «компьютерщикам» привычнее - байт).

Двоичная информация передается в Манчестерском коде. В середине каждого битового интервала происходит изменение состояние в линии: от -V к +V для единичного бита, от +V к -V — для нулевого. В начале битового интервала изменение может быть, а может и не быть. Передатчик является источником тока 40 мА, приемник — детектором уровня напряжения с высоким входным сопротивлением. Узел, не передающий в данный момент, вносит нагрузку с сопротивлением более 100 кОм. Приемник и передатчик подключаются к общему коаксиальному кабелю с импедансом 50 Ом, который с обоих концов оканчивается 50-омными терминаторами. Т-образные Ответвления кабеля недопустимы: Два терминатора образуют нагрузку с сопротивлением постоянному току 25 Ом, с учетом сопротивления кабеля эта нагрузка может доходить и до 30 Ом (худший случай, когда узел расположен в середине самого длинного сегмента). На номинальной нагрузке ток 40 мА от одного передатчика вызывает падение напряжения 1 В. Коллизия определяется передающим трансивером по большому уровню (более 1,5 В) сигнала в линии, вызванному одновременной работой двух и более передатчиков.



Принятый метод доступа и способ передачи сигналов критичен к импедансу кабеля и терминаторов. При использовании иного, (например, 75 Ом) кабеля и нарушении терминации (не два терминатора или не 50 Ом) сеть практически неработоспособна — из-за большого сопротивления нагрузки сигнал одиночного передатчика будет превышать порог срабатывания детектора коллизий.

Порог срабатывания детектора коллизий (1,5-1,6 В) выбирается с таким расчетом, чтобы сигнал от одного передатчика гарантированно не приводил к срабатыванию детектора, а сумма сигналов от двух передатчиков вызывала срабатывание, причем для самых худших случаев.

Коллизии могут выявляться в двух режимах: в режиме передачи и в режиме приема. При выявлёнии коллизий в режиме передачи детектор обязан обнаружить коллизию двух (и более) передатчиков, один из которых - его собственный. Это более легкий (в плане тонкости подбора порогов) случай.

При выявлении коллизий в режиме приема (receive mode collision detection) детектор обязан обнаружить коллизии любых двух (и более) передатчиков, при этом «вилка» возможных значений порогов сужается.

Кадр начинается с преамбулы (preamble) длиной в 7 байт с кодами 10101010, за которой следует 1-байтный разделитель начала кадра SFD (Start Frame Delimiter) с кодом 10101011. За ним. следует 6-байтный адрес назначения, 6-байтный адрес источника, заголовок, поле данных и 4-байтное поле контрольного CRC- кода, с помощью которого контролируется целостность всего кадра. Заголовок и поле данных в разных типах кадров трактуются по-разному, но их суммарная длина не может быть меньше 48 байт и больше 1502 байт. Если требуется передать кадр с меньшим числом байт, после действительных

данных вводится заполнитель (Pad), доводящий размер кадра до минимально разрешенного. Таким образом, размер нормального кадра (включая адресную информацию и CRC-код) может быть в диапазоне 64—1518 байт. Адаптер приемника способен распознавать следующие ошибки кадров (конец кадра определяется по пропаданию несущей):

Длинный кадр (long, oversized) — более 1518 байт с правильным CRC-кодом. Может порождаться некорректным драйвером адаптера.

Короткий кадр (runt, undersized) — менее 64 байт с правильным CRC-кодом. Может порождаться некорректным драйвером адаптера. _

Болтливый» кадр (jabber) — более 1518 байт с неправильным CRC-кодом. Может порождаться неисправным трансивером (адаптером).

