[**1. ПРАВИЛА И ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ ПЕРЕНОСИМЫХ ПРОГРАММ.** 3](#_Toc123068772)

[**2. СЕТЕВОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ. АРХИТЕКТУРА СЕТИ. ИНКАПСУЛЯЦИЯ ДАННЫХ.** 6](#_Toc123068773)

[**3. ОСНОВНЫЕ ПРОБЛЕМЫ, ВОЗНИКАЮЩИЕ ПРИ РАЗРАБОТКЕ СТЕКА ПРОТОКОЛОВ.** 7](#_Toc123068774)

[**4. СЛУЖБЫ И ПРОТОКОЛЫ. ТИПЫ СЛУЖБ.** 8](#_Toc123068775)

[**5. СЛУЖБЫ И ПРОТОКОЛЫ. ПРИМИТИВЫ СЛУЖБ.** 9](#_Toc123068776)

[**6. ЭТАЛОННАЯ МОДЕЛЬ TCP/IP.** 10](#_Toc123068777)

[**7. ПРОТОКОЛ IPV4. АДРЕСАЦИЯ IPV4.** 11](#_Toc123068778)

[**8. ПРОТОКОЛ IPV4. ФРАГМЕНТАЦИЯ IPV4-СЕГМЕНТОВ.** 12](#_Toc123068779)

[**9. ХАРАКТЕРИСТИКА, НАЗНАЧЕНИЕ И ОСОБЕННОСТИ РАБОТЫ ПРОТОКОЛА ICMP.** 13](#_Toc123068780)

[**10. ХАРАКТЕРИСТИКА ПРОТОКОЛА UDP. ПОНЯТИЕ «ПОРТ ПРОТОКОЛА».** 14](#_Toc123068781)

[**11. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. ПОНЯТИЕ «ПОРТ ПРОТОКОЛА».** 15](#_Toc123068782)

[**12. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. НАЗНАЧЕНИЕ ФЛАГОВ В TCP-СЕГМЕНТЕ.** 16](#_Toc123068783)

[**13. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. УСТАНОВЛЕНИЕ И РАЗРЫВ СОЕДИНЕНИЯ TCP (НА УРОВНЕ ПРОТОКОЛА).** 17](#_Toc123068784)

[**14. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. УПРАВЛЕНИЕ ПОТОКОМ. СКОЛЬЗЯЩЕЕ ОКНО. МЕХАНИЗМ МЕДЛЕННОГО СТАРТА.** 18](#_Toc123068785)

[**15. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. АЛГОРИТМ НЭГЛА (NAGLE) И СИНДРОМ УЗКОГО ОКНА.** 19](#_Toc123068786)

[**16. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. ДИАГРАММА СОСТОЯНИЙ TCP.** 20](#_Toc123068787)

[**17. ПОНЯТИЕ ИНТЕРФЕЙСА СОКЕТОВ. ФУНКЦИОНАЛЬНЫЕ СРЕДСТВА ИНТЕРФЕЙСА.** 21](#_Toc123068788)

[**18. ПОНЯТИЕ СОКЕТА. ТИПЫ И СТРУКТУРЫ СОКЕТОВ. СЕМЕЙСТВА АДРЕСОВ.** 22](#_Toc123068789)

[**19. СОКЕТЫ BERKELEY. РАЗРЕШЕНИЕ ИМЕН ХОСТОВ, ПРОТОКОЛОВ, СЕТЕВЫХ СЕРВИСОВ.** 23](#_Toc123068790)

[**20. СОКЕТЫ BERKELEY. ОСНОВНЫЕ СИСТЕМНЫЕ ВЫЗОВЫ.** 24](#_Toc123068791)

[**21. СОКЕТЫ BERKELEY. ОПЦИИ СОКЕТОВ. УРОВНИ ОПЦИЙ.** 25](#_Toc123068792)

[**22. МОДЕЛЬ СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ КЛИЕНТ-СЕРВЕР.** 26](#_Toc123068793)

[**23. ТРЕХУРОВНЕВАЯ КЛИЕНТ-СЕРВЕРНАЯ МОДЕЛЬ, ВАРИАНТЫ РАСПРЕДЕЛЕНИЯ НА ФИЗИЧЕСКИ ДВУХЗВЕННОЙ АРХИТЕКТУРЕ. ВЕРТИКАЛЬНОЕ, ГОРИЗОНТАЛЬНОЕ И PEER-TO-PEER РАСПРЕДЕЛЕНИЕ.** 27](#_Toc123068794)

[**24. АРХИТЕКТУРА КЛИЕНТА. ОПРЕДЕЛЕНИЕ МЕСТОНАХОЖДЕНИЯ СЕРВЕРА.** 29](#_Toc123068795)

[**25. АРХИТЕКТУРА КЛИЕНТА. АЛГОРИТМЫ РАБОТЫ КЛИЕНТОВ. ФУНКЦИЯ CONNECT ДЛЯ UDP.** 30](#_Toc123068796)

[**26. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ ПОСЛЕДОВАТЕЛЬНОГО СЕРВЕРА С УСТАНОВЛЕНИЕМ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ.** 31](#_Toc123068797)

[**27. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ ПОСЛЕДОВАТЕЛЬНОГО СЕРВЕРА БЕЗ УСТАНОВЛЕНИЯ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ.** 32](#_Toc123068798)

[**28. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ ПАРАЛЛЕЛЬНОГО СЕРВЕРА БЕЗ УСТАНОВЛЕНИЯ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ.** 33](#_Toc123068799)

[**29. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ ПАРАЛЛЕЛЬНОГО СЕРВЕРА С УСТАНОВЛЕНИЕМ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ.** 34](#_Toc123068800)

[**30. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ СЕРВЕРА С АСИНХРОННЫМ ВВОДОМ-ВЫВОДОМ.** 35](#_Toc123068801)

[**31. КЛАССИФИКАЦИЯ СЕРВЕРОВ. СЕРВЕРЫ С ПРЕДВАРИТЕЛЬНЫМ ПОРОЖДЕНИЕМ ПОТОКОВ.** 36](#_Toc123068802)

[**32. КЛАССИФИКАЦИЯ СЕРВЕРОВ. СЕРВЕРЫ С ПРЕДВАРИТЕЛЬНЫМ ПОРОЖДЕНИЕМ ПРОЦЕССОВ.** 37](#_Toc123068803)

[**33. КЛАССИФИКАЦИЯ СЕРВЕРОВ. МУЛЬТИСЕРВИСНЫЕ СЕРВЕРЫ (НА ПРИМЕРЕ XINETD).** 38](#_Toc123068804)

[**34. КЛАССИФИКАЦИЯ СЕРВЕРОВ. МЕТОДЫ УЛУЧШЕНИЯ ФУНКЦИОНИРОВАНИЯ СЕРВЕРОВ.** 39](#_Toc123068805)

[**35. ТЕКСТОВОЕ И БИНАРНОЕ ПРЕДСТАВЛЕНИЕ ДАННЫХ В ПРОТОКОЛАХ.** 40](#_Toc123068806)

[**36. ПРОЕКТИРОВАНИЕ ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ. МЕХАНИЗМЫ ПРОТОКОЛА.** 41](#_Toc123068807)

[**37. ПРОЕКТИРОВАНИЕ ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ. СВОЙСТВА ПРОТОКОЛА.** 43](#_Toc123068808)

[**38. ХАРАКТЕРИСТИКА И ФУНКЦИОНИРОВАНИЕ ПРОТОКОЛА HTTP.** 45](#_Toc123068809)

[**39. ХАРАКТЕРИСТИКА ПРОТОКОЛА HTTP. ЗАПРОСЫ HTTP.** 46](#_Toc123068810)

[**40. ХАРАКТЕРИСТИКА ПРОТОКОЛА HTTP. HTTP-ОТВЕТ.** 48](#_Toc123068811)

[**41. ХАРАКТЕРИСТИКА ПРОТОКОЛА HTTP. БЕЗОПАСНОСТЬ, АУТЕНТИФИКАЦИЯ, COOKIES.** 49](#_Toc123068812)

[**42. ПРОЕКТИРОВАНИЕ ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ НА БАЗЕ HTTP.** 50](#_Toc123068813)

[**43. ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА ICMP (НА ПРИМЕРЕ РЕАЛИЗАЦИИ УТИЛИТ PING ИЛИ TRACEROUTE).** 53](#_Toc123068814)

[**44. АТАКИ ICMP-FLOODING И SMURF: ОСОБЕННОСТИ РЕАЛИЗАЦИИ.** 55](#_Toc123068815)

[**45. IP-SPOOFING. ОСОБЕННОСТИ СОЗДАНИЯ ПАКЕТОВ ПРОТОКОЛА TCP (НА ПРИМЕРЕ АТАК SYN-FLOODING И LAND).** 56](#_Toc123068816)

[**46. ПРОТОКОЛ IPV6. ХАРАКТЕРИСТИКИ. ПРЕИМУЩЕСТВА И ОТЛИЧИЯ ОТ IPV4.** 57](#_Toc123068817)

[**47. ПРОТОКОЛ IPV6. ТИПЫ АДРЕСОВ. ЗАПИСЬ АДРЕСА.** 59](#_Toc123068818)

[**48. МНОГОАДРЕСНАЯ ПЕРЕДАЧА. ПОНЯТИЕ И МЕХАНИЗМЫ. АДРЕСАЦИЯ И ОБЛАСТЬ ДЕЙСТВИЯ IPV4.** 61](#_Toc123068819)

[**49. МНОГОАДРЕСНАЯ ПЕРЕДАЧА. ПОНЯТИЕ И МЕХАНИЗМЫ. АДРЕСАЦИЯ И ОБЛАСТЬ ДЕЙСТВИЯ IPV6.** 63](#_Toc123068820)

# **1. ПРАВИЛА И ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ ПЕРЕНОСИМЫХ ПРОГРАММ.**

Главные ориентиры:

- кроссплатформенная переносимость

- открытые стандарты

Дуг Макилрой, изобретатель каналов «pipes», сформулировал несколько постулатов, применимых для разработки ПО:

- пишите программы, которые выполняют одну функцию и делают это хорошо;

- пишите программы, которые будут работать вместе;

- пишите программы, поддерживающие текстовые потоки, поскольку они являются универсальным интерфейсом.

"Философия" UNIX - это не философия, а общие рекомендации по проектированию ПО, накопленные сообществом программистов на опыте десятилетий разработок программ, которые взаимодействуют друг с другом.

Правила:

1. Правило модульности

Следует писать простые части, связанные ясными интерфейсами.

Единственным способом создания сложной программы, не обреченной заранее на провал, является сдерживание ее глобальной сложности.

Т.е. построение программы из простых частей, соединенных четко определенными интерфейсами, так что большинство проблем являются локальными, и тогда можно рассчитывать на обновление одной из частей без разрушения целого.

1. Правило ясности

Ясность лучше чем мастерство.

Последующее обслуживание программы важная и дорогостоящая часть жизненного цикла программы.

Писать программы необходимо так, как если бы вы знали, что последующей поддержкой будет заниматься неуравновешенный псих с топором, знающий ваш домашний адрес!

1. Правило композиции

Следует разрабатывать программы, которые будут взаимодействовать с другими программами.

Если разрабатываемые программы не способны взаимодействовать друг с другом, то очень трудно избежать создания сложных монолитных программ.

Методы взаимодействия могут быть сильными и слабыми по возможности рекомендуется использовать слабые методы и текстовые форматы передачи данных.

1. Правило разделения

Следует отделять политику от механизма и интерфейсы от основных модулей (engine).

Примеры политики и механизма:

вид GUI и операции отрисовки, клиент (front-end) - сервер (back-end), сценарии и библиотеки и др.

При жесткой связи политики и механизма:

- политика становится негибкой и усложняется ее изменение;

- изменение политики имеет строгую тенденцию к дестабилизации механизмов.

1. Правило простоты

Необходимо проектировать простые программы и «добавлять сложность» только там, где это необходимо.

Основные причины добавления сложности:

- человеческий фактор (часто желание «выпендриться»);

- проектные требования, продиктованные текущей модой, маркетингом или «левой пяткой заказчика»;

1. Правило расчетливости

Пишите большие программы, только если после демонстрации становится ясно, что ничего другого не остается.

Под «большими программами» здесь понимаются программы с большим объемом кода и значительной внутренней сложностью.

1. Правило прозрачности

Для того, чтобы упростить проверку и отладку программы, ее конструкция должна быть обозримой.

Программа прозрачна, если при ее минимальном изучении можно понять, что она делает и как.

Программа воспринимаема, когда она имеет средства для мониторинга и отображения внутреннего состояния.

Необходимо использовать достаточно простые форматы входных и выходных данных.

Интерфейс должен быть приспособлен для использования в отладочных сценариях.

1. Правило устойчивости

Устойчивость следствие прозрачности и простоты.

Программа является устойчивой, когда она выполняет свои функции в неожиданных условиях, которые выходят за рамки предположений разработчика, как и в нормальных условиях.

Программа является простой, если происходящее в ней не представляется сложным для восприятия человеком.

Один из способов организации модульность (простые блоки, ясные интерфейсы)

Следует избегать частных случаев!

1. Правило представления

Знания следует оставлять в данных, чтобы логика программы могла быть примитивной и устойчивой.

Даже простую логику бывает сложно проверить, но даже сложные структуры данных являются довольно простыми для моделирования и анализа (например диаграмма 50 узлов дерева и блок-схема 50 строк кода)

Если можно выбирать между усложнением структуры данных и усложнением кода, то лучше выбирать первое.

Примеры: ascii, генератор html-таблицы.

1. Правило наименьшего удивления

При проектировании интерфейсов всегда следует использовать наименее неожиданные элементы.

Необходимо учитывать характер предполагаемой аудитории и традиции платформы.

Оборотная сторона: следует избегать создания внешне похожих вещей, слегка отличающихся в действительности, поскольку кажущаяся привычность порождает ложные ожидания.

1. Правило тишины

Если программе нечего сказать, то пусть лучше молчит.

Внимание и сосредоточенность пользователя ценный и ограниченный ресурс, который требуется только в случае необходимости.

Важная информация не должна смешиваться с подробными сведениями о работе программы.

1. Правило восстановления

Когда программа завершается аварийно, это должно происходить явно (шумно) и по возможности быстро.

Если программа не способна справиться с ошибкой, то необходимо завершить ее работу так, чтобы максимально упростить диагностику.

Для сетевых служб следует следовать рекомендации Постела: «Будьте либеральны к тому, что принимаете, и консервативны к тому, что отправляете»

1. Правило экономии

Время программиста дорого поэтому задача экономии его времени более приоритетна, по сравнению с экономией машинного времени.

Компьютер железный ему не скучно (с) программистская мудрость

Использование высокоуровневых языков и «обучение» машины выполнять больше низкоуровневой работы по программированию, что приводит к правилу 14.

1. Правило генерации

Избегайте кодирования вручную; если есть возможность пишите программы для создания программ.

Использование генераторов кода оправданно, когда они могут повысить уровень абстракции, т.е. когда язык спецификации для генератора проще, чем сгенерированный код, и код впоследствии не потребует ручной доработки.

Примеры: грамматические и лексические анализаторы, генераторы make-файлов, построители GUI-интерфейсов.

1. Правило оптимизации

Сначала опытный образец, потом оптимизирование.

Добейтесь стабильной работы, только потом оптимизируйте.

Керниган и Плоджер:

90% актуальной и реальной функциональности лучше, чем 100% функциональности перспективной и сомнительной

Кнут: преждевременная оптимизация корень всех зол

Кент Бек (экстремальное программирование): заставьте программу работать, заставьте работать ее верно, а затем сделайте ее быстрой

1. Правило разнообразия

Не следует доверять утверждениям о «единственно правильном пути».

Никто не обладает умом, достаточным для оптимизации всего или для предвидения всех возможных вариантов использования создаваемой программы.

1. Правило расширяемости

Разрабатывайте для будущего. Оно наступит быстрее, чем вы думаете.

При проектировании протоколов или форматов файлов следует делать их самоописательными, для того, чтобы их можно было расширить.

