Компьютерные системы и сети

Дмитрий Андреевич Сурков
Владислав Александрович Савёнок

КСиС: цель дисциплины

- Формирование целостной картины устройства локальных и глобальных сетей:
  - от кодирования сигналов в линиях связи и работы аппаратных сетевых адаптеров;
  - до протоколов передачи данных, веб служб и интернет-технологий.
- Профессиональное развитие специалиста в области информационных технологий
  - позволяет в дальнейшем совершенствовать навыки разработки профессиональных программных средств, отвечающих современному этапу развития компьютерной техники.

КСиС: значимость дисциплины

- КСиС – это название нашего факультета.
- Компьютер как инструмент общения, а общение – это связь и средства коммуникаций.
- В современном мире доступ к любой информации осуществляется через сеть (интернет).
- Большая часть современного ПО использует сеть.
- Дальнейший прогресс в индустрии ПО обусловлен развитием сетей передачи данных, стандартов и протоколов интернета:
  - высокоскоростные беспроводные сети 5G;
  - глобальный спутниковый интернет.
- Сеть – это часть «ноосферы» (термин Вернадского).

Источники

- Конспект лекций Суркова Д.А.
- В. Олифер, Н. Олифер – Компьютерные сети. Принципы, технологии, протоколы. Учебник для вузов
- Интернет-источники:
  - RFC: https://www.ietf.org/rfc/
  - Перевод RFCs на русский: http://rfc.com.ru/
  - MDN Web Docs: https://developer.mozilla.org/
  - High Performance Browser Networking: https://hpbn.co/

RFC
Request For Comments

MDN Web Docs
moz://a

Терминология

- Вычислительная (компьютерная) сеть –
  - совокупность вычислительных устройств, соединенных с помощью каналов связи и средств коммутации в единую систему для обмена сообщениями и совместного решения общей задачи (доступа пользователей к программным, техническим, информационным и организационным ресурсам сети).
- Топология –
  - граф вычислительной сети, узлами которого являются вычислительные устройства, а ветвями – соединяющие их каналы связи.
- Сообщение –
  - данные, передаваемые между двумя узлами вычислительной сети.

Терминология (продолжение)

- Трафик –
  - количество сообщений, передаваемых в сети (во всей сети) за определённый период времени.
- Пропускная способность –
  - предельное количество сообщений, которые могут передаваться в единицу времени между двумя узлами.
- Задержка –
  - время (среднее) доставки сообщения между двумя узлами (туда и обратно).

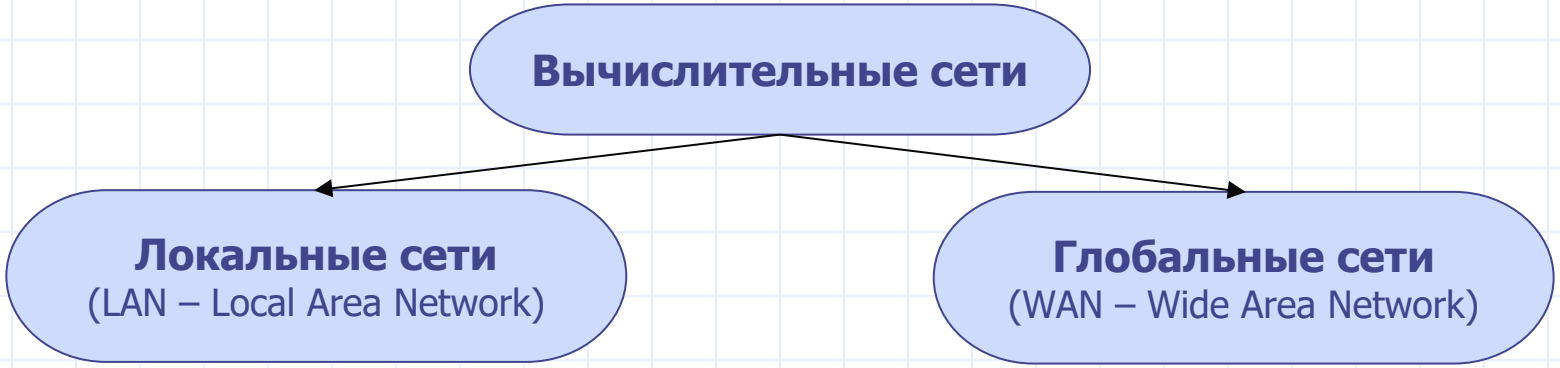
Терминология (продолжение)

- Сервер –
  - узел сети, который выполняет запросы клиентов и обеспечивает им совместный доступ к некоторому ресурсу. Ресурсом может быть файловое хранилище, база данных, устройство печати, веб сайт, вычислительные мощности и др.
- Клиент –
  - узел сети, обращающийся к ресурсам удалённого узла-сервера с помощью сообщений-запросов. Узел может одновременно выступать в роли клиента и сервера.
- Служба (сетевой сервис) –
  - один или несколько взаимодействующих логических серверов, обеспечивающих доступ к конкретному типу ресурса с помощью установленных правил обмена сообщениями (протокола).

Экскурс в историю

- Многотерминальные системы – прообраз сети
- Появление глобальных сетей
  - сети с коммутацией каналов, сети с коммутацией пакетов
- Появление локальных сетей
  - проводные (Ethernet) и беспроводные (Wi-Fi)
- Сближение локальных и глобальных сетей
- Появление интернета
- Конвергенция телекоммуникационных и компьютерных сетей

Локальные и глобальные сети



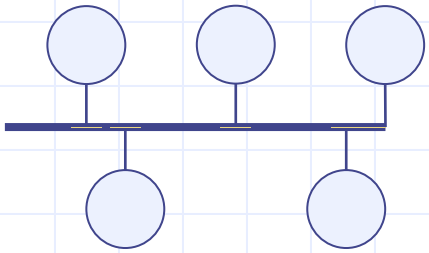
	Количество узлов	Принцип адресации	Адрес компьютера
Локальная	Ограничено (несколько сотен)	Физическая, адрес компьютера – адрес сетевого адаптера	Адрес компьютера уже имеется, компьютер может узнать своих соседей путем опроса
Глобальная	Не ограничено (миллионы), географически разнесены	Логическая адресация (IP-адрес)	Нельзя узнать, кто есть в сети

Топология сети

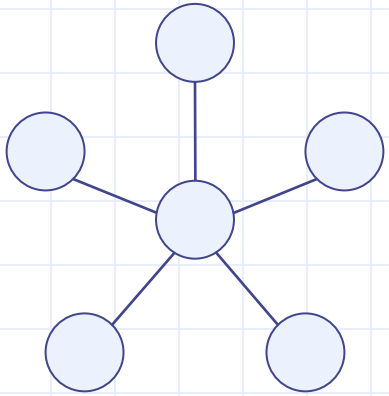
- Топология – граф связей между узлами сети.
  - полно-связные, неполно-связные.

- Неполно-связные:

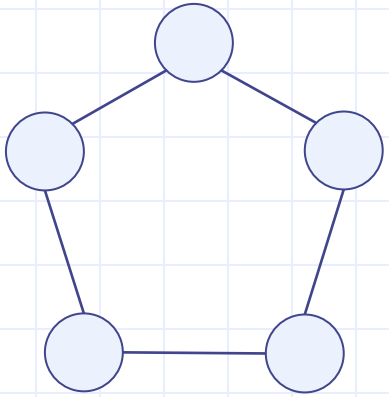
Шина



Звезда

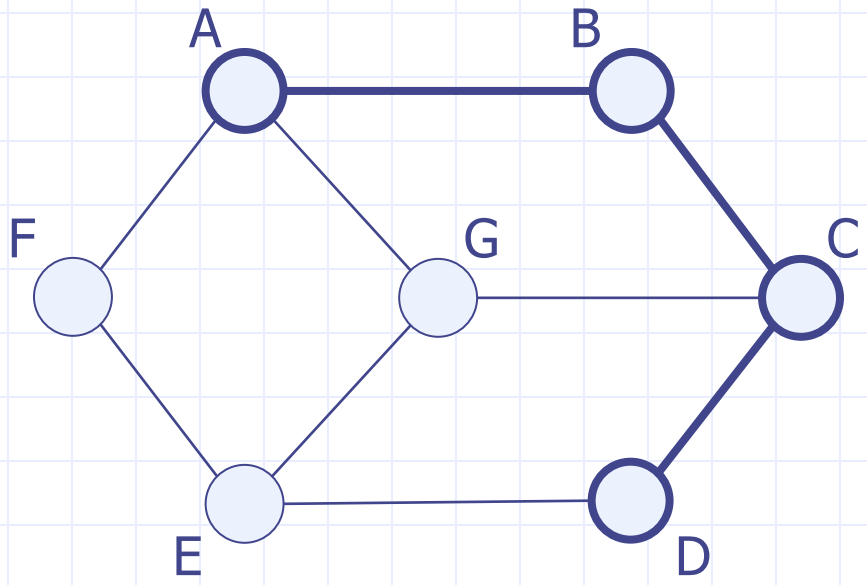


Кольцо



Сети с коммутацией каналов

- Канал устанавливается по всему маршруту от начального узла к конечному.
- После установки соединения скорость передачи информации высока, но каналы связи используются крайне неэффективно.

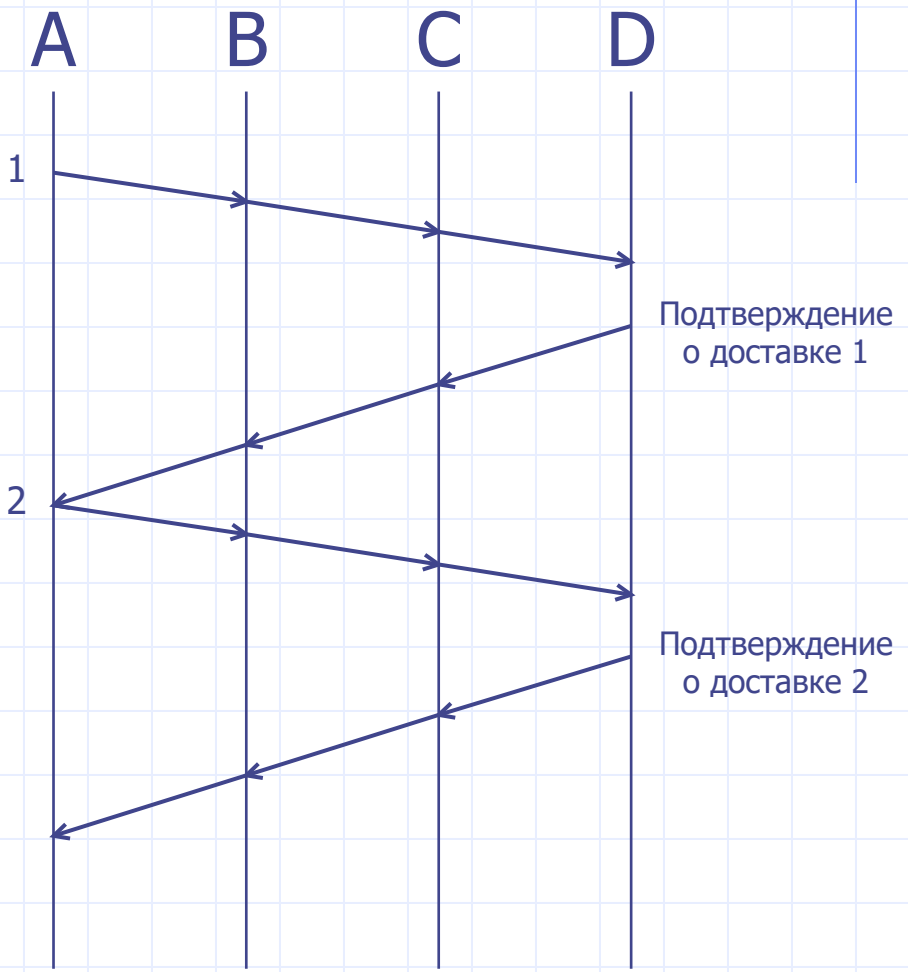


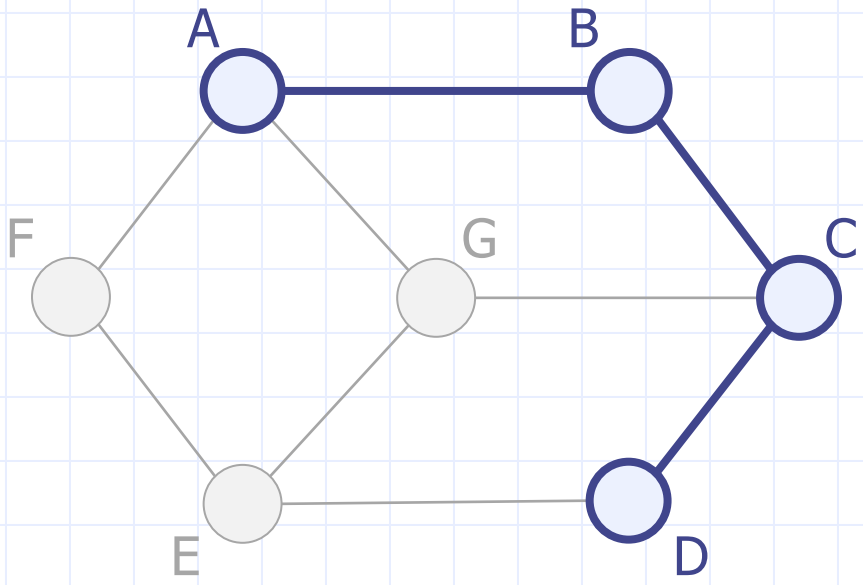
$A \rightarrow D$

$A \rightarrow B \rightarrow C \rightarrow D$

Сети с коммутацией каналов

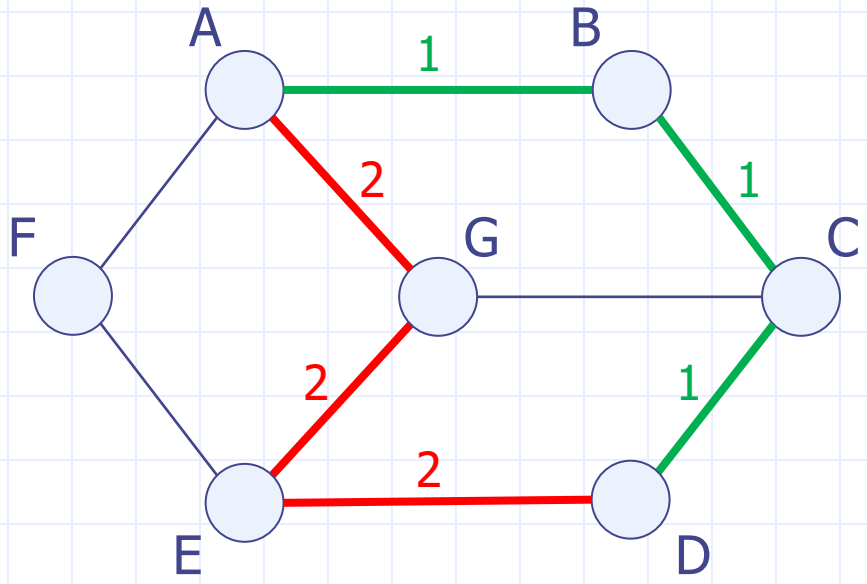
- Во время передачи канал занят.
- До отправки следующего блока необходимо дождаться подтверждения о доставке предыдущего.





Сети с коммутацией пакетов

- Канал устанавливается только к соседнему узлу.
- По каналу передаётся пакет данных, в который включается информация о конечном узле. Далее за передачу этого пакета отвечает узел, который его получил. После передачи пакета соединение может быть разорвано.
- Каждый узел, получив данные, буферизирует их и передаёт дальше, после чего канал может быть вновь использован.



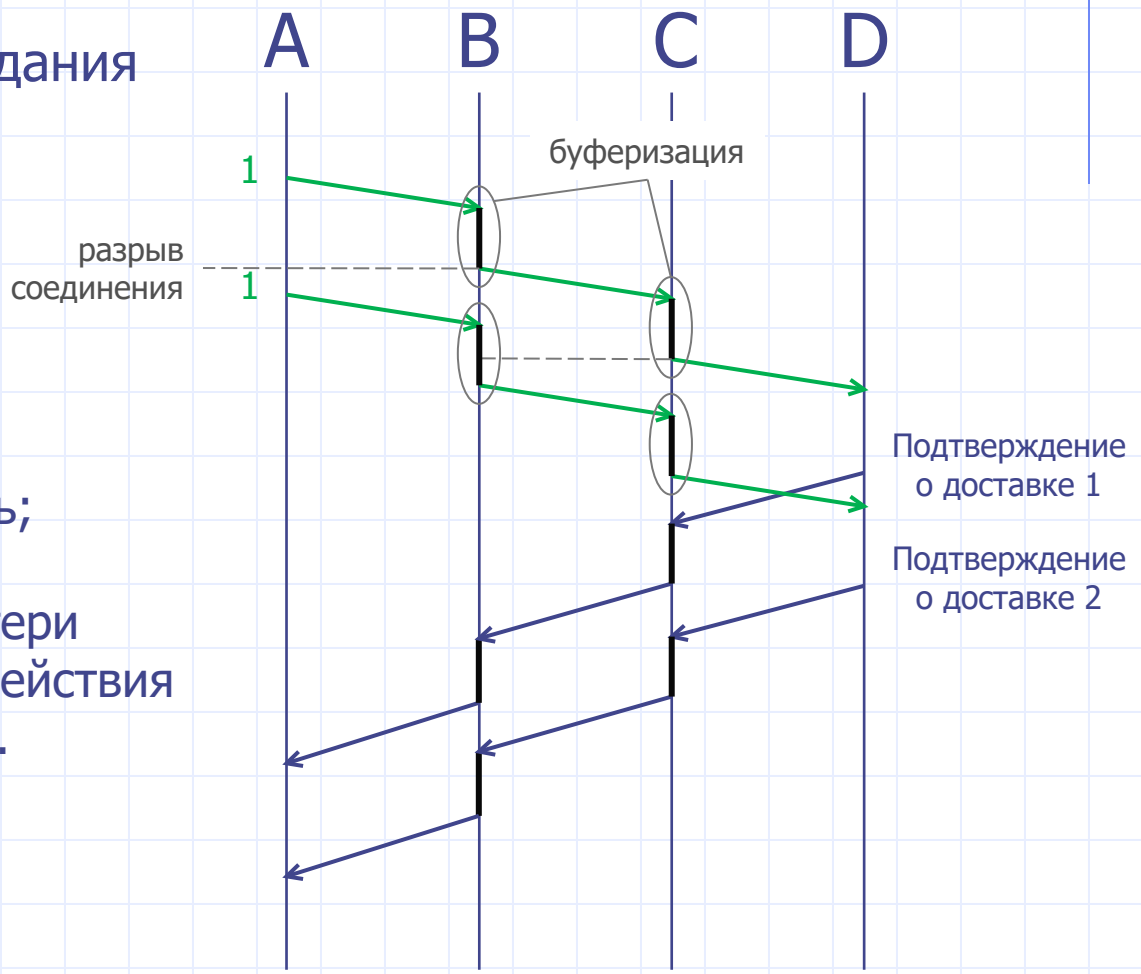
$A \rightarrow D$

1 $A \rightarrow B \rightarrow C \rightarrow D$

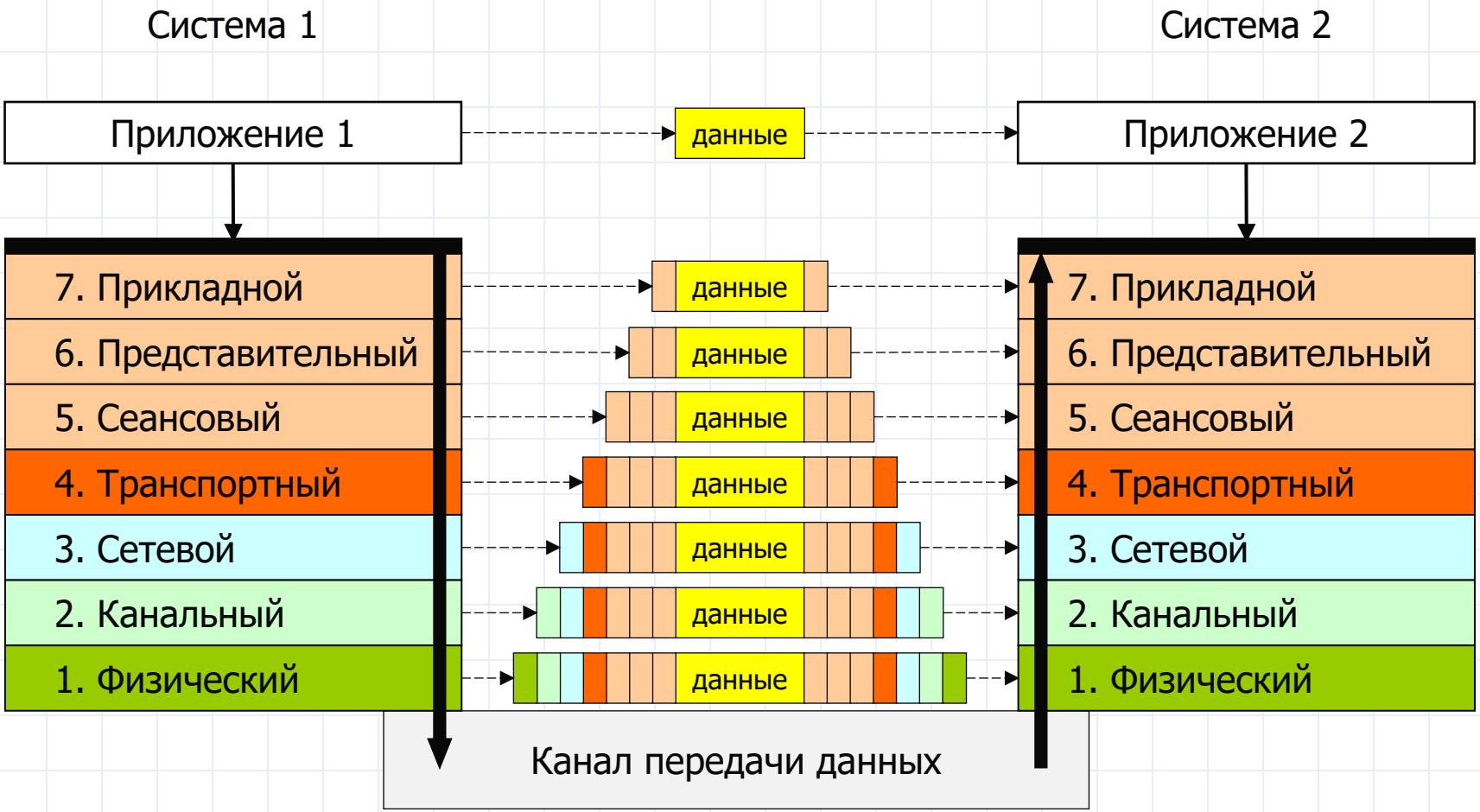
2 $A \rightarrow G \rightarrow E \rightarrow D$

Сети с коммутацией пакетов

- Следующий блок передаётся без ожидания того, что получен предыдущий блок.
- Пакеты идентифицируются.
- Преимущества:
  - меньшая стоимость;
  - выше надежность;
  - позволяют без потери контекста взаимодействия изменять маршрут.

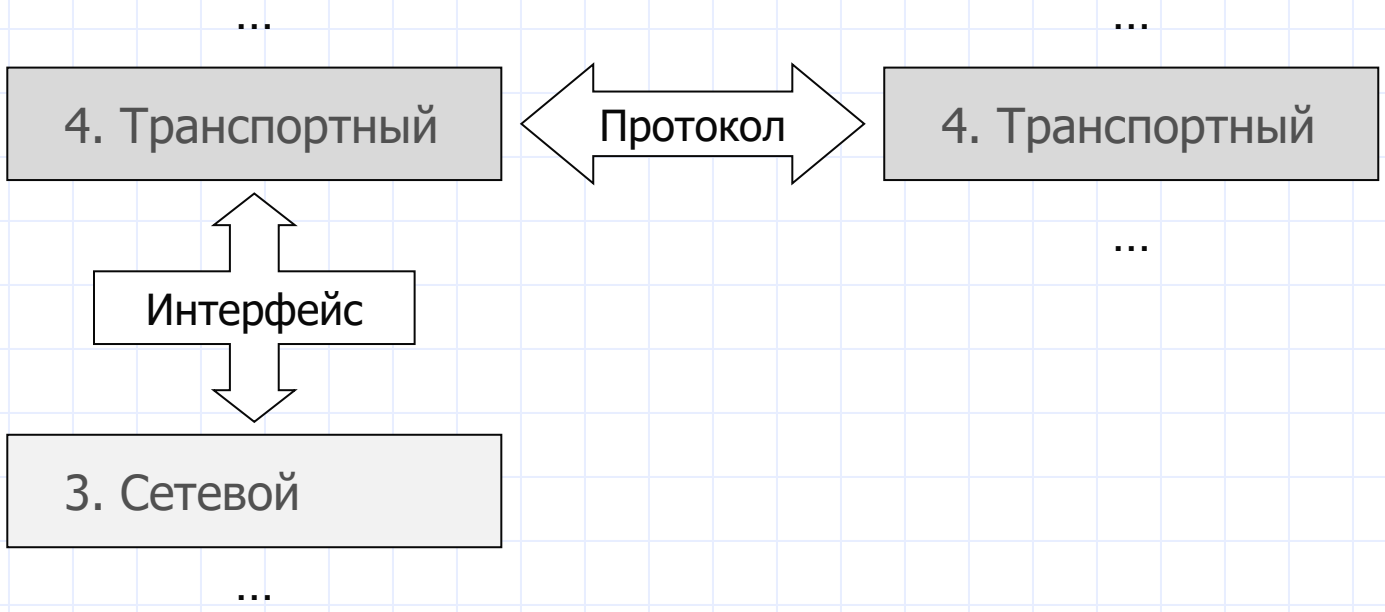


Семиуровневая модель взаимодействия открытых систем – OSI



Протокол и интерфейс

- Протокол – правила и стандарты взаимодействия между узлами на одном уровне сети.
- Интерфейс – правила и стандарты взаимодействия между уровнями сети на одном узле.



Физический уровень OSI

- Отвечает за создание физической связи между узлами в сети (кабели, разъемы, сигналы).
- Стандартизируются параметры сигналов, способы кодирования сигналов, виды сигналов.
- Передача битов.

- Аналогия:

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Канальный уровень OSI

- Определяет логическую топологию сети, правила получения доступа к среде передачи данных.
- Отвечает за физическую адресацию устройств сети, управление передачей и службу соединений.
- Аналогия:

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Сетевой уровень OSI

- Отвечает за доставку сообщений между узлами разных логических сетей.
- Отвечает за логическую адресацию сетевых устройств и маршрутизацию сообщений.

- Аналогия: **IP**

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Транспортный уровень OSI

- Отвечает за надежную передачу сообщений между узлами сети (программами-абонентами на узлах).
- Добавляет к логической адресации понятие порта – для идентификации программ-абонентов.
- Обеспечивает разбиение сообщений на пакеты, передачу их по сети и сборку этих пакетов в правильном порядке.
- Обеспечивает надежность доставки за счёт передачи уведомлений (квитанций) о доставке и повторной передачи утерянных пакетов.
- Аналогия: **TCP**, **UDP**

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Сеансовый уровень OSI

- Обеспечивает сеансы связи между устройствами, запрашивающими и поставляющими услуги.
- Отвечает за установку и поддержку соединений, а также за синхронизацию и управление диалогом между абонентами.
- Помогает верхним уровням идентифицировать доступный сетевой сервис и соединиться с ним.

- Аналогия: **Sockets**

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Представительный уровень OSI

- Основная задача — преобразование данных во взаимно согласованные форматы (синтаксис обмена), понятные всем сетевым приложениям и компьютерам, на которых работают приложения.
- Обеспечивает компрессию и декомпрессию данных, а также шифрование данных.

- Аналогия: **TLS**, **SSL**, **MIME**

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Прикладной уровень OSI

- Предоставляет конкретные сетевые службы.
- Шесть нижних уровней объединяют задачи и технологии, обеспечивающие общую поддержку сетевой службы, в то время как прикладной уровень обеспечивает протоколы, необходимые для выполнения конкретных функций сетевой службы.

- Аналогия: **HTTP**, **FTP**, **Telnet**, **SMTP**, **POP3**, **IMAP**

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

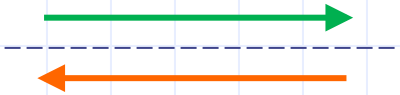
Физический уровень OSI – Уровень кабелей

- Виды связи
  - симплексная, дуплексная, полудуплексная
- Виды сигналов
  - аналоговые, цифровые
- Виды кодирования аналоговых сигналов
  - частотная, амплитудная, импульсная модуляция
- Виды кодирования цифровых сигналов
  - TTL, NRZ, RZ, Манчестерское кодирование
- Среда передачи данных (СПД)
  - коаксиальный кабель, витая пара, оптоволокно, эфир
- Физические топологии
  - полно-связная; неполно-связные: шина, звезда, кольцо, гибридная

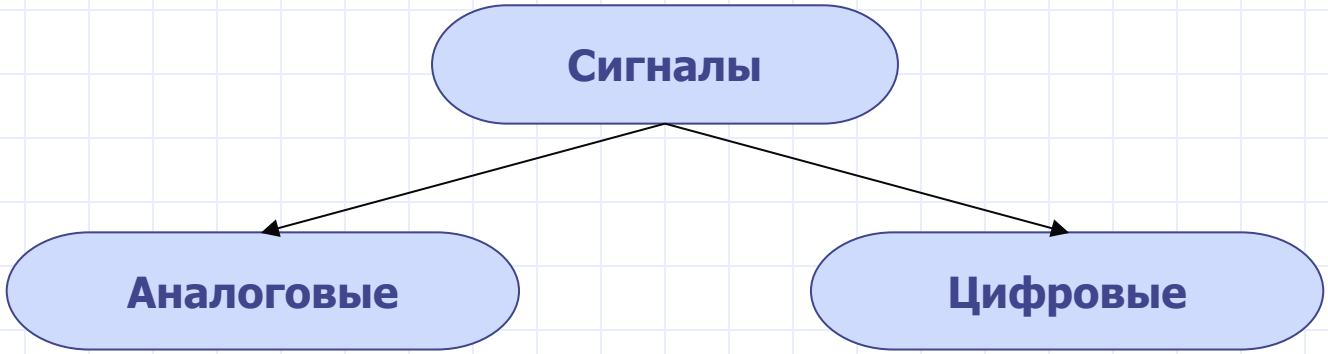
7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Виды связи

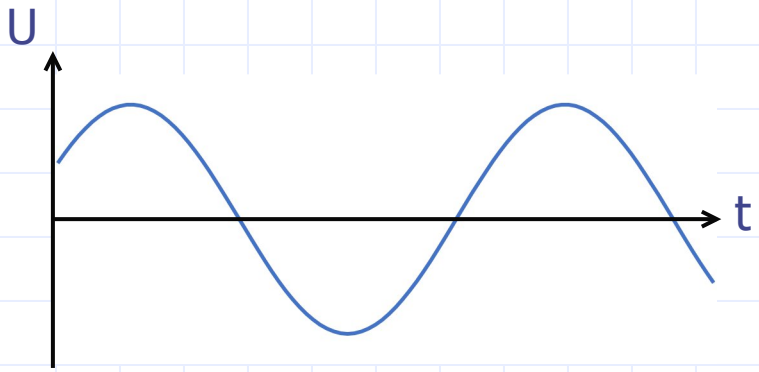
Связь

Симплексная	**Дуплексная**	**Полудуплексная**
Информация передаётся только в одном направлении.	Одновременная двухсторонняя связь (двукратный расход кабеля).	Поочерёдные приём и передача по одному каналу (правила общения).
		
Телевидение	Телефонная связь	Рация
		

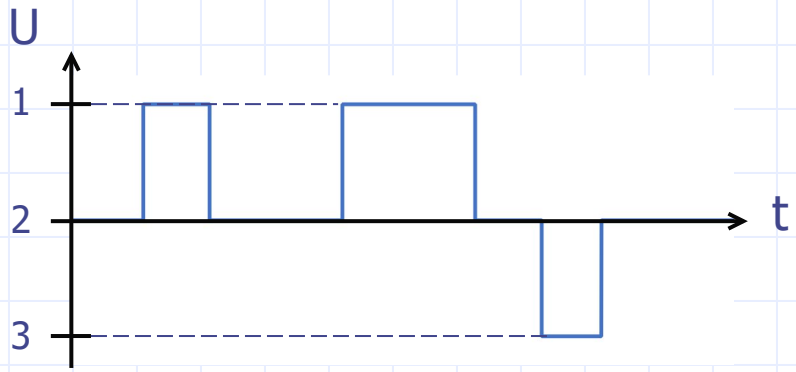
Виды сигналов



Непрерывное изменение физической величины (в заданных пределах).

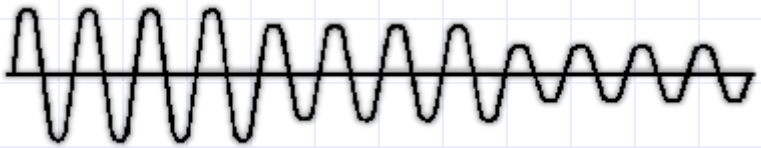


Изменение физической величины в ограниченном наборе значений.

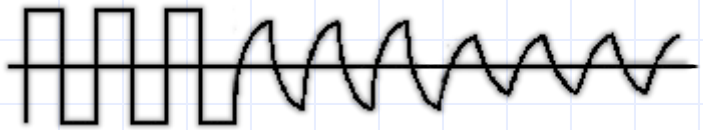


Затухание сигнала

- Затухание аналогового сигнала –
  - уменьшение амплитуды; восстанавливается усилением.

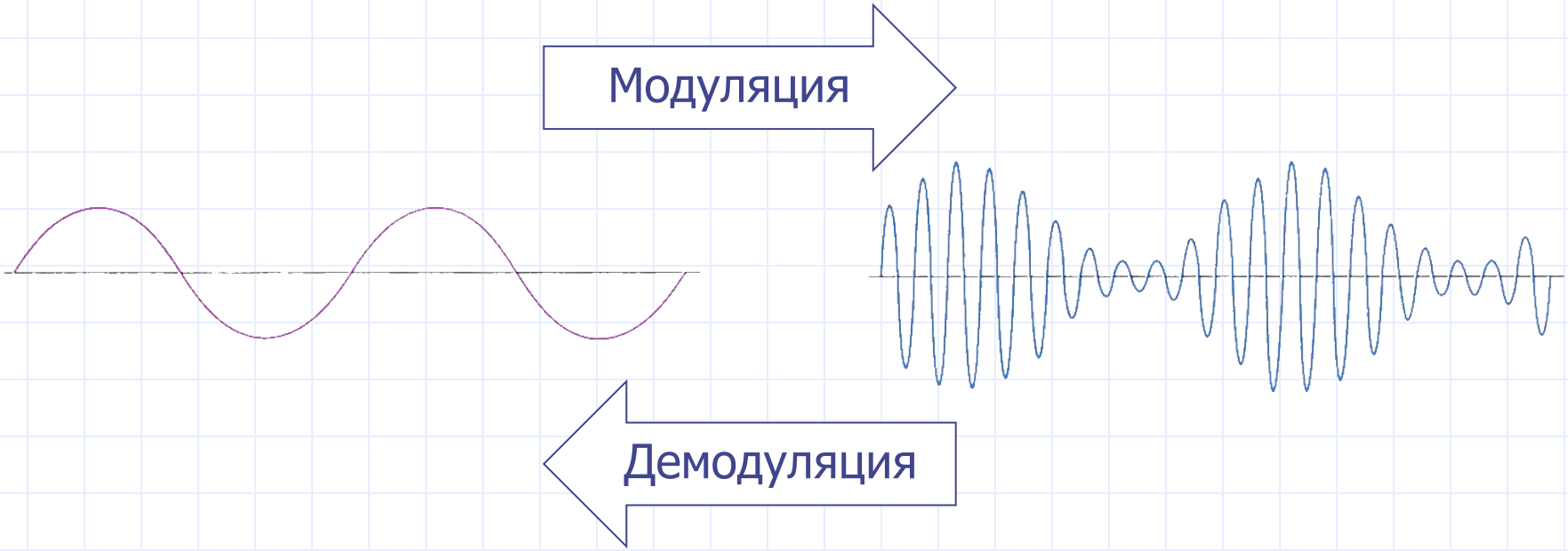


- Затухание цифрового сигнала –
  - уменьшение амплитуды и нарушение формы; восстанавливается повторением.



Кодирование аналоговых сигналов

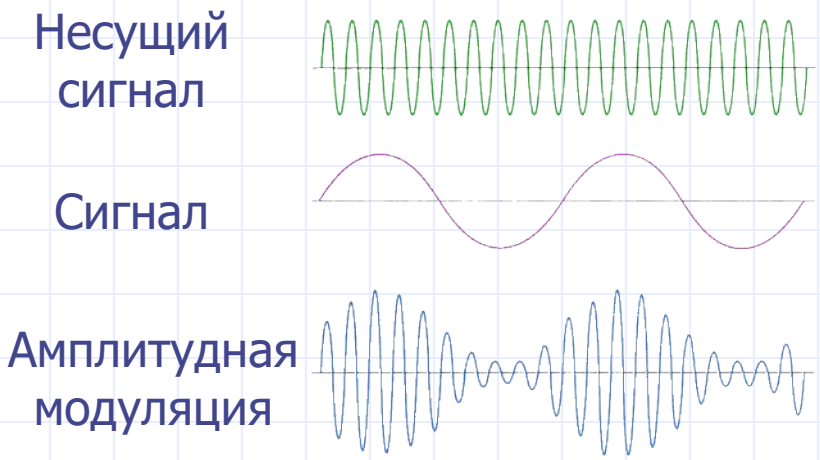
- Модуляция –
  - изменение какой-либо физической величины по определённому закону.
- Демодуляция –
  - восстановление исходного сигнала.



Амплитудная модуляция (АМ)

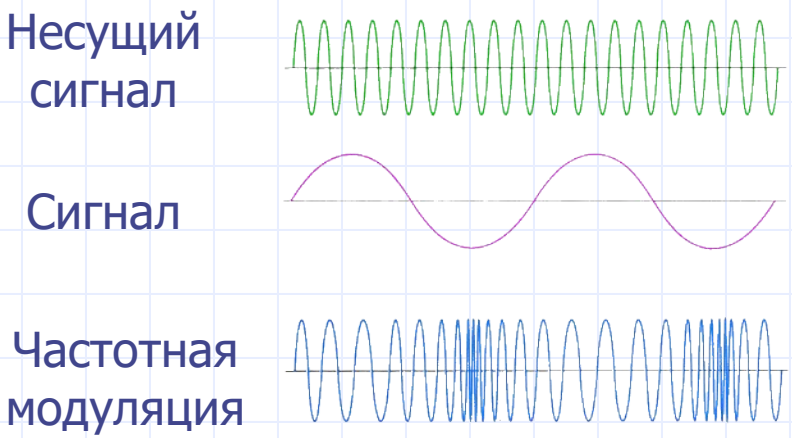
АМ – изменение амплитуды несущего сигнала.

- Применение: радиосвязь на длинных волнах (ДВ).
- Достоинства:
  - узкая ширина спектра;
  - простота кодирования.
- Недостатки:
  - требуется высокая мощность передатчика;
  - низкая помехоустойчивость.



Частотная модуляция (ЧМ)

ЧМ – изменение частоты несущего сигнала при той же амплитуде.

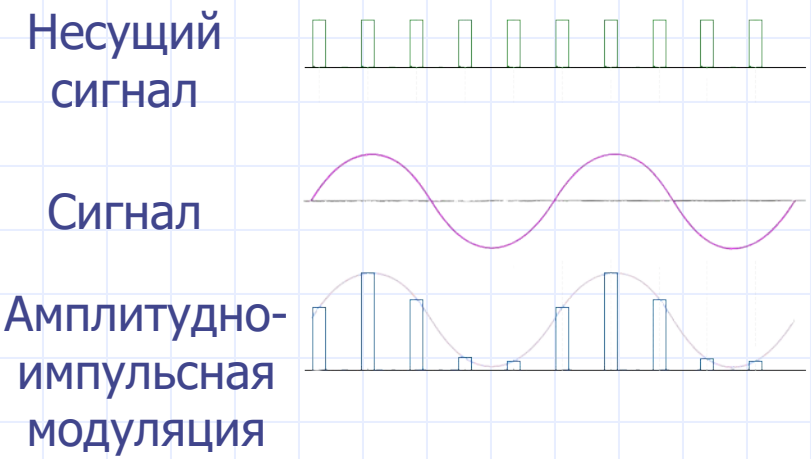


- Применение: радиосвязь на КВ и УКВ.
- Достоинства:
  - высокая помехоустойчивость;
  - эффективное использование мощности передатчика;
  - сравнительная простота получения.
- Недостатки:
  - широкий спектр сигнала.

Импульсная модуляция (ИМ)

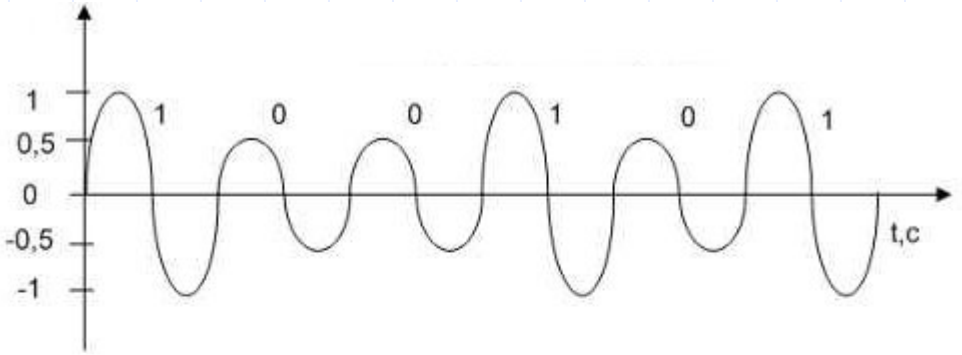
ИМ – изменение амплитуды, частоты, фазы периодических импульсов.

- Применение: радиорелейная и тропосферная связь.
- Достоинства:
  - многоканальность;
  - высокая помехоустойчивость.
- Недостатки:
  - сложность аппаратуры.

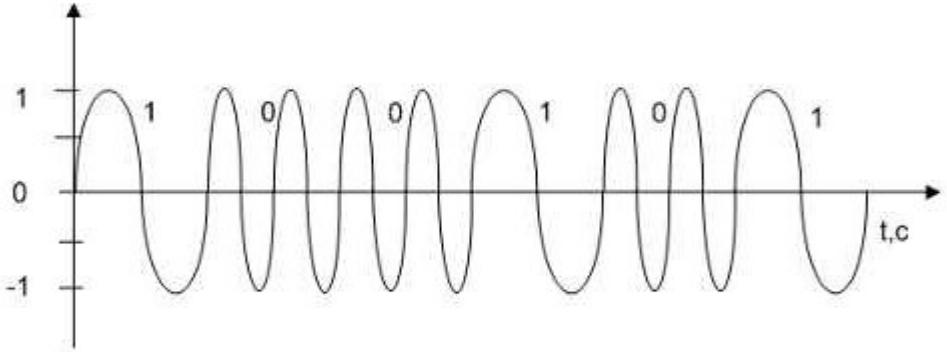


Передача цифрового сигнала с помощью аналогового

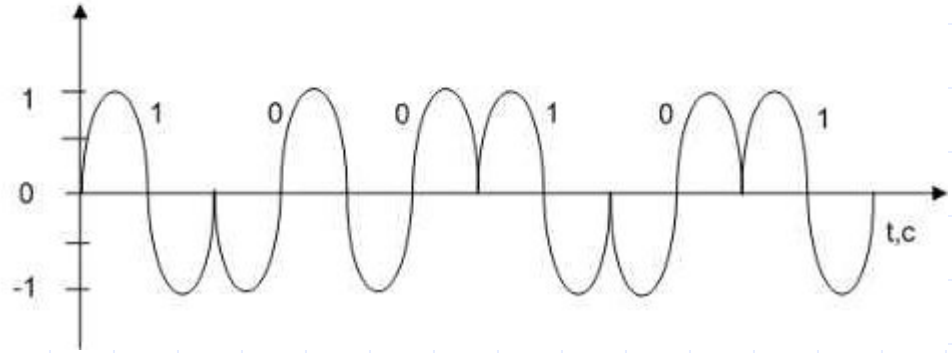
Амплитудное кодирование



Частотное кодирование

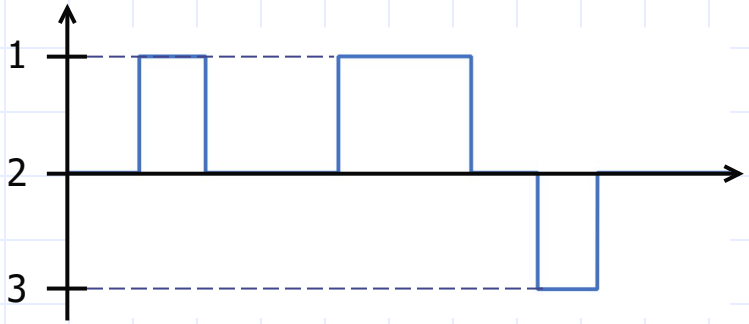


Фазовое кодирование



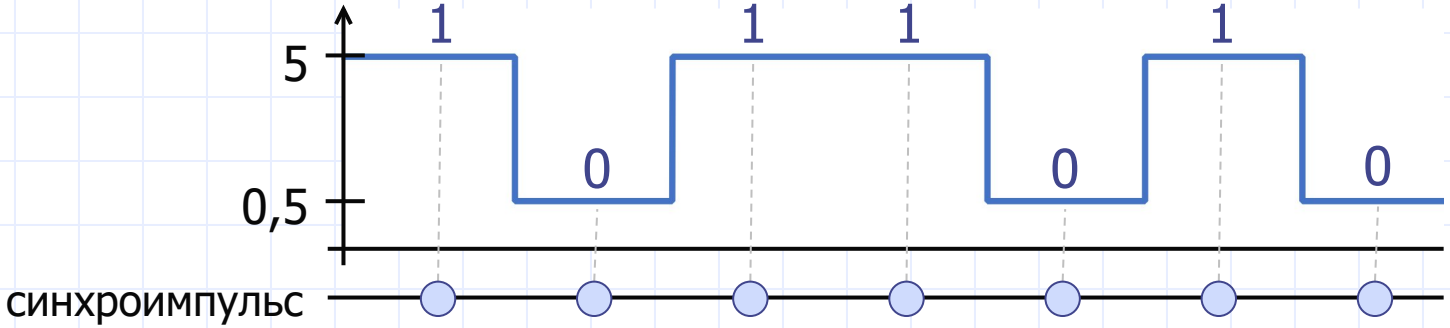
Кодирование цифровых сигналов

- TTL – Transistor-Transistor Logic
- NRZ – Non-Return to Zero
- RZ – Return to Zero
- Манчестерский код



TTL – Transistor-Transistor Logic

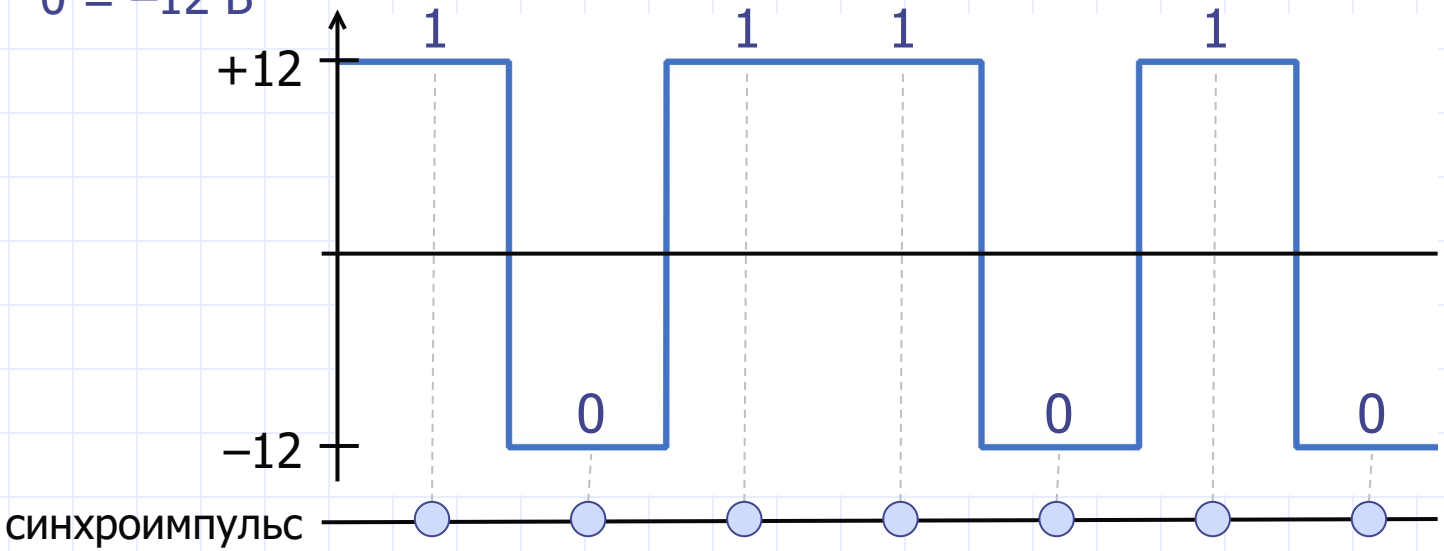
- Кодирование сигнала уровнем напряжения:
  - 1 = 4,5 – 5,0 В
  - 0 = 0,2 – 0,5 В



- Тактовый генератор + линия связи для синхроимпульса.
- Сигнал быстро затухает уже на расстоянии 5 – 10 м.
- Применяется в микросхемах.

NRZ – Non-Return to Zero

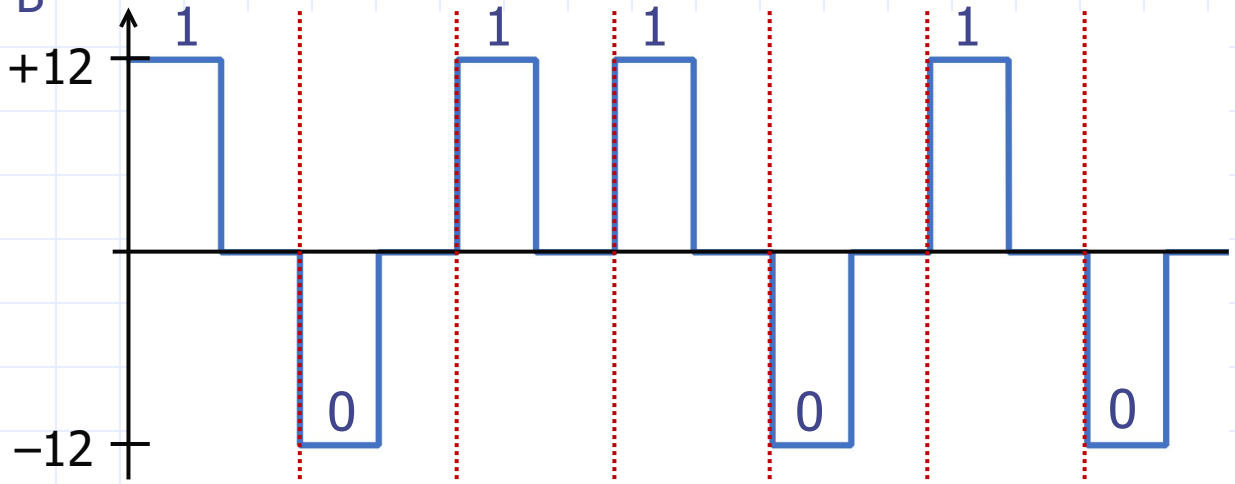
- Кодирование сигнала уровнем напряжения:
  - 1 = +12 В
  - 0 = –12 В



- Тактовый генератор + линия связи для синхроимпульса.
- Сигнал затухает медленнее, устойчив к помехам.

RZ – Return to Zero

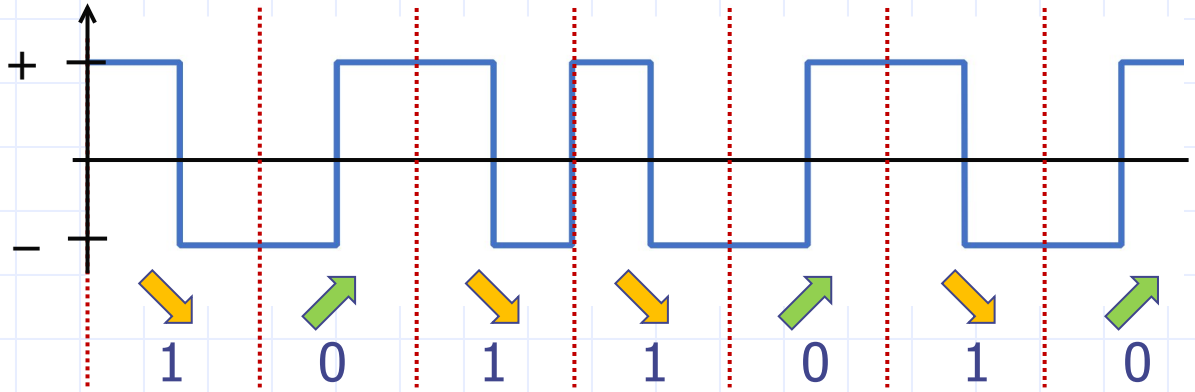
- Кодирование сигнала уровнем напряжения:
  - 1 = +12 В
  - 0 = –12 В



- Самосинхронизирующийся.
- Высокая помехозащищённость.
- Выше частота – выше скорость передачи.

Манчестерский код

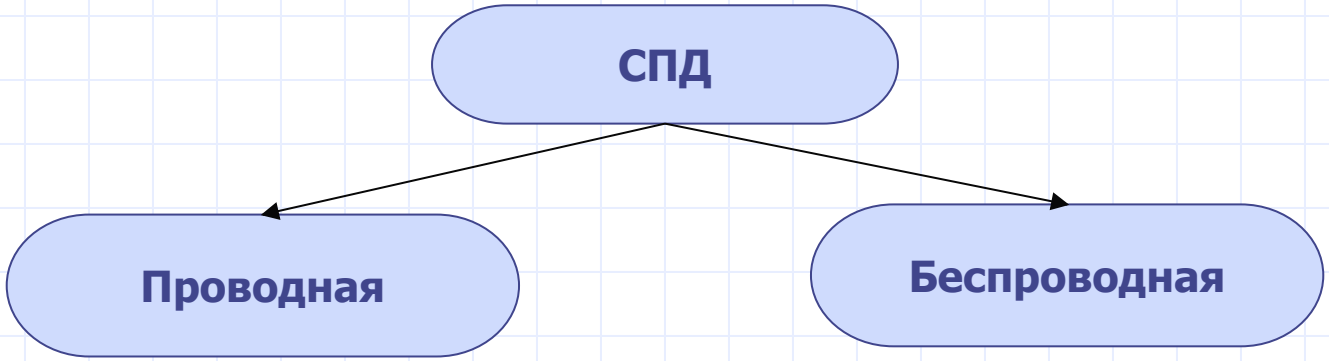
- Кодирование сигнала моментом перехода от «+» к «–»:
  - 1 = «+» → «–»
  - 0 = «–» → «+»



- Самосинхронизирующийся.
- Высокая помехозащищённость.
- Используется в локальных сетях (технология Ethernet).

Среда передачи данных (СПД)

СПД – физическая среда, по которой распространяется сигнал, использующийся для передачи данных.



- коаксиальный кабель
- витая пара
- оптоволокно

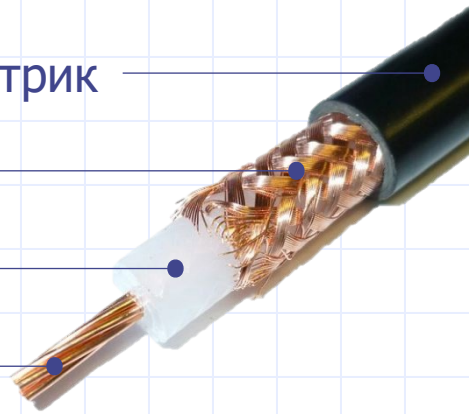
- эфир

Коаксиальный кабель RG-58

- Сигнал передаётся по центральной жиле.
- Оплётка – экран для защиты сигнала от наводок.

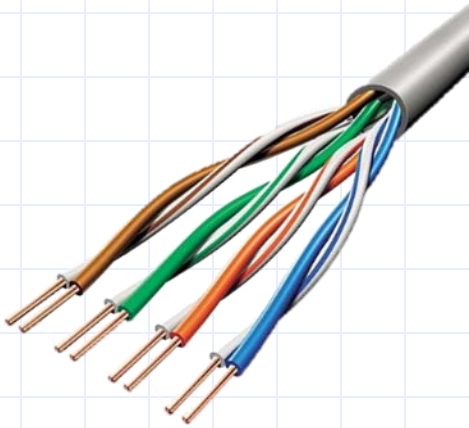
- Эффективная длина:
  - тонкий – до 25 м
  - толстый – до 500 м

- Скорость ≈ 10 Мбит/с



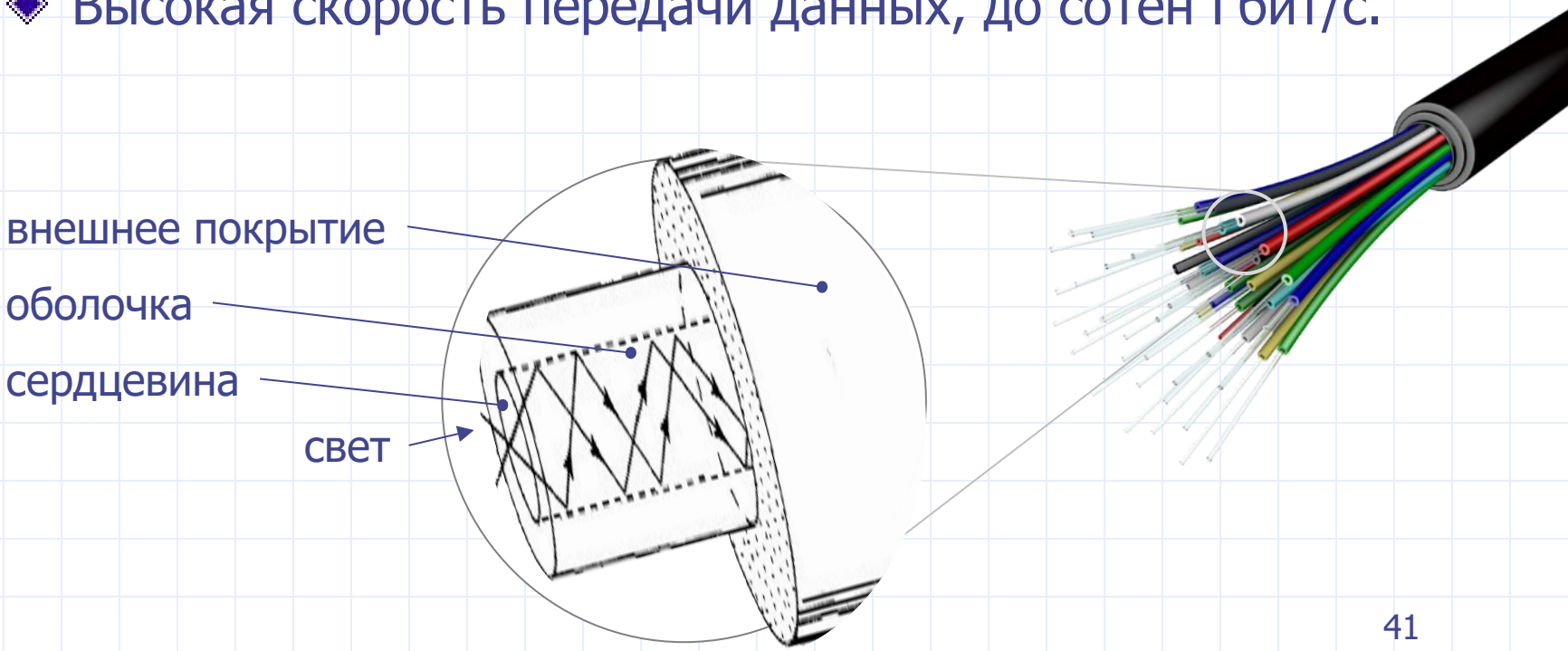
Витая пара

- Одна или несколько свитых пар медных проводов.
- Одновременная передача сигнала по паре проводов позволяет устранить внешние наводки.
- Скручивание позволяет устранить перекрёстные помехи между парами.
- 7 категорий витой пары, определяют:
  - количество проводов;
  - число витков на единицу длины;
  - степень экранирования;
  - в итоге – скорость передачи данных:
    - категория 5e обеспечивает скорость до 1 Гбит/с;
    - категория 7 обеспечивает скорость до 10 Гбит/с.



Оптоволокно

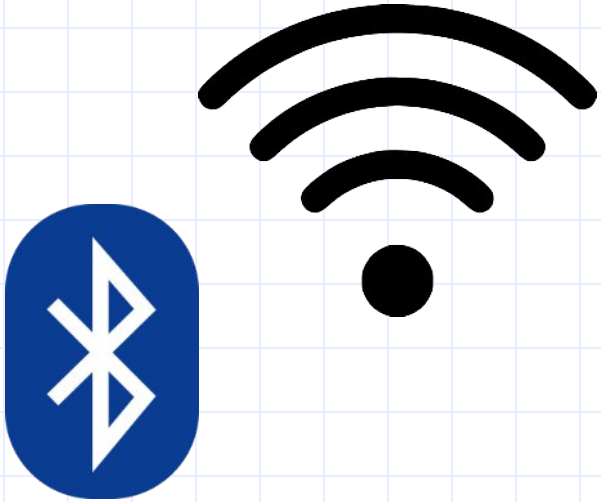
- Носитель сигнала – пучок света.
- Сердцевина с высоким коэффициентом преломления, оболочка с низким коэффициентом преломления.
- Высокая помехозащищённость и секретность.
- Высокая скорость передачи данных, до сотен Гбит/с.



Эфир

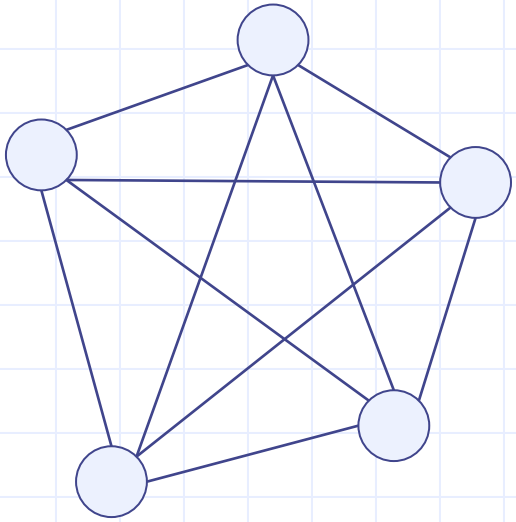
- Носитель сигнала – радиоволны в заданном диапазоне частот.
- Чем выше частота, тем выше скорость передачи, но уменьшается дальность распространения сигнала и проникающая способность.

- Bluetooth
  - 2,4 ГГц до 2 Мбит/с
- Wi-Fi
  - 2,4 ГГц до 300 Мбит/с
  - 5 ГГц выше 1 Гбит/с



Полно-связная сеть

Каждый узел связан со всеми узлами сети.

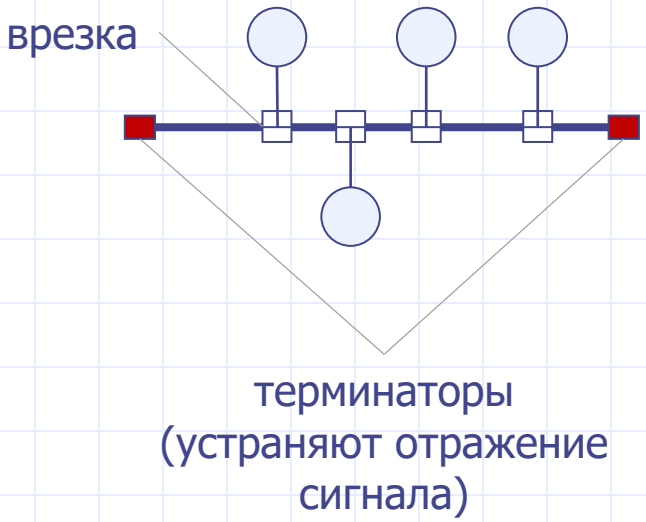


- Достоинства:
  - высокое быстродействие;
  - нет конфликтов при передаче данных;
  - надёжность.
- Недостатки:
  - число узлов в сети ограничено количеством доступных портов на каждом узле;
  - высокий расход кабеля;
  - высокая стоимость.

На практике не используется.

Физическая топология шина

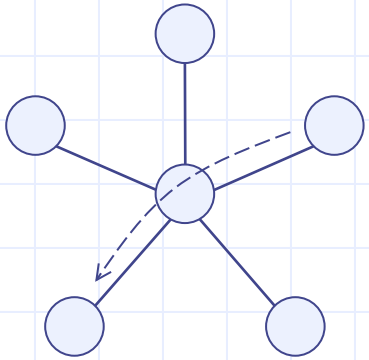
Все узлы подключены к общему каналу передачи данных (коаксиальный кабель).



- Полудуплексная связь.
- Достоинства:
  - простота реализации широковещательной передачи;
  - простота подключения (при наличии врезки).
- Недостатки:
  - относительно низкая скорость;
  - подключение без врезки = остановка сети;
  - разрыв кабеля – Х.

Физическая топология звезда

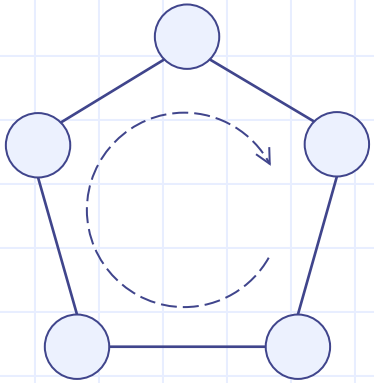
Сигнал предаётся между узлами сети через один центральный узел.



- Полудуплексная или дуплексная связь.
- Достоинства:
  - разрыв кабеля не выводит из строя всю сеть.
- Недостатки:
  - надёжность зависит от центрального узла;
  - высокий расход кабеля.

Физическая топология кольцо

Сигнал передаётся в одном направлении между узлами по кругу.



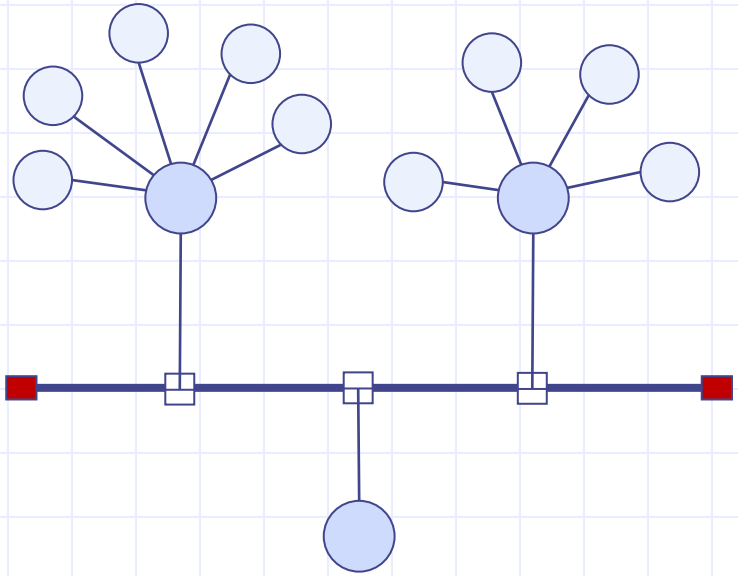
- Симплексная связь между узлами, дуплексная связь по всей сети.

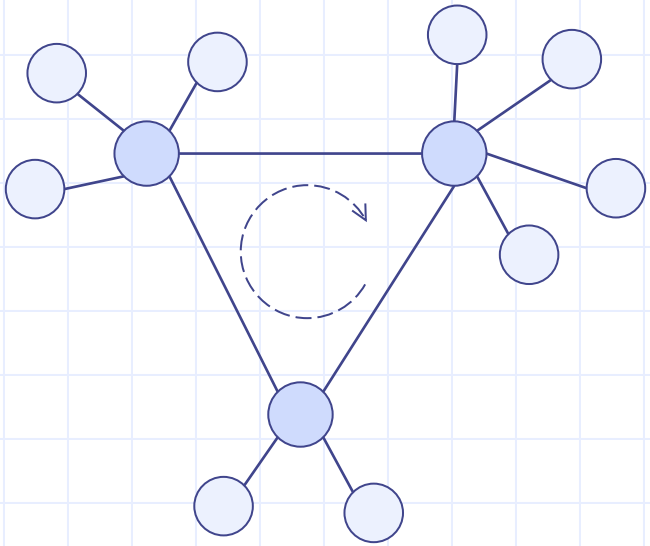
- Достоинства:
  - низкий расход кабеля;
  - возможность одновременной передачи данных в разных сегментах сети.
- Недостатки:
  - низкая надёжность;
  - подключение узла = остановка сети;
  - разрыв кабеля – X.

Гибридная топология

Комбинация шины, звезды, кольца.

- Применяется для связывания сетей разных топологий.
- Часто используется «кольцо из звёзд».



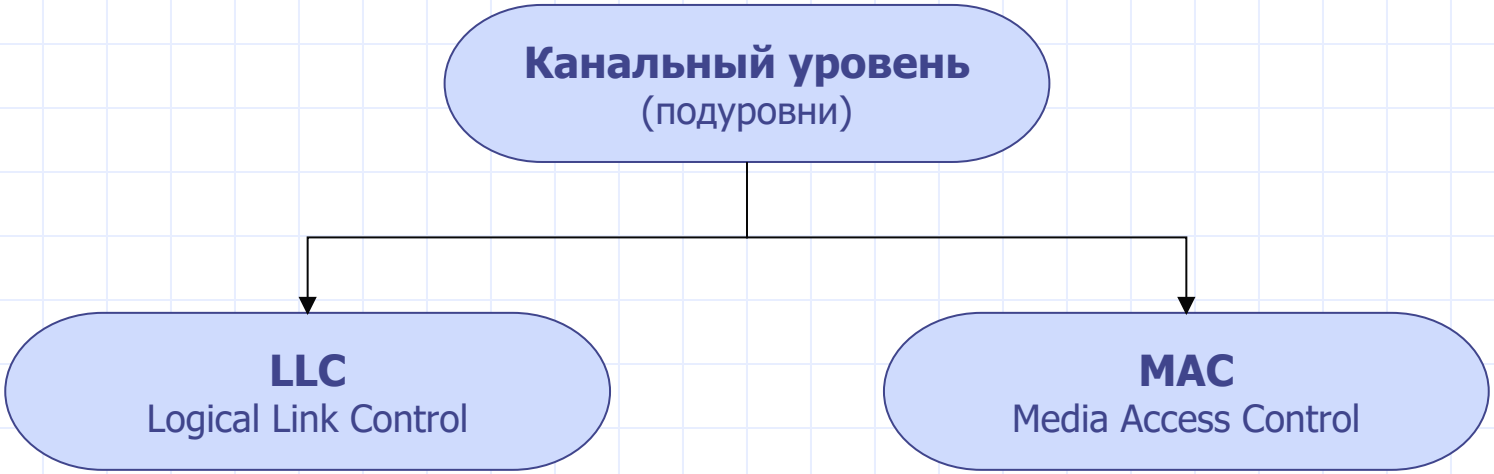


Канальный уровень OSI – Уровень сетевой платы

- Физическая адресация устройств (MAC адреса)
- Логическая топология сети
- Правила доступа и совместного использования СПД несколькими парами абонентов
- Передача пакетов (кадров) данных (англ. frame)
- Проверка ошибок физического уровня

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Подуровни канального уровня



Подуровень управления логическими связями

Передача данных:

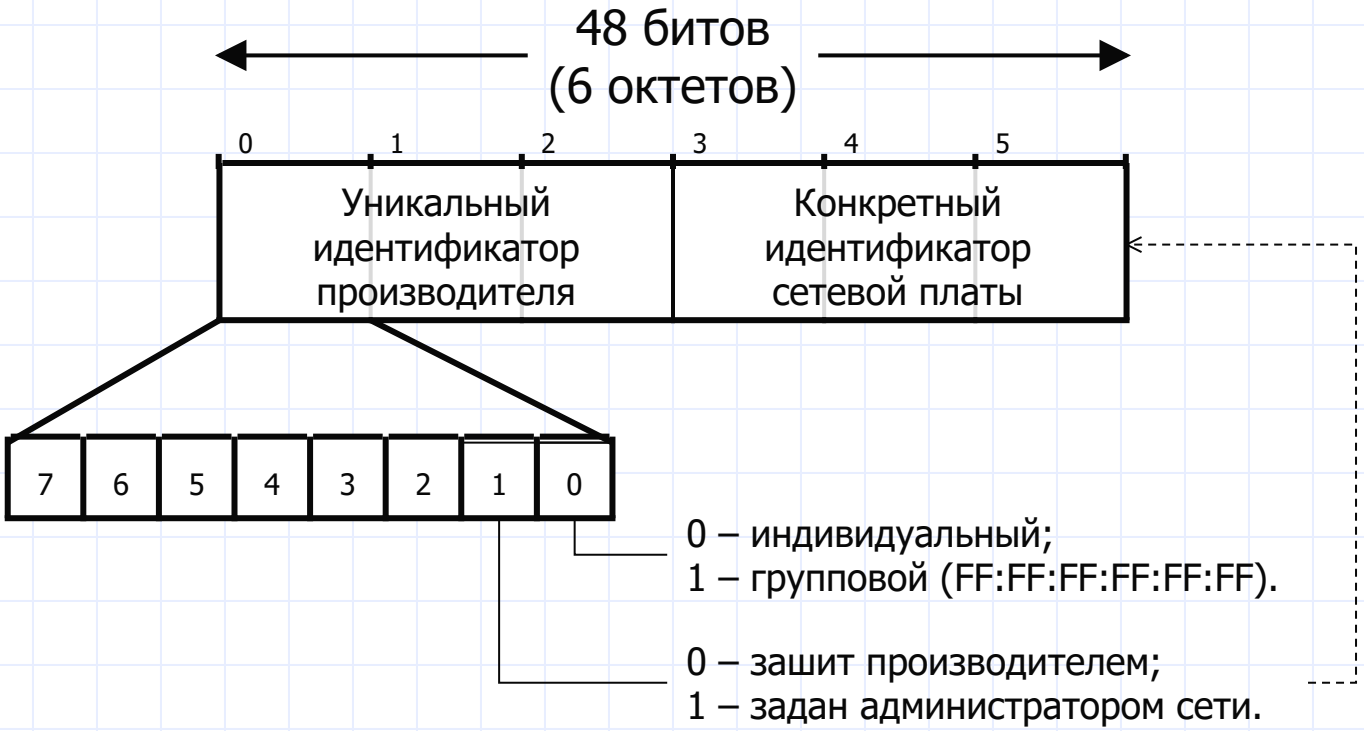
- с установкой соединения;
- без установки соединения, с подтверждением;
- без установки соединения, без подтверждения.

Подуровень управления доступом к среде передачи данных

- физическая адресация узлов;
- способы доступа к СПД.

MAC-адрес

MAC-адрес – физический адрес устройства; уникальный в пределах локальной сети, зашитый производителем или заданный администратором сети.



https://hwaddress.com/

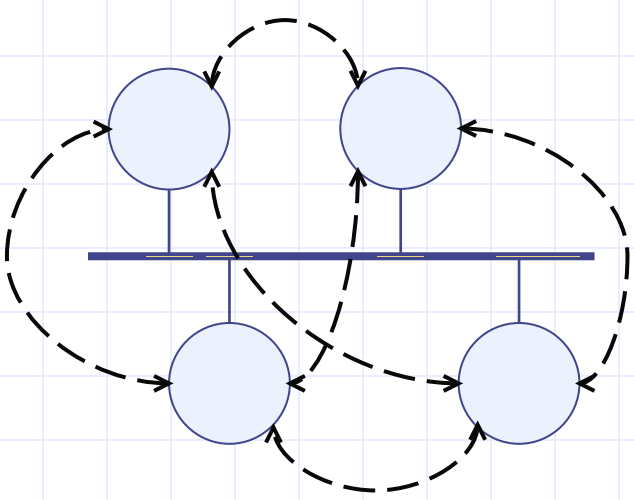
Структура кадра данных

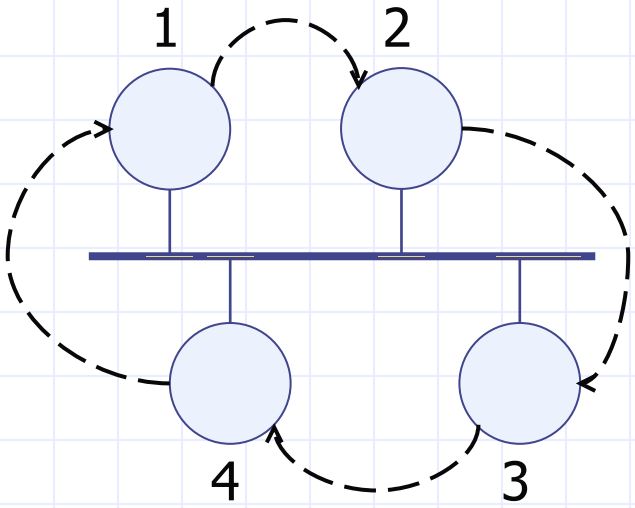
Преамбула	MAC-адрес получателя	MAC-адрес отправителя	Тип / Размер	Содержимое	Контрольная сумма
8 октетов	6 октетов	6 октетов	2 октета	46 – 1500 октетов	4 октета

- Сетевая плата читает кадры с соответствующим MAC-адресом получателя.
- После вычитывания кадра сетевая плата пересчитывает его контрольную сумму и сравнивает с принятой:
  - если значения совпадают, то пакет принимается.
  - если значения отличаются, то пакет отбрасывается.

Логическая топология сети

- Граф вычислительной сети, в котором ребра соответствуют путям передачи данных между узлами (без учёта физических связей).





СПД с коллективным доступом

- Варианты СПД:
  - коаксиальный кабель с Т-образными врезками;
  - звезда с ретранслятором в центре;
  - эфир.

- Коллизия –
  - наложение нескольких сигналов в общей СПД при попытке нескольких узлов осуществить передачу одновременно.
- Домен коллизий –
  - сегмент сети с общей СПД, в котором одновременная передача данных несколькими узлами приводит к коллизии.

Методы доступа к СПД

- CSMA/CD – Carrier Sense Multiple Access with Collision Detection
  - множественный доступ с прослушиванием несущей и обнаружением коллизий.
- CSMA/CA – Carrier Sense Multiple Access with Collision Avoidance
  - множественный доступ с прослушиванием несущей и избеганием коллизий.
- Логическое «кольцо» на физической «шине»
  - метод с передачей маркера в топологии «шина».
- Логическое «кольцо» на физическом «кольце»
  - метод с передачей одного или нескольких маркеров.

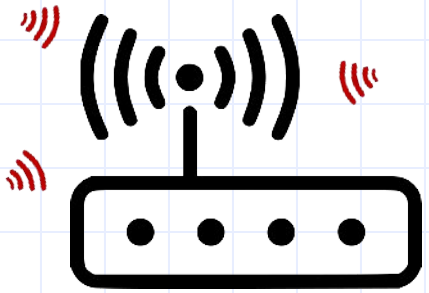
CSMA/CD – обнаружение коллизий

- Узел прослушивает СПД на предмет занятости. При отсутствии сигнала данных (СПД свободна) узел выполняет передачу данных.
- Во время передачи кадра узел продолжает слушать СПД и если обнаруживает коллизию (слышит не то, что пишет), то останавливает передачу, посылает всем пакет затора (jam signal) и ждёт в течение случайного промежутка времени (интервал отсрочки), находимого с помощью специального алгоритма, после чего повторяет передачу кадра.
- После 16 неудавшихся попыток отправки кадра он отбрасывается – ошибка передачи.
- Используется проводной технологией Ethernet (IEEE 802.3).

CSMA/CD: Диаграмма перехода между состояниями (логика сетевого адаптера)

CSMA/CD в беспроводной сети?

- Передатчик заглушает свой же приёмник – во время передачи трудно установить факт коллизии.

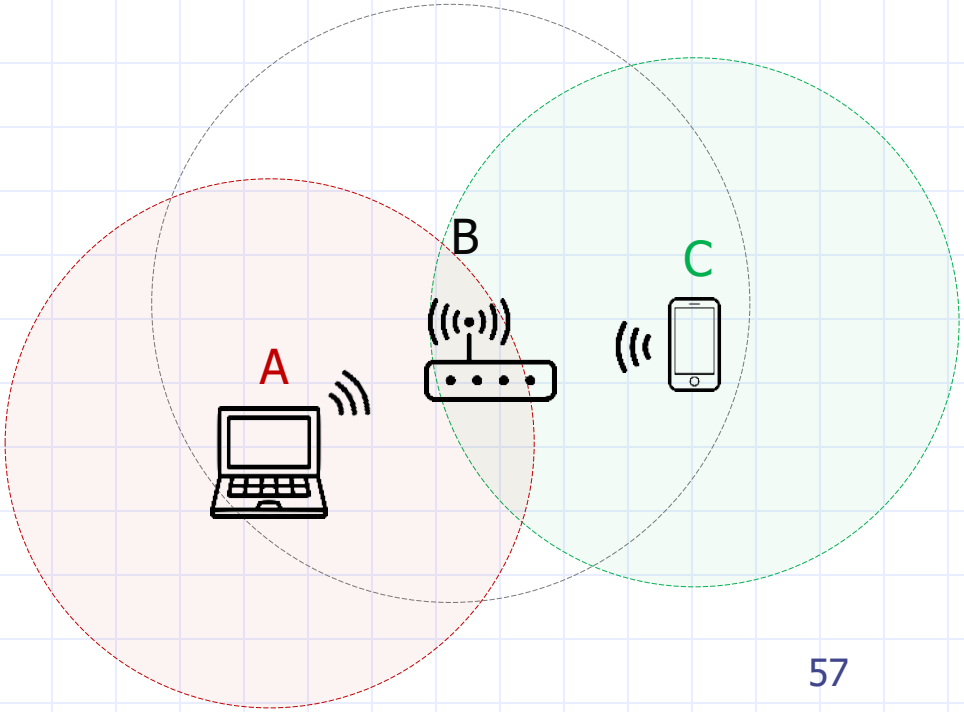


- Проблема скрытого узла:

A ↔ B

B ↔ C

A ? C



CSMA/CA – избегание коллизий

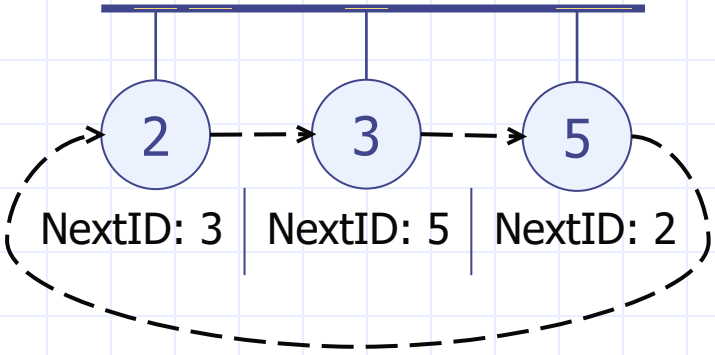
- Узел прослушивает СПД. Когда среда освобождается, узел отсчитывает случайный интервал времени, и если среда всё ещё свободна – передаёт широковещательный служебный кадр RTS (Request to Send).
- Узел назначения, получив RTS-кадр, вырабатывает кадр занятости среды CTS (Clear To Send) с указанием, как долго среда будет занята.
- Передающий узел, получив CTS, начинает передачу данных. Все остальные узлы, получив CTS, ждут указанное время.
- После приёма данных узел назначения отвечает квитанцией о доставке – кадром ACK. Если передающий узел не получил ACK в течение определенного времени, он фиксирует факт коллизии, ждёт в течение случайного промежутка времени (интервал отсрочки) и процесс передачи повторяется.
- Используется беспроводными технологиями Wi-Fi (IEEE 802.11).

CSMA/CA – коллизия возможна?

- Коллизия может возникнуть, если несколько узлов выберут один и тот же случайный интервал отсрочки для передачи кадра RTS. В этом случае кадры искажаются, и CTS от узлов назначения не приходят. Не получив CTS в течение определенного времени, отправители фиксируют факт коллизии, ждут в течение случайных интервалов отсрочки и пытаются передать свои кадры снова.

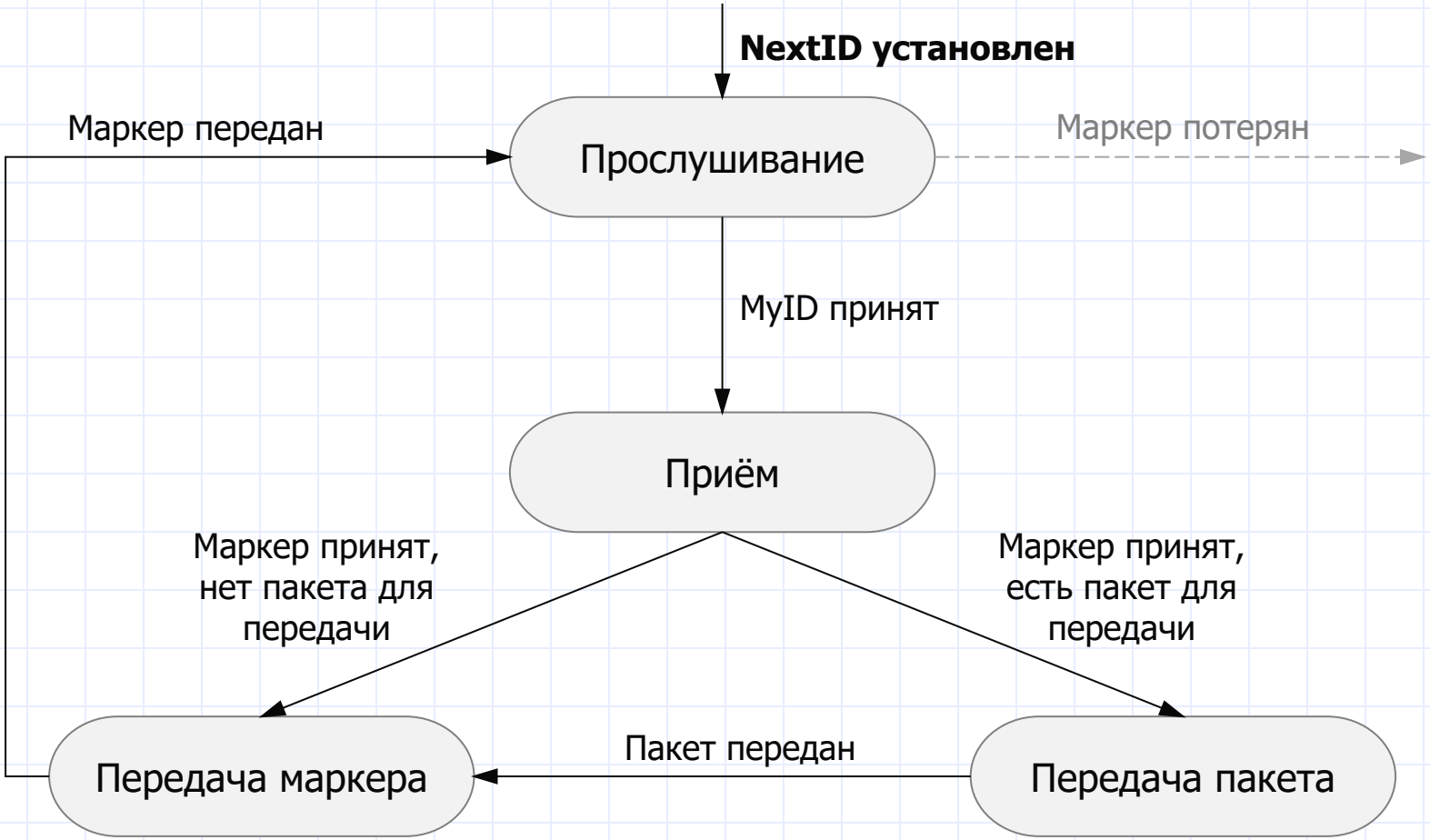
Метод с передачей маркера в топологии «шина»

- Логическое «кольцо».
- Физическая «шина».
- Право передачи переходит от узла к узлу по кольцу – передача маркера.

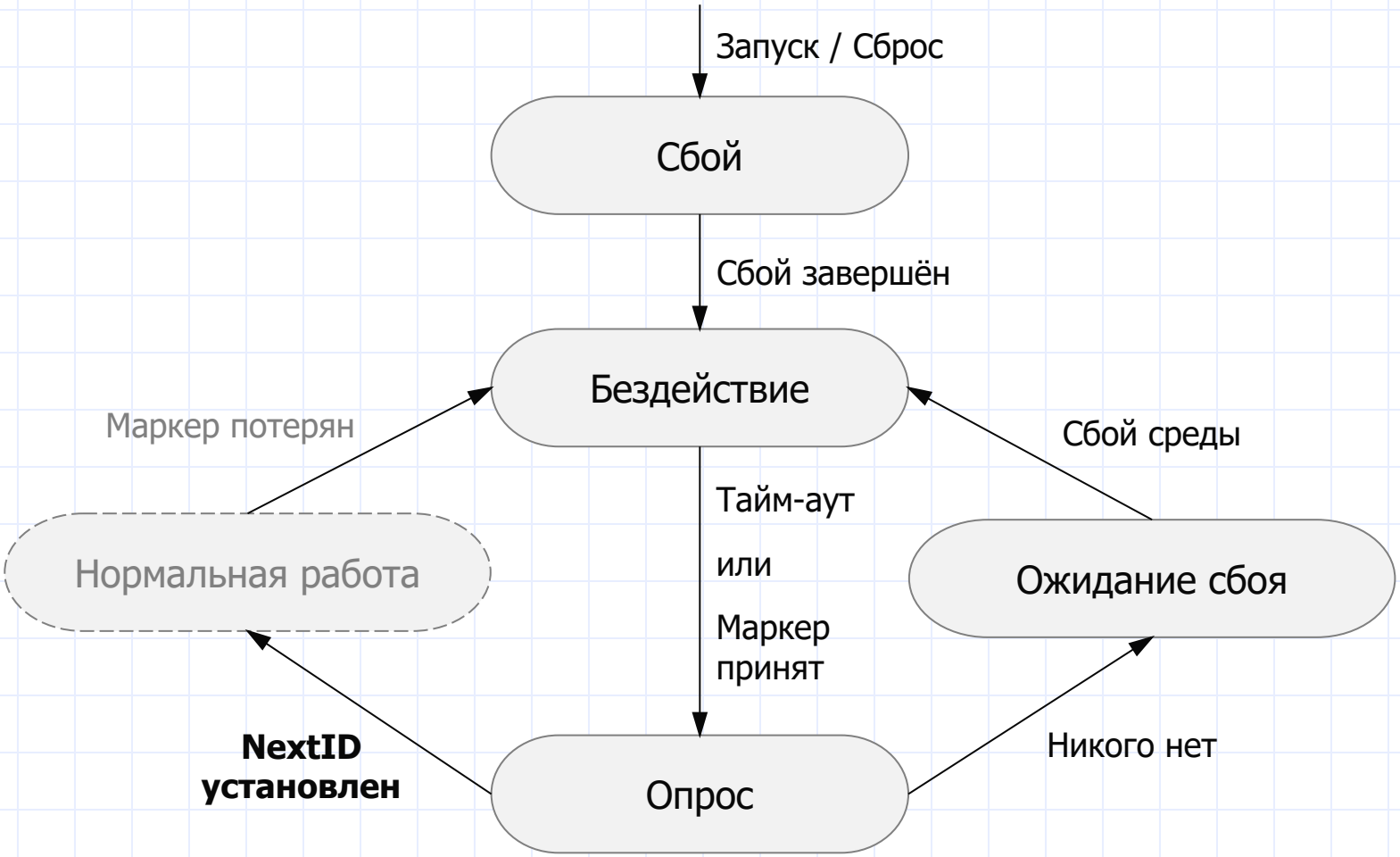


- Каждый узел кроме собственного адреса (MyID) хранит адрес следующего узла (NextID), которому передаётся маркер.
- Если у узла с маркером имеется пакет для передачи, то он передаёт пакет, а затем и маркер, целевому узлу. Если пакета для передачи нет, то маркер передаётся узлу с адресом NextID.

Метод с передачей маркера в топологии «шина»: Диаграмма перехода между состояниями



Метод с передачей маркера в топологии «шина»: Установка NextID при подключении узлов в сети

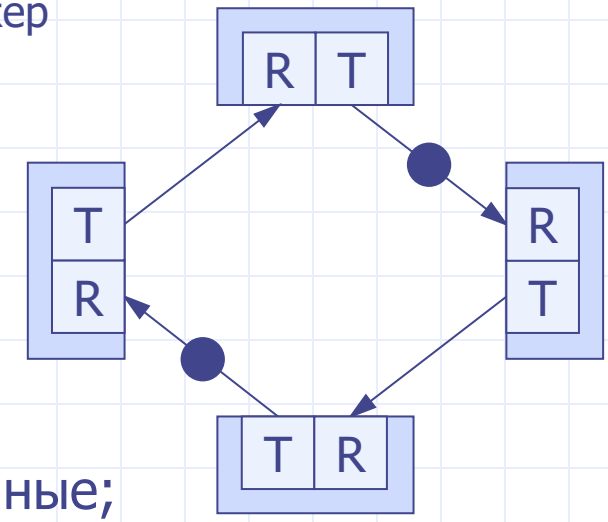


перебор адресов (с переполнением счётчика)

Метод с передачей маркера в топологии «кольцо»

- Логическое «кольцо».
- Физическое «кольцо».
- В «кольце» циркулирует один или несколько маркеров.
- Если необходимо передать данные, то узел дожидается прихода маркера, передаёт маркер (с ID получателя в заголовке) и данные;

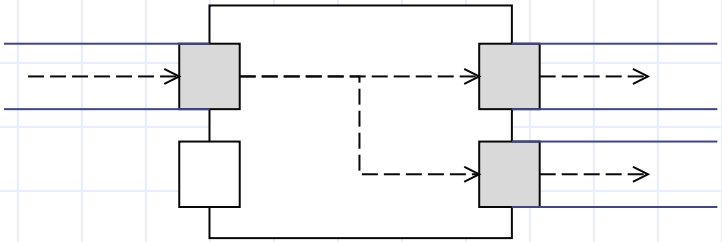
T – Transmitter (передатчик)
R – Receiver (приёмник)
• – маркер



- Когда маркер и данные доходят до узла-получателя, тот их принимает, а маркер и данные передаёт дальше по кольцу.
- Когда маркер с данными возвращается к узлу-отправителю, проверяется, совпадает ли пришедший пакет с переданным (корректно ли переданы данные).

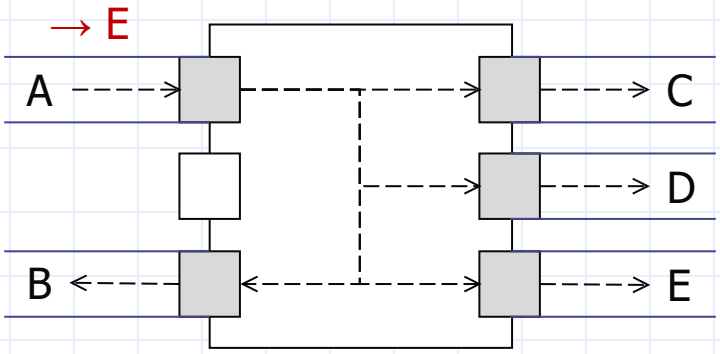
Устройства физического уровня

- Усилитель –
  - используется для удлинения аналоговых линий связи;
    - увеличивает амплитуду сигнала.
- Повторитель –
  - используется для удлинения цифровых линий связи;
    - повторяет сигнал, восстанавливает форму.
- Концентратор (англ. hub) –
  - ретранслирует входящий сигнал с одного порта на все остальные подключенные порты;
    - ничего не знает о MAC-адресах;
    - применяется в топологии «звезда».

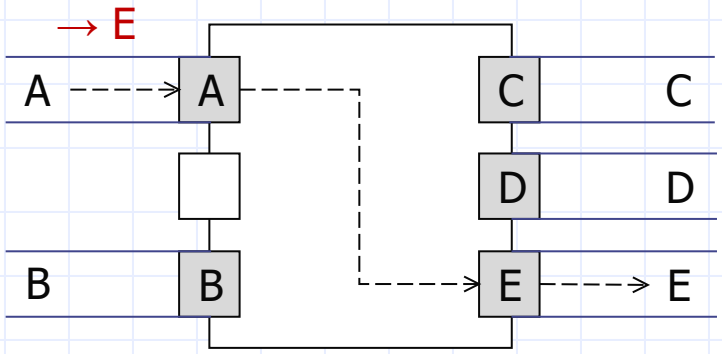


Устройства канального уровня

- Коммутатор (англ. switch) –
  - ретранслирует сигнал с одного входа на определённый выход;
    - распознаёт MAC-адреса и интерпретирует пакеты канального уровня;
    - применяется в топологии «звезда».



Концентратор
(hub)



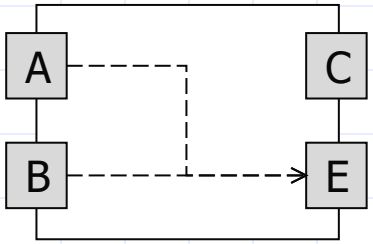
Коммутатор
(switch)

Устройства канального уровня

Коммутатор

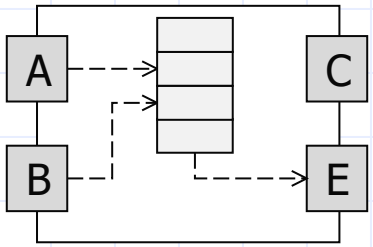
Без буферизации

Перенаправляет пакет в нужный порт по MAC-адресу в заголовке.



Буферизирующий

Принимает пакет целиком в оперативную память (буфер), анализирует его заголовок и передаёт в нужный порт по MAC-адресу в заголовке.



Маршрутизирующий

Буферизирующий с дополнительной логикой маршрутизации пакетов канального уровня. Сокращает конфликты за счёт внутренней изоляции маршрутов.

Сетевые технологии Ethernet и Wi-Fi

- Сетевая технология сочетает протоколы и стандарты нескольких уровней.
- Ethernet (100 Мбит / 1 Гбит / 10 Гбит):
  - проводная среда,
  - физическая «звезда»,
  - логическая «шина»,
  - метод доступа CSMA/CD.
- Wi-Fi (2.4 ГГц / 5 ГГц):
  - беспроводная среда,
  - физическая «шина»,
  - логическая «шина»,
  - метод доступа CSMA/CA.

Сетевые технологии Ethernet и Wi-Fi

- Метод CSMA/CA лучше работает при большом трафике, а метод CSMA/CD лучше работает при маленьком трафике.

Вопрос:

Почему в технологии Ethernet применяется метод доступа CSMA/CD, а в технологии Wi-Fi – метод доступа CSMA/CA?

Ответ:

Технологии Wi-Fi использует общую для всех узлов физическую среду, в которой необходимо бороться с коллизиями, поэтому в технологии Wi-Fi используется метод CSMA/CA.

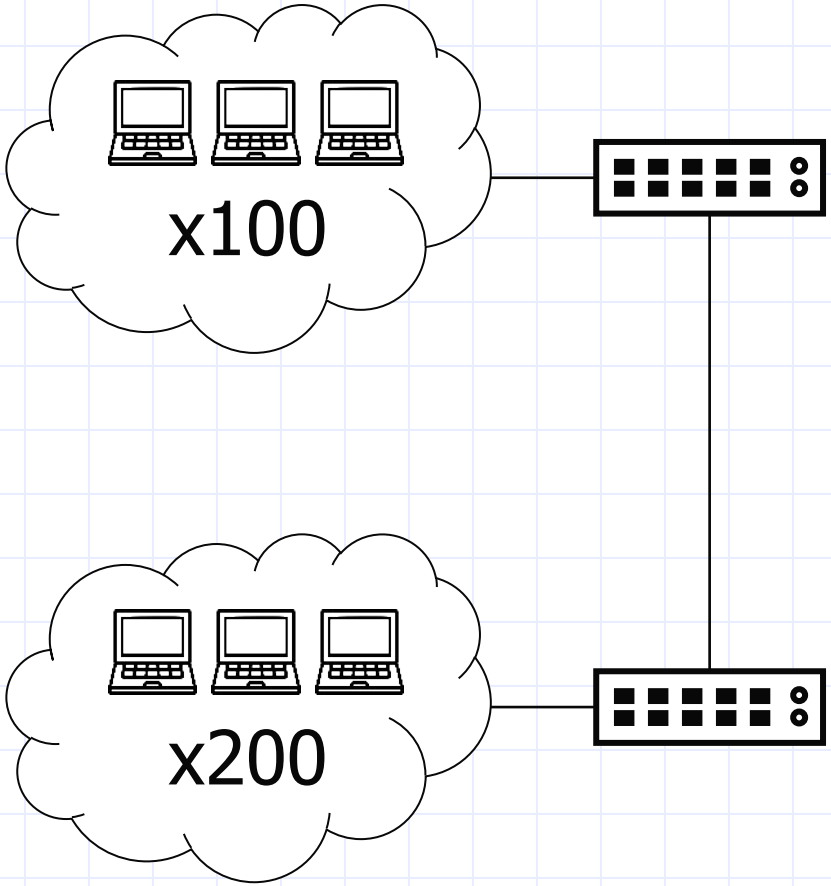
Современная технология Ethernet использует проводную среду и маршрутизирующий коммутатор, который обеспечивает избегание коллизий за счёт приёма пакетов в оперативную память и маршрутизации без коллизий, поэтому в технологии Ethernet используется более скоростной метод CSMA/CD.

Сетевой уровень OSI – Интернет-протокол IP

- Доставка сообщений в глобальной сети (между узлами разных локальных сетей)
- Логическая адресация узлов сети
- Маршрутизация сообщений
- Передача пакетов данных (дейтаграмм) с заданными параметрами качества обслуживания
- Обеспечивается протоколом интернета IP

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Глобальная сеть на коммутаторах?



- Коммутатор хранит таблицу соответствий:
  - MAC-адрес → порт.
- По 300 записей в таблице на каждом коммутаторе.
- В глобальной сети сотни миллионов узлов…
- MAC-адреса не являются глобально уникальными.

Интернет-протокол IP

- Проект DARPA Defense Advanced Research Projects Agency
- IPv4 – современная версия:
  - 32-разрядный адрес (4 байта);
  - В адресе кодируется номер узла и номер подсети;
  - Не обеспечивает безопасность;
  - Не расширяемый.
- IPv6 – перспективная версия:
  - 128-разрядный адрес (16 байтов);
  - Адрес многоуровневый, содержит 3 уровня агрегации для маршрутизации;
  - Обеспечивает безопасность;
  - Расширяемый.

Интернет-протокол IPv4. Логический адрес

- Формат адреса – 4 десятичных числа (0 – 255), разделённых точкой.
- Глобально уникальные адреса
  - **46.216.181.43** – сервер сайта БГУИР
  - **77.88.55.66** – сервер поисковика Яндекс
  - **8.8.8.8** – сервер DNS компании Google
- Адреса для локальных сетей
  - **10.0.0.1** – адреса, начинающиеся с 10.0.0.
  - **172.16.1.21** – адреса, начинающиеся с 172.16.1-32.
  - **192.168.100.1** – адреса, начинающиеся с 192.168.
- В адресе кодируется номер подсети и номер узла

Кодирование номера подсети и номера узла. Классовая модель IP-адресов (устарела)

- Номер подсети и номер узла выровнены на границе байта.

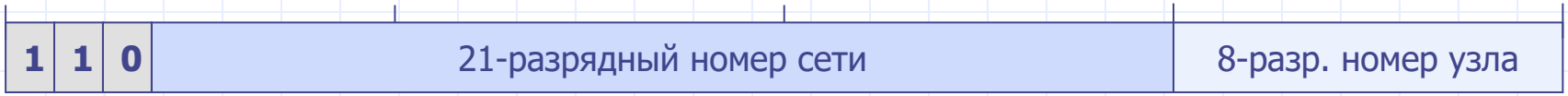
Класс A

> 16 000 000 адресов

Класс B

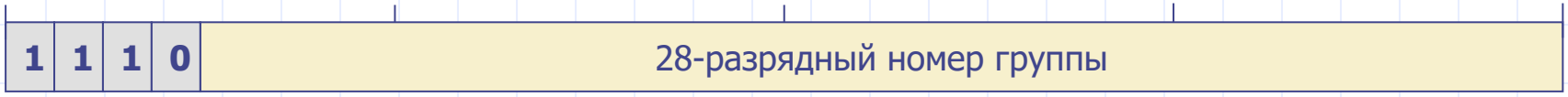
65 536 адресов

Класс C



256 адресов

Класс D – для маршрутизаторов



Кодирование номера подсети и номера узла. Бесклассовая модель IP-адресов. Маска подсети

- Маска подсети –
  - битовая маска, старшие разряды которой установлены в единицу, а младшие – в ноль; определяет, какая часть IP-адреса относится к номеру подсети, а какая – к номеру узла.
- Вычисление номера подсети
  - IP **&** Маска (побитовая операция «И»)
- Пример
  - IP: **192.168.100.1** `11000000.10101000.01100100.00000001`
  - Маска: **255.255.255.0** `11111111.11111111.11111111.00000000`
  - Номер подсети IP & Маска: **192.168.100.0** `11000000.10101000.01100100.00000000`
- Задание IP-адреса с маской подсети
  - **192.168.100.1/24** – указано количество единиц в маске

Количество узлов в локальной сети в зависимости от маски подсети

- **192.168.100.1/24 (маска 255.255.255.0)**
  - номер подсети: 192.168.100.0
  - широковещательный адрес: 192.168.100.255
  - диапазон адресов узлов: **192.168.100.1 – 192.168.100.254**
  - максимум **254** узла

- **192.168.100.1/23 (маска 255.255.254.0)**
  - номер подсети: 192.168.100.0
  - широковещательный адрес: 192.168.101.255
  - диапазон адресов узлов: **192.168.100.1 – 192.168.101.254**
  - максимум **510** узлов

Зарезервированные IP-адреса

- **0.0.0.0**
  - адрес несуществующего узла, имеет особое значение.
- **127.0.0.1**
  - адрес для обращения узла к самому себе (англ. loopback);
  - передача сообщений по такому адресу не приводит к задействованию канального уровня.
- **255.255.255.255**
  - широковещательный адрес в любой сети;
  - используется для передачи сообщений одновременно нескольким узлам в локальной сети;
  - широковещательная передача обеспечивается канальным уровнем.

Протоколы ARP и RARP

- Для передачи IP-дейтаграммы по локальной сети необходимо знать MAC-адрес.
- ARP – англ. Address Resolution Protocol
  - Позволяет по IP-адресу узнать MAC-адрес.
- RARP – англ. Reverse Address Resolution Protocol
  - Позволяет по MAC-адресу узнать IP-адрес.
- Формат пакета ARP

<table>
<tr><td colspan="2">Тип канального протокола (Ethernet)</td><td>Тип сетевого протокола (IP)</td></tr>
<tr><td>Длина физич. адреса</td><td>Длина логич. адреса</td><td>Операция</td></tr>
<tr><td colspan="3">Физический адрес отправителя (MAC)</td></tr>
<tr><td colspan="3">Логический адрес отправителя (IP)</td></tr>
<tr><td colspan="3">Физический адрес получателя (MAC)</td></tr>
<tr><td colspan="3">Логический адрес получателя (IP)</td></tr>
</table>

Алгоритм работы протокола ARP

- Посылается широковещательный кадр в сеть с указанием искомого IP-адреса.
- Узел с искомым IP-адресом посылает ответ, в который вкладывает свой MAC-адрес.

ARP-запрос

MAC-получателя	**FF:FF:FF:FF:FF:FF**
MAC-отправителя	AA:AA:AA:AA:AA:AA
Тип: ARP	
MAC отправителя	AA:AA:AA:AA:AA:AA
IP отправителя	192.168.100.1
MAC получателя	**00:00:00:00:00:00**
IP получателя	**192.168.100.2**

ARP-ответ

MAC-получателя	AA:AA:AA:AA:AA:AA
MAC-отправителя	BB:BB:BB:BB:BB:BB
Тип: ARP	
MAC отправителя	**BB:BB:BB:BB:BB:BB**
IP отправителя	**192.168.100.2**
MAC получателя	AA:AA:AA:AA:AA:AA
IP получателя	192.168.100.1

Кэширование ARP-ответов

- Если перед посылкой каждого сообщения делать ARP-запрос, то в сети увеличится трафик и снизится скорость.
- Чтобы этого избежать, результаты ARP-запросов кэшируются узлом-отправителем.
- ARP-кэш – это таблица, которая содержит записи соответствий: IP-адрес ↔ MAC-адрес.
  - Заданы автоматически (dynamic) или вручную (static).
  - Со временем dynamic записи устаревают.

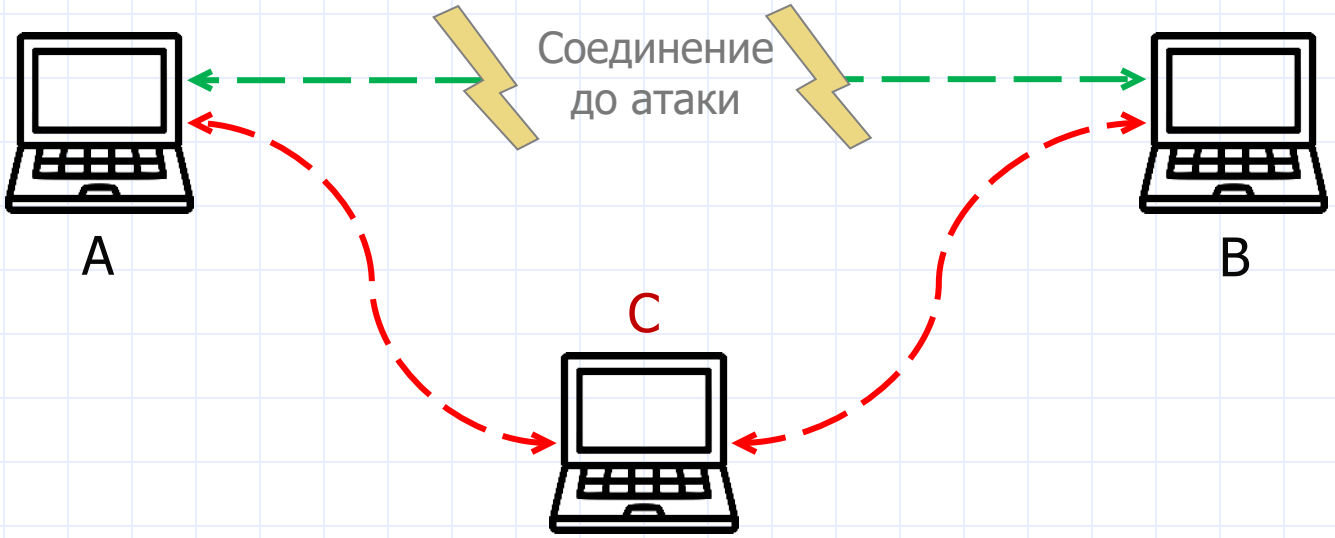
Internet Address	Physical Address	Type
192.168.100.2	**bb-bb-bb-bb-bb-bb**	**dynamic**
192.168.100.255	ff-ff-ff-ff-ff-ff	static

- Утилита **arp**

Уязвимость ARP к атаке посредника

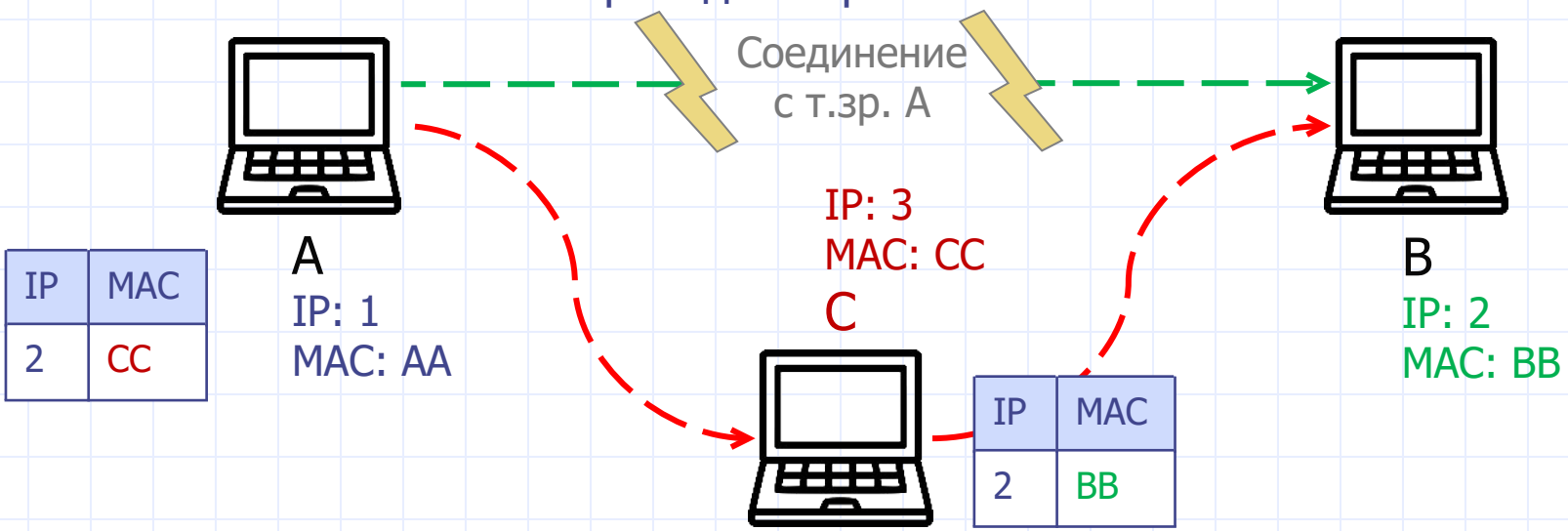
- Отсутствует проверка подлинности ARP-ответа.

- Атака посредника (англ. Men In The Middle, MITM) –
  - злоумышленник тайно ретранслирует (и может изменять) сообщения между двумя сторонами, которые считают, что они непосредственно общаются друг с другом.



Уязвимость ARP к атаке посредника

- Злоумышленник выдаёт себя за другой узел, подменив запись ARP у жертвы (англ. ARP-spoofing):
  - > A: У кого IP 2?
  - > Ответ от C: IP 2 у CC.
  - > Ответ от B: IP 2 у BB.
  - > **A решает, что IP 2 у CC, т.к. этот ответ пришёл раньше.**
  - > C: У кого IP 2?
  - > Ответ от B: IP 2 у BB.
- IP-пакеты от A к B проходят через C.



Структура IPv4-пакета

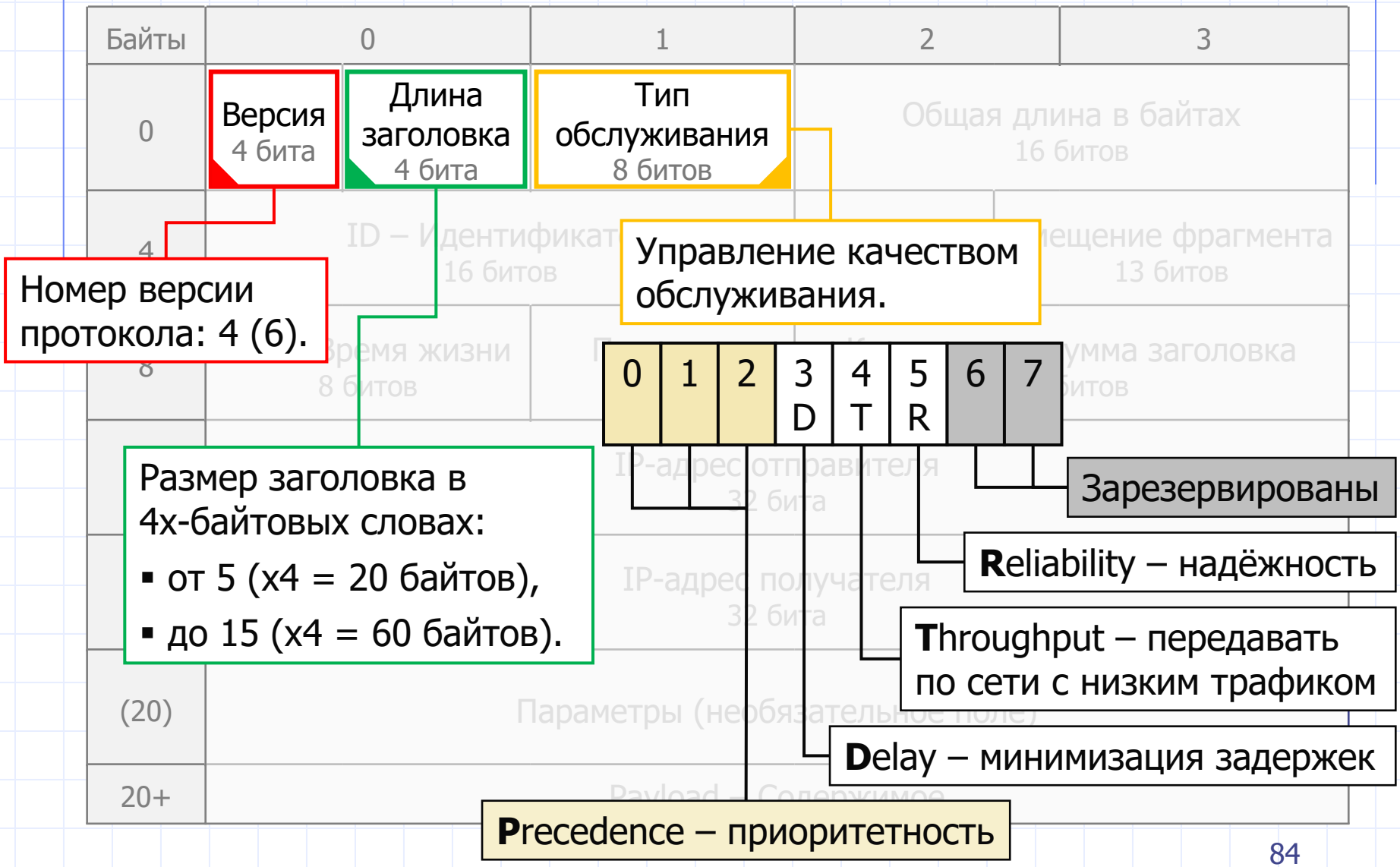
Байты	0		1	2	3
0	Версия 4 бита	Длина заголовка 4 бита	Тип обслуживания 8 битов	Общая длина в байтах 16 битов	
4	ID – Идентификатор 16 битов			Флаги 3 бита	Смещение фрагмента 13 битов
8	TTL – Время жизни 8 битов		Протокол 8 битов	Контрольная сумма заголовка 16 битов	
12	IP-адрес отправителя 32 бита				
16	IP-адрес получателя 32 бита				
(20)	Параметры (необязательное поле)				
20+	Payload – Содержимое				

Заголовок

Структура IPv4-пакета (продолжение)

- Версия $^{4\ бита}$ – номер версии протокола – 4.
- Длина заголовка $^{4\ бита}$ – размер заголовка в 4х-байтовых словах; зависит от размера Параметров;
  - минимальное корректное значение – 5 (х4 = 20 байтов);
  - максимальное значение – 15 (х4 = 60 байтов).
- Тип обслуживания $^{8\ битов}$ – параметры, управляющие качеством обслуживания; по битам:
  - [0-2] – приоритетность,
  - [3] – важность минимизации задержек при передаче,
  - [4] – важность скорости передачи (передавать по сети с низким трафиком),
  - [5] – важность надёжности передачи,
  - [6-7] – зарезервированы.

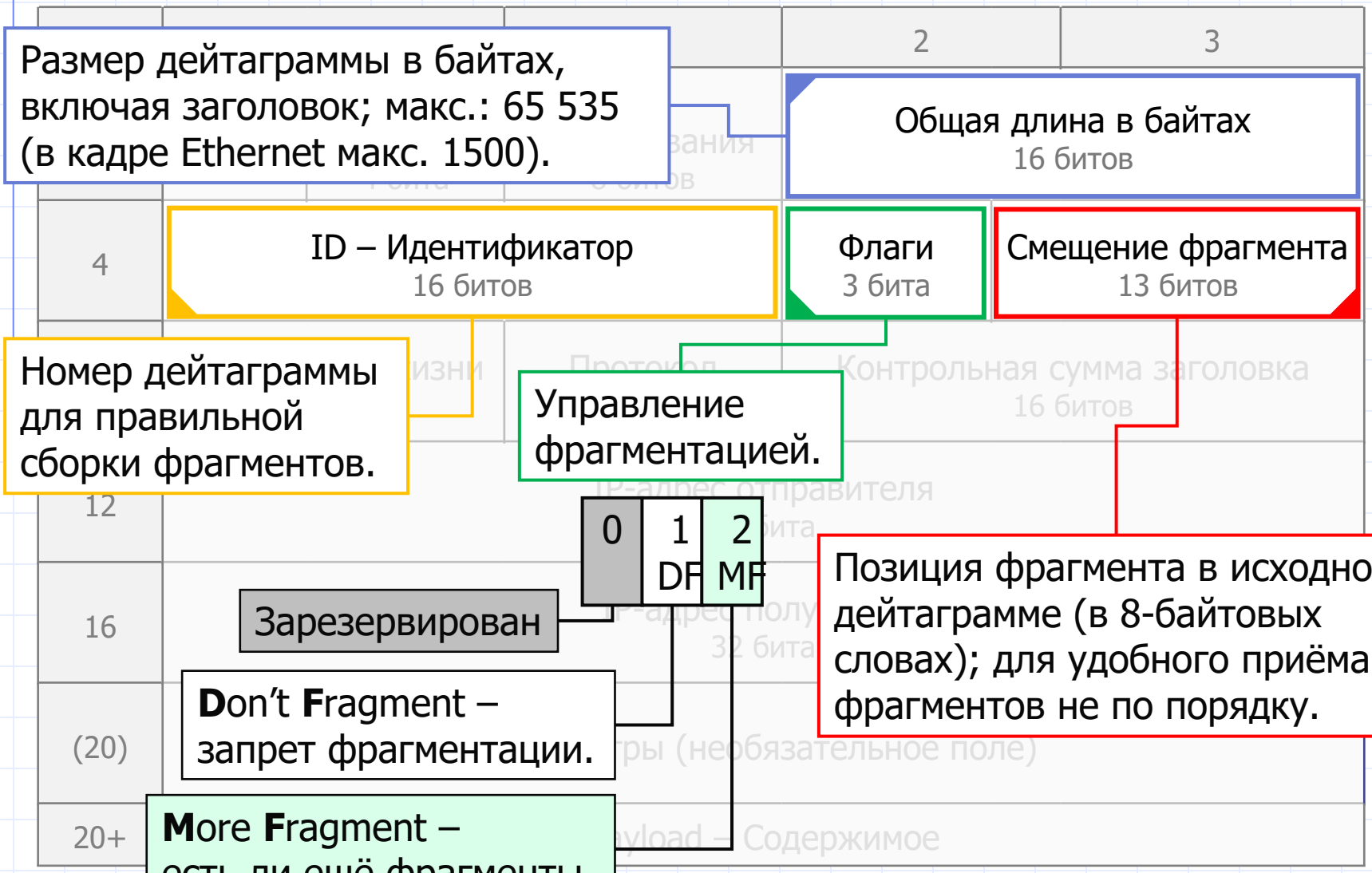
Структура IPv4-пакета (продолжение)



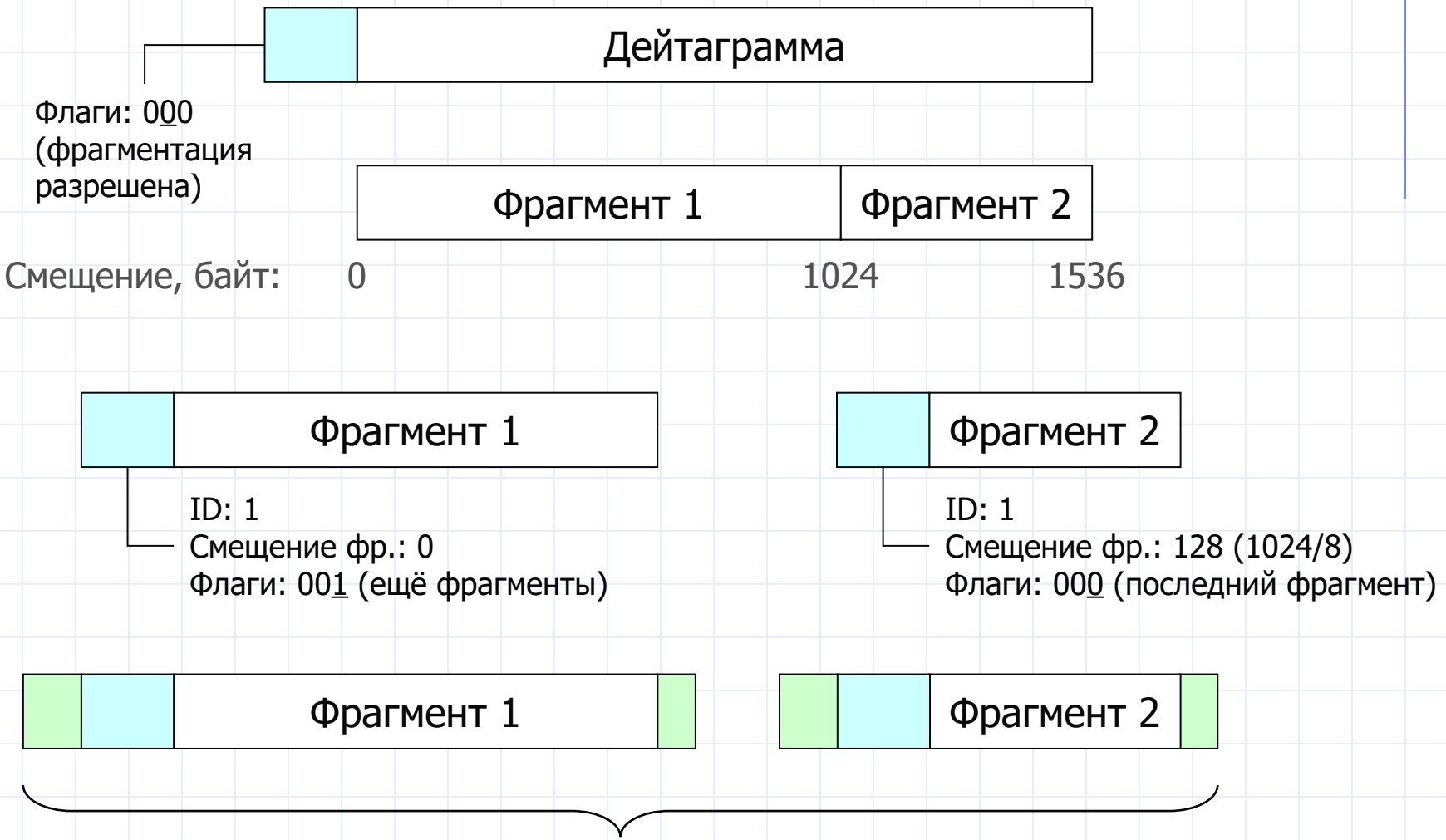
Структура IPv4-пакета (продолжение)

- Общая длина в байтах [16 битов] – размер дейтаграммы, включая заголовок; максимум 65 535 байт (содержимое кадра Ethernet – макс. 1500 байт).
- ID – Идентификатор [16 битов] – номер дейтаграммы для правильной сборки фрагментов.
- Флаги [3 бита] – управление фрагментацией:
  - [0] – зарезервирован,
  - [1] – запрет фрагментации (Don't Fragment),
  - [2] – есть ли ещё фрагменты (More Fragments).
- Смещение фрагмента [13 битов] – позиция фрагмента в исходной дейтаграмме (в 8-байтовых словах); для удобного приёма фрагментов не по порядку.

Структура IPv4-пакета (продолжение)



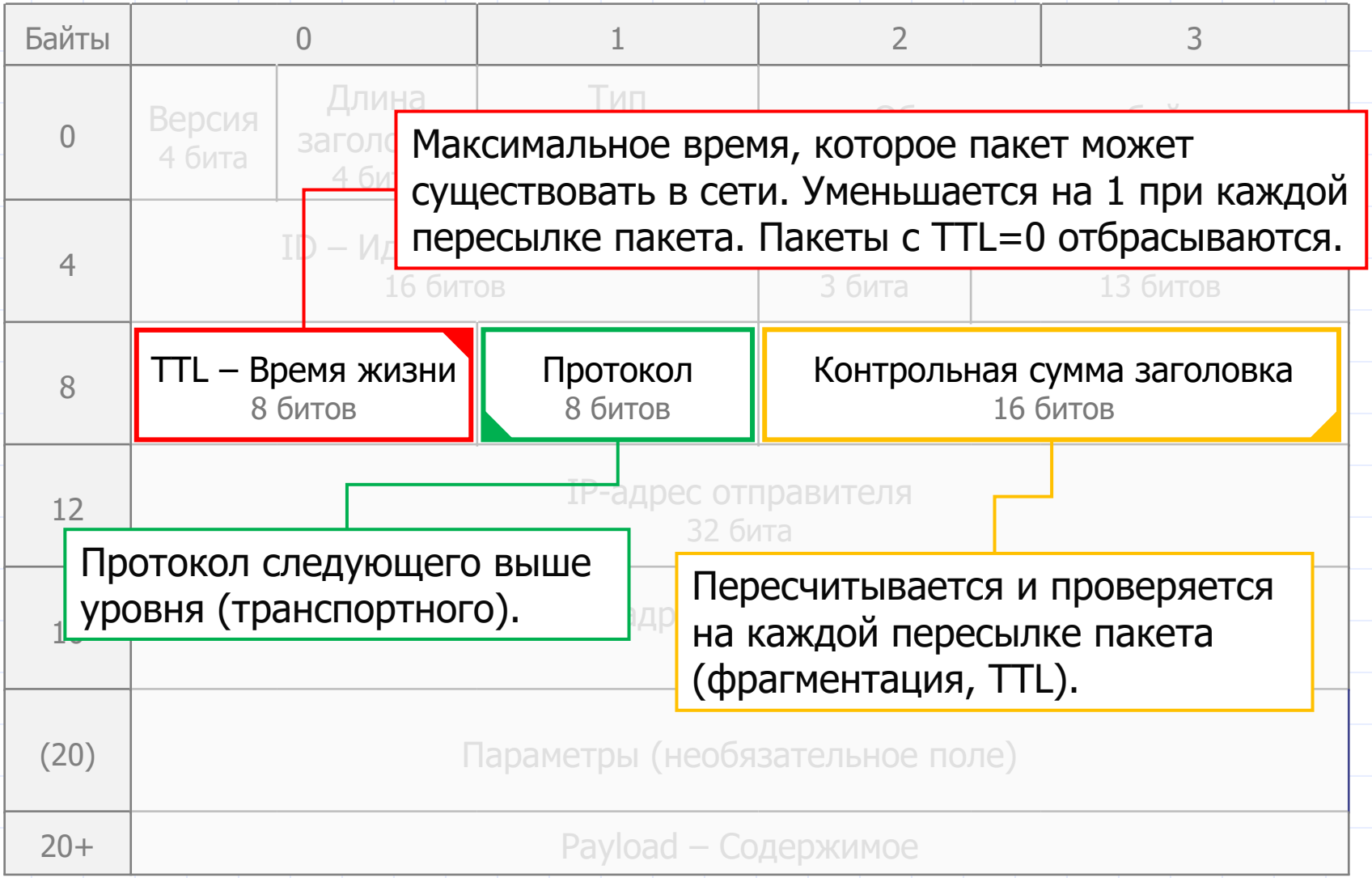
Фрагментация IP-дейтаграммы



Структура IPv4-пакета (продолжение)

- TTL – Время жизни 8 битов – максимальное время, которое пакет может существовать в сети. Уменьшается на 1 при каждой пересылке пакета. Пакеты с TTL=0 отбрасываются.
- Протокол 8 битов – указывает, какой протокол следующего выше уровня (транспортного) инкапсулируется в дейтаграмме.
- Контрольная сумма заголовка 16 битов – пересчитывается и проверяется на каждой пересылке пакета, т.к. фрагментация и уменьшение TTL приводят к изменению заголовка.

Структура IPv4-пакета (продолжение)



Структура IPv4-пакета (продолжение)

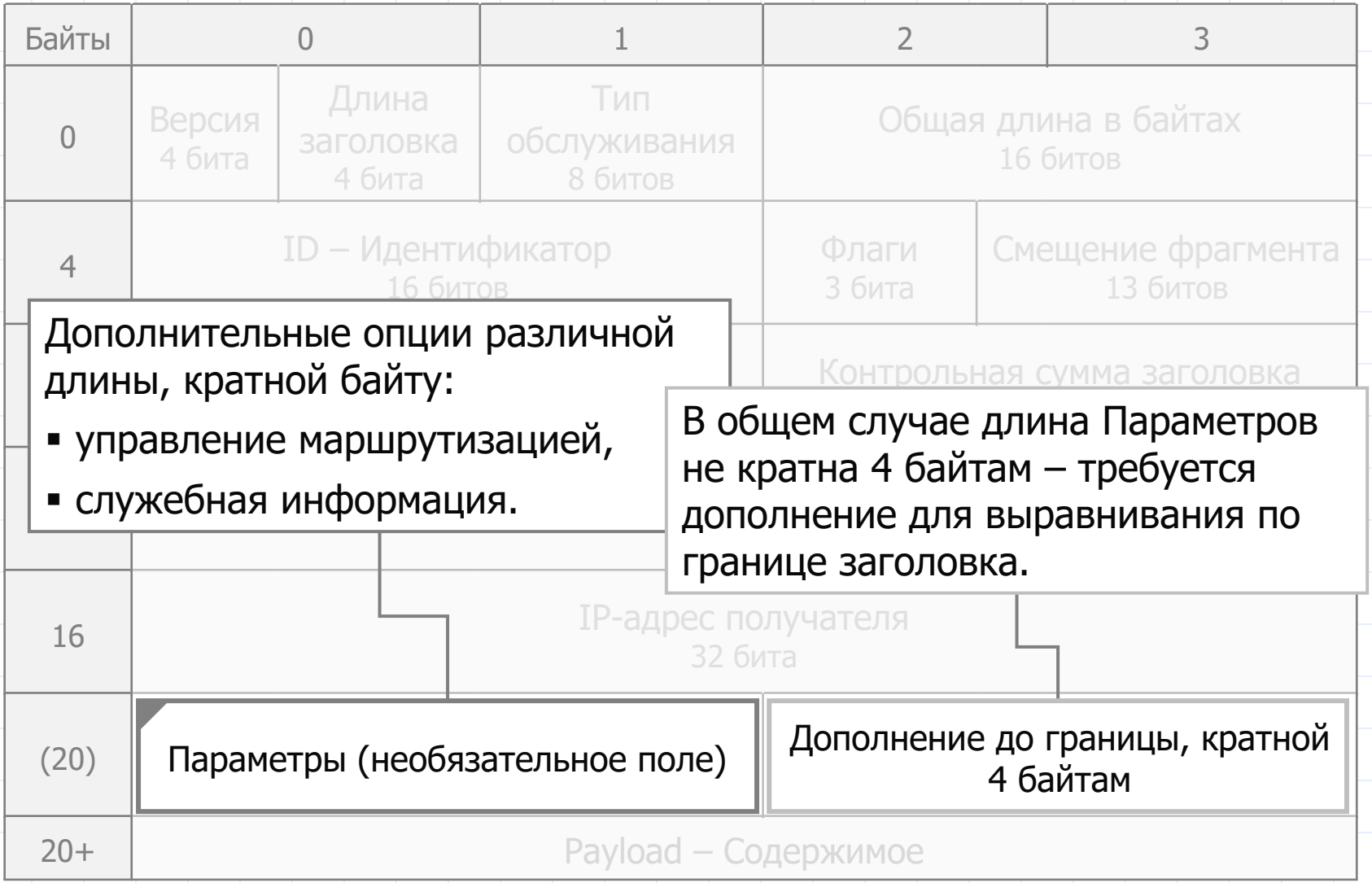
- IP-адрес отправителя 32 бита
- IP-адрес получателя 32 бита

- Параметры ? байтов – необязательное поле; содержит дополнительные опции различной длины, кратной байту. В общем случае длина не кратна 4 байтам – требуется дополнение для выравнивания по границе заголовка.
  - управление маршрутизацией,
  - служебная информация.

Структура IPv4-пакета (продолжение)

Байты	0		1	2	3
0	Версия 4 бита	Длина заголовка 4 бита	Тип обслуживания 8 битов	Общая длина в байтах 16 битов	
4	ID – Идентификатор 16 битов			Флаги 3 бита	Смещение фрагмента 13 битов
8	TTL – Время жизни 8 битов		Протокол 8 битов	Контрольная сумма заголовка 16 битов	
12	IP-адрес отправителя 32 бита				
16	IP-адрес получателя 32 бита				
(20)	Параметры (необязательное поле)				
20+	Payload – Содержимое				

Структура IPv4-пакета (продолжение)



Конфигурирование узлов сети

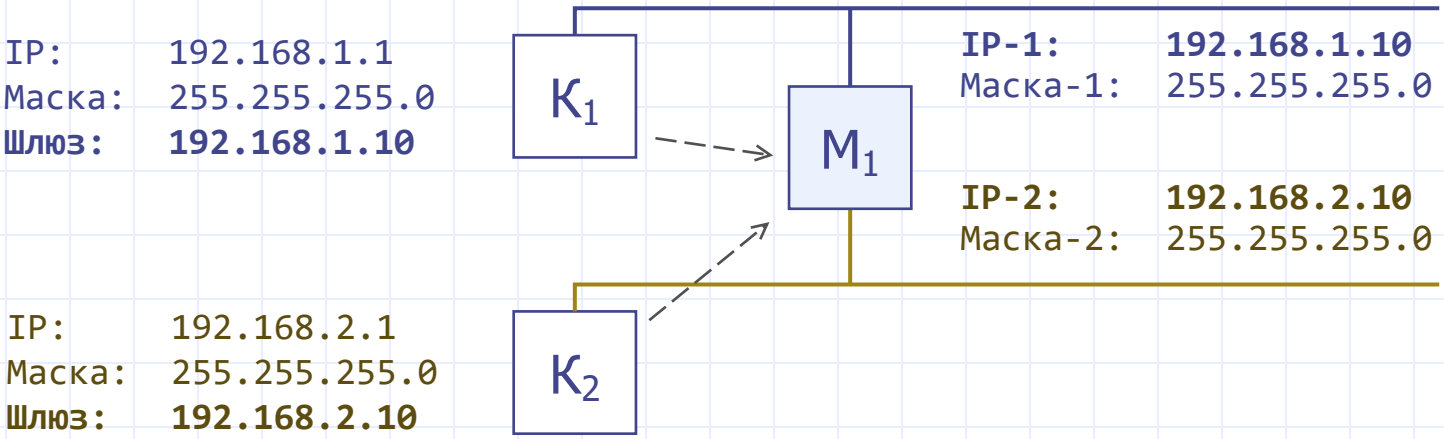
- Уникальный IP-адрес – достаточно для работы в локальной сети.
- Маска подсети – позволяет определить принадлежность адреса к данной локальной сети.
- Шлюз (англ. Default Gateway) – на этот адрес передаются все пакеты, адресованные не в данную сеть.

```
IP:   192.168.101.4
Mask: 255.255.255.0
Default Gateway: 192.168.101.1
```

DHCP (Dynamic Host Control Protocol)

- DHCP-сервер содержит у себя таблицу MAC-адресов и соответствующих им IP-адресов. Клиенты, которые вошли в сеть и не имеют параметров протокола IP для работы, обращаются с широковещательным запросом и обнаруживают DHCP-сервер, который им отвечает, указывает свой IP адрес и MAC-адрес, на который узел посылает запрос с просьбой выдать ему параметры конфигурации для работы в локальной сети.
- DHCP-сервер ведет у себя таблицу IP-адресов и соответствующих им MAC-адресов. Когда к серверу приходит запрос, то в таблице появляется новая запись. Из пула свободных адресов выдается IP, в таблицу вписывается MAC адрес того узла, от которого пришел запрос, а также время, до которого это соответствие является действительным («время аренды»).
- В DHCP можно указать, что для определенных MAC адресов необходимо выдавать определенные IP-адреса.

Передача IP-дейтаграммы узлу в другой сети



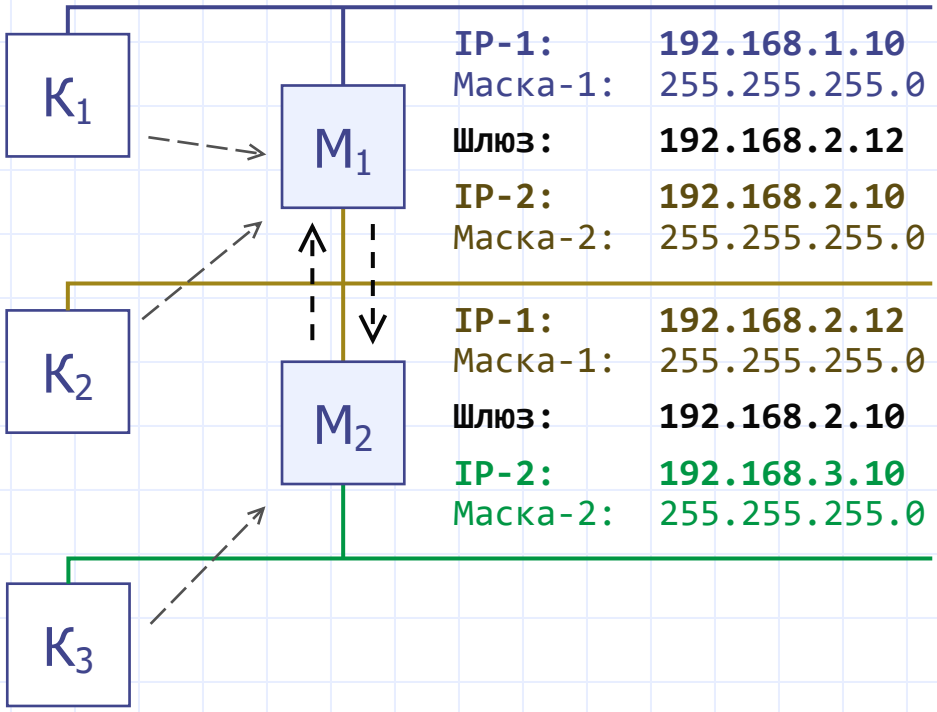
- Мост (англ. bridge) –
  - устройство, объединяющие две подсети в единую сеть;
    - мост сетевого уровня пересылает IP-пакеты.

Связывание N сетей N-1 мостом

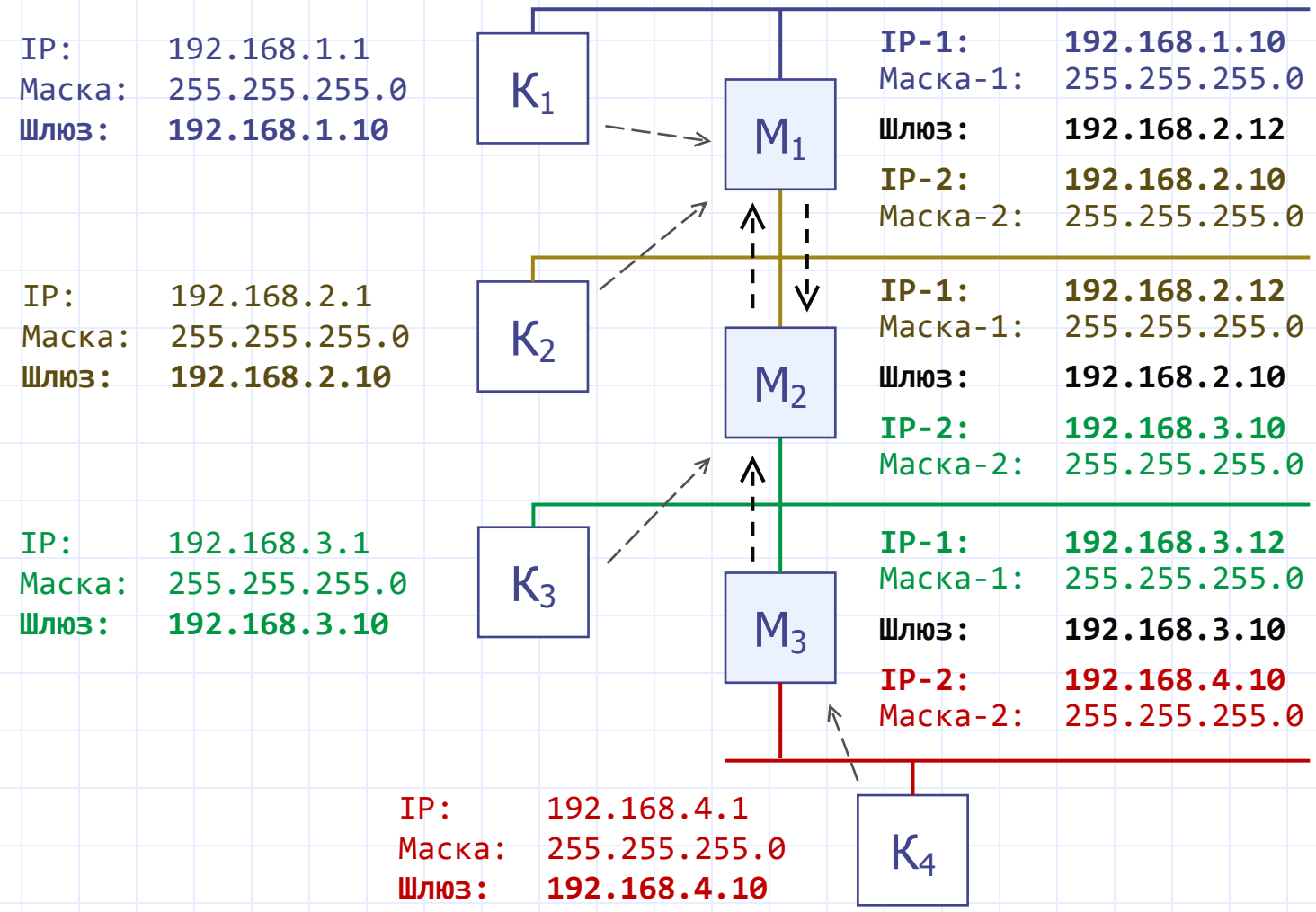
IP: 192.168.1.1
Маска: 255.255.255.0
Шлюз: 192.168.1.10

IP: 192.168.2.1
Маска: 255.255.255.0
Шлюз: 192.168.2.10

IP: 192.168.3.1
Маска: 255.255.255.0
Шлюз: 192.168.3.10



Связывание N сетей N-1 мостом

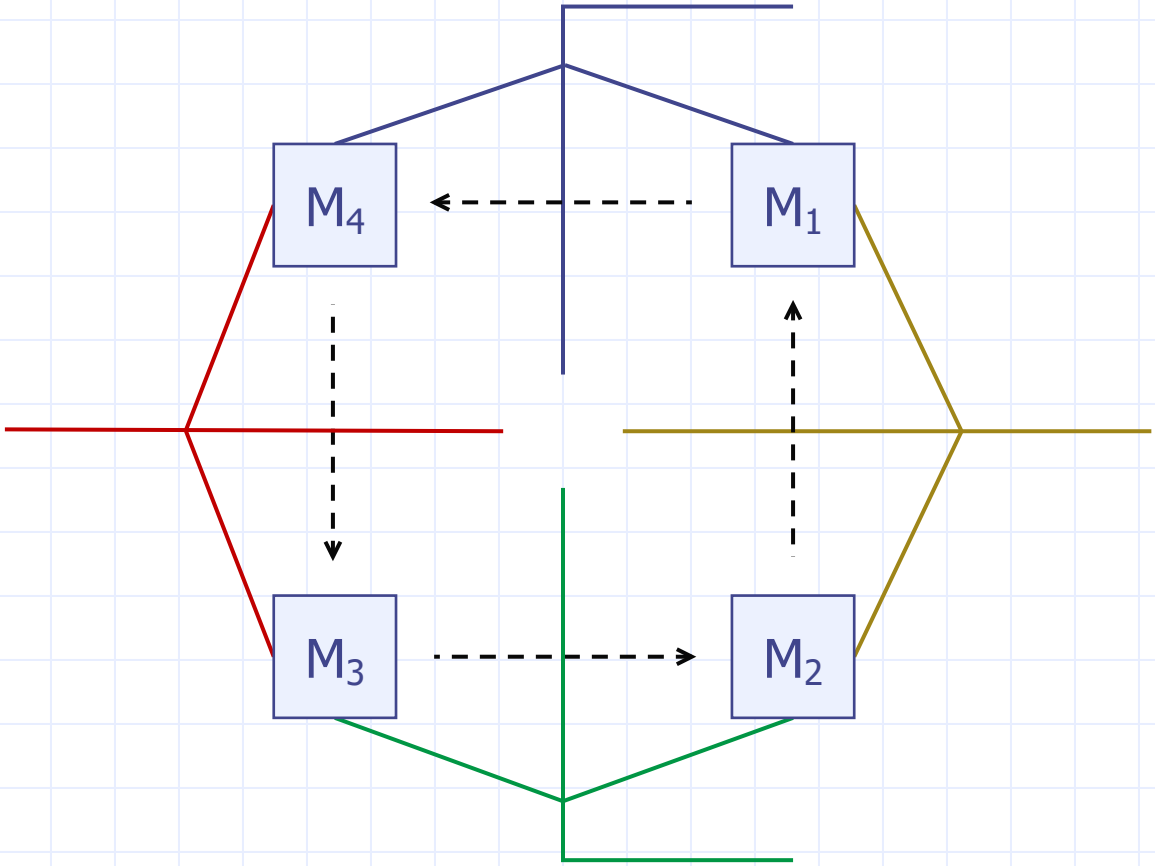


Маршрутизация

- Домашнее задание.

*Связывание N сетей N мостами

- Добавить N-й мост и задать его в качестве шлюза для первого моста – «кольцо» из мостов.



Инструменты инженера-программиста

- Стандартные инструменты ОС
  - arp
  - ipconfig (Windows), ifconfig (Linux)
  - ping
  - tracert (Windows), traceroute (Linux)
- Специальные инструменты анализа трафика
  - Wireshark + Npcap

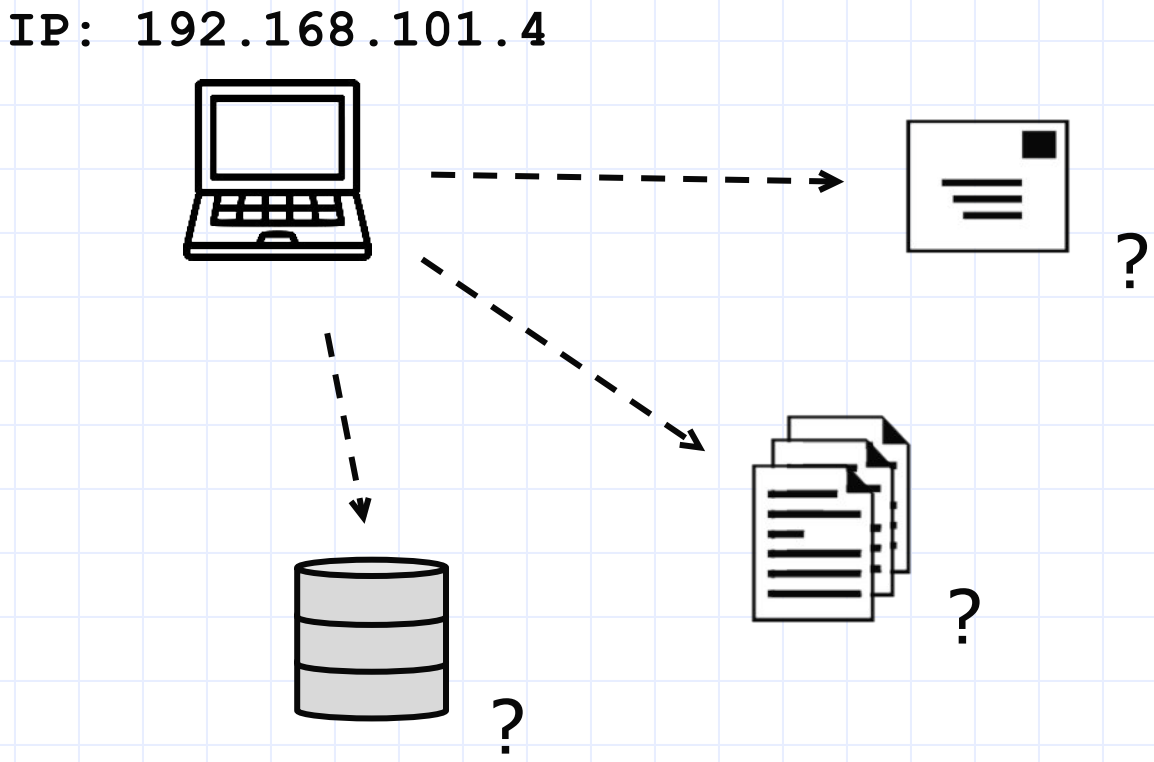
Транспортный уровень OSI

- Идентификация программ-абонентов на узлах сети
- Передача сообщений с установкой и без установки соединения
- Разбиение сообщений на пакеты и их сборка в правильном порядке
- Уведомления (квитанции) о доставке с повторной передачей утерянных пакетов

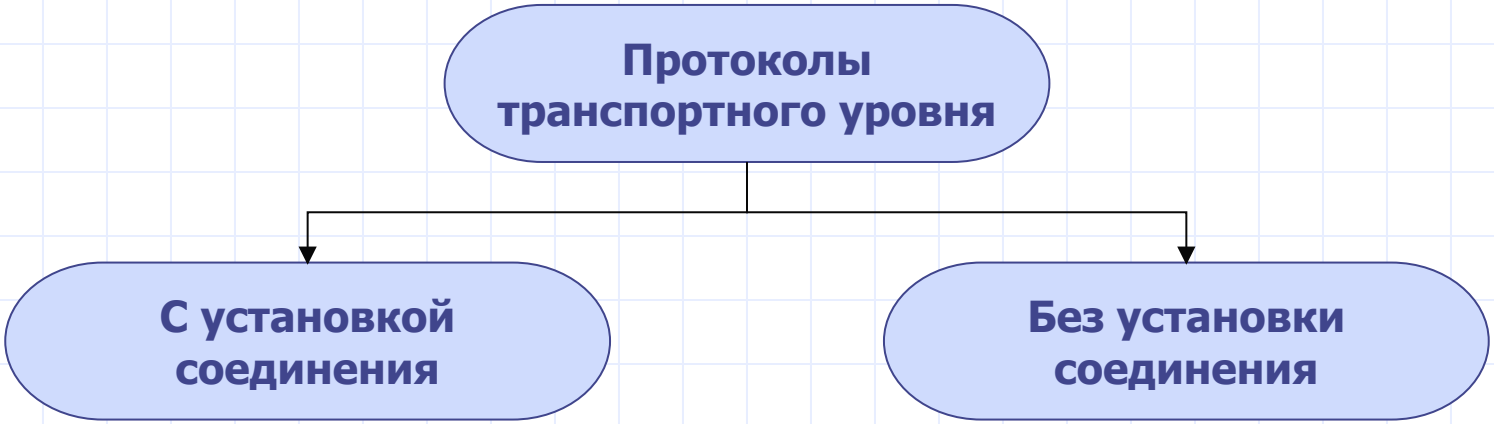
7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Идентификация программ-абонентов

- Порт – двухбайтовый числовой идентификатор, который обозначает одного логического абонента на узле.



Классы протоколов транспортного уровня



- Передача потока данных с разбиением на фрагменты;
- Сборка фрагментов в правильном порядке;
- Гарантированная доставка с подтверждением (квитированием);
- Повторная передача потерянных пакетов.

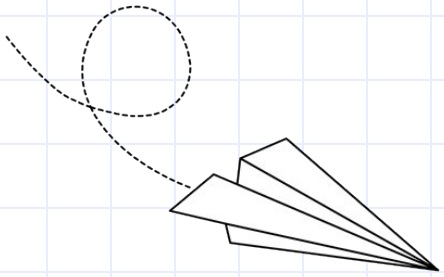
TCP

- Передача одиночных сообщений;
- Нет упорядочивания приходящих сообщений;
- Доставка не гарантируется, подтверждение отсутствует;
- Нет повторной передачи в случае утери сообщения.

UDP

Протокол UDP (англ. User Datagram Protocol)

- UDP – протокол транспортного уровня, обеспечивающий передачу дейтаграмм фиксированного размера:
  - без установки соединения,
  - без гарантий доставки,
  - без подтверждения о доставке,
  - без упорядочивания принимаемых пакетов в порядке отправления.
- Обеспечивает более высокую скорость передачи данных, чем протокол TCP, но не предоставляет гарантий доставки.
- Используется для передачи мультимедиа-данных (аудио и видео), для которых реальный масштаб времени передачи важнее надёжности (допускается потеря пакетов).



Структура UDP-пакета

Байты	0	1	2	3
0	Порт отправителя 16 битов		Порт получателя 16 битов	
4	Длина дейтаграммы 16 битов		Контрольная сумма* 16 битов	
8+	Данные			

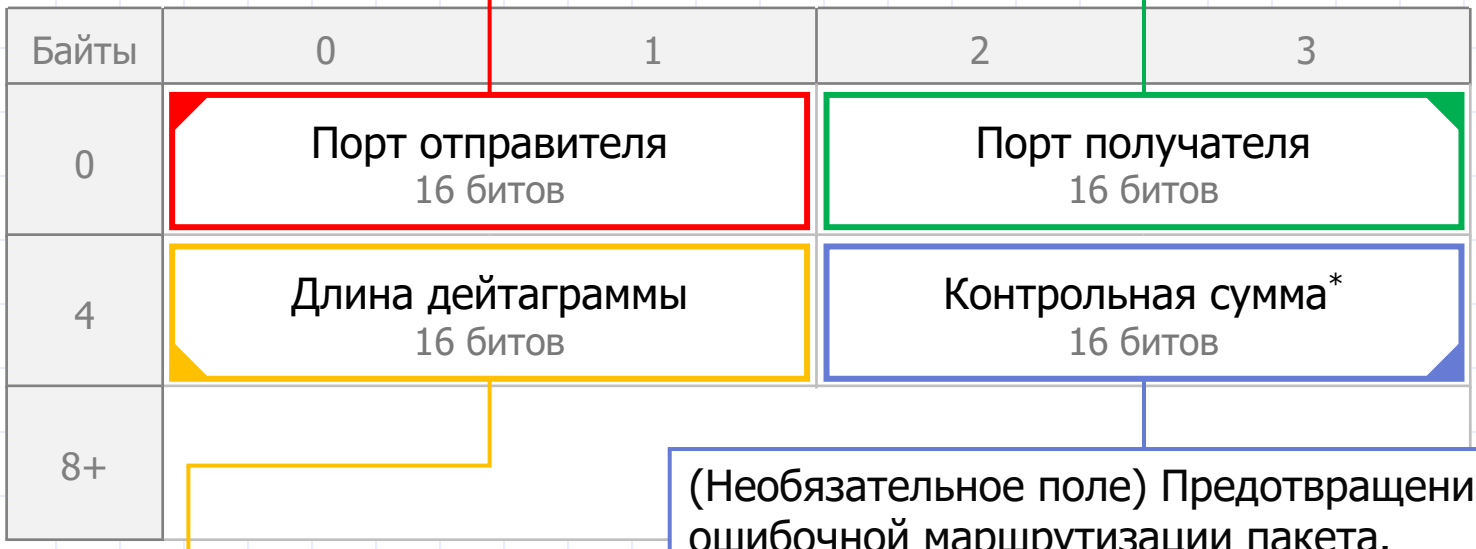
Структура UDP-пакета (продолжение)

- Номер порта отправителя 16 битов – задаёт порт, на который может быть отправлен ответ, если таковой ожидается; иначе – 0.
- Номер порта получателя 16 битов – обязательное поле.
- Длина дейтаграммы 16 битов – размер заголовка и данных в байтах.
  - Минимальная длина пакета равна длине заголовка – 8 байтов.
  - Фактический максимум: 65 515 (-20 байтов на заголовок IPv4).
  - Таким образом, фактический предел размера данных в UDP с IPv4 – 65 507 (ещё -8 байтов заголовок UDP).
- Контрольная сумма* 16 битов – для предотвращения ошибочной маршрутизации дейтаграмм; рассчитывается на основании части заголовка IP (IP-адрес отправителя, IP-адрес получателя, протокол), заголовка UDP и данных. Если контроль обеспечивается протоколом вышестоящего уровня, заполняется нулями. Не является обязательным для IPv4.

Структура UDP-пакета (продолжение)

(Необязательное поле) Указывается, если ожидается получение ответа, иначе – 0.

Обязательное поле.



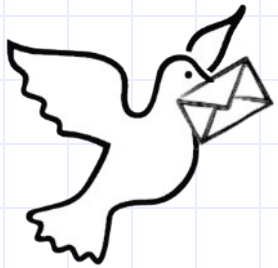
Длина пакета (заголовок + данные) в байтах. Минимум: 8. Фактический максимум: 65 515 (-20 байтов заголовок IPv4).

(Необязательное поле) Предотвращение ошибочной маршрутизации пакета.

- часть заголовка IP (IP-адрес отправителя, IP-адрес получателя, протокол);
- заголовок UDP;
- данные.

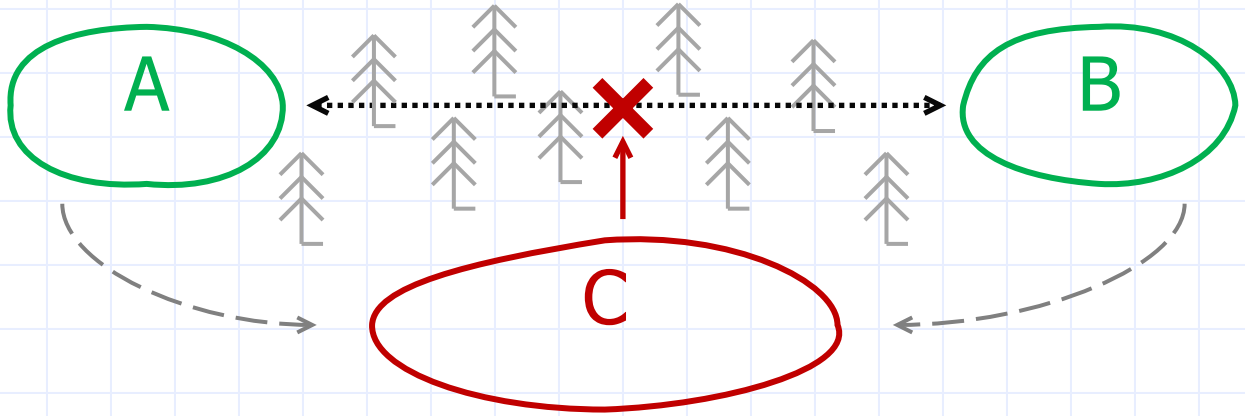
Проблема подтверждения доставки сообщения

- Ромео Монтекки и Джульетта Капулетти сидят взаперти и могут общаться только посредством голубиной почтой.
- Джульетта хочет предупредить Ромео, что о назначенной ими ранее встрече стало известно, и на него готовят засаду.
- Джульетта посылает голубя с зашифрованным письмом и ожидает ответ.
- Голубя могут перехватить на стороне Монтекки и выкинуть письмо. Голубя с ответом также могут перехватить на стороне Капулетти и выкинуть ответ.
- Как Джульетте гарантированно сообщить Ромео о засаде и быть спокойной за своего возлюбленного?



Проблема подтверждения доставки сообщения

- Две армии союзников A и B разделены лесом и могут общаться только отправив письмо гонцом.
- Командир армии A хочет объединиться с армией B, чтобы провести совместный удар по армии противника C.
- Командир армии A посылает гонца с зашифрованным сообщением через лес и ожидает ответ.
- Гонца могут перехватить диверсанты противника, которые прячутся в лесу. Гонца с ответом также могут перехватить.
- Как союзникам гарантированно скооперироваться?



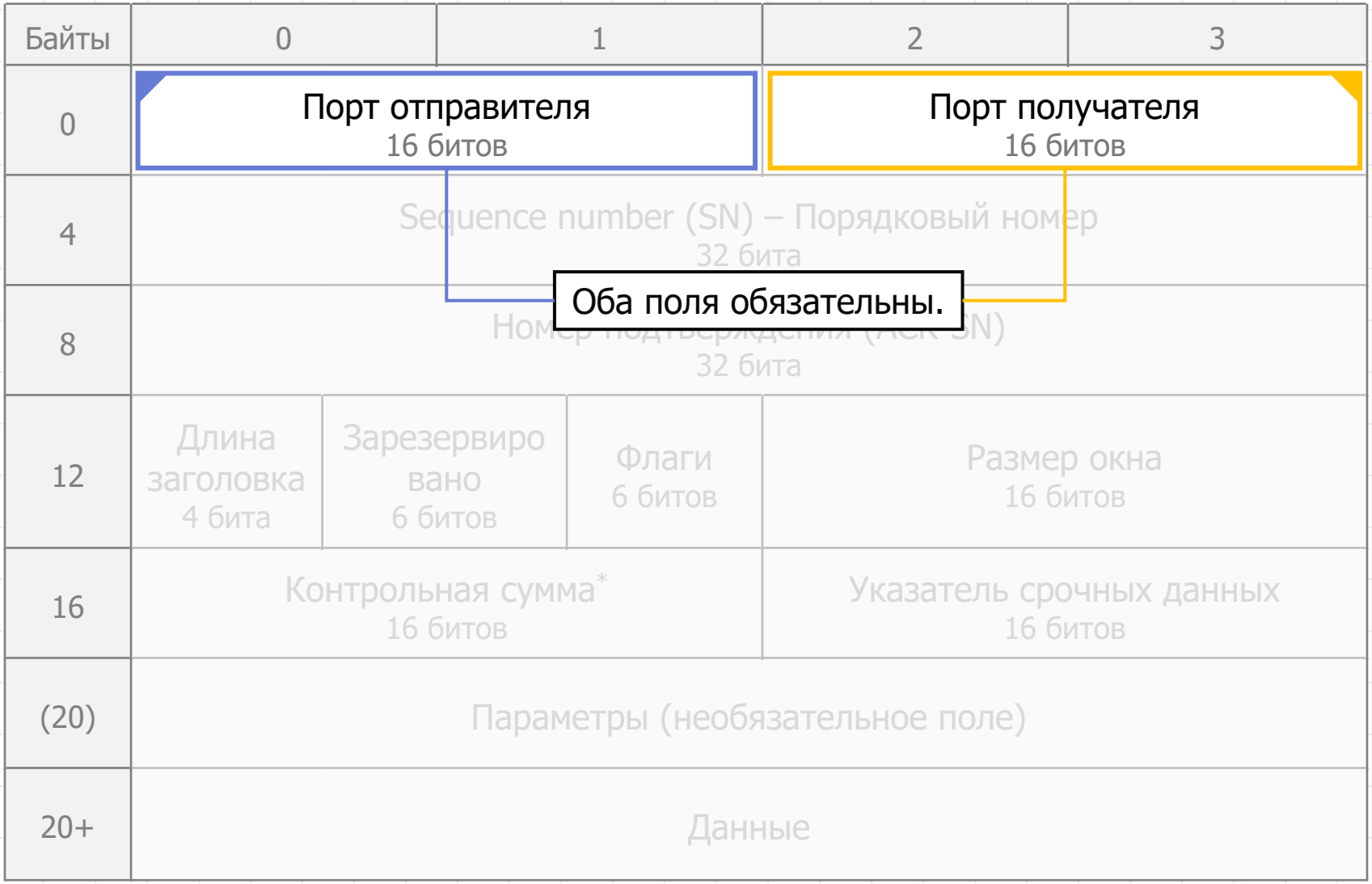
Протокол TCP (англ. Transmission Control Protocol)

- TCP – протокол транспортного уровня, обеспечивающий передачу потока данных:
  - с установкой соединения,
  - с гарантированной (надёжной) доставкой и повторной передачей утерянных пакетов,
  - с подтверждением о доставке (квитирование),
  - с разделением потока данных на сегменты и их сборкой в правильном порядке на принимающей стороне.
- TCP позволяет управлять параметрами передачи данных и динамически подстраивать их под пропускную способность канала.
- TCP обеспечивает дуплексное соединение с двумя потоками данных: «туда» (на передачу) и «обратно» (на приём).

Структура TCP-пакета

Байты	0	1	2	3
0	Порт отправителя 16 битов		Порт получателя 16 битов	
4	Sequence number (SN) – Порядковый номер 32 бита			
8	Номер подтверждения (ACK-SN) 32 бита			
12	Длина заголовка 4 бита	Зарезервировано 6 битов	Флаги 6 битов	Размер окна 16 битов
16	Контрольная сумма* 16 битов		Указатель срочных данных 16 битов	
(20)	Параметры (необязательное поле)			
20+	Данные			

Структура TCP-пакета (продолжение)



Структура TCP-пакета (продолжение)

- Sequence number (SN) – Порядковый номер $^{32\ бита}$ – смещение отправляемого сегмента в общем потоке данных в байтах.
  - Начальное значение смещения для первого сегмента (логический ноль) выбирается при установке соединения.
  - Порядковый номер для следующего сегмента = номер текущего + размер текущего сегмента. Допустимо переполнение 32-разрядного значения смещения, т.е. диапазон «зацикливается» и отсчёт байтов продолжается с 0+, что позволяет передавать поток данных бесконечной длины.
- Номер подтверждения (ACK-SN) $^{32\ бита}$ – (ответ принимающей стороны) квитанция; смещение в общем потоке данных в байтах, до которого все сегменты успешно приняты. Указывает, какой следующий порядковый номер ожидает принимающая сторона.
- TCP-соединение – дуплексное, с двумя потоками данных: «туда» (на передачу) и «обратно» (на приём). Один пакет может содержать и данные для передачи, и квитанцию о приёме.

Структура TCP-пакета (продолжение)

Байты	0	1	2	3
0	Порт отправителя 16 битов		Порт получателя 16 битов	
4	Sequence number (SN) – Порядковый номер 32 бита			
8	Номер подтверждения (ACK-SN) 32 бита			
(20)	Параметры (необязательное поле)			
20+				

Смещение отправляемого сегмента в общем потоке данных в байтах.

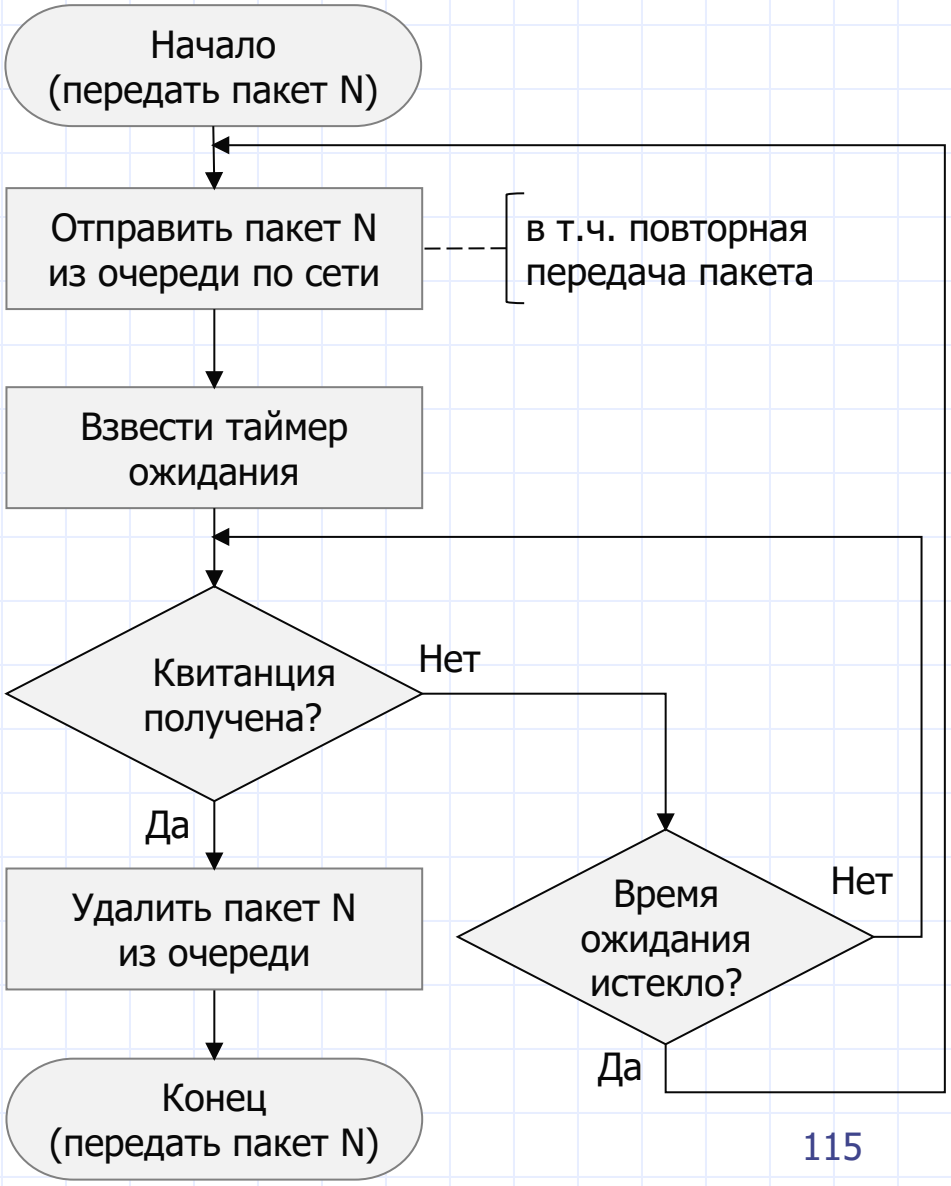
Начальное смещение для первого сегмента (логический ноль) выбирается псевдослучайно при установке соединения.

Поток данных считается бесконечным: при переполнении 32-разрядного значения диапазон «зацикливается» и отсчёт байтов продолжается с 0+.

Квитанция; смещение в общем потоке данных в байтах, до которого все сегменты успешно приняты (флаг ACK).

Доставка сообщения с подтверждением

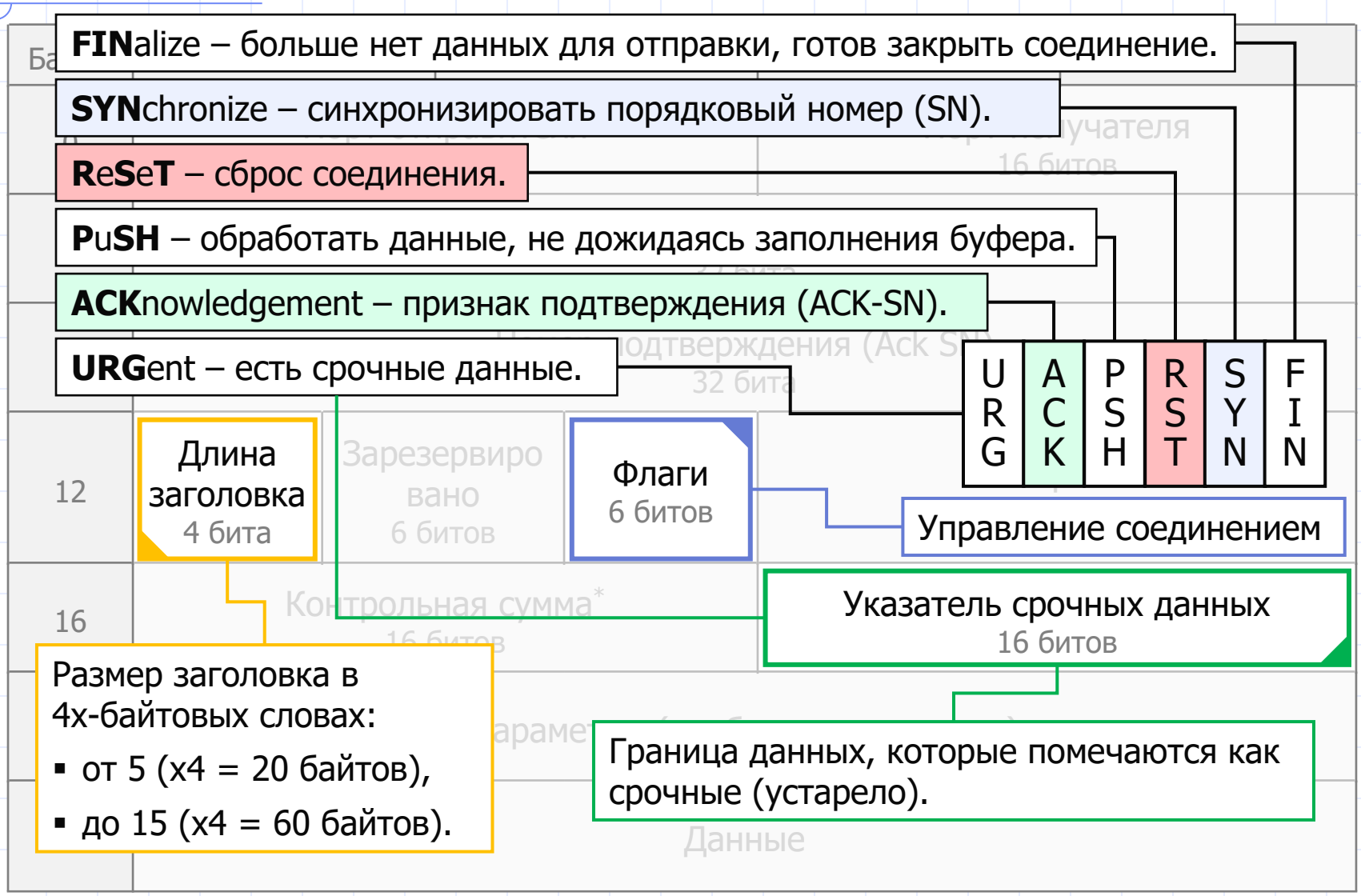
- Данные перед отправкой накапливаются в буфер и разделяются на пакеты (сегменты).
- Пакеты ставятся в очередь на отправку.
- После отправки пакета ожидается квитанция в течение некоторого интервала времени.
- При приёме пакеты также накапливаются в буфер.
- Квитанция может приходить сразу на несколько пакетов.



Структура TCP-пакета (продолжение)

- Длина заголовка 4 бита – размер заголовка TCP в 4х-байтовых словах; минимальное корректное значение – 5 (х4 = 20 байтов).
- Флаги 6 битов – биты, управляющие процессом передачи данных:
  - URG (Urgent) – признак срочных данных; задействует поле Указатель срочных данных.
  - ACK (Acknowledgement) – признак подтверждения; задействует поле Номер подтверждения (ACK-SN).
  - PSH (Push) – обработать данные, не дожидаясь заполнения буфера: для отправляющей стороны – отправить по сети; для принимающей стороны – отдать данные на вышестоящий уровень.
  - RST (Reset) – сброс соединения; применяется в ситуациях, когда произошла рассинхронизация соединения либо необходимо разорвать старое соединение после сбоя системы.
  - SYN (Synchronize) – запрос синхронизации Порядкового номера (SN).
  - FIN (Finalize) – готовность закрыть соединение: больше нет данных для отправки, но т.к. соединение дуплексное, то необходимо продолжать читать до получения FIN от другой стороны.
- Указатель срочных данных 16 битов – граница данных, которые помечаются как срочные (устарело).

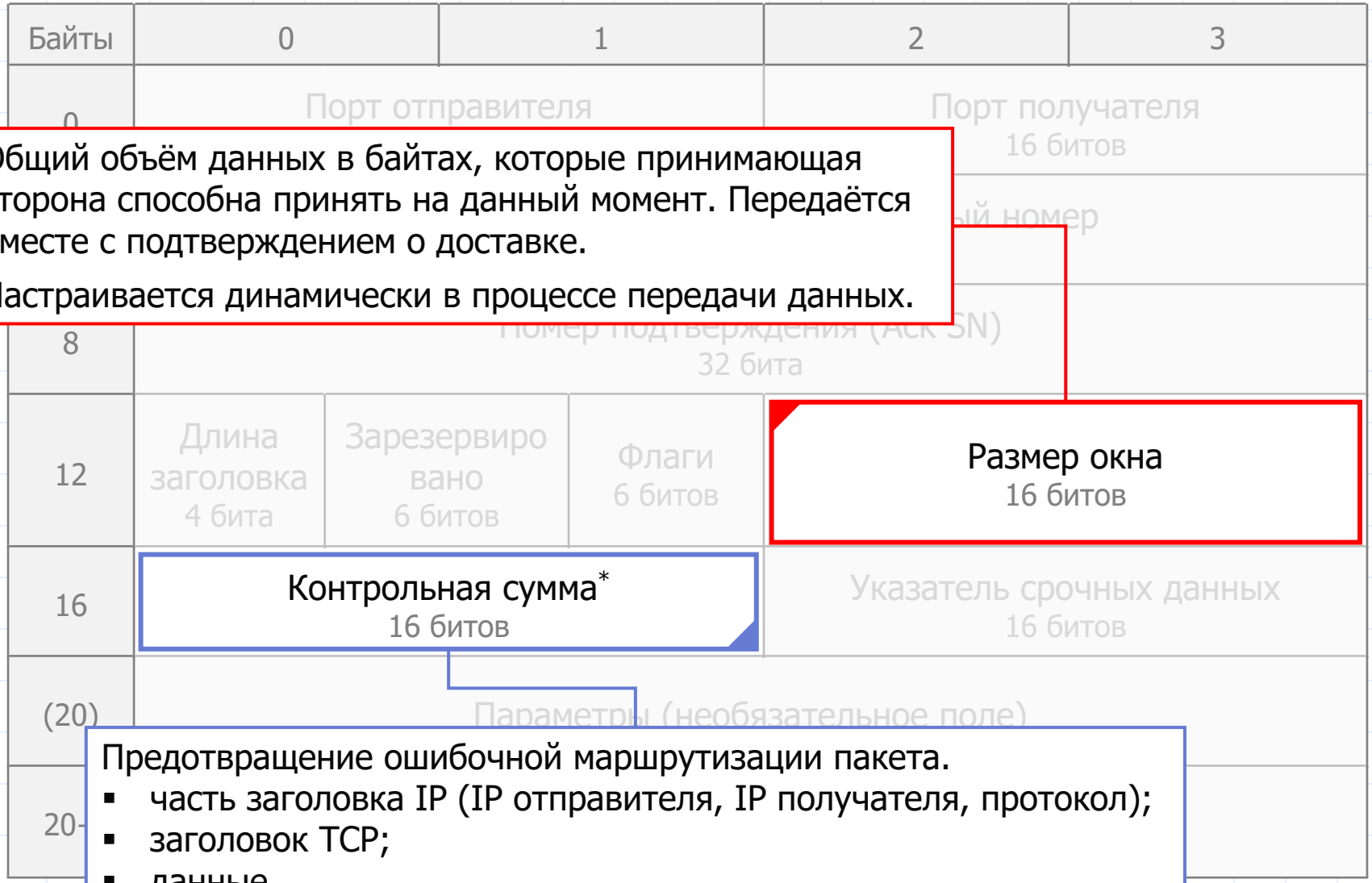
Структура TCP-пакета (продолжение)



Структура TCP-пакета (продолжение)

- Контрольная сумма* 16 битов – для предотвращения ошибочной маршрутизации пакетов; рассчитывается на основании части заголовка IP (IP-адрес отправителя, IP-адрес получателя, протокол), заголовка TCP и данных. Всегда заполняется отправителем и проверяется получателем.
- Размер окна 16 битов – общий объём данных в байтах, которые принимающая сторона способна принять на данный момент. Позволяет «приёмнику» управлять «передатчиком».
  - Передаётся вместе с квитанцией о доставке.
  - Для стороны, которая получает квитанцию, означает, сколько байтов могут быть отправлены и ожидать подтверждение о доставке.
  - Динамически подстраивается под пропускную способность канала в процессе передачи данных.

Структура TCP-пакета (продолжение)



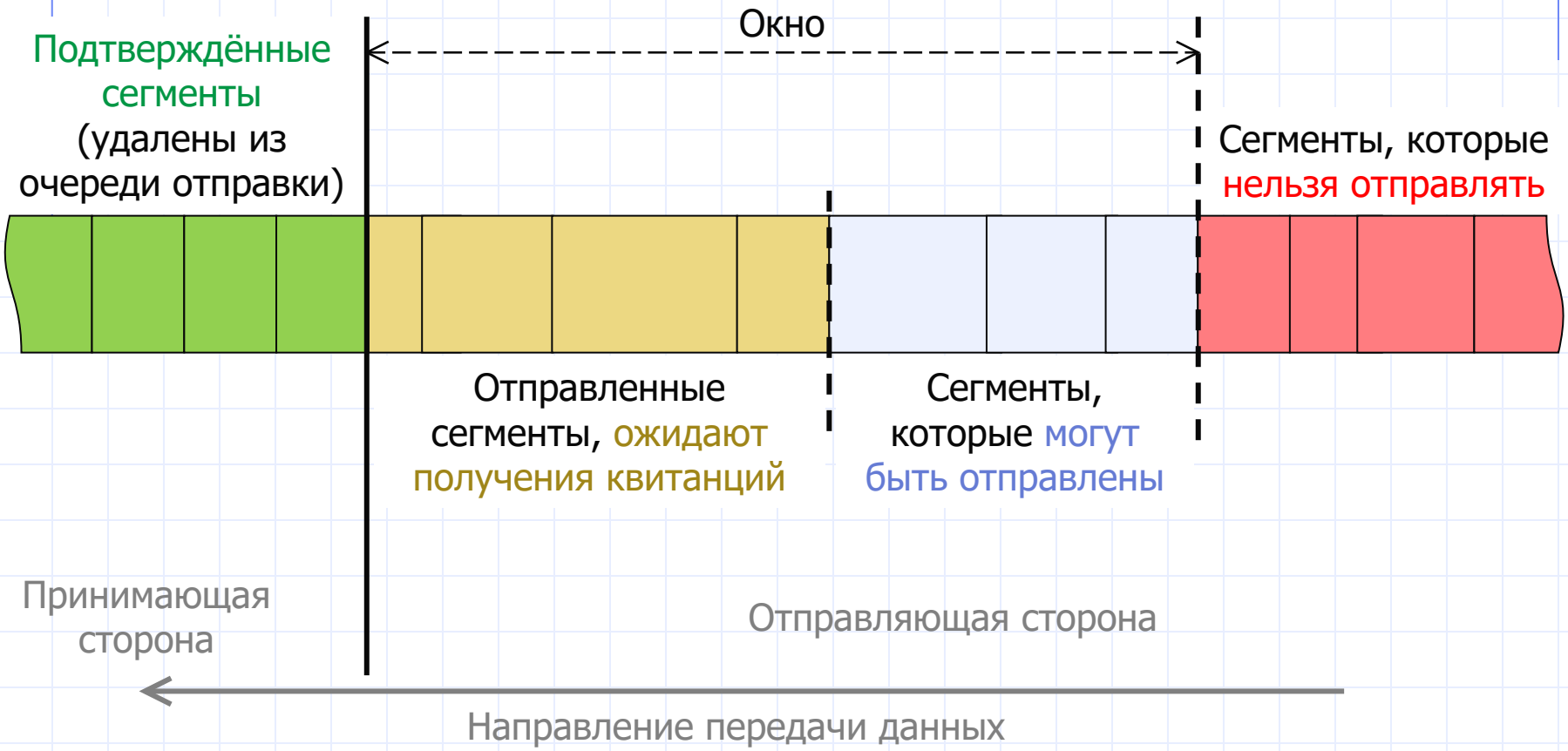
Общий объём данных в байтах, которые принимающая сторона способна принять на данный момент. Передаётся вместе с подтверждением о доставке.

Настраивается динамически в процессе передачи данных.

Предотвращение ошибочной маршрутизации пакета.

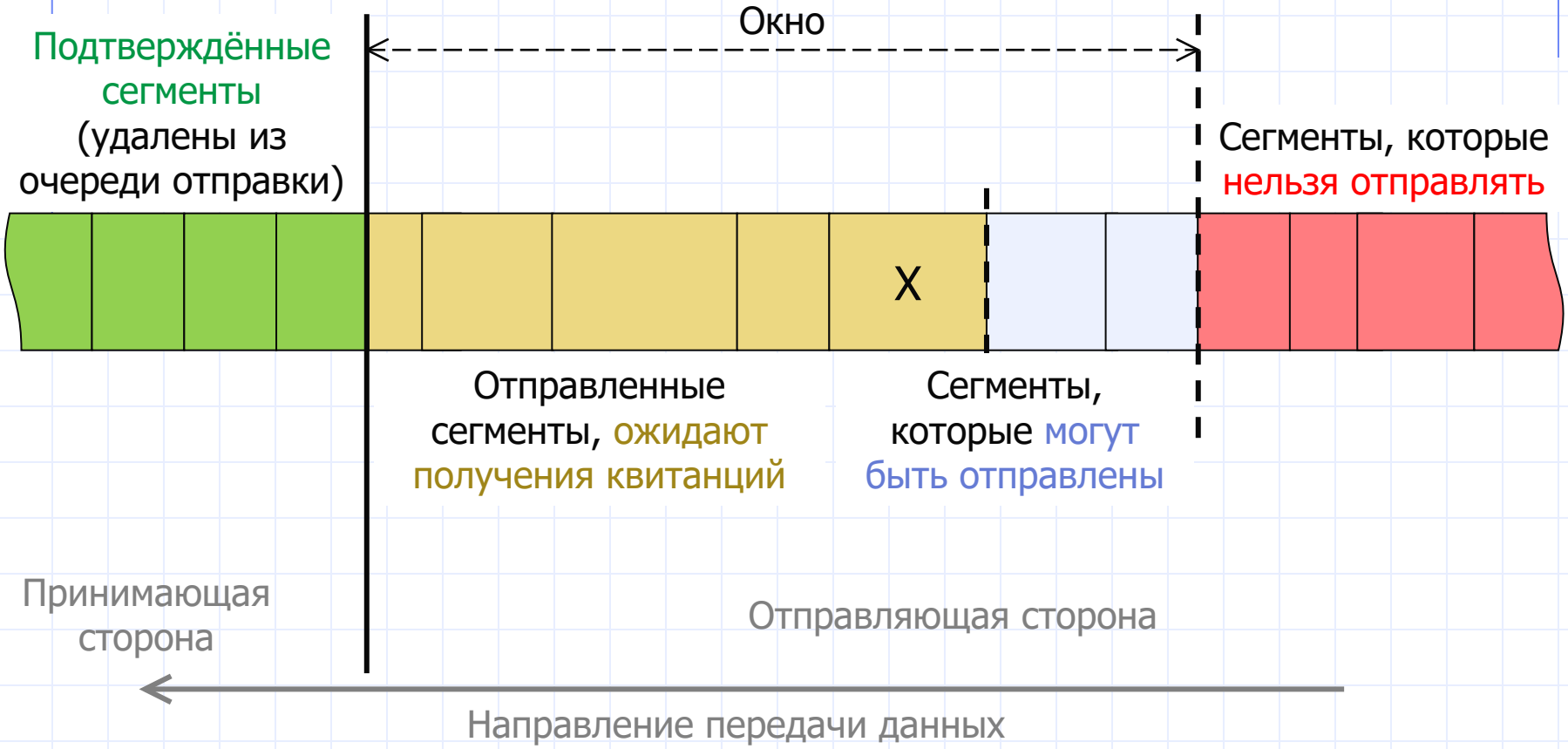
- часть заголовка IP (IP отправителя, IP получателя, протокол);
- заголовок TCP;
- данные.

Механизм скользящего окна



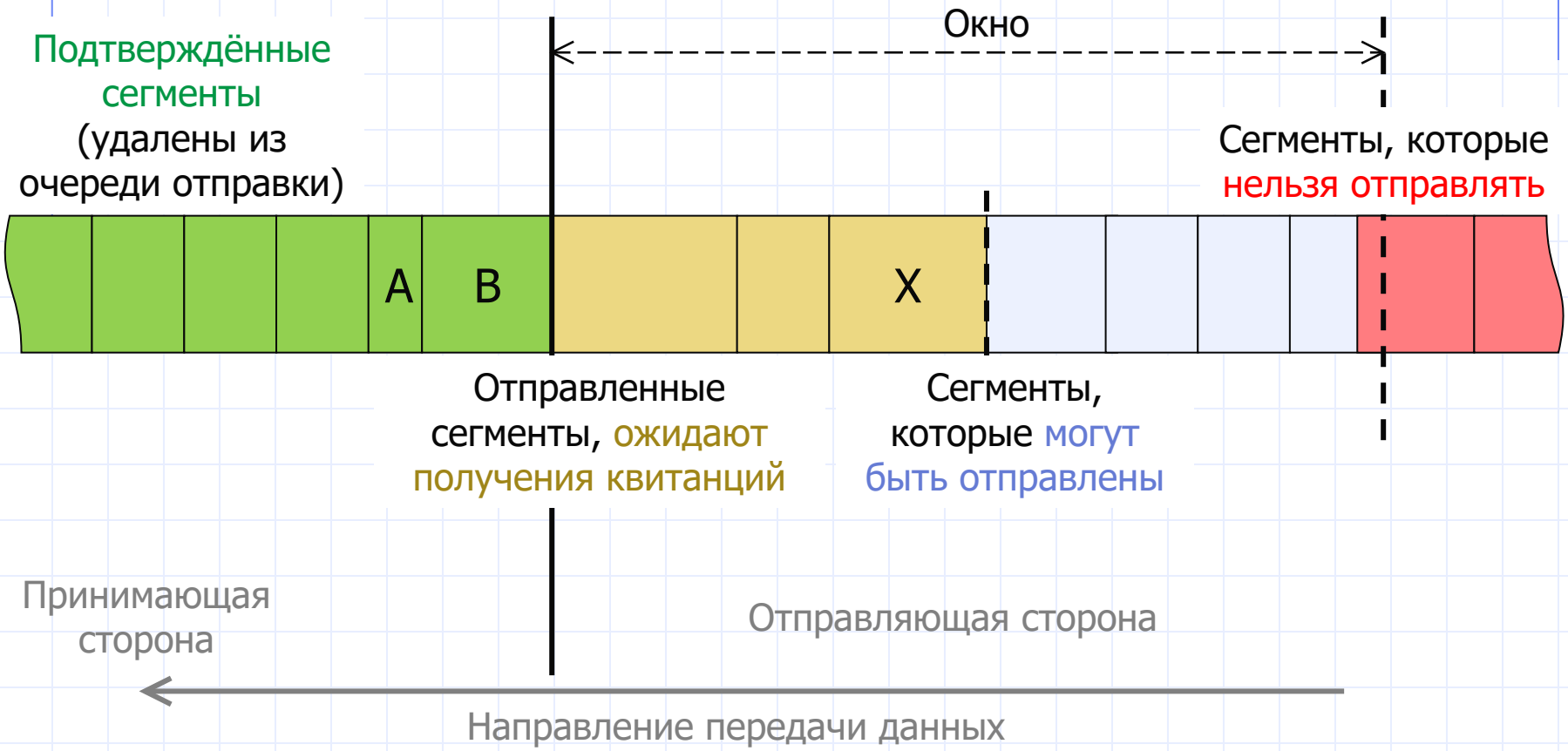
Механизм скользящего окна (продолжение)

- Передан сегмент X



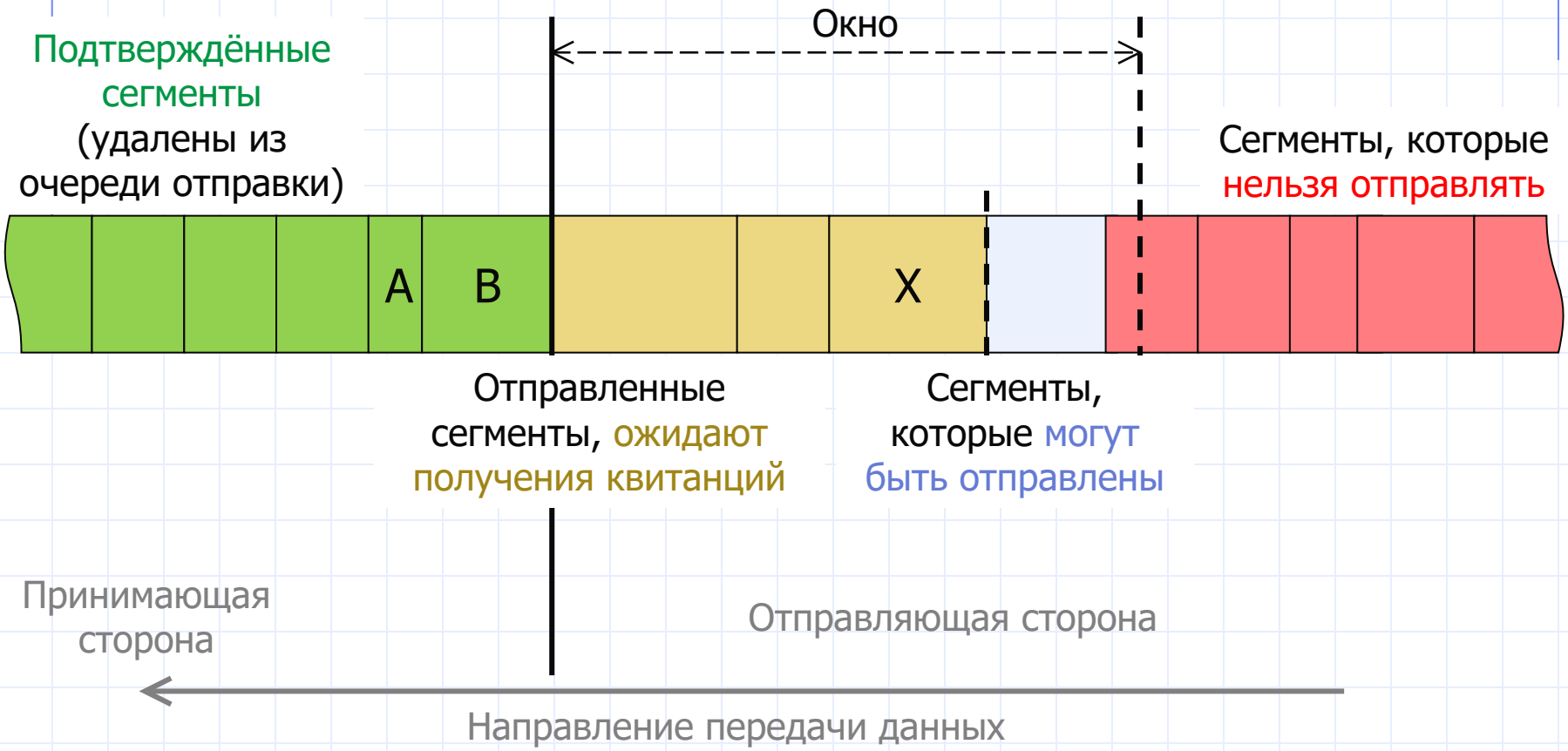
Механизм скользящего окна (продолжение)

- Получена квитанция на A и B



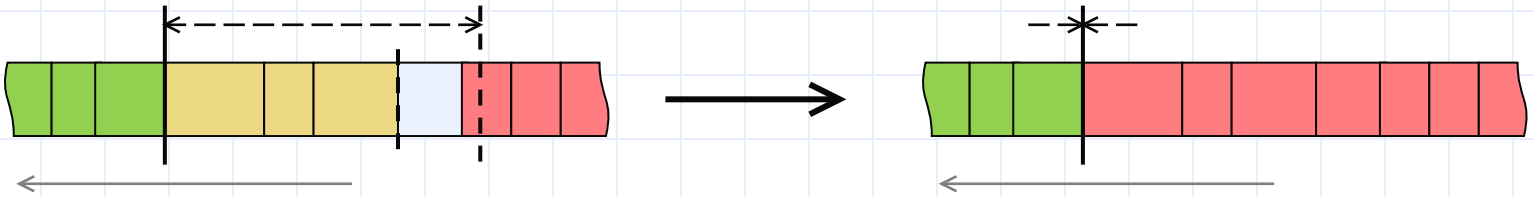
Механизм скользящего окна (продолжение)

- Получен новый размер окна



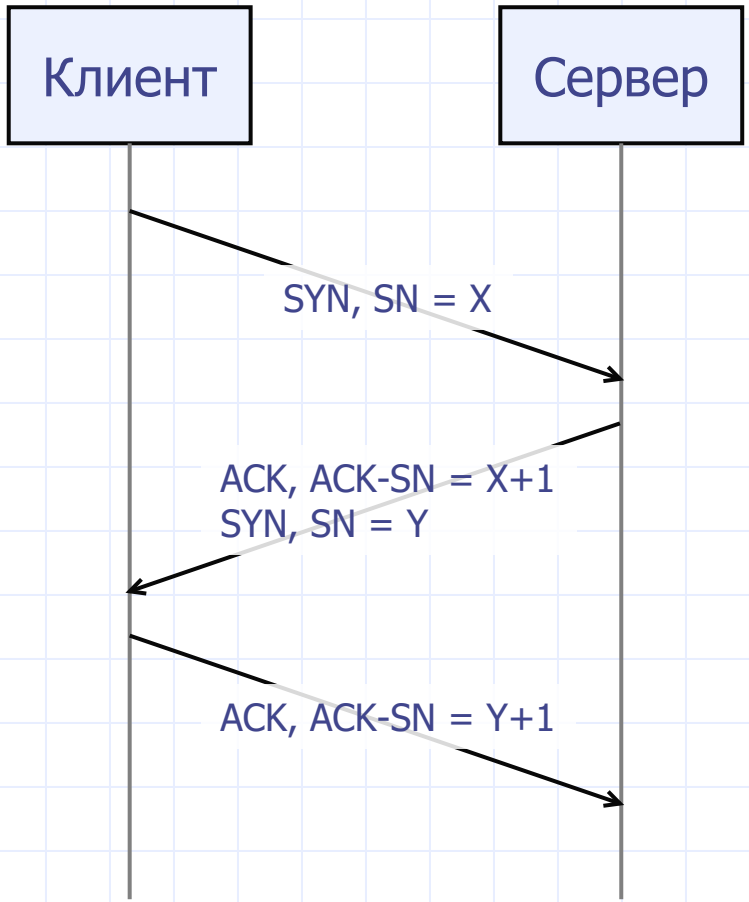
Управление размером окна TCP

- Размер окна зависит от пропускной способности канала и подстраивается на основе адаптивного алгоритма.
- Размер окна используется для согласования скоростей «приёмника» и «передатчика»:
  - если принимающая сторона не готова принять новые сегменты (буфер заполнен, некуда писать), то она устанавливает размер окна равный нулю;
  - отправляющая сторона перестаёт посылать сегменты, пока размер окна равен нулю;
  - принимающая сторона периодически «зондирует» отправляющую сторону, посылая квитанцию на последний принятый байт.



Установка TCP-соединения. Трёхкратное «рукопожатие»

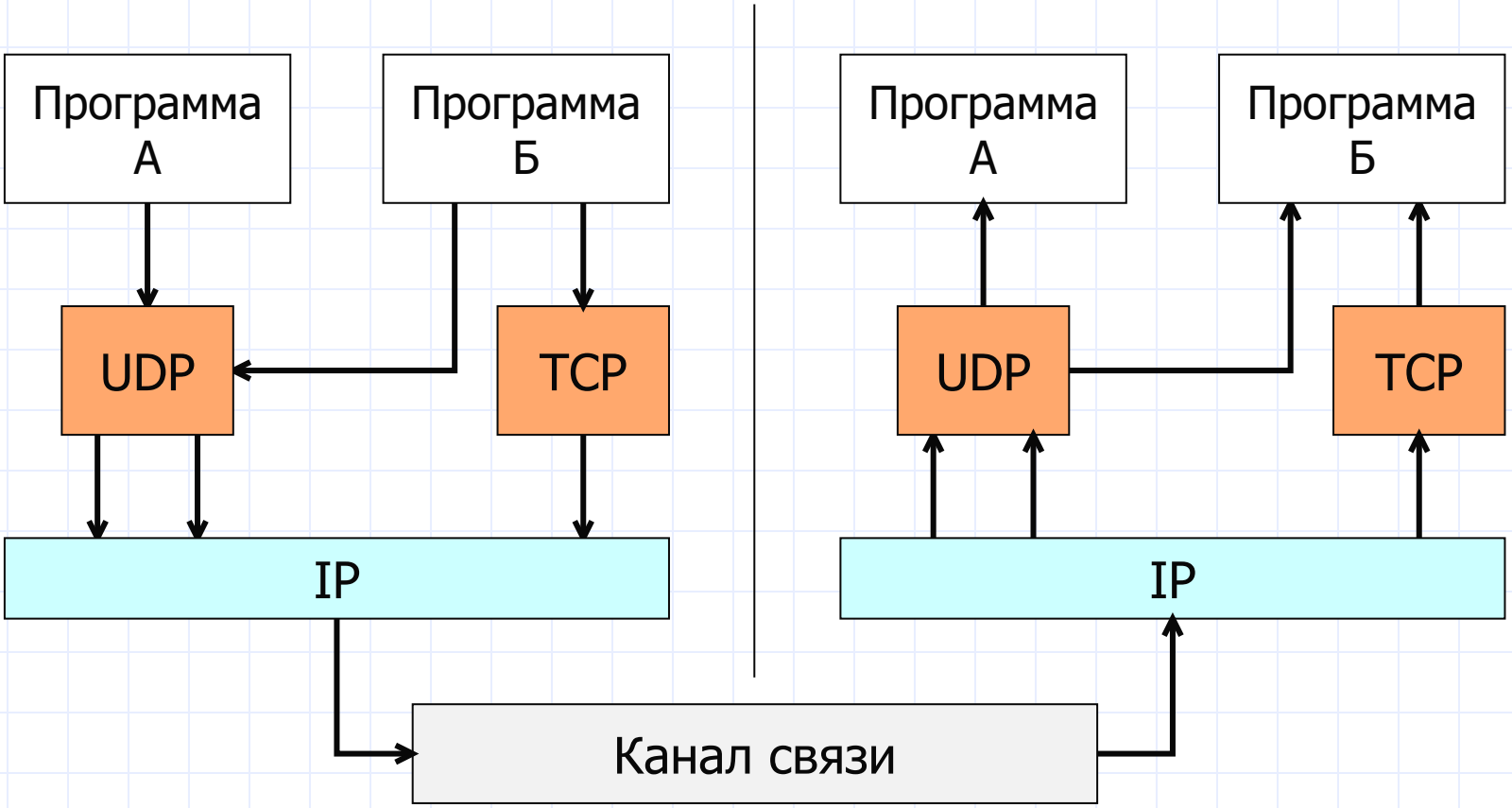
- К → С:
  - флаг SYN;
  - номер последовательности на передачу (X).
- К ← С:
  - флаг ACK;
  - номер подтверждения на передачу (X+1);
  - флаг SYN;
  - номер последовательности на приём (Y).
- К → С
  - флаг ACK;
  - номер последовательности на приём (Y+1).



Установка TCP-соединения. Проблема уязвимости

- TCP разрешает послать запрос на установку соединения на один IP-адрес и порт, а получить подтверждение с другого IP-адреса и порта. Это может использоваться для балансировки соединений.
- Во время установки TCP-соединения значения SN (номер последовательности) является по сути идентификатором абонента.
- Злоумышленник может попытаться угадать значение ACK-SN в ответе от сервера и послать подтверждение на установку соединения вместо сервера, предварительно заставив его «молчать», перегрузив большим количеством запросов на установку соединения.
- В некоторых Unix-серверах очень легко предсказать значение SN, выдаваемое сервером, из-за возникающего «парадокса дней рождений».

Мультиплексирование и демультиплексирование канала связи

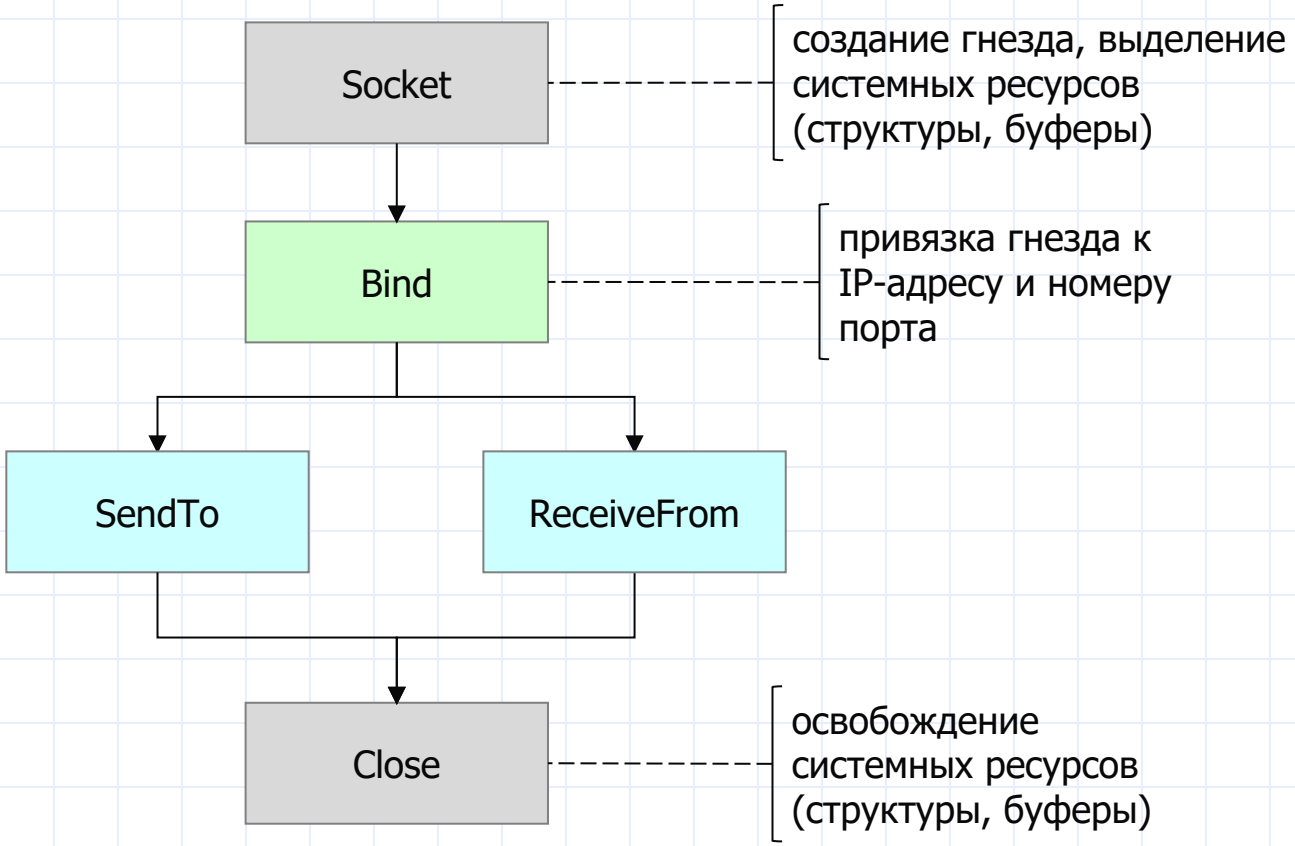


Сеансовый уровень OSI

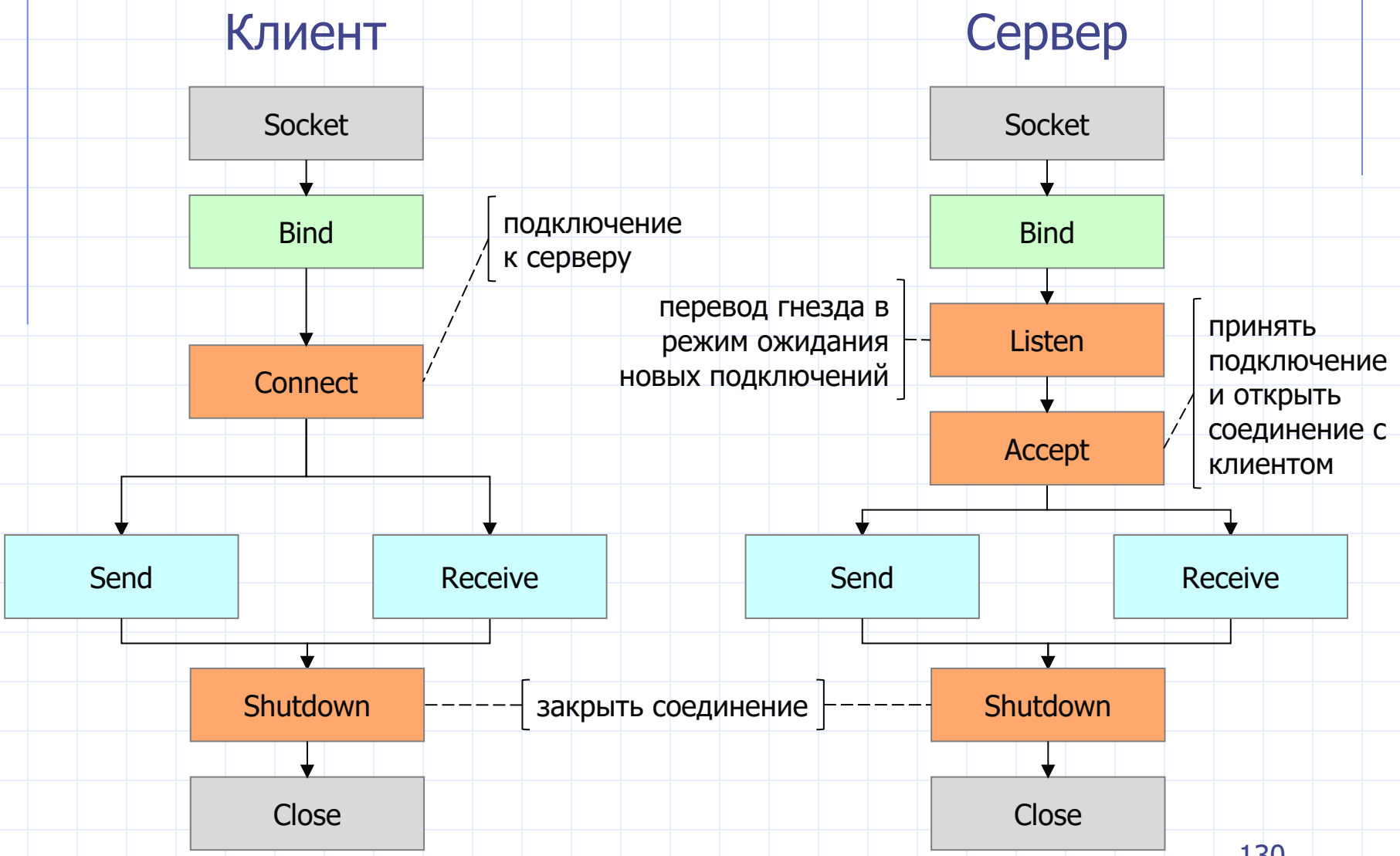
- Обеспечивает сеансы связи между устройствами, запрашивающими и поставляющими услуги.
- Отвечает за установку и поддержку соединений, а также за синхронизацию и управление диалогом между абонентами.
- Помогает верхним уровням идентифицировать доступный сетевой сервис.
- Гнёзда (англ. sockets)

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Гнёзда UDP


Socket
создание гнезда, выделение системных ресурсов (структуры, буферы)
Bind
привязка гнезда к IP-адресу и номеру порта
SendTo
ReceiveFrom
Close
освобождение системных ресурсов (структуры, буферы)

Гнёзда TCP


Клиент
Сервер
Socket
Bind
Connect
Send
Receive
Shutdown
Close
Socket
Bind
Listen
Accept
Send
Receive
Shutdown
Close
подключение к серверу
перевод гнезда в режим ожидания новых подключений
принять подключение и открыть соединение с клиентом
закрыть соединение

Программный интерфейс сетевых гнёзд Sockets

- **Socket** – создаёт гнездо с необходимыми ресурсами – буферами, таймерами, счётчиками и пр.
- **Bind** – связывает гнездо со своим локальным IP-адресом и портом.
- **Listen** – создаёт очередь заданного размера для приёма входящих запросов на установку соединения и переводит гнездо в состояние прослушивания. Никакого ожидания не выполняет.
  - Пришедшие запросы остаются в очереди пока не будет вызвана функция Accept. Для них периодически отправляются пакеты с флагом ACK (но без флага SYN).
  - Очередь не может быть большой.
- **Accept** – ждёт появления запроса на установку соединения, затем выбирает его из очереди, подтверждает его, отсылая пакет с флагами SYN и ACK, и дожидается ответного подтверждения. Создаёт и возвращает вызывающей стороне новое гнездо, которое служит для обмена данными с клиентом.
- **Connect** – посылает запрос на установку соединения на указанный IP-адрес и порт.

…

Программный интерфейс сетевых гнёзд Sockets (продолжение)

- **Send** – отсылает данные по установленному соединению.
- **SendTo** – отсылает данные абоненту с заданными IP-адресом и портом.
- **Receive** – принимает данные по установленному соединению.
- **ReceiveFrom** – принимает данные и сообщает IP-адрес и порт абонента от которого они пришли.

- **Shutdown** – закрывает дуплексное соединение в одну сторону или в две стороны.
- **Close** – закрывает гнездо и освобождает выделенные ему ресурсы (буферы, таймеры, счётчики).

Интерфейс System.Net.Sockets.Socket (на языке C# для платформы .NET)

```
public enum AddressFamily
{
    Unspecified = 0,
    Unix = 1,
    InterNetwork = 2,
    ImpLink = 3,
    Pup = 4,
    Chaos = 5,
    NS = 6,
    Ipx = 6,
    Iso = 7,
    Osi = 7,
    Ecma = 8,
    DataKit = 9,
    Ccitt = 10,
    Sna = 11,
    DecNet = 12,
    DataLink = 13,
    Lat = 14,
    HyperChannel = 15,
    AppleTalk = 16,
    NetBios = 17,
    VoiceView = 18,
    FireFox = 19,
    Banyan = 21,
    Atm = 22,
    InterNetworkV6 = 23,
    Cluster = 24,
    Ieee12844 = 25,
    Irda = 26,
    NetworkDesigners = 28,
}
```

```
public enum ProtocolType
{
    IP = 0,
    IPv6HopByHopOptions = 0,
    Unspecified = 0,
    Icmp = 1,
    Igmp = 2,
    Ggp = 3,
    IPv4 = 4,
    Tcp = 6,
    Pup = 12,
    Udp = 17,
    Idp = 22,
    IPv6 = 41,
    IPv6RoutingHeader = 43,
    IPv6FragmentHeader = 44,
    IPSecEncapsulatingSecurityPayload = 50,
    IPSecAuthenticationHeader = 51,
    IcmpV6 = 58,
    IPv6NoNextHeader = 59,
    IPv6DestinationOptions = 60,
    ND = 77,
    Raw = 255,
    Ipx = 1000,
    Spx = 1256,
    SpxII = 1257
}
```

```
public enum SocketType
{
    Unknown = -1,
    Stream = 1,
    Dgram = 2,
    Raw = 3,
    Rdm = 4,
    Seqpacket = 5
}
```

```
public enum SocketShutdown
{
    Receive = 0,
    Send = 1,
    Both = 2
}
```

```
public class IPEndPoint
{
    public IPEndPoint(long address, int port);
    public IPEndPoint(IPAddress address, int port);
    public IPAddress Address { get; set; }
    public int Port { get; set; }
}
```

Интерфейс System.Net.Sockets.Socket (продолжение)

```
public class Socket : IDisposable
{
    public Socket(AddressFamily addressFamily, SocketType socketType, ProtocolType protocolType);
    public EndPoint? LocalEndPoint { get; }
    public EndPoint? RemoteEndPoint { get; }
    public bool IsBound { get; }
    public bool Blocking { get; set; }
    public bool Connected { get; }
    public bool DontFragment { get; set; }
    public short Ttl { get; set; }
    public int SendTimeout { get; set; }
    public int SendBufferSize { get; set; }
    public int ReceiveBufferSize { get; set; }
    public int ReceiveTimeout { get; set; }
    public void Bind(EndPoint localEP);
    public void Connect(EndPoint remoteEP);
    public void Listen(int backlog);
    public Socket Accept();
    public bool Poll(int microSeconds, SelectMode mode);
    public static void Select(IList? checkRead, IList? checkWrite, IList? checkError, int microSeconds);
    public int Receive(byte[] buffer, int size, SocketFlags socketFlags);
    public int Receive(byte[] buffer);
    public int ReceiveFrom(byte[] buffer, int size, SocketFlags socketFlags, ref EndPoint remoteEP);
    public int ReceiveFrom(byte[] buffer, ref EndPoint remoteEP);
    public int Send(byte[] buffer);
    public int Send(byte[] buffer, int size, SocketFlags socketFlags, out SocketError errorCode);
    public int SendTo(byte[] buffer, EndPoint remoteEP);
    public int SendTo(byte[] buffer, int size, SocketFlags socketFlags, EndPoint remoteEP);
    public void Shutdown(SocketShutdown how);
    public void Close();
    public void Close(int timeout);
}
```

Пример TCP-сервера (на языке C# для платформы .NET)

```
namespace TcpServer
{
    class Program
    {
        static string ipAddress = "127.0.0.1"; // адрес сервера для локальных клиентов
        static int port = 5555; // порт для приёма входящих запросов
        static void Main(string[] args)
        {
            // Создаём точку доступа к службе (адрес и порт)
            IPEndPoint ipPoint = new IPEndPoint(IPAddress.Parse(ipAddress), port);
            // Создаём гнездо
            Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            // Связываем гнездо с точкой доступа
            socket.Bind(ipPoint);
            // Переводим гнездо в режим прослушивания (создаём очередь запросов на подключение клиентов)
            socket.Listen(10);
            Console.WriteLine("Сервер ожидает подключений...");
            while (true)
            {
                // Ожидаем запроса на подключение клиента и принимаем его
                Socket clientSocket = socket.Accept();
                // Взаимодействуем с клиентом
                CommunicateWithClient(clientSocket);
                // Закрываем дуплексное соединение с клиентом
                clientSocket.Shutdown(SocketShutdown.Both);
                // Уничтожаем гнездо и освобождаем его ресурсы (буферы, таймеры, счётчики)
                clientSocket.Close();
            }
        }
        ...
    }
}
```

Пример TCP-сервера (продолжение)

```
namespace TcpServer
{
    class Program
    {
        ...
        static void CommunicateWithClient(Socket clientSocket)
        {
            StringBuilder inputMessage = new StringBuilder(); // накапливаемое сообщение
            int bytesRead = 0; // количество байтов, полученных за одну операцию чтения
            byte[] inputData = new byte[256]; // буфер для получаемых данных
            // Получаем данные в цикле, пока гнездо не закроется на чтение
            do
            {
                // Читаем пришедшую порцию данных
                bytesRead = clientSocket.Receive(inputData);
                // Перекодируем данные из формата UTF8 в строку в формате Unicode
                string text = Encoding.UTF8.GetString(inputData, 0, bytesRead);
                // Добавляем принятый текст (строку в формате Unicode)
                inputMessage.Append(text);
            }
            while (bytesRead > 0);

            // Выводим на экран текст принятого сообщения для отладки
            Console.WriteLine(DateTime.Now.ToShortTimeString() + ": " + inputMessage.ToString());

            string outputMessage = "Ваше сообщение доставлено"; // ответ клиенту
            // Перекодируем строку в формате Unicode в данные в формате UTF8
            byte[] outputData = Encoding.UTF8.GetBytes(outputMessage);
            // Отправляем данные клиенту (цикл отправки порциями реализует .NET)
            clientSocket.Send(outputData);
        }
    }
}
```

Пример TCP-клиента (на языке C# для платформы .NET)

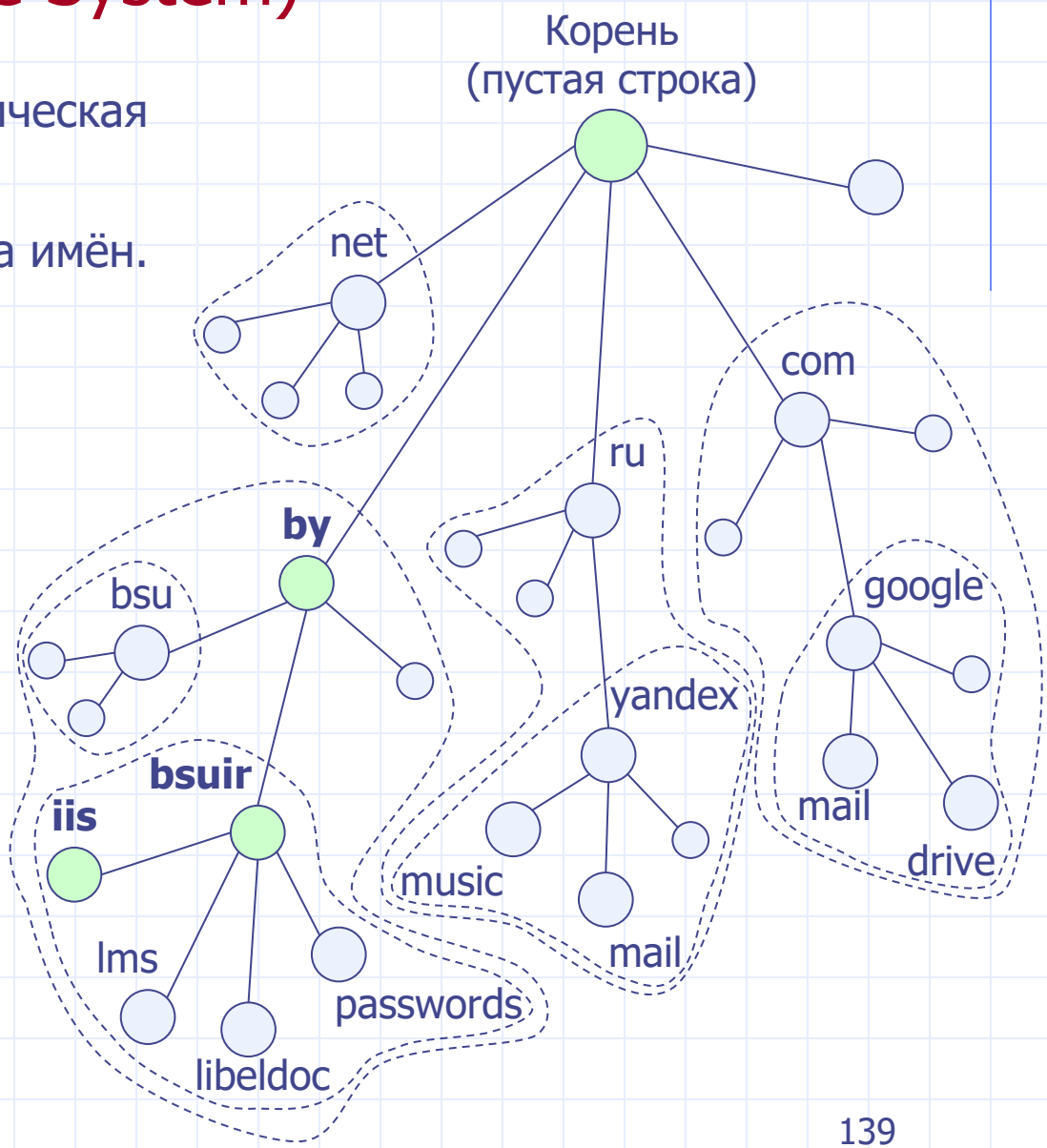
```
namespace TcpClient
{
    class Program
    {
        static string ipAddress = "127.0.0.1"; // адрес сервера
        static int port = 5555; // порт сервера
        static void Main(string[] args)
        {
            // Создаём точку доступа к службе (адрес и порт)
            IPEndPoint ipPoint = new IPEndPoint(IPAddress.Parse(ipAddress), port);
            // Создаём гнездо
            Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            // Подключаемся к серверу. IP-адрес и порт клиенту назначаются автоматически
            socket.Connect(ipPoint);

            // Взаимодействуем с сервером
            CommunicateWithServer(socket);

            // Закрываем дуплексное соединение с сервером
            socket.Shutdown(SocketShutdown.Both);
            // Уничтожаем гнездо и освобождаем его ресурсы (буферы, таймеры, счётчики)
            socket.Close();

            Console.Read();
        }
        ...
    }
}
```

Пример TCP-клиента (продолжение)

```
namespace TcpClient
{
    class Program
    {
        ...
        static void CommunicateWithServer(Socket socket)
        {
            Console.Write("Введите сообщение: ");
            string outputMessage = Console.ReadLine(); // сообщение серверу
            // Перекодируем текст в формате Unicode в данные в формате UTF8
            byte[] outputData = Encoding.UTF8.GetBytes(outputMessage);
            socket.Send(outputData);
            // Закрываем соединение на отправку данных
            socket.Shutdown(SocketShutdown.Send);
            StringBuilder inputMessage = new StringBuilder(); // накапливаемое сообщение
            int bytesRead = 0; // количество байтов, полученных за одну операцию чтения
            byte[] inputData = new byte[256]; // буфер для получаемых данных
            // Получаем данные в цикле пока гнездо не закроется на чтение
            do
            {
                // Читаем пришедшую порцию данных
                bytesRead = socket.Receive(inputData);
                // Перекодируем данные из формата UTF8 в строку в формате Unicode
                string text = Encoding.UTF8.GetString(inputData, 0, bytesRead);
                // Добавляем принятый текст (строку в формате Unicode)
                inputMessage.Append(text);
            }
            while (bytesRead > 0);
            // Выводим на экран текст принятого сообщения для отладки
            Console.WriteLine("Ответ сервера: " + inputMessage.ToString());
        }
    }
}
```

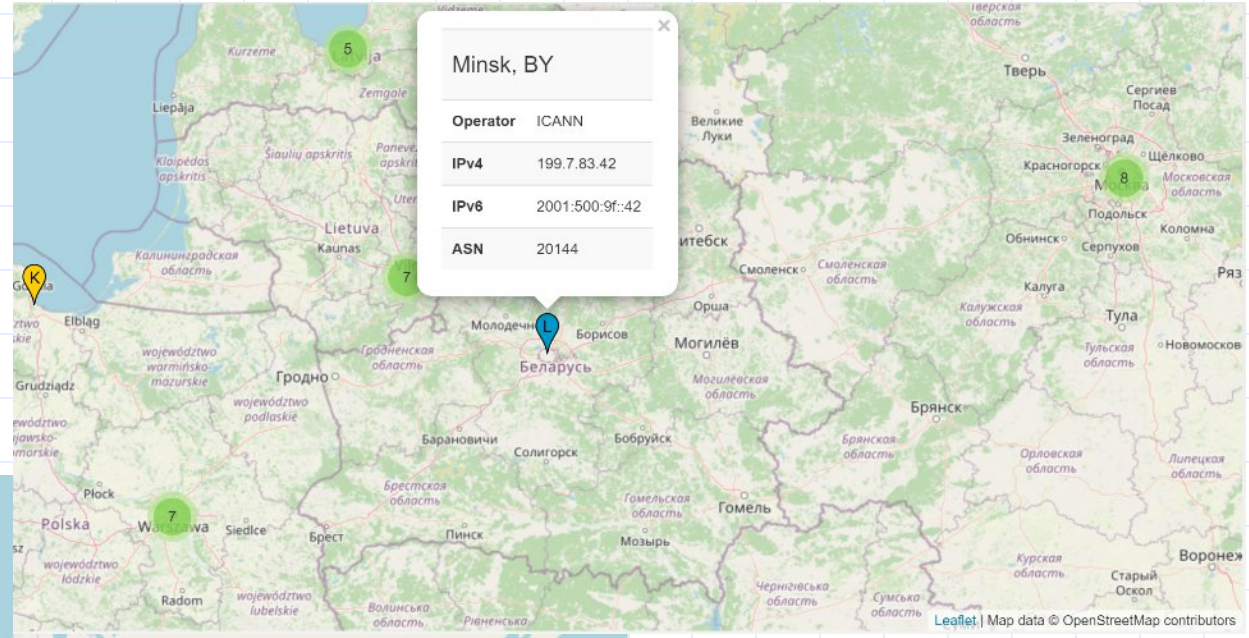
Система доменных имён DNS (англ. Domain Name System)

- Распределённая иерархическая база данных.
- Иерархическая структура имён. Пример: **iis.bsuir.by.**
- Домены первого уровня зафиксированы.
- Типы записей
  - A – имя в IPv4
  - AAAA – имя в IPv6
  - NS – имя DNS-сервера для домена
  - MX – имя почтового сервера для домена
  - и др.
- Утилита **`nslookup`**



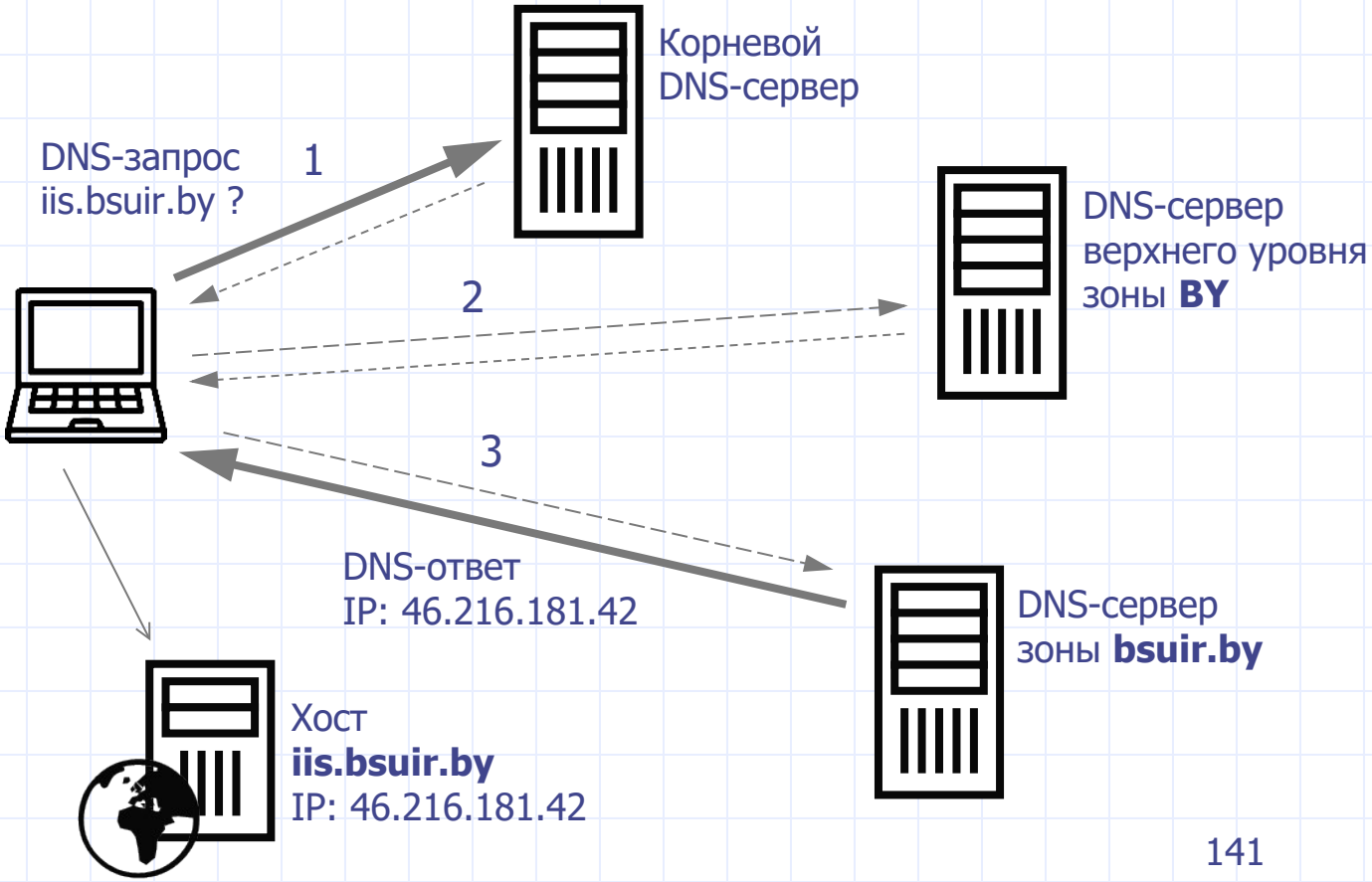
Карта корневых DNS-серверов

https://www.iana.org/domains/root/servers



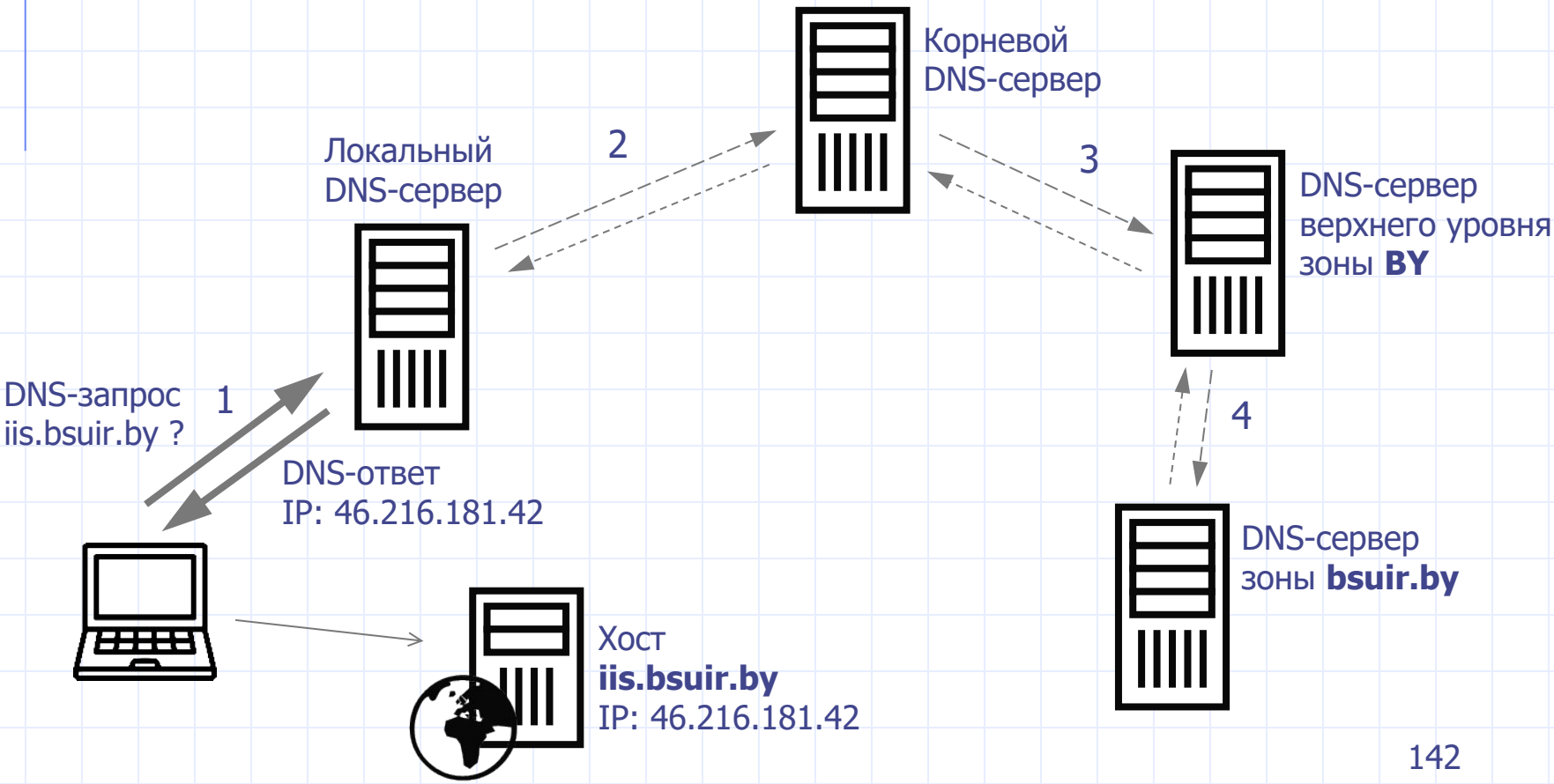
Итеративный DNS-запрос

- DNS-клиент опрашивает DNS-сервера по очереди.



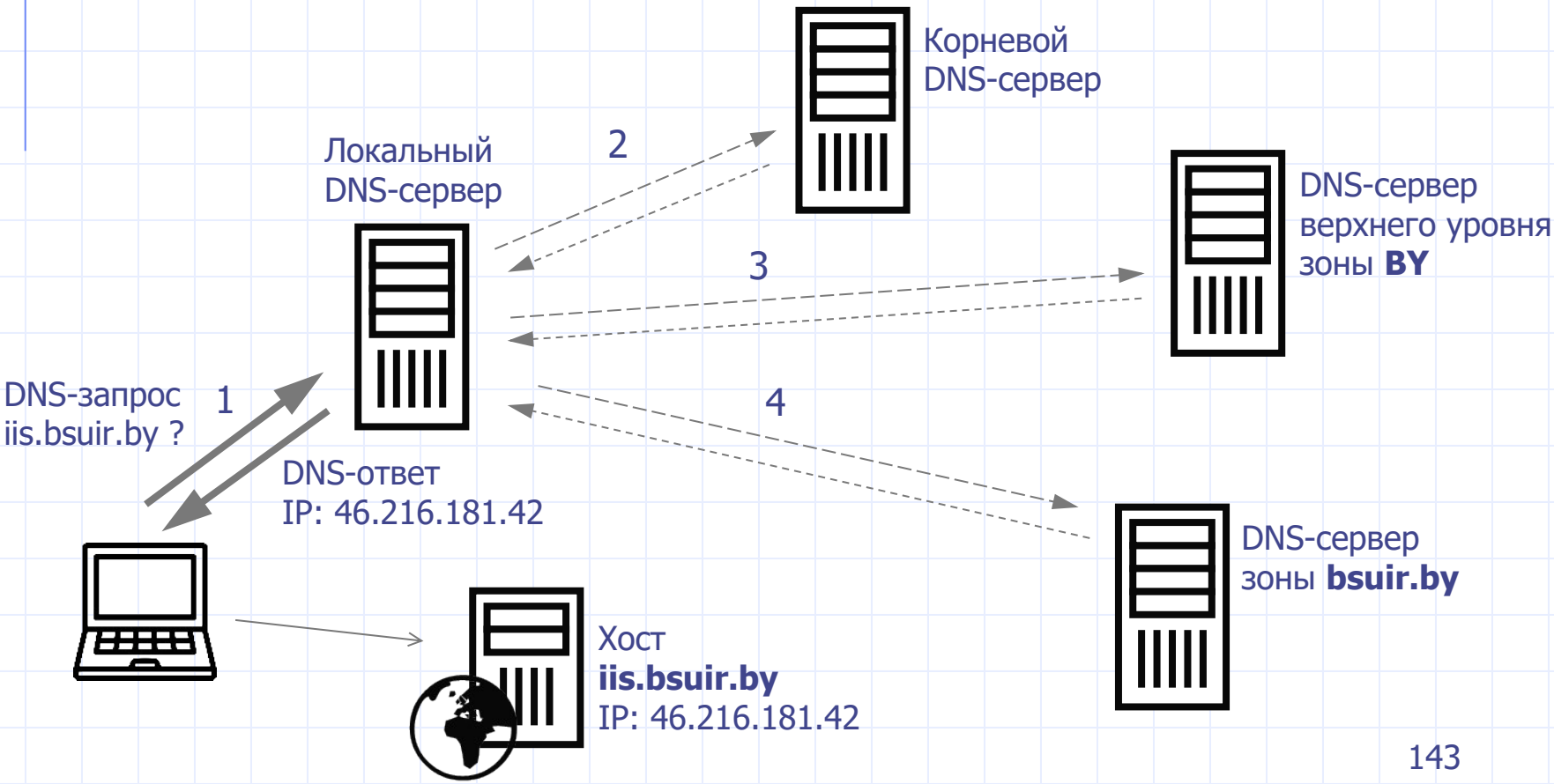
Рекурсивный DNS-запрос

- DNS-клиент перепоручает работу DNS-серверам.



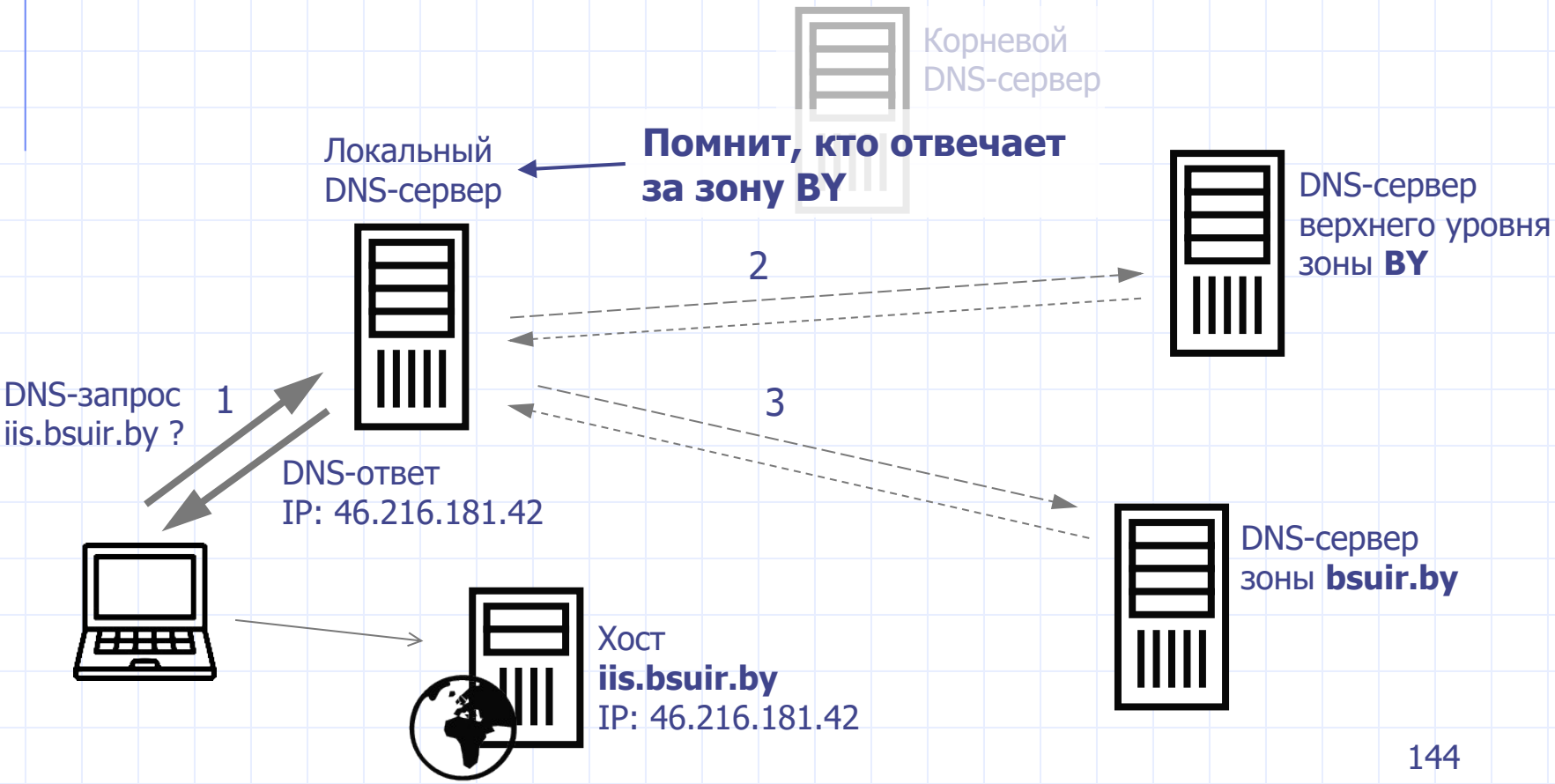
Смешанный DNS-запрос

- DNS-клиент делает рекурсивный DNS-запрос к локальному DNS-серверу, который выполняет итеративный DNS-запрос.



Кэширование DNS

- Ответы DNS кэшируются локальным DNS-сервером.
- Снижает нагрузку на корневые DNS-сервера.



Получение информации о домене

- Протокол WHOIS – получение информации о домене:
  - дата регистрации,
  - срок действия,
  - информация о владельце (название организации, адрес, электронная почта, телефон) и др.
- Порт: 43
- Запрос: строка формата

  <*domain*> \r \n
- Ответ: произвольная строка

Пример WHOIS-клиента (на языке C# для платформы .NET)

```
namespace TcpClient
{
    class Program
    {
        static int port = 43; // порт WHOIS-сервера
        static void Main(string[] args)
        {
            // Список who-is серверов: https://snipp.ru/handbk/whois-servers
            Console.Write("Введите имя WHOIS-сервера (например, whois.iana.org): ");
            string serverName = Console.ReadLine();
            // Обращаемся к службе DNS для получения IP-адреса по имени сервера
            IPHostEntry hostEntry = Dns.GetHostEntry(serverName);
            // Выбираем из полученного в DNS-ответе списка первый IP-адрес
            IPAddress ipAddress = hostEntry.AddressList[0];
            Console.WriteLine("Адрес сервера: " + ipAddress.ToString());
            // Создаем гнездо
            Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            // Подключаемся к серверу. IP-адрес и порт клиенту назначаются автоматически
            socket.Connect(new IPEndPoint(ipAddress, port));

            // Взаимодействуем с сервером
            CommunicateWithServer(socket);

            // Закрываем дуплексное соединение с сервером
            socket.Shutdown(SocketShutdown.Both);
            // Уничтожаем гнездо и освобождаем его ресурсы (буферы, таймеры, счётчики)
            socket.Close();

            Console.Read();
        }
        ...
    }
}
```

Пример WHOIS-клиента (продолжение)

```
namespace TcpClient
{
    class Program
    {
        ...
        static void CommunicateWithServer(Socket socket)
        {
            Console.Write("Введите запрос: ");
            string request = Console.ReadLine() + "\r\n"; // WHOIS-запрос должен оканчиваться "\r\n"
            // Перекодируем текст в формате Unicode в данные в формате UTF8
            byte[] outputData = Encoding.UTF8.GetBytes(request);
            socket.Send(outputData);

            StringBuilder response = new StringBuilder(); // накапливаемое сообщение
            int bytesRead = 0; // количество байтов, полученных за одну операцию чтения
            byte[] inputData = new byte[256]; // буфер для получаемых данных
            // Получаем данные в цикле пока гнездо не закроется на чтение
            do
            {
                // Читаем пришедшую порцию данных
                bytesRead = socket.Receive(inputData);
                // Перекодируем данные из формата UTF8 в строку в формате Unicode
                string text = Encoding.UTF8.GetString(inputData, 0, bytesRead);
                // Добавляем принятый текст (строку в формате Unicode)
                response.Append(text);
            }
            while (bytesRead > 0);
            // Выводим на экран текст принятого сообщения
            Console.WriteLine("\n" + response.ToString());
        }
    }
}
```

Протокол IPv6. Сравнение с протоколом IPv4

- Новая версия призвана решить проблемы IPv4:
  - недостаточная разрядность IP-адреса;
  - отсутствие средств безопасности;
  - большие накладные расходы на маршрутизаторах.
- Адреса длиной 128 битов (16 байтов) вместо 32 битов (4 байтов).
- Удалены функции, которые усложняли работу маршрутизаторов, например, маршрутизаторы больше не разбивают пакет на фрагменты.
- Появились заголовки расширений, при этом минимальный заголовок пакета удлинился с 20 байтов до всего 40 байтов.
- Добавлены средства шифрования и безопасности.
- Появились метки потоков.
- Добавилось многоадресное вещание.
- Поддержка сверхбольших дейтаграмм (англ. jambogram) с теоретическим пределом 4 ГБ.

Формат IPv6-адреса

- Формат адреса – 8 шестнадцатеричных числа в диапазоне от 0000 до FFFF, разделённых двоеточием:

  2001:0db8:85a3:08d3:1319:8a2e:0370:7348

- Все ведущие нули можно опустить:

  0db8 → db8

- Одну последовательную группу нулей можно сократить до ::

  2001:db8:85a3:0:0:0:370:7334 → 2001:db8:85a3::370:7334

  0:0:0:0:0:0:0:1 → ::1

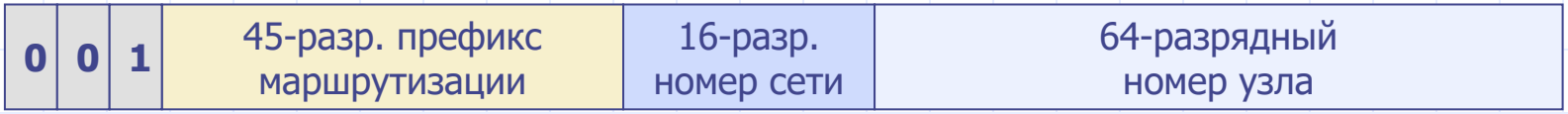
  0:0:0:0:0:0:0:0 → ::

- При использовании в URL IPv6-адрес заключают в [ ]

  http://[2001:0db8:11a3:09d7:1f34:8a2e:07a0:765d]/

Структура IPv6-адреса

- Структура глобально уникальных IPv6-адресов:



- IPv6 адрес структурный, иерархический. Содержит
  - префикс маршрутизации,
  - номер подсети,
  - номер узла.
- Номер узла может содержать MAC-адрес.
- В IPv6 маска задаётся указанием длины префикса.
  - Пример: **`2001:db8:85a3::370:7334/64`**

Зарезервированные IPv6-адреса

- **:: (все нули)**
  - адрес несуществующего узла, имеет особое значение.
- **::1**
  - адрес для обращения узла к самому себе (англ. loopback);
  - передача сообщений по такому адресу не приводит к задействованию канального уровня.
- В IPv6 отсутствует широковещательный адрес, но имеется многоадресное вещание.

Интернет-протокол IPv6. Логический адрес

- Глобальный индивидуальный адрес
  - **2a00:1450:400f:801::200e** – сервер сайта google.com
  - **2a02:6b8:a::a** – сервер сайта yandex.by
- Адреса для одной локальной сети
  - диапазон **fe80::/10** – пример локального адреса
  - не маршрутизируются
- Адреса групповой рассылки
  - диапазон от **FF00::/8** до **FFFF::/8**
  - **FF02::1** – группа многоадресной рассылки по всем узлам (аналог широковещательного адреса в IPv4)
  - **FF02::2** – группа многоадресной рассылки по всем маршрутизаторам

Сравнение заголовков IPv4 и IPv6

IPv4

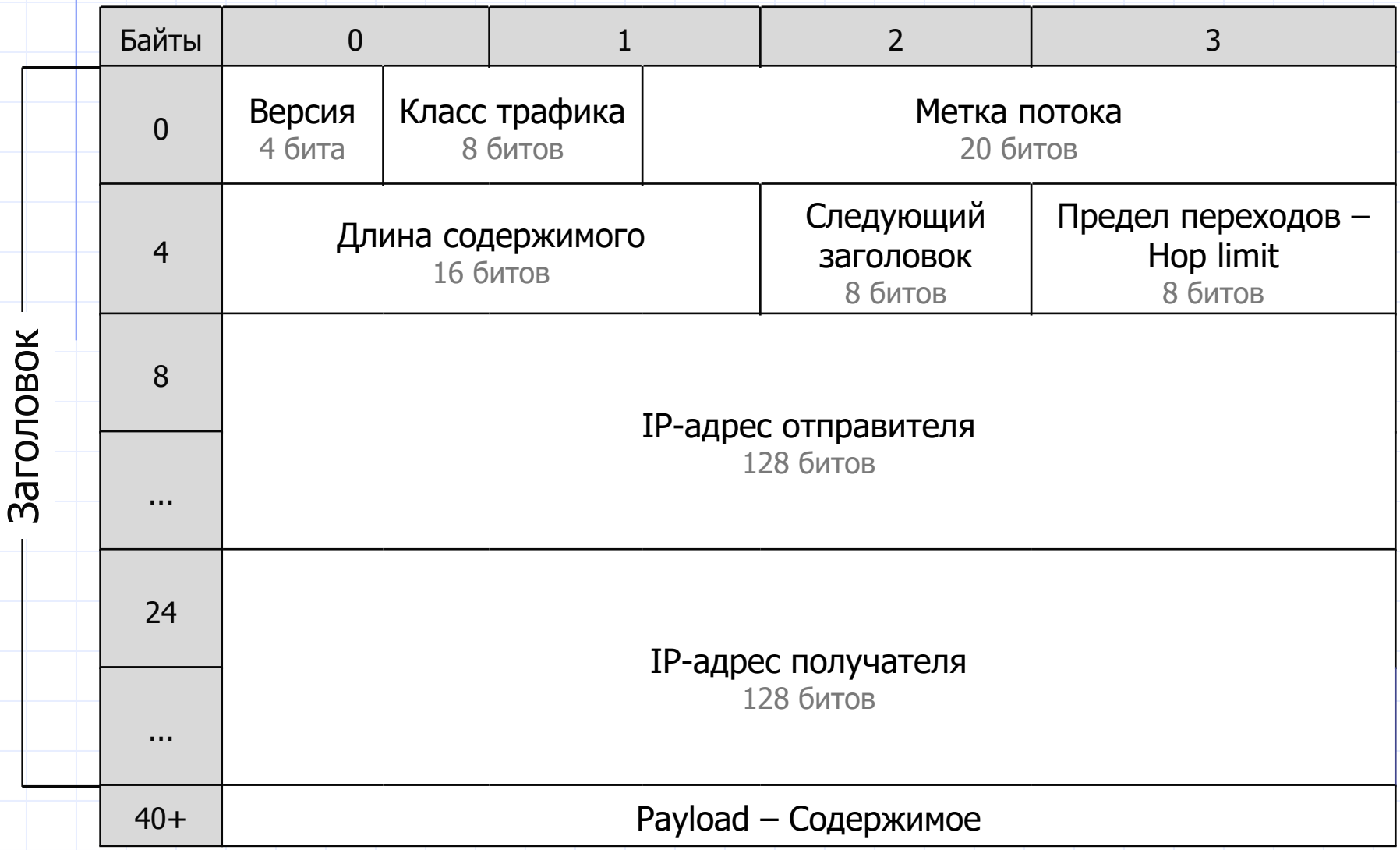
Байты	0		1	2	3
0	Версия 4 бита	Длина заголовка 4 бита	Тип обслуживания 8 битов	Общая длина в байтах 16 битов	
4	ID – Идентификатор 16 битов			Флаги 3 бита	Смещение фрагмента 13 битов
8	TTL – Время жизни 8 битов		Протокол 8 битов	Контрольная сумма заголовка 16 битов	
12	IP-адрес отправителя 32 бита				
16	IP-адрес получателя 32 бита				
(20)	Параметры (необязательное поле				
20+	Payload – Содержимое				

IPv6

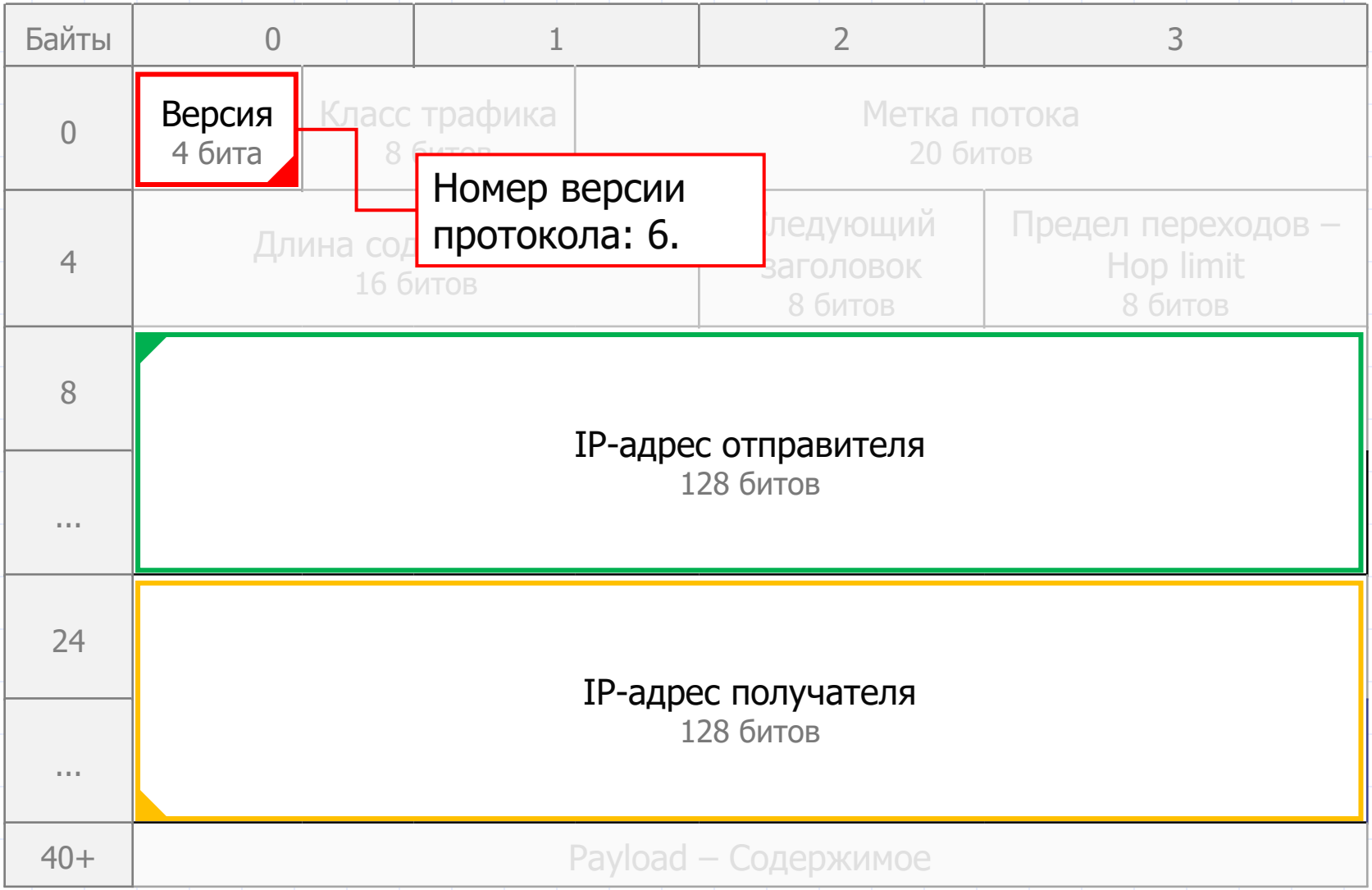
Байты	0	1	2	3
0	Версия 4 бита	Класс трафика 8 битов	Метка потока 20 битов	
4	Длина содержимого 16 битов		Следующий заголовок 8 битов	Предел переходов – Hop limit 8 битов
8	IP-адрес отправителя 128 битов			
…				
24	IP-адрес получателя 128 битов			
…				
40+	Payload – Содержимое			

- не сохранены в IPv6
- сохранены в IPv6
- изменены в IPv6
- добавлены в IPv6

Структура IPv6-пакета



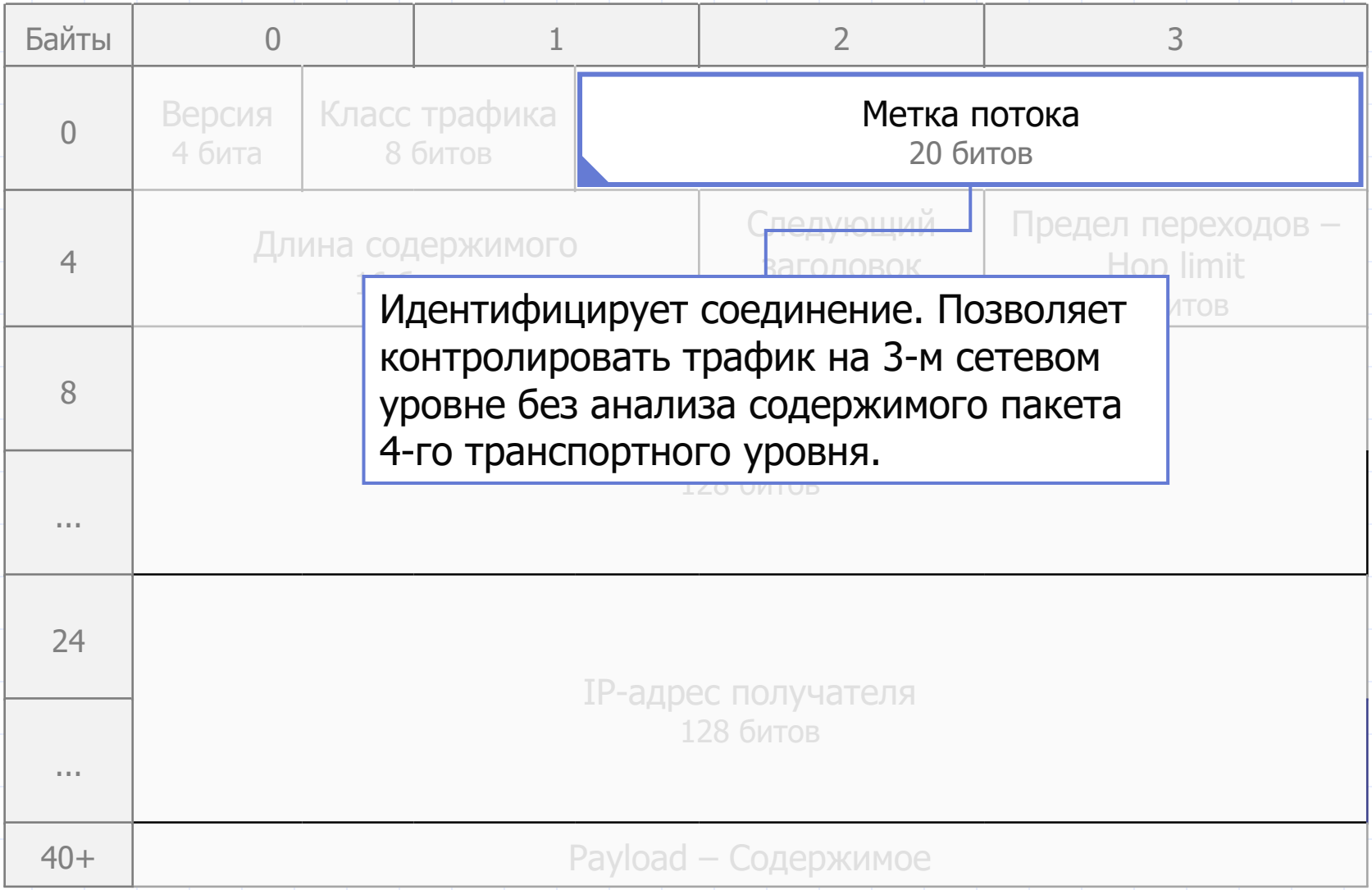
Структура IPv6-пакета (продолжение)



Структура IPv6-пакета (продолжение)

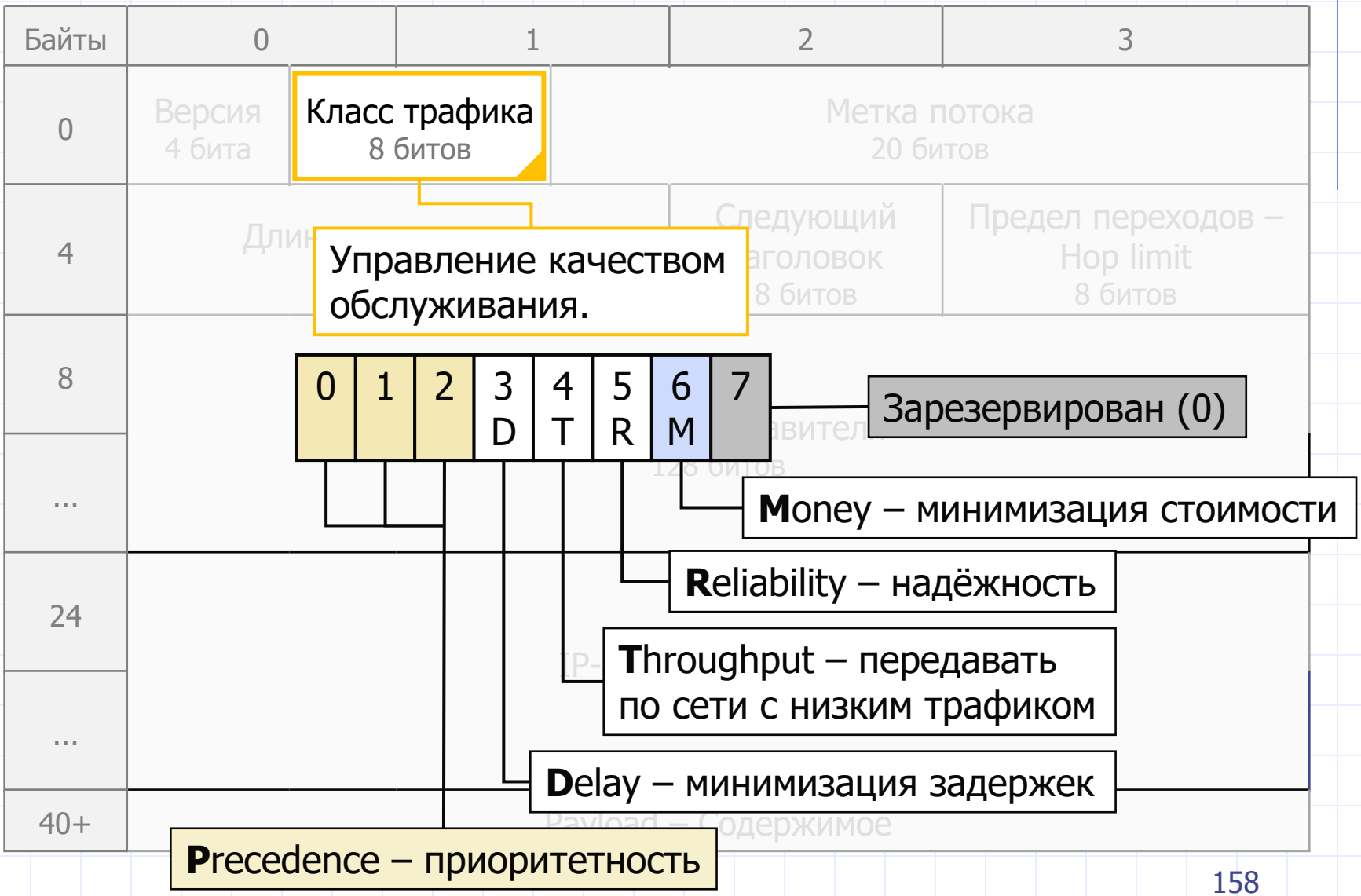
- Промежуточные узлы могут реализовывать возможность управления трафиком, например, блокировки определённых соединений или установкой приоритетов.
- Классическим вариантом идентификации соединения является кортеж из **пяти** элементов: *адрес отправителя, адрес получателя, порт отправителя, порт получателя, тип протокола транспортного уровня*. Такой подход требует наличия средств анализа содержимого IP-пакетов и разбора заголовков 4-го транспортного уровня. Это приводит к замедлению обработки пакетов на промежуточных узлах и снижению пропускной способности сети в целом.
- Метка потока $^{20\ \text{битов}}$ позволяет идентифицировать соединение без необходимости анализа содержимого пакета 4-го транспортного уровня, что упрощает обработку трафика на промежуточных узлах. Таким образом для идентификации соединения используется кортеж из **трёх** элементов: *адрес отправителя, адрес получателя, метка потока*.

Структура IPv6-пакета (продолжение)

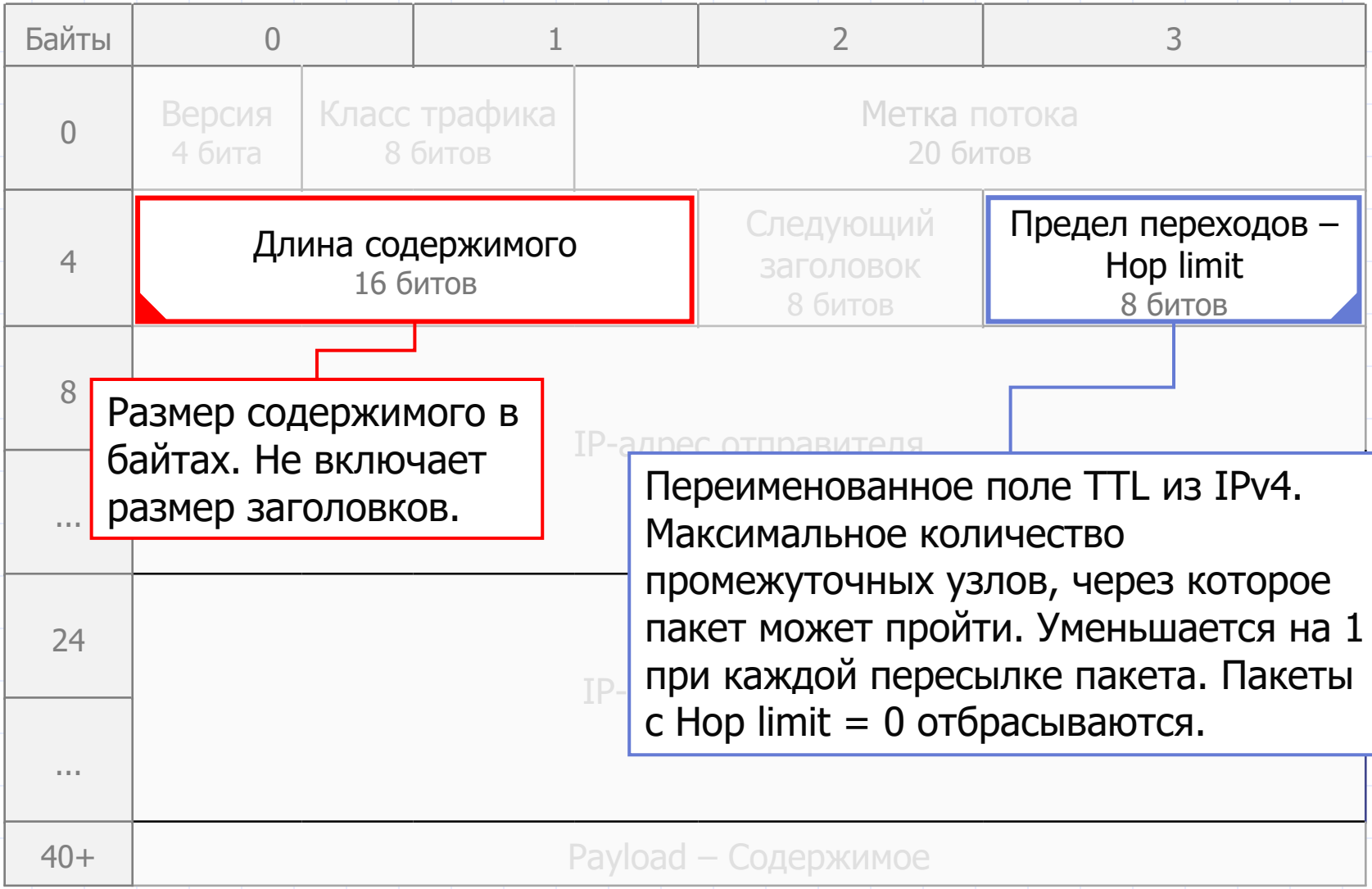


Идентифицирует соединение. Позволяет контролировать трафик на 3-м сетевом уровне без анализа содержимого пакета 4-го транспортного уровня.

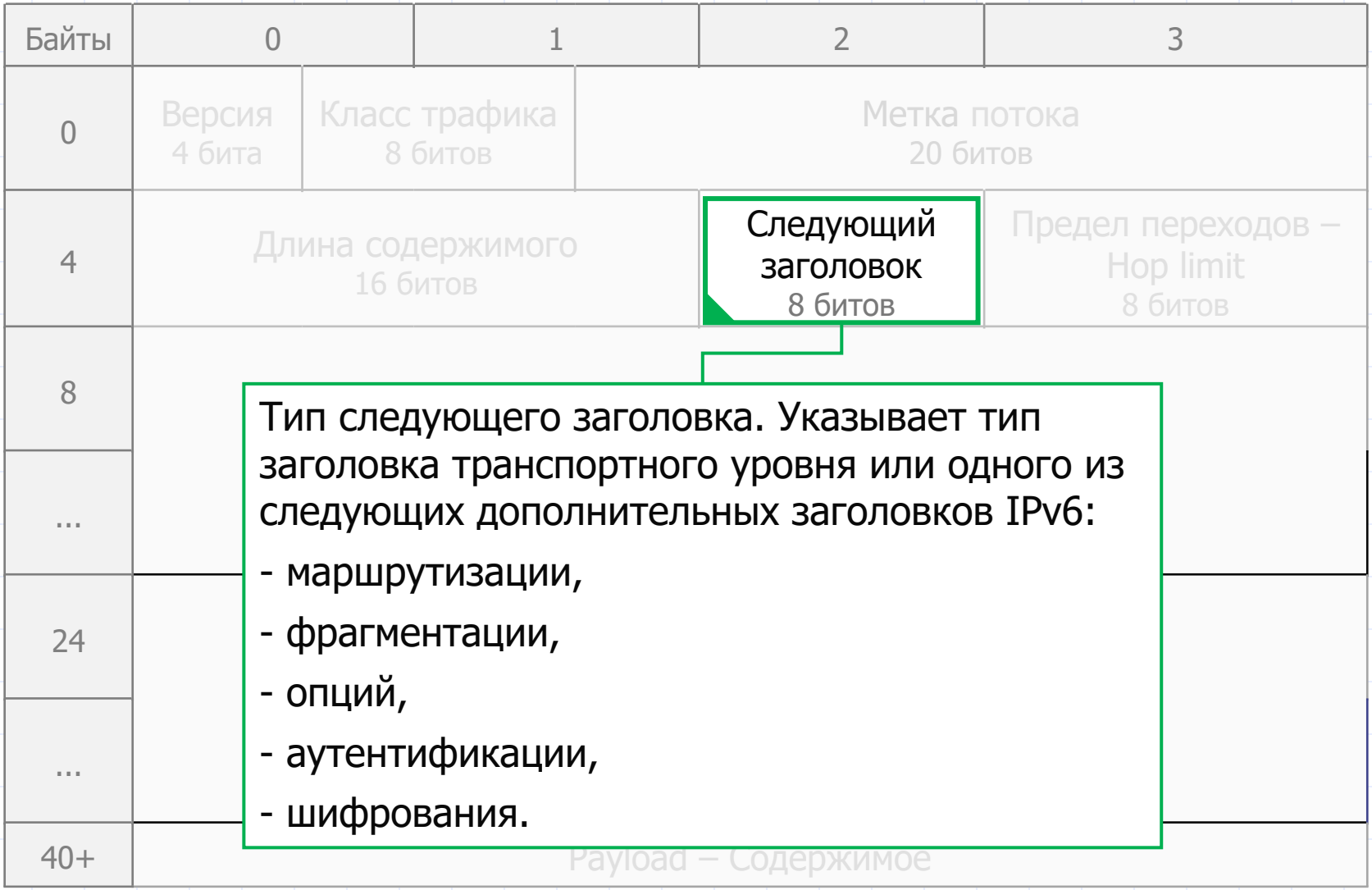
Структура IPv6-пакета (продолжение)



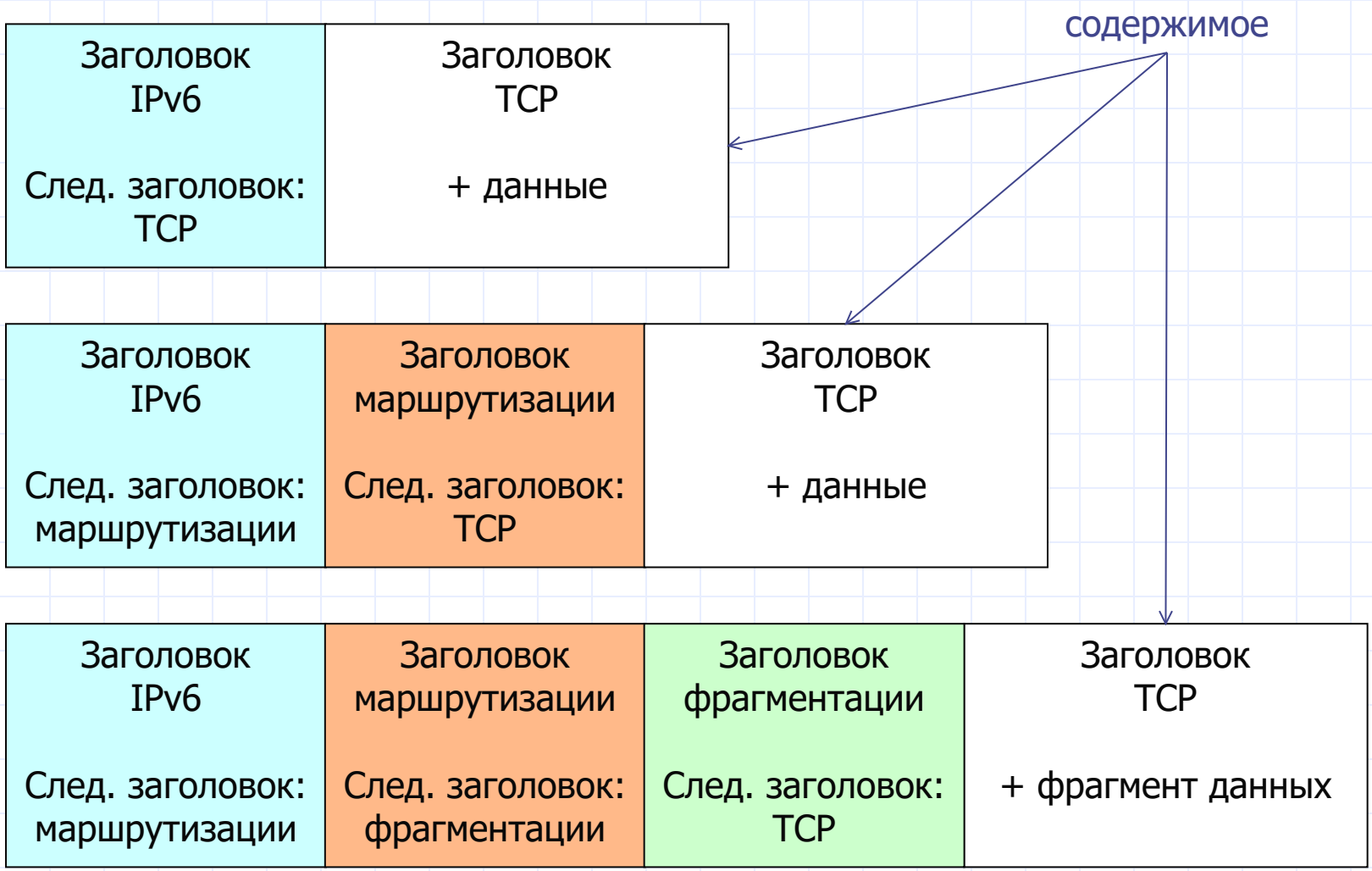
Структура IPv6-пакета (продолжение)



Структура IPv6-пакета (продолжение)



Цепочка заголовков в IPv6

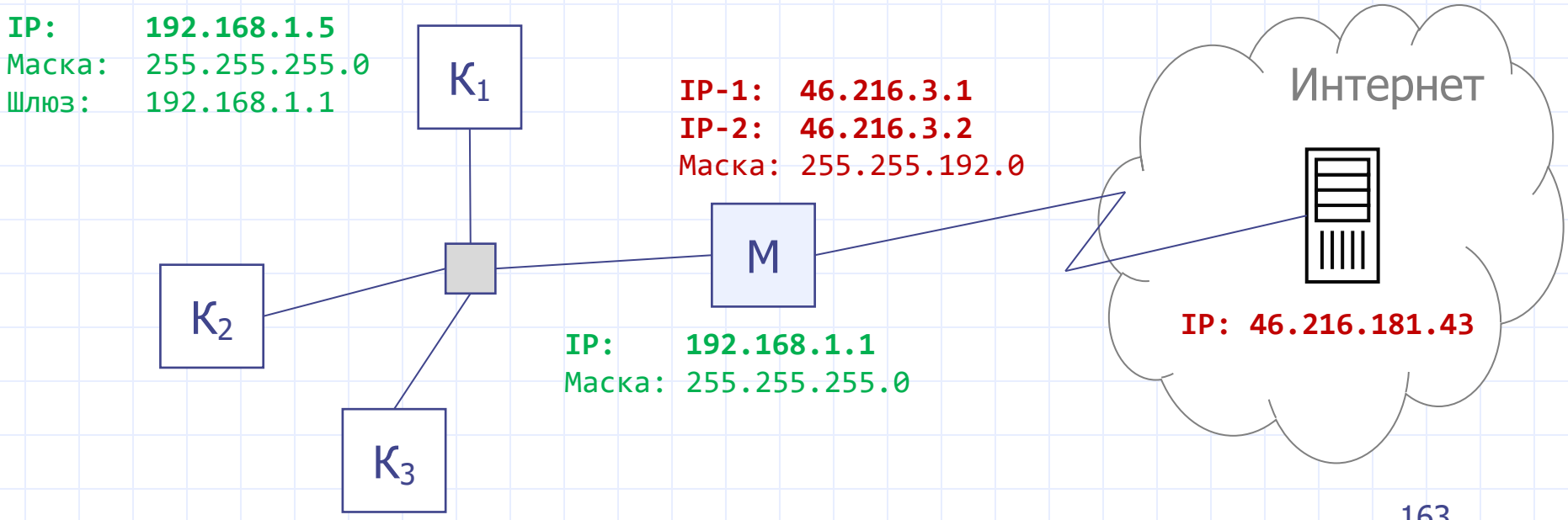


Дополнительные заголовки в IPv6

- Заголовок фрагментации в IPv6 служит той же цели, что и Смещение фрагмента и Флаги фрагментации в IPv4.
  - Разделение IP-дейтаграмм на фрагменты и их сборку обеспечивают оконечные узлы, которые должны учитывать параметр MTU (англ. Maximum Transmission Unit) канального уровня. Маршрутизаторы избавлены от этой функциональности.
- Заголовок маршрутизации позволяет накопить информацию о маршруте при передаче пакета в одну сторону, а потом использовать её для быстрой маршрутизации в обратную сторону.
- Заголовки аутентификации и шифрования обеспечивают безопасность.
- Заголовок опций позволяет передавать сверхбольшие дейтаграммы (англ. jumbogram), превышающие максимальный размер 65,535 байтов.

Трансляция сетевых адресов (англ. NAT)

- NAT (англ. Network Address Translation) – сетевая технология, которая позволяет узлам локальной сети с неуникальными локальными IP-адресами выходить в глобальную сеть Интернет.
- Разновидности NAT
  - Статический
  - Динамический (PAT – Port Address Translation)



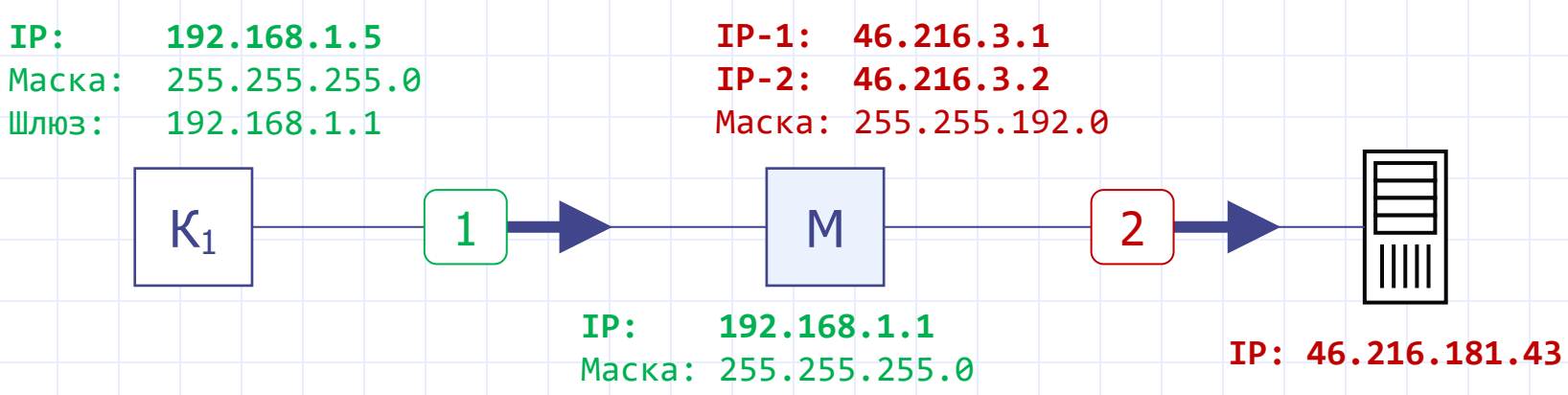
Статический NAT. Передача пакета в глобальную сеть

1

Протокол	Адрес источника	Адрес назначение	Порт источника	Порт назначения
TCP	192.168.1.5	46.216.181.43	3001	80

2

Протокол	Адрес источника	Адрес назначение	Порт источника	Порт назначения
TCP	46.216.3.2	46.216.181.43	3001	80



Локальный адрес	Глобальный адрес
192.168.1.5	46.216.3.2

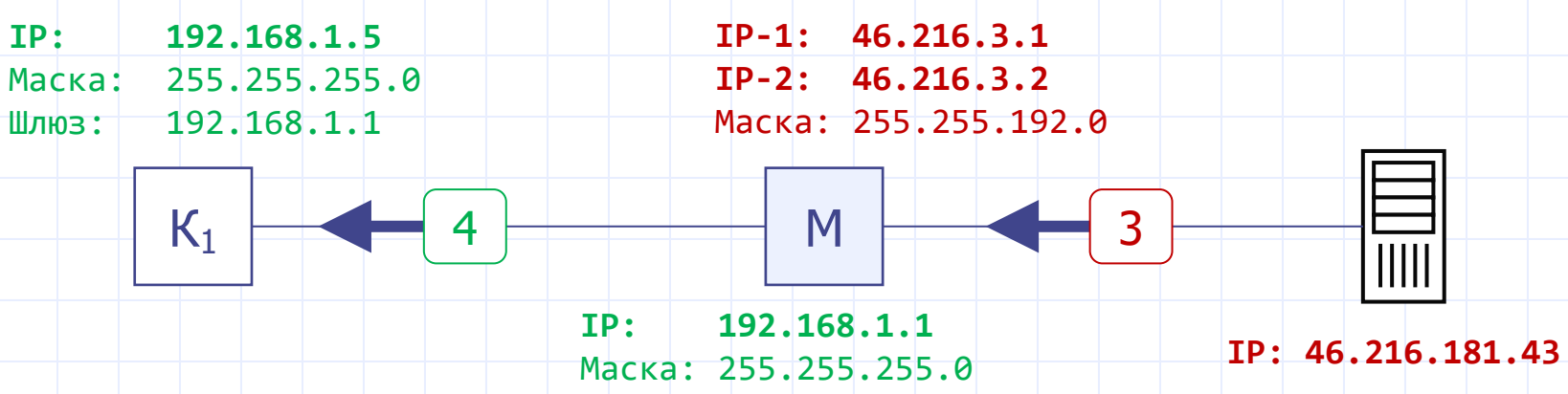
Статический NAT.
Приём пакета из глобальной сети

3

Протокол	Адрес источника	Адрес назначение	Порт источника	Порт назначения
TCP	46.216.181.43	46.216.3.2	80	3001

4

Протокол	Адрес источника	Адрес назначение	Порт источника	Порт назначения
TCP	46.216.181.43	192.168.1.5	80	3001



Локальный адрес	Глобальный адрес
192.168.1.5	46.216.3.2

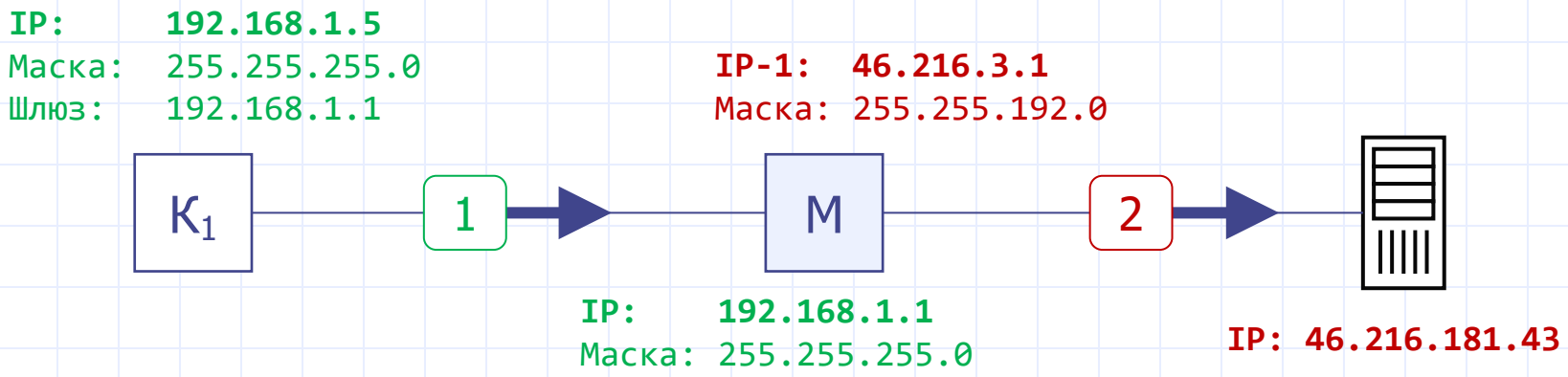
Динамический NAT (PAT). Передача пакета в глобальную сеть

1

Протокол, флаги	Адрес источника	Адрес назначение	Порт источника	Порт назначения
TCP, SYN	192.168.1.5	46.216.181.43	3001	80

2

Протокол, флаги	Адрес источника	Адрес назначение	Порт источника	Порт назначения
TCP, SYN	46.216.3.1	46.216.181.43	40001	80



Локальный порт	Глобальный порт	Протокол	Локальный адрес	Глобальный адрес
3001	40001	TCP	192.168.1.5	46.216.3.1

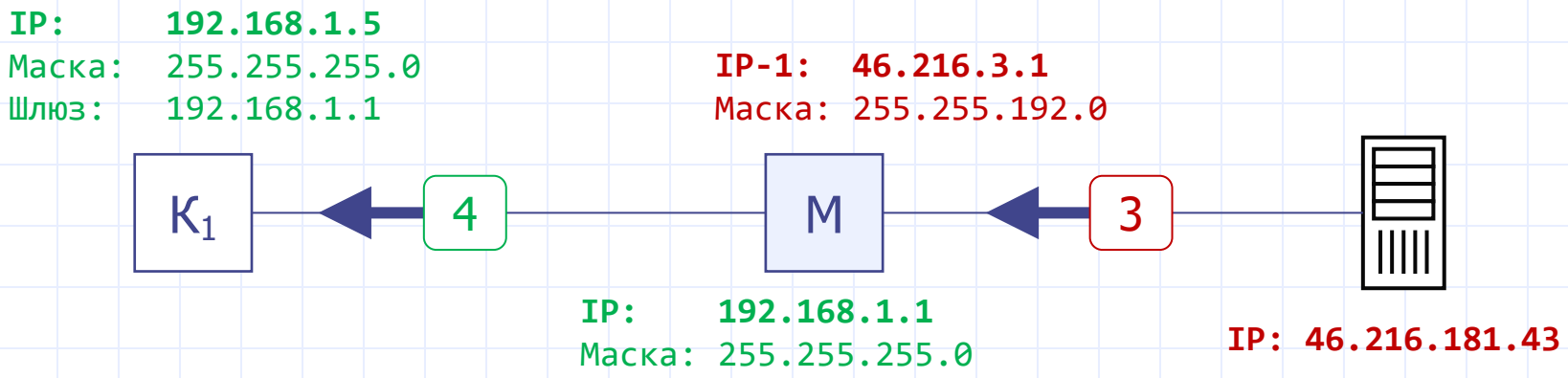
Динамический NAT (PAT). Приём пакета из глобальной сети

3

Протокол, флаги	Адрес источника	Адрес назначение	Порт источника	Порт назначения
TCP, ACK	46.216.181.43	46.216.3.1	80	40001

4

Протокол, флаги	Адрес источника	Адрес назначение	Порт источника	Порт назначения
TCP, ACK	46.216.181.43	192.168.1.5	80	3001



Локальный порт	Глобальный порт	Протокол	Локальный адрес	Глобальный адрес
3001	40001	TCP	192.168.1.5	46.216.3.1

Отличие обработки протоколов TCP и UDP в технологии динамического NAT (PAT)

- Для протокола TCP новая запись в таблице NAT создаётся при передаче TCP-пакета с флагом SYN (установка соединения) из локальной сети в глобальную.
  - Пакеты на установку соединения, переданные из глобальной сети на адреса и порты NAT, отбрасываются.
- Для протокола UDP (в котором отсутствует соединение) новая запись в таблице NAT создаётся при передаче первого UDP-пакета из локальной сети в глобальную.
  - Первый исходящий UDP-пакет «открывает» NAT для приёма ответных UDP-пакетов на выделенный глобальный порт от любого IP-адреса глобальной сети.

Статический и динамический NAT

Статический NAT

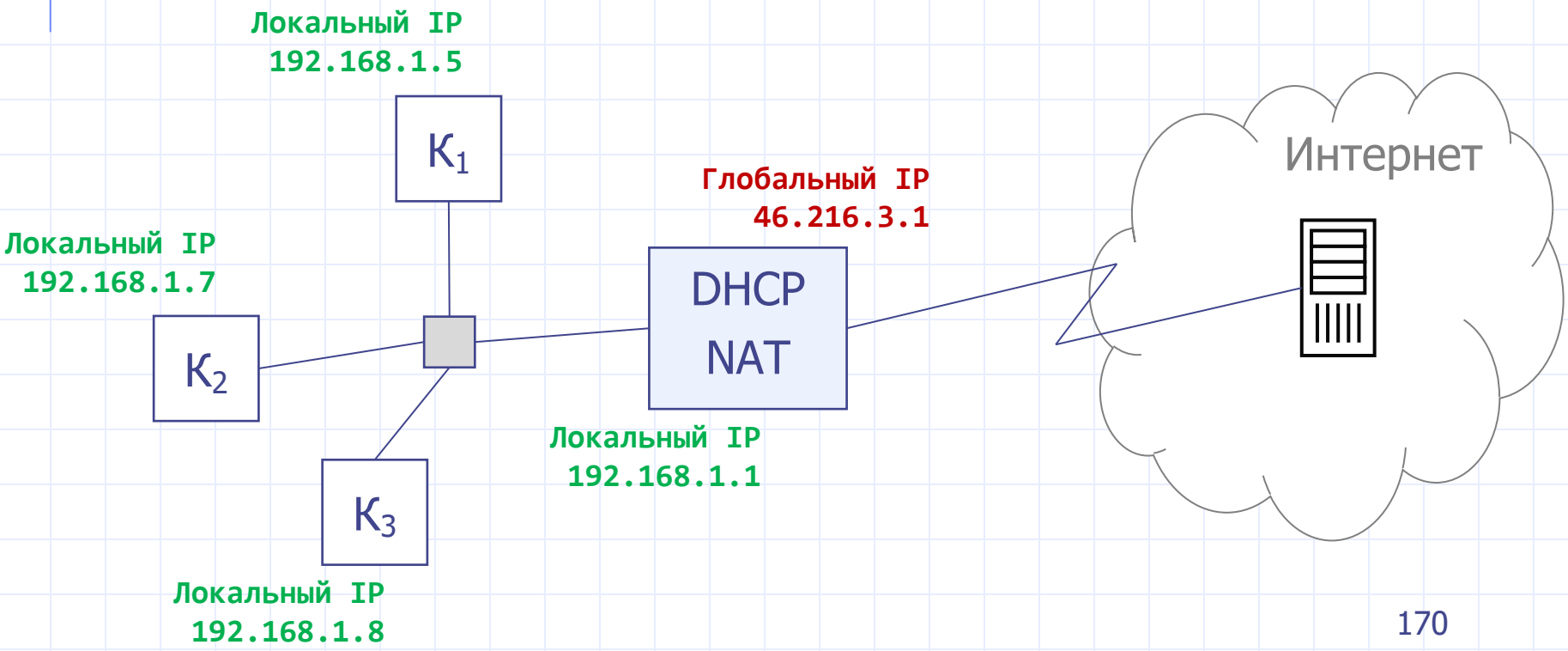
- Выделяет в пользование локальному узлу один из глобальных адресов.
- Подменяет локальный адрес глобальным адресом.
- Количество локальных узлов с одновременным доступом в глобальную сеть невелико.
- Обеспечивает установку соединения из глобальной сети с узлом в локальной сети.

Динамический NAT (PAT)

- Выделяет в пользование локальному узлу один из своих портов.
- Подменяет локальный адрес и порт глобальным адресом и портом из заданного диапазона.
- Количество локальных узлов с одновременным доступом в глобальную сеть велико.
- Запрещает установку соединение из глобальной сети с узлом в локальной сети.

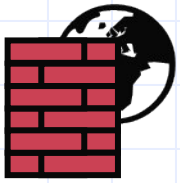
Автоматическое подключение узлов к сети Интернет (NAT + DHCP)

- DHCP – обеспечивает автоматическую выдачу локальных IP-адресов.
- NAT – обеспечивает трансляцию локальных IP-адресов в глобальный IP-адрес.



Сетевой экран

- Сетевой экран (иногда называемый «брандмауэр», англ. firewall) – программа или аппаратно-программное устройство для фильтрации сетевого трафика с целью предотвращения угроз, исходящих из сети.
- В настоящее время является обязательным компонентом операционной системы.
- Работает на основе настраиваемых правил на разных уровнях OSI:
  - канальном – фильтрует кадры (Ethernet, Wi-Fi и др.);
  - сетевом – фильтрует IP-дейтаграммы;
  - транспортном – фильтрует UDP- и TCP-пакеты;
  - прикладном – распознаёт протоколы прикладного уровня (HTTP, FTP и др.) и фильтрует их данные; использует эвристические (не являющиеся гарантированно точными) подходы для фильтрации трафика.

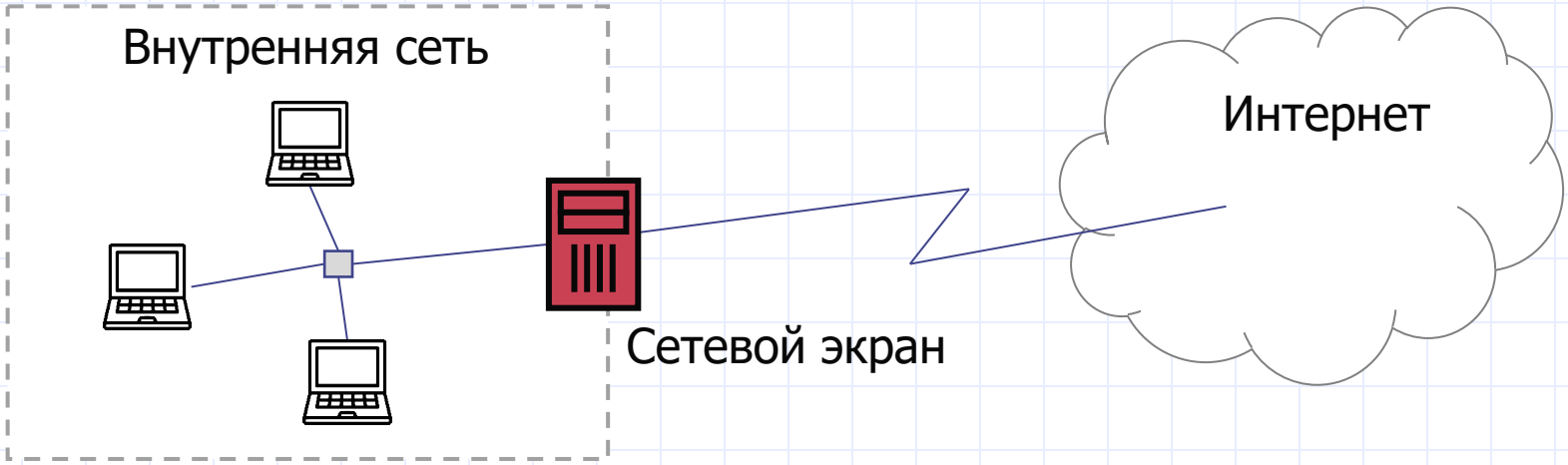


Виды сетевых экранов по способу фильтрации

- Сетевой экран без запоминания состояния.
  - Использует статические правила, учитывающие только текущие параметры заголовков пакетов.
- Сетевой экран с запоминанием состояния.
  - Принимает решение динамически на основе текущего состояния сеанса и его предыстории.

Назначения сетевых экранов

- Сетевой экран для защиты периметра



- Сетевой экран для защиты оконечного узла

Правила работы сетевых экранов

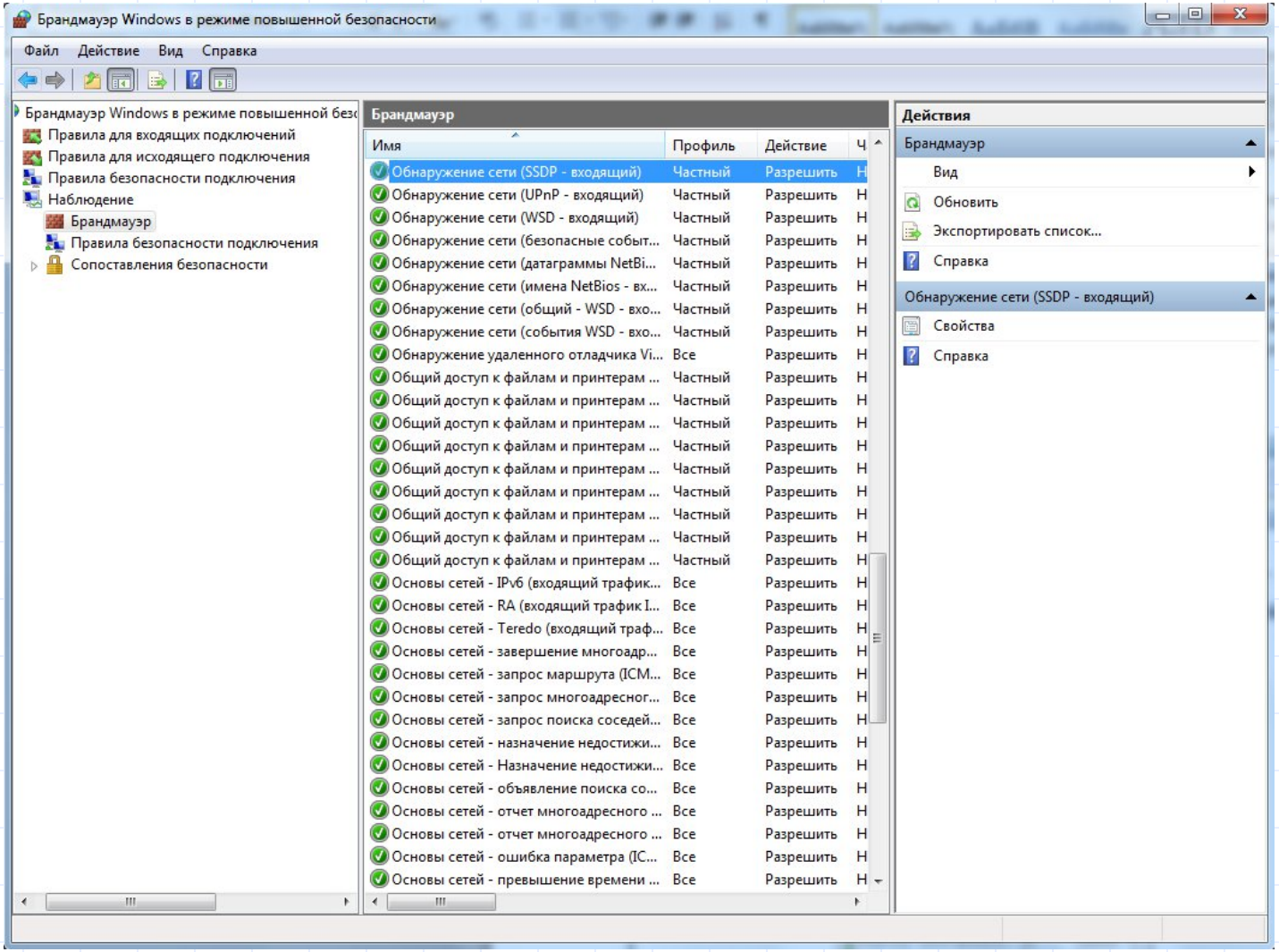
- Правила фильтрации бывают:
  - Разрешающие
  - Запрещающие
- Правила применяются в определённом порядке, который по сути задаёт приоритет правил.
- Стандартное поведение позволяет:
  - запретить всё, что не разрешено явно;
  - разрешить всё, что не запрещено явно;
  - запросить у пользователя создание нового правила с заданным поведением.

Параметры правил фильтрации

- Направление трафика:
  - Входящий
  - Исходящий
- IP-адреса
- Тип протокола
- Порты
- Программы

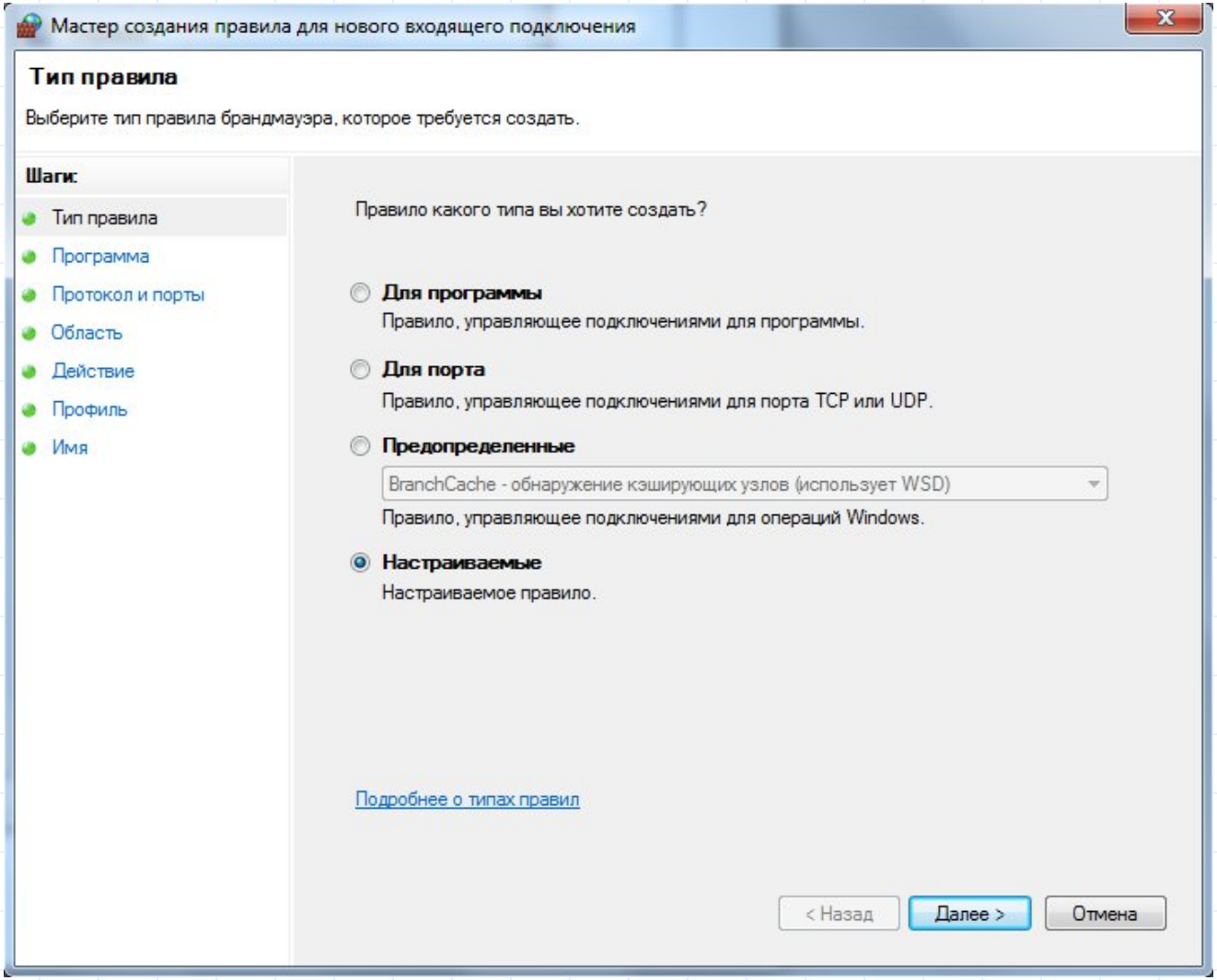
Пример настройки правил сетевого экрана (1)

- Панель Управления → Брандмауэр Windows → Дополнительные параметры



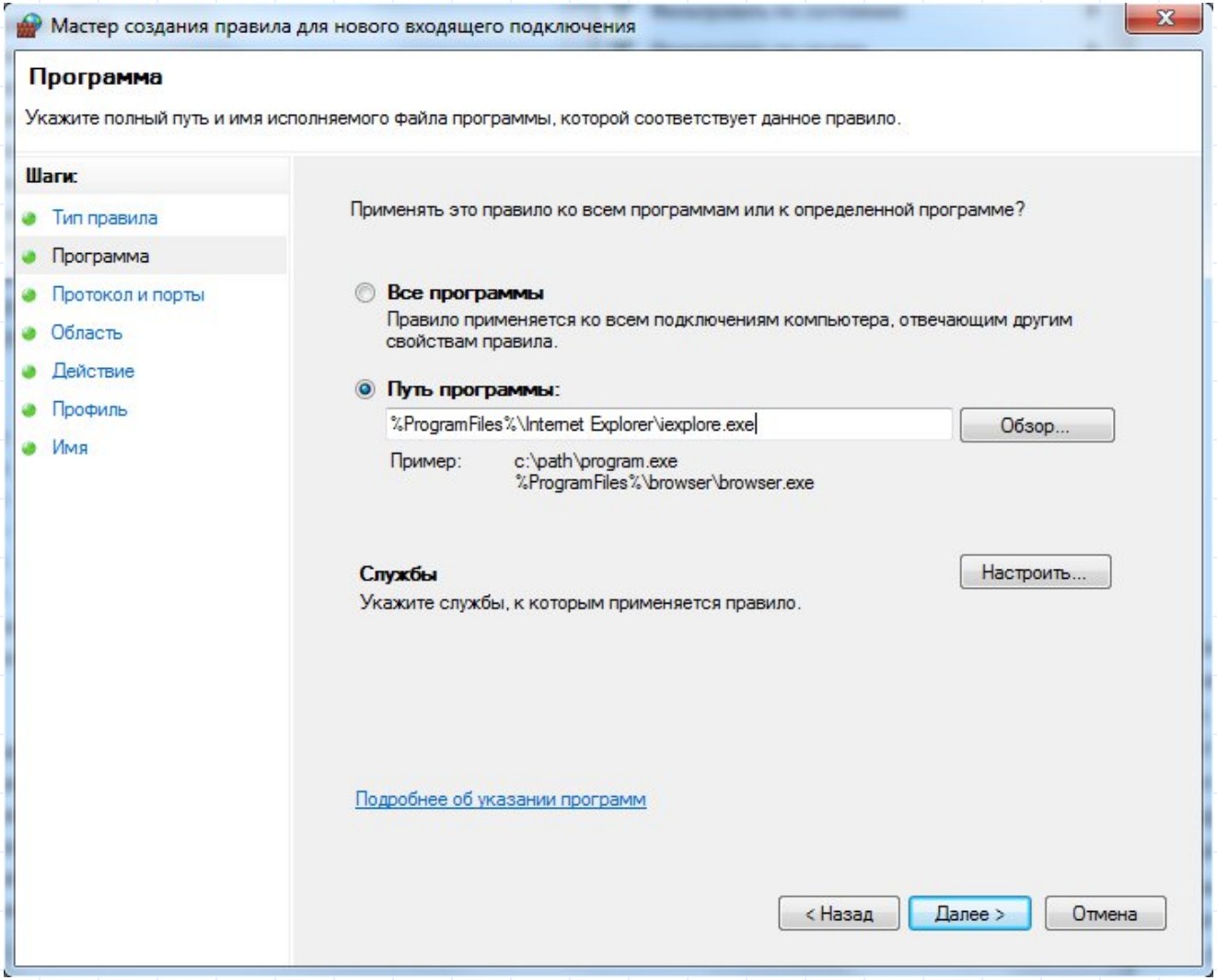
Пример настройки правил сетевого экрана (2)

- Создание нового правила



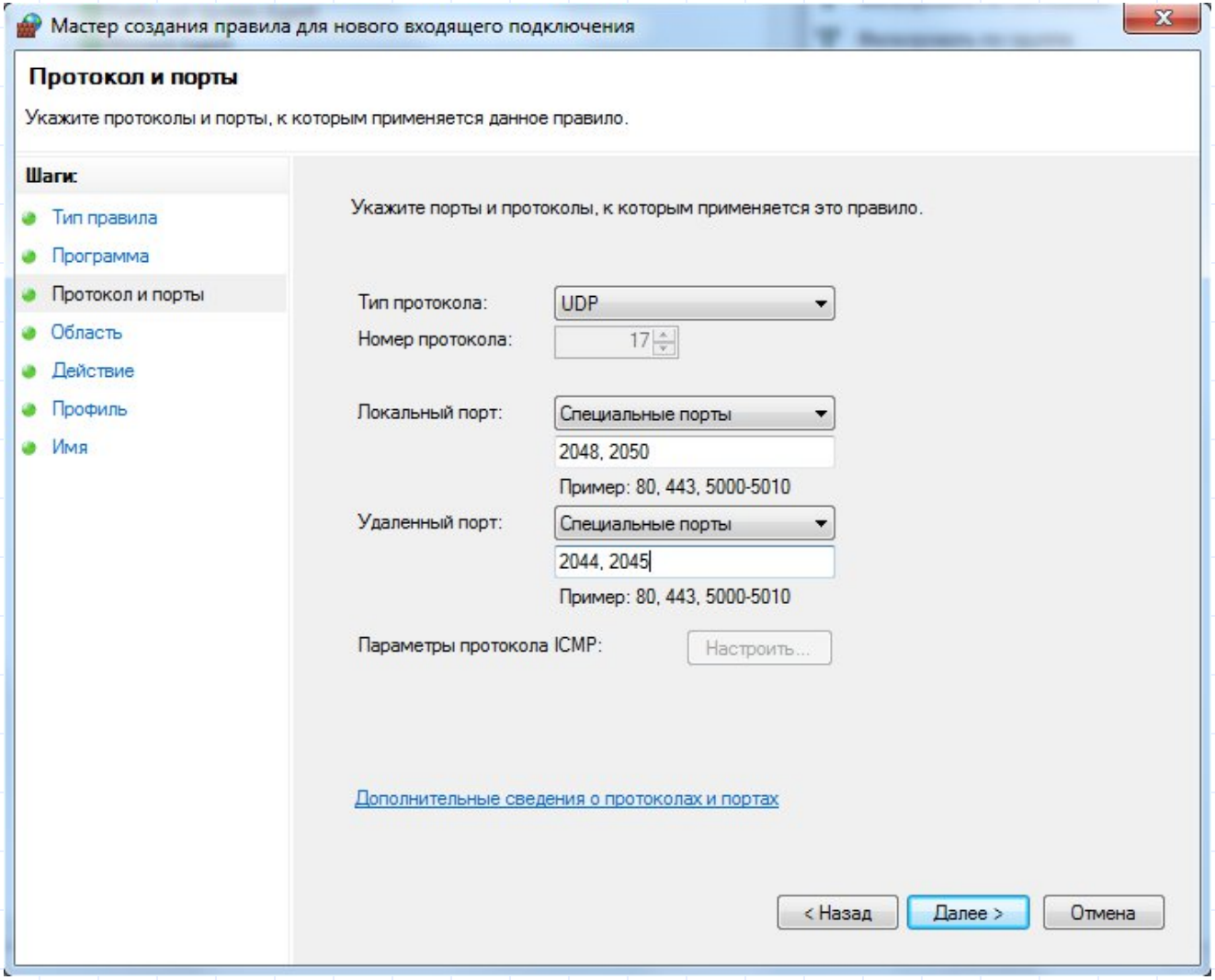
Пример настройки правил сетевого экрана (3)

- Выбор программы, к которой применяется правило



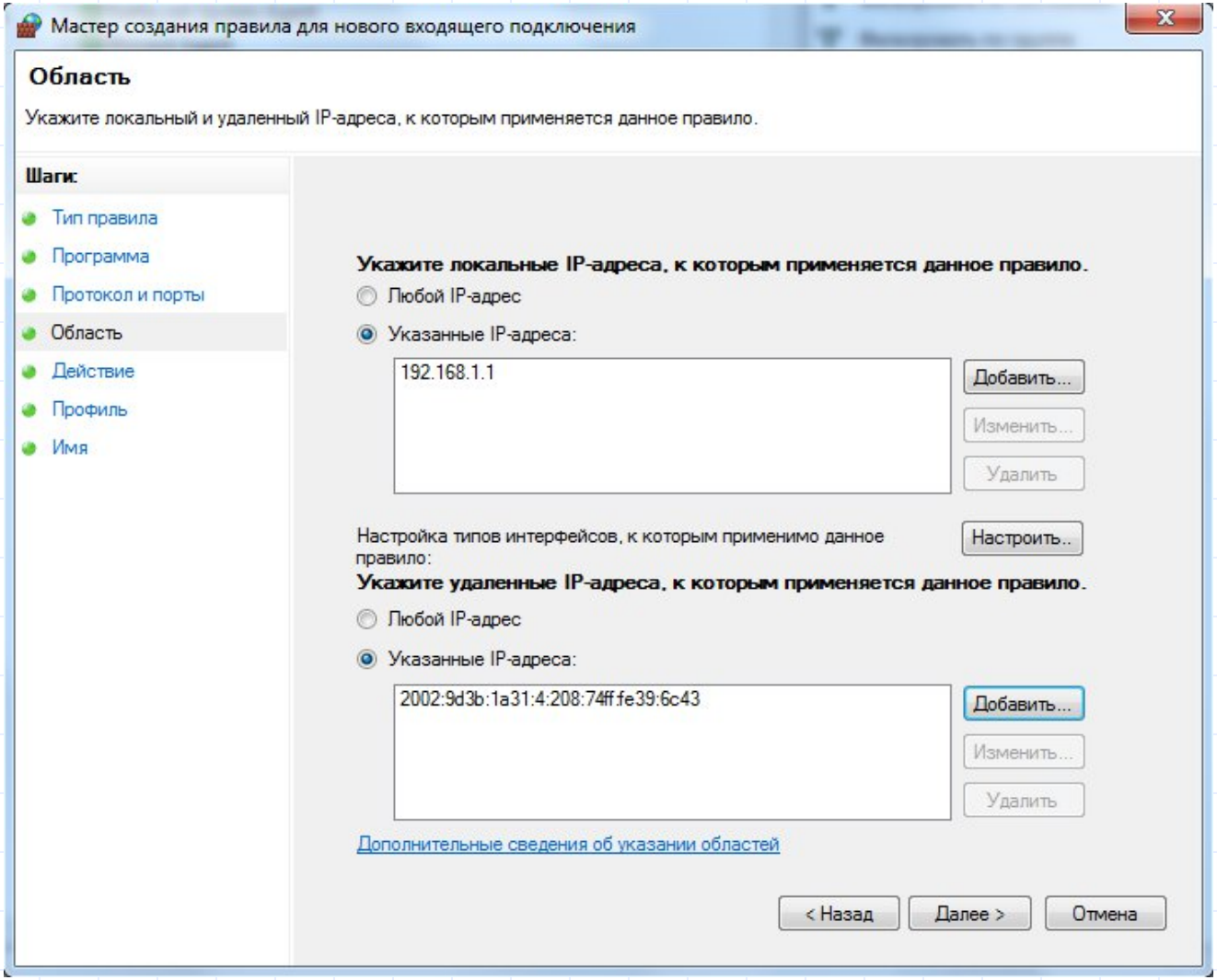
Пример настройки правил сетевого экрана (4)

- Указание типа протокола и портов



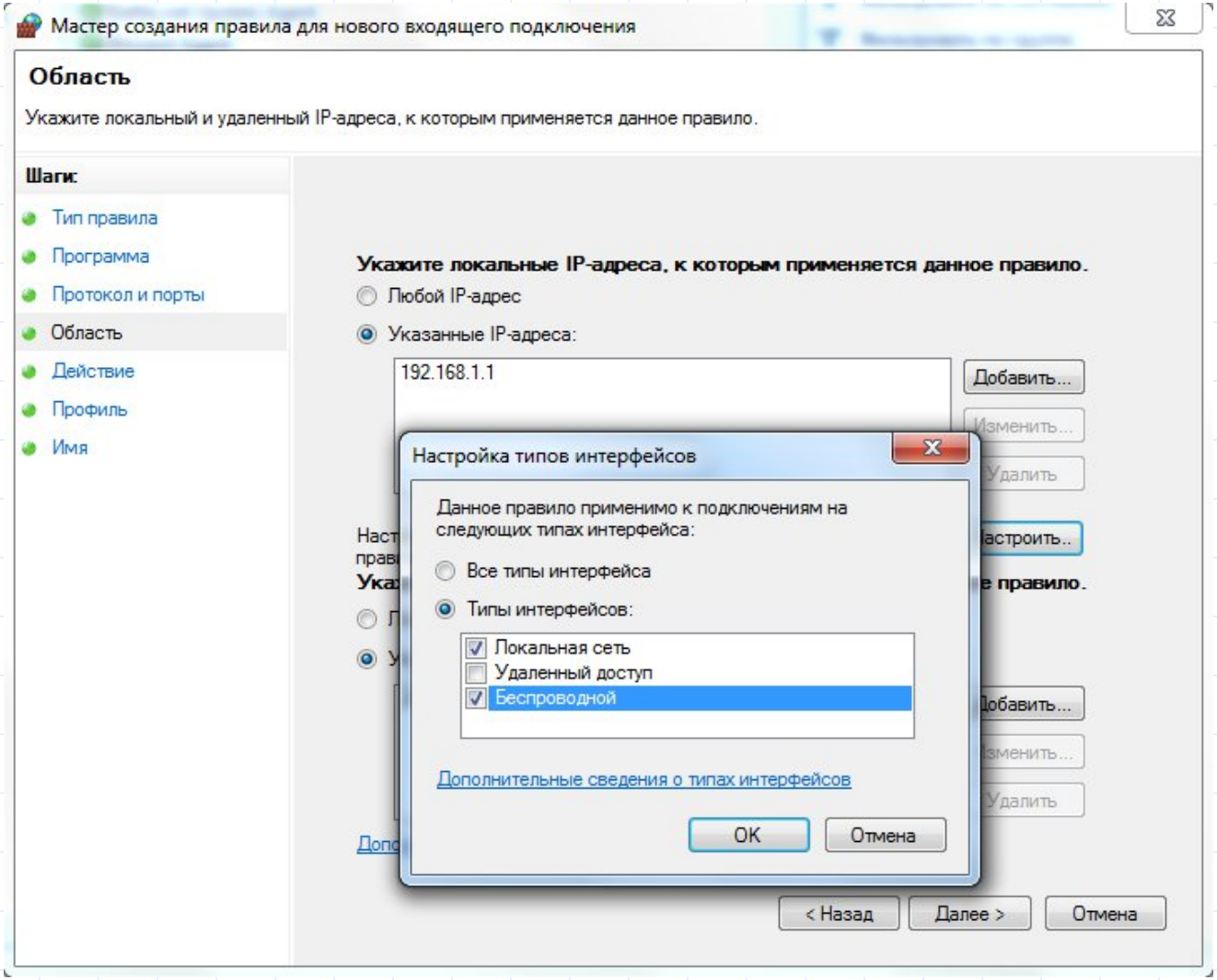
Пример настройки правил сетевого экрана (5)

- Указание IP-адресов



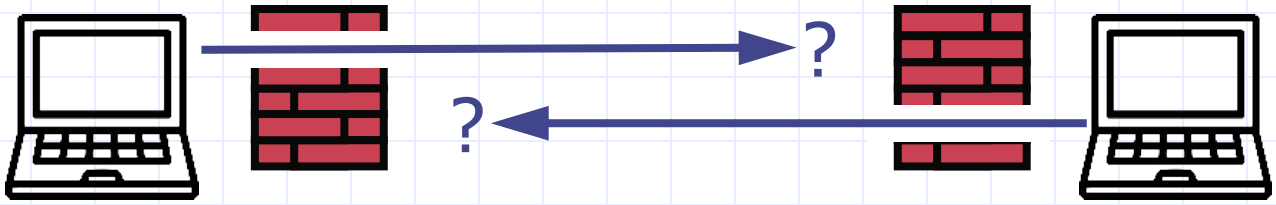
Пример настройки правил сетевого экрана (6)

- Указание сетевого интерфейса (подключения)



Равноправное взаимодействие по протоколу UDP

- Позволяет двум узлам, защищенным сетевыми экранами, установить равноправный прямой обмен данными по протоколу UDP.
- Решает задачу обмена данными даже для узлов, находящихся в разных локальных сетях за устройствами с динамическим NAT (PAT).
- Требует наличия третьей стороны для обмена информацией о точках доступа друг к другу (IP-адреса, порты).
- Англ. UDP hole punching («проделывание дырок» для UDP).

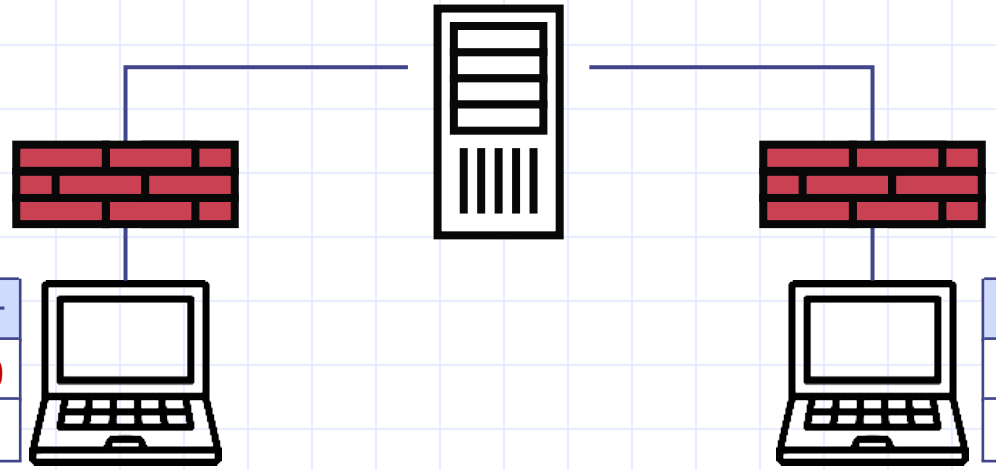


Равноправное взаимодействие по протоколу UDP

- Узлы A и B находятся за сетевыми экранами.
- Узлы A и B знают имена друг друга, но не знают IP-адресов и портов друг друга.
- Узлы A и B знают IP-адрес и порт третьей стороны – узла C.

Узел: C
IP: 46.216.103.3
Порт: 2000

Имя	IP-адрес	Порт
A		
B		



Имя	IP-адрес	Порт
C	46.216.103.3	2000
B		

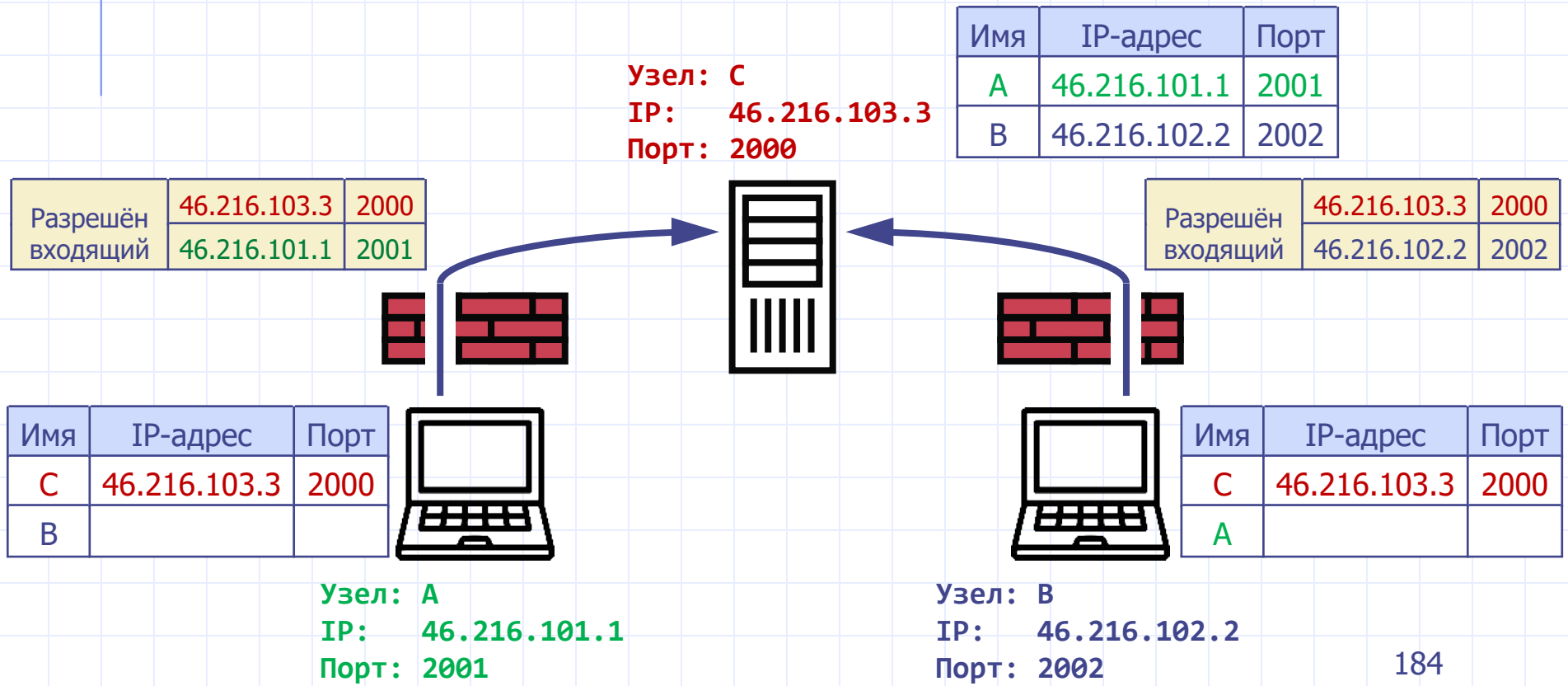
Узел: A
IP: 46.216.101.1
Порт: 2001

Имя	IP-адрес	Порт
C	46.216.103.3	2000
A		

Узел: B
IP: 46.216.102.2
Порт: 2002

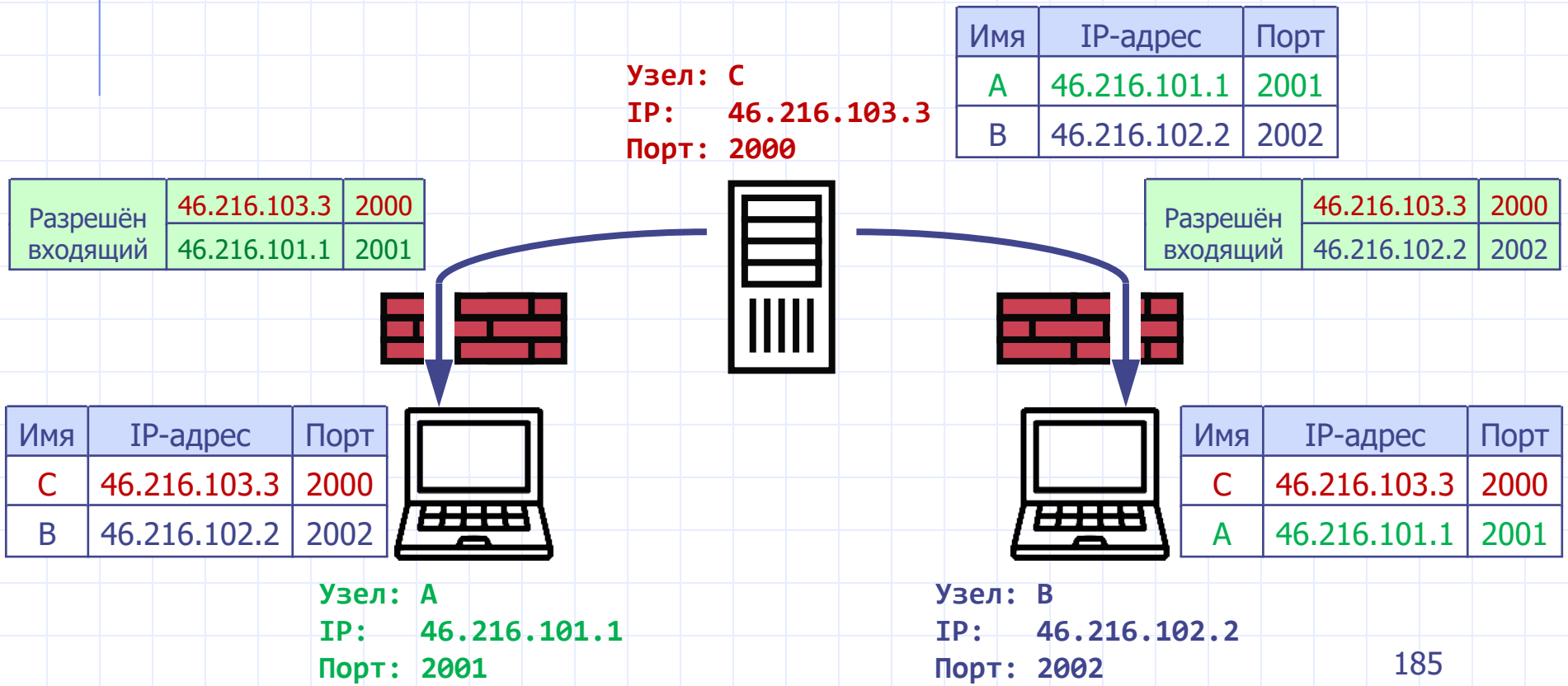
Равноправное взаимодействие по протоколу UDP. Шаг 1

- Узлы A и B соединяются с C по протоколу UDP и аутентифицируются.
  - Сетевые экраны узлов A и B создают правила, разрешающие входящий трафик на IP-адреса и порты, с которых они обратились к C.
- Узел C запоминает IP-адреса и порты узлов A и B.



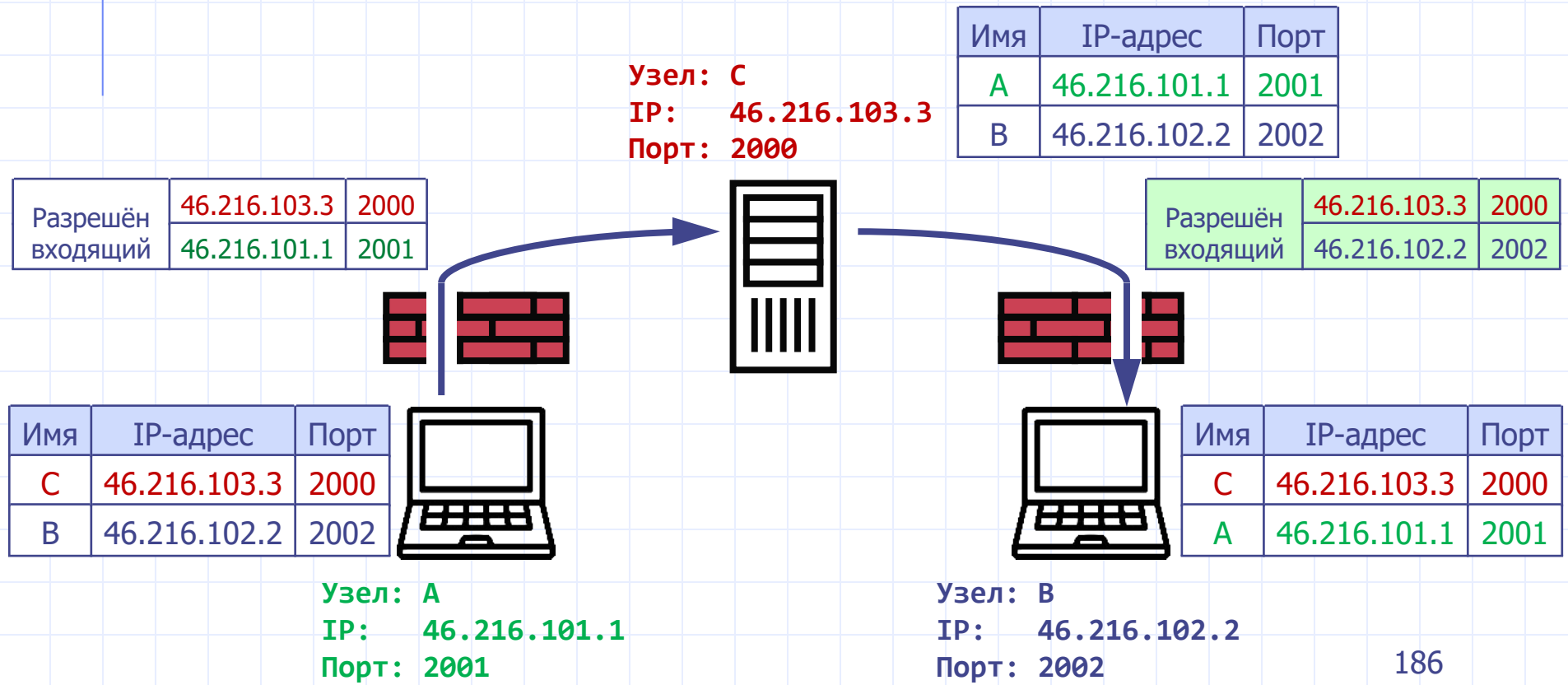
Равноправное взаимодействие по протоколу UDP. Шаг 2

- Узел С сообщает узлам А и В IP-адреса и порты собеседников.
  - Сетевые экраны узлов А и В пропускают ответы от узла С на IP-адреса и порты, с которых они обращались к С ранее.



Равноправное взаимодействие по протоколу UDP. Шаг 3

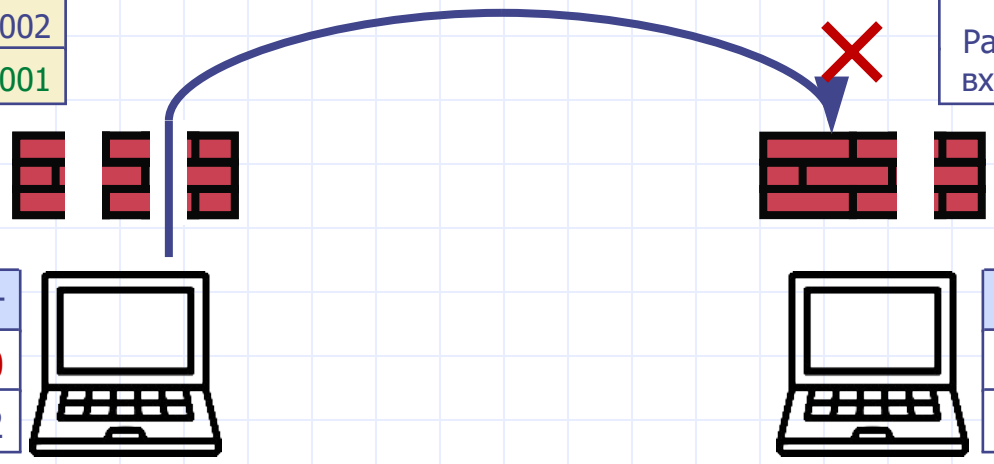
- Узел A сообщает узлу C, что он начинает «звонить» узлу B.
- Узел C сообщает узлу B, что ему «звонит» узел A.



Равноправное взаимодействие по протоколу UDP. Шаг 4

- Узел A «звонит» узлу B.
  - Сетевой экран узла A создаёт правило, разрешающее входящий трафик на IP-адрес и порт, с которых он «звонит» узлу B.
  - Сетевой экран узла B блокирует входящий трафик от узла A.

Разрешён входящий	46.216.103.3	2000
	46.216.101.1	2001
Разрешён входящий	46.216.102.2	2002
	46.216.101.1	2001



Разрешён входящий	46.216.103.3	2000
	46.216.102.2	2002

Имя	IP-адрес	Порт
C	46.216.103.3	2000
B	46.216.102.2	2002

```
Узел: A
IP:   46.216.101.1
Порт: 2001
```

Имя	IP-адрес	Порт
C	46.216.103.3	2000
A	46.216.101.1	2001

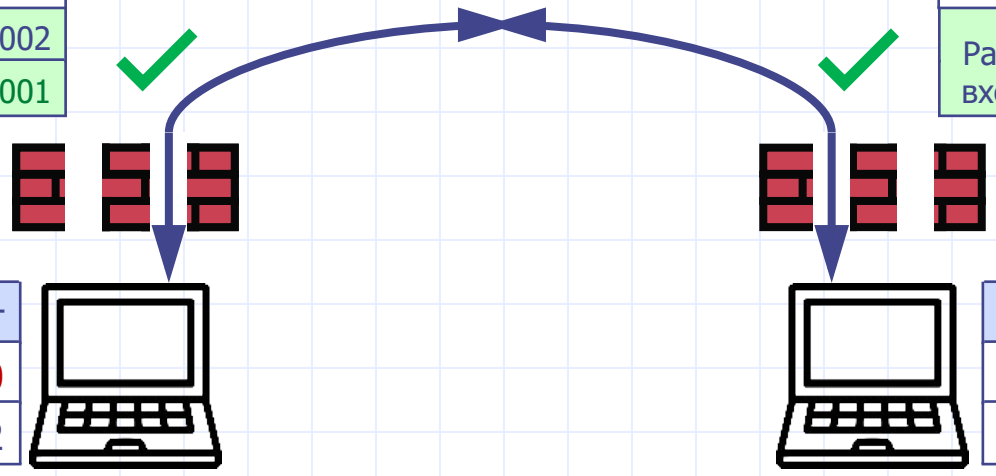
```
Узел: B
IP:   46.216.102.2
Порт: 2002
```

Равноправное взаимодействие по протоколу UDP. Шаг 5

- Узел B «звонит» узлу A навстречу.
  - Сетевой экран узла B создаёт правило, разрешающее входящий трафик на IP-адрес и порт, с которых он «звонит» узлу A.
  - Сетевые экраны узлов A и B пропускают встречные «звонки».

Разрешён входящий	46.216.103.3	2000
	46.216.101.1	2001
Разрешён входящий	46.216.102.2	2002
	46.216.101.1	2001

Разрешён входящий	46.216.103.3	2000
	46.216.102.2	2002
Разрешён входящий	46.216.101.1	2001
	46.216.102.2	2002



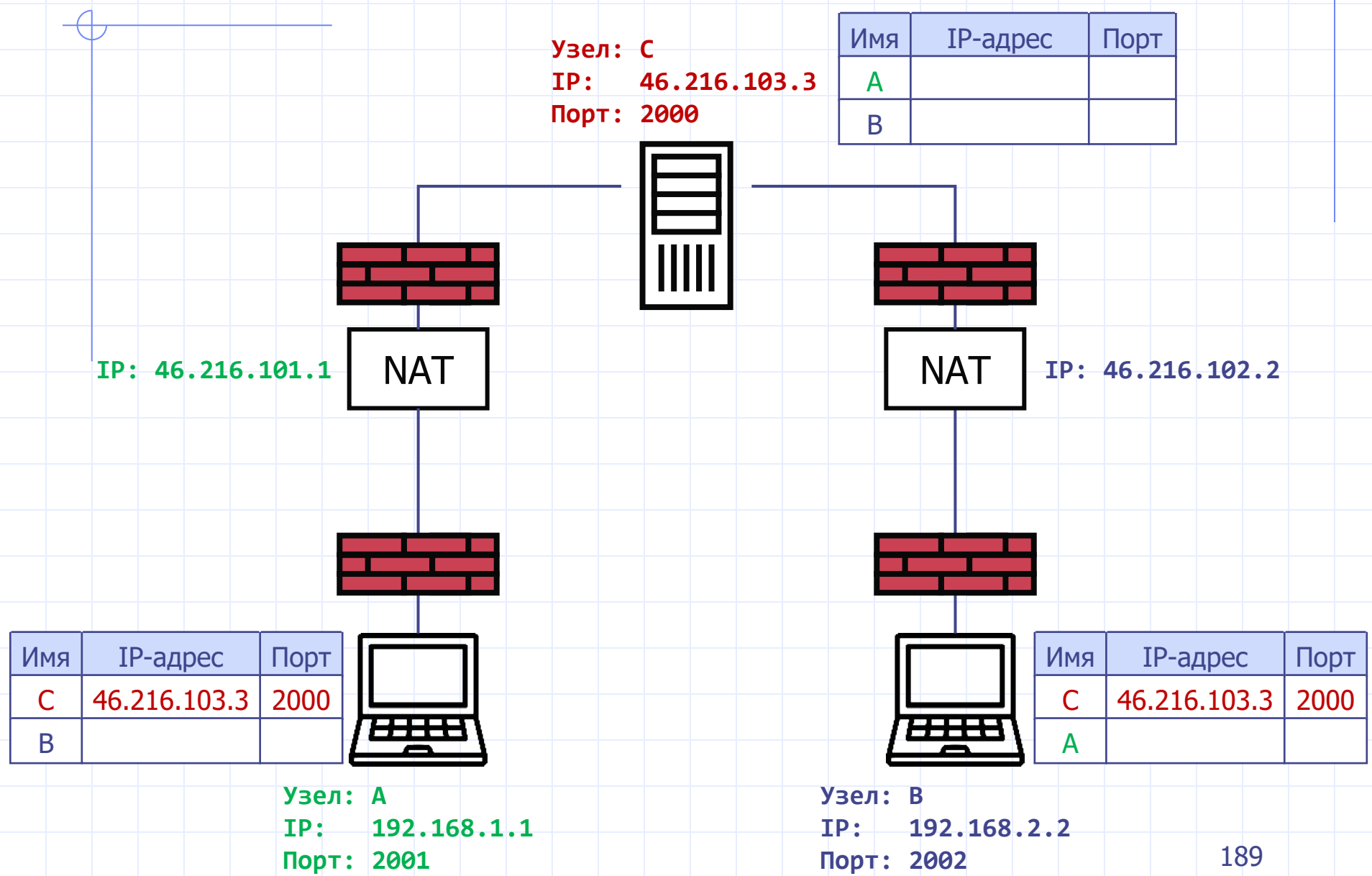
Имя	IP-адрес	Порт
C	46.216.103.3	2000
B	46.216.102.2	2002

Имя	IP-адрес	Порт
C	46.216.103.3	2000
A	46.216.101.1	2001

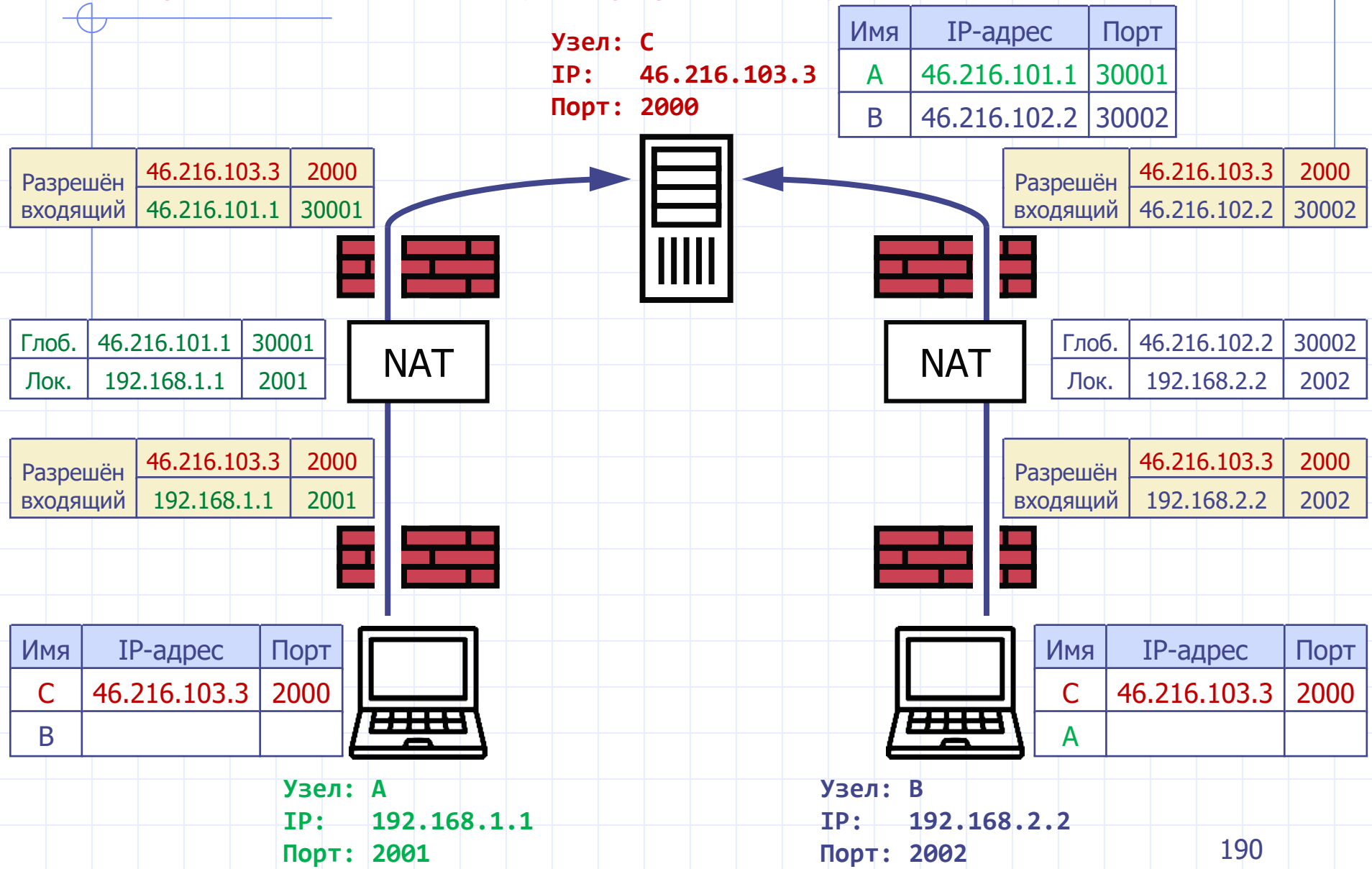
```
Узел: A
IP:   46.216.101.1
Порт: 2001
```

```
Узел: B
IP:   46.216.102.2
Порт: 2002
```

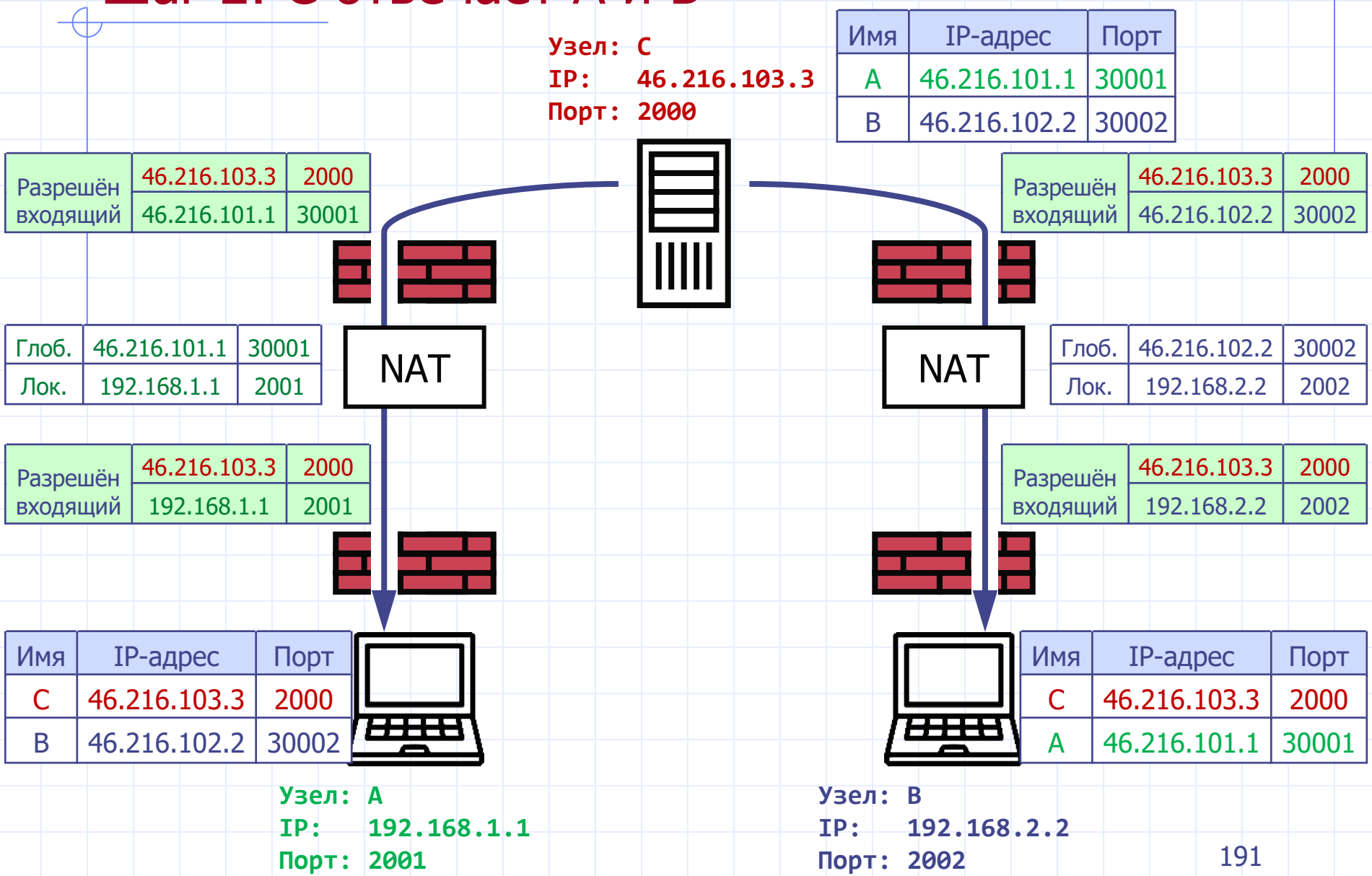
Равноправное взаимодействие по UDP + NAT.



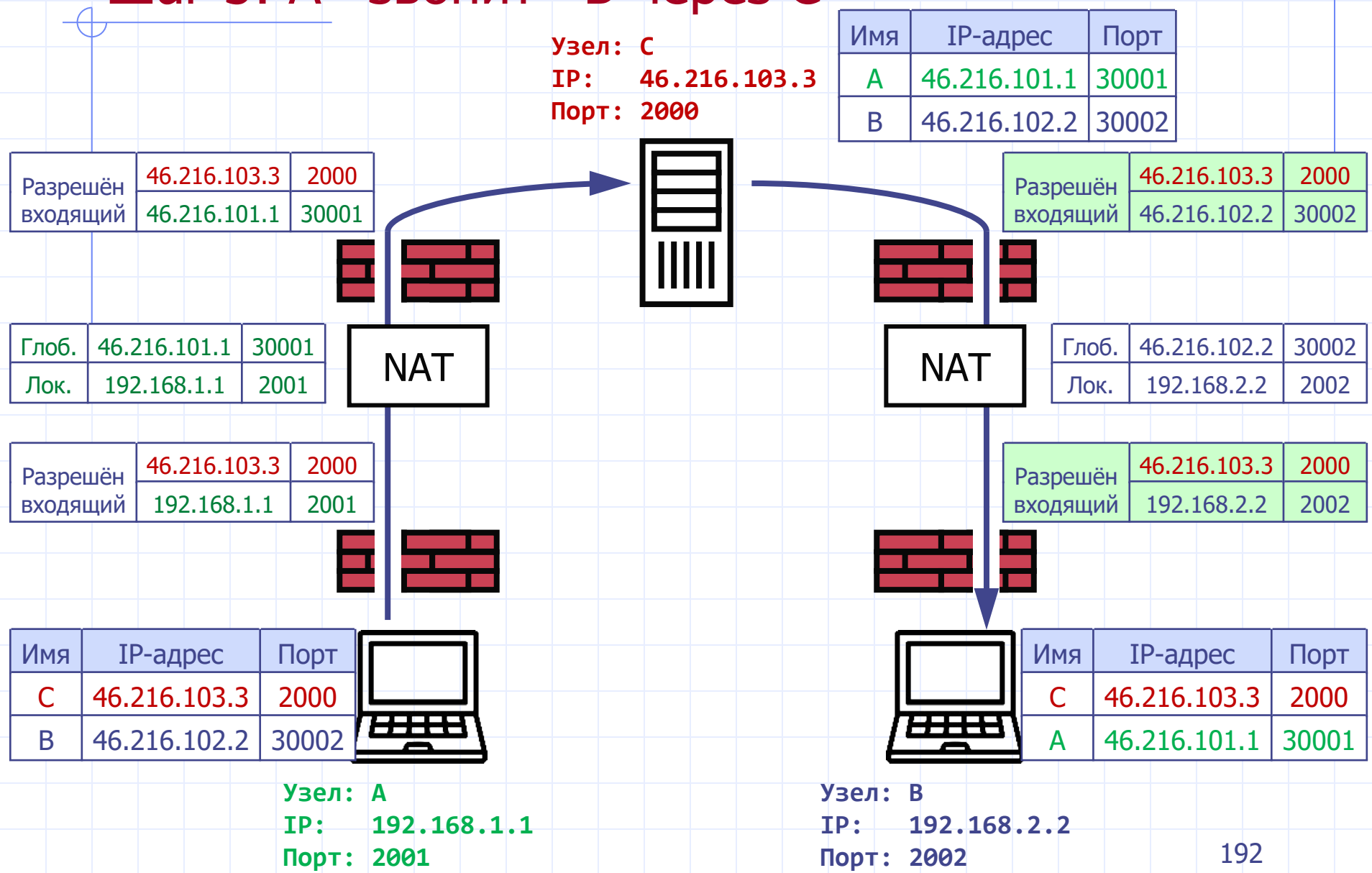
Равноправное взаимодействие по UDP + NAT. Шаг 1: A и B связываются с C



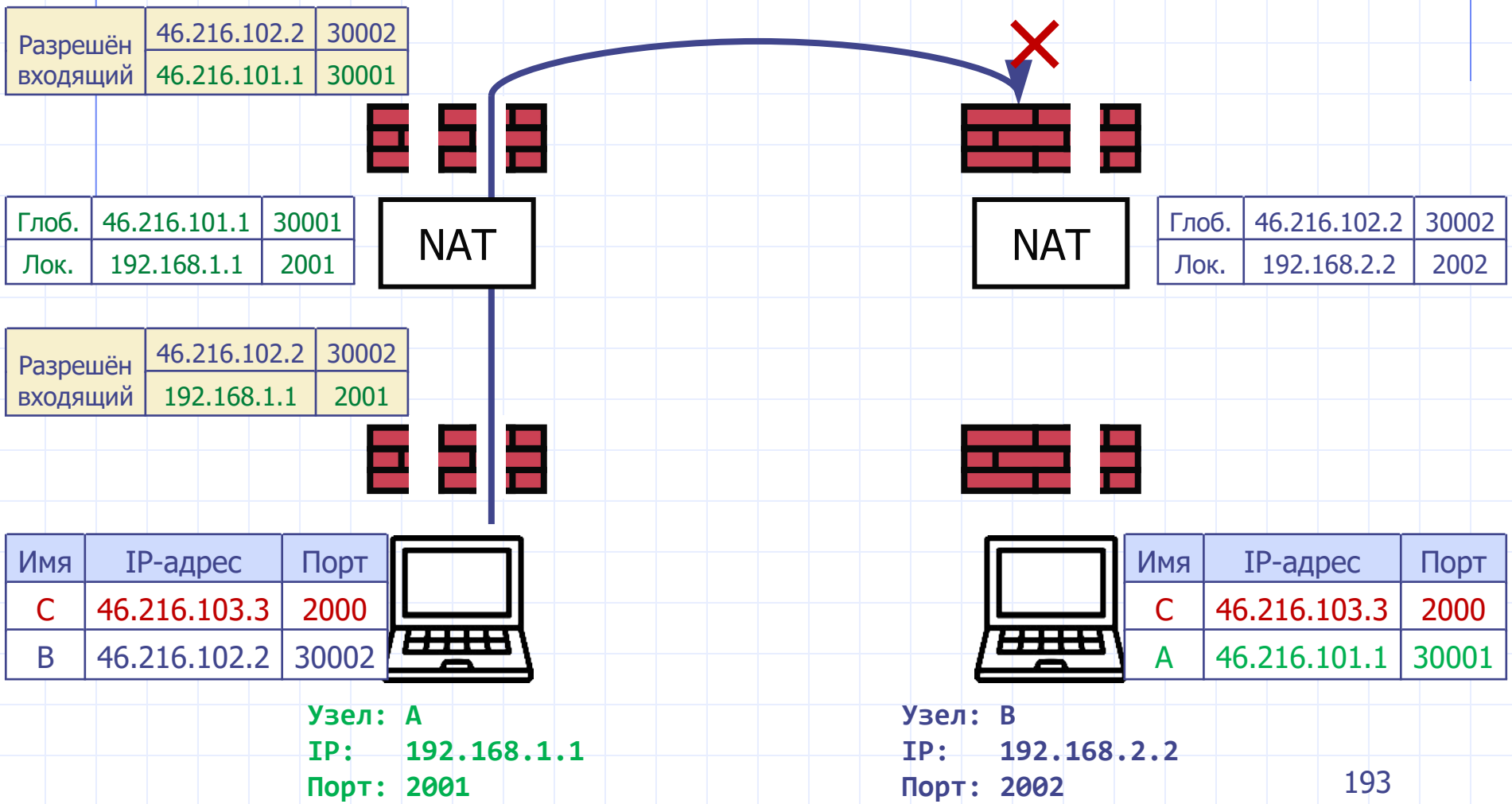
Равноправное взаимодействие по UDP + NAT. Шаг 2: C отвечает A и B



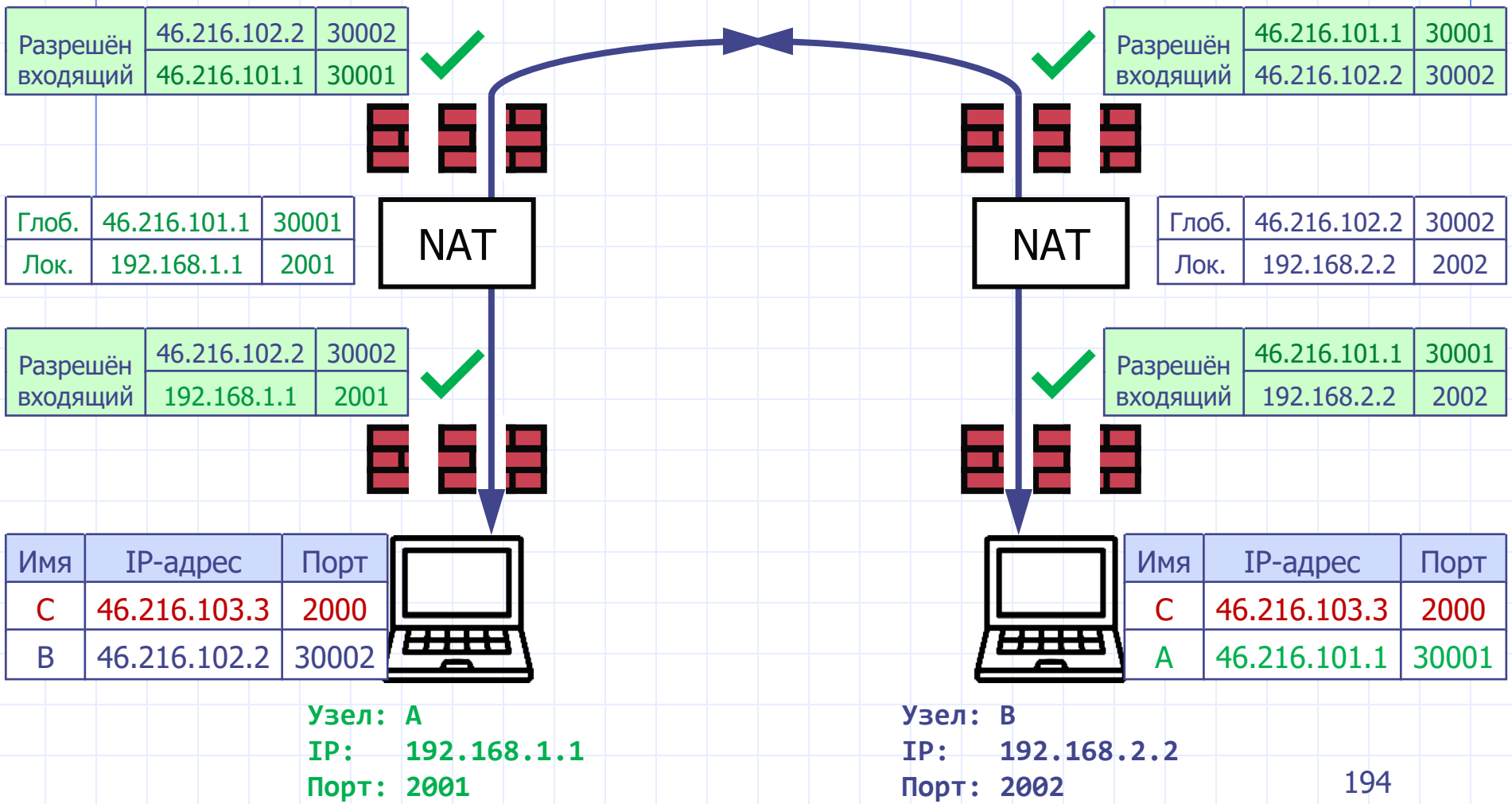
Равноправное взаимодействие по UDP + NAT. Шаг 3: A «звонит» B через C



Равноправное взаимодействие по UDP + NAT. Шаг 4: A «звонит» B напрямую



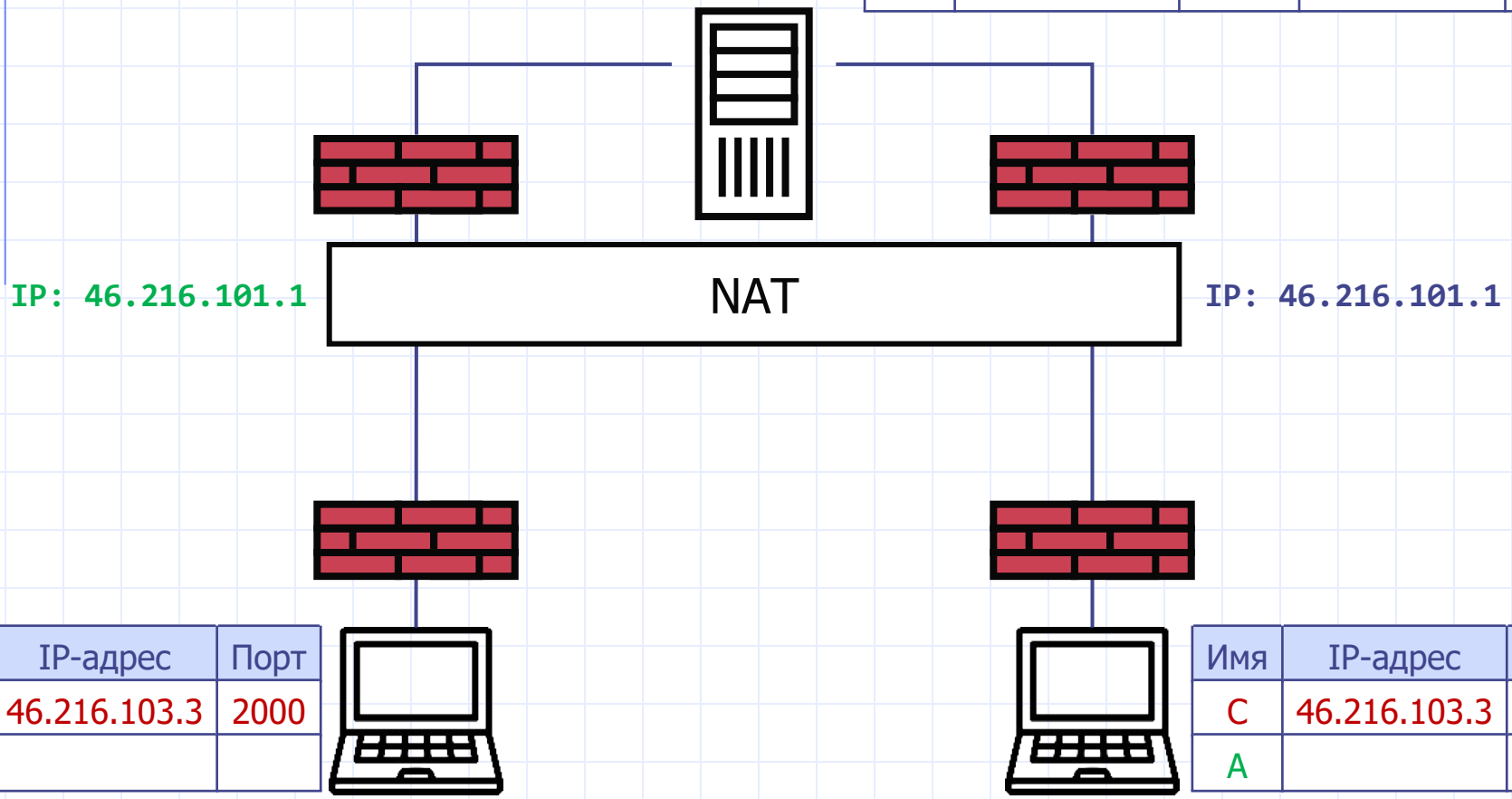
Равноправное взаимодействие по UDP + NAT. Шаг 5: B «звонит» A навстречу



Равноправное взаимодействие по UDP + NAT, когда узлы находятся в одной локальной сети.

Узел: C
IP: 46.216.103.3
Порт: 2000

Имя	IP-адрес	Порт	IP-адрес	Порт
A				
B				



Имя	IP-адрес	Порт
C	46.216.103.3	2000
B		

Узел: A
IP: 192.168.1.1
Порт: 2001

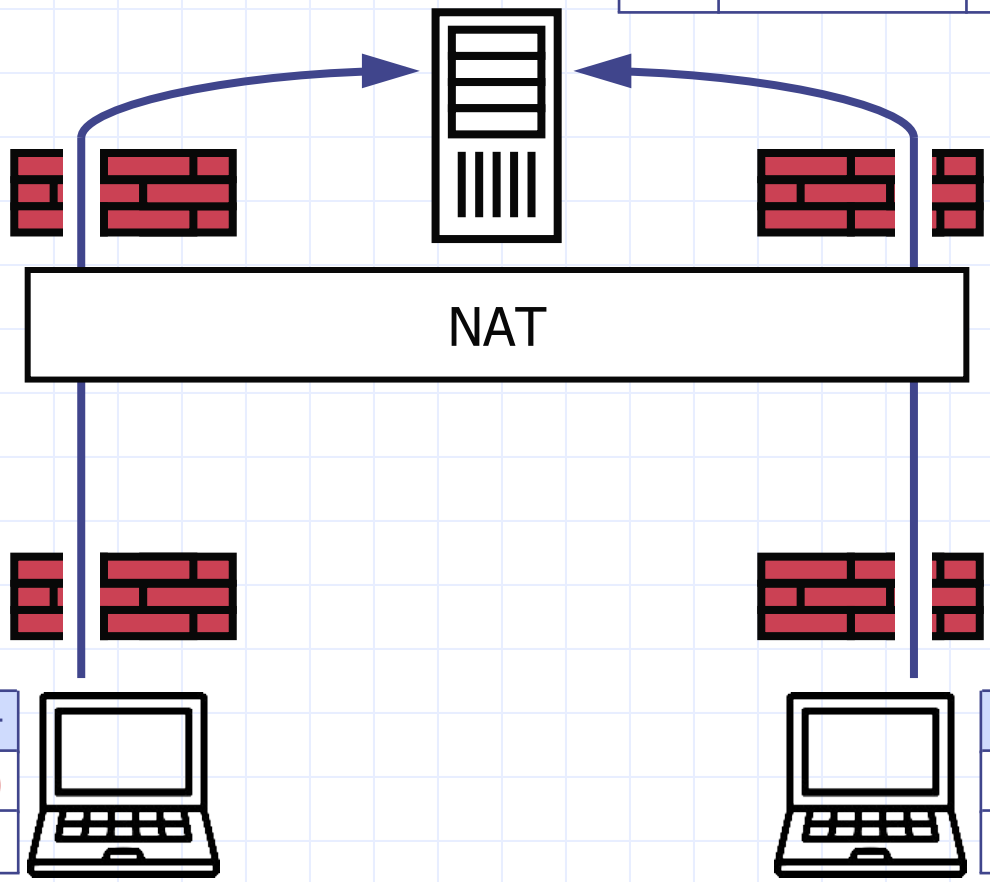
Имя	IP-адрес	Порт
C	46.216.103.3	2000
A		

Узел: B
IP: 192.168.2.2
Порт: 2002

Равноправное взаимодействие по UDP + NAT. Шаг 1: A и B связываются с C

Узел: C
IP: 46.216.103.3
Порт: 2000

Имя	IP-адрес	Порт	IP-адрес	Порт
A	46.216.101.1	30001	192.168.1.1	2001
B	46.216.101.1	30002	192.168.2.2	2002



Глоб.	46.216.101.1	30001
Лок.	192.168.1.1	2001

Глоб.	46.216.101.1	30002
Лок.	192.168.2.2	2002

Имя	IP-адрес	Порт
C	46.216.103.3	2000
B		

Имя	IP-адрес	Порт
C	46.216.103.3	2000
A		

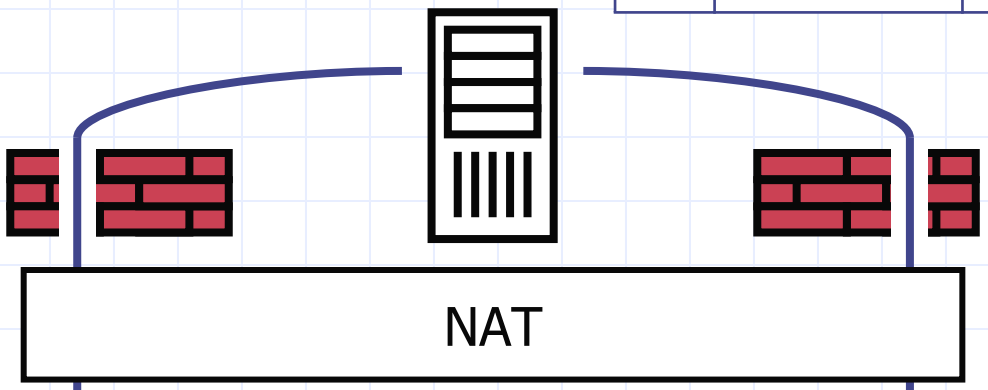
Узел: A
IP: 192.168.1.1
Порт: 2001

Узел: B
IP: 192.168.2.2
Порт: 2002

Равноправное взаимодействие по UDP + NAT. Шаг 2: C отвечает A и B

Узел: C
IP: 46.216.103.3
Порт: 2000

Имя	IP-адрес	Порт	IP-адрес	Порт
A	46.216.101.1	30001	192.168.1.1	2001
B	46.216.101.1	30002	192.168.2.2	2002



Глоб.	46.216.101.1	30001
Лок.	192.168.1.1	2001

Глоб.	46.216.101.1	30002
Лок.	192.168.2.2	2002

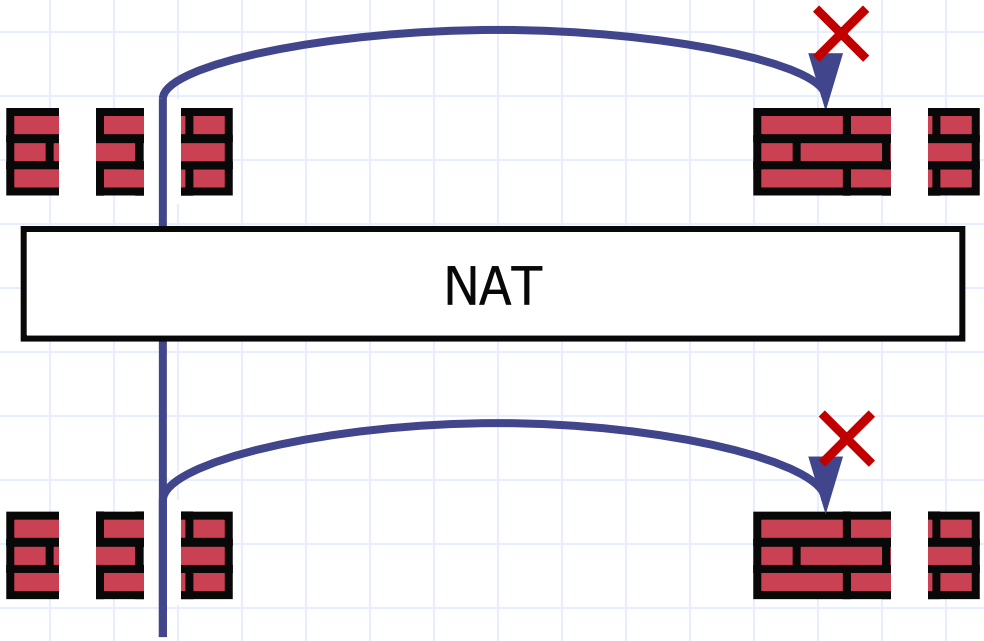
Имя	IP-адрес	Порт
C	46.216.103.3	2000
B	46.216.101.1	30002
	192.168.2.2	2002

Узел: A
IP: 192.168.1.1
Порт: 2001

Имя	IP-адрес	Порт
C	46.216.103.3	2000
A	46.216.101.1	30001
	192.168.1.1	2001

Узел: B
IP: 192.168.2.2
Порт: 2002

Равноправное взаимодействие по UDP + NAT. Шаг 4: A «звонит» B напрямую



Глоб.	46.216.101.1	30001
Лок.	192.168.1.1	2001

Глоб.	46.216.101.1	30002
Лок.	192.168.2.2	2002

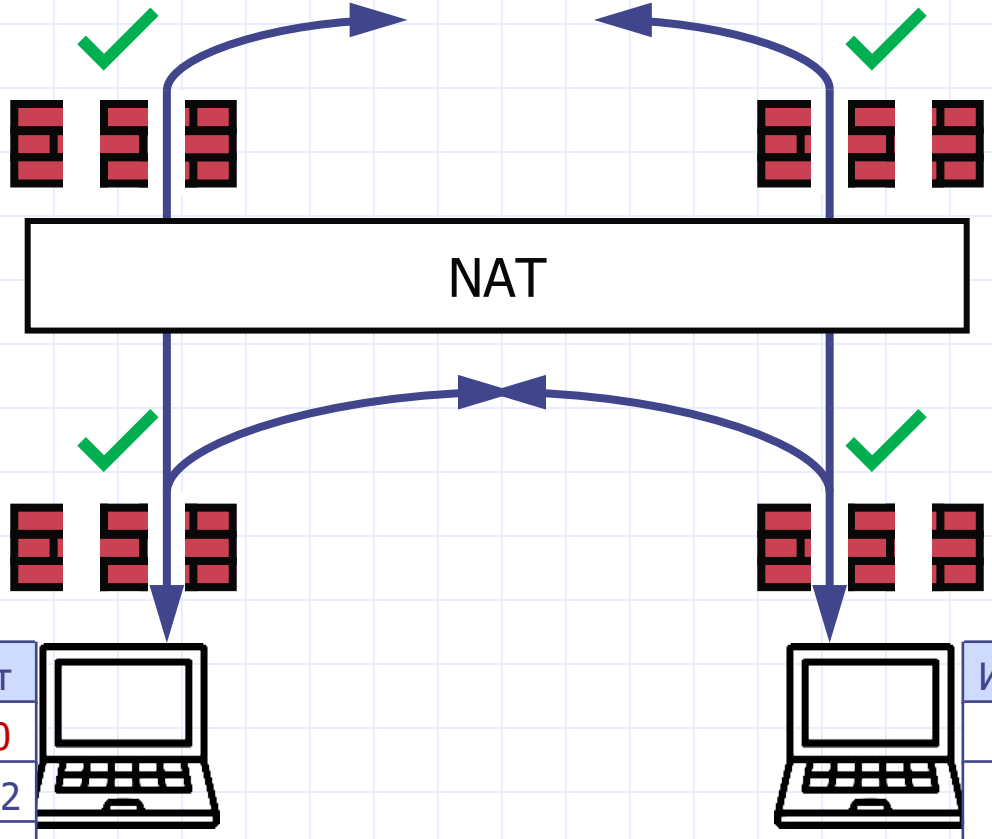
Имя	IP-адрес	Порт
C	46.216.103.3	2000
B	46.216.101.1	30002
	192.168.2.2	2002

Имя	IP-адрес	Порт
C	46.216.103.3	2000
A	46.216.101.1	30001
	192.168.1.1	2001

```
Узел: A
IP:   192.168.1.1
Порт: 2001
```

```
Узел: B
IP:   192.168.2.2
Порт: 2002
```

Равноправное взаимодействие по UDP + NAT. Шаг 5: B «звонит» A навстречу



Глоб.	46.216.101.1	30001
Лок.	192.168.1.1	2001

Глоб.	46.216.101.1	30002
Лок.	192.168.2.2	2002

Имя	IP-адрес	Порт
C	46.216.103.3	2000
B	46.216.101.1	30002
	192.168.2.2	2002

Имя	IP-адрес	Порт
C	46.216.103.3	2000
A	46.216.101.1	30001
	192.168.1.1	2001

Узел: A
IP: 192.168.1.1
Порт: 2001

Узел: B
IP: 192.168.2.2
Порт: 2002

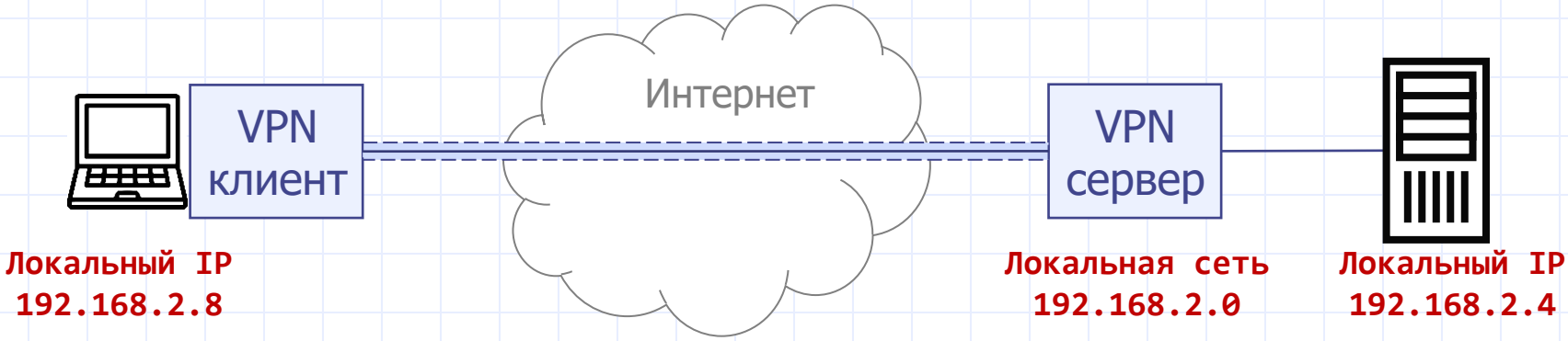
Технология виртуальных частных сетей VPN (англ. Virtual Private Network)

- Реализуется за счёт *псевдо-каналов* и *туннелирования* пакетов.
  - Псевдо-канал – эмуляция телекоммуникационного канала поверх сети с коммутацией пакетов.
  - Туннелирование – передача пакетов некоторого протокола (например, IP) в пакетах другого протокола.
- Решает задачу объединения нескольких физически обособленных сетей в одну виртуальную частную сеть.
  - Например, объединить несколько офисов одного предприятия.
- Позволяет удалённому клиенту подключиться к другой локальной сети и стать частью этой сети.
  - Например, сотруднику подключиться удалённо к локальной сети предприятия.
- Позволяет виртуально изменить своё местоположение с помощью поставщика услуг VPN.
  - Например, выйти в глобальную сеть от имени узла, расположенного в другой стране.

Способы построения виртуальной частной сети VPN

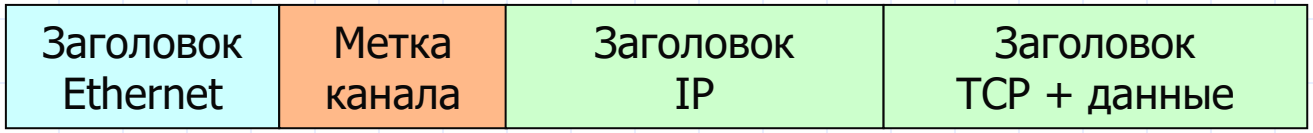
- Объединение локальных сетей в одну

- Удалённое подключение к локальной сети

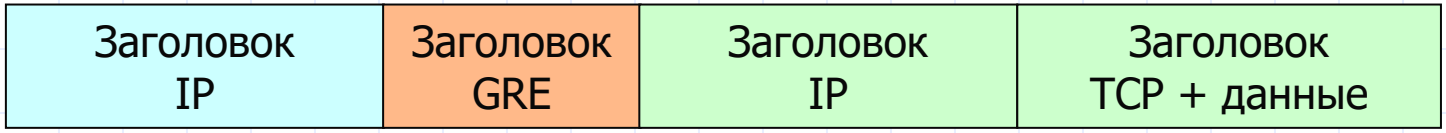


Туннелирование. Способы вложения (инкапсуляции) пакетов

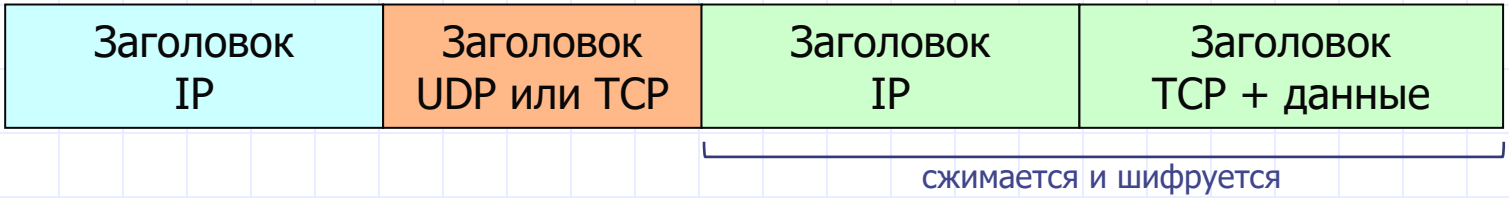
- На канальном уровне



- На сетевом уровне

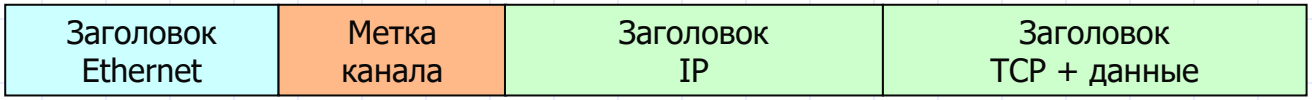


- На транспортном уровне

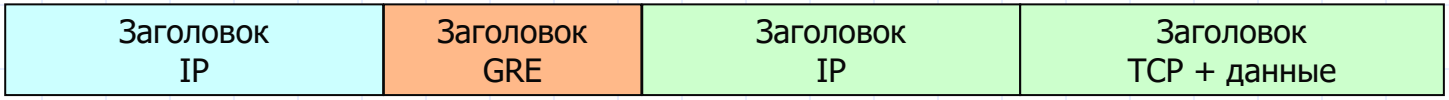


Объединение нескольких обособленных локальных сетей в одну (виртуальную) частную сеть

- Соединение на физическом уровне.
  - Ethernet, Wi-Fi имеют малое расстояние и плохо подходят. Модемное (ADSL) соединение по 2-х проводной линии (до 6 км). Радиорелейная линия (десятки км). Оптоволокно (сотни км).
- Создание псевдо-каналов на канальном уровне.
  - MPLS (англ. Multi-Protocol Label Switching) – многопротокольная коммутация по меткам.
  - VPWS (англ. Virtual Private Wire Service) – «глобальный кабель», подключение двух ЛВС.
  - VPLS (англ. Virtual Private LAN Service) – виртуальная локальная сеть, полно-связная топология.

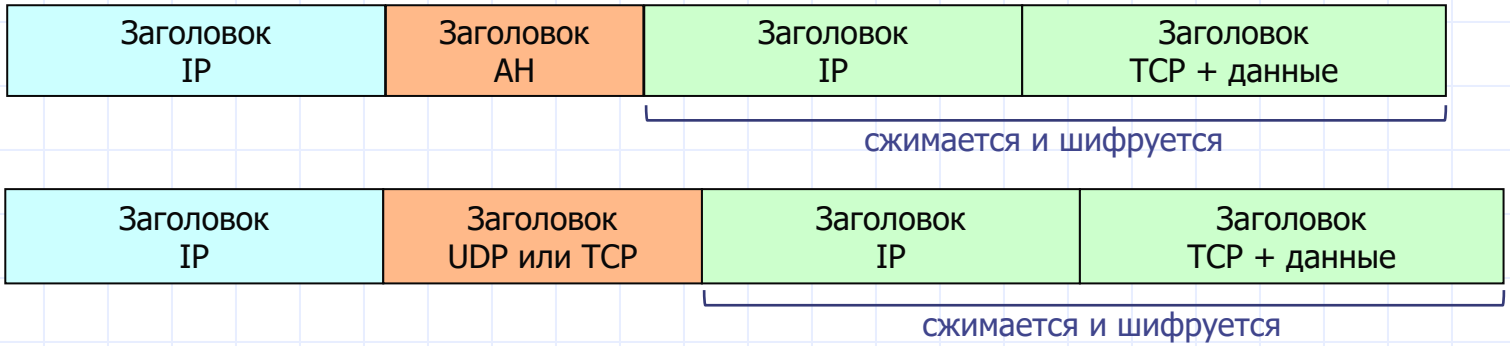


- Создание туннелей на сетевом уровне.
  - GRE (англ. Generic Routing Encapsulation) – транспортный протокол инкапсуляции данных.
  - IPsec (англ. Internet Protocol Security) – набор протоколов для обеспечения защиты данных, передаваемых по IP-сети. Работает на сетевом уровне. Шифрует весь IP-пакет, используя:
    - Authentication Header (AH), который ставит цифровую подпись на каждом пакете;
    - Encapsulating Security Protocol (ESP), который обеспечивает конфиденциальность, целостность и аутентификацию пакета при передаче.
    - Internet Key Exchange protocol (IKE) для согласование алгоритмов шифрования, проверки целостности и аутентификации участниками друг друга.



Подключение удалённого клиента к другой локальной сети и прозрачная работа в этой сети

- PPTP (англ. Point-to-Point Tunneling Protocol) – протокол туннелирования точка-точка. Использует два соединения: одно для управления (на основе TCP, порт 1723), другое для инкапсуляции данных (на основе протокола GRE).
- SSTP (англ. Secure Socket Tunneling Protocol) – использует TCP и SSL (порт 443).
- L2TP/IPsec. Layer 2 Tunneling Protocol (L2TP) работает на основе UDP, сам по себе не обеспечивает шифрование или аутентификацию, поэтому используется вместе с IPsec. L2TP/IPsec инкапсулирует передаваемые данные дважды, что делает его менее эффективным и более медленным, чем другие VPN-протоколы.
- IKEv2/IPsec. Internet Key Exchange version 2 (IKEv2) является протоколом IPsec, используемым для выполнения взаимной аутентификации.
- OpenVPN – универсальный протокол VPN с открытым исходным кодом. Использует стандартные протоколы TCP и UDP. Требует специальное клиентское ПО.



Виртуальное изменение местоположения выхода в глобальную сеть

- VPN-клиент обычно создаёт виртуальный (мнимый) сетевой адаптер с сетевым подключением через VPN-соединение. Для работы через VPN-соединение клиент получает отдельные настройки IP-адреса, маски и шлюза.
- Клиентский узел оказывается подключённым как бы к двум (или более) сетям: к своей локальной сети и к виртуальной частной сети. У каждой сети – свои настройки IP-адреса, маски и шлюза.
- При выходе к глобальную сеть клиент может использовать один из двух шлюзов:
  - шлюз, соответствующий VPN-соединению. В этом случае клиент выходит в глобальную сеть от имени VPN-сервера из виртуальной частной сети;
  - шлюз, соответствующий своей локальной сети В этом случае клиент выходит в глобальную сеть от имени своего маршрутизатора (с NAT) из своей локальной сети.

Прикладной уровень OSI

- Реализует прикладные сетевые службы:
  - FTP (англ. File Transfer Protocol) – протокол передачи файлов.
  - Telnet (англ. Teletype network) – протокол виртуального терминала (для ввода-вывода текста).
  - POP3 (англ. Post Office Protocol) – протокол доступа к почтовому ящику.
  - IMAP (англ. Internet Message Access Protocol) – протокол доступа и управления сообщениями почтового ящика.
  - SMTP (англ. Simple Mail Transfer Protocol) – протокол передачи почты.
  - HTTP (англ. Hyper-Text Transfer Protocol) – протокол передачи гипертекстовой информации.
- Базовые прикладные протоколы являются диалоговыми (запрос-ответ) и используют текстовое представление запросов и ответов.

7. Прикладной
6. Представительный
5. Сеансовый
4. Транспортный
3. Сетевой
2. Канальный
1. Физический

Протокол передачи файлов FTP (англ. File Transfer Protocol)

- Использует раздельные соединения для управления и передачи файлов.
- Соединение для управления устанавливается на порт сервера 21. Через это соединение передаются текстовые команды и ответы на них.
- Соединение для передачи файлов открываются по мере необходимости. Инициатором соединения может выступать как сервер, так и клиент.
- Режимы работы:
  - Активный – инициатором соединения является сервер. Не пригоден, если клиент находится за устройством PAT или сетевым экраном.
  - Пассивный – инициатором соединения является клиент. Позволяет клиенту использовать технологию PAT и сетевой экран.
- После установления TCP-соединения клиент выполняет аутентификацию и авторизацию. Имя пользователя и пароль передаются в открытом виде.
- После авторизации клиент с помощью соответствующих команд может: просматривать содержимое каталогов, менять текущий каталог, управлять файлами и каталогами на сервере, отправлять файлы на сервер и получать файлы с сервера.

Команды FTP для управления доступом и параметрами передачи

Команда	Описание
Команды управления доступом	
USER name	Войти на сервер с использованием указанного пользователя (подключение уже должно быть установлено). Далее FTP-сервер запросит пароль. Возможен анонимный вход.
PASS pass	Передать серверу пароль. Выполняется сразу после команды USER.
ACCT info	Войти на сервер с использованием указанной учётной записи. Учётная запись не обязательно связана с пользователем, а команда с командой USER.
CWD path	Сменить текущий каталог на сервере.
CDUP	Перейти в родительский каталог на сервере.
SMNT path	Монтирование другой файловой системы по указанному пути на сервере.
REIN	Выход пользователя без разрыва соединения с сервером.
QUIT	Выход пользователя и разрыв соединения с сервером.
Команды параметров передачи	
PORT value	Войти в активный режим с указанными в команде IP-адресом и портом. Сервер сам подключается к клиенту для передачи данных.
PASV	Войти в пассивный режим. Сервер вернёт IP-адрес и порт, которые он «прослушивает» и к которым нужно подключиться, чтобы забрать данные.
TYPE code	Задать тип представления и хранения данных, в частности: ASCII (текстовый), Image (бинарный), интерпретацию «CRLF», а также логический размер байта в битах.
STRU code	Задать структуру файла: файл без структуры, файл из записей, файл из страниц.
MODE code	Задать режим передачи: потоком, блоками, со сжатием повторяющихся байтов.

Команды FTP для передачи файлов

Команды передачи файлов	
RETR path	Начать передачу файла от сервера клиенту.
STOR path	Начать передачу файла от клиента серверу.
STOU	Аналогична STOR, но для файла генерируется уникальное имя в каталоге.
APPE path	Аналогична STOR, но данные добавляются в конец файла, если он уже существует.
ALLO args	Выделяет указанное количество байтов для файла в файловой системе на сервере.
REST marker	Возобновить разорванную передачу файла с заданной контрольной точки.
RNFR, RNTO	Переименовать файл. RNFR — что переименовывать, RNTO — во что переименовать.
ABOR	Прервать выполнение предыдущей команды (передачи). Соединение с сервером остаётся.
DELE path	Удалить указанный файл на сервере.
RMD path	Удалить указанный каталог на сервере (по абсолютному или относительному пути).
MKD path	Создать указанный каталог на сервере (по абсолютному или относительному пути).
PWD	Получить путь текущего каталога на сервере.
LIST [path]	Получить список файлов указанного каталога или информацию об указанном файле.
NLST [path]	Получить список файлов указанного или текущего каталога
SITE string	Специальная команда, позволяющая реализовать свои особые функции FTP-серверу.
SYST	Узнать тип операционной системы сервера.
STAT path	Узнать состояние передачи файлов.
HELP	Получить дополнительную информацию о сервере.
NOOP	Пустая команда «ничего не делать» для проверки сервера.

Пример диалога по протоколу FTP

```
C: USER anonymous\r\n
S: 331 Please specify the password.\r\n
C: PASS secret123\r\n
S: 230 Login successful.\r\n
C: PORT 46,216,50,103,5,231\r\n
S: 200 PORT command successful. Consider using PASV.\r\n
C: NLST\r\n
S: 150 Here comes the directory listing.\r\n
D: byfly\r\ndebian\r\ndebian-backports\r\ndebian-cd\r\ndebian-multimedia\r\n
debian-security\r\npub\r\nreleases\r\nrobots.txt\r\ntest\r\nubuntu\r\n
S: 226 Directory send OK.\r\n
C: CWD test\r\n
S: 250 Directory successfully changed.\r\n
C: PASV\r\n
S: 227 Entering Passive Mode (82,209,230,71,225,243).\r\n
C: RETR 10Mb\r\n
S: 150 Opening BINARY mode data connection for 10Mb (10485760 bytes).\r\n
D: <передача 10’485’760 байтов бинарных данных по новому соединению>
S: 226 Transfer complete.\r\n
C: QUIT\r\n
S: 221 Goodbye.\r\n
```

Адрес и порт записаны побайтово в десятичной форме, каждый байт через запятую (порт: 2 байта).

C – Client
S – Server
D – Data

D: Передача данных по новому соединению от сервера (порт 20) клиенту (порт 1511 = 256*5 + 231).

D: Клиент забирает данные с сервера (порт сервера 57843 = 256*225 + 243).

Протокол доступа к почтовому ящику POP3 (англ. Post Office Protocol v3)

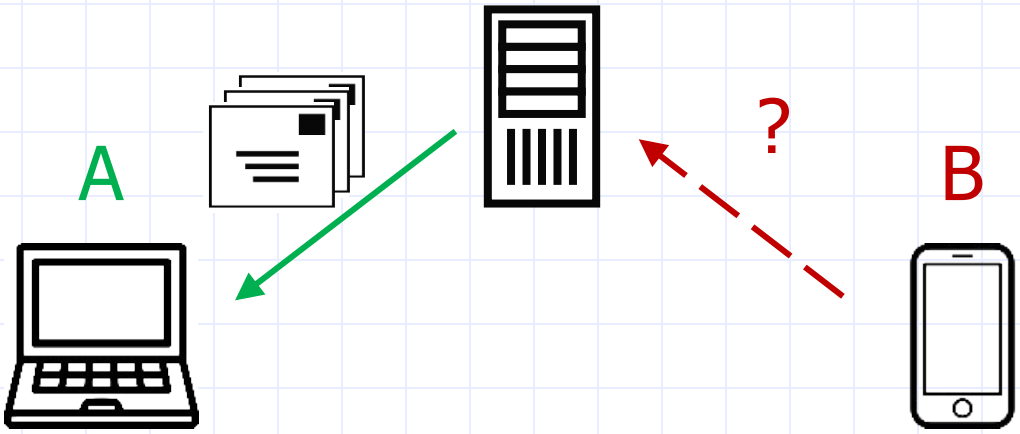
- Порт сервера 110.
- Служит для изъятия части или всей почты из почтового ящика.
- Ориентирован на малый размер хранилища почтовых ящиков на сервере.
- Ориентирован на работу с почтой в автономном режиме.
- Состояния сеанса:
  - AUTHORIZATION (требуется вход) – состояние после подключения клиента к серверу; необходимо авторизоваться для получения доступа к ящику.
  - TRANSACTION (накопление изменений) – состояние после успешной авторизации; сервер заблокировал ящик, и клиент работает с почтой; все изменения накапливаются, но не применяются сразу.
  - UPDATE (применение изменений) – состояние после выхода клиента: сервер применяет накопленные изменения.
- После авторизации клиент может: просматривать состояние ящика, просматривать весь список и содержимое отдельных писем, помечать их к удалению, отменять все текущие изменения.
- Положительный ответ сервера: **+OK ...**, отрицательный: **-ERROR ...**

Команды POP3

Команда	Описание
Команды, доступные в любом состоянии	
NOOP	Пустая команда «ничего не делать» для проверки сервера.
QUIT	Выход пользователя и разрыв соединения с сервером.
Команды в состоянии AUTHORIZATION	
USER name	Войти на сервер с использованием указанного пользователя (подключение уже должно быть установлено). Далее POP3-сервер запросит пароль.
PASS password	Передать серверу пароль. Выполняется сразу после команды USER.
APOP name digest	Войти на сервер без передачи пароля в открытом виде. **digest** рассчитывается как MD5-хэш от специальной временной метки сервера, которая выдаётся клиенту при подключении, и пароля. Поддерживается не на всех серверах.
Команды в состоянии TRANSACTION	
STAT	Отобразить состояние ящика: количество писем в ящике, общий размер в байтах.
LIST [msg]	Если параметр [msg] (номер сообщения) не указан – показать список писем: **<номер-сообщения> <размер-в-байтах>**. Если параметр [msg] указан – вывести информацию о письме: заголовок (тема, отправитель, получатель и т.д.)
RETR msg	Получить содержимое письма с номером msg.
TOP msg n	Вывести заголовок и первые n строк письма с номером msg. Поддерживается не на всех серверах.
DELE msg	Пометить письмо к удалению.
RSET	Сбросить текущие изменения – убрать с писем пометки к удалению.

Недостатки протокола POP3

- Отсутствие возможности управления письмами на сервере – перемещение по папкам, назначение ярлыков.
- Необходимость «забирать» письма с почтового сервера для полноценной работы с почтой.
- Отсутствие возможности работы несколькими клиентами с одним почтовым ящиком.
  - Клиент A «забрал» письма, и клиент B их «не увидит».



Протокол доступа и управления сообщениями IMAP (англ. Internet Message Access Protocol)

- Порт сервера 143.
- Ориентирован на интерактивный режим работы с почтовым ящиком через интернет.
- Возможен одновременный доступ нескольких клиентов к одному ящику.
- Возможность работы с несколькими ящиками (папками: «Входящие», «Исходящие», «Помеченные» и др.).
- Гибкое управление почтой на сервере: ярлыки, пометки «важное», перемещение писем между ящиками.
- Возможность отслеживать состояние письма (например, «прочитано», «отправлен ответ», «отложено», «удалено»).
- Состояния сеанса:
  - Not Authenticated (клиент не аутентифицировался)
  - Authenticated (клиент аутентифицировался)
  - Selected (клиент выбрал ящик)
  - Logout (клиент выходит)

Команды IMAP

Команда	Описание
Команды, доступные в любом состоянии	
CAPABILITY	Получить список команд, поддерживаемых сервером.
NOOP	Пустая команда «ничего не делать» для проверки сервера.
LOGOUT	Выход пользователя и разрыв соединения с сервером.
Команды в состоянии Not Authenticated	
AUTHENTICATE	Запросить альтернативный метод аутентификации.
LOGIN	Передать серверу имя пользователя и пароль в открытом виде для аутентификации.
Команды в состоянии Authenticated	
SELECT	Выбрать ящик для работы с письмами. Позволяет изменять содержимое ящика.
EXAMINE	Выбрать ящик для «осмотра». Просмотр содержимого и писем без изменения (в т.ч. остаётся пометка «Recent»).
CREATE	Создать новый ящик с заданным именем.
DELETE	Удалить ящик с заданным именем.
RENAME	Переименовать ящик.
SUBSCRIBE	«Подписаться» на изменения в ящике, добавить его в список активных.
UNSUBSCRIBE	«Отписаться» от изменений в ящике, изъять его из списка активных.
LIST	Отобразить ящики, имена которых соответствуют заданному шаблону.
LSUB	Отобразить ящики, на которые произведена «подписка», с именами по шаблону.
STATUS	Отобразить состояние ящика с заданным именем: общее число писем, число новых и др.
APPEND	Добавить новое сообщение в конец указанного ящика. Задаётся заголовок и содержимое.

Команды IMAP (продолжение)

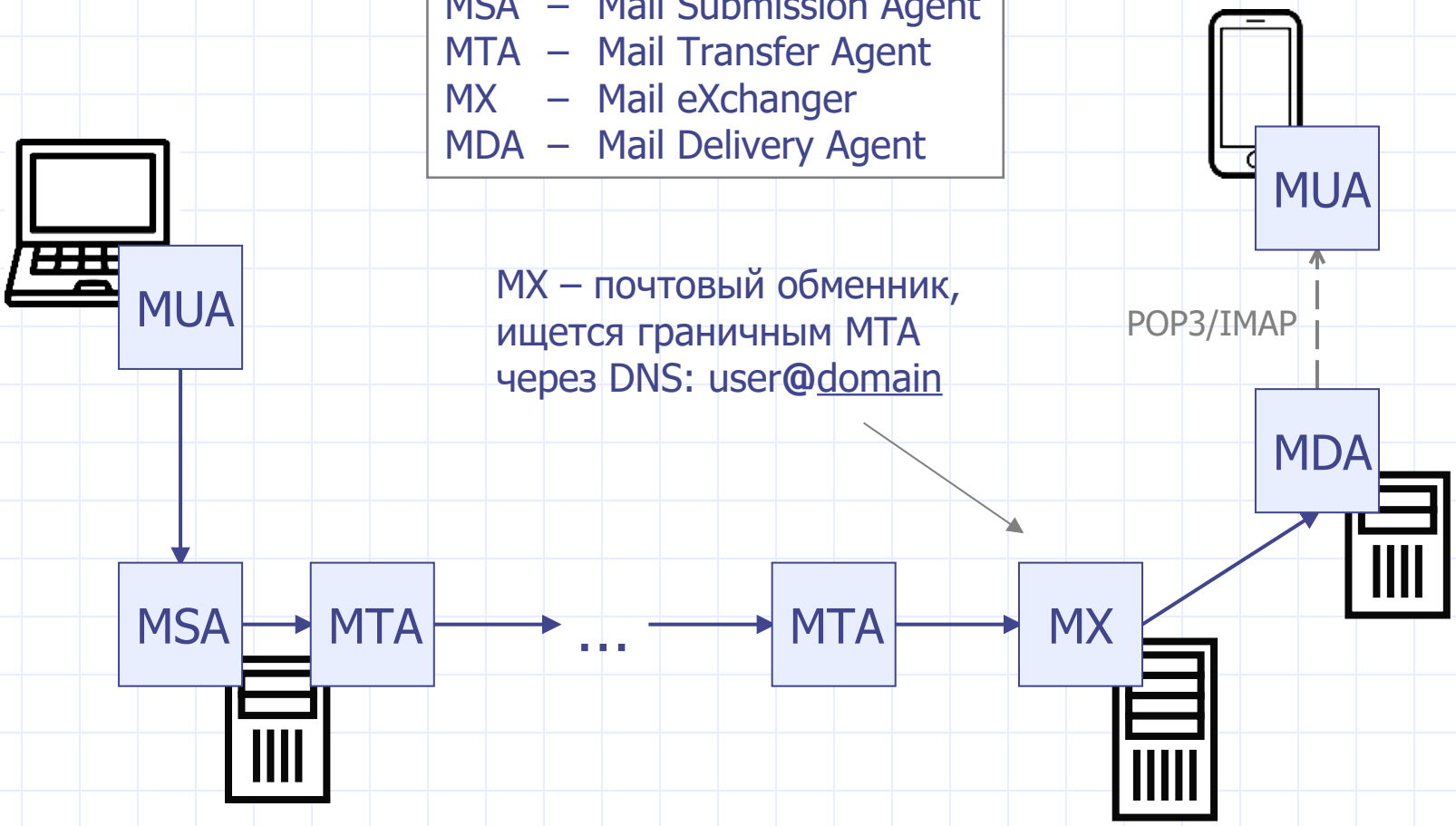
Команда	Описание
Команды в состоянии Selected	
CHECK	Провести проверку выбранного ящика. Зависит от реализации сервера.
CLOSE	Закрыть выбранный ящик и перейти в состояние Authenticated. Удаляет все помеченные к удалению письма (EXPUNGE), если ящик был открыт командой SELECT.
EXPUNGE	Окончательно удалить все помеченные к удалению письма.
SEARCH	Поиск писем по разным параметрам: отправитель, получатель, тема, копия, скрытая копия, флаги, дата, размер, содержимое и т.д.
FETCH	Отобразить письмо с разной степенью детализации (флаги, дата, заголовок, тело).
STORE	Изменить метаданные письма (флаги).
COPY	Скопировать одно или несколько сообщений в другой ящик.
UID	Позволяет командам COPY, FETCH, STORE работать с уникальными идентификаторами писем вместо их порядкового номера в ящике. Преобразует результат команды SEARCH из порядковых номеров в уникальные идентификаторы.

Протокол передачи почты SMTP (англ. Simple Mail Transfer Protocol)

- Порт сервера 25.
- Обеспечивает доставку электронных писем, не рассчитан на работу с входящей почтой.
- Почтовые клиенты используют SMTP для отправки писем (современный порт сервера для отправки – 587).
- Промежуточные сервера используют SMTP для пересылки отправленных писем адресату, на нужный почтовый сервер.
- Данные в SMTP представлены в 7-разрядных символах ASCII. Каждый символ передаётся как 8 битов с установленным в 0 старшим разрядом. Расширения стандарта SMTP могут разрешить передачу полноценных 8-разрядных байтов данных.

Модель передачи электронной почты

MUA – Mail User Agent
MSA – Mail Submission Agent
MTA – Mail Transfer Agent
MX – Mail eXchanger
MDA – Mail Delivery Agent



Команды SMTP

CR – Carriage Return (\r)
LF – Line Feed (\n)

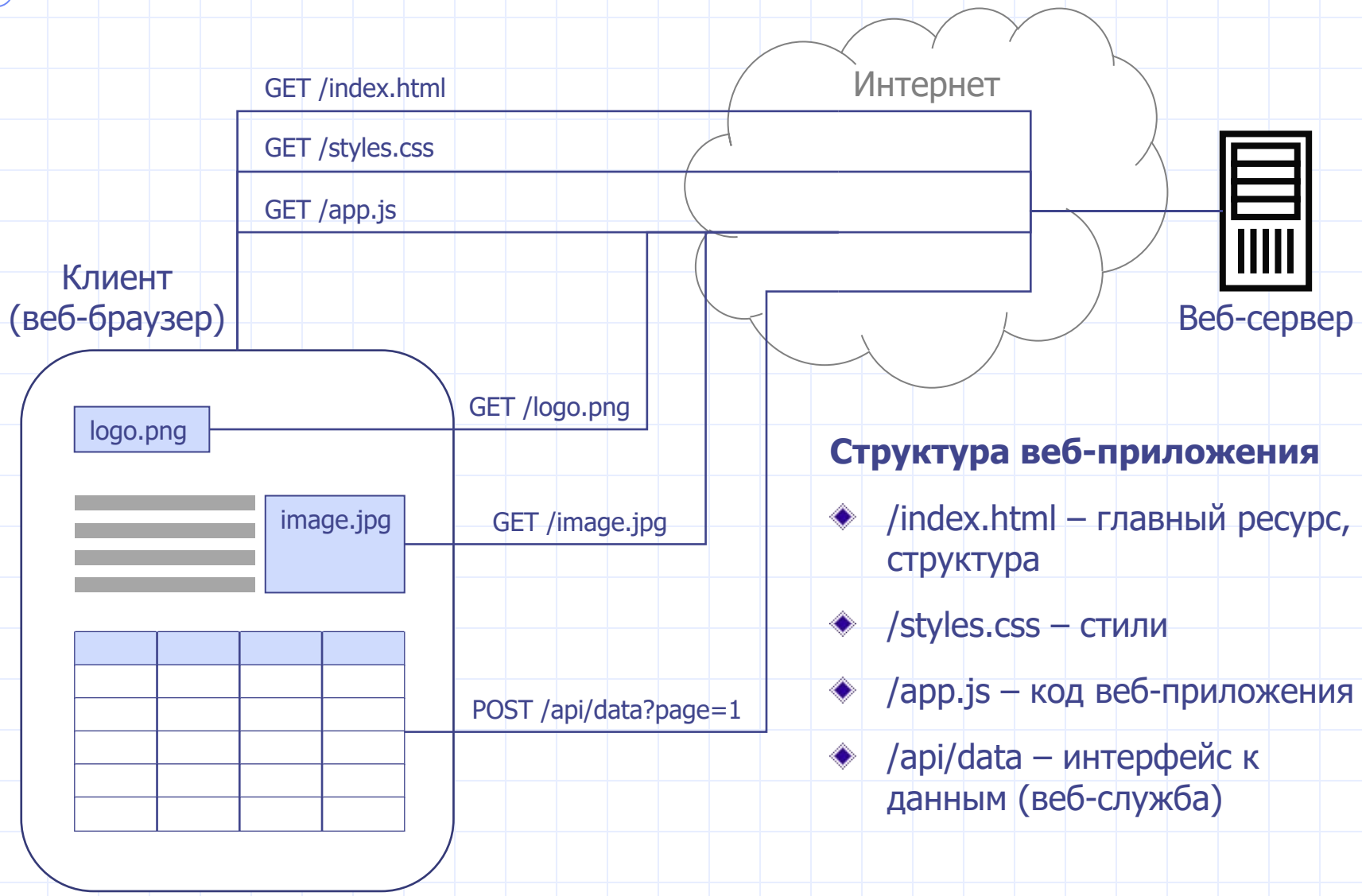
Команда	Описание
HELO, EHLO	«Приветствие», в ответ на которое сервер SMTP идентифицирует себя. EHLO – расширенная версия, указывает варианты аутентификация.
AUTH	Аутентифицировать отправителя (расширение SMTP).
VRFY	Verify – Проверить правильность указанного почтового адреса. Если на текущем сервере имеется указанный пользователь, возвращает информацию о нём.
EXPN	Expand – Проверить правильность указанного списка рассылки. Если на текущем сервере имеется информация об указанном списке, возвращает её.
HELP	Получить справочную информацию от сервера (список доступных команд).
NOOP	Пустая команда «ничего не делать» для проверки сервера.
QUIT	Закрыть соединение с сервером.
Команды почтовых транзакций	
MAIL	Начать транзакцию по отправке письма. Указывается отправитель – **FROM**.
RCPT	Recipient – Задать получателя или список получателей письма – **TO**.
DATA	Задать содержимое письма. Должно заканчиваться символами **<CRLF>.<CRLF>**
RSET	Reset – Отменить текущую транзакцию по отправке письма. Сбрасывает все сохранённые сервером данные о письме, но не закрывает соединение.

- Строгий порядок вызова команд почтовых транзакций:
  MAIL → RCPT → DATA

Протокол передачи гипертекста HTTP (англ. Hyper-Text Transfer Protocol)

- Порт сервера 80.
- Надёжный, работает поверх TCP.
- Вездесущий, легко преодолевает сетевые экраны.
- Текстовый, понимается всеми известными платформами.
- Диалоговый, использует режим «запрос-ответ».
- Многофункциональный за счёт гибкой структуры заголовков.
- Расширяемый за счёт дополнительных заголовков (X-Custom).
- Гибкий за счёт встроенных средств переадресации.
- Масштабируемый за счёт отсутствия хранения состояния сеанса на сервере.
- Универсальный – используется для создания веб-приложений и прикладного интерфейса веб-служб.
- Существующие версии: HTTP/1.0, HTTP/1.1, HTTP/2

Протокол HTTP – основа веб-приложений

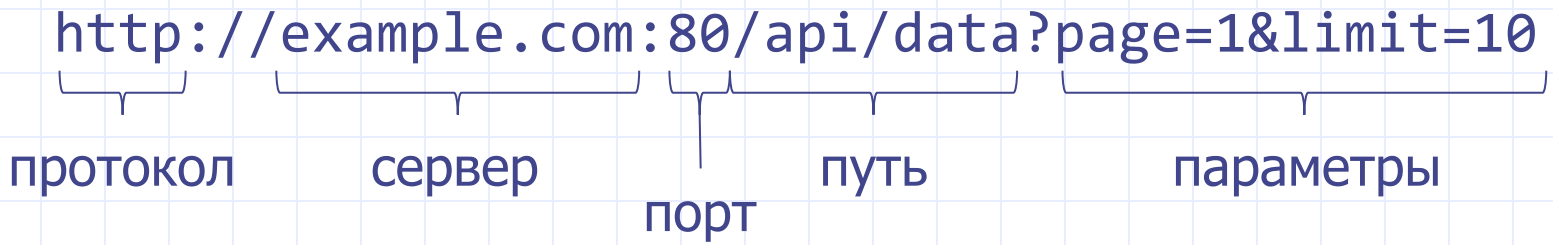


Структура веб-приложения

- /index.html – главный ресурс, структура
- /styles.css – стили
- /app.js – код веб-приложения
- /api/data – интерфейс к данным (веб-служба)

Унифицированный указатель ресурса URL (англ. Uniform Resource Locator)

- Один из видов унифицированного идентификатора ресурса URI (англ. Uniform Resource Identifier).
- Структура URL:



- Структура URL в веб-браузере:

Пример запроса и ответа HTTP/1.1

- Запрос:

```
GET / HTTP/1.1
Host: www.bntu.by
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: text/html,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en,ru-RU;q=0.9,ru;q=0.8,en-US;q=0.7
If-Modified-Since: Sun, 11 Apr 2021 18:35:42 GMT
```

- Ответ:

```
HTTP/1.1 200 OK
Server: nginx
Date: Sun, 11 Apr 2021 18:37:30 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Expires: Thu, 01 Jan 1970 00:00:01 GMT
Last-Modified: Sun, 11 Apr 2021 18:37:30 GMT
Cache-Control: no-cache
Pragma: no-cache
Content-Encoding: gzip

… (содержимое)
```

Структура запросов и ответов HTTP/1.1

- Запросы и ответы – это текст, состоящий из заголовков и содержимого.
- Заголовок – это строка, оканчивающаяся символами «CRLF».
- Блок заголовков отделяется от содержимого пустой строкой (двумя парами символов «CRLF»).
- Первый заголовок запроса содержит тип, путь и версию протокола.
- Первый заголовок ответа содержит версию протокола и код ответа.
- Все заголовки делятся на:
  - заголовки запроса;
  - заголовки ответа;
  - заголовки запросов и ответов.

```
GET / HTTP/1.1
```

```
HTTP/1.1 200 OK
```

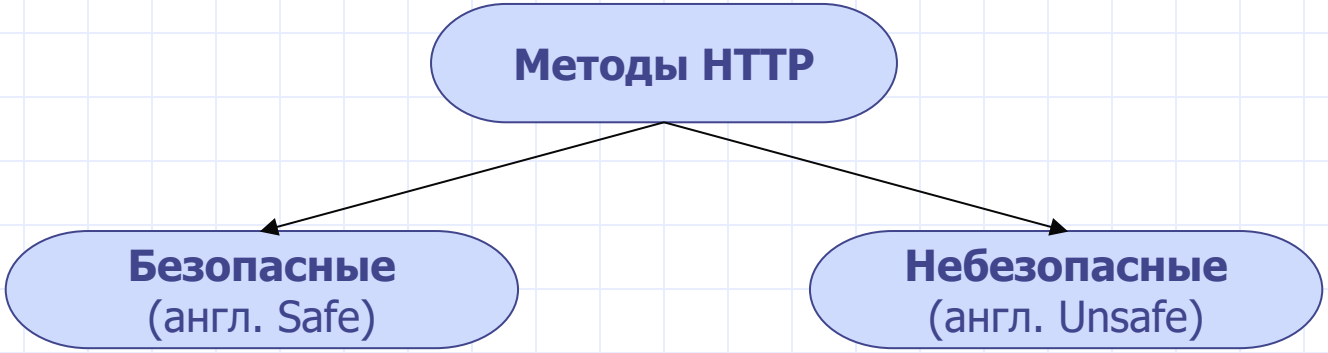
```
Host: www.bntu.by
Server: nginx
Connection: keep-alive
```

Типы запросов HTTP (методы)

- GET – получить текущее представление указанного ресурса.
- HEAD – получить информацию о текущем представлении ресурса (аналогичен GET, но в ответе только заголовки, без содержимого).
- PUT – заменить текущее представление указанного ресурса или создать новый ресурс.
- POST – отправить данные на сервер, добавив к указанному ресурсу.
- DELETE – удалить указанный ресурс.
- PATCH – обновить, частично изменить указанный ресурс (добавлен в 2010 году).
- OPTIONS – получить параметры соединения с указанным ресурсом: доступные методы, форматы.
- TRACE – проверить прохождение запроса к указанному ресурсу.

+ CONNECT – установить соединение (при использовании прокси-сервера).

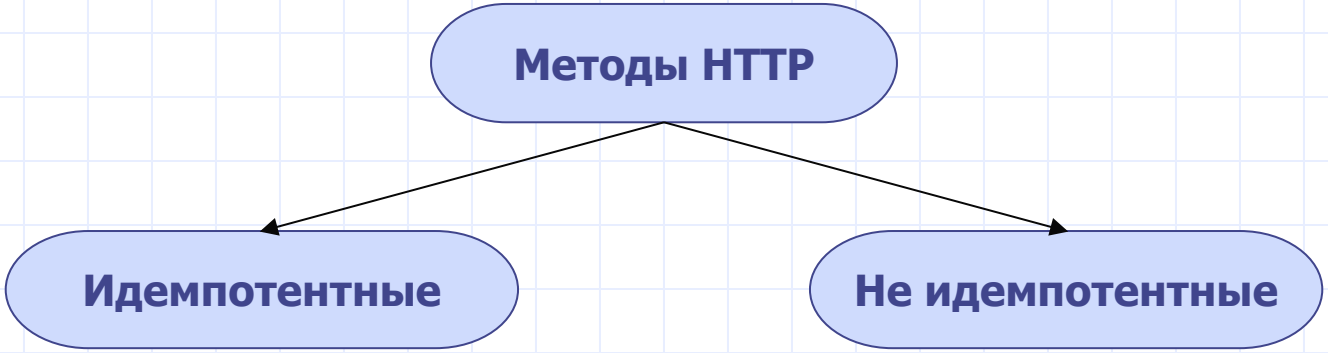
Классификация методов HTTP



- Не изменяют ресурсы на сервере (в БД).
- Запросы на чтение:
  GET, HEAD, OPTIONS, TRACE.
- Результаты GET и HEAD могут кэшироваться.

- Могут изменять ресурсы на сервере (в БД).
- Запросы на изменение:
  PUT, POST, DELETE, PATCH.

Классификация методов HTTP (продолжение)



- Результат повторных вызовов идентичен результату одиночного вызова.
- GET, HEAD, OPTIONS, TRACE, PUT, DELETE.

- Повторный вызов приводит к новому изменению.
- POST, PATCH.

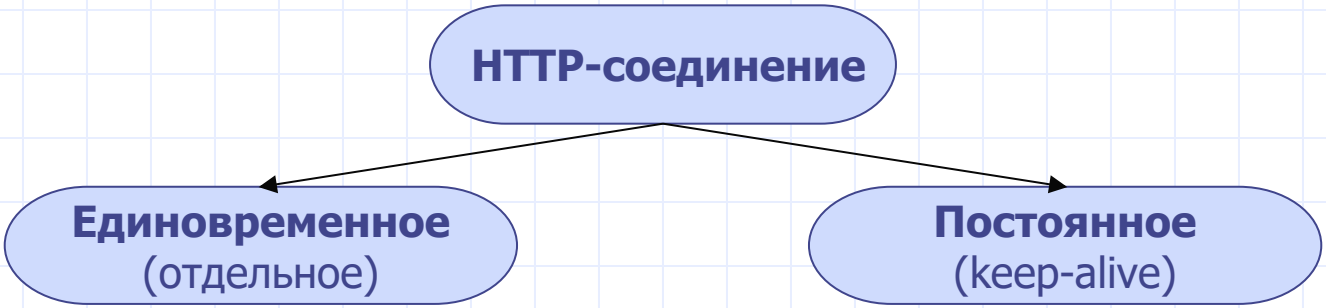
Коды ответов HTTP

- 100-199 – Информационные
- 200-299 – Успех
- 300-399 – Переадресация
- 400-499 – Ошибка со стороны клиента
- 500-599 – Ошибка со стороны сервера
- Примеры:
  - 101 – Switching Protocols
  - 200 – OK
  - 202 – Accepted
  - 301 – Moved Permanently
  - 401 – Unauthorized
  - 404 – Not Found
  - 500 – Internal Server Error

Основные заголовки HTTP

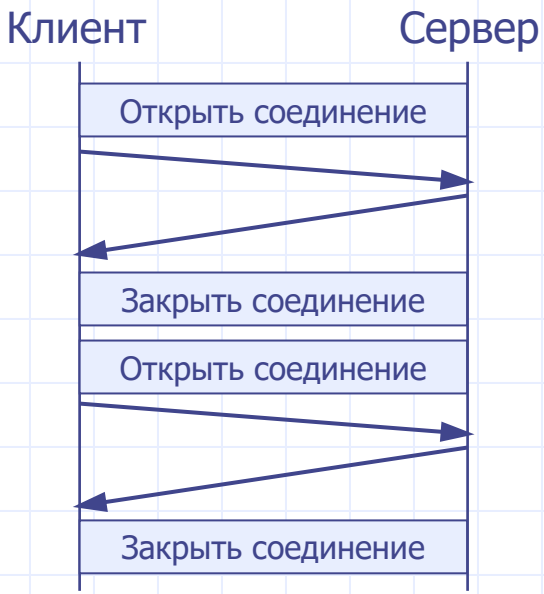
- Заголовки управления соединением
  - Connection
  - Keep-Alive
- Заголовки согласования типа содержимого
  - Accept
  - Accept-Encoding
- Заголовки свойств содержимого
  - Content-Type
  - Content-Encoding
  - Content-Length
- Заголовки условий
  - Last-Modified
  - ETag
  - If-Match
- Заголовки «куки»
  - Set-Cookie
  - Cookie
- Заголовок переадресации
  - Location
- Заголовки кэширования
  - Cache-Control
  - Expires
- Заголовки контекста запроса
  - Host
  - User-Agent
- Заголовки контекста ответа
  - Allow
  - Server
- Заголовки диапазона
  - Accept-Ranges
  - Range
- Заголовки аутентификации
  - WWW-Authenticate
  - Authorization
- Заголовки разрешений CORS
  - Origin
  - Access-Control-Allow-Origin
- Заголовки безопасности

HTTP-соединение



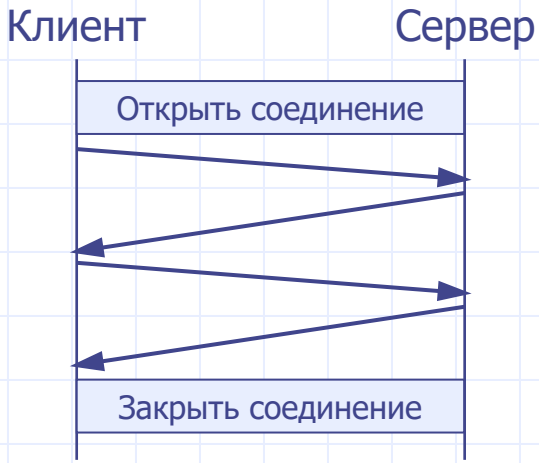
`Connection: close`

- По одному соединению передаётся только один запрос и один ответ.



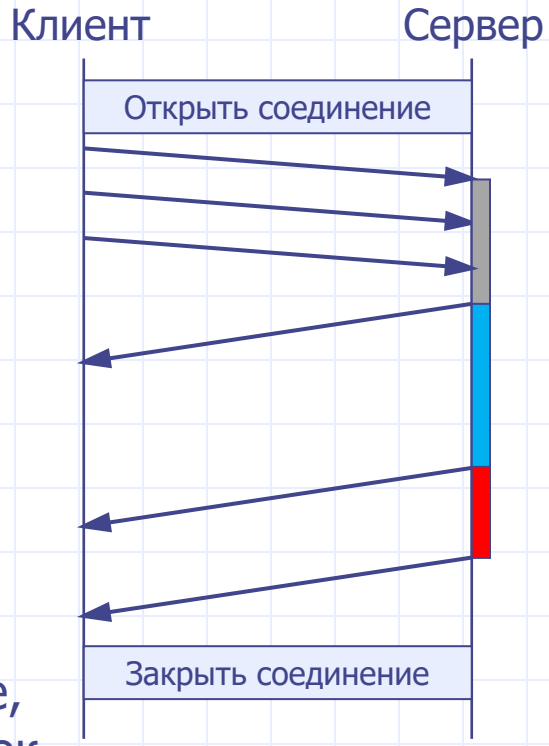
`Connection: keep-alive`

- По одному соединению может передаваться несколько запросов и ответов последовательно.
- В ответе обязательно указываются **Content-Length** и **Content-Type**.



Конвейер (англ. pipeline) запросов-ответов в режиме постоянно открытого соединения

- Отсылка новых запросов по одному соединению без ожидания ответов на предыдущие запросы (HTTP/1.1).
- Поддерживает только идемпотентные запросы (GET, HEAD, PUT, DELETE), повторное выполнение которых безопасно.
- Можно повысить производительность, упаковав несколько HTTP запросов в один TCP-пакет.
- В HTTP/1.1 отсутствуют средства сопоставления запросов и ответов, поэтому ответы обязаны приходить в том же порядке, что и отправленные запросы. Этот недостаток не позволяет полноценно использовать конвейерный режим запросов-ответов.



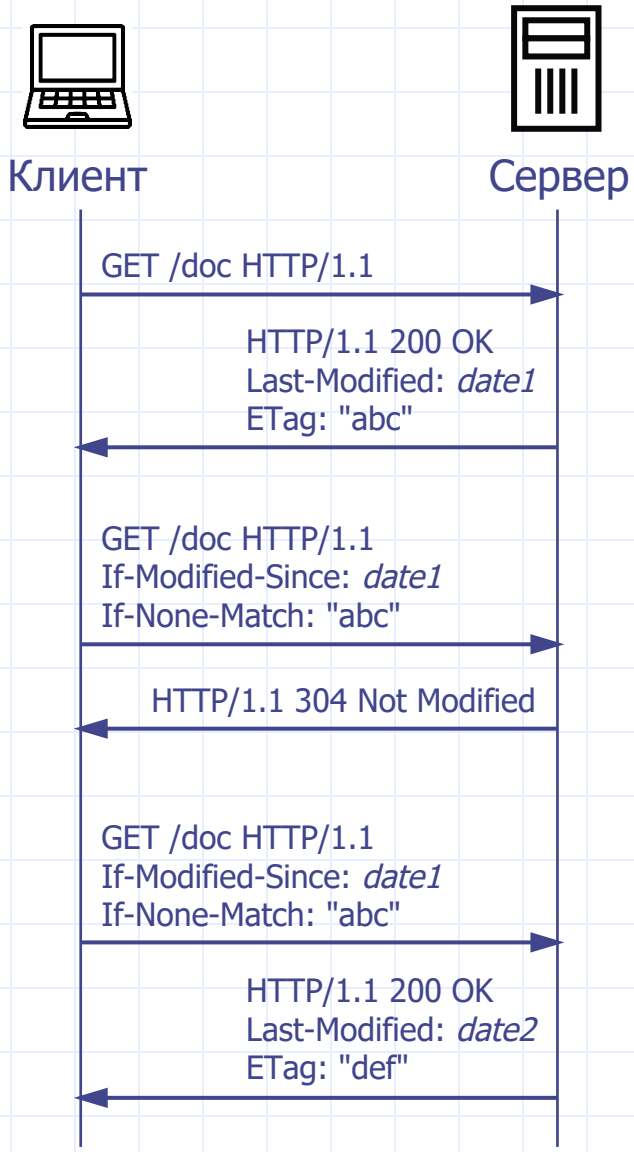
Составной HTTP-ответ, размер которого не известен в момент отправки заголовков

- Указывается служебный заголовок **Transfer-Encoding: chunked**.
- Ответ отправляется по частям (англ. chunks) с указанием размера каждой части. Конец ответа обозначается пустым блоком нулевой длины.

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

3\r\n
Per\r\n
9\r\n
 aspera\r\n\r\n
3\r\n
ad \r\n
5\r\n
astra\r\n
0\r\n
\r\n
```

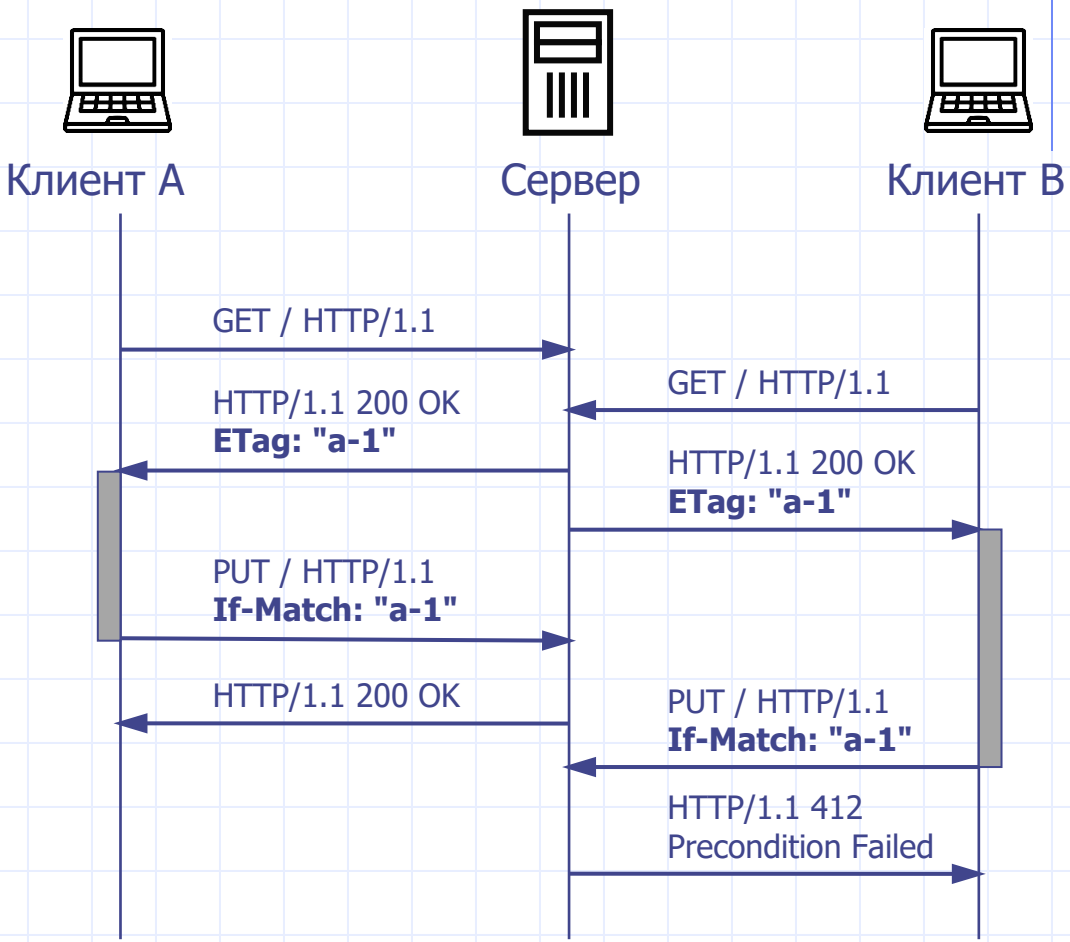
Использование заголовков условий HTTP и кэширование содержимого ответов

- Клиент запрашивает данные.
- Сервер возвращает данные с посчитанным хэш-кодом и датой последнего изменения.
- Клиент кеширует данные.
- При повторном запросе данных заголовки условий позволяют серверу узнать версию данных у клиента и избежать повторной передачи содержимого.



Использование заголовков условий HTTP для совместного редактирования данных

- Клиент А получает данные, изменяет их локально и отправляет серверу.
- Клиент В получает данные одновременно с клиентом А, изменяет их локально и отправляет серверу свои изменения позже, чем клиент А.
- Изменения клиента А применяются, а изменения клиента В отбрасываются с ошибкой.



Хранение состояния HTTP-сеанса на клиенте с помощью механизма «куки» (англ. cookie)

- HTTP cookie – это небольшой фрагмент данных, хранящий состояние для протокола HTTP. Отправляется сервером клиенту, который может сохранить эти данные и отсылать обратно с новыми запросами к данному серверу.
- Применение:
  - Управление сеансом – логины, корзины для виртуальных покупок;
  - Персонализация – пользовательские предпочтения, оформление;
  - Мониторинг – отслеживания поведения пользователя, таргетированная реклама.
- «Куки» бывают:
  - временные (сессионные) или постоянные (указан атрибут Expires или Max-Age);
  - «безопасные», требующие протокол HTTPS (указан атрибут Secure);
  - недоступные из JavaScript (указан атрибут HttpOnly).

Сервер:

```
HTTP/1.1 200 OK
Set-Cookie: theme=dark
Set-Cookie: id=12345; Expires=Mon, 12 Apr 2021 20:16:00 GMT; Secure; HttpOnly
```

Клиент:

```
GET /mailbox.html HTTP/1.1
Cookie: theme=dark; id=12345
```

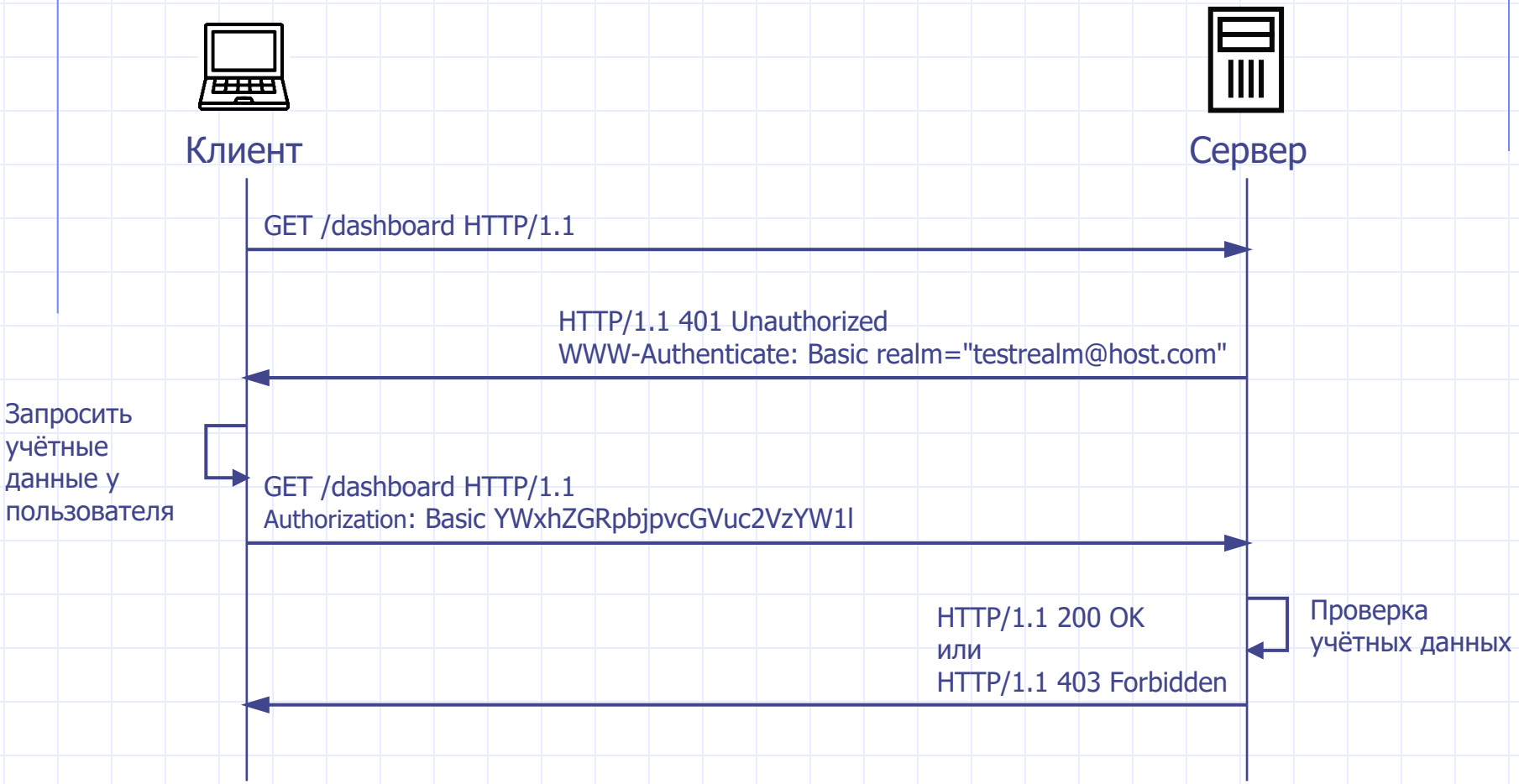
Аутентификация и авторизация в HTTP

- Аутентификация (англ. Authentication) – установление личности клиента.
- Авторизация (англ. Authorization) – проверка прав доступа клиента.
- Аутентификация в протоколе HTTP:
  - Осуществляется для каждого запроса отдельно.
  - Сервер просит аутентификацию для доступа к ресурсу с помощью заголовка ответа WWW-Authenticate.
  - Клиент выполняет аутентификацию и авторизацию с помощью заголовка Authorization.
  - Основные типы аутентификации в HTTP:
    - Basic – имя пользователя + пароль,
    - Digest – на основе дайджеста (хэша),
    - Bearer – на основе маркера безопасности.

```
WWW-Authenticate: <тип> realm="область"
```

```
Authorization: <тип> <учётные данные>
```

Общая схема аутентификации в HTTP



Basic-аутентификация в HTTP

- Имя типа: Basic
- Имя пользователя и пароль объединяются в одну строку через двоеточие и кодируются в Base64.
- Пример:
  - Имя: «username»
  - Пароль: «password»
  - Строка для кодирования: «username:password»
  - Запрос:

```
GET / HTTP1.1
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

```
Устаревший формат URL:
http://user:password@host:port/...
```

JavaScript:

btoa(s) – кодировать строку в Base64

atob(x) – восстановить строку из Base64

Кодирование Base64

- Кодирование двоичных данных при помощи 64-х символов ASCII.
- Алфавит Base64: A-Z, a-z, 0-9 (62 знака), + и / (или * и -).
- Поток битов разбивается на шестёрки.
- Каждая шестёрка битов кодируется значением из алфавита.
- Количество битов в потоке байтов не всегда кратно шести. В этом случае поток дополняется двумя или четырьмя нулевыми битами до кратности, на что указывает один или два знака «=» в конце закодированной строки.

Данные:	M								a								n						
	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
Base64(Man):	T						W						F				u							
	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	0						
Base64(Ma):	T						W						E				=							
	0	1	0	0	1	1	0	1	0	0	0	0												
Base64(M):	T						Q						=				=							

Digest-аутентификация в HTTP

- Имя типа: Digest
- Позволяет избежать передачи имени пользователя и пароля в открытом виде. Передаётся хэш-код пароля, имени пользователя, текущего времени и других параметров, полученных от сервера.
- Заголовок WWW-Authenticate содержит:
  - realm – защищённая область сайта;
  - qop – степень защиты: обычная (auth) или с проверкой целостности (auth-int);
  - algorithm – алгоритм хэширования (MD5, SHA-256);
  - nonce – однократно используемая строка в формате Base64
    - `nonce="<точное-время> MD5(<точное-время>:ETag:<секрет>)"`
- Заголовок Authorization содержит:
  - username – имя или хэш-код имени пользователя;
  - nc – счётчик повторного использования nonce;
  - cnonce – однократно используемая строка от клиента;
  - response – хэш-код, включающий: имя пользователя, пароль, значение nonce, полученное от сервера, а также значения realm и URI
    - `response="MD5(HA1:nonce:nc:cnonce:qop:HA2)"`, где
      `HA1 = MD5(username:realm:password),`
      `HA2 = MD5(HTTP-method:URI)`

Digest-аутентификация в HTTP (продолжение)

Клиент

Сервер

GET /dir/index.html HTTP/1.1

HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="testrealm@host.com",
qop="auth", algorithm="MD5"
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093"

GET /dir/index.html HTTP/1.1
Authorization: Digest username="Mufasa",
realm="testrealm@host.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
uri="/dir/index.html",
qop=auth,
nc=00000001,
cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1"

Запросить учётные данные у пользователя

HTTP/1.1 200 OK
или
HTTP/1.1 403 Forbidden

Проверка учётных данных

Bearer-аутентификация в HTTP (на основе маркера безопасности)

- Имя типа: Bearer (носитель, предъявитель)
- Используется для избегания частых аутентификаций.
- Ориентирован на использование шифрованных соединений.
- Ориентирован на авторизацию через стороннюю службу аутентификации.
- Обеспечивает аутентификацию и авторизацию без раскрытия учётных данных пользователя (имени, пароля).
- Маркер безопасности (англ. security token):
  - Маркер часто представляет собой зашифрованный и/или подписанный идентификатор пользователя со списком прав доступа.
  - Полученный маркер безопасности может быть использован для осуществления доступа к ресурсам без необходимости аутентификации в течение определённого периода времени.
  - Время действия маркера ограничено. После его истечения необходимо провести повторную аутентификацию.
  - Существуют механизмы обновления маркера без повторной аутентификации (на основе другого маркера безопасности с бóльшим временем действия).

Маркер JWT – JSON Web Token

```
// Заголовок (header)
Base64({
  "alg": "HS256", // алгоритм
  "typ": "JWT" // тип маркера
})
```

```
// Тело (payload)
Base64({
  "sub": "user-1", // id пользователя
  "name": "User",
  "iat": 1618865746 // время выдачи маркера
})
```

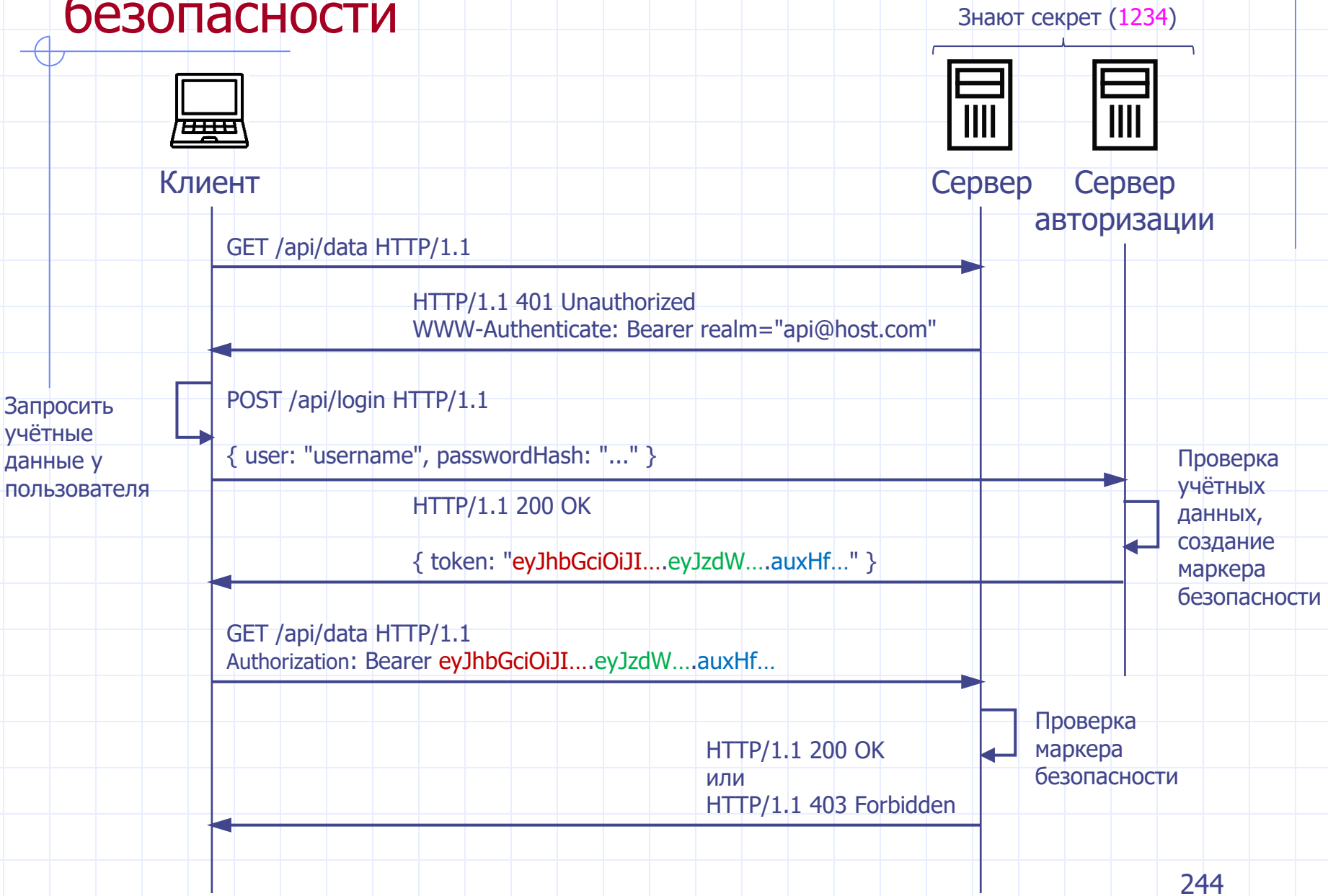
```
Authorization: Bearer eyJhbG
ciOiJIUzI1NiIsInR5cCI6IkpXVC
J9.eyJzdWIiOiJ1c2VyLTEiLCJuY
W1lIjoiVXNlciIsImlhdCI6MTYxO
Dg2NTc0Nn0.auxHfX5VsqmiGozrU
kkYCVygFnmQe5DwrZZEIRzqL40
```

```
// Цифровая подпись (signature)
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  1234 // секрет сервера
)
```

https://jwt.io/

HMAC – hash-based message authentication code – код аутентификации, основанный на хэше

Аутентификация на основе маркера безопасности



HTTPS – HTTP Secure. TLS/SSL

- Протокол HTTP не предоставляет средств обеспечения безопасности.
- Для установки и поддержания защищённого соединения используется протокол защиты транспортного уровня TLS (англ. Transport Layer Security), который основан на ныне устаревшем протоколе SSL (англ. Secure Socket Layer).
- TLS предоставляет три основных функции, которые обеспечивают безопасный и защищённый обмен данными:
  - Аутентификацию – клиент проверяет, что установил соединение с истинным сервером.
  - Шифрование – защита от просмотра передаваемых данных третьей стороной.
  - Целостность – защита передаваемых данных от искажения и подмены.

Порт HTTPS: 443

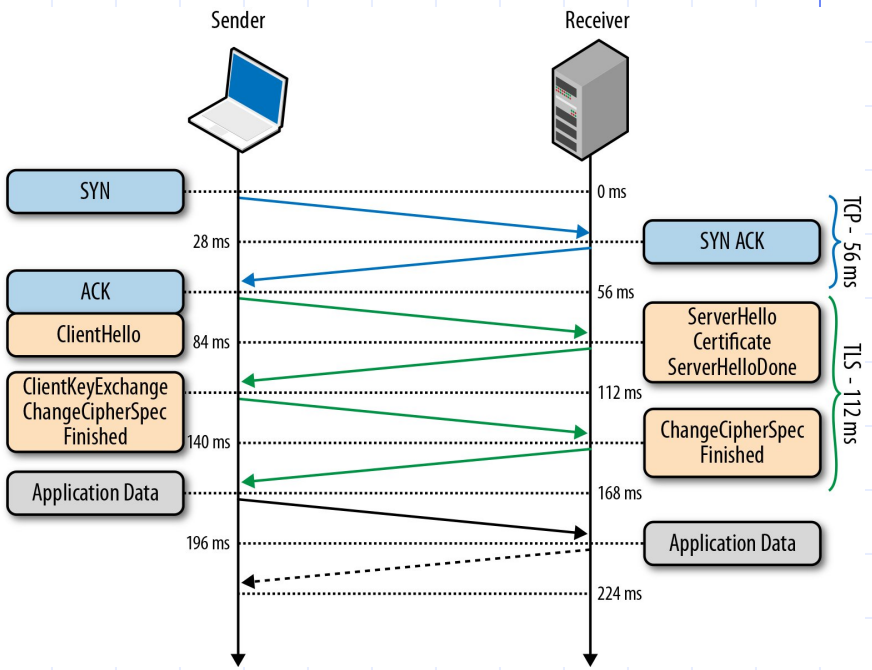
Азы криптографии

- Симметричное шифрование:
  - один ключ для шифрования и дешифрования;
  - две стороны используют общий ключ.
- Асимметричное шифрование:
  - открытый ключ для шифрования;
  - закрытый ключ для дешифрования;
  - отправитель использует открытый ключ, а получатель – закрытый ключ (зашифровать может кто угодно, расшифровать может только один).
- Цифровая подпись:
  - открытый ключ для дешифрования;
  - закрытый ключ для шифрования;
  - подписать может только один, а проверить подпись может кто угодно.

Сертификаты – основа работы TLS

- Сертификат – подписанный цифровой подписью файл, который содержит наборы ключей для симметричного и асимметричного шифрования.
- Сертификаты может выдавать и подписывать только ограниченный круг доверенных организаций (Setigo, GlobalSign, Comodo, ISRG)
  - Let's Encrypt от ISRG: https://letsencrypt.org/ru/about/
  - Self-signed сертификаты (подписанные самим собой) можно сгенерировать автоматически; не используются публично.
- Сертификат состоит из двух частей:
  - Публичной – передаётся клиенту при подключении к серверу; проверяется у доверенной компании, выдавшей и подписавшей сертификат (центр сертификации, англ. CA – Certificate Authority).
  - Приватной – секретный ключ, который «знает» только сервер; служит для идентификации сервера.
- Публичный и приватный ключи сертификата используются для реализации асимметричного шифрования.
- При установке TLS-соединения используется асимметричное шифрование, а при передаче данных – симметричное шифрование.

Соединение TLS



1. Клиент устанавливает TCP-соединение с сервером.
2. Клиент посылает на сервер спецификацию в виде обычного текста (версию протокола, которую он хочет использовать, поддерживаемые методы шифрования).
3. Сервер утверждает версию используемого протокола, выбирает способ шифрования, и отправляет ответ клиенту вместе со своим публичным сертификатом.
4. Клиент проверяет присланный сертификат и инициирует безопасный обмен ключами, используя асимметричное шифрование (открытый ключ сервера). В результате обмена клиент и сервер получают общий ключ для симметричного шифрования.
5. Сервер обрабатывает присланное клиентом сообщение, расшифровывает его своим закрытым ключом, проверяет подпись, и отправляет клиенту заключительное сообщение, зашифрованное симметричным алгоритмом.
6. Клиент расшифровывает полученное сообщение, проверяет подпись, и если всё хорошо, то соединение считается установленным и начинается обмен данными приложений.

Виды веб-приложений

Классическое веб-приложение

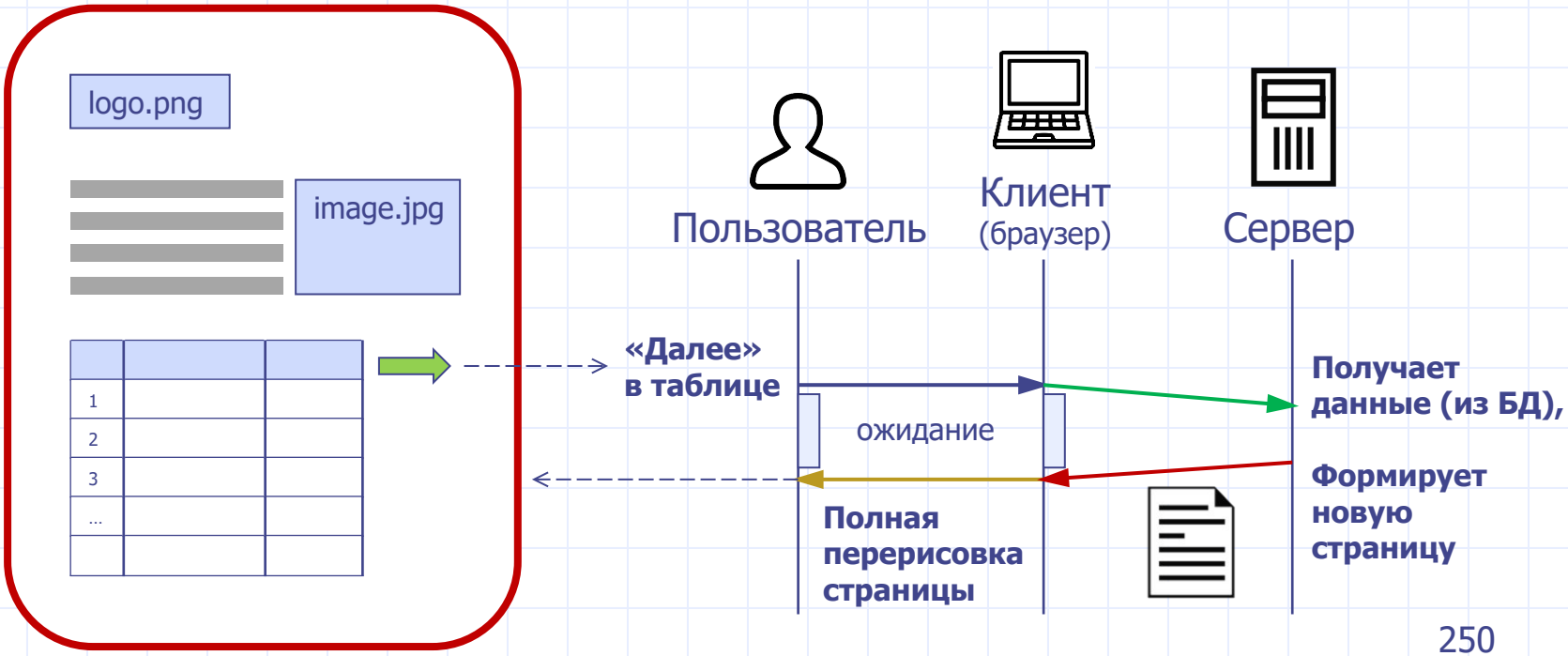
- Программная логика целиком реализуется на сервере с помощью любого языка программирования.
- Переход по ссылке внутри сайта приводит к HTTP-запросу и загрузке новой HTML-страницы с сервера.
- Данные приходят вместе с HTML-страницей.

Одностраничное веб-приложение

- Программная логика целиком реализуется на клиенте с помощью языка JavaScript.
- Переход по ссылке внутри сайта приводит к видоизменению структуры HTML-страницы без необходимости выполнять HTTP-запрос к серверу.
- Данные приходят отдельно (обычно в формате JSON).

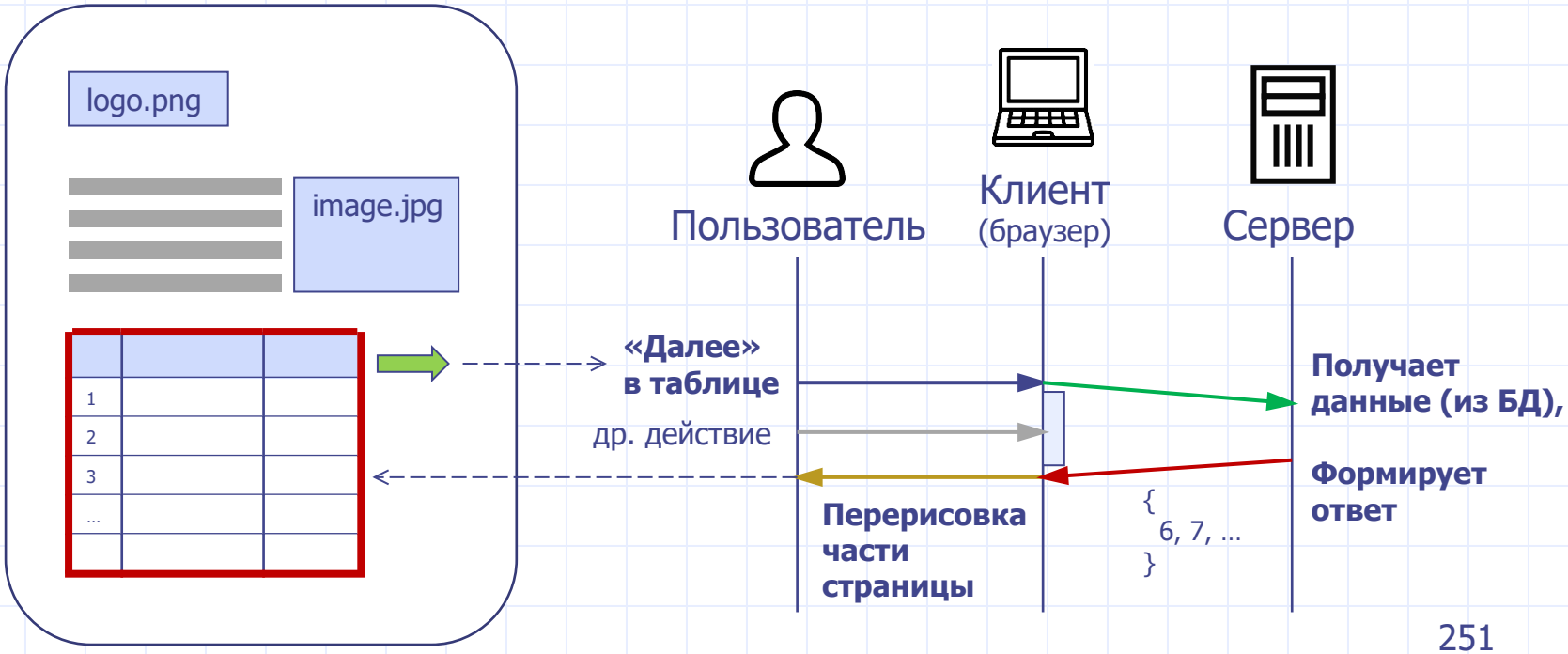
Классическое веб-приложение

- Действие пользователя приводит к отправке запроса на сервер.
- Сервер формирует и отправляет новую страницу для отображения.
- Клиент (браузер) перерисовывает страницу целиком.
- Интерфейс пользователя блокируется до загрузки новой страницы.



Одностраничное веб-приложение

- Действие пользователя приводит к отправке запроса на сервер.
- Сервер передаёт в ответе только данные.
- Код на клиенте обновляет содержимое страницы, клиент (браузер) перерисовывает только часть страницы.
- Нет блокировки интерфейса пользователя.

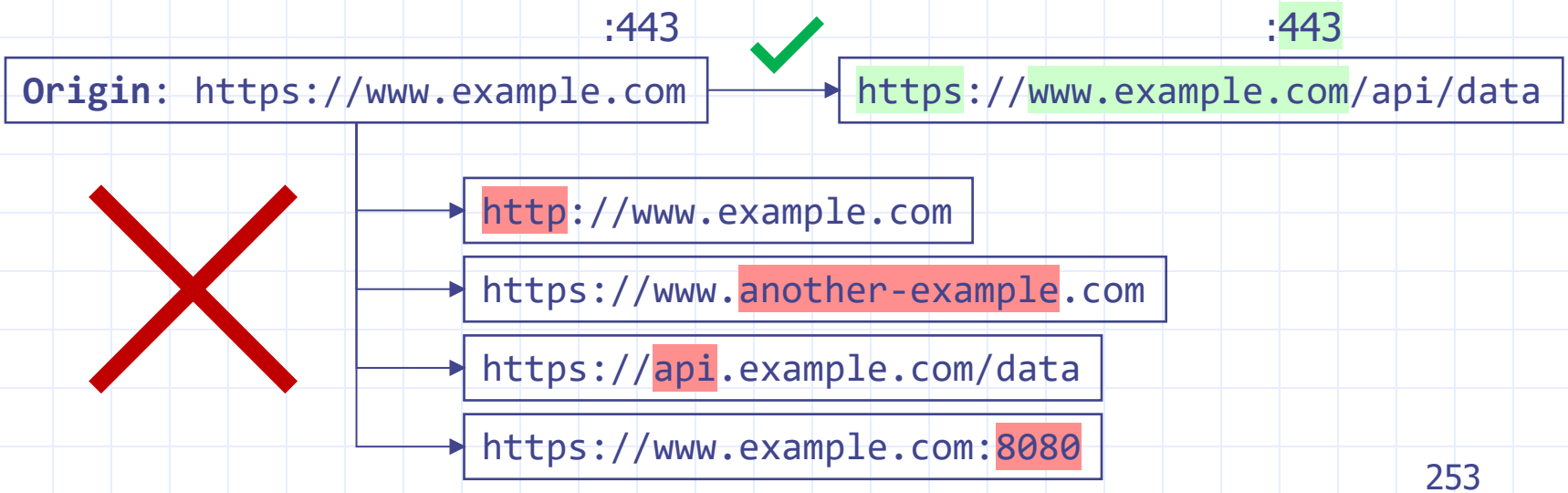


Асинхронные запросы (англ. Asynchronous JavaScript and XML – AJAX)

- Технология для взаимодействия с сервером без полной перезагрузки веб-страницы.
  - По наступлению определённого события (таймер, действие пользователя) клиент (браузер) посылает запрос на сервер в фоне, не блокируя пользовательский интерфейс.
  - Сервер в ответе передаёт необходимые данные для обновления содержимого страницы в текстовом формате или HTML, XML, JSON, в виде кода JavaScript.
  - Код на клиенте обновляет содержимое (часть) страницы.
- Сокращает трафик и время ожидания пользователя.
- Примеры:
  - «живой» поиск с автоматическим дополнением запроса,
  - веб-чат в социальной сети.

Правило ограничения домена (принцип одного источника, англ. Same-origin policy – SOP)

- Политика безопасности, по которой код на странице может получить доступ только к ресурсам из того же источника.
- Заголовок **Origin** устанавливается браузером (недоступен из кода).
- Источники считается одинаковыми, если у них совпадают протокол, сервер (домен и поддомены) и порт.
- Защищает веб-приложения от внедрения вредоносного кода и кражи cookies.



Совместное использование ресурсов между разными источниками (Cross-origin resource sharing – CORS)

- Позволяет смягчить политику одинакового источника.
- Сервер устанавливает, из каких источников можно получить доступ к ресурсу, о чём сообщает в заголовке ответа Access-Control-Allow-Origin.

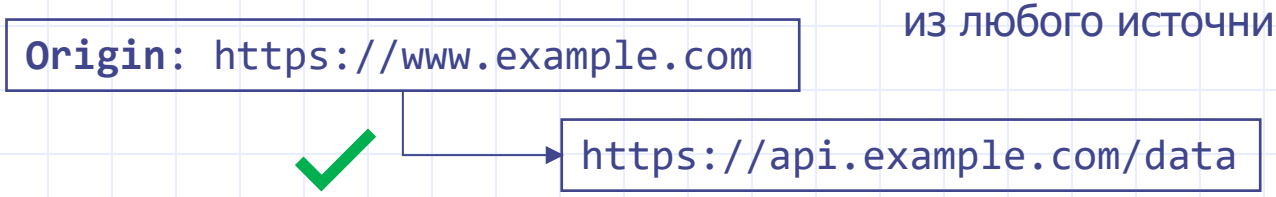
Запрос:

```
GET /data HTTP/1.1
Host: api.example.com
...
Origin: https://www.example.com
```

Ответ:

```
HTTP/1.1 200 OK
Server: nginx
...
Access-Control-Allow-Origin: *
```

Разрешить доступ
из любого источника



Строгое ограничение:

```
Access-Control-Allow-Origin: https://wwww.example.com
```

Предварительный запрос в CORS

- Не нужен для «простых» запросов (методы GET, HEAD и POST без дополнительных заголовков).
- Клиент посылает предварительный (англ. preflight) запрос OPTIONS, чтобы узнать, доступен ли ресурс для текущего источника.
- Заголовки предварительного запроса, помимо Origin:
  - Access-Control-Request-Method,
  - Access-Control-Request-Headers.
- Заголовки ответа:
  - Access-Control-Allow-Origin – источник,
  - Access-Control-Allow-Methods – методы,
  - Access-Control-Allow-Headers – заголовки,
  - Access-Control-Max-Age – сколько секунд актуальны результаты предварительного запроса (Methods, Headers).

Пример запросов, использующих CORS

Клиент

Сервер

OPTIONS /data HTTP/1.1
Origin: https://example.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: Content-Type, Content-Length

HTTP/1.1 204 No Content
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS
Access-Control-Allow-Headers: X-Custom, Content-Type, Content-Length
Access-Control-Max-Age: 86400

PUT /data HTTP/1.1
Origin: https://example.com
Content-Type: application/json; charset=UTF-8
Content-Length: 123

{ ... }

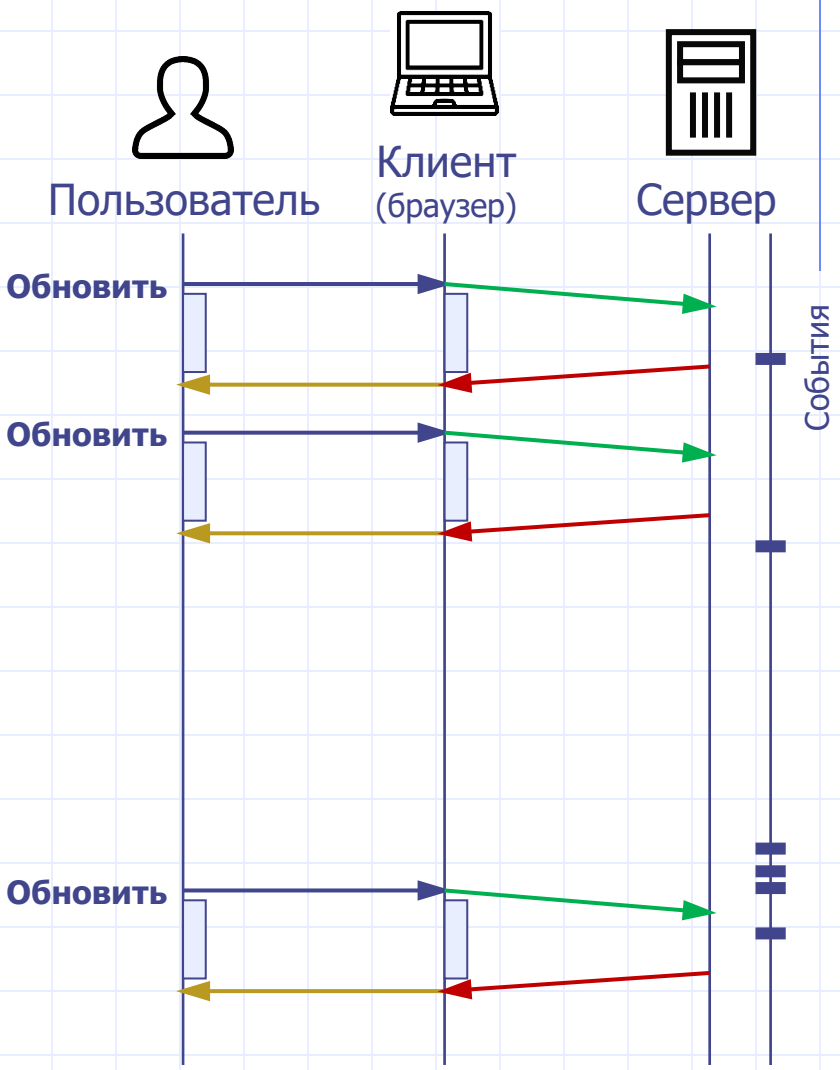
HTTP/1.1 202 Accepted
Access-Control-Allow-Origin: https://example.com

Передача данных по HTTP в режиме, приближенном к режиму реального времени

- Постановка задачи:
  - Пользователь хочет отслеживать изменения на сервере.
  - Пример: клиент, взаимодействующий со службой погоды и отображающий показания температуры, влажности, давления и пр. для заданного населённого пункта.
- Подходы к реализации:
  - Pull-модели (клиент «вытаскивает» данные с сервера):
    - Ручное обновление
    - Опрос данных (AJAX polling)
  - Push-модели (сервер «проталкивает» данные клиенту):
    - Долгий опрос данных (Comet long polling)
    - Передача потока данных (Comet streaming)
  - Двусторонний обмен сообщениями
    - Веб-гнёзда (Web-Sockets)

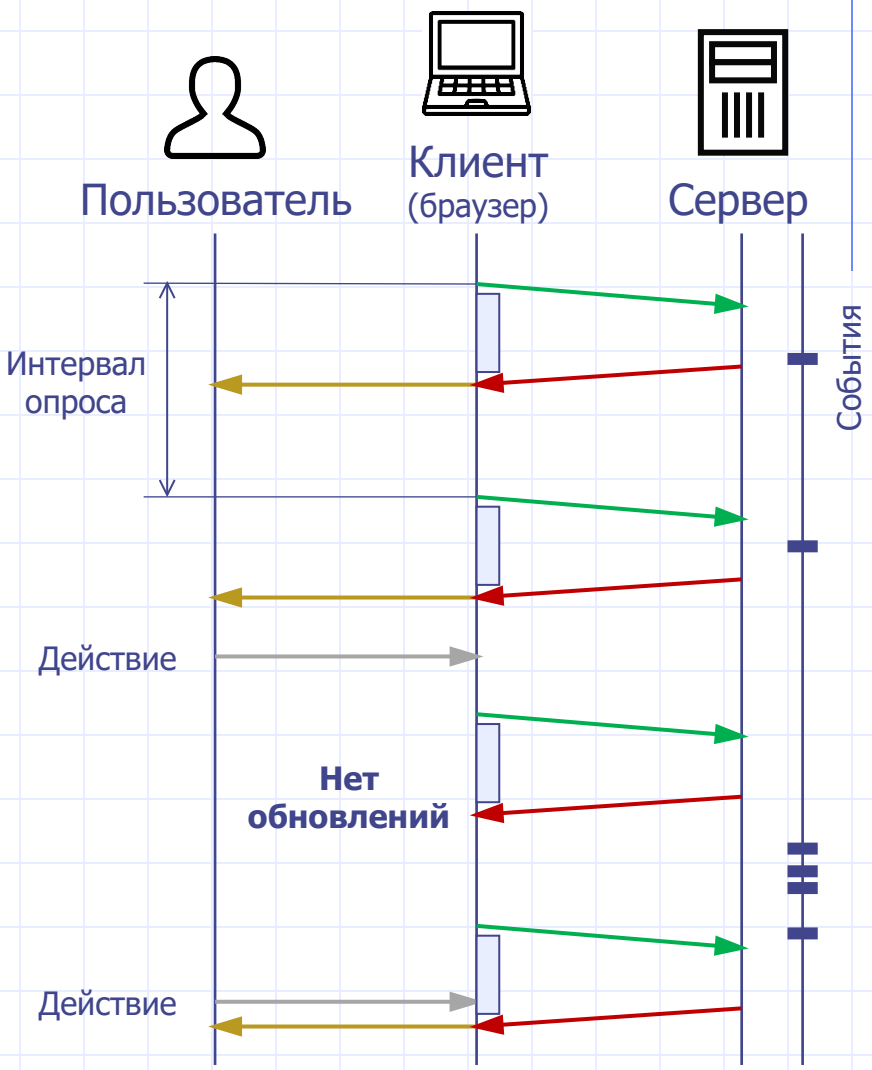
Ручное обновление

- Недостатки:
  - Низкая частота обновлений, реальный масштаб времени отсутствует.
  - Высокий трафик (полная перезагрузка страницы).
  - Высокая нагрузка на сервер и сеть.



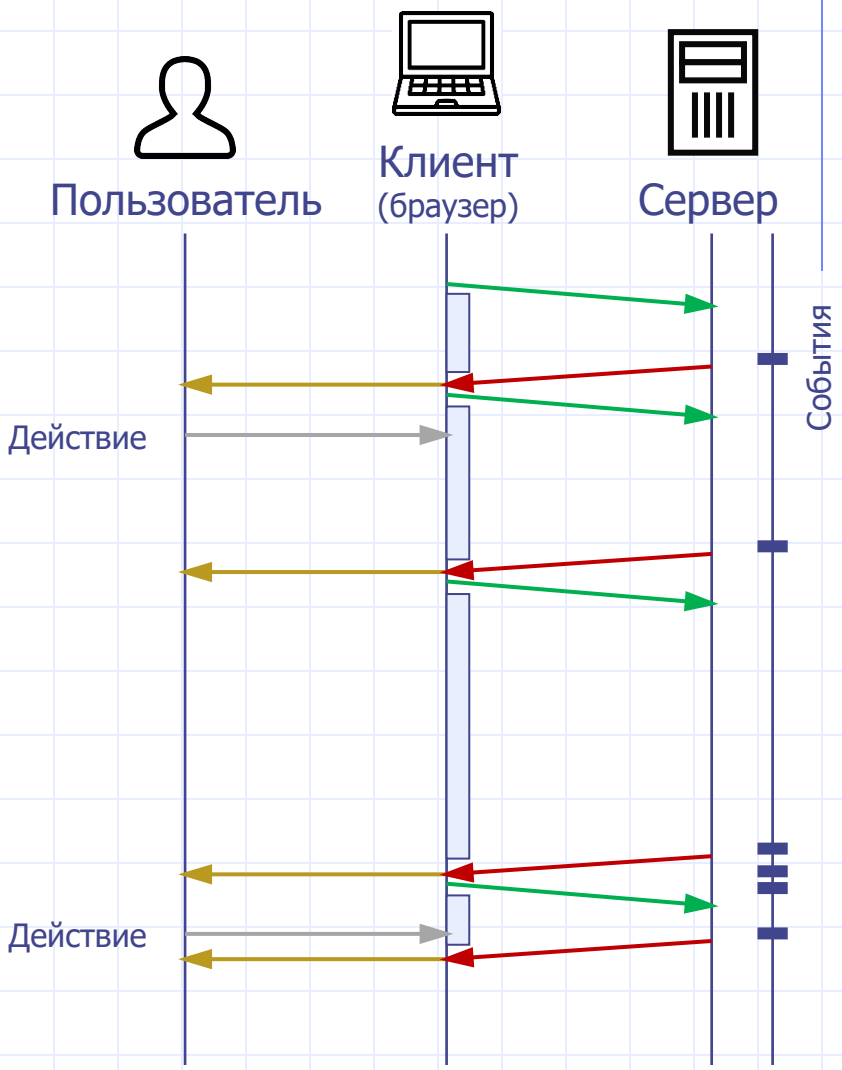
Опрос данных (AJAX Polling)

- Недостатки:
  - Низкая частота обновлений, реальный масштаб времени отсутствует.
  - Высокий трафик
    - частые запросы;
    - передача заголовков запроса и ответа.
  - Высокая нагрузка на сервер и сеть.
- Достоинства:
  - Отсутствует блокировка пользовательского интерфейса.



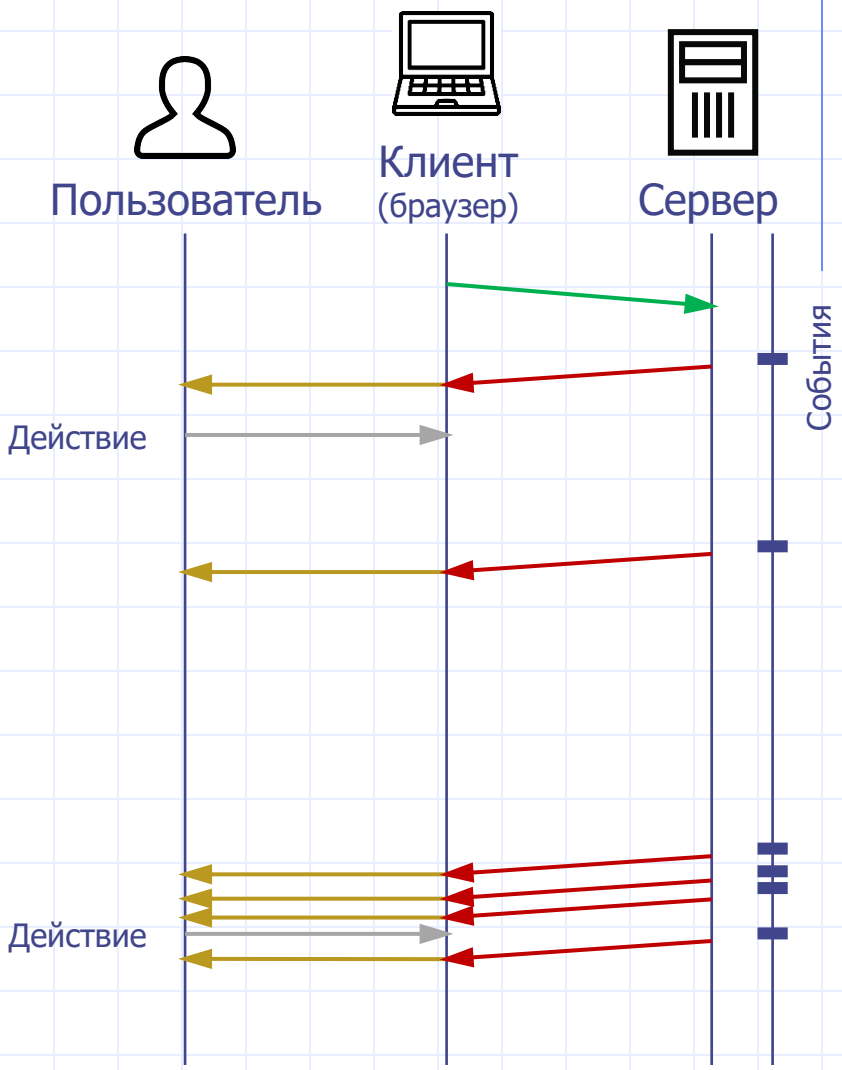
Долгий опрос данных (Comet Long Polling)

- Недостатки:
  - Средняя частота обновлений, близко к реальному масштабу времени.
  - Средний трафик
    - передача заголовков запроса и ответа.
  - Средняя нагрузка на сеть.
- Достоинства:
  - Отсутствует блокировка пользовательского интерфейса.
  - Низкая задержка для нечастых событий.



Потоковая передача данных (Comet Streaming)

- Недостатки:
  - Может блокироваться на промежуточных узлах.
- Достоинства:
  - Высокая частота обновлений, низкая задержка, реальный масштаб времени.
  - Низкий трафик
    - нет передачи заголовков запроса и ответа.
  - Низкая нагрузка на сеть.



Веб-гнёзда (англ. WebSocket) – двусторонний обмен сообщениями

- Цель: полнодуплексная связь между клиентом и сервером, чтобы не только клиент мог отслеживать изменения на сервере, но и сервер мог отслеживать изменения на клиенте.
  - TCP не подходит: проблемы безопасности, отсутствует CORS, порты блокируются сетевыми экранами.
  - HTTP не подходит: диалоговый режим, не гарантирована поддержка соединения, не гарантирован порядок сообщений.
- Создаётся HTTP-запросом со специальными заголовками, который переводит HTTP-соединение в режим Веб-гнезда.
- Клиент и сервер могут посылать данные друг другу асинхронно.
- Ориентированы на передачу отдельных сообщений (а не потока данных, как протокол TCP).
- Обеспечивает фрагментацию с помощью кадров (англ. frames) для передачи сообщений с неизвестной заранее длинной и мультиплексирования канала связи.
- Сообщения от клиента серверу маскируются операцией XOR со случайным ключом для исключения «отравления кэша» на прокси-серверах. Сообщения от сервера клиенту не маскируются.

Установка соединения для протокола WebSocket

- Запрос: ws://example.com/api (wss:// для использования TLS)

```
GET /api HTTP/1.1
Host: example.com
Origin: http://example.com
Connection: keep-alive, Upgrade
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: GqR/lpAbebWlbv7cj3Hcsg==
Sec-WebSocket-Extensions: permessage-deflate
```

- Ответ:

```
HTTP/1.1 101 Switching Protocols
Server: nginx/1.18.0
Date: Mon, 26 Apr 2021 15:39:20 GMT
Connection: upgrade
Upgrade: websocket
Sec-WebSocket-Accept: XbCVqACk+DvSAnfIuMPJlew1lgI=
```

Проблемы HTTP/1.1

- Загрузка типичной веб-страницы в среднем требует совершить ~100 HTTP-запросов и передать ~2 МБ данных.
- Рост пропускной способности сетей не сопровождается уменьшением задержки при передаче.
- Производительность падает из-за высокой задержки при выполнении HTTP-запросов. Чем больше запросов, тем больше общая задержка и время загрузки страницы.
- В HTTP/1.1 ограничено число параллельных соединений одного клиента с одним сервером, что ограничивает скорость загрузки страницы.
- HTTP/1.1 работает поверх TCP, но не использует его возможности в полной мере. В частности, заголовок Content-Length делает невозможным прерывание передачи данных по HTTP-соединению без разрыва и повторной установки TCP-соединения.
- Конвейеризация запросов работает плохо: длительный запрос в начале очереди блокирует последующие короткие запросы.

Некоторые способы преодоления проблем HTTP/1.1

- Создание спрайтов – объединение множества мелких картинок в одну большую для передачи одним запросом. На клиенте спрайт «нарезается» для отображения маленьких картинок с помощью JavaScript-кода или CSS-стилей.
- Встраивание небольших файлов напрямую в веб-страницу (Data URL). Изображения можно встроить в CSS-стили. Нетекстовые данные кодируются в Base64.
- Объединение множества файлов JavaScript в один большой файл (bundle).
- Распределение изображений между несколькими серверами в разных доменах для преодоления ограничения на количество соединений одного клиента. Также позволяет избегать передачи cookie, которые бывают объёмными.

HTTP/2

- Совместим с HTTP/1.1 по методам, кодам ответа, формату идентификатора ресурса (URI) и большинству заголовков.
- HTTP/2 – бинарный протокол, разбор запросов и ответов ускоряется.
- Бинарный формат, ориентированный на передачу сообщений, состоящих из кадров, в рамках потока данных.
- Сжатие заголовков методом HPACK – ускоряет серию запросов к одному серверу.
- Мультиплексирование потоков данных в рамках одного соединения – вместо конвейеризации запросов.
- Приоритеты и зависимости между потоками данных.
- Возможность прервать передачу данных без разрыва соединения.
- Посылка от сервера (server push) – сервер может самостоятельно отправить клиенту некоторые данные без запроса со стороны клиента.
- Шифрование TLS 1.2+ не обязательно по стандарту, но требуется всеми реализациями.

Согласование протокола (HTTP/1.1, HTTP/2 или более поздняя версия)

- Если клиент не знает, поддерживает ли сервер HTTP/2, он открывает соединение HTTP/1.1 и пытается обновить его до HTTP/2.
- Если сервер не поддерживает новый протокол, то он отвечает клиенту, используя HTTP/1.1.
- Если сервер поддерживает новый протокол, то он сообщает об этом клиенту ответом с кодом 101 и переходит на протокол HTTP/2 сразу после пустой строки, которой заканчивается ответ.

Запрос:

```
GET /index.html HTTP/1.1
Host: server.example.com
Connection: Upgrade
Upgrade: HTTP/2.0
```

Ответ (HTTP/2 не поддерживается):

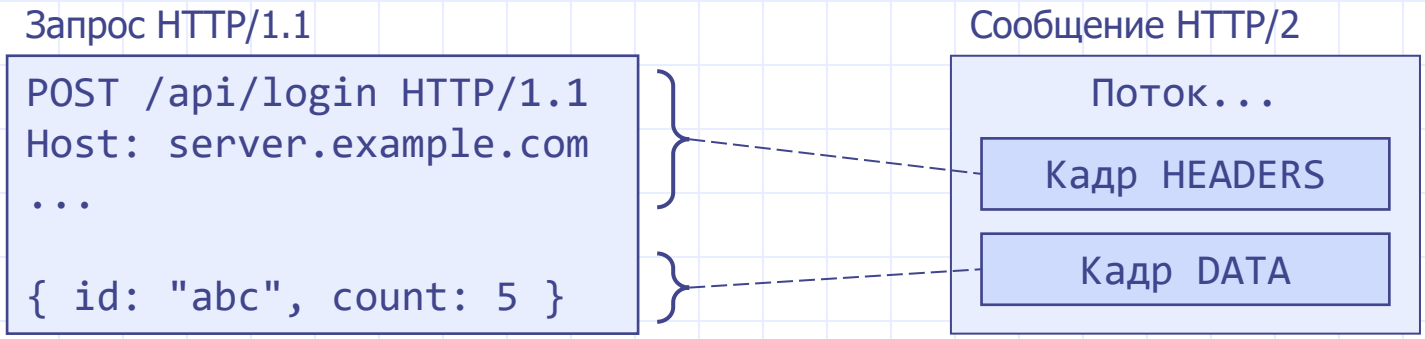
```
HTTP/1.1 200 OK
Content-length: 123
Content-type: text/html
...
```

Ответ (HTTP/2 поддерживается):

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: HTTP/2.0

[Кадр HTTP/2]
```

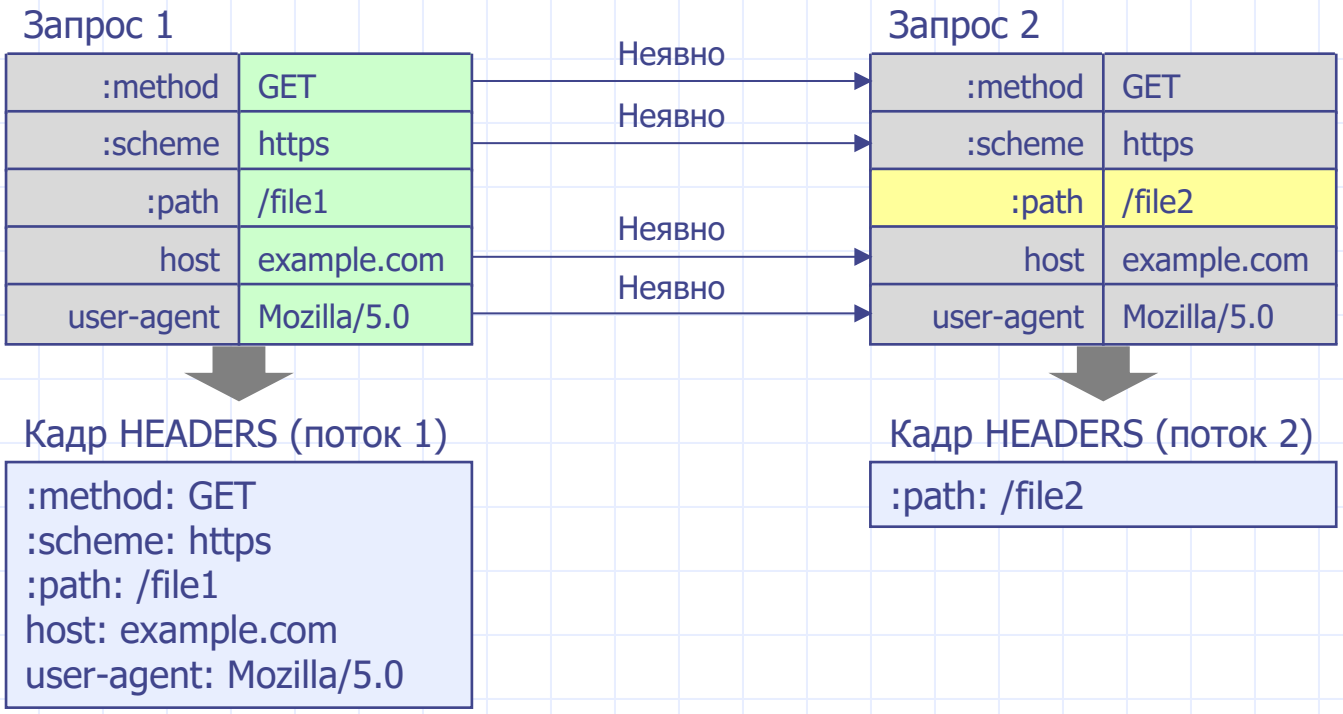
Потоки, сообщения и кадры в HTTP/2

- Поток – двунаправленный поток байтов в установленном соединении, который может нести одно или несколько сообщений.
- Сообщение – полная последовательность кадров, которая соответствует логическому запросу или ответу.
- Кадр – наименьшая единица связи в HTTP/2, которая содержит тип, длину, флаги, идентификатор потока и полезную нагрузку.
- Типы кадров:
  - HEADERS – соответствует заголовкам запроса HTTP/1.1,
  - DATA – соответствует телу запроса HTTP/1.1,
  - PRIORITY – задаёт приоритет потока,
  - RST_STREAM – позволяет прервать поток, и др.



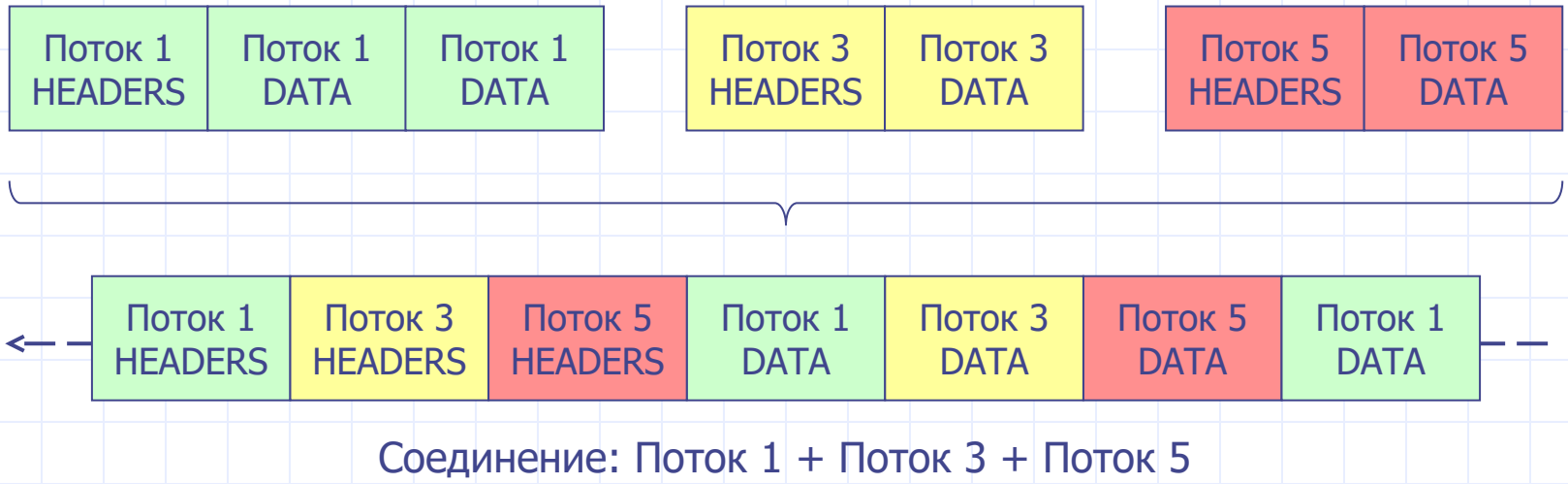
Сжатие заголовков в HTTP/2 методом HPACK

- В HTTP/2 все заголовки пишутся строчными буквами.
- Поля заголовков кодируются с помощью статических кодов Хаффмана.
- Клиент и сервер должны поддерживать индексированный список переданных ранее заголовков, который позволяет избежать повторной передачи одинаковых значений для разных запросов.
- Первый заголовок с HTTP-методом разбивается на несколько псевдо-заголовков (:method, :scheme, :path) для улучшения сжатия.



Мультиплексирование потоков в HTTP/2

- Передача нескольких потоков в рамках одного TCP-соединения.
- Позволяет чередовать несколько запросов/ответов и передавать их параллельно.
- Решает проблему блокировки очереди.



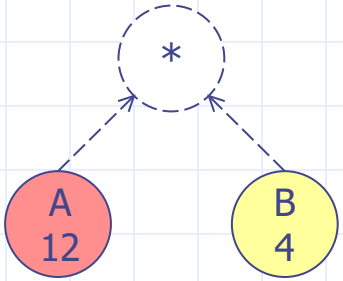
Кадр RST_STREAM закрывает поток без закрытия соединения.

Приоритетность потоков в HTTP/2

- Порядок чередования и доставки кадров клиентом/сервером влияет на производительность.
  - Каждый поток может иметь вес от 1 до 256.
  - Каждый поток может зависеть от другого потока.

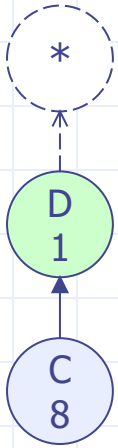
  } Дерево приоритетов

- Вес описывает «важность» потоков с точки зрения клиента и может служить указанием серверу, в какой пропорции выделять ресурсы на их обработку.

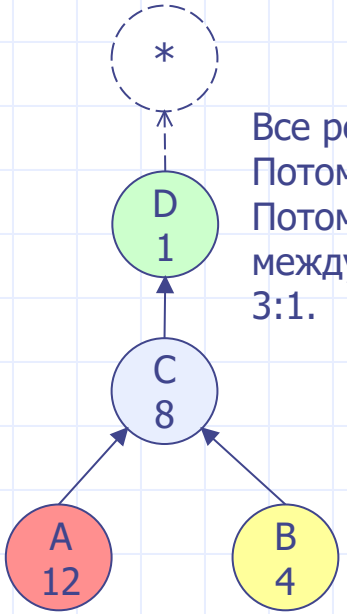


Разделить ресурсы между A и B в пропорции 3:1.

12+4=16
A: 12/16 = 3/4
B: 4/16 = 1/4



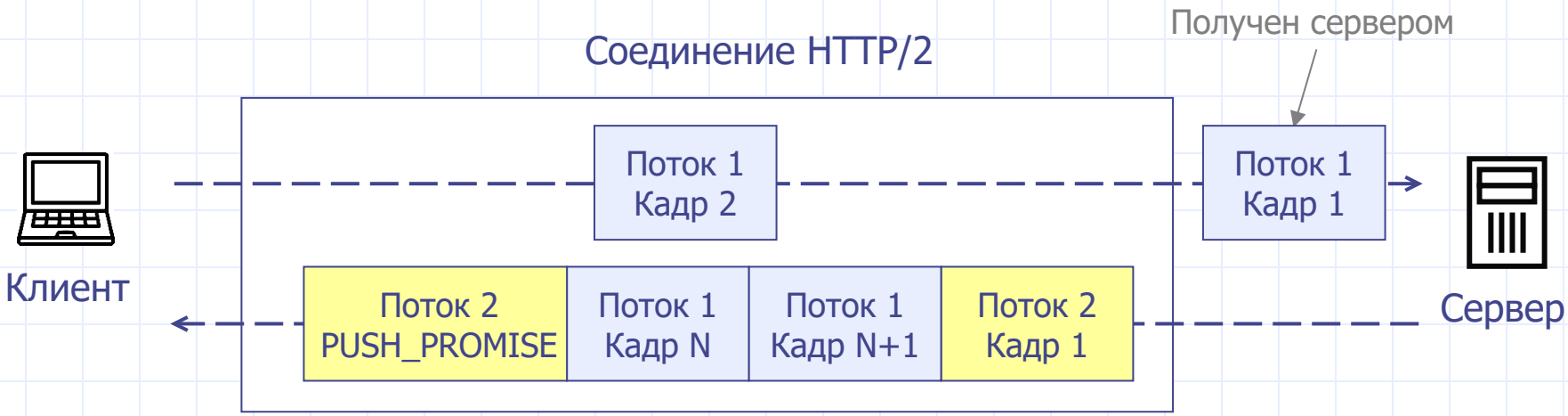
Все ресурсы для D.
Потом все ресурсы для C.



Все ресурсы для D.
Потом все ресурсы для C.
Потом разделить ресурсы между A и B в пропорции 3:1.

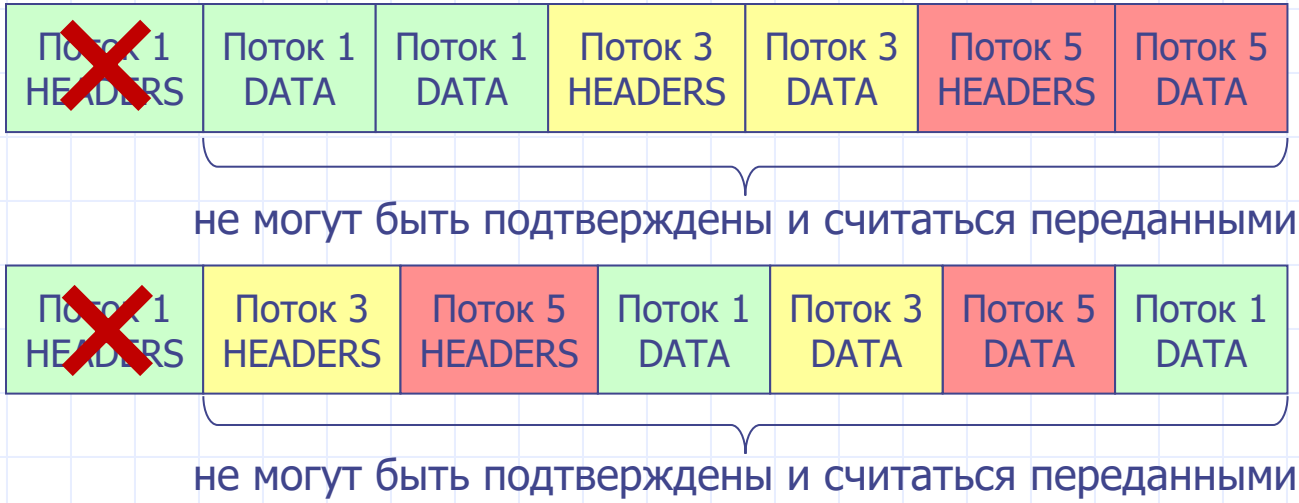
Посылка от сервера (server push) в HTTP/2

- На запрос клиента, помимо основного ответа, сервер может отправить дополнительные ресурсы, предвосхищая следующие запросы клиента. Эти ресурсы могут кэшироваться на стороне клиента для снижения количества запросов к серверу («посылка в кэш»).
- Для посылки в кэш клиента сервер перед кадром основного ответа посылает клиенту кадр PUSH_PROMISE, резервируя идентификатор потока для передачи дополнительных данных.
- Если клиенту не нужны дополнительные данные (уже в кэше), он может закрыть созданный сервером поток кадром RST_STREAM.



Проблемы HTTP/2

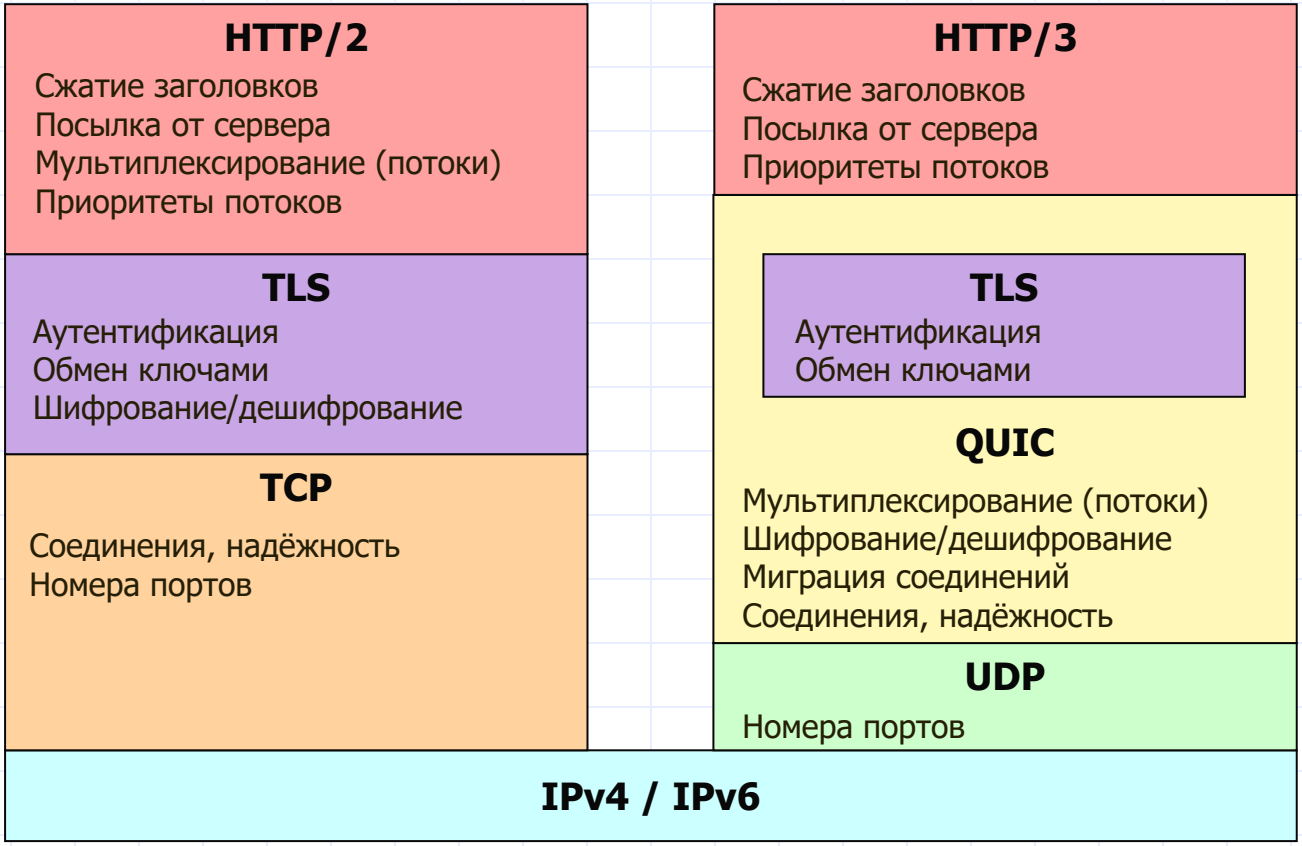
- HTTP/2 работает поверх TLS, который работает поверх TCP.
- Долго устанавливается соединение: сначала TCP устанавливает соединение, затем TLS устанавливает соединение поверх TCP.
- TCP передаёт один поток байтов. Если теряется один пакет, то все остальные пакеты не могут быть приняты и обработаны (блокировка начала очереди).
- Мультиплексирование соединения в HTTP/2 требует поддержки на транспортном уровне. Изменить TCP не представляется возможным.



HTTP/3

- HTTP/2, работающий поверх транспортного протокола QUIC.
- Мультиплексирование соединения реализовано на уровне QUIC.
- QUIC:
  - Работает поверх UDP, а не IP (чтобы не блокироваться защитными экранами)
  - Глубокая интеграция с TLS 1.3, требуется обязательное шифрование
  - Установка соединения QUIC совмещена с установкой соединения TLS.
  - Подтверждения о доставке (квитанции)
  - Повторные передачи
  - Механизмы управления скользящим окном и предотвращения заторов
  - Независимые потоки данных на основе фреймов.
  - Идентификаторы соединений и миграция соединений при переходе клиентов между сетями (например, из одной сети Wi-Fi в другую).
  - Высокие издержки на шифрование и зависимость от монополии Google.

HTTP/3

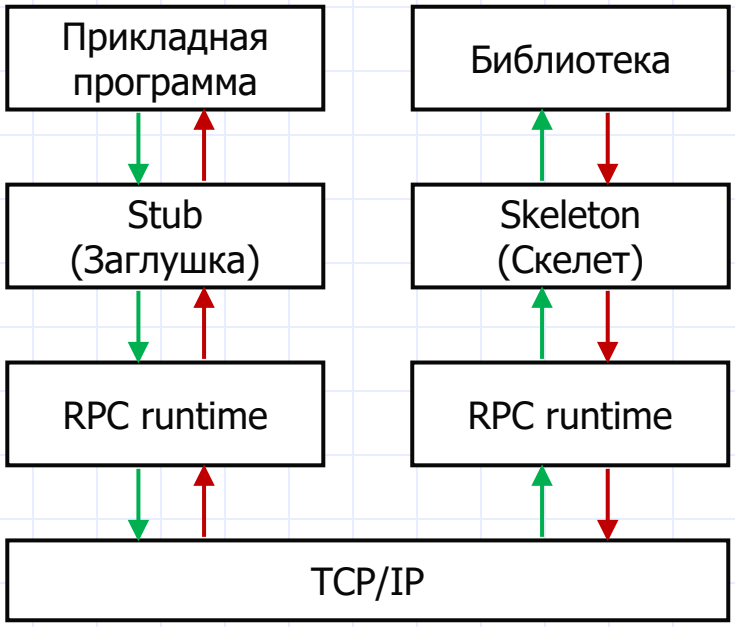

HTTP/2
Сжатие заголовков
Посылка от сервера
Мультиплексирование (потоки)
Приоритеты потоков
TLS
Аутентификация
Обмен ключами
Шифрование/дешифрование
TCP
Соединения, надёжность
Номера портов
HTTP/3
Сжатие заголовков
Посылка от сервера
Приоритеты потоков
TLS
Аутентификация
Обмен ключами
QUIC
Мультиплексирование (потоки)
Шифрование/дешифрование
Миграция соединений
Соединения, надёжность
UDP
Номера портов
IPv4 / IPv6

Технологии распределённых вычислений

- RPC (вызов удалённых процедур, англ. Remote Procedure Call)
  - ориентированность на конкретный язык программирования;
  - бинарный формат передачи данных, зависит от архитектуры и ОС;
  - не учитывает распределённую среду и связанные с ней задержки и ошибки.
- CORBA (запросы к удалённым объектам, англ. Common Object Request Broker Architecture)
  - ссылки на удалённые объекты => чрезмерная сложность;
  - недостатки RPC.
- Веб-службы (Web Services)
  - возможность взаимодействия в разнородной среде (архитектуры, ОС, ЯП);
  - учитывает распределённую среду и связанные с ней задержки и ошибки;
  - текстовый протокол SOAP (англ. Simple Object Access Protocol).

Технология удалённого вызова процедур RPC

- Шаги
  - На языке IDL описывается интерфейс библиотеки.
  - Компилятором IDL генерируются заглушка (stub) библиотеки для клиента и скелет (skeleton) сервера.
  - Заглушка заменяет собой библиотеку и отвечает за установку соединения со скелетом сервера, отправку информации о вызываемой функции, сериализацию и отправку входных данных, а также десериализацию результирующих данных.



  - Скелет сервера отвечает за прослушивание входящих запросов от заглушек клиентов, приём информации о вызываемой функции, десериализацию входных данных, вызов библиотеки, сериализацию и отправку результирующих данных.
- Язык описания интерфейсов IDL (англ. Interface Definition Language)

C:
```
double GetTemperature(char *location, int locationLength);
```

IDL:
```
double GetTemperature(
    [in, array(locationLength)] char *location,
    [in] int locationLength);
```

Технология объектных запросов CORBA

- Ссылки на удалённые объекты, которые можно передавать между клиентами и другими службами.
- IDL с поддержкой ООП: создание удалённых объектов, вызов методов, изменение значений полей.

Проблемы:

- Отдельный вызов по сети на каждом обращении к объекту. Для эффективной реализации требуется изменить интерфейс (в примере: установка всех полей адреса одной функцией).
- Все накладные расходы вызова по сети неявные.
- Возможны ошибки временной недоступности сервера.

```
interface IAddress
{
    string country { get; set; }
    string city { get; set; }
    string streetAddress { get; set; }
}
```

```
Address a = RemoteLibrary.getAddress();
a.country = "Belarus";
a.city = "Minsk";
a.streetAddress = "P.Brovki, 6";
```

```
a.set("Belarus", "Minsk", "P.Brovki, 6");
```

- Ошибочным является сам подход с сокрытием факта взаимодействия по сети при выполнении вызовов функций и обращениям к удалённым объектам.

Веб-служба

- Веб-служба (сетевой сервис) – один или несколько взаимодействующих логических серверов, обеспечивающих доступ к конкретному типу ресурса с помощью веб-протоколов (HTTP и базирующихся на нём).
- Архитектура, ориентированная на службы (SOA – Service Oriented Architecture).
- SOAP (Simple Object Access Protocol) – открытый стандарт на основе XML для передачи данных между клиентами и серверами веб-служб.
  - В протоколе HTTP используется только метод POST, который не кэшируется и позволяет передать данные в запросе и в ответе.
- WSDL (Web Services Description Language) – открытый формат на основе XML для описания интерфейсов (контрактов) веб-служб.
- HTTP REST – архитектурный стиль построения веб-служб, использующий всю инфраструктуру HTTP (все методы и заголовки). В качестве формата передачи данных обычно используется JSON или XML.

Программная архитектура служб (SOA – Service Oriented Architecture).

- Совместимость (англ. Interoperability)
  - Стандартный протокол (HTTP), стандартный формат передачи данных (XML, JSON).
  - Контракт вместо программного интерфейса. Контракт – правила взаимодействия клиента со службой, описывающие доступные операции, типы данных, входные и выходные параметры операций. Контракт имеет стандартный формат (WSDL).
  - Возможность взаимодействия клиента и сервера, написанных на разных языках и разных платформах.
- Автономность
  - Устойчивость к (временной) недоступности зависимостей.
  - Библиотека не способна работать в отсутствие других библиотек, от которых она зависит. Служба должна быть способна работать автономно – в отсутствие других служб, от которых она зависит.
- Мобильность
  - Отделение функциональности от хостинга, а хостинга от конфигурации (протоколов, форматов и точек доступа к службе).
  - Позволяет переносить службу с одного сервера/адреса на другой без перекомпиляции.

Программная архитектура служб SOA (продолжение)

- Оговоренный жизненный цикл и многоверсионность
  - В библиотеках версия кодируется четырьмя числами: «major.minor.patch.build». Версия службы определяется датой выпуска (год, месяц, день).
  - Для службы оговаривается срок жизни каждой версии.
  - Переход на новую версию предполагает одновременную работу старой и новой версии службы с заданным сроком миграции зависимых от неё служб и клиентов.
- Самодокументируемость
  - Служба должна предоставлять всю необходимую информацию для её использования. Для этого создаются специальные точки доступа к службе.
  - Обычно службы предоставляют информацию о контракте (WSDL-схему).
- Обнаруживаемость
  - Наличие реестра и стандартных способов обнаружения службы для её использования.
- Готовность служб к объединению в общую среду взаимодействия (шину)
  - Маршрутизация сообщений между службами, очереди сообщений.
  - Мониторинг, аудит, протоколирование.
  - Управление развёртыванием служб.

Технология WCF платформы .NET для создания веб-служб на основе протокола SOAP

```
using System.Runtime.Serialization;
using System.ServiceModel;

namespace WcfServiceLibrary1
{
    [ServiceContract(Name = "WeatherService",
        Namespace = "http://example.com/weather/2021/05/24")]
    public interface IWeatherService
    {
        [OperationContract]
        double GetTemperature(string location);
        [OperationContract]
        WeatherInfo GetWeatherInfo(string location);
    }

    [DataContract]
    public class WeatherInfo
    {
        public WeatherInfo() { }
        [DataMember]
        public double Temperature { get; set; }
        [DataMember]
        public double Humidity { get; set; }
        [DataMember]
        public double Pressure { get; set; }

        public double GetTemperatureAsFahrenheit()
        {
            return Temperature * 9 / 5 + 32;
        }
    }
}
```

Интерфейс веб-службы

Контракт веб-службы

```
[ServiceContract]
[OperationContract]
[DataContract]
  [DataMember]
```

WCF – Windows Communication Foundation

Технология WCF платформы .NET для создания веб-служб на основе протокола SOAP

Реализация веб-службы

[ServiceBehavior]

```
using System;
using System.ServiceModel;

namespace WcfServiceLibrary1
{
    [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession,
        ConcurrencyMode = ConcurrencyMode.Multiple)]
    public class WeatherService : IWeatherService
    {
        public WeatherService()
        {
            Console.WriteLine("WeatherService instance created");
        }

        public double GetTemperature(string location)
        {
            return +15;
        }

        public WeatherInfo GetWeatherInfo(string location)
        {
            return new WeatherInfo() { Temperature = 15, Humidity = 65, Pressure = 710 };
        }

        static void Main()
        {
            var host = new ServiceHost(typeof(WeatherService));
            host.Open();
            Console.WriteLine("Press any key to stop the service");
            Console.ReadKey();
        }
    }
}
```

Технология WCF платформы .NET для создания веб-служб на основе протокола SOAP

Конфигурация веб-службы

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="WcfServiceLibrary1.WeatherService"
               behaviorConfiguration="Service1Behavior">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8731/WeatherService/"/>
          </baseAddresses>
        </host>
        <endpoint address ="" binding="wsHttpBinding" contract="WcfServiceLibrary1.IWeatherService">
          <identity>
            <dns value="localhost"/>
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="Service1Behavior">
          <serviceMetadata httpGetEnabled="True"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

http://localhost:8731/WeatherService/?wsdl – WSDL-описание веб-службы

Технология WCF платформы .NET для создания веб-служб на основе протокола SOAP

Клиент веб-службы

```
using System;
using WcfServiceClient.WeatherServiceReference;

namespace WcfServiceClient
{
    class Program
    {
        static void Main(string[] args)
        {
            var client = new WeatherServiceClient();
            var t = client.GetTemperature("Minsk");
            Console.WriteLine("Temperature: {0}", t);
            var w = client.GetWeatherInfo("Minsk");
            Console.WriteLine("Temperature: {0}, Humidity: {1}, Pressure: {2}",
                w.Temperature, w.Humidity, w.Pressure);
            client.Close();
        }
    }
}
```

Добавление ссылки на службу в Visual Studio:
> Обозреватель решений (Solution Explorer)
> ПКМ – открыть контекстное меню проекта
> Добавить (Add)
> Ссылка на службу (Service Reference)
> Адрес (Address): `http://localhost:8731/WeatherService/mex`
> Перейти (Go)
> Пространство имён (Namespace): `WeatherServiceReference`
> OK

Технология WCF платформы .NET для создания веб-служб на основе протокола SOAP

Конфигурация клиента веб-службы

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <bindings>
            <wsHttpBinding>
                <binding name="WSHttpBinding_WeatherService" />
            </wsHttpBinding>
        </bindings>
        <client>
            <endpoint address="http://localhost:8731/WeatherService/"
                      binding="wsHttpBinding"
                      bindingConfiguration="WSHttpBinding_WeatherService"
                      contract="WeatherServiceReference.WeatherService"
                      name="WSHttpBinding_WeatherService">
            </endpoint>
        </client>
    </system.serviceModel>
</configuration>
```

Генерируется автоматически при добавлении ссылки на службу в Visual Studio.

HTTP REST

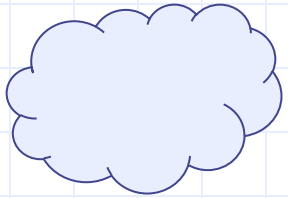
- REST (англ. Representational State Transfer – «передача состояния представления»)
  - используется вся инфраструктура HTTP (все методы, заголовки);
  - формат передачи данных: JSON, XML;
- Основные принципы:
  - Строгое разделение клиента и сервера.
  - Сервер не хранит клиентское состояние представления, все необходимые данные передаются в запросе.
  - Возможность кэширования на стороне клиента.
  - Единообразие интерфейса.
  - Возможность добавления промежуточных слоёв между клиентом и сервером (балансировка нагрузки, шифрование, кэширование и др.).
- Позволяет строить производительные, масштабируемые, переносимые приложения (службы) с унифицированным интерфейсом.

Пример интерфейса службы HTTP REST

- RESTful веб-служба для работы с личным блогом
  - Авторизация – HTTP-заголовок `Authorization: Bearer <jwt>`

Действие	**Метод**	**URI**	**Тело**
Добавить новую запись	POST	/my/posts	{ title, text }
Получить список своих записей	GET	/my/posts?page={n}	-
Получить представление записи #id	GET	/my/posts/{id}	-
Обновить содержимое записи #id	PUT	/my/posts/{id}	{ title, text }
Удалить запись #id	DELETE	/my/posts/{id}	-
Получить список записей другого пользователя	GET	/{user-id}/posts	-
Отметить запись другого пользователя	POST	/{user-id}/posts/{post-id}/like	-

Облачные технологии



- Облачные технологии – это технологии распределённой обработки и хранения данных, которые абстрагируют место и способ хранения данных, платформу и особенности выполнения распределённых вычислений.
- Центр (хранения и) обработки данных (сокр. ЦОД/ЦХОД).
- Виртуализация как основа облачных вычислений.
- Виды облачных моделей:
  - Infrastructure as a Service (IaaS) – инфраструктура как услуга;
  - Platform as a Service (PaaS) – платформа как услуга;
  - Software as a Service (SaaS) – приложение как услуга.

Масштабирование

- Масштабируемость – способность системы справляться с увеличением рабочей нагрузки путём добавления ресурсов.
- Виды масштабирования:
  - Вертикальное масштабирование – наращивание ресурсов одного узла (ядер процессоров, оперативной памяти, дискового пространства).
  - Горизонтальное масштабирование – наращивание количества узлов в распределённой системе.

Конец