Глава 3
Транспортный уровень

Компьютерные сети: Нисходящий подход

Эксмо, 2015

Примечание по использованию презентаций:
Данная презентация свободно доступна для всех (преподавателей, студентов, читателей). Вы можете просматривать слайды и использовать информацию из презентации о своему усмотрению. Взамен, авторы просят о следующем:

- При использовании слайдов (например, в аудитории) указывайте источник (чтобы другие люди использовали нашу книгу!)
- Если вы выкладываете слайды на сайт, укажите информацию об авторских правах.

Спасибо и приятного чтения!

Глава 3: Транспортный уровень

Наши цели:

- Понять принципы работы служб транспортного уровня:
  - Мультиплексирования, демультиплексирования
  - Надежной передачи данных
  - Управления потоком
  - Управления перегрузками
- Узнать о Интернет протоколах транспортного уровня:
  - UDP: протокол передачи данных без установления логического соединения
  - TCP: протокол надежной передачи данных с установлением логического соединения
  - Механизм управление перегрузками протокола TCP

Глава 3: План

3.1 Службы транспортного уровня

3.2 Мультиплексирование и демультиплексирование

3.3 Передача данных без установления логического соединения: протокол UDP

3.4 Принципы надежной передачи данных

3.5 Передача данных с установлением логического соединения: TCP

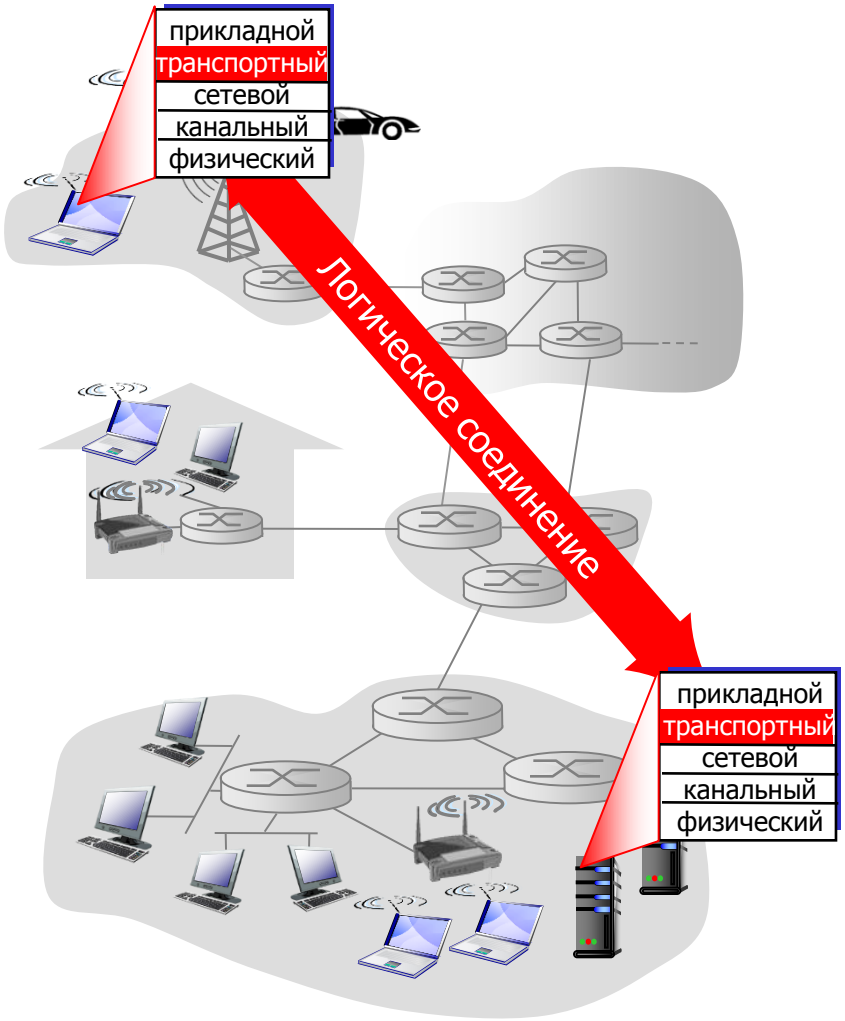
- Структура сегмента
- Надежная передача данных
- Управление потоком
- Управление соединением

3.6 Принципы управления перегрузками

3.7 Механизм управления перегрузками протокола TCP

Протоколы и службы транспортного уровня

- Предоставляют *логическое соединение* между процессами приложений, запущенными на разных хостах
- Транспортные протоколы действуют на конечных системах соединения
  - Сторона отправителя: разделяя сообщения приложения на *сегменты*, передает их на сетевой уровень
  - Сторона получателя: передает прикладному уровню сообщения, собранные из сегментов
- Для приложений доступно несколько транспортных протоколов
  - Интернет-протоколы: TCP и UDP



Транспортный и сетевой уровни

- *Сетевой уровень:* логическое соединение между хостами
- *Транспортный уровень:* логическое соединение между процессами
  - Опираясь на сервисы сетевого уровня, функционально дополняет их

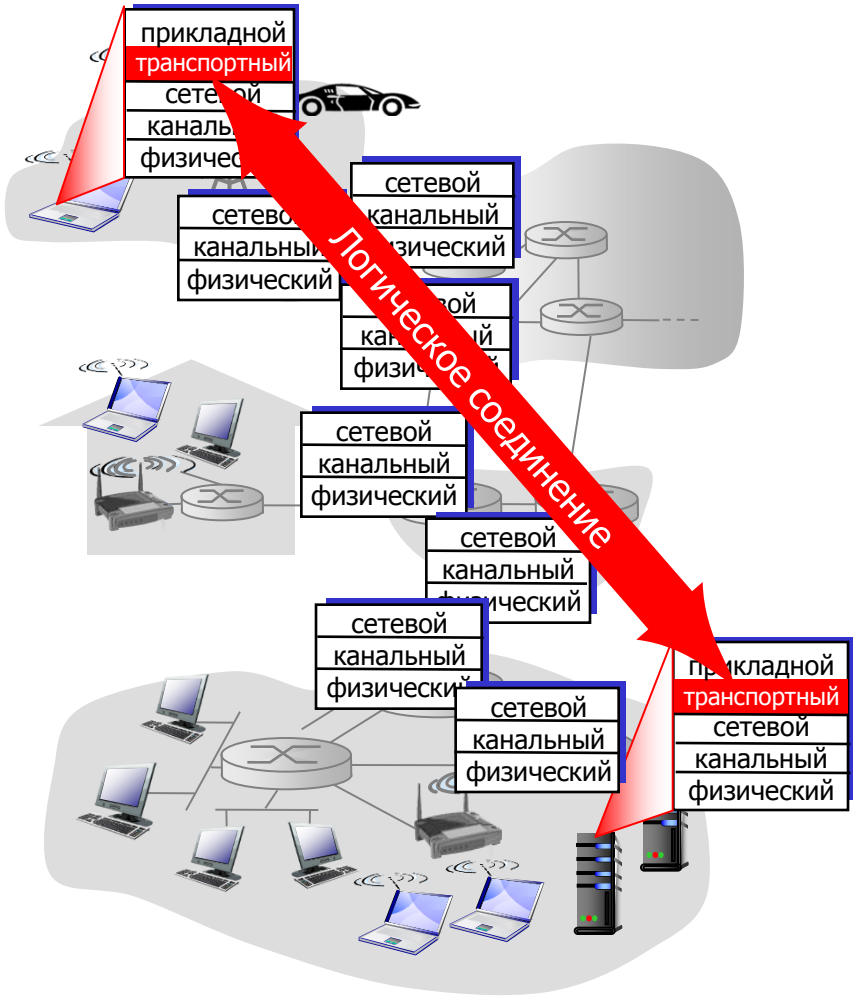
Семейная аналогия

12 детей в доме Анны отправляют письма 12 детям в доме Билла:

- хосты = дома
- процессы = дети
- Сообщения приложений = письма в конвертах
- Транспортный протокол = Анна и Билл, занимающиеся сбором и отправкой почты в своих домах
- Протокол сетевого уровня = почтовая служба

Интернет-протоколы транспортного уровня

- Надежная, упорядоченная передача данных (протокол TCP)
  - Управление перегрузками
  - Управление потоком
  - Установление соединения
- Ненадежная, неупорядоченная передача данных (протокол UDP)
  - Простое и эффективное расширение протокола IP, работающего по принципу негарантированной доставки
- Не гарантируется:
  - Максимальная допустимая задержка
  - Стабильная пропускная способность



Глава 3: План

3.1 Службы транспортного уровня

3.2 Мультиплексирование и демультиплексирование

3.3 Передача данных без установления логического соединения: протокол UDP

3.4 Принципы надежной передачи данных

3.5 Передача данных с установлением логического соединения: TCP

- Структура сегмента
- Надежная передача данных
- Управление потоком
- Управление соединением

3.6 Принципы управления перегрузками

3.7 Механизм управления перегрузками протокола TCP

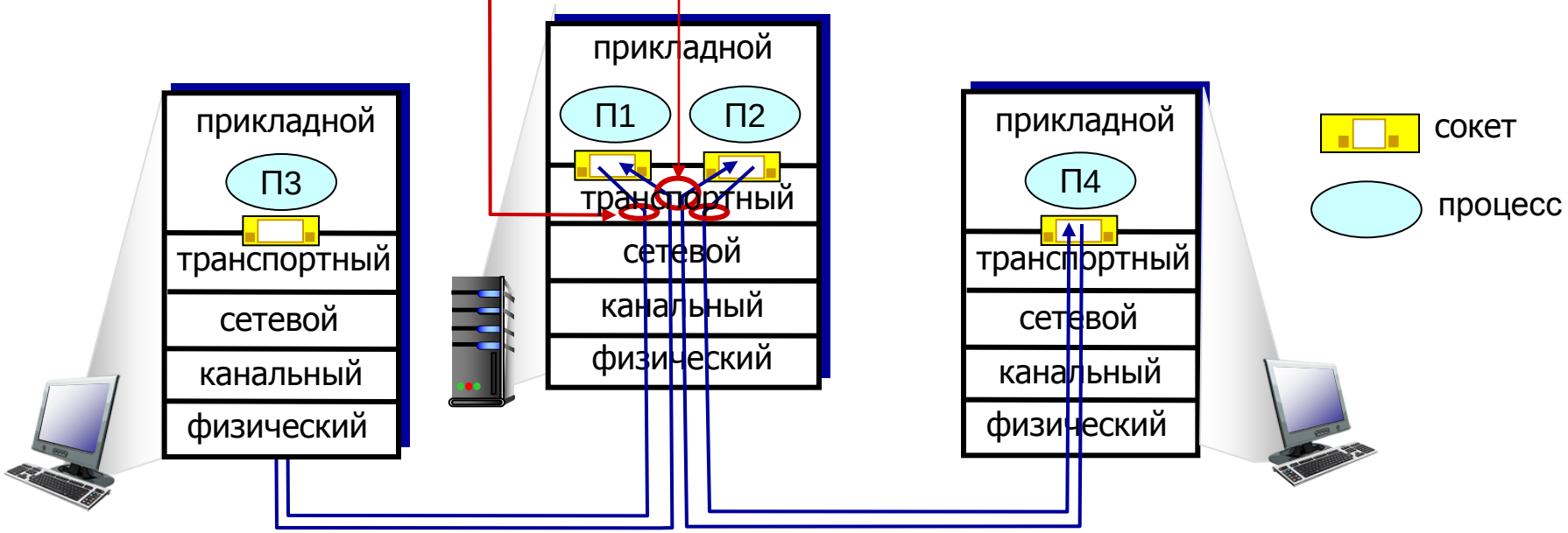
Мультиплексирование / демультиплексирование

Мультиплексирование у отправителя

Обработка данных из нескольких сокетов, добавление заголовка транспортного уровня (позже используемого при демультиплексировании)

Демультиплексирование у получателя

Использует информацию заголовка для передачи сегментов на нужный сокет



Как работает демультиплексирование

- Хост получает IP-дейтаграммы
  - Каждая дейтаграмма содержит IP-адрес отправителя и получателя
  - В каждой IP-дейтаграмме содержится один сегмент транспортного уровня
  - В каждом сегменте указаны номера портов отправителя и получателя
- Хост использует *IP-адреса и номера портов* для передачи сегмента на нужный сокет

Структура сегмента TCP/UDP

Демультиплексирование без установления логического соединения

- Созданный сокет имеет номер порта локального хоста:

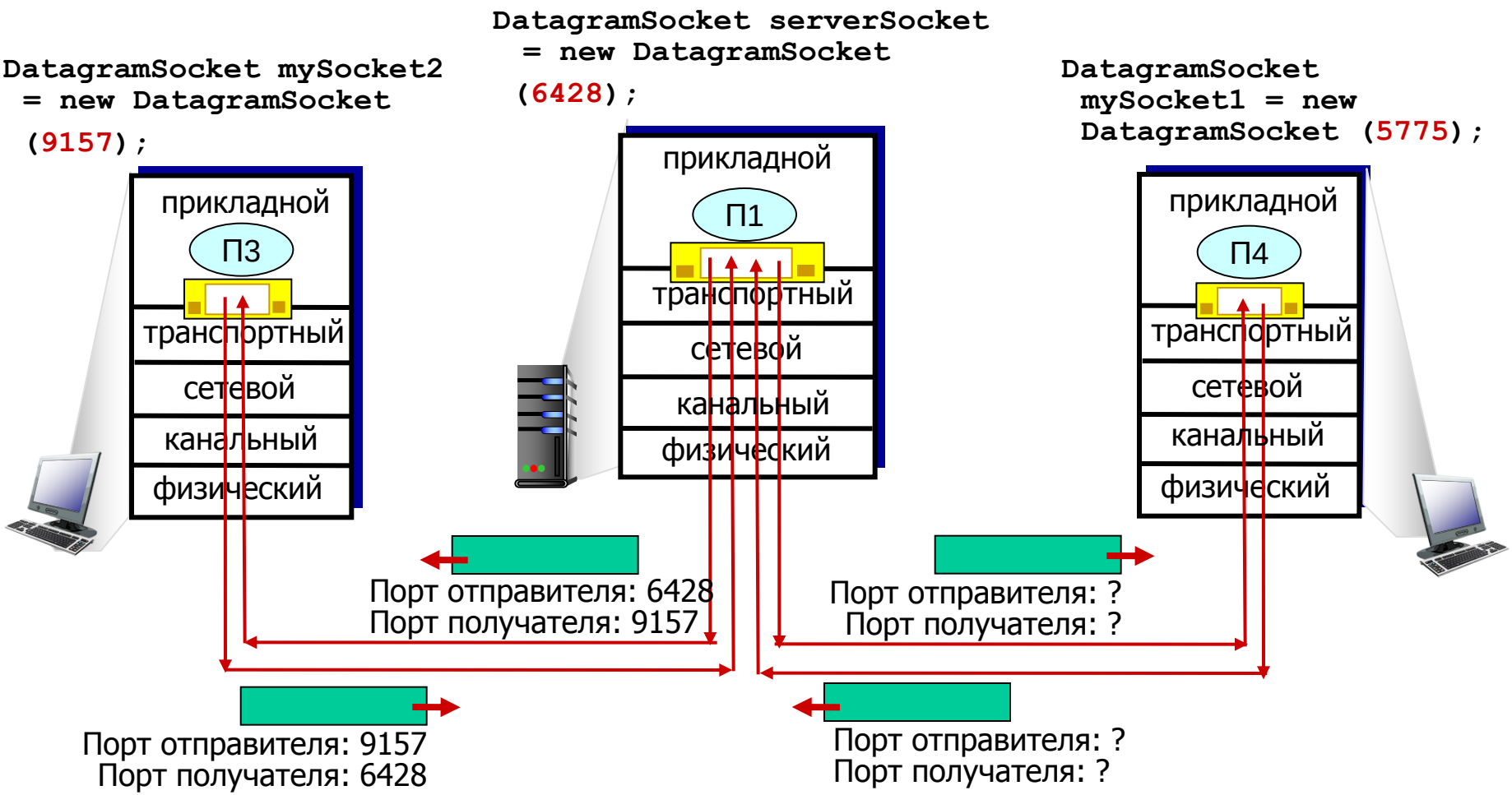
```
DatagramSocket mySocket1 =
new DatagramSocket(12534);
```

- При создании дейтаграммы для отправки UDP-сегмента необходимо определить
  - IP -адрес получателя
  - Номер порта получателя

- При получении хостом UDP-сегмента:
  - Проверка номера порта получателя в сегменте
  - UDP-сегмент направляется сокету с таким же номером порта

IP-дейтаграммы с *одинаковыми номерами портов получателя,* но с разными IP-адресами отправителей и/или номерами портов отправителя будут направлены *одному и тому же сокету* получателя

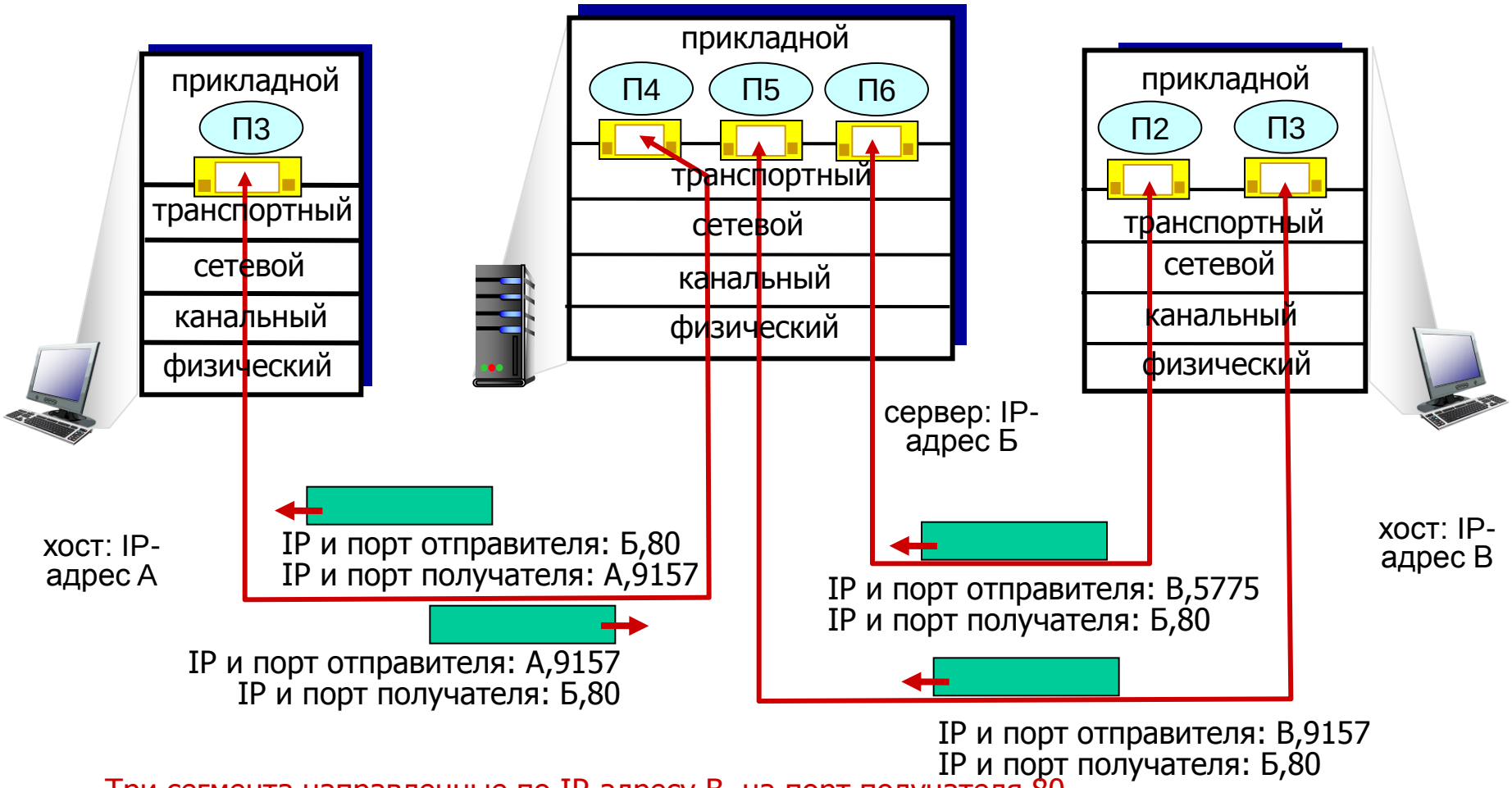
Демультиплексирование без установления логического соединения: пример



Демультиплексирование с установлением логического соединения

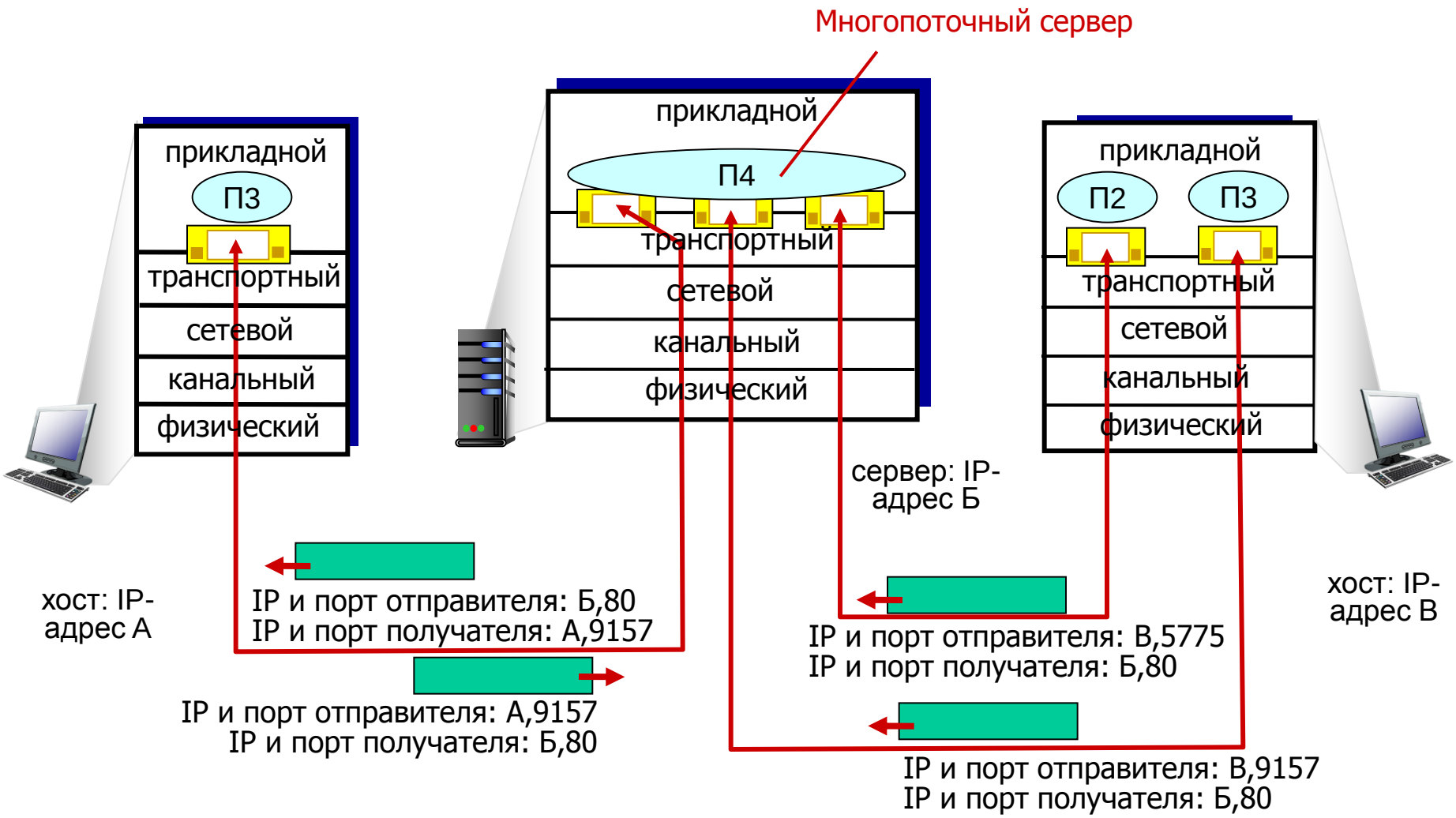
- TCP-сокет идентифицируется октетом:
  - IP-адрес отправителя
  - Номер порта отправителя
  - IP-адрес получателя
  - Номер порта получателя
- демультиплексирование: получатель использует все четыре значения для передачи сегмента на соответствующий сокет
- Хост-сервер может поддерживать множество одновременных TCP-сокетов:
  - Каждый сокет определяется собственным октетом
- Веб-серверы имеют разные сокеты для каждого клиента
  - При непостоянном соединений каждое HTTP-соединение будет иметь отдельный сокет

Демультиплексирование с установлением логического соединения: пример



Три сегмента направленные по IP-адресу В, на порт получателя 80, демультиплексированы к *разным* сокетам

Демультиплексирование с установлением логического соединения: пример



Глава 3: План

3.1 Службы транспортного уровня

3.2 Мультиплексирование и демультиплексирование

3.3 Передача данных без установления логического соединения: протокол UDP

3.4 Принципы надежной передачи данных

3.5 Передача данных с установлением логического соединения: TCP

- Структура сегмента
- Надежная передача данных
- Управление потоком
- Управление соединением

3.6 Принципы управления перегрузками

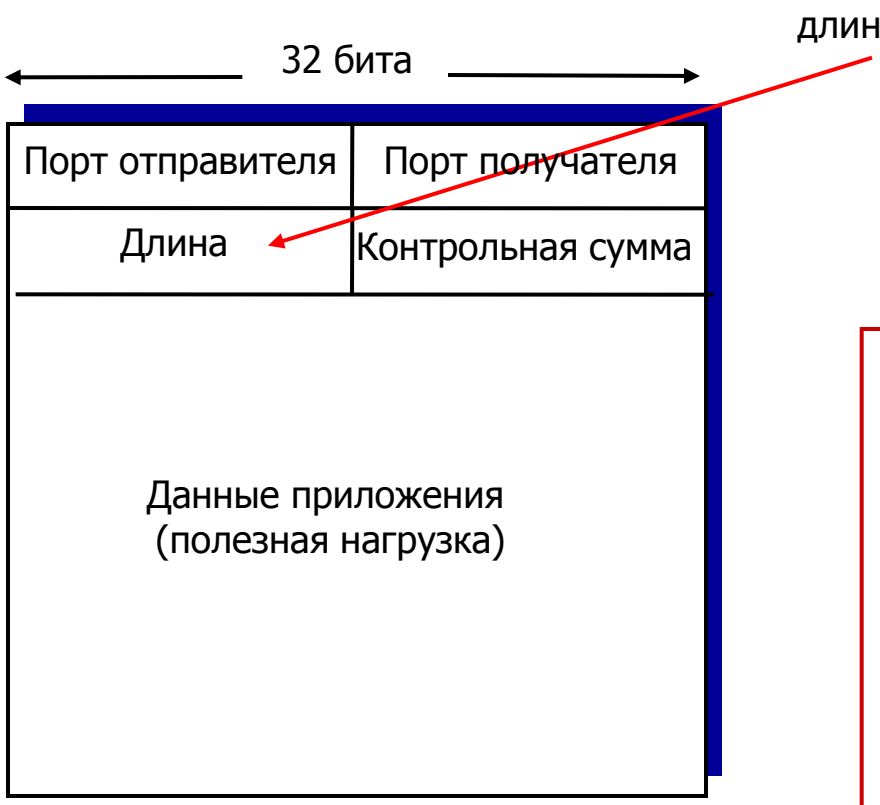
3.7 Механизм управления перегрузками протокола TCP

UDP: Протокол пользовательских дейтаграмм [RFC 768]

- «без излишеств», «минимальный» транспортный протокол Интернета
- Служба негарантированной доставки допускает:
  - потерю UDP-сегментов
  - доставку UDP-сегментов приложению с нарушением порядка
- *Без установления соединения:*
  - Нет процедуры установления соединения между UDP-отправителем и получателем
  - каждый UDP-сегмент обрабатывается независимо от других

- Протокол UDP используют:
  - Потоковые мультимедийные приложения (допускающие потери и чувствительные к скорости)
  - DNS
  - SNMP
- Надежная передача данных по протоколу UDP:
  - Добавление надежной передачи на прикладном уровне
  - Анализ ошибок для конкретных приложений!

UDP: заголовок сегмента



Структура UDP-сегмента

Для чего нужен UDP?

- Нет установления соединения(что может добавлять задержку)
- простота: состояния соединения не сохраняются у получателя и отправителя
- Малый размер заголовка
- Нет контроля перегрузки: UDP может действовать так быстро, как пожелает

Контрольная сумма UDP

Цель: обнаружение «ошибок» (например, инвертированные разряды) в переданном сегменте

отправитель:

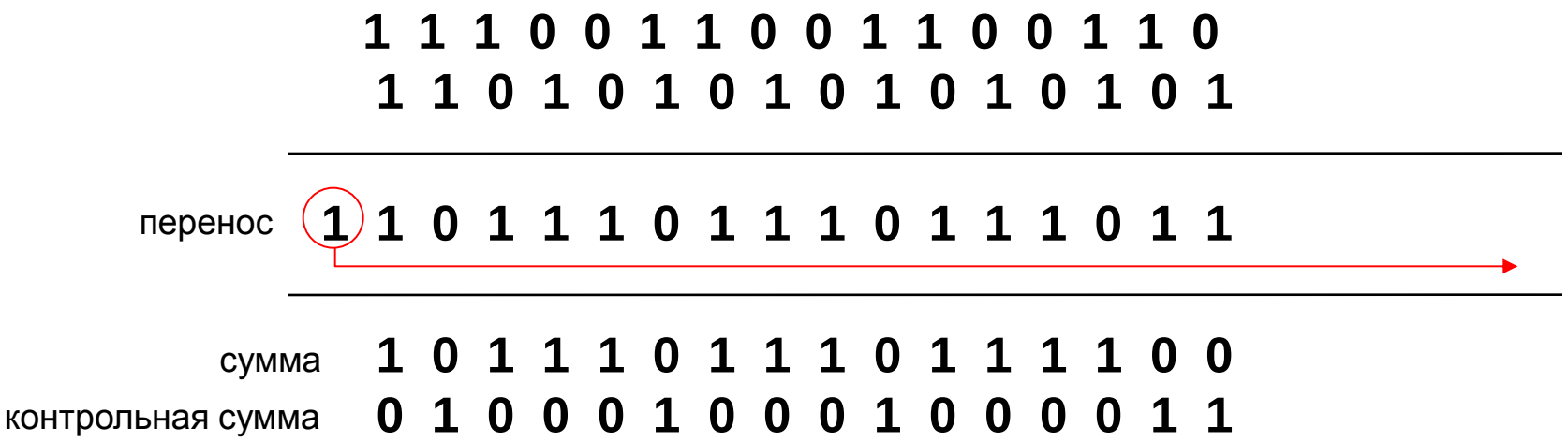
- Представление содержимого сегмента, включая поля заголовка в виде последовательности 16-битных целых
- Контрольная сумма: дополнение к содержимому сегмента (дополнение суммы до единицы)
- Отправитель помещает значение контрольной суммы в поле контрольная сумма UDP-сегмента

получатель:

- вычисление контрольной суммы полученного сегмента
- Соответствует полученное значение и значению в поле контрольная сумма:
  - NO – обнаружена ошибка
  - YES – ошибок не обнаружено. *Но, тем не менее ошибки возможны?* Об этом далее ….

Контрольная сумма: пример

пример: складываем два 16-битных целых числа



Примечание: при сложении чисел, знак переноса от старшего разряда должен быть добавлен к результату

Глава 3: План

3.1 Службы транспортного уровня

3.2 Мультиплексирование и демультиплексирование

3.3 Передача данных без установления логического соединения: протокол UDP

3.4 Принципы надежной передачи данных

3.5 Передача данных с установлением логического соединения: TCP

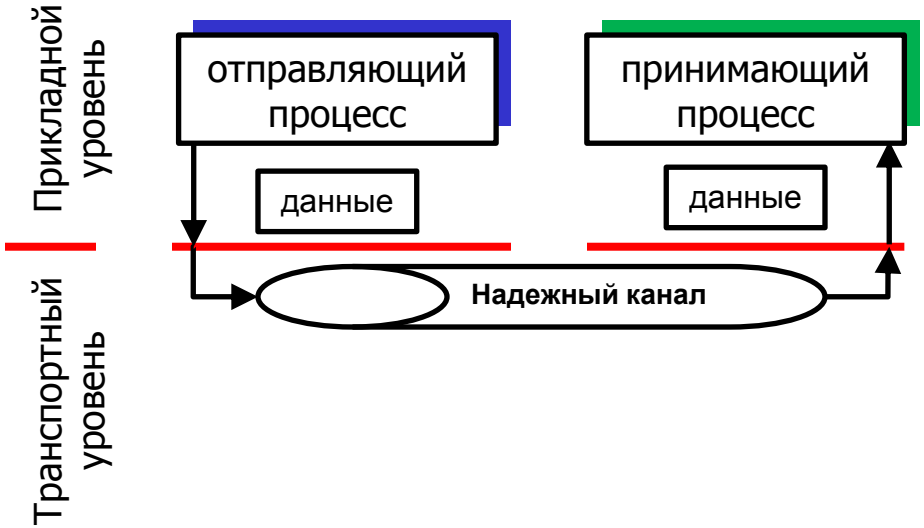
- Структура сегмента
- Надежная передача данных
- Управление потоком
- Управление соединением

3.6 Принципы управления перегрузками

3.7 Механизм управления перегрузками протокола TCP

Принципы надежной передачи данных

- Важно для прикладного, транспортного и канального уровней
  - 10 важнейших тем сетевого администрирования!

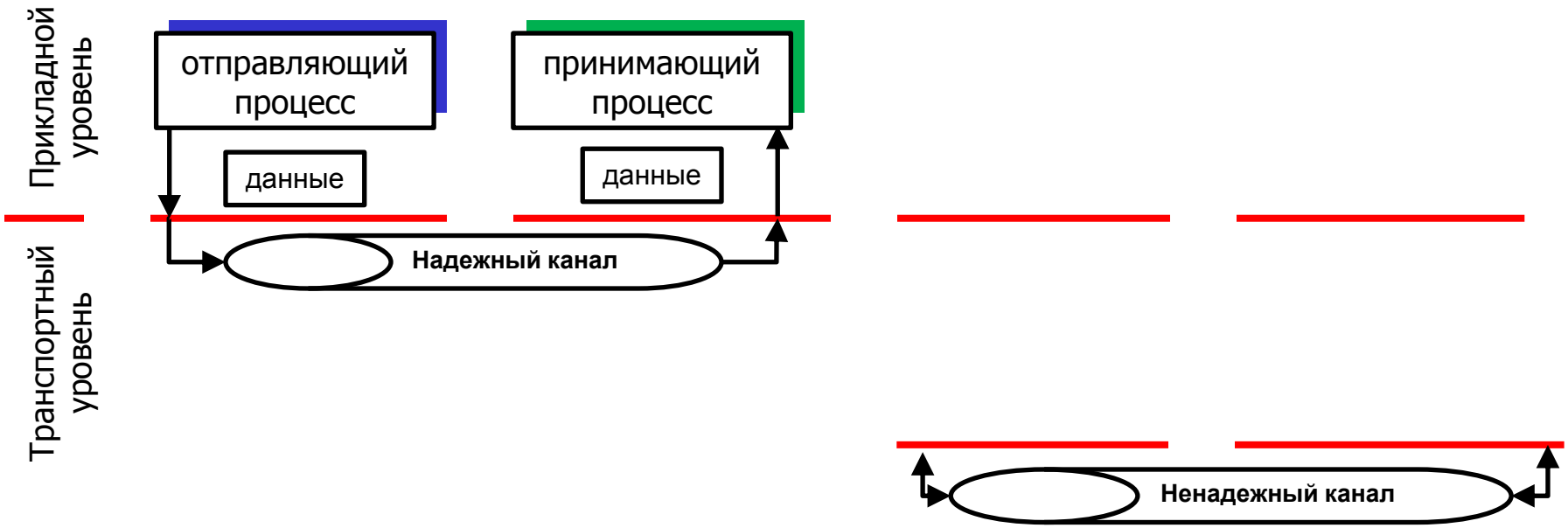


(а) предоставляемая служба

- Характеристики ненадежного канала будут полностью определять сложность протокола надежной передачи данных (**rdt**)

Принципы надежной передачи данных

- Важно для прикладного, транспортного и канального уровней
  - 10 важнейших тем сетевого администрирования!



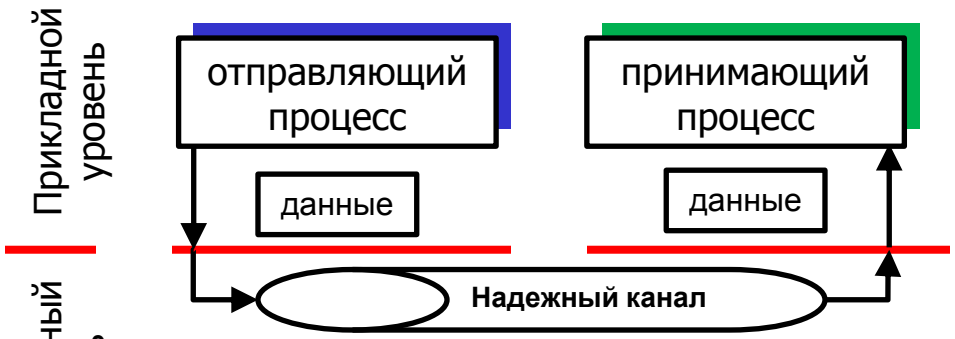
(а) предоставляемая служба

(б) реализация службы

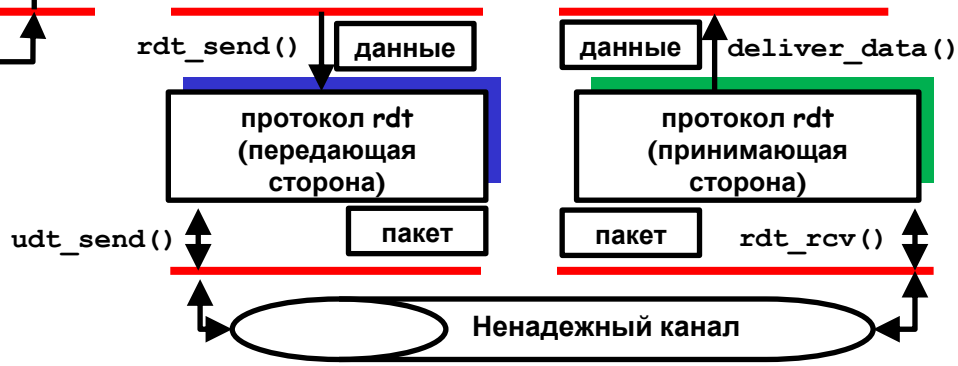
- Характеристики ненадежного канала будут полностью определять сложность протокола надежной передачи данных (**rdt**)

Принципы надежной передачи данных

- Важно для прикладного, транспортного и канального уровней
  - 10 важнейших тем сетевого администрирования!



(а) предоставляемая служба



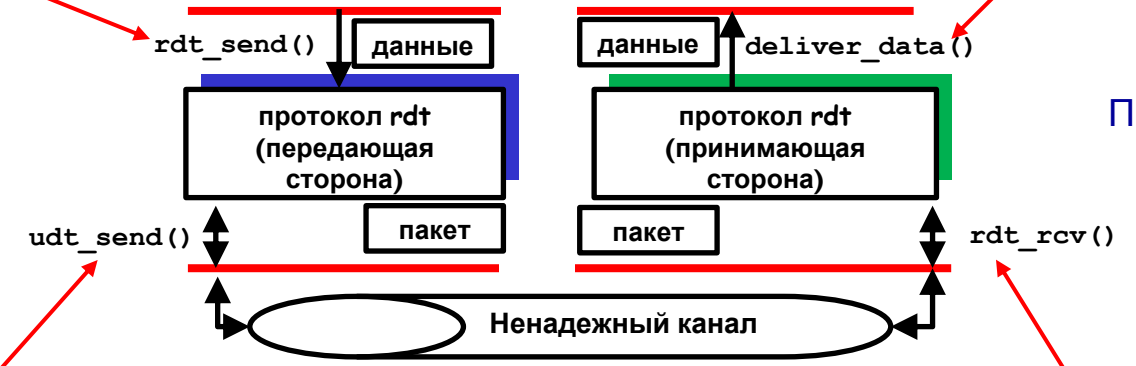
(б) реализация службы

- Характеристики ненадежного канала будут полностью определять сложность протокола надежной передачи данных (**rdt**)

Надежная доставка данных: начало

rdt_send(): вызывается сверху, например, приложением. Передает данные вышестоящему уровню получателя

deliver_data(): вызывается протоколом rdt для передачи данных наверх



udt_send(): вызывается протоколом rdt для передачи данных получателю по ненадежному каналу

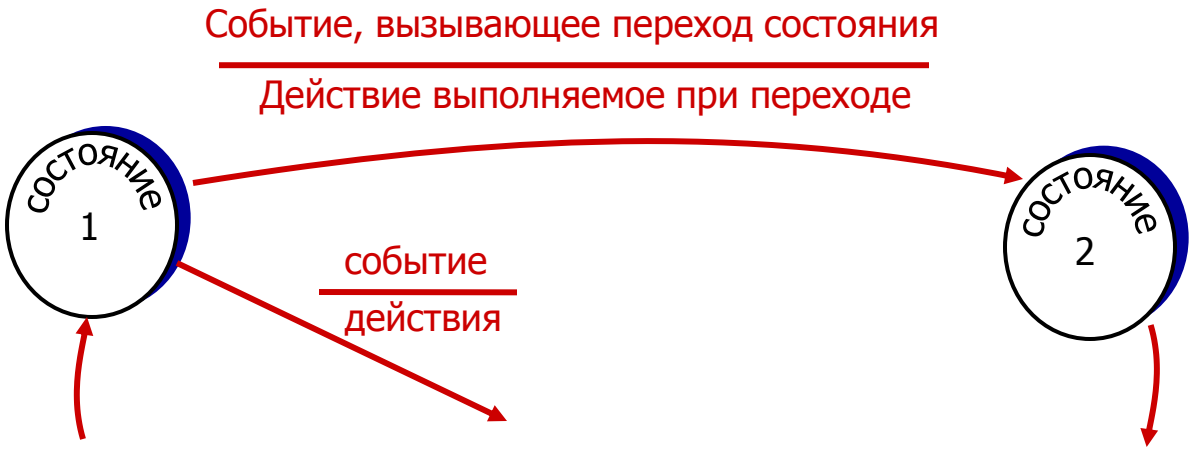
rdt_rcv(): вызывается при передаче пакета получателю из канала

Надежная передача данных: начало

Наши действия:

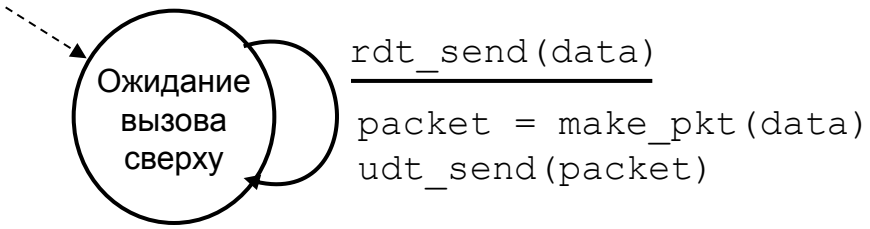
- Постепенно разработаем стороны отправителя и получателя протокола надежной передачи данных (rdt)
- Рассмотрим только однонаправленную передачу данных
  - Но, управляющая соединением информация будет распространяться в обе стороны!
- Для описания отправителя и получателя FSM-схемы

состояние: в этом «состоянии» следующее состояние однозначно определяется следующим событием

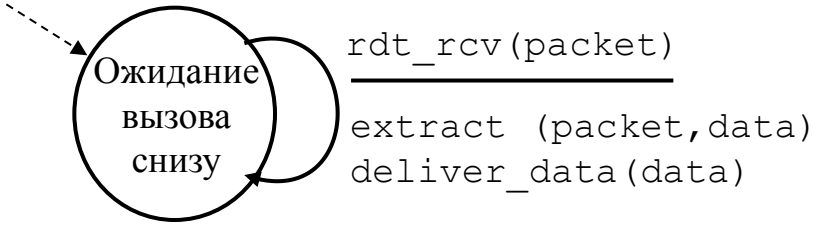


Протокол `rdt1.0`: передача данных по надежному каналу

- Нижерасположенный канал абсолютно надежен
  - Нет битовых ошибок
  - Нет потерь пакетов
- отдельные FSM-схемы отправителя и получателя:
  - Отправитель отправляет данные в нижерасположенный канал
  - Получатель считывает данные из нижерасположенного канала



отправитель



получатель

Протокол `rdt2.0`: канал с ошибками бит

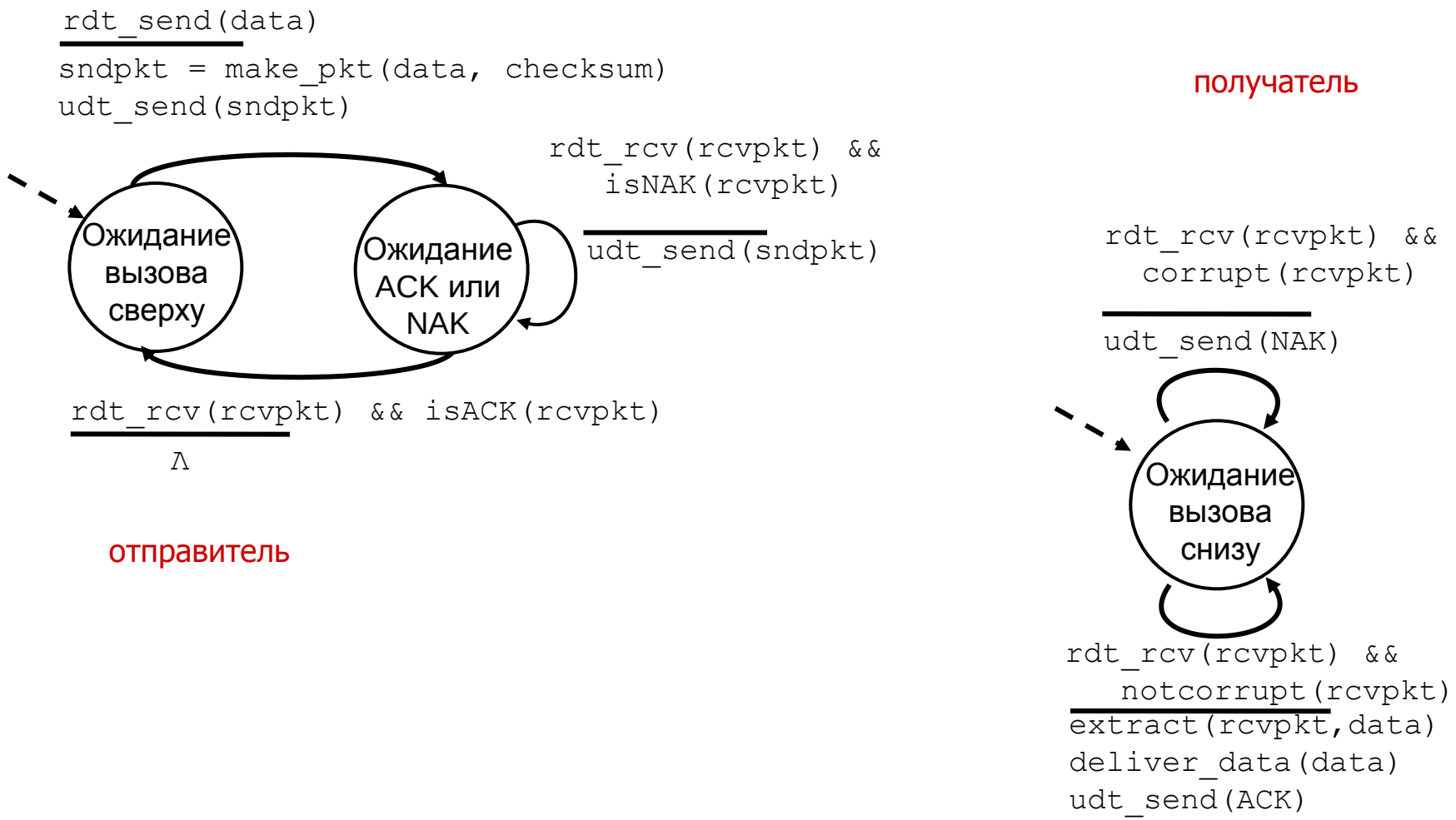
- Ниже расположенный канал может искажать биты в пакете
  - Контрольная сумма для обнаружения ошибок бит
- *Вопрос*: как исправить ошибки?

Как люди исправляют «ошибки» во время общения?

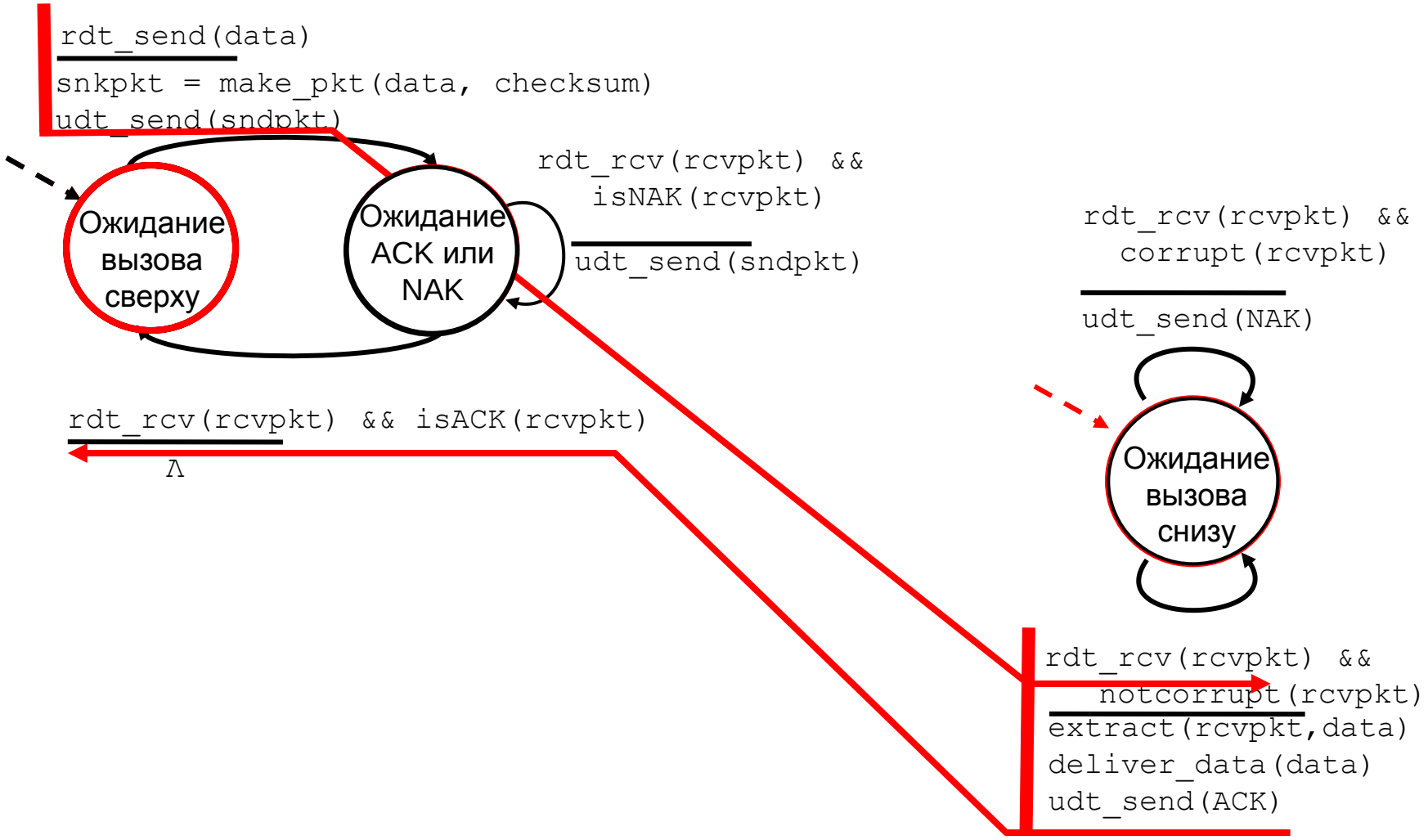
Протокол `rdt2.0`: канал с ошибками бит

- Ниже расположенный канал может искажать биты в пакете
  - Контрольная сумма для обнаружения ошибок бит
- *Вопрос*: как исправить ошибки?
  - *Подтверждения (ACK-пакеты):* получатель явно сообщает отправителю, что пакет получен корректно
  - *Отрицательные подтверждения (NAK-пакеты):* получатель явно сообщает отправителю, что в пакете есть ошибки
  - Отправитель повторно передает пакет данных при получении NAK-пакетов
- Новые механизмы в **`rdt2.0`** (отличие от **`rdt1.0`**):
  - Обнаружение ошибок
  - подтверждения (ACK-пакеты и NAK-пакеты), как ответ получателя отправителю

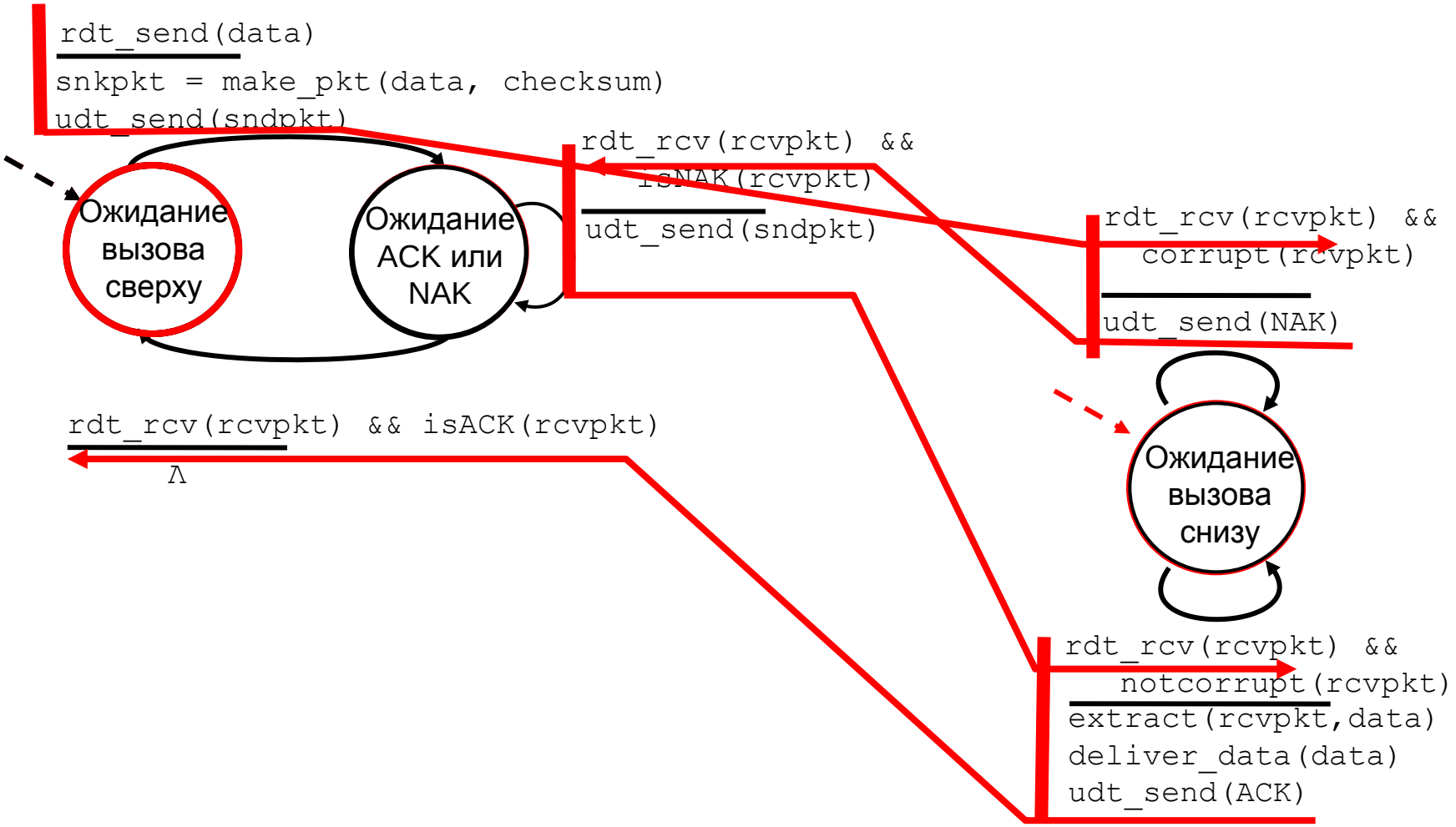
Протокол **rdt2.0**: FSM-схема



Протокол rdt2.0: сценарий в отсутствие ошибок



Протокол rdt2.0: сценарий при наличии ошибок



Протокол `rdt2.0`: имеет серьезный недостаток!

Что произойдет, если ACK/NAK-пакеты повреждены?

- Отправитель не знает, что произошло у получателя!
- Нет возможности просто повторить отправку так, как возможно дублирование пакетов

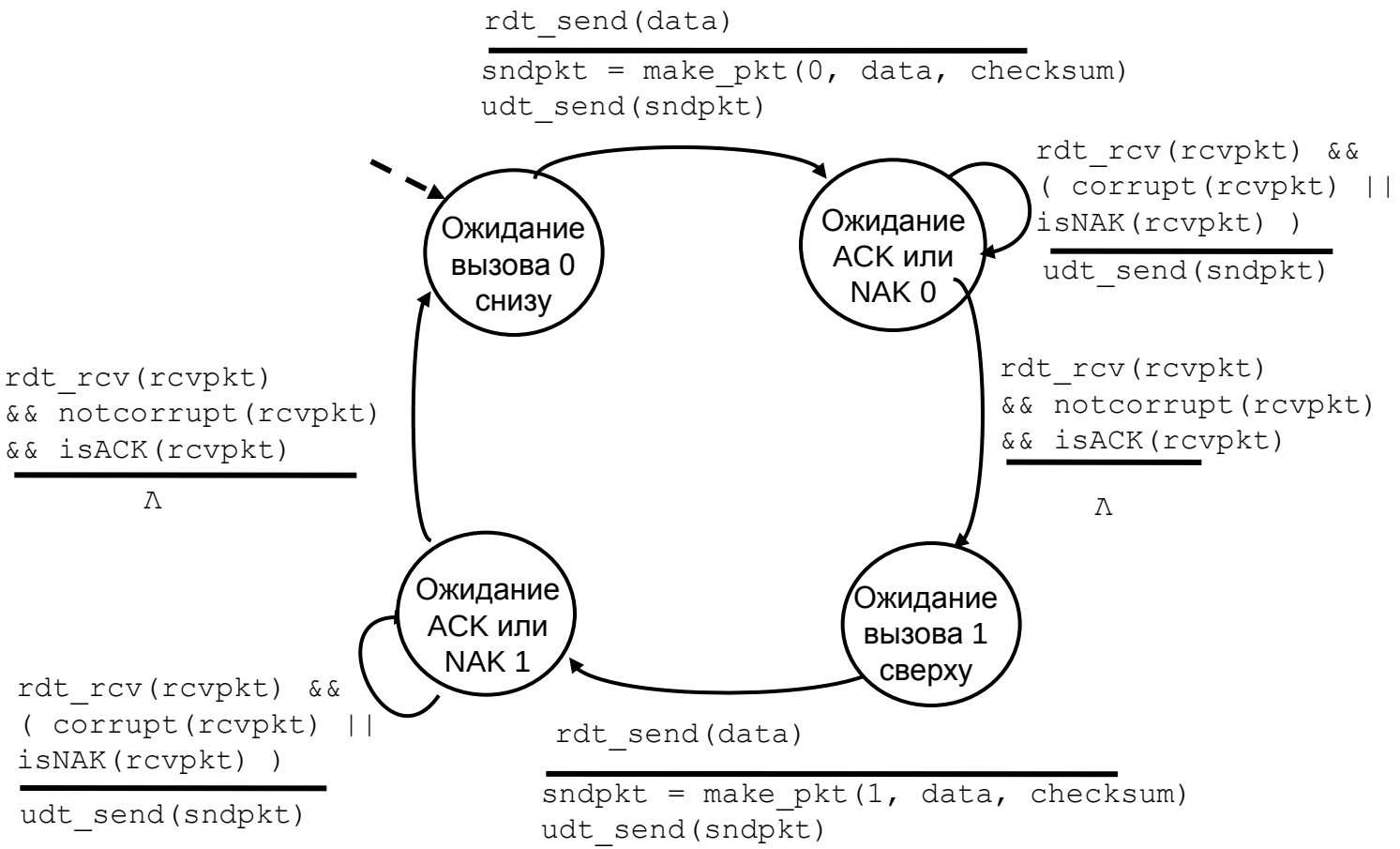
Обработка дублирующих пакетов:

- Отправитель повторно передает текущий пакет, если ACK/NAK-пакеты повреждены
- Отправитель добавляет *порядковый номер* каждому пакету
- Получатель отклоняет (не передает выше) дубликат пакета

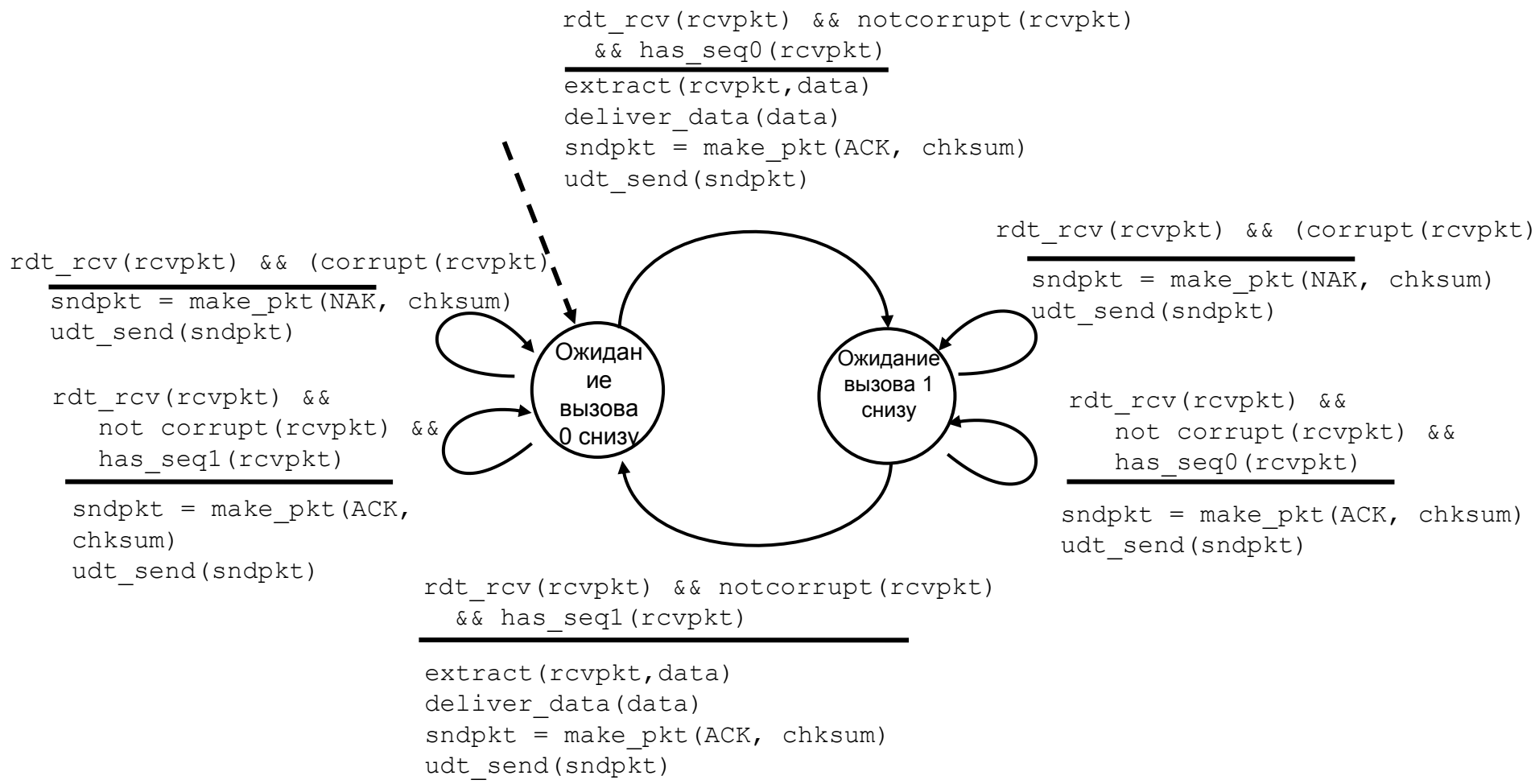
Протокол с ожиданием подтверждений

Отправитель передает один пакет, затем ждет ответа получателя

Протокол **rdt2.1**: FSM-схема отправителя, обработка ACK/NAK-пакетов



Протокол **rdt2.1**: FSM-схема получателя, обработка ACK/NAK-пакетов



Протокол `rdt2.1`: описание

отправитель:

- К пакету добавляется порядковый номер
- Двух порядковых номеров (0,1) будет достаточно. почему?
- Необходима проверка, если получен искаженный ACK/NAK-пакет
- В два раза больше состояний
  - Состояния должны «запоминать», какой порядковый номер должен быть у ожидаемого пакета 0 или 1

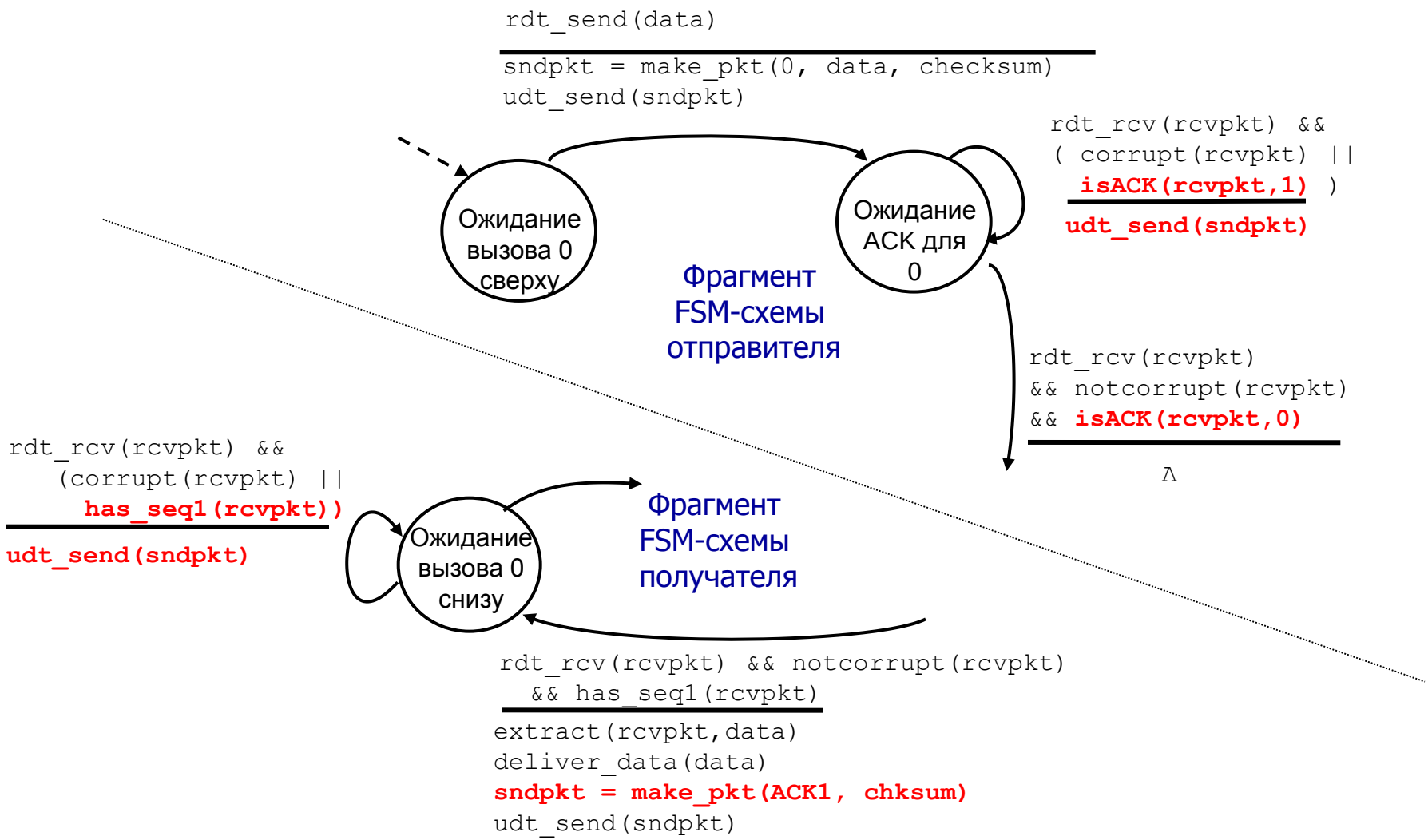
получатель:

- Должен выполнять проверку, если полученный пакет дублирующим
  - Состояние означает ожидание либо 0, либо 1 в порядковом номере пакета
- примечание: получатель *не* может знать были ли корректно получены его последние ACK/NAK-пакеты отправителем

Протокол `rdt2.2`: протокол без NAK-пакетов

- Тот же функционал, что и у протокола rdt2.1, использующего только ACK-пакеты
- вместо NAK-пакета, получатель отправляет ACK-пакет на последний верно полученный пакет данных
  - Получатель должен *явно* указывать порядковый номер пакета, на который отправляет подтверждение
- Отправитель реагирует на дублирование ACK-пакета точно как и на получение NAK-пакета: *повторяет передачу текущего пакета*

Протокол **rdt2.2**: фрагменты FSM-схем отправителя и получателя



Протокол `rdt3.0`: передача данных по каналу с возможными ошибками *и* потерями

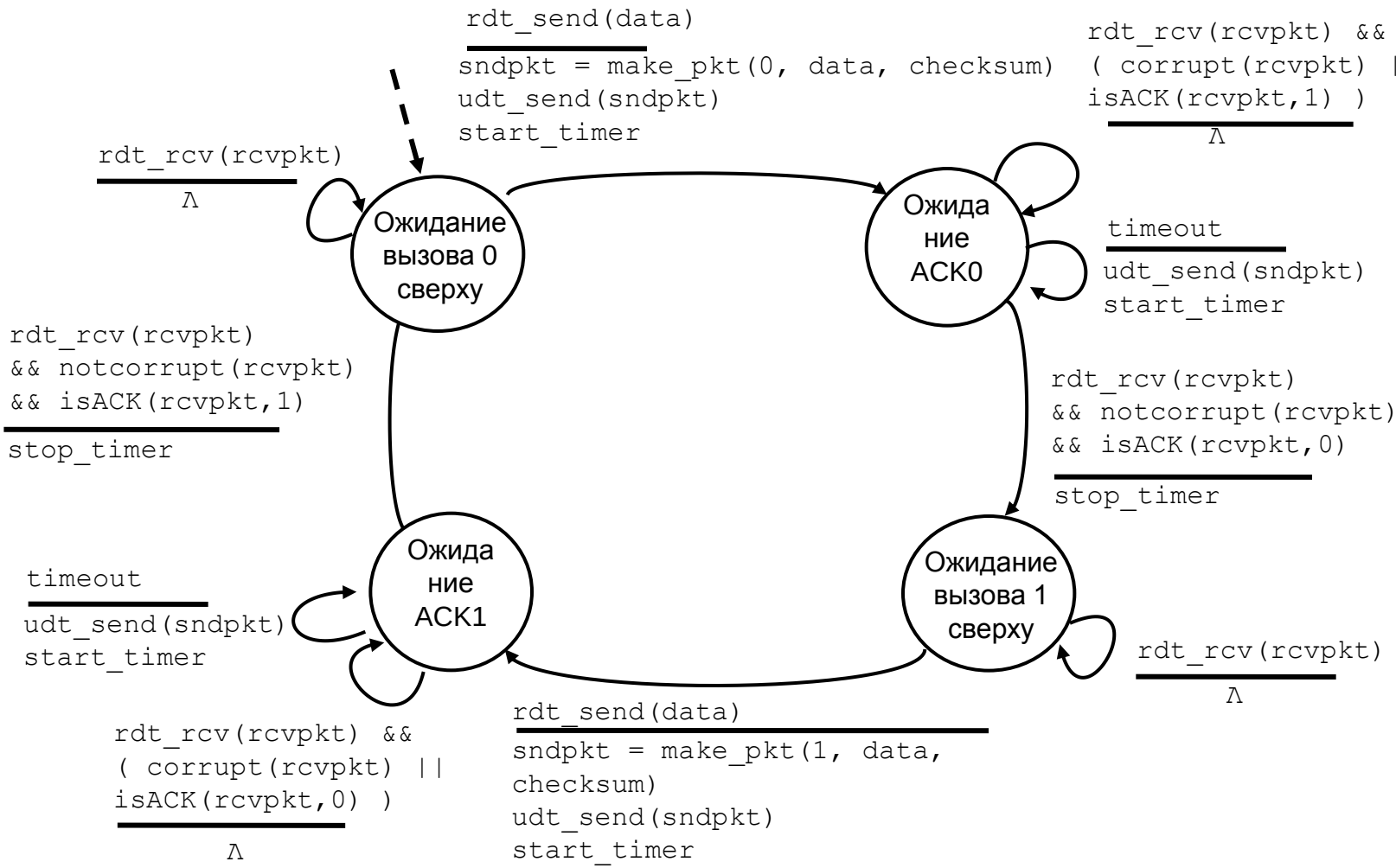
новое допущение:
нижерасположенный канал может терять пакеты данных и ACK-пакеты

- Контрольная сумма, порядковые номера, ACK-пакеты, механизм повторных передач, недостаточны, хотя и могут быть полезны

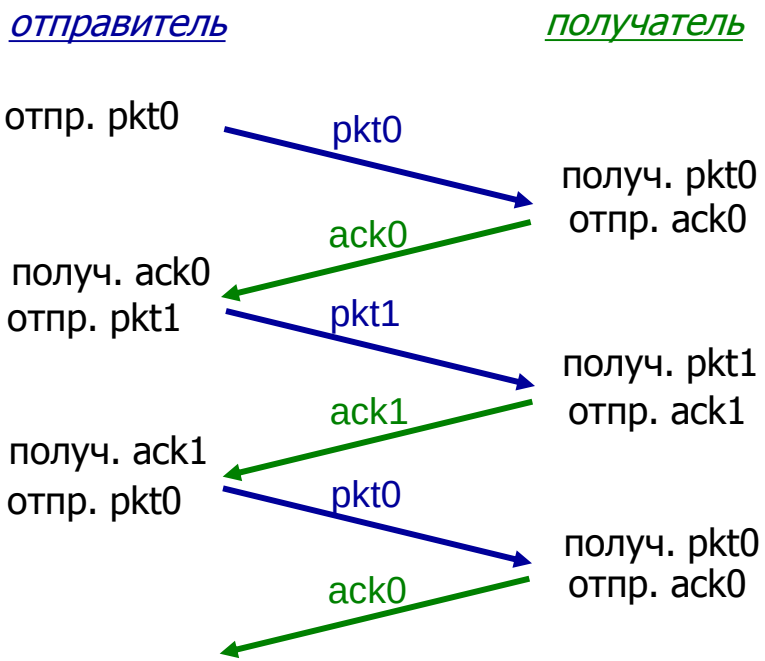
подход: отправитель ожидает ACK-пакет в течение «приемлемого» времени

- Выполняет повторную передачу, если ACK-пакет не был получен за это время
- Если пакет данных (или ACK-пакет) только задержался, но не был потерян:
  - Повторная передача приведет к дублированию, но порядковый номер покажет, что пакет уже был обработан
  - Получатель должен определить порядковый номер пакета, который уже был подтвержден
- Необходим таймер для обратного отсчета времени

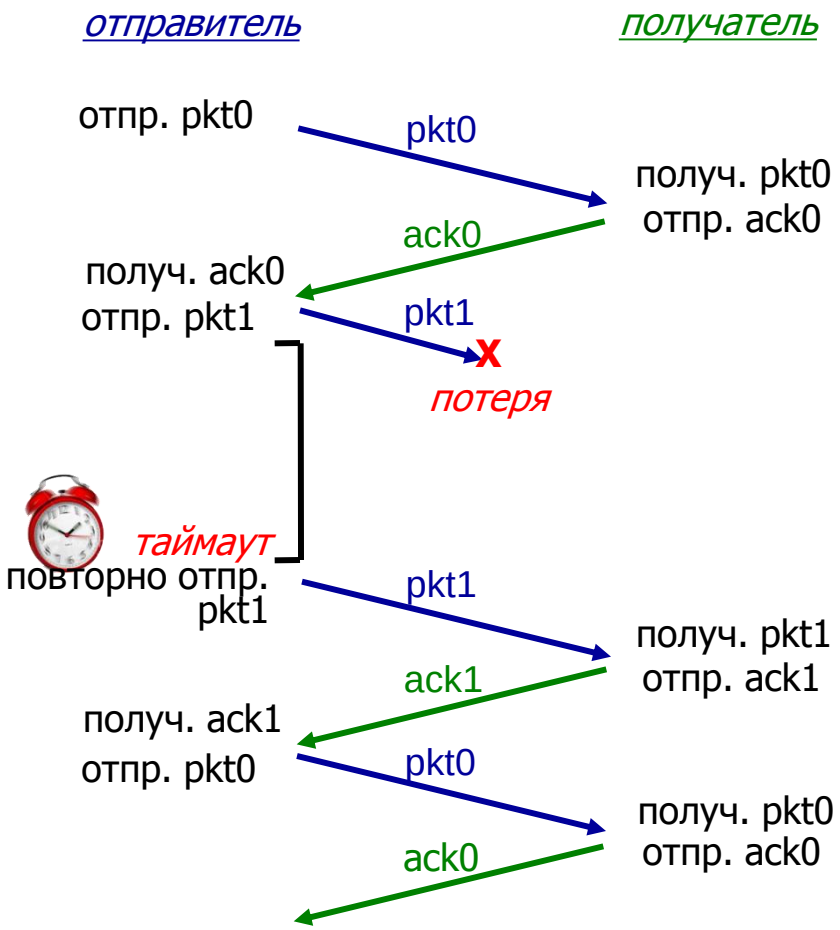
Протокол rdt3.0: FSM-схема отправителя



Протокол rdt3.0 в действии

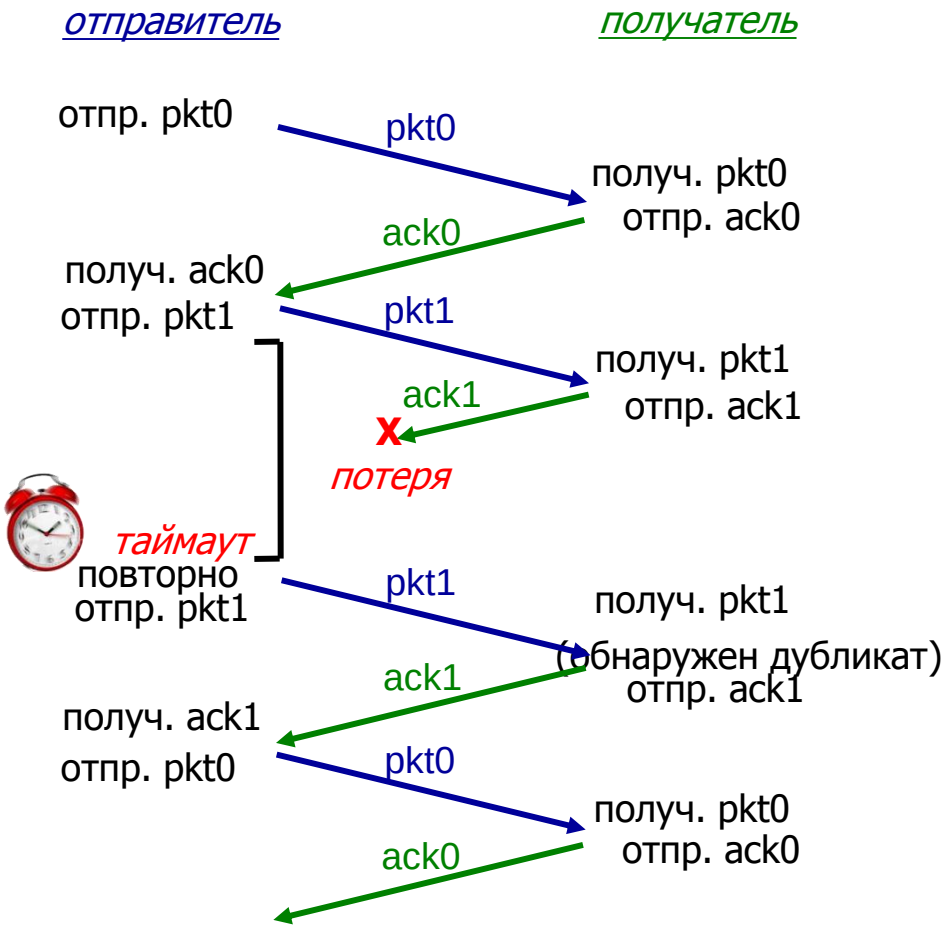


(а) Нет потерь

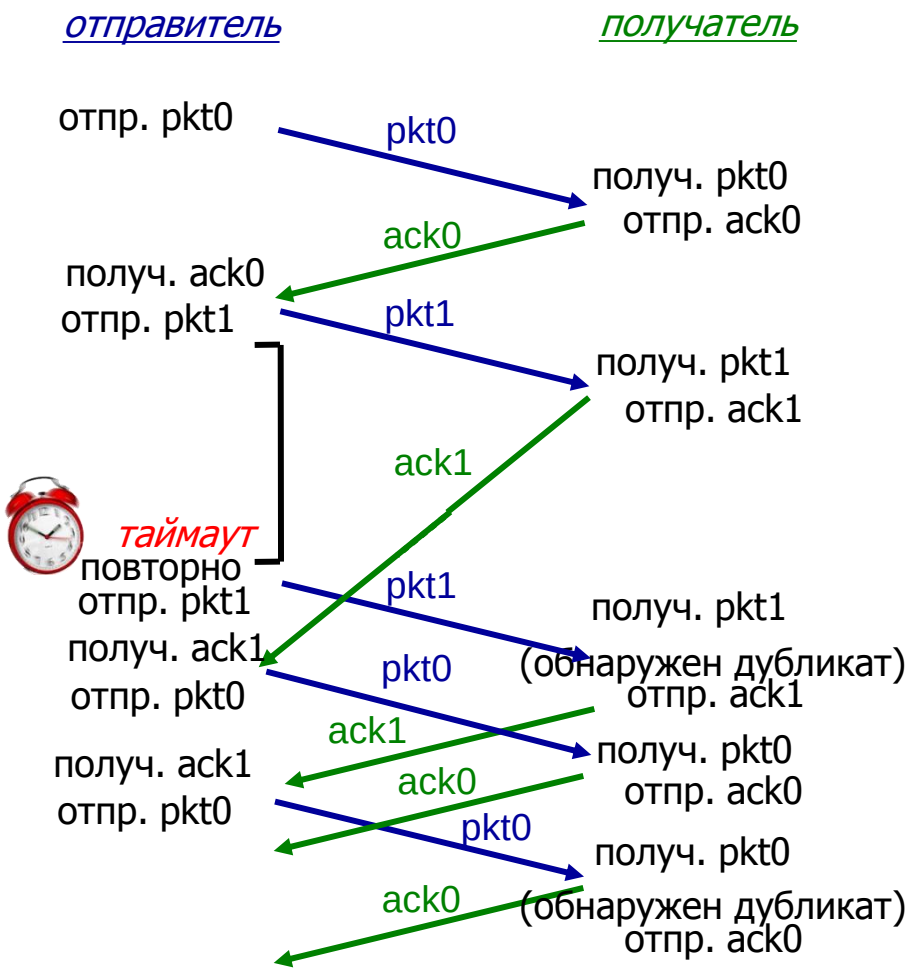


(б) Потеря пакета

Протокол rdt3.0 в действии



(в) Потеря ACK



(г) Преждевременный таймаут/ задержка ACK

Производительность протокола `rdt3.0`

- Протокол `rdt3.0` корректный, но производительность снижена
- например: канал 1 Гбит/с, задержка ожидания 15 мс, размер пакета 8000 бит:

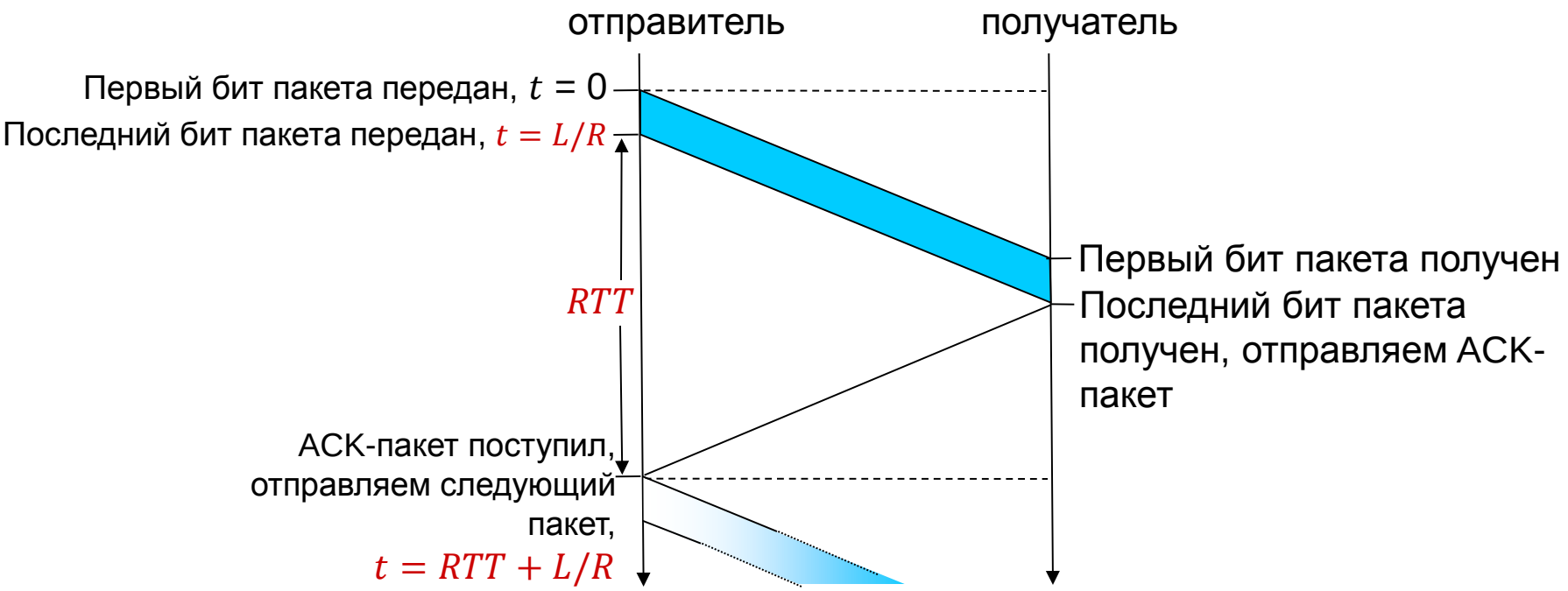
$$D_{\text{передача}} = \frac{L}{R} = \frac{8000 \text{ бит}}{10^9 \text{бит/с}} = 8 \text{ мс}$$

  - $U_{\text{отправитель}}$: *использование* – часть времени, которую отправитель занят отправкой

$$U_{\text{отправитель}} = \frac{L/R}{RTT + L/R} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

  - если RTT=30 мс, пакет 1 Кб каждые 30 мс: пропускная способность 1 Гбит/с канала равна 33 Кбит/с
- Сетевой протокол ограничивает использование физических ресурсов!

Протокол `rdt3.0`: алгоритм с ожиданием подтверждений

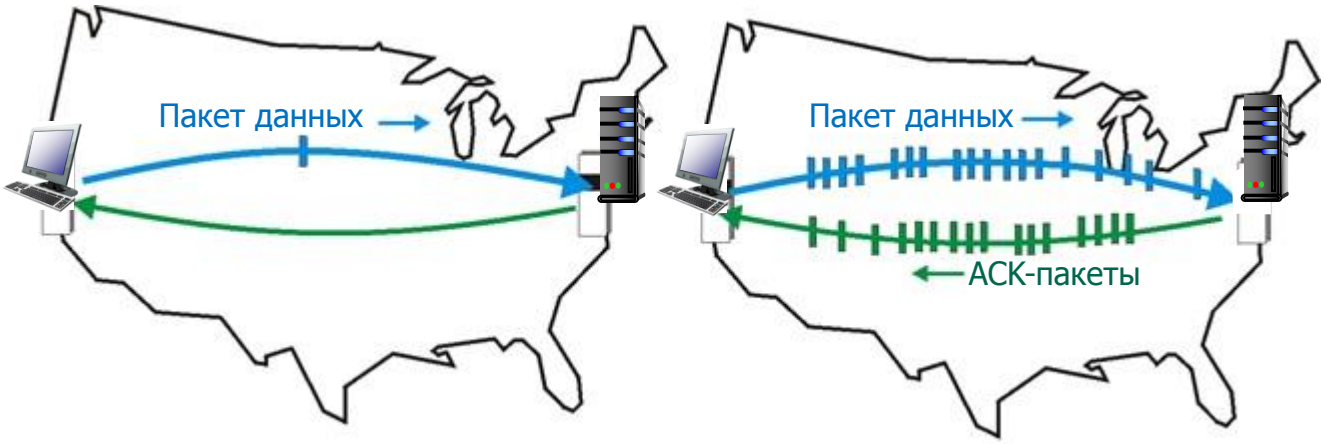


$$U_{\text{отправитель}} = \frac{L/R}{RTT + L/R} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

Протоколы с конвейеризацией

конвейеризация: отправитель может одновременно отправлять несколько пакетов не дожидаясь подтверждений

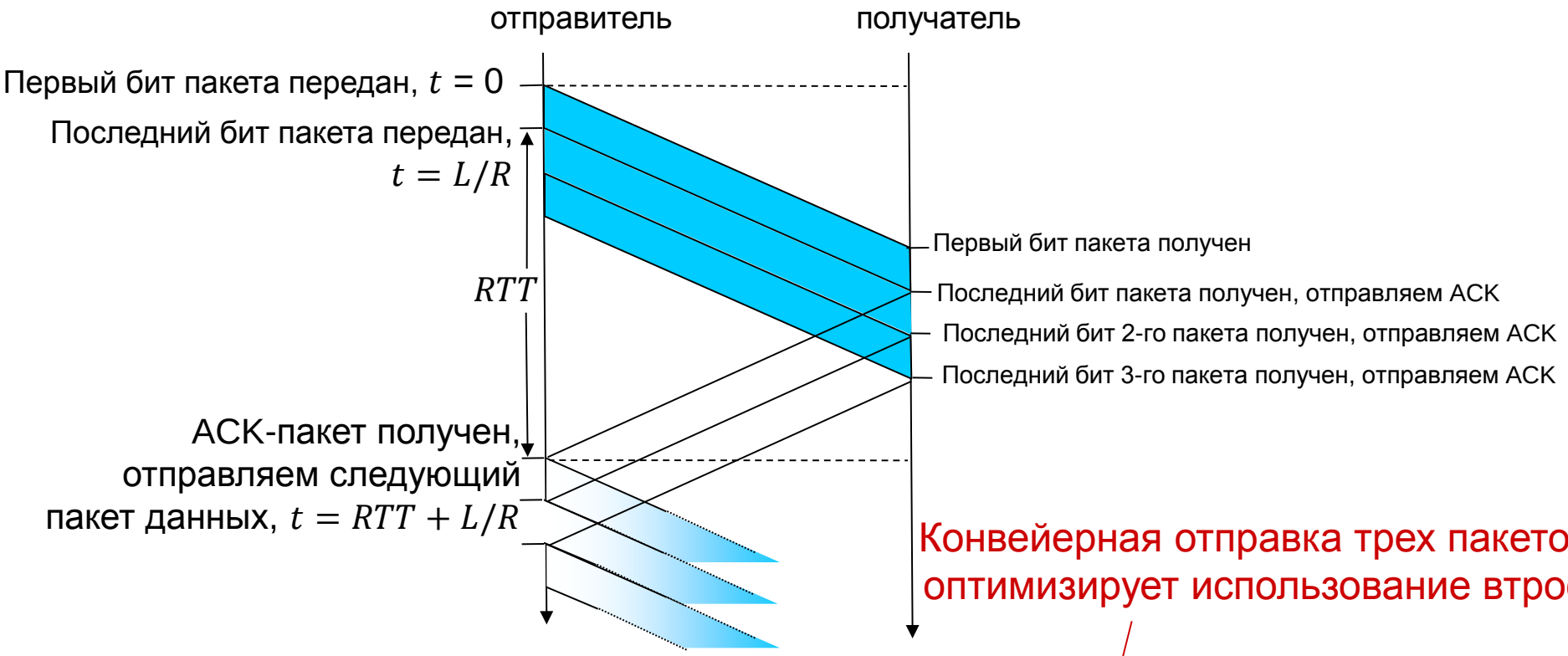
- Диапазон порядковых номеров должен быть увеличен
- Буферизация у отправителя и/или получателя



(а) Протокол с ожиданием подтверждений (б) Протокол с конвейеризацией

- Два базовых вида протокола с конвейеризацией: *на N шагов назад и выборочное повторение*

Конвейеризация: повышение использования канала



Конвейерная отправка трех пакетов оптимизирует использование втрое!

$$U_{\text{отправитель}} = \frac{3L/R}{RTT + L/R} = \frac{0{,}0024}{30{,}008} = 0{,}00081$$

Протоколы с конвейеризацией: обзор

Возврат N шагов назад:

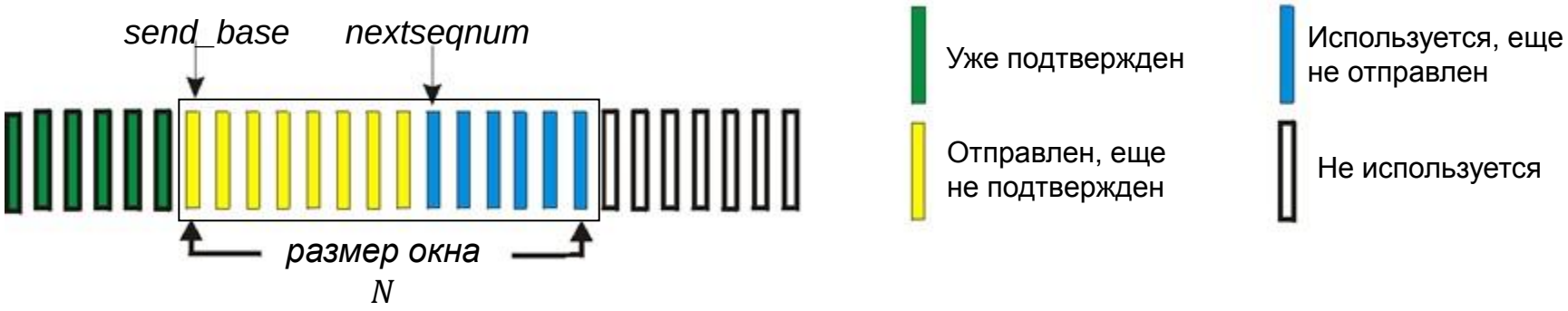
- Отправитель может одновременно предавать до N неподтвержденных пакетов
- Получатель отправляет только *общий ACK-пакет*
  - Не подтверждает пакеты в случае разрыва соединения
- Отправитель использует таймер наиболее раннего неподтвержденного пакета
  - Когда таймер истекает, повторно передаются *все* пакеты

Выборочное повторение:

- Отправитель может одновременно передавать до N неподтвержденных пакетов
- Получатель отправляет *отдельные ACK-пакет для каждого пакета*
- Отправитель использует таймер для каждого неподтвержденного пакета
  - Когда таймер истекает, повторно передаются только неподтвержденные пакеты