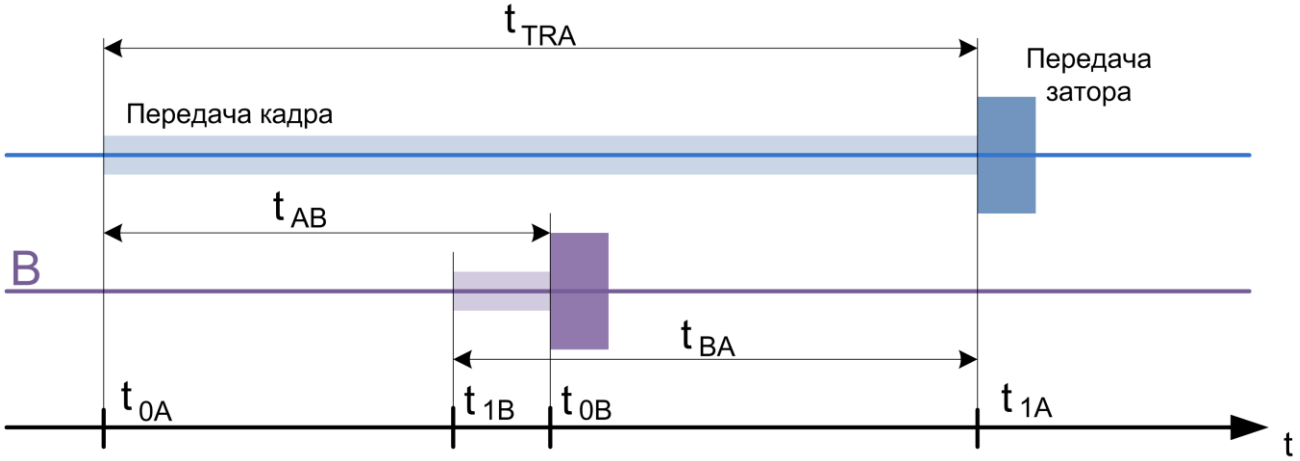
Ошибка выравнивания (alignment error) — кадр, длина которого не кратна байту. Может порождаться неисправным адаптером, трансивером, кабелем.

Ошибка контрольного кода (CRC error) — кадр правильной длины, но с неправильным CRC-кодом. Может порождаться помехами, слишком большой длиной кабеля.

На вышестоящие протокольные уровни передаются только кадры, не имеющие перечисленных ошибок.

Трансивер, как относительно независимый узел, может (и должен) контролировать работу адаптера. Если он обнаружил «болтливость» адаптера (слишком долгое формирование сигналов передачи), формирование сигналов передачи), он прекращает передачу в линию и блокируется до тех пор, пока адаптер не «помолчит» определенное время. Таким образом обеспечивается защита среды передачи от ее монопольного захвата неисправным узлом. Адаптер может считать, что ему удалось получить доступ к среде передачи, если он не обнаружил коллизий при передаче первых 64 байт кадра, и рапортовать об этом на более высокий протокольный уровень.

Если он обнаружил коллизию, то обязан вместо продолжения пакета послать короткую (32-48-битовую) цепочку затор (jam), после чего прекратить передачу. Цель посылки затора — дать возможность всем передатчикам, вовлеченным в коллизию, ее заметить. Посылкой затора обеспечивается оповещение о коллизии узлов, разделенных повторителями. Ситуация, когда коллизия обнаружена позже. 64-байтно-го окна (collision window), называется поздней коллизией (later collision) и является ненормальной для сети Ethernet. Интервал времени до повторной попытки доступа t(RT) определяется через интервал отсрочки TS и случайное число t, зависящее от номера попытки n: t(RT)=TS x t. Интервал отсрочки TS называется также тайм-слотом (time slot) и составляет 512 bt. Число t является случайным целым, равномерно распределенным в диапазоне от 0 до 2 в степени n для n = 1, 2,...10 и в диапазоне от 0 до2 в степени 10 для n > 10. После 16 неудачных попыток передачи адаптер отказывается от дальнейших попыток доступа, сообщая о неудаче на вышестоящие уровни. Максимальное время между двумя повторными попытками может доходить до 2 в 10й степени x TS = 524 288 bt ~ 52,4 мс, минимальное — 0 (сразу после зазора). С механизмом обнаружения коллизии связаны пространственные ограничения на размер домена коллизий, обусловленные конечностью скорости распространения сигнала в среде передачи и задержками, вносимыми повторителями. На рисунке приведена временная диаграмма действий двух узлов, заметно удаленных друг от друга.

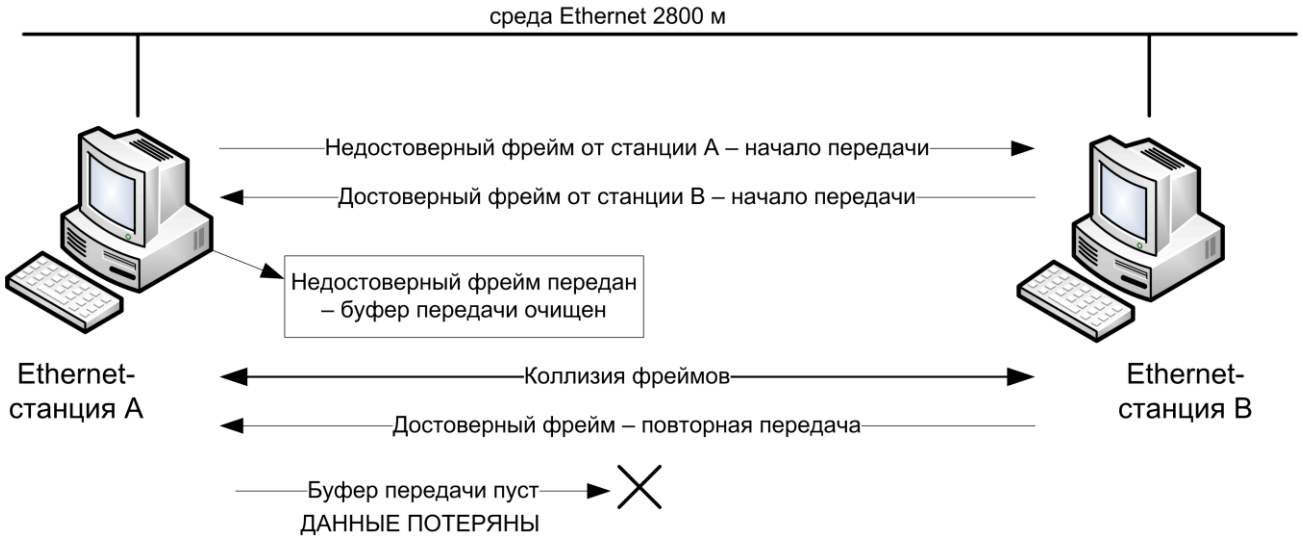


Пусть узел A начал передавать кадр в момент t0A, и вскоре появилась потребность в передаче у узла B. Узел B будет видеть линию свободной вплоть до момента t0B, и в момент tlB ему ничто не мешает

начать передачу. Вскоре его передатчик обнаружит коллизию, и он вместо продолжения кадра начнет передавать сигнал затора. Передатчик А обнаружит коллизию только в момент tlA и тоже прекратит передачу кадра. Максимальное время, в течение которого передатчик А.будет «беззаботно» передавать, пакет, составит время t(TRAmax) = t(AB)+ t (BA) — так называемое время двойного оборота по сети (round trip, time). Это время плюс время на передачу затора должно быть меньше, чем время передачи самого короткого кадра, иначе кадры, оборванные коллизией, приемник будет пытаться трактовать как нормальные. Таким образом, время двойного оборота не должно превышать время передачи кадра минимальной длины. Для надежности берут еще и запас, с учетом которого время двойного оборота не должно превышать 45 мкс. Поскольку сеть симметрична, для определения ограничений достаточно определить время прохождения сигнала между двумя самыми удаленными друг от друга узлами домена коллизий. В это время входит задержка распространения сигнала в кабеле, задержки вносимые повторителями (если они встречаются на пути), и время реакции адаптера на обнаружение коллизии. Это время, не должно превышать .25,6 мкс, а для надежности следует еще оставить запас в 1-5 мкс. Расстояние между максимально удаленными узлами называется диаметром домена коллизий. Скоростные технологии — Fast Ethernet и Gigabit Ethernet — имеют тот же механизм обнаружения коллизий, и из-за более высокой частоты передачи (bt=10 ns в Fast и bt=1 ns в Gigabit Ethernet) ограничения на диаметр домена коллизий жестче. Для их смягчения в Gigabit Ethernet пошли на увеличение минимального размера кадра (следует заметить что в Gagabit Ethenet вообще отказались от повторителей, т.е. все соединения по сути работают точка-точка и лишь на канальном уровне реализован механизм широковещания).