Всегда, следует либо включать номер версии, либо составлять формат из самодостаточных, самоописательных команд так, чтобы можно было легко добавить новые директивы, а старые удалить, «не сбивая с толку» код чтения формата.

Все правила сразу: K.I.S.S. (Keep It Simple, Stupid!)

Правило SPOT (Single Point of Truth)

Внутри системы каждый блок знаний должен иметь единственное, недвусмысленное и надежное представление.

Дублирование данных?

- Если дублирование данных существует из-за необходимости иметь два различных представления в двух различных местах, то возможно ли написать функцию, средство или генератор кода для создания одного представления из другого или обоих из общего источника?

- Если документация дублирует данные из кода, то можно ли создать фрагменты документации из кода или наоборот, или и то, и другое из общего представления более высокого уровня?

- Если файлы заголовков и объявления интерфейсов дублируют сведения в реализации кода, то существует ли способ создания файлов заголовков и объявлений интерфейсов из данного кода?

SPOT для структур данных

«нет лишнего нет путаницы»

Предпочтительна структура данных, состояния которой имеют однозначное соответствие с состояниями реальной системы, которая будет моделироваться.

# **2. СЕТЕВОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ. АРХИТЕКТУРА СЕТИ. ИНКАПСУЛЯЦИЯ ДАННЫХ.**

Сетевое ПО

При создании первых ЛВС основное внимание уделялось аппаратуре, а вопросы программного обеспечения оставлялись на потом. Современное сетевое программное обеспечение в высокой степени структурировано.

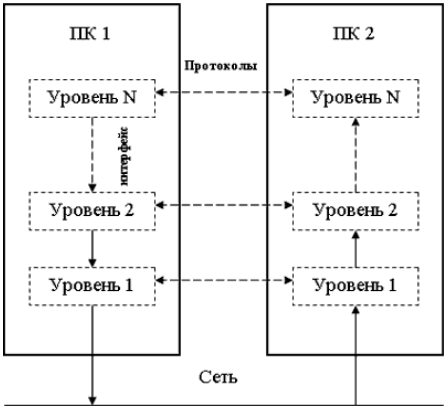
Архитектура сети

Набор уровней и протоколов называется архитектурой сети.

Спецификация архитектуры должна содержать достаточно информации для написания программного обеспечения и создания аппаратуры для каждого уровня, чтобы они выполняли требования протокола.

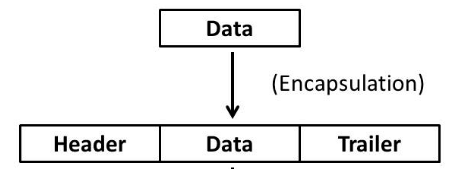
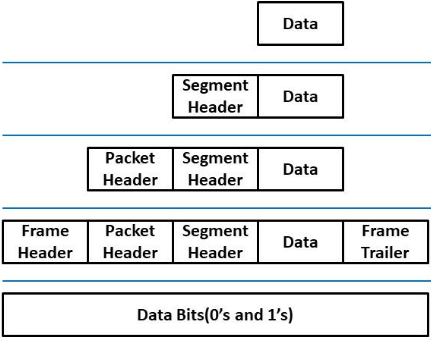
Детали реализации и спецификации интерфейсов не являются частями архитектуры, поскольку они спрятаны внутри машины.

При этом не требуется, чтобы интерфейсы были одинаковыми, главное, чтобы протоколы соответствовали спецификации.



Инкапсуляция данных - это процесс, в котором некоторая дополнительная информация добавляется к элементу данных, чтобы добавить к нему некоторые функции. Передача данных обычно проходит через различные уровни, поэтому происходит добавление к передаваемым данным информации протокола, чтобы передача данных могла происходить надлежащим образом. Эта информация может быть добавлена в заголовок (header) или в конец (footer или trailer) данных.

Каждый уровень берет инкапсулированные данные из предыдущего слоя и добавляет некоторую дополнительную информацию для их инкапсуляции и некоторые другие функции с данными. Эти функции могут включать в себя последовательность данных, контроль и обнаружение ошибок, управление потоком, контроль перегрузки, информацию о маршрутизации и так далее.



# **3. ОСНОВНЫЕ ПРОБЛЕМЫ, ВОЗНИКАЮЩИЕ ПРИ РАЗРАБОТКЕ СТЕКА ПРОТОКОЛОВ.**

Проблемы при разработке стека протоколов

Некоторые из ключевых аспектов разработки, возникающих при создании компьютерных сетей, присутствуют на нескольких уровнях.

Адресация

Каждый уровень нуждается в механизме идентификации отправителей и получателей, следовательно, необходима система адресации.

Логические каналы

Протокол должен определять количество логических каналов, относящихся к соединению, а также их приоритеты. Так, многие сети обеспечивают как минимум два канала на соединение: один для обычных данных, другой для срочных.

Контроль над ошибками

В протоколе должен быть предусмотрен контроль над ошибками. Получатель должен иметь возможность сообщить отправителю, какие из сообщений были получены неправильно.

Нумерация

Некоторые каналы связи (ориентированные на коммутацию пакетов) могут доставлять сообщения не в порядке отправления, поэтому протокол должен явно снабжать сообщение номерами, чтобы их можно было собрать в правильном порядке.

Управление потоком

Также возникает вопрос: что делать, чтобы более быстрая сторона не завалила своими пакетами более медленную? Одно из решений контроль состояния сторон, другое решение договоренность по ограничению скорости передачи между сторонами. В целом это называется управлением потоком.

Размер сообщений

Еще одна проблема возникает с размерами сообщений: что делать с огромными сообщениями и очень малыми, которые неэффективно пересылать? Например, разбивать пакет на много малых либо наоборот.

Мультиплексирование

Иногда неэффективно или невозможно установить отдельное соединение для каждой пары общающихся процессов, тогда нижестоящий уровень может принять решение использовать одно и то же соединение для передачи нескольких соединений несвязанных между собой. Это называется уплотнением каналов или мультиплексированием и происходит прозрачно для вышестоящих уровней.

Маршрутизация

Бывает так, что между отправителем и получателем существует несколько путей следования сообщений, в этом случае возникает проблема выбора оптимального пути. Эта задача маршрутизации.

# **4. СЛУЖБЫ И ПРОТОКОЛЫ. ТИПЫ СЛУЖБ.**

Протокол

Уровень одной машины поддерживает связь с уровнем другой машины. Правила и соглашения, используемые в данном общении, называются протоколом уровня N.

С сетевой точки зрения В протокол включаются форматы обмена данными (кадры, сообщения) и правила обмена данными (последовательность)

Протоколы могут меняться, при условии, что предоставляемые ими сервисы останутся неизменными.

Список протоколов, используемых системой, по одному протоколу на уровень, называется стеком протоколов.

Интерфейс

Между парой смежных уровней находится интерфейс(служба), определяющий набор примитивных операций, предоставляемых нижним уровнем верхнему.

Когда разработчики решают, сколько уровней включать в сеть и что должен делать каждый уровень, одной из важнейших задач является определение ясных интерфейсов между ними.

Эта задача требует, чтобы каждый уровень выполнял особый набор хорошо понятных функций. Служба описывает интерфейс между двумя уровнями, где нижний уровень является поставщиком сервиса, а верхний потребителем.

Типы служб

Уровни могут предоставлять вышестоящим уровням услуги 2-х типов: с наличием или отсутствием соединения. При наличии соединения клиент сначала устанавливает соединение, а в конце разрывает его. При отсутствии соединения клиент просто отсылает свое сообщение, в котором содержится полный адрес получателя, при этом каждое сообщение может идти своим путем и порядок получения данных не гарантируется.

Службы, ориентированные на соединение:

- Надежный поток сообщений;

- Надежный поток байт;

- Ненадежное соединение.

Службы без установки соединения:

- Ненадежная дейтаграмма;

- Дейтаграмма с подтверждением;

- Запрос-Ответ.

# **5. СЛУЖБЫ И ПРОТОКОЛЫ. ПРИМИТИВЫ СЛУЖБ.**

Протокол

Уровень одной машины поддерживает связь с уровнем другой машины. Правила и соглашения, используемые в данном общении, называются протоколом уровня N.

С сетевой точки зрения В протокол включаются форматы обмена данными (кадры, сообщения) и правила обмена данными (последовательность)

Протоколы могут меняться, при условии, что предоставляемые ими сервисы останутся неизменными.

Список протоколов, используемых системой, по одному протоколу на уровень, называется стеком протоколов.

Интерфейс

Между парой смежных уровней находится интерфейс(служба), определяющий набор примитивных операций, предоставляемых нижним уровнем верхнему.

Когда разработчики решают, сколько уровней включать в сеть и что должен делать каждый уровень, одной из важнейших задач является определение ясных интерфейсов между ними.

Эта задача требует, чтобы каждый уровень выполнял особый набор хорошо понятных функций. Служба описывает интерфейс между двумя уровнями, где нижний уровень является поставщиком сервиса, а верхний потребителем.

Примитивы служб

Служба формально описывается набором примитивов или операций, доступных пользователю или другой сущности для получения сервиса. Эти примитивы заставляют службу выполнять некоторые действия или служат ответами на действия сущности такого же уровня. Если набор протоколов входит в состав ОС, то примитивы являются системными вызовами.

Набор примитивов может быть разным и зависит от типа предоставляемого сервиса. Например, минимальный набор примитивов для простой передачи с установлением соединения будет выглядеть так:

- LISTEN (ожидание) ожидание входящего соединения

- CONNECT (соединение) установка соединения с ожидающей сущностью такого же уровня

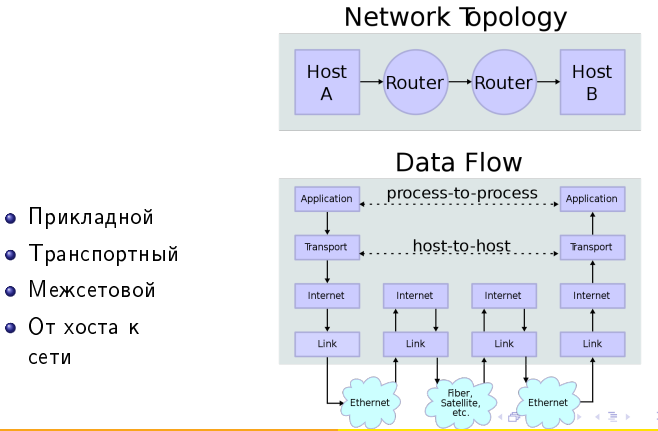
- RECEIVE (прием) ожидание входящего сообщения

- SEND (передача) отправка сообщения сущности того же уровня

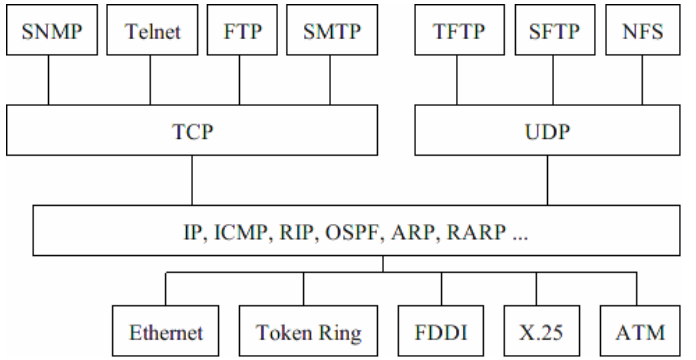
- DISCONNECT (разрыв) разрыв соединения

# **6. ЭТАЛОННАЯ МОДЕЛЬ TCP/IP.**

Топология сети



Стек TCP/IP



# **7. ПРОТОКОЛ IPV4. АДРЕСАЦИЯ IPV4.**

Межсетевой протокол IP (RFC 791):

- реализует обмен информации пакетами (IP-сегментами) (максимальный размер 65535 байт);

- является протоколом взаимодействия без установления логического соединения;

- для адресации узлов сети используется адрес длиной 4 байта;

- обеспечивает в случае необходимости фрагментацию IP-сегментов;

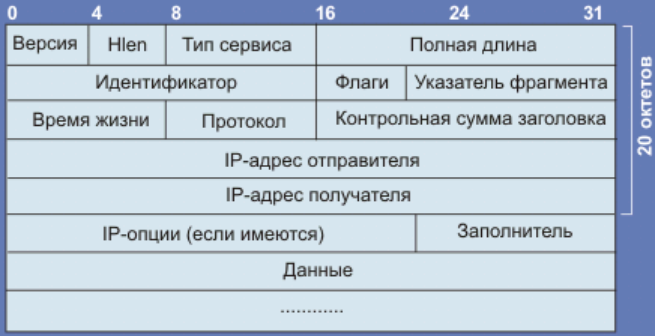
- IP-сегменты имеют конечное время жизни в сети;

- не гарантирует надежность доставки IP-сегментов адресату;

- не имеет средств управления интенсивностью передачи IP-сегментов посылающей стороной (ow control);

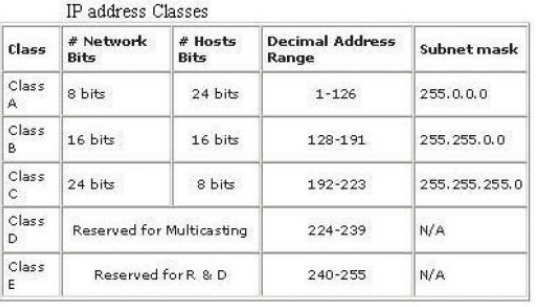
- не гарантирует правильную последовательность IP-сегментов на принимающей стороне.

Заголовок IP-сегмента

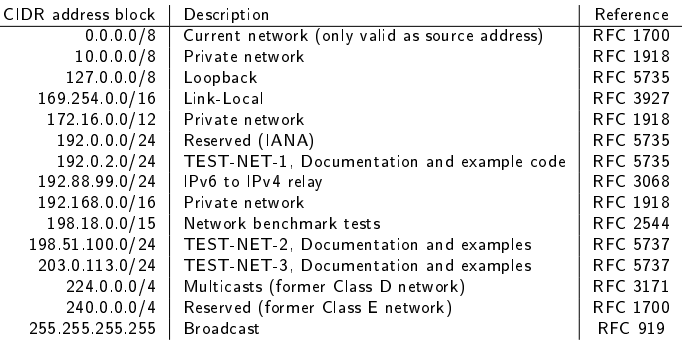


Адреса IPv4

IP-адрес представляет собой четырехбайтовое число, старшие (крайние левые) биты которого определяют класс IP-адреса.



Специальные IP адреса



# **8. ПРОТОКОЛ IPV4. ФРАГМЕНТАЦИЯ IPV4-СЕГМЕНТОВ.**

Межсетевой протокол IP (RFC 791):

- реализует обмен информации пакетами (IP-сегментами) (максимальный размер 65535 байт);

- является протоколом взаимодействия без установления логического соединения;

- для адресации узлов сети используется адрес длиной 4 байта;

- обеспечивает в случае необходимости фрагментацию IP-сегментов;

- IP-сегменты имеют конечное время жизни в сети;

- не гарантирует надежность доставки IP-сегментов адресату;

- не имеет средств управления интенсивностью передачи IP-сегментов посылающей стороной (ow control);

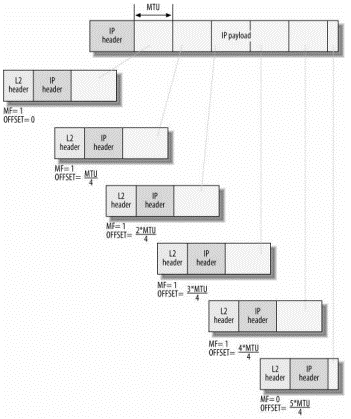
- не гарантирует правильную последовательность IP-сегментов на принимающей стороне

Фрагментация IP

Для того, чтобы существовала возможность передачи IP-сегментов через сети различного типа, межсетевой протокол обеспечивает адаптацию их размера к требованиям каждой сети. Это дает возможность, например, IP-сегментам, порожденным в сети на базе Ethernet (максимальный размер кадра 1526 байт), беспрепятственно перемещаться до адресата по сети на базе X.25 (максимальный размер кадра - 128 байт).

Изменение размера IP-сегмента в процессе перемещения по сети может быть связано и с соображениями эффективности передачи.

Каждый IP-фрагмент представляет собой полноценный IP-сегмент со своим собственным IP-заголовком. Однако заголовки всех IP-фрагментов содержат одинаковый идентификатор, совпадающий с идентификатором исходного IP-сегмента. Это позволяет распознавать все IP-фрагменты, относящиеся к одному исходному IP-сегменту.



IP-модуль на принимающем IP-фрагменты узле в ситуации, когда он должен транслировать IP-сегмент далее по сети, имеет три варианта действий с фрагментами:

- переслать IP-фрагменты далее неизменными;

- разбить (если в этом есть необходимость) полученные IP-фрагменты на более короткие IP-фрагменты;

- восстановить исходный IP-сегмент из фрагментов.

В работе с IP-фрагментами на принимающей стороне используется специальный таймер, который с приходом первого фрагмента IP-сегмента устанавливается в исходное состояние (для UNIX-реализаций это, обычно, 30 сек) и начинает обратный счет.

# **9. ХАРАКТЕРИСТИКА, НАЗНАЧЕНИЕ И ОСОБЕННОСТИ РАБОТЫ ПРОТОКОЛА ICMP.**

Протокол ICMP

Протокол передачи команд и сообщений об ошибках (ICMP internet control message protocol, RFC-792)

ICMP позволяет маршрутизатору либо конечному узлу сообщить узлу-отправителю об ошибках, с которыми маршрутизатор столкнулся при передаче какого-либо IP пакета от данного конечного узла.

ICMP только для конечных узлов

Управляющие сообщения ICMP не могут направляться промежуточному маршрутизатору, который участвовал в передаче пакета, с которым возникли проблемы, так как для такой посылки нет адресной информации - пакет несет в себе только адрес источника и адрес назначения, не фиксируя адреса промежуточных маршрутизаторов.

Протокол ICMP - это протокол сообщения об ошибках, а не протокол коррекции ошибок.

Конечный узел может предпринять некоторые действия для того, чтобы ошибка больше не возникала, но эти действия протоколом ICMP не регламентируются.

ICMP-протокол сообщает об ошибках в IP-дейтограммах, но не дает информации об ошибках в самих ICMP-сообщениях.

ICMP использует IP, а IP-протокол должен использовать ICMP.

В случае ICMP-фрагментации сообщение об ошибке будет выдано только один раз на дейтограмму, даже если ошибки были в нескольких фрагментах.

Задачи ICMP:

- передача отклика на пакет или эхо на отклик;

- контроль времени жизни дейтограмм в системе;

- реализует переадресацию пакета;

- выдает сообщения о недостижимости адресата или о некорректности параметров;

- формирует и пересылает временные метки;

- выдает запросы и отклики для адресных масок и другой информации.

ICMP-сообщения об ошибках не выдаются:

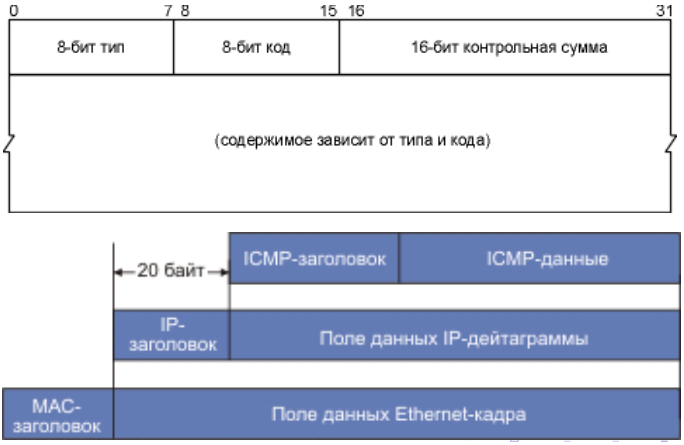
- на ICMP-сообщение об ошибке;

- при мультикастинг или широковещательной адресации;

- для фрагмента дейтограммы (кроме первого);

- для дейтограмм, чей адрес отправителя является нулевым, широковещательным или мультикастинговым.

Заголовок ICMP



# **10. ХАРАКТЕРИСТИКА ПРОТОКОЛА UDP. ПОНЯТИЕ «ПОРТ ПРОТОКОЛА».**

Порт

Порт протокола транспортного уровня – идентифицируемый номером системный ресурс, выделяемый приложению, выполняемому на некотором сетевом хосте, для связи с приложениями, выполняемыми на других сетевых хостах (в том числе c другими приложениями на этом же хосте).

Порты необходимы по двум причинам:

- используются как идентификатор для разделения транспортных сессий между одинаковыми оконечными точками

- используются для идентификации протокола прикладного уровня и ассоциированного с ним сервиса

1 порт = 16 бит

Для каждого из протоколов TCP и UDP стандарт определяет возможность одновременного выделения на хосте до 65536 уникальных портов, идентифицирующихся номерами от 0 до 65535.

Порты TCP не пересекаются с портами UDP. То есть, порт 1234 протокола TCP не будет мешать обмену по UDP через порт 1234!!!

Диапазоны портов:

- Системные (0-1023)

- Пользовательские (1024-49151)

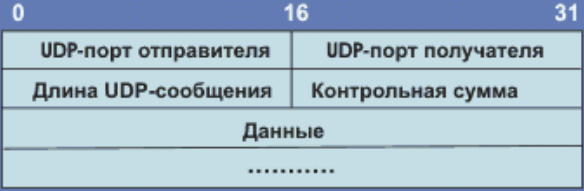
- Динамические и/или частные (49152-65535)

Для большинства ОС системный диапазон доступен только с привилегиями администратора!

Протокол UDP (RFC-768) является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP.

Протокол UDP обеспечивает доставку дейтограмм, но не требует подтверждения их получения. Протокол UDP не требует соединения с удаленным модулем UDP («бессвязный» протокол).

Формат дейтаграммы UDP

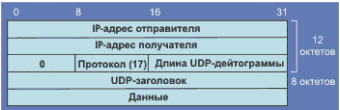


Длина включает в себя заголовок.

Контрольная сумма рассчитывается с использованием псевдозаголовка!

Контрольная сумма

Контрольная сумма с использованием псевдозаголовка необходима как защита от дейтаграмм, ошибочно направленных по другому адресу.



Контрольная сумма UDP не является обязательной. Если она не подсчитывается, ее значение равно 0 (настоящая нулевая контрольная сумма кодируется всеми единицами).

# **11. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. ПОНЯТИЕ «ПОРТ ПРОТОКОЛА».**

Порт

Порт протокола транспортного уровня – идентифицируемый номером системный ресурс, выделяемый приложению, выполняемому на некотором сетевом хосте, для связи с приложениями, выполняемыми на других сетевых хостах (в том числе c другими приложениями на этом же хосте).

Порты необходимы по двум причинам:

- используются как идентификатор для разделения транспортных сессий между одинаковыми оконечными точками

- используются для идентификации протокола прикладного уровня и ассоциированного с ним сервиса

Для каждого из протоколов TCP и UDP стандарт определяет возможность одновременного выделения на хосте до 65536 уникальных портов, идентифицирующихся номерами от 0 до 65535.

Порты TCP не пересекаются с портами UDP. То есть, порт 1234 протокола TCP не будет мешать обмену по UDP через порт 1234!!!

Диапазоны портов:

- Системные (0-1023)

- Пользовательские (1024-49151)

- Динамические и/или частные (49152-65535)

Для большинства ОС системный диапазон доступен только с привилегиями администратора!

TCP RFC-793

TCP Transmission Control Protocol – надежный протокол с установлением соединений, позволяющий без ошибок доставлять байтовый поток с одной машины на любую другую машину объединенной сети. Он разбивает входной поток байтов на отдельные сообщения и передает их межсетевому уровню. В пункте назначения этот протокол собирает из полученных сообщений выходной поток. Кроме того, TCP осуществляет управление потоком.

Механизмы надежности TCP

Контрольная сумма позволяет определить повреждение данных и/или заголовка TCP-сегмента.

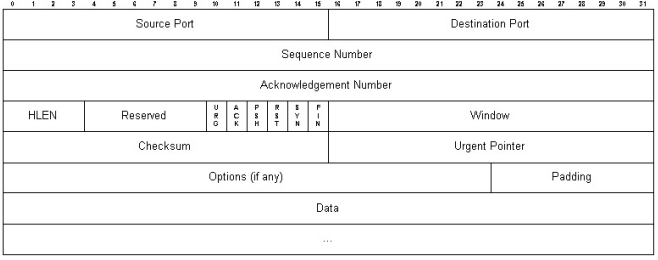
Определение дублирующихся сегментов – «дубли» выкидываются.

Повторная передача используются подтверждения о доставке данных. Отсутствие подтверждений вместе с механизмом таймеров приводит к повторной передаче данных.

Корректная последовательность TCP собирает сегменты в правильном порядке.

Таймеры позволяют управлять повторной передачей сегментов.

Формат пакета TCP



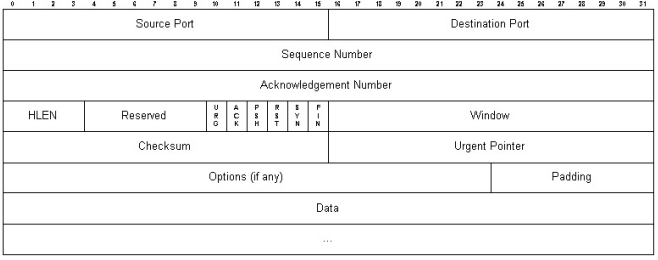
HLEN длина заголовка в 32-хбитных словах. В RFC обозначается, как "data offset"

window указывает отправителю, сколько данных получатель готов принять

Urgent pointer указывает смещение октета, следующего за срочными данными, относительно первого октета

# **12. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. НАЗНАЧЕНИЕ ФЛАГОВ В TCP-СЕГМЕНТЕ.**

Формат пакета TCP



HLEN длина заголовка в 32-хбитных словах. В RFC обозначается, как "data offset"

window указывает отправителю, сколько данных получатель готов принять

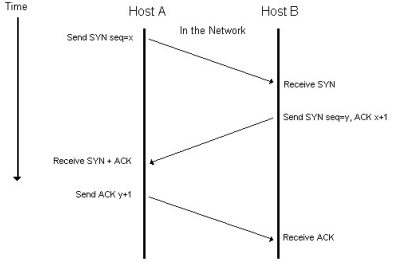
Urgent pointer указывает смещение октета, следующего за срочными данными, относительно первого октета

Поле Flags

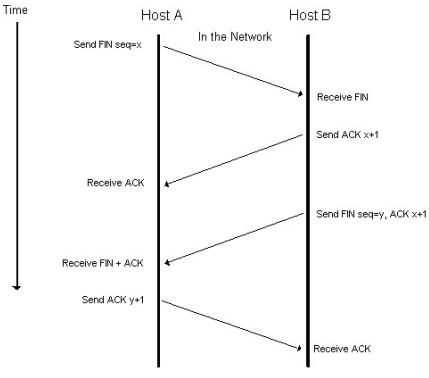
* Urgent Pointer (URG) – флаг значимости указателя на экстренные данные;
* Acknowledgement (ACK) – флаг значимости подтверждающего номера;
* Push Function (PSH) – флаг принудительной доставки данных (без буферизации);
* Reset the Connection (RST) – флаг разрыва соединения;
* Synchronize (SYN) – флаг синхронизации последовательных номеров;
* No More Data from Sender (FIN) – флаг последних данных.

# **13. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. УСТАНОВЛЕНИЕ И РАЗРЫВ СОЕДИНЕНИЯ TCP (НА УРОВНЕ ПРОТОКОЛА).**

Установка соединения TCP

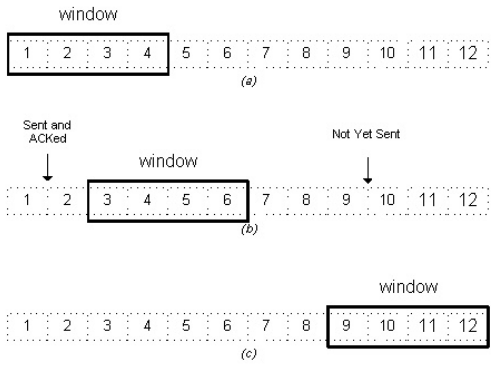


Разрыв соединения TCP



# **14. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. УПРАВЛЕНИЕ ПОТОКОМ. СКОЛЬЗЯЩЕЕ ОКНО. МЕХАНИЗМ МЕДЛЕННОГО СТАРТА.**

Управление потоком с помощью скользящего окна



Управление насыщением механизм медленного старта

Congestion Control:

- Slow Start с каждой успешной передачей размер буфера удваивается до половины точки насыщения далее по 1 сегменту. Вплоть до размера окна.

- Congestion Avoidance противовес медленному старту. Уменьшает размер окна в 2 раза, но не менее 2 сегментов. В случае таймаута уменьшается до 1 сегмента и запускается механизм медленного старта.

# **15. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. АЛГОРИТМ НЭГЛА (NAGLE) И СИНДРОМ УЗКОГО ОКНА.**

Алгоритм Нэгла(Nagle) – Борется с неэффективной передачей.

if есть новые данные для отправки

if размер окна >= MSS and доступные данные is >= MSS

отправить полный сегмент MSS сейчас

else

if все еще есть неподтвержденные данные

помещать данные в буфер пока не будет получено подтверждение

else

отправить данные незамедлительно

end if

end if

end if

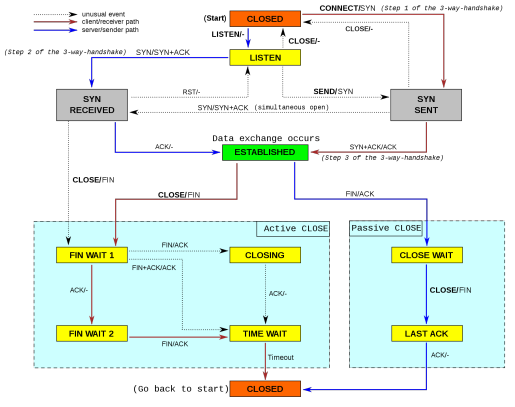
Синдром узкого окна

Такого рода проблема возникает в том случае, когда данные поступают отправителю крупными блоками, а интерактивное приложение адресата считывает информацию побайтно.

Кларк предложил не посылать уведомление о ненулевом значении ширины окна при считывании одного байта, а лишь после освобождения достаточно большого пространства в буфере. Например, когда адресат готов принять MSS байтов или когда буфер наполовину пуст.

# **16. ХАРАКТЕРИСТИКА ПРОТОКОЛА TCP. ДИАГРАММА СОСТОЯНИЙ TCP.**

Диаграмма состояний TCP



# **17. ПОНЯТИЕ ИНТЕРФЕЙСА СОКЕТОВ. ФУНКЦИОНАЛЬНЫЕ СРЕДСТВА ИНТЕРФЕЙСА.**

В стандартах TCP/IP не даны подробные сведения о том, каким образом прикладное ПО должно взаимодействовать с ПО протоколов TCP/IP;

в них описаны только необходимые функциональные средства, а возможность определять конкретные требования к реализации API-интерфейса предоставлена системным проектировщикам.

Интерфейс между TCP/IP и приложениями, в которых используются эти протоколы, определены неформально, в виде рекомендаций, а не требований.

Неформальная спецификация

К преимуществам относятся гибкость и широкая применяемость, что позволяет проектировщикам реализовывать протоколы TCP/IP для любых ОС. Также проектировщики имеют возможность использовать организацию интерфейса, наиболее подходящую для операционной системы (например, процедурную или основанную на передаче сообщений).

Недостатком неформальной спецификации является то, что ее применение может привести к появлению различий в отдельных деталях реализации интерфейса для каждой ОС.

Сокеты Berkeley

Разработчики решили применять уже существующие системные вызовы и вводить новые только для поддержки функций TCP/IP, которые нельзя легко включить в существующий набор функций. Результаты этого проекта получили широкую известность под названием API-интерфейса сокетов или просто интерфейса сокетов, а разработанная система стала называться Berkeley UNIX или просто UNIX . Протокол ТСР впервые появился в выпуске 4.1 дистрибутива BSD (Berkeley Software Distribution).

На практике существует лишь небольшое количество API-интерфейсов, позволяющих использовать протоколы TCP/IP во всех приложениях. На данный момент получили наиболее широкое распространение 3 интерфейса:

- сокеты Беркли (сокеты, интерфейс сокетов), разработанный в ун-те Беркли;

- Windows Sockets, разработанный в компании Microsoft;

- TLI (Transport Layer Interface), разработанный AT&T для системы Unix версии System V.

Функциональные средства интерфейса:

- распределение локальных ресурсов для связи;

- задание локальной и удаленной оконечных точек связи;

- инициирование соединения (клиент);

- передача дейтаграммы (клиент);

- ожидание входящего запроса на установление соединения (сервер);

- передача или прием данных;

- определение момента поступления данных;

- выработка срочных данных;

- обработка входящих срочных данных;

- корректное завершение соединения;

- обработка запроса на завершение соединения от удаленного участника соединения;

- аварийное прекращение связи;

- устранение последствий аварийных ситуаций или аварийного прекращения связи;

- освобождение локальных ресурсов после завершения связи.

# **18. ПОНЯТИЕ СОКЕТА. ТИПЫ И СТРУКТУРЫ СОКЕТОВ. СЕМЕЙСТВА АДРЕСОВ.**

Сокет

В API-интерфейсе сокетов реализовано новое абстрактное понятие для сетевой связи сокет. Как и файл, сокет обозначается целым числом, называемым дескриптором сокета. Операционная система размещает дескрипторы сокетов в той же таблице дескрипторов, что и дескрипторы файлов.

Поэтому в приложении не может присутствовать и дескриптор файлов и дескриптор сокетов с одним и тем же значением.

Сокет может применяться для любой связи. Поэтому приложение должно указывать, как он будет использоваться. В частности необходимо указать номера портов протокола и адреса локального и удаленного компьютеров.

На практике существует лишь небольшое количество API-интерфейсов, позволяющих использовать протоколы TCP/IP во всех приложениях. На данный момент получили наиболее широкое распространение 3 интерфейса:

- сокеты Беркли (сокеты, интерфейс сокетов), разработанный в ун-те Беркли;

- Windows Sockets, разработанный в компании Microsoft;

- TLI (Transport Layer Interface), разработанный AT&T для системы Unix версии System V.

Сокеты Berkeley

Разработчики решили применять уже существующие системные вызовы и вводить новые только для поддержки функций TCP/IP, которые нельзя легко включить в существующий набор функций. Результаты этого проекта получили широкую известность под названием API-интерфейса сокетов или просто интерфейса сокетов, а разработанная система стала называться Berkeley UNIX или просто UNIX . Протокол ТСР впервые появился в выпуске 4.1 дистрибутива BSD (Berkeley Software Distribution).

Проектировщики из BSD предусмотрели возможность применения различных семейств протоколов связи, далеко выходящую за рамки протоколов TCP/IP.

Типы сокетов

В пределах каждого адресного семейства могут существовать сокеты нескольких типов. В стандарте POSIX-2001 их четыре:

- SOCK\_STREAM Сокеты данного типа поддерживают надежные, упорядоченные, полнодуплексные потоки октетов в режиме с установлением соединения.

- SOCK\_SEQPACKET Аналог SOCK\_STREAM с дополнительным сохранением границ между записями.

- SOCK\_DGRAM Передача данных в виде датаграмм в режиме без установления соединения.

- SOCK\_RAW Аналог SOCK\_DGRAM с дополнительной возможностью доступа к протокольным заголовкам и другой информации нижнего уровня. Также известны, как неструктурированные или сырые сокеты.

Семейства адресов

Адресное семейство соответствует конкретной среде взаимодействия. Стандарт POSIX-2001 определяет три таких семейства:

- AF\_UNIX Адресное семейство UNIX поддерживает межпроцессное взаимодействие в пределах одной системы.

- AF\_INET Адресное семейство, поддерживающее взаимодействие по протоколам IPv4.

- AF\_INET6 Взаимодействие по протоколам IPv6 (необязательная возможность).

# **19. СОКЕТЫ BERKELEY. РАЗРЕШЕНИЕ ИМЕН ХОСТОВ, ПРОТОКОЛОВ, СЕТЕВЫХ СЕРВИСОВ.**

Получение имен хостов

Структура hostent определена в файле netdb.h:

struct hostent

{

char \*h\_name; // официальное имя хоста

char \*\*h\_aliases; // массив псевдонимов хоста

int h\_addrtype; // тип адреса (на данный момент AF\_INET или AF\_INET6)

int h\_length; // длина адреса в байтах

char \*\*h\_addr\_list; // массив указателей на сетевые адреса хоста

}

Пример содержимого файла /etc/hosts

127.0.0.1 alien.home localhost.localdomain localhost

::1 alien.home localhost6.localdomain6 localhost6

Доступ к базе данных сетевых протоколов

Локальная база данных сетевых протоколов располагается по адресу /etc/protocols, а содержимое этого файла выглядит следующим образом:

ip 0 IP # internet protocol, pseudo protocol number

icmp 1 ICMP # internet control message protocol

igmp 2 IGMP # internet group management protocol

tcp 6 TCP # transmission control protocol

udp 17 UDP # user datagram protocol

Структура protoent содержит по крайней мере следующие поля:

- char \*p\_name официальное имя протокола;

- char \*\*p\_aliases массив указателей на альтернативные имена протокола, завершаемый пустым указателем;

- int p\_proto номер протокола.

Доступ к базе данных сетевых сервисов

Локально база данных сетевых сервисов располагается по адресу /etc/services, а содержимое этого файла выглядит следующим образом:

# service-name port/protocol [aliases \ldots] [# comment]

domain 53/tcp # Domain Name Server

domain 53/udp # Domain Name Server

bootps 67/tcp # Bootstrap Protocol Server

bootps 67/udp # Bootstrap Protocol Server

bootpc 68/tcp # Bootstrap Protocol Client

bootpc 68/udp # Bootstrap Protocol Client

Структура servent содержит следующие поля:

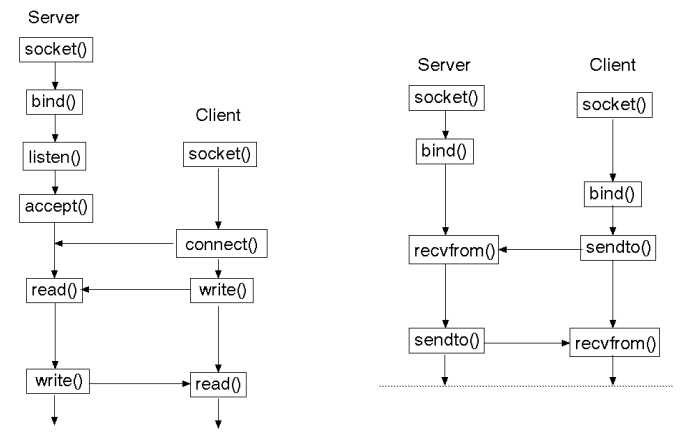
char \*s\_name Официальное имя сервиса

char \*\*s\_aliases Массив указателей на альтернативные имена сервиса, завершаемый пустым указателем

int s\_port Номер порта, соответствующий сервису (в сетевом порядке байт)

char \*s\_proto Имя протокола для взаимодействия с сервисом

# **20. СОКЕТЫ BERKELEY. ОСНОВНЫЕ СИСТЕМНЫЕ ВЫЗОВЫ.**



# **21. СОКЕТЫ BERKELEY. ОПЦИИ СОКЕТОВ. УРОВНИ ОПЦИЙ.**

Опции сокетов

С сокетами могут быть ассоциированы опции, влияющие на их функционирование. Значения этих опций можно опросить или изменить с помощью функций getsockopt и setsockopt.

int getsockopt (int sd, int level,

int option\_name,

void \*restrict option\_value,

socklen\_t \*restrict option\_len);

int setsockopt (int sd, int level,

int option\_name,

const void \*option\_value,

socklen\_t option\_len);

Уровни опций

level указывает к какому уровню относится опция

- SOL\_SOCKET опции сокета

- SOL\_TCP опции для протокола TCP

- SOL\_IP опции для протокола IP

SOL\_SOCKET

SO\_ERROR (int) Статус ошибок (после опроса очищается)

SO\_TYPE (int) Тип сокета

SO\_PROTOCOL (int) Протокол

SO\_DEBUG (bool) Отладочная информация о работе сокета.

SO\_ACCEPTCONN (bool) Указывает, является ли сокет слушающим.

SO\_SNDBUF (int) Размер буфера для передаваемых данных (выходного буфера)

SO\_RCVBUF (int) Размер входного буфера.

SO\_RCVLOWATM (int) Минимальное число байт, обрабатываемых при вводе.

SO\_SNDLOWAT (int) Минимальное число байт, обрабатываемых при выводе.

SO\_RCVTIMEO (timeval) Длительность ожидания поступления данных при вводе.

SO\_SNDTIMEO (timeval) Длительность ожидания отправки данных при выводе.

SO\_TIMESTAMP (bool) Включить передачу отметок времени

SO\_BROADCAST (bool) Переводит сокет в широковещательный режим передачи данных

SO\_OOBINLINE (bool) Если установлена, то данные out-of-band помещаются в «стандартный» поток приема

SO\_REUSEADDR (bool) Использование «занятого» адреса (для bind)

SOL\_TCP

TCP\_NODELAY (bool) Отключает алгоритм Нэгла (Nagle)

TCP\_MAXSEG (int) устанавливает или сообщает макс. размер сегмента для исх. TCP-пакетов

TCP\_CORK (bool) При включении этой опции, перестают отсылаться частичные кадры.

SOL\_IP

IP\_HDRINCL (bool) Включение этого флага означает, что пользователь уже добавил заголовок IP в начало своих данных.

IP\_OPTIONS Устанавливает или возвращает те опции IP, которые посылаются с каждым пакетом из данного сокета. Аргументами являются указатель на область памяти, содержащую эти опции, и размер опции

IP\_TTL (byte) Устанавливает или получает текущее значение поля TTL

IP\_TOS (byte) Устанавливает или получает значение поля Тип-Сервиса

IP\_PMTU\_DISCOVER (int) Устанавливает или возвращает значение опции Path MTU Discovery (Обнаружение MTU Маршрута) установленной для сокета

IP\_MTU (int) Возвращает используемое в данный момент значение MTU маршрута текущего сокета

# **22. МОДЕЛЬ СЕТЕВОГО ВЗАИМОДЕЙСТВИЯ КЛИЕНТ-СЕРВЕР.**

Архитектура

Термин «клиент-сервер» означает такую архитектуру программного комплекса, в которой его функциональные части взаимодействуют по схеме «запрос-ответ».

Клиент или сервер?

С сетевой точки зрения

Если рассмотреть две взаимодействующие части этого комплекса, то одна из них (клиент) выполняет активную функцию, т. е. инициирует запросы, а другая (сервер) пассивно на них отвечает.

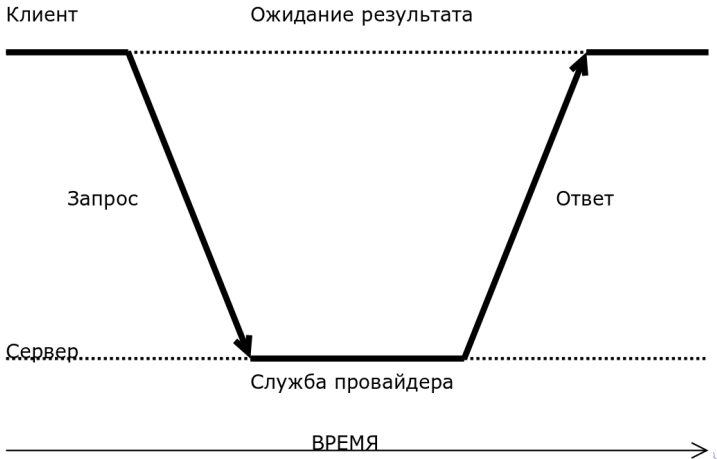
По мере развития системы роли могут меняться, например, некоторый программный блок будет одновременно выполнять функции сервера по отношению к одному блоку и клиента по отношению к другому.

С функциональной точки зрения

Процессы, реализующие некоторую службу, например, ФС или БД называются серверами.

Процессы, запрашивающие службы у серверов службы путем пересылки запроса и последующего ожидания ответа от сервера, называются клиентами.

Сетевое взаимодействие



Модель?

Модель клиент-сервер всегда была предметом множества дебатов и споров. Один из главных вопросов состоял в том, как точно разделить клиента и сервер.

Четкого разделения здесь не может быть.

# **23. ТРЕХУРОВНЕВАЯ КЛИЕНТ-СЕРВЕРНАЯ МОДЕЛЬ, ВАРИАНТЫ РАСПРЕДЕЛЕНИЯ НА ФИЗИЧЕСКИ ДВУХЗВЕННОЙ АРХИТЕКТУРЕ. ВЕРТИКАЛЬНОЕ, ГОРИЗОНТАЛЬНОЕ И PEER-TO-PEER РАСПРЕДЕЛЕНИЕ.**

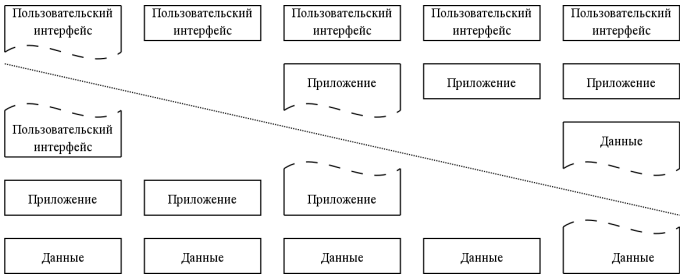
Трехуровневая модель

- уровень пользовательского интерфейса (Уровень пользовательского интерфейса обычно реализуется на клиентах. Этот уровень содержит программы, посредством которых пользователь может взаимодействовать с приложением. Простейший вариант пользовательского интерфейса не содержит ничего, кроме символьного (или графического) дисплея.)

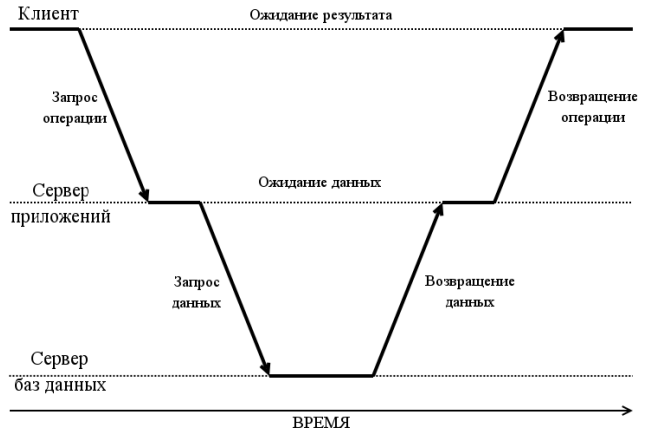
- уровень обработки (На этом уровне трудно выделить какие-то закономерности, обычно здесь реализуется логика работы программы.)

- уровень данных (На этом уровне находятся программы, которые поставляют данные обрабатывающим их приложениям. В простейшем варианте уровень данных реализуется ФС, однако часто может использоваться и полнофункциональная БД. В модели клиент-сервер этот уровень обычно находится на стороне сервера. Специфическим требованием этого уровня является требование сохранности это означает, что, когда приложение не работает, данные должны сохраняться в определенном месте в расчете на дальнейшее использование.)

Физически двухзвенная архитектура



Физически трехзвенная архитектура



Вертикальное распределение

Во множестве приложений обработка данных организована как многозвенная архитектура приложений клиент-сервер.

Особенностью такого типа является то, что он о достигается размещением логически разных компонентов на различных физических машинах.

Горизонтальное распределение

При таком типе распределения клиент или сервер может содержать физически разделенные части логически однородного модуля, причем работа с каждой из частей может протекать независимо. Это делается для выравнивания нагрузки.

peer-to-peer

Для некоторых несложных приложений выделенного сервера может не быть вообще. Такую организацию обычно называют одноранговым распределением.

Модель взаимодействия типа клиент-сервер и проектирование ПО

С точки зрения прикладного программиста, протоколы TCP/IP, как и большинство других протоколов компьютерной связи, просто предоставляет основные механизмы передачи данных. В частности, протоколы TCP/IP, позволяют устанавливать соединение между 2-мя прикладными программами и передавать данные в прямом и обратном направлении.

Поэтому принято считать, что протоколы TCP/IP обеспечивают одноранговую или прямую связь.

Основная модель сетевого взаимодействия

Необходимость применения принципа организации взаимодействия типа клиент-сервер связана с решением проблемы согласования условий соединения.

Модель взаимодействия типа клиент-сервер предусматривает согласование условий соединения наиболее простым способом:

она требует, чтобы для взаимодействия любой пары приложений, один из участников приступал к работе заранее и ждал (возможно в течение неопределенного времени) до тех пор, пока к нему не обратится второй участник соединения.

# **24. АРХИТЕКТУРА КЛИЕНТА. ОПРЕДЕЛЕНИЕ МЕСТОНАХОЖДЕНИЯ СЕРВЕРА.**

Приложения, действующие в качестве клиентов, концептуально проще приложений выполняющих функции серверов.

- основной части клиентского программного обеспечения не нужно взаимодействовать с несколькими серверами

- основная часть клиентского программного обеспечения применяется в виде обычных прикладных программ не требуются привилегии

- большая часть клиентского обеспечения не заботится о правилах защиты, поскольку в этом вопросе оно полагается на ОС

Определение местонахождения сервера первейшая и одна из самых важных задач клиентского программного обеспечения!!!

- доменное имя или IP-адрес сервера могут быть заданы в виде константы во время трансляции клиентской программы

- клиентская программа может требовать у пользователя указывать имя сервера при ее вызове

- информация о местонахождении сервера предоставляется из постоянного хранилища данных

- для поиска сервера используется отдельный протокол

Еще один аспект поиска сервера следует из того, какой тип сервиса предоставляется сервером.

От этого зависит нужно ли использовать какой-либо определенный сервер или же можно использовать первый ответивший.

В качестве альтернативы в ответе сервера может также содержаться список других серверов.

# **25. АРХИТЕКТУРА КЛИЕНТА. АЛГОРИТМЫ РАБОТЫ КЛИЕНТОВ. ФУНКЦИЯ CONNECT ДЛЯ UDP.**

Алгоритм клиентов с установлением логического соединения

Задача построения клиента с использованием протокола TCP является самой простой из всех задач сетевого программирования.

Алгоритм клиента TCP:

1. Найти IP-адрес и номер порта протокола сервера, с которым необходимо установить связь.

2. Распределить сокет

3. Указать, что для соединения нужен произвольный, неиспользуемый порт протокола на локальном компьютере и позволить ПО TCP выбрать такой порт

4. Подключить сокет к серверу

5. Выполнить обмен данными с сервером по протоколу прикладного уровня

6. Закрыть соединение

Алгоритм клиентов без установления логического соединения

В этом алгоритме проблема надежности игнорируется.

1. Найти IP-адрес и номер порта протокола сервера, с которым необходимо установить связь.

2. Распределить сокет

3. Указать, что для соединения нужен произвольный, неиспользуемый порт протокола на локальном компьютере и позволить ПО UDP выбрать такой порт

4. Указать сервер, на который должны передаваться сообщения.

5. Выполнить обмен данными с сервером по протоколу прикладного уровня

6. Закрыть сокет

Подключенный и неподключенный режимы

В клиентском приложении сокет UDP может использоваться в одном из 2-х основных режимов: подключенном и неподключенном.

Подключенные сокеты удобно применять для взаимодействия с конкретным сервером, а неподключенные если адресат может меняться, в этом случае нужно для каждого сообщения указывать адрес сервера.

Если используется подключение, то вызов функции connect только записывает информацию об удаленной точке в дескриптор сокета. Даже если вызов успешен, это еще не означает, что адрес удаленной точки действителен или что сервер является достижимым.

# **26. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ ПОСЛЕДОВАТЕЛЬНОГО СЕРВЕРА С УСТАНОВЛЕНИЕМ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ.**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Последовательный сервер с установлением логического соединения

Используется в службах, требующих незначительного времени для обработки каждого запроса, однако требующих надежного протокола доставки сообщений. За счет больших издержек на установку и завершение соединения среднее время отклика часто значительно выше, чем у последовательного сервера без установления логического соединения.

Алгоритм последовательного сервера с установлением логического соединения

1. Создать сокет и привязать его к общепринятому адресу службы

2. Перевести сокет в пассивный режим

3. Принять из сокета запрос на установление соединения и получить новый сокет для установления соединения

4. Считывать в цикле запросы от клиента, формировать ответы и отправлять их клиенту, в соответствии с прикладным протоколом

5. После завершения обмена данными с клиентом закрыть соединение и перейти к 3 пункту

# **27. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ ПОСЛЕДОВАТЕЛЬНОГО СЕРВЕРА БЕЗ УСТАНОВЛЕНИЯ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ.**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Последовательный сервер без установления логического соединения

Используется в службах, требующих незначительного времени для обработки каждого запроса

Алгоритм последовательного сервера без установления логического соединения

1. Создать сокет и привязать его к общепринятому адресу службы

2. Считывать в цикле запросы от клиента, формировать ответы и отправлять их клиенту, в соответствии с прикладным протоколом

# **28. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ ПАРАЛЛЕЛЬНОГО СЕРВЕРА БЕЗ УСТАНОВЛЕНИЯ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ.**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Параллельный сервер без установления логического соединения

Редко применяемый тип сервера.

Во многих случаях затраты на создание потоков или процессов не оправдывают повышения эффективности, достигнутого за счет параллелизма.

Алгоритм параллельного сервера без установления логического соединения

Ведущий поток:

1. Создать сокет и привязать его к общепринятому адресу службы

2. В цикле считывать запросы с помощью recvfrom и создавать новые ведомые потоки (процессы) для формирования ответа

Ведомый поток:

1. Работа потока начинается с получения конкретного запроса от ведущего потока, а также доступа к сокету

2. Сформировать ответ согласно прикладному протоколу и отправить его клиенту с использованием функции sendto

3. Завершить работу потока

# **29. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ ПАРАЛЛЕЛЬНОГО СЕРВЕРА С УСТАНОВЛЕНИЕМ ЛОГИЧЕСКОГО СОЕДИНЕНИЯ.**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Параллельный сервер с установлением логического соединения

Наиболее распространенный тип сервера, поскольку сочетает надежный протокол (подходит и для глобальных сетей) с возможностью одновременной работы с несколькими клиентами.

Алгоритм параллельного сервера с установлением логического соединения

Ведущий поток:

1. Создать сокет и привязать его к общепринятому адресу службы

2. Перевести сокет в пассивный режим

3. Вызвать в цикле функцию accept для получения очередного запроса от клиента и создавать новый ведомый поток или процесс для формирования ответа

Ведомый поток:

1. Работа начинается с получения доступа к соединению, полученному от ведущего потока

2. Выполнять в цикле работу с клиентом через соединение

3. Закрыть соединение и завершить работу.

# **30. КЛАССИФИКАЦИЯ СЕРВЕРОВ. АЛГОРИТМ СЕРВЕРА С АСИНХРОННЫМ ВВОДОМ-ВЫВОДОМ.**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Алгоритм сервера с асинхронным вводом/выводом

1. Создать сокет и привязать его к общепринятому адресу службы. Добавить сокет к списку сокетов, через которые может осуществляться ввод/вывод

2. Использовать функцию select для получения информации о готовности существующих сокетов к вводу/выводу

3. Если готов первоначальный сокет, то использовать функцию accept для получения очередного запроса на установление соединения и добавить новый сокет к списку сокетов, через которые может осуществляться ввод/вывод

4. Если готов сокет, отличный от первоначального, использовать функцию recv для получения очередного запроса, сформировать ответ и передать ответ клиенту с использованием функции send

5. Перейти к пункту 2

# **31. КЛАССИФИКАЦИЯ СЕРВЕРОВ. СЕРВЕРЫ С ПРЕДВАРИТЕЛЬНЫМ ПОРОЖДЕНИЕМ ПОТОКОВ.**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Альтернативное устройство параллельного сервера

- предварительное создание дочерних процессов (preforking)

- предварительное создание потоков (prethreading)

- мультисервисные серверы (суперсервер)

prethreaded server

- сервер с предварительным созданием потоков с использованием взаимного исключения для защиты accept

- сервер с предварительным созданием потоков, главный поток вызывает accept

# **32. КЛАССИФИКАЦИЯ СЕРВЕРОВ. СЕРВЕРЫ С ПРЕДВАРИТЕЛЬНЫМ ПОРОЖДЕНИЕМ ПРОЦЕССОВ.**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Альтернативное устройство параллельного сервера

- предварительное создание дочерних процессов (preforking)

- предварительное создание потоков (prethreading)

- мультисервисные серверы (суперсервер)

preforked server

- сервер с предварительным созданием дочерних процессов с параллельным вызовом accept

- сервер с предварительным созданием дочерних процессов с блокировкой для защиты accept

+ с помощью файла

+ с помощью взаимного исключения (семафор, мьютекс, критические секции windows)

- сервер с предварительным созданием дочерних процессов с использованием взаимного исключения для защиты accept

- сервер с предварительным созданием дочерних процессов с последующей передачей дескриптора сокета дочерним процессам

# **33. КЛАССИФИКАЦИЯ СЕРВЕРОВ. МУЛЬТИСЕРВИСНЫЕ СЕРВЕРЫ (НА ПРИМЕРЕ XINETD).**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Альтернативное устройство параллельного сервера

- предварительное создание дочерних процессов (preforking)

- предварительное создание потоков (prethreading)

- мультисервисные серверы (суперсервер)

В суперсервере используется не отдельный сервер, а "враппер" для сервисов.

- уменьшение надежности

- ограничения по количеству открытых сокетов

- потребляет меньше ресурсов

Мультисервисный сервер без установления соединения

1. Сервер открывает набор сокетов UDP и привязывает к портам служб.

2. Используется таблица соответствия сокетов службам

3. с помощью select сервер переводится в состояние ожидания дейтаграммы

4. для обработки каждой отдельной дейтаграммы вызывается соответствующая функция

Мультисервисный сервер с установлением соединения

1. Сервер открывает набор сокетов TCP и привязывает к портам служб.

2. Используется таблица соответствия сокетов службам

3. с помощью select сокеты переводится в состояние ожидания нового соединения

4. Если готов один из первоначальных сокетов, то создаем новый сокет.

5. Для обработки каждого соединения вызывается соответствующая функция

Модульный мультисервисный сервер с установлением соединения

1. Сервер открывает набор сокетов TCP и привязывает их к портам служб.

2. Используется таблица соответствия сокетов службам

3. с помощью select сокеты переводится в состояние ожидания нового соединения

4. После поступления запроса вызываем fork для создания ведомого процесса

5. ведомый процесс закрывает все ненужные сокеты

6. ведомый процесс производит замещение процесса с помощью вызова из семейства exec

Работа через initd/xinitd

// Если работаем с помощью xinetd, то перенаправляем stdout и stderr в сокет)

if( xinetd){

close( stdout); close( stderr);

if ( dup2( 0, 1) == -1) return 1;

if ( dup2( 0, 2) == -1) return 1;}

# **34. КЛАССИФИКАЦИЯ СЕРВЕРОВ. МЕТОДЫ УЛУЧШЕНИЯ ФУНКЦИОНИРОВАНИЯ СЕРВЕРОВ.**

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:

в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Классификация серверов

По внутренней архитектуре:

- Последовательный

- Параллельный

По типу используемого сервиса:

- С установлением соединения

- Без установления соединения

По состоянию:

- С сохранением состояния (stateful)

- Без сохранения состояния (stateless)

Общие рекомендации по работе сервера

- Функционирование в фоновом режиме:

с помощью fork запускаем копию программы, а родительский процесс убивается. При этом должны закрываться все унаследованные дескрипторы файлов.

- Сервер должен отключать управляющий терминал, чтобы не получать от него сигналы:

fd=open(/dev/tty, O\_RDWR);

ioctl( fd, TIOCNOTTY, 0);

close( fd);

- Сервер должен переходить в безопасный каталог используя chdir

- Необходимо изменить маску по умолчанию для создания файлов используя umask( 027)

- Необходимо открыть стандартные дескрипторы для корректной работы библиотечных процедур:

fd=open(/dev/null, O\_RDWR); //ввод

dup( fd); //вывод

dup( fd); //ошибки

- Сервер должен предотвращать запуск нескольких копий (лок-файлы или др. системные функции)

- Желательно игнорировать сигналы, не относящиеся к работе сервера

- Если есть возможность, то желательно использовать системные методы ведения журнала событий

# **35. ТЕКСТОВОЕ И БИНАРНОЕ ПРЕДСТАВЛЕНИЕ ДАННЫХ В ПРОТОКОЛАХ.**

Представление данных

Ясные протоколы лежат в основе хорошей практики.

Схожесть задач

- Проектирование форматов файлов для сохранения данных приложений в постоянном хранилище данных.

- Проектирование протоколов прикладного уровня для передачи (в том числе и через сеть) данных и команд между взаимодействующими программами.

Важное при проектировании прикладных протоколов

- способность к взаимодействию;

- прозрачность;

- расширяемость;

- экономичность транзакций.

Типы данных

С машинной точки зрения удобно хранить данные сложной структуры в бинарном виде все поля имеют характерный для конкретной машины формат и указатели в виде адресов.

Однако такие формы представления слабо подходят для длительного хранения и передачи.

Минусы сериализации данных (структура в байтовый поток) при использовании в протоколах

- проблемы взаимодействия между машинами с разной архитектурой;

- непрозрачность для других средств;

- для сетевых протоколов часто целесообразно представлять структуру данных не в виде одного большого двоичного объекта, а в виде последовательности транзакций или сообщений, которые могут быть отклонены принимающей машиной.

Когда выгодно использовать бинарное представление?

- Необходимость передачи больших блоков данных и разработчик формата действительно позаботился о достижении максимальной плотности полезной информации в потоке. Пример: мультимедиа форматы.

- Существует жесткое ограничение времени и/или инструкций, необходимых для интерпретации данных. Пример: протоколы сетевого уровня.

Текстовое представление данных

Дуг Макилрой (изобретатель pipes):

- Будьте готовы к тому, что вывод каждой программы станет вводом другой, еще неизвестной программы.

- Не загромождайте вывод посторонней информацией.

- Избегайте строгих табличных или двоичных форматов ввода.

- Не настаивайте на интерактивном вводе.

Текстовые потоки универсальный формат передачи данных

- Они просты для чтения, записи и редактирования человеком без использования специальных инструментов.

Текстовые форматы прозрачны.

- Если необходима производительность, то можно внедрить сжатие текстового потока на уровне выше или ниже.

Часто такая конструкция является более производительной, чем бинарное представление.

- Упрощается интерпретация и анализ взаимодействия приложений, а также написание тестовых программ.

- Серверные процессы часто запускаются с помощью суперсерверов подобных inetd/xinetd, так что сервер получает команды на стандартный ввод и отправляет ответ на стандартный вывод.

- Можно взаимодействовать с сервером или клиентом с помощью программ telnet или netcat.

# **36. ПРОЕКТИРОВАНИЕ ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ. МЕХАНИЗМЫ ПРОТОКОЛА.**

Проектирование прикладных протоколов

Disclaimer: основано на ~~реальных событиях~~ опыте проектирования протокола Blocks eXtensible eXchange Protocol (BXXP).

Возможные пути при проектировании протокола прикладного уровня

1. Попытайтесь найти существующий протокол, который более или менее подходит для ваших нужд.

2. Определите модель для обмена данными поверх инфраструктуры WWW, если более или менее подходит для ваших нужд.

3. Определите модель для обмена данными поверх инфраструктуры электронной почты, если она более или менее подходит для ваших нужд.

4. Разработать свой собственный протокол с 0.

Определение свойств проектируемого протокола

- Ориентация на соединение

+ Прикладной протокол ориентирован на соединения (т.е. работает поверх TCP или SCTP)

+ Без установления соединения к ним относятся прикладные протоколы, для которых нет нужды в задержках на установление/разрыв соединений, а также поддержке надежной передачи данных. При необходимости надежность реализуется средствами прикладного протокола.

- Обмен сообщениями

Под сообщением подразумевается простая структура данных, которыми обмениваются слабо-связные системы. В качестве противовеса можно привести сильно-связные системы, например, RPC. Проблема в том, что слабо- и сильно-связные системы - граничные части целого спектра.

- Асинхронный обмен

Синхронная peer-to-peer модель

Модель в стиле "запрос-ответ", где каждая из сторон может являться клиентом и/или сервером. Большинство протоколов являются синхронными: почта, WWW и т. п.

Часть протоколов не может быть построена по синхронным принципам, например: сетевые файловые системы, системы именования, мультикаст-сообщения и т.п.

Механизмы протокола (задачи):

- Структурирование - заголовки, концевики.

Octet-stung

Например, SMTP команды заканчиваются переводом каретки (CR-LF)

Octet-counting

Сообщение разделяется на заголовок и тело; в заголовке указывается размер данных в теле сообщения.

Connection-blasting

Например, информационное соединение FTP.

- Кодирование - представление.

Зависит от задачи!

Под кодированием подразумевается формат представления команд и данных. Например для SMTP он определяется в RFC 822.

В Интернете MIME является "де-факто" стандартом представления данных.

- Отчетность - каким образом описываются ошибки.

"Теория ответных кодов"

Состоит из 3-х цифр:

1. Успех-неуспех, либо постоянное или временное состояние;

2. ответственная часть системы;

3. идентификация конкретной ситуации.

Код может дополняться текстовым сообщением для человека.

2 нерешенных недостатка:

+ коды используются как для ответа на операцию, так и для индикации изменения состояния прикладного протокола;

+ код не указывает, кому адресуется ошибка пользователю, администратору или программисту.

- Асинхронность - как производится обмен независимыми сообщениями.

Протокол является асинхронным, если он поддерживает независимые обмены по одному соединению.

Наиболее широко распространенный подход pipelining, позволяет делать множество запросов к серверу, но требует, чтобы запросы были обслужены последовательно.

- Аутентификация - каким образом стороны идентифицируют и проверяют друг друга.

Механизмы специфичны для каждого протокола.

Например, FTP использует один механизм, HTTP другой, а SMTP (классический) не использует его вовсе.

Рекомендуется к использованию SASL (RFC 222 Simple Authentication and Security Layer) фреймворк для аутентификации сторон, использующий в том числе OTP (one-time-passwords).

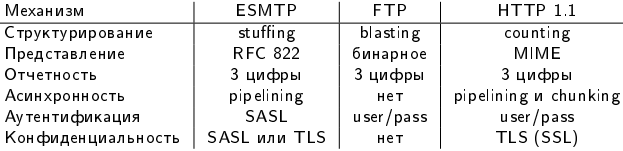
- Конфиденциальность - защита от перехвата или изменения данных.

Как правило, подразумевается использование криптографических средств для шифрования данных.

Можно использовать различные подходы для организации:

+ прослушивать на одном порту незащищенные соединения, а на другом защищенные (SSL);

+ прослушивать и защищенные, и незащищенные соединения на одном и том же порту, что требует поддержки со стороны протокола прикладного уровня (TLS).



# **37. ПРОЕКТИРОВАНИЕ ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ. СВОЙСТВА ПРОТОКОЛА.**

Проектирование прикладных протоколов

Disclaimer: основано на ~~реальных событиях~~ опыте проектирования протокола Blocks eXtensible eXchange Protocol (BXXP).

Возможные пути при проектировании протокола прикладного уровня

1. Попытайтесь найти существующий протокол, который более или менее подходит для ваших нужд.

2. Определите модель для обмена данными поверх инфраструктуры WWW, если более или менее подходит для ваших нужд.

3. Определите модель для обмена данными поверх инфраструктуры электронной почты, если она более или менее подходит для ваших нужд.

4. Разработать свой собственный протокол с 0.

Определение свойств проектируемого протокола

- Ориентация на соединение

+ Прикладной протокол ориентирован на соединения (т.е. работает поверх TCP или SCTP)

+ Без установления соединения к ним относятся прикладные протоколы, для которых нет нужды в задержках на установление/разрыв соединений, а также поддержке надежной передачи данных. При необходимости надежность реализуется средствами прикладного протокола.

- Обмен сообщениями

Под сообщением подразумевается простая структура данных, которыми обмениваются слабо-связные системы. В качестве противовеса можно привести сильно-связные системы, например, RPC. Проблема в том, что слабо- и сильно-связные системы - граничные части целого спектра.

- Асинхронный обмен

Синхронная peer-to-peer модель

Модель в стиле "запрос-ответ", где каждая из сторон может являться клиентом и/или сервером. Большинство протоколов являются синхронными: почта, WWW и т. п.

Часть протоколов не может быть построена по синхронным принципам, например: сетевые файловые системы, системы именования, мультикаст-сообщения и т.п.

Свойства протокола

При проектировании протокола необходимо учитывать некоторые свойства:

- Масштабируемость

Хорошо спроектированный протокол должен быть масштабируемым.

Не все протоколы поддерживают асинхронность.

Частая практика использовать несколько соединений для утилизации канала, однако при этом необходимо учитывать:

+ свойства транспортного протокола;

+ сервер трактует каждое новое соединение по протоколу прикладного уровня, как независимое (проблема аутентификации и доступа к ресурсам сессии).

- Эффективность

Хорошо спроектированный протокол должен быть эффективным.

Например, использование octet-stang'а упрощает реализацию протокола, в то время, как octet-counting потребляет немного меньше ресурсов.

- Простота

Хорошо спроектированный протокол должен быть простым.

Хорошее правило для определения насколько протокол прост:

+ в хорошо спроектированном протоколе усилия для изменения пропорциональны сложности изменений;

+ в плохо спроектированном протоколе необходимо приложить много усилий для любых изменений.

- Расширяемость

Хорошо спроектированный протокол должен быть расширяемым.

Невозможно предсказать заранее, с какими проблемами столкнется протокол.

Таким образом желательно предусмотреть способы расширения функциональности.

- Устойчивость

Хорошо спроектированный протокол должен быть устойчивым.

Внезапно! Но для протоколов принцип устойчивости Постела "будь либерален к тому, что принимаешь и консервативен к тому, что отсылаешь" может привести к обратному эффекту.

Проблема в количестве и качестве различных реализаций одного и того же протокола.

# **38. ХАРАКТЕРИСТИКА И ФУНКЦИОНИРОВАНИЕ ПРОТОКОЛА HTTP.**

HTTP - универсальный протокол прикладного уровня

Прикладной протокол работает поверх HTTP. WEB-серверы управляют транспортным уровнем и мультиплексированием HTTP-запросов. Легко расширяется.

HTTP используется также в качестве базового протокола для коммуникации пользовательских агентов с прокси-серверами и другими системами Интернет, в том числе и использующие протоколы SMTP, NNTP, FTP, Gopher, XMPP и многих других.

Центральным объектом в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы.

Изначально URL (Uniform Resource Locator) предназначался для обозначения мест расположения ресурсов (чаще всего файлов) во Всемирной паутине. Сейчас URL применяется для обозначения адресов почти всех ресурсов Интернета. Стандарт URL закреплён в RFC 1738

Структура URL

<схема>://<логин>:<пароль>@<хост>:<порт>/<URL путь>?<параметры>#<якорь>

URL-путь - уточняющая информация о месте нахождения ресурса; зависит от протокола.

Параметры - строка запроса с передаваемыми на сервер (методом GET) параметрами. Разделитель параметров — знак &. Якорь - идентификатор «якоря», ссылающегося на некоторую часть (раздел) открываемого документа.

HTTP - протокол без памяти

Протокол не хранит информацию о предыдущих запросах клиентов и ответах сервера.

Компоненты, использующие HTTP, могут самостоятельно осуществлять сохранение информации о состоянии, связанной с последними запросами и ответами.

Категории ПО HTTP

- Серверы - поставщики услуг хранения и обработки информации (обработка запросов).

- Клиенты - конечные потребители услуг сервера (отправка запросов).

- Прокси-серверы для поддержки работы транспортных служб.

HTTP-сеанс

"Классическая" схема HTTP-сеанса выглядит так.

1. Установление TCP-соединения. 2. Запрос клиента.

3. Ответ сервера. 4. Разрыв TCP-соединения.

Пример:

telnet ftp.mgts.by 80

C: GET /README HTTP/1.1

C: Host: ftp.mgts.by

C: User-Agent: telnet

C:

S: HTTP/1.1 200 OK

S: Server: nginx

S: Date: Mon, 14 Feb 2011 10:17:05 GMT

S: Content-Type: application/octet-stream

S: Content-Length: 197

S: Last-Modified: Wed, 27 Jan 2010

23:30:38 GMT

S: Connection: keep-alive

S: Accept-Ranges: bytes

S:

######## Тут сам файл #######

Connection closed by foreign host.

# **39. ХАРАКТЕРИСТИКА ПРОТОКОЛА HTTP. ЗАПРОСЫ HTTP.**

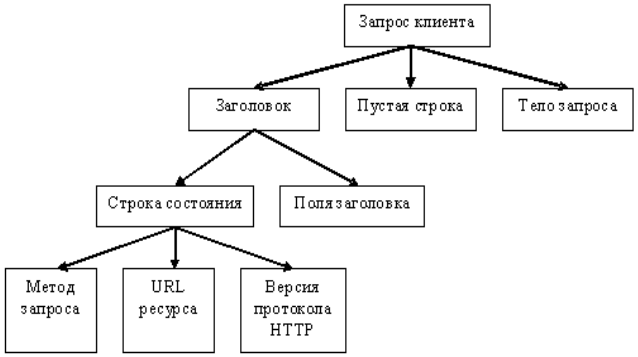
Cостав HTTP-запроса

- Строка состояния (строка-статус, или строка запроса)

- Поля заголовка.

- Пустая строка.

- Тело запроса.



Некоторые методы запроса:

- OPTIONS (Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. Формат тела и порядок работы с ним в настоящий момент не определён. Запросы «OPTIONS \* HTTP/1.1» могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1)

- GET (Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса)

- HEAD (Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения)

- POST (Применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML- форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы на сервер)

- PUT (Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус 201 (Created). Фундаментальное различие методов POST и PUT заключается в понимании предназначений URI ресурсов. Используя PUT, клиент предполагает, что загружаемое содержимое соответствует находящемуся по данному URI ресурсу)

- PATCH (Аналогично PUT, но применяется только к фрагменту ресурса)

- DELETE (Удаляет указанный ресурс)

- TRACE (Возвращает полученный запрос так, что клиент может увидеть, какую информацию промежуточные сервера добавляют или изменяют в запросе)

- CONNECT (Преобразует соединение запроса в прозрачный TCP/IP туннель, обычно чтобы содействовать установлению защищенному SSL соединению через не шифрованный прокси)

Часто используемые поля заголовка запроса HTTP:

- Host (Доменное имя или IP-адрес узла, к которому обращается клиент)

- Referer (URL документа, который ссылается на ресурс, указанный в строке состояния)

- From (Адрес электронной почты пользователя, работающего с клиентом)

- Accept (MIME-типы данных, обрабатываемых клиентом. Это поле может иметь несколько значений, отделяемых одно от другого запятыми. Часто поле заголовка Accept используется для того, чтобы сообщить серверу о том, какие типы графических файлов поддерживает клиент)

- Accept-Language (Набор двухсимвольных идентификаторов, разделенных запятыми, которые обозначают языки, поддерживаемые клиентом)

- Accept-Charset (Перечень поддерживаемых наборов символов)

- Content-Type (MIME-тип данных, содержащихся в теле запроса (если запрос не состоит из одного заголовка))

- Content-Length (Число символов, содержащихся в теле запроса (если запрос не состоит из одного заголовка))

- Accept-Encoding (Схож с полем Accept, но регламентирует кодировку содержимого, которая приемлема в отклике. Пустое поле Accept-Encoding указывает на то, что не приемлемо никакое кодирование. Accept-Encoding: compress, gzip)

- Range (Присутствует в том случае, если клиент запрашивает не весь документ, а лишь его часть)

- Connection (Используется для управления TCP-соединением. Если в поле содержится Close, это означает, что после обработки запроса сервер должен закрыть соединение. Значение Keep-Alive предлагает не закрывать TCP-соединение, чтобы оно могло быть использовано для последующих запросов)

- User-Agent (Информация о клиенте)

Пример запроса стандартного браузера

GET /robots.txt HTTP/1.1

Host: ftp.mgts.by

User-Agent: Mozilla/5.0 (X11; U; Linux x86\_64; en-US; rv:1.9.2.14pre)

Gecko/20101222 ALTLinux/Sisyphus/3.6.13-alt1.20101222 Firefox/3.6.13

Accept:

text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: ru,en-us;q=0.7,en;q=0.3

Accept-Encoding: gzip,deflate

Accept-Charset: windows-1251,utf-8;q=0.7,\*;q=0.7

Keep-Alive: 115

Connection: keep-alive

Referer: http://ftp.mgts.by/

# **40. ХАРАКТЕРИСТИКА ПРОТОКОЛА HTTP. HTTP-ОТВЕТ.**

Структура ответа сервера

- Строка состояния.

*Версия\_протокола Код\_ответа Пояснительное\_сообщение*

Версия\_протокола - задается формате, что и в запросе клиента, с тем же смыслом

Код\_ответа - это трехзначное десятичное число, представляющее в закодированном виде результат обслуживания запроса сервером.

Пояснительное\_сообщение - дублирует код ответа в символьном виде. Это строка символов, которая не обрабатывается клиентом.

Классы кодов ответов

1 - специальный класс сообщений, называемых информационными. Код ответа, начинающийся с 1, означает, что сервер продолжает обработку запроса.

2 - успешная обработка запроса клиента.

3 - перенаправление запроса. Чтобы запрос был обслужен, необходимо предпринять дополнительные действия.

4 - ошибка клиента. Как правило, возвращается в том случае, если в запросе клиента встретилась синтаксическая ошибка.

5 - ошибка сервера.

- Поле заголовка.

Часто используемые поля заголовков ответов

+ Server (Имя и номер версии сервера)

+ Age (Время в секундах, прошедшее с момента создания ресурса)

+ Allow (Список методов, допустимых для данного ресурса)

+ Content-Language (Языки, которые должен поддерживать клиент для того, чтобы корректно отобразить передаваемый ресурс)

+ Content-Type (MIME-тип данных, содержащихся в теле запроса (если запрос не состоит из одного заголовка))

+ Content-Length (Число символов, содержащихся в теле запроса (если запрос не состоит из одного заголовка))

+ Content-Encoding (Значения кодировки содержимого указывают на кодовое преобразование, которое было или может быть выполнено над объектом. Кодировки содержимого первоначально применены для того, чтобы иметь возможность архивировать документ или преобразовать его каким-то другим способом без потери идентичности или информации.)

+ Last-Modified (Дата и время последнего изменения ресурса)

+ Date (Дата и время, определяющие момент генерации ответа)

+ Expires (Дата и время, определяющие момент, после которого информация, переданная клиенту, считается устаревшей)

+ Location (В этом поле указывается реальное расположение ресурса. Оно используется для перенаправления запроса)

+ Cache-Control (Директивы управления кэшированием. Например, no-cache означает, что данные не должны кэшироваться)

+ Transfer-Encoding (указывает, какой тип преобразования (если таковое использовано) применен к телу сообщения, для того чтобы безопасно осуществить передачу между отправителем и получателем. Transfer-Encoding: chunked)

- Пустая строка.

- Тело ответа.

Пример ответа сервера

HTTP/1.1 200 OK

Server: nginx

Date: Mon, 14 Feb 2011 11:17:40 GMT

Content-Type: application/octet-stream

Content-Length: 197

Last-Modified: Wed, 27 Jan 2010 23:30:38 GMT

Connection: keep-alive

Accept-Ranges: bytes

# **41. ХАРАКТЕРИСТИКА ПРОТОКОЛА HTTP. БЕЗОПАСНОСТЬ, АУТЕНТИФИКАЦИЯ, COOKIES.**

Безопасность передачи данных HTTP

Протокол HTTP предназначен для передачи символьных данных в открытом (незашифрованном) виде.

Самым простейшим является расширение HTTPS, при котором данные, передаваемые по протоколу HTTP, "упаковываются" в криптографический протокол SSL или TLS, тем самым обеспечивая защиту этих данных. В отличие от HTTP, для HTTPS по умолчанию используется TCP-порт 443

Аутентификация в HTTP

Basic - базовая аутентификация, при которой имя пользователя и пароль передаются в заголовках http-пакетов.

Digest - пароль пользователя передается в хешированном виде. Использование SSL является обязательным.

Integrated - интегрированная аутентификация, с помощью протоколов NTLM или Kerberos.

“Печеньки” :-)

HTTP-сервер не помнит предыстории запросов клиентов, поэтому каждый запрос обрабатывается независимо от других, и у сервера нет возможности определить, исходят ли запросы от одного клиента или разных клиентов.

Механизм cookie позволяет серверу хранить информацию на компьютере клиента и извлекать ее оттуда.

Cookie

Инициатором записи cookie выступает сервер.

Если в ответе сервера присутствует поле заголовка Set-cookie, клиент воспринимает это как команду на запись cookie. В дальнейшем, если клиент обращается к серверу, от которого он ранее принял поле заголовка Set-cookie, помимо прочей информации он передает серверу данные cookie. Для передачи указанной информации серверу используется поле заголовка Cookie.

Формат полей для печенек

В одном запросе или ответе может содержаться несколько полей Set-Cookie/Cookie.

Set-Cookie: name=newvalue; expires=date; path=/; domain=.example.org.

Set-Cookie: RMID=732423sdfs73242; expires=Fri, 31-Dec-2010 23:59:59 GMT; path=/;

domain=.example.net

Cookie: name=value; name2=value2

# **42. ПРОЕКТИРОВАНИЕ ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ НА БАЗЕ HTTP.**

Причины для использования:

- знакомый и хорошо известный протокол

- совместим с существующими браузерами

- возможность реиспользования существующих клиентских и серверных библиотек

- удобно, если для проекта в любом случае необходимо использовать http

- скорость прототипирования сервера (например cgi)

- возможность использования существующих механизмов безопасности (HTTP digest auth, SSL, TLS)

- пропускается стандартными файерволами и/или прокси-серверами

При проектировании своего собственного протокола поверх уже существующего, возникают вопрос о том, нужно ли вообще использовать существующий протокол? А если нужно, то каким образом?

1. Должно ли приложение использовать порт, отличный от 80?

Ответ: Необходимо использовать новый порт, если любое из этих условий подходит для вашего приложения:

- “новый сервис” и классический http-сервер используют различные наборы данных;

- существует причина для создания нового сервиса в виде отдельного процесса или отдельного от “классического” кода;

- существует причина для необходимости легкого различения трафика относящегося к вашему приложению;

В остальных случаях предпочтительно создавать новые расширения, за счет добавления новых методов

2. Должно ли приложение использовать стандартные методы (GET/POST etc.) или необходимо объявить свои собственные методы?

3. Нужно ли приложению использовать стандартную схему (идентификатор протокола) “http:” или необходимо использовать свою?

Ответ: Схема указывает на тип используемого ресурса или сервиса. Для решения можно использовать критерии:

- планируется широкое распространение схемы;

- в любом случае будет использоваться новый порт;

- требуется отличная от http предварительная настройка либо взаимодействие;

- пользовательские приложения могут значительно облегчить жизнь пользователя (например: http: и mailto: в браузере)

4. Есть ли необходимость в объявлении своих собственных типов MIME?

Ответ: Типы MIME подсказывают приложению, как обработать данные ответа. При этом надо уделить внимание:

- может ли использоваться существующие фреймфорки (например text/directory или XML)?

- можно ли использовть типы сообщений или multipart MIME?

- планируется ли пересылать эти же данные по электронной почте?

- MIME описывает тип данных, и не должен использоваться для описания семантики.

Проблемы выбора при проектировании протокола с использованием HTTP

Необходимо определить, может ли вообще HTTP протокол использоваться в качестве основы, а если может, то в каком объеме?

1. Сложность (Complexity)

HTTP – простой протокол. Однако существуют расширения увеличивающие его сложность: поддержка постоянных соединений, байтовые диапазоны, согласовывание контента, поддержка кэширования и другие. Эти расширения удобны для WEB-приложений, но далеко не всегда требуются приложениям работающим поверх HTTP. Даже при использовании существующих библиотек для работы с HTTP, сложность взаимодействия может получится больше, чем при использовании своего протокола.

2. Избыточность (Overhead)

Использование TCP сессий и полей HTTP приводит к задержкам и избыточности при частой передаче данных небольшого размера. С другой стороны, можно использовать постоянные соединения для передачи нескольких запросов/ответов HTTP.

3. Безопасность (Security)

Не для всех приложений может подойти схема с использованием digest-like аутентификации или TLS, поскольку для них существует множество ограничений. Проблема basic/digest аутентификации — общие секретные данные. Подходит для небольших групп. Нет смысла использовать протокол HTTP только ради использования встроенных в фреймворк систем безопасности. SSL/TLS позволяют аутентифицировать обе стороны, однако есть проблема получения корневых и самоподписанных сертификатов. Кроме того клиент не всегда может иметь возможность аутентификации. Используемые по-умолчанию механизмы шифрования SSL/TLS, как правило слабы и, при использовании современного оборудования, расшифровываются за несколько дней или часов.

4. Совместимость с прокси, сетевыми экранами и NAT

Это одна из основных причин для использования HTTP.

5 вопросов на которые надо ответить, при решении использовать HTTP

1. Подходит ли HTTP в качестве подходящего транспорта для приложения?

2. Можно ли будет использовать новый протокол в браузерах без изменений?

3. Подходят ли существующие механизмы безопасности HTTP для приложения?

4. Подходит ли парадигма “теории ответных кодов” для вашего приложения?

5. Есть ли необходимость в использовании HTTP для сервера, вне зависимости от вашего приложения?

Проблема выбора между существующими и новыми методами HTTP

Хотя предпочтительно использовать новые методы вместо использования нового порта, это само по себе не означает, что можно отказаться от нового порта или новой схемы URL.

Использование существующего кода

Хотя использование существующего кода для клиента, сервера или прокси является одной из важнейших причин, необходимо помнить, что протокол HTTP не проектировался для использования в качестве траспортного, поэтому в существующих реализациях многие вещи считаются неизменными или существуют проблемы с расширением этих реализаций.

Проблемы при использовании “теории ответных кодов”

Нежелательно расширять статусные коды – возможно, что они будут закреплены за другими ситуациями. Основная проблема возникает при использовании прокси- или коширующего сервера, т.к. стандартные реализации не знают о ваших расширениях.

Желательно придерживаться следующего руководства при проектировании:

- использовать стандартные статусы для уведомления об ошибках в запросе или заголовке запроса;

- если ошибки обнаружены в теле сообщения, то можно использовать коды 200 или 500 для уведомления, причем необходимо предусмотреть механизм “антикэширования” в случае использования кода 200;

- приложение может возвращать код 200 в случае успеха и неуспеха, в таком случае приложение должно определить конкретную ситуацию в теле сообщения;

- приложение, которое не может корректно работать с прокси- и кэширующими серверами не должно использовать протокол HTTP в качестве своей основы.

***Выводы и рекомендации по использованию HTTP, как основы для своего приложения***

- Все протоколы должны предоставлять адекватный уровень защиты.

Требования к защите зависят от приложения, однако требуется помнить, что использование HTTP/TLS не предоставляет адекватной защиты во всех случаях и окружениях.

- Новые протоколы – вне зависимости, основаны они на http или нет – не должны маскироваться под уже существующие протоколы, с целью обойти пользовательские сетевые фильтры.

- В общем случае новые протоколы или сервисы не должны использовать существующие схемы такие, как “http:”

- Каждый новый протокол, базирующийся на HTTP обязан учитывать в своей спецификации, каким образом будет использоваться HTTP, включая то, как клиент и сервер будут взаимодействовать посредством прокси-сервера.

- Необходимо следовать руководству, описанному выше, по использованию статусных кодов HTTP.

# **43. ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ПРОТОКОЛА ICMP (НА ПРИМЕРЕ РЕАЛИЗАЦИИ УТИЛИТ PING ИЛИ TRACEROUTE).**

Протокол ICMP

Протокол передачи команд и сообщений об ошибках (обмена управляющими сообщениями) (ICMP - internet control message protocol) позволяет маршрутизатору сообщить конечному узлу об ошибках, с которыми машрутизатор столкнулся при передаче какого-либо IP-пакета от данного конечного узла.

Управляющие сообщения ICMP не могут направляться промежуточному маршрутизатору, который участвовал в передаче пакета, с которым возникли проблемы, так как для такой посылки нет адресной информации - пакет несет в себе только адрес источника и адрес назначения, не фиксируя адреса промежуточных маршрутизаторов.

Протокол ICMP - это протокол сообщения об ошибках, а не протокол коррекции ошибок.

Конечный узел может предпринять некоторые действия для того, чтобы ошибка больше не возникала, но эти действия протоколом ICMP не регламентируются.

ICMP-протокол сообщает об ошибках в IP-дейтограммах, но не дает информации об ошибках в самих ICMP-сообщениях.

ICMP использует IP, а IP-протокол должен использовать ICMP.

В случае ICMP-фрагментации сообщение об ошибке будет выдано только один раз на дейтограмму, даже если ошибки были в нескольких фрагментах.

Задачи ICMP

- передачу отклика на пакет или эхо на отклик;

- контроль времени жизни дейтограмм в системе;

- реализует переадресацию пакета;

- выдает сообщения о недостижимости адресата или о некорректности параметров;

- формирует и пересылает временные метки;

- выдает запросы и отклики для адресных масок и другой информации.

ICMP-сообщения об ошибках не выдаются

- ICMP-сообщение об ошибке.

- При мультикастинг или широковещательной адресации.

- Для фрагмента дейтограммы (кроме первого).

- Для дейтограмм, чей адрес отправителя является нулевым, широковещательным или мультикастинговым.

Передача сообщений ICMP

На примере программ ping и traceroute

Ping - утилита для проверки доступности удаленного узла и оценки качества связи.

Также может использоваться для зондирования сети (ping sweep), а принципы работы для реализации атак типа ICMP flooding, Smurf и Ping-of-Death.

\* Некоторые провайдеры в договоре оговаривают отдельным пунктом ограничение на скорость ICMP-пакетов, оставляя за собой право прекращения предоставления услуги в случае ICMP флуда, нарушающего работу сетевого оборудования.

Большинство реализаций TCP/IP поддерживают Ping-сервер непосредственно в ядре - сервер не является пользовательским процессом.

Реализация утилиты ping (ключевые особенности)

- Для начала получаем свой PID для использования в качестве ID в своих пакетах, т.к. в пакете ICMP нет поля порт, то при запуске нескольких процессов PING одновременно может возникнуть проблема с тем какому из процессов следует передать тот или иной отклик. Для преодоления этой неопределенности следует использовать уникальные значения полей идентификатор.

- Установка обработчика сигналов SIGALRM и SIGINT (catcher() - обработчик функции)

- Для приема-отправки ICMP сообщений необходимо создать raw-сокет c указанием IPPROTO\_ICMP в функции socket() (Raw-сокет может создавать только root (используется suid-bit))

- Восстановление первоначальных пользовательских прав вызываем после создания сокета

- Если есть необходимость, то устанавливаем широковещательный режим с увеличением приемного буфера, для защиты от переполнения

- Запускаем интервальный таймер, посылающий сигнал SIGALRM. таймер сработает через 1 микросекунду... и будет активироваться каждую секунду

- Далее запускаем бесконечный цикл, в котором будем принимать пакеты

- Определяем текущее системное время

- Вызываем функцию для разбора принятого пакета и вывода данных на экран

- Отправка пакета происходит в функции catcher каждую секунду по сигналу SIGALRM

Утилита traceroute предназначена для определения маршрута прохождения пакета до узла, что позволяет исследовать логическую топологию сети и путей проникновения в нее.

Реализуется в 2-х вариантах ― с использованием UDP (\*nix) и ICMP (win & \*nix) протоколов.

Алгоритм работы для UDP

1. формируется udp дейтаграмма

2. поле TTL в IP пакете устанавливается = 1

3. на указанный адрес посылается udp дейтаграмма

4. промежуточный маршрутизатор уменьшает поле TTL на 1 и, если оно = 0, то посылает ICMP-сообщение TIME\_EXCEEDED (время жизни пакета истекло)

5. если пакет достиг получателя, то в ответ будет передано ICMP-сообщение PORT\_UNREACHABLE (что бы случайно не попасть на работающую службу, посылаются 3 сообщения по случайным адресам выше 33434)

6. если не достигли получателя, то наращиваем TTL на 1 и переходим к п.3

В программе создается 2 сокета для UDP и ICMP, чтобы определить к какому приложению относится вернувшееся ICMP-сообщение, используем привязку к порту = pid процесса

Возвращающиеся ICMP сообщения содержат IP заголовок и 64 бита исходной дейтаграммы, что позволяет определить процесс, которому предназначено сообщение.

Traceroute с использованием ICMP

Алгоритм работы для traceroute с использованием ICMP почти не отличается от варианта с UDP:

- вместо исходящей UDP-дейтаграммы посылается ICMP (Echo Request)

- от конечного узла должен прийти ICMP (Echo Reply)

- используется один и тот же сокет для отправки и приема сообщений (как в ping)

# **44. АТАКИ ICMP-FLOODING И SMURF: ОСОБЕННОСТИ РЕАЛИЗАЦИИ.**

ICMP-flooding

Размер ICMP-запроса обычно небольшой (около 64 байт, при максимальном размере пакета IP 64 кбайт).

В результате, при формальном сохранении небольшого трафика, возникает перегрузка по количеству пакетов, и устройство начинает пропускать остальные пакеты (по другим интерфейсам или протоколам), что и является целью атаки.

Пример команды для атаки:

ping -f -s 4096 victum.domain.com

отправляет поток icmp запросов на атакуемого без задержек и с размером пакета = 4K

Противодействие:

- отключение ответов на ICMP-запросы (отключение соотвествующих служб или предотвращение отклика на определенный тип сообщения) на целевой системе;

- понижение приоритета обработки ICMP-сообщений (при этом весь остальной трафик обрабатывается в обычном порядке, а ICMP-запросы обрабатываются по остаточному принципу, в случае перегрузки ICMP-сообщениями часть из них игнорируется).

- отбрасывание или фильтрация ICMP-трафика средствами межсетевого экрана.

Smurf

Для реализации данного типа атак используется тот же принцип, что и для реализации ICMP-flooding.

Отличие от предыдущего типа состоит в том, что используется широковещательная рассылка пакета ICMP Echo Request, а в качестве адреса источника указывается ip-адрес жертвы.

# **45. IP-SPOOFING. ОСОБЕННОСТИ СОЗДАНИЯ ПАКЕТОВ ПРОТОКОЛА TCP (НА ПРИМЕРЕ АТАК SYN-FLOODING И LAND).**

IP-spoofing

Подделка обратного адреса в посылаемых пакетах называется IP-spoofing и используется для сокрытия исходного узла.

Работа с IP-заголовком

Для самостоятельной настройки заголовка IP нам необходимо использовать тип сокета RAW с опцией IP\_HDRINCL:

\* Для максимального эффекта необходимо использовать максимальный размер пакета, но не превышающий размера MTU интерфейса, через который будет отправлен пакет, чтобы не реализовывать в своей программе алгоритм фрагментации ip-пакетов.

Подменяем адрес (правильно заполняем поля заголовка IP и ICMP)

SYN-flooding

Суть атаки состоит в посылке бесконечного числа TCP пакетов с установленным флагом SYN на атакуемый хост

Принцип работы:

- Приходит TCP пакет с установленным флагом SYN

- Cистема отсылает SYN/ACK, ставит пакет в очередь и ждет ACK от отправителя

Влияет на всю систему! Например, видеоплеер «замерзает».

Работа с заголовками IP и TCP

Основная особенность программы состоит в том, что необходимо использовать сокеты RAW и вручную заполнять необходимые поля IP и TCP протоколов

Для протоколов TCP и UDP подсчет контрольной суммы имеет несколько особенностей:

1. Добавляются 12-байтные псевдо-заголовки, содержащие IP-адреса отправителя и получателя, код протокола и длину дейтограммы

2. Если вычисленная контрольная сумма равна нулю, в соответствующее поле будет записан код 65535 (тоже самое для заголовка IP).

\*Для протокола UDP контрольную сумму можно не рассчитывать, а установить 0.

Land

Атака отказа в локальной сети (LAND) — это тип атаки типа «отказ в обслуживании» (DOS), при которой злоумышленник атакует сеть, устанавливая одинаковые IP-адреса и порты источника и назначения сегмента TCP. Атака Land завершается успешно, заставляя компьютер отвечать самому себе, так что целевой хост отправляет ответ; Пакет SYN-ACK сам себе до тех пор, пока машина не выйдет из строя или не зависнет из-за многократной обработки пакета стеком TCP

В качестве порта и адреса отправителя используется адрес и порт назначения.

# **46. ПРОТОКОЛ IPV6. ХАРАКТЕРИСТИКИ. ПРЕИМУЩЕСТВА И ОТЛИЧИЯ ОТ IPV4.**

Отличия от IPv4

- Расширение адресации

+ длина адреса расширена до 128 бит позволяет обеспечить больше уровней иерархии адресации, увеличить число адресуемых узлов, упростить авто-конфигурацию

+ добавлено субполе "scope" (группа адресов)

+ новый тип адреса "anycast"

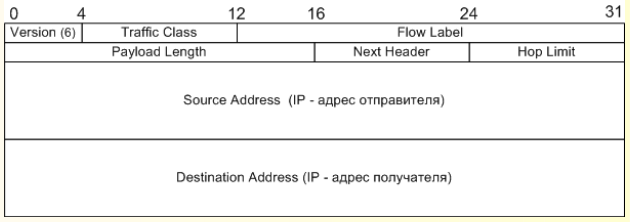
- Спецификация формата заголовков (Некоторые поля заголовка IPv4 отбрасываются или делаются опционными, уменьшая издержки, связанные с обработкой заголовков)

- Улучшенная поддержка расширений и опций (Изменение кодирования опций IP-заголовков позволяет облегчить переадресацию пакетов, ослабляет ограничения на длину опций, и делает более доступным введение дополнительных опций в будущем)

- Возможность пометки потоков данных (Введена возможность помечать пакеты, принадлежащие определенным транспортным потокам)

- Идентификация и защита частных обменов (Введена спецификация идентификации сетевых объектов или субъектов, для обеспечения целостности данных и, при желании, защиты частной информации)

Формат заголовка IPv6:



Поля заголовка IPv6

- Traffic class (8 бит). Класс трафика. Используется для идентификации и различения различных классов либо приоритетов (0 по-умолчанию, Промежуточные узлы могут изменять значение)

- Flow Label (20 бит). Метка потока. Может использоваться отправителем для того, чтобы помечать пакеты, которые требуют специальной обработки сетевыми модулями Ipv6. Все пакеты, принадлежащие к одному потоку, должны отправляться по одному и тому же адресу назначения и с одним и тем же приоритетом.

- Payload Length (16 бит). Длина данных. Длина данных пакета (в байтах), которые следуют за заголовком. Если величина этого поля равна 0, то длина данных дейтаграммы более 65535 и хранится в поле Jumbo Payload

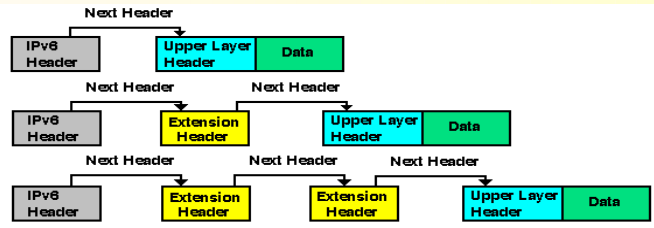
- Next Header (8 бит). Поле следующего заголовка. Это поле содержит информацию типа заголовка, который следует за заголовком IPv6.

- Hop Limit (8 бит). Поле ограничения пересылок. Величина этого поля уменьшается на 1 при прохождении дейтаграммой шлюза или хоста. Если величина этого поля равна 0, дейтаграмма уничтожается.

- Source Address (128 бит). Адрес отправителя.

- Destination Address (128 бит). Адрес получателя. (Если в заголовке присутствует вложенный заголовок маршрутизации (Routing header), то поле адреса получателя может и не быть адресом назначения)

Инкапсуляция заголовков



Заголовки расширений

1. IPv6.

2. Hop-by-Hop Options (Тут может быть опция “Jumbo Frames”)

3. Destination Options (опции получателя 1).

4. Routing (маршрутизация).

5. Fragment (фрагментация).

6. Authentication (аутентификация).

7. Encapsulating Security Payload (дополнительная аутентификация).

8. Destination Options (опции получателя 2).

9. Заголовок верхнего уровня (например, TCP).

# **47. ПРОТОКОЛ IPV6. ТИПЫ АДРЕСОВ. ЗАПИСЬ АДРЕСА.**

Типы адресов:

- Unicast (Идентификатор одиночного интерфейса. Пакет, посланный по уникастному адресу, доставляется интерфейсу, указанному в адресе.)

- Anycast (Идентификатор набора интерфейсов (принадлежащих разным узлам). Пакет, посланный по эникастному адресу, доставляется одному из интерфейсов, указанному в адресе (ближайший, в соответствии с мерой, определенной протоколом маршрутизации))

- Multicast (Идентификатор набора интерфейсов (обычно принадлежащих разным узлам). Пакет, посланный по мультикастинг-адресу, доставляется всем интерфейсам, заданным этим адресом)

В IPv6 не существует широковещательных адресов, их функции переданы мультикастинг-адресам.

Представление записи адресов

Основная форма имеет вид

x:x:x:x:x:x:x:x, где 'x' шестнадцатеричные 16-битовые числа.

fedc:ba98:7654:3210:FEDC:BA98:7654:3210

1080:0:0:0:8:800:200C:417A

Синтаксис для удаления лишних нулей в адресе.

Использование записи "::" указывает на наличие групп из 16 нулевых бит. Комбинация "::" может появляться только при записи адреса.

Последовательность "::" может также использоваться для удаления из записи начальных или завершающих нулей в адресе.

1080:0:0:0:8:800:200c:417a = 1080::8:800:200c:417a

Ff01:0:0:0:0:0:0:43 = ff01::43

0:0:0:0:0:0:0:1 = ::1

Альтернативная форма для работы с ipv4 и Ipv6

x:x:x:x:x:x:d.d.d.d

где 'x' шестнадцатеричные 16-битовые коды адреса, а 'd' десятичные 8-битовые, составляющие младшую часть адреса (стандартное IPv4 представление).

0:0:0:0:0:FFFF:129.144.52.38 = ::FFFF:129.144.52.38

Идентификация типа адреса

Идентификация происходит по первым битам адреса.



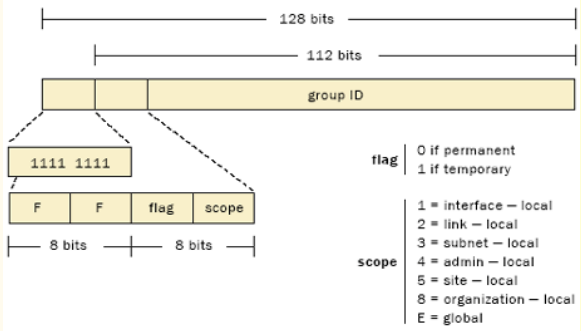
0:0:0:0:0:0:0:0 – Специальный тип адреса, который не должен присваиваться ни одному узлу

0:0:0:0:0:0:0:1 – Loopback (адрес обратной связи)

Link-local unicast:

1111111010 :: interface ID (64 бита)

Multicast Address



# **48. МНОГОАДРЕСНАЯ ПЕРЕДАЧА. ПОНЯТИЕ И МЕХАНИЗМЫ. АДРЕСАЦИЯ И ОБЛАСТЬ ДЕЙСТВИЯ IPV4.**

Multicast

специальная форма широковещания, при которой копии пакетов направляются определённому подмножеству адресатов.

Дейтаграмму многоадресной передачи должны получать только заинтересованные в ней интерфейсы – т.е. на этих интерфейсах работают приложения, желающие принять участие в сеансе многоадресной передачи данных.

IPv4 multicast

Используются адреса класса D:

224.0.0.0 – 239.255.255.255

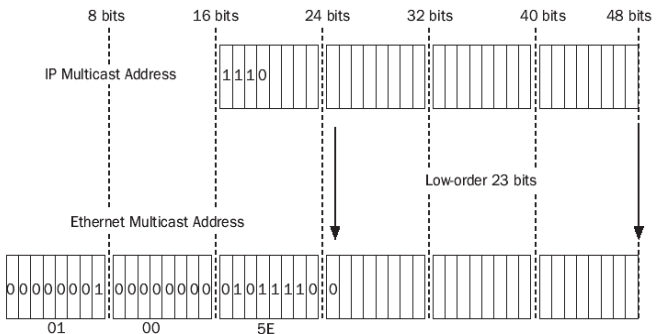
Младшие 28 бит – идентификатор группы многоадресной передачи (multicast group ID)

Multicast и канальный уровень

Современные сетевые карты умеют принимать кадры с установленным битом режима многоадресной передачи.

Для беспроводных сетей 802.11, передача кадра может задержаться в зависимости от режима сохранения энергии (power-save mode).

Multicast: Ethernet & IPv4



Используется универсально управляемый адрес Ethernet

Специальные адреса IPv4

- 224.0.0.1

группа всех узлов (all-hosts group)

К этой группе должны присоединиться все узлы в сети, имеющие возможность многоадресной передачи.

- 224.0.0.2

группа всех маршрутизаторов (all-routers group)

К этой группе должны присоединиться все маршрутизаторы сети, поддерживающие многоадресную передачу.

- 224.0.0.0 -224.0.0.255

Локальные адреса на канальном уровне (link local)

Предназначены для низкоуровневого определения топологии и служебных протоколов. Дейтаграммы с такими адресами никогда не передаются маршрутизатором дальше.

Ограничение области действия

Исторически сложилось, что поле TTL в IPv4 выполняло роль поля области действия многоадресной передачи:

0: локальный в пределах узла

1: локальный в пределах сети

<32: локальный в пределах сайта

<64: локальный в пределах региона

<128: локальный в пределах континента

<255: глобальный

Более предпочтительно использовать административное управление областями действия

Адреса 239.0.0.0 - 239.255.255.255 – пространство адресов с административным ограничением областью действия

Адреса 239.255.0.0 – 239.255.255.255 – локальная область действия (~site-local)

Адреса 239.192.0.0 – 239.195.255.255 – локальная область действия в пределах организации

Сеансы многоадресной передачи

Сочетание адреса многоадресной передачи и порта транспортного уровня часто называют сеансом (session).

Пример: потоковое мультимедиа делится на 2 сеанса – аудио и видео. При этом иногда выгодно использовать не только разные порты, но и разные группы.

Многоадресная передача в глобальной сети

Маршрутизаторы общаются между собой при помощи какого-либо протокола маршрутизации многоадресной передачи (например, DVMRP).

Когда узел присоединяется к группе, он отправляет всем маршрутизаторам в своей сети сообщение IGMP.

Маршрутизаторы обмениваются этой информацией по протоколу маршрутизации многоадресной передачи.

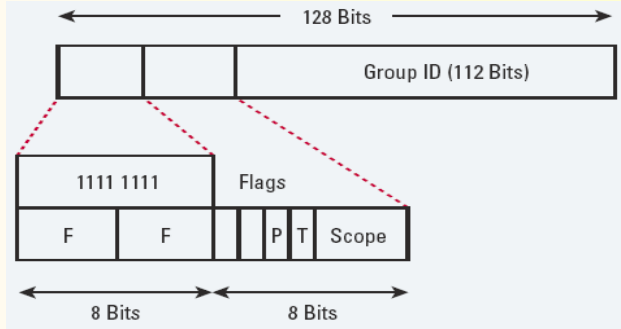
# **49. МНОГОАДРЕСНАЯ ПЕРЕДАЧА. ПОНЯТИЕ И МЕХАНИЗМЫ. АДРЕСАЦИЯ И ОБЛАСТЬ ДЕЙСТВИЯ IPV6.**

Multicast

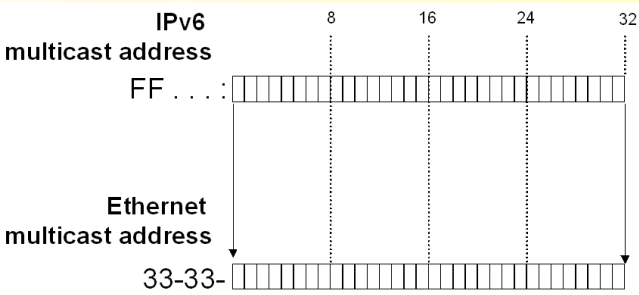
специальная форма широковещания, при которой копии пакетов направляются определённому подмножеству адресатов.

Дейтаграмму многоадресной передачи должны получать только заинтересованные в ней интерфейсы – т.е. на этих интерфейсах работают приложения, желающие принять участие в сеансе многоадресной передачи данных.

Multicast IPv6 Address



Multicast: Ethernet & IPv6



Используется локально управляемый групповой адрес Ethernet

Группы адресов multicast IPv6

Флаги P=0, T=0: группа заранее известных адресов (well-known)

Флаги P=0, T=1: группа временных адресов (transient)

Флаги P=1, T=1: адрес считается назначенным на основе одноадресного префикса

Специальные адреса IPv6

- FF02::1

группа всех узлов (all-nodes group)

К этой группе должны присоединиться все узлы в сети, имеющие возможность многоадресной передачи (включая маршрутизаторы, принтеры, тостеры и холодильники, если они имеют IPv6 адрес).

В отличие от IPv4 присоединение является обязательным!

- FF02::2

группа всех маршрутизаторов (all-routers group)

К этой группе должны присоединиться все маршрутизаторы сети, поддерживающие многоадресную передачу.

Аналогичен адресу 224.0.0.2.

Область действия: scope

1: Node-local – локальная в пределах узла

2: Link-local – локальная в пределах физической сети

4: Admin-local – локальная в пределах области администрирования

5: Site-local – локальная в пределах сайта

8: Organization-local – локальная в пределах организации

14: глобальная

4,5,8 определяются администратором

Сеансы многоадресной передачи

Сочетание адреса многоадресной передачи и порта транспортного уровня часто называют сеансом (session).

Пример: потоковое мультимедиа делится на 2 сеанса – аудио и видео. При этом иногда выгодно использовать не только разные порты, но и разные группы.

Многоадресная передача в глобальной сети

Маршрутизаторы общаются между собой при помощи какого-либо протокола маршрутизации многоадресной передачи (например, DVMRP).

Когда узел присоединяется к группе, он отправляет всем маршрутизаторам в своей сети сообщение IGMP.

Маршрутизаторы обмениваются этой информацией по протоколу маршрутизации многоадресной передачи.