Диаметр сети Ethernet и ее интервал

Диаметр сети определяется расстоянием между Ethernet-станциями, расположен ными на максимально удаленных (противоположных) сторонах широковещательного домена. Устройства могут быть соединены с использованием хабов, повторителей, коммутаторов или мостов. Правила, установленные для сетей Ethernet стандарта 802.3, требуют, чтобы коллизия была обнаружена в течение времени, которое необходимо для передачи наименьшего по длительности фрейма, допустимого в сети Ethernet. Размер наименьшего допустимого фрейма составляет 64 байт, или 512 бит. С учетом скорости передачи электрического сигнала по проводам и скорости передачи данных (10 Мбит/с) получаем, что максимально допустимая длина провода в сетях Ethernet составляет 2800 м. Время, необходимое фрейму Ethernet для преодоления диаметра се ти, называется интервалом Ethernet (slot time Ethernet). Рассмотрим две станции, расположенные на противоположных сторонах широковещательного домена.



Станция А передает фрейм размером менее 512 бит
В тот же самый момент времени начинает передачу фрейма станция В
Станция А передает последний бит фрейма
Станция А не обнаруживает коллизию при передаче и выгружает фрейм из буфера передачи
Станция А полагает, что станция назначения переданного фрейма приняла его
Фрейм станции А вступает в коллизию с фреймом станции В
Станция А уже выгрузила фрейм из буфера передачи и поэтому не может передать его повторно

Этот сценарий справедлив и для случая, когда протяженность передающей среды превышает 2800 м

CSMA/CD — это методология, на которой основаны полудуплексный Ethernet и Fast Ethernet. Как уже говорилось ранее, технология CSMA/CD напоминает селекторное совещание, проводимое с помощью телефонной связи. Каждый участник должен ждать, пока среда передачи освободится, и только после этого он может говорить. В 1995 году IEEE утвердил стандарт 802.3x, в котором описывается новая методология передачи сигналов в сетях Ethernet, известная как работа в полнодуплексном режиме. Такой режим работы позволяет передавать и принимать сигналы одновременно, благодаря чему более полно используются возможности среды передачи

Полнодуплексный режим применим только к устройствам, соединенным по схеме "точка—точка".

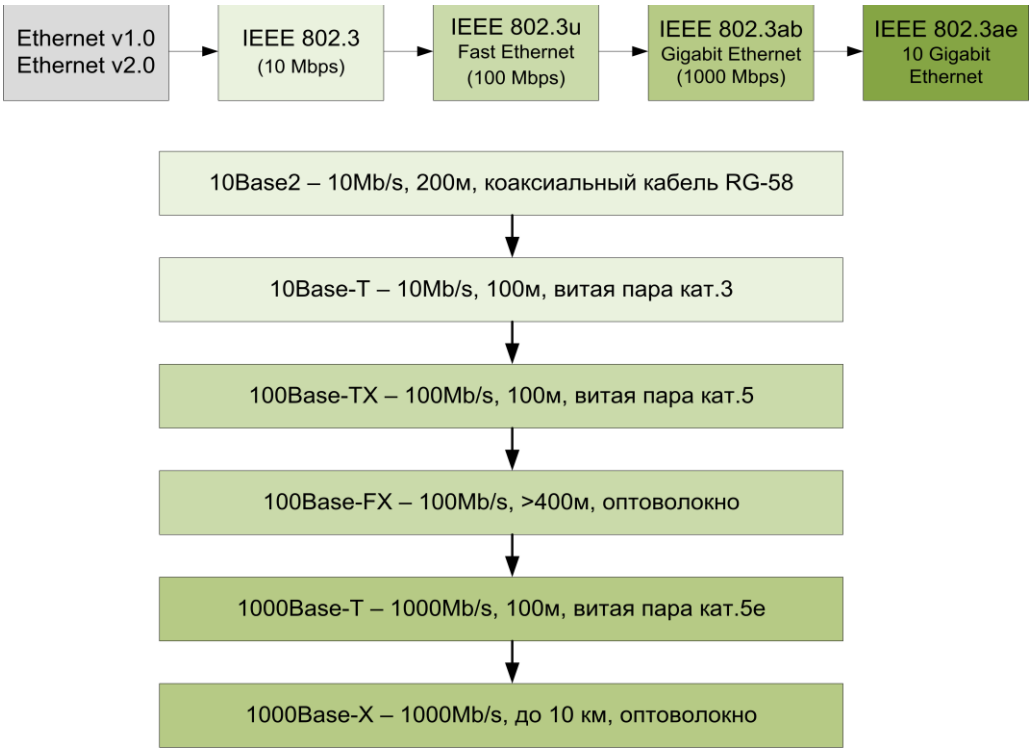
Полнодуплексные устройства не могут работать совместно с полудуплексными, т.к. возникают ошибки дуплексного рассогласования. Полнодуплексные устройства начинают передавать данные, как только могут сделать это, не контролируя наличие несущей в среде передачи. Если полудуплексное устройство передает в это время информацию, возникает коллизия, которую полнодуплексное устройство не обнаруживает.

Этапы развития Ethernet

Ethernet имеет ряд разновидностей.

10BASE-T — наиболее часто используемая Ethernet-среда, представляющая собой витую пару. Она позволяет создавать сети с помощью кабелей на основе неэкранированных витых пар (категории 3). 10BASE-T позволяет передавать сигнал на расстояние, примерно равное 100 м, хотя сама по себе технология Ethernet работоспособна при расстояниях между станциями до 2800 м. Разница в возможностях обусловлена затуханием сигнала в кабелях, созданных на основе неэкранированных витых пар.

Прежде чем топология 10BASE-T завоевала популярность, в мире небольших сетей Ethernet тон задавала топология 10BASE2. Под этой аббревиатурой понимается передача сигнала со скоростью 10 Мбит/с на расстояние до 200 м по коаксиальному кабелю типа RG-58. Хотя цифра "2" в обозначении указывает на 200 м, максимально допустимая длина кабеля для сетей 10BASE2 составляет 185 м. Технология 10BASE2 была популярной потому, что кабель стоил относительно недорого и сеть можно было быстро развернуть. Сигналы в сетях 10BASE2 передаются по кабелю RG-58 или RG-59, и вся сеть физически соединена в одну непрерывную линию. Станции подключены непосредственно к среде передачи с помощью Т-образных соединителей. Повреждение кабеля на любом участке приводит к выходу из строя всей сети.



По мере того как Ethernet становился все более востребованным стандартом передачи данных в сетях, пользователи начинали требовать расширения полосы пропускания. В 1995 году IEEE анонсировала стандарт 802.3u – Ethernet со скоростью 100 Мбит/с. Наибольшее распространение получили два стандарта: 100BASE-TX и 100BASE-FX

100BASE-TX ориентируется на кабели категории 5 на основе витой пары и во многом аналогична технологии 10BASE-T, но, в отличие от нее, рассчитана на использование кабеля категории 5, а не 3.

Ограничения на расстояния при использовании технологии 100BASE-TX точно такие же (100 м), как и в 10BASE-T.

Технология 100BASE-FX использует такой же механизм кодирования, как и 100BASE-TX, но на этом сходство между ними и заканчивается. Поскольку 100BASE-FX использует в качестве носителя данных свет, ее сигналы не подвержены влиянию электромагнитных помех. Благодаря этому можно использовать более совершенные схемы передачи сигналов. Диаметр сети для технологии 100BASE-FX составляет примерно 400 м при работе в полудуплексном режиме. В сетях на основе 100BASE-FX можно использовать также полнодуплексный режим.

Gigabit Ethernet имеет две основные разновидности.

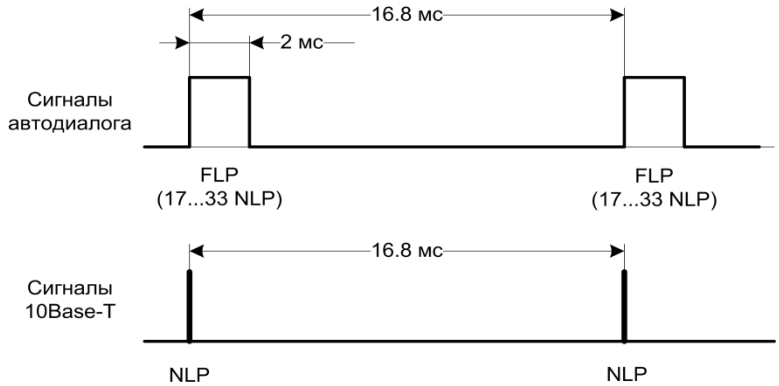
1000BASE-T. Как и технологии 10BASE-T и 100BASE-TX, может использовать кабели с неэкранированными витыми парами длиной не более 100 м.

1000BASE-X имеет несколько вариантов, основанных на волоконно-оптической среде передачи

Автоматическое согласование (Auto-Negotiation)

Функция автодиалога или автосогласования (так можно перевести Auto-Negotiation) позволяет адаптерам, в которых предусмотрено переключение скорости передачи, автоматически подстраиваться под скорость обмена в сети, а концентраторам, в которых предусмотрен авто диалог, самим определять скорость передачи адаптеров, подключенных к их портам. При этом пользователь сети не должен следить за тем, на какую скорость обмена настроена его аппаратура: система сама выберет максимально возможную скорость.

Сразу отметим, что режим автодиалога применяется только в сетях на основе сегментов, использующих витые пары. Для сегментов на базе коаксиального кабеля и оптоволоконного кабеля автодиалог не предусмотрен. Шинные сегменты на коаксиальном кабеле не дают возможности двухточечной связи, а в оптоволоконных сегментах применяется другая система служебных сигналов.



Автодиалог основан на использовании сигналов, передаваемых в Fast Ethernet, которые называются FLP (Fast Link Pulse) по аналогии с сигналами NLP (Normal Link Pulse), применяемыми в сегментах 10BASE-T. Так же, как и NLP, сигналы FLP начинают вырабатываться с включением питания соответствующей аппаратуры (адаптера или концентратора) и формируются в паузах между передаваемыми сетевыми пакетами, поэтому они никак не влияют на загрузку сети. Именно сигналы FLP и передают информацию о возможностях подключенной к данному сегменту аппаратуры.

Так как аппаратура 10BASE-T разрабатывалась до создания механизма автодиалога, для автоматического определения типа сети необходимо обрабатывать не только сигналы FLP, но и сигналы NLP. Это также предусмотрено в аппаратуре, поддерживающей автодиалог. Естественно, в такой аппаратуре, как правило, предусматривается и возможность отключения режима автодиалога, чтобы пользователь сам мог задать режим работы своей сети.

Как известно из теории связи, связь бывает симплексная (всегда только в одну сторону), полудуплексная (по очереди то в одну сторону, то в другую) и полнодуплексная (одновременно в две стороны). Классический Ethernet использует полудуплексную связь: по его кабелю в разное время может проходить разнонаправленная информация. Это позволяет легко реализовать обмен между большим

количеством абонентов, но требует сложных методов доступа к сети (CSMA/CD). Полнодуплексная версия Ethernet гораздо проще. Она предназначена для обмена только между двумя абонентами по двум разнонаправленным кабелям, причем передавать могут оба абонента сразу. Два преимущества такого подхода понятны сразу: во-первых, не требуется никакого механизма доступа к сети, а во-вторых, в идеале пропускная способность такой линии связи оказывается вдвое выше, чем при полудуплексной передаче. Полнодуплексные версии Ethernet и Fast Ethernet находятся еще на стадии стандартизации, поэтому единых правил обмена пока не выработано, и аппаратура разных производителей может основываться на разных принципах обмена. Тем не менее, автодиалог уже ориентирован на их распознавание и использование.

При проведении автодиалога применяется таблица приоритетов, в которой полнодуплексные версии имеют более высокие приоритеты, чем классические полудуплексные, так как они более быстрые.

Из таблицы следует, что если аппаратура на обоих концах сегмента поддерживает обмен с двумя скоростями, например, в режимах 10BASE-T и 100B ASE-TX, то в результате автодиалога будет выбран режим 100BASETX, как имеющий больший приоритет (обеспечивающий большую скорость).

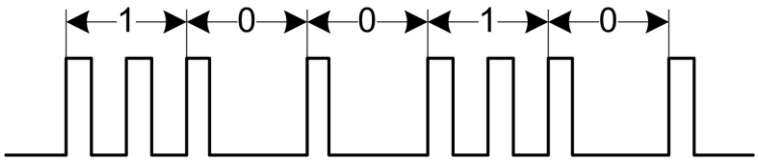
Приоритет	Тип сети
1 (высший)	100BASE-TX Full Duplex
2	100BASE-T4
3	100BASE-TX
4	10BASE-T Full Duplex
5 (низший)	10BASE-T

Автодиалог предусматривает также разрешение ситуаций, когда на одном конце кабеля подключена двухскоростная аппаратура, а на другом -односкоростная. Например, если двухскоростной адаптер присоединен к концентратору 10BASE-T, в котором не предусмотрена возможность автодиалога, то он не будет получать сигналов FLP, а будет получать только сигналы NLP. В результате действия механизма автодиалога адаптер будет переключен в режим концентратора 10BASE-T. Точно так же, если двухскоростной концентратор присоединен к односкоростному адаптеру 100BASE-TX, не рассчитанному на автодиалог, то концентратор перейдет в режим адаптера 100BASE-TX. Этот механизм одностороннего определения типа сети называется параллельным детектированием.

Естественно, в любом случае автодиалог не может обеспечить большей скорости, чем самый медленный из компонентов сети.

Помимо собственно определения типа сети и выбора максимально возможной скорости обмена автодиалог обеспечивает и некоторые дополнительные возможности. В частности, он позволяет определять, почему нарушилась связь в процессе работы, а также обмениваться информацией об ошибках. Для передачи этой дополнительной информации используется тот же самый механизм, что и для основного автодиалога, но только после того, как установлен тип сети и скорость передачи. Данная функция называется «функцией следующей страницы» (Next Page function).

Обмен информацией при автодиалоге производится посылками (пакетами) FLP-импульсов, которыми кодируется 16-битное слово. Каждая посылка содержит от 17 до 33 импульсов, идентичных импульсам NLP, которые используются в 10BASE-T. Посылки имеют длительность около 2 мс и передаются с периодом 16,8 мс.



Код, применяемый при автодиалоге

Для кодирования битов в FLP применяется следующий код. В начале каждого битового интервала передается импульс. В середине бита, соответствующего логической единице, передается еще один импульс. В середине бита, соответствующего логическому нулю импульса нет. В начале посылки передается стартовый нулевой бит, именно поэтому общее количество импульсов в посылке FLP может изменяться в пределах от 17 до 33.

Обмен информацией при автодиалоге осуществляется 16-битными словами, называемыми LCW (Link Code Word), с форматом, представленным на рисунке.

Формат слова LCW, применяемого в автодиалоге

Пятиразрядное поле селектора (Selector Field) определяет один из 32 возможных типов стандарта сети.

Восьмиразрядное поле технологических особенностей (Technology Ability Field) определяет тип сети в пределах стандарта, заданного битами поля селектора. Для стандарта IEEE 802.3 пока что определены пять типов.

Бит удаленной ошибки RF (Remote Fault) позволяет передавать информацию о наличии ошибок. Бит подтверждения Ack (Acknowledge) используется для подтверждения получения посылки. Наконец, бит следующей страницы NP (Next Page) говорит о поддержке функции следующей страницы, о том, что абонент собирается передавать еще и дополнительную информацию.

Вероятно, в дальнейшем принцип автодиалога будет совершенствоваться и развиваться, включая в себя другие стандарты и типы сети, давая возможность решения все новых задач. Но его реализация в принципе невозможна при стандартной топологии «шина», поэтому, скорее всего, доля шинных сегментов (10BASE2 и 10BASE5) будет все больше сокращаться. В новых сетях (Fast Ethernet, Gigabit Ethernet) шинные сегменты вряд ли появятся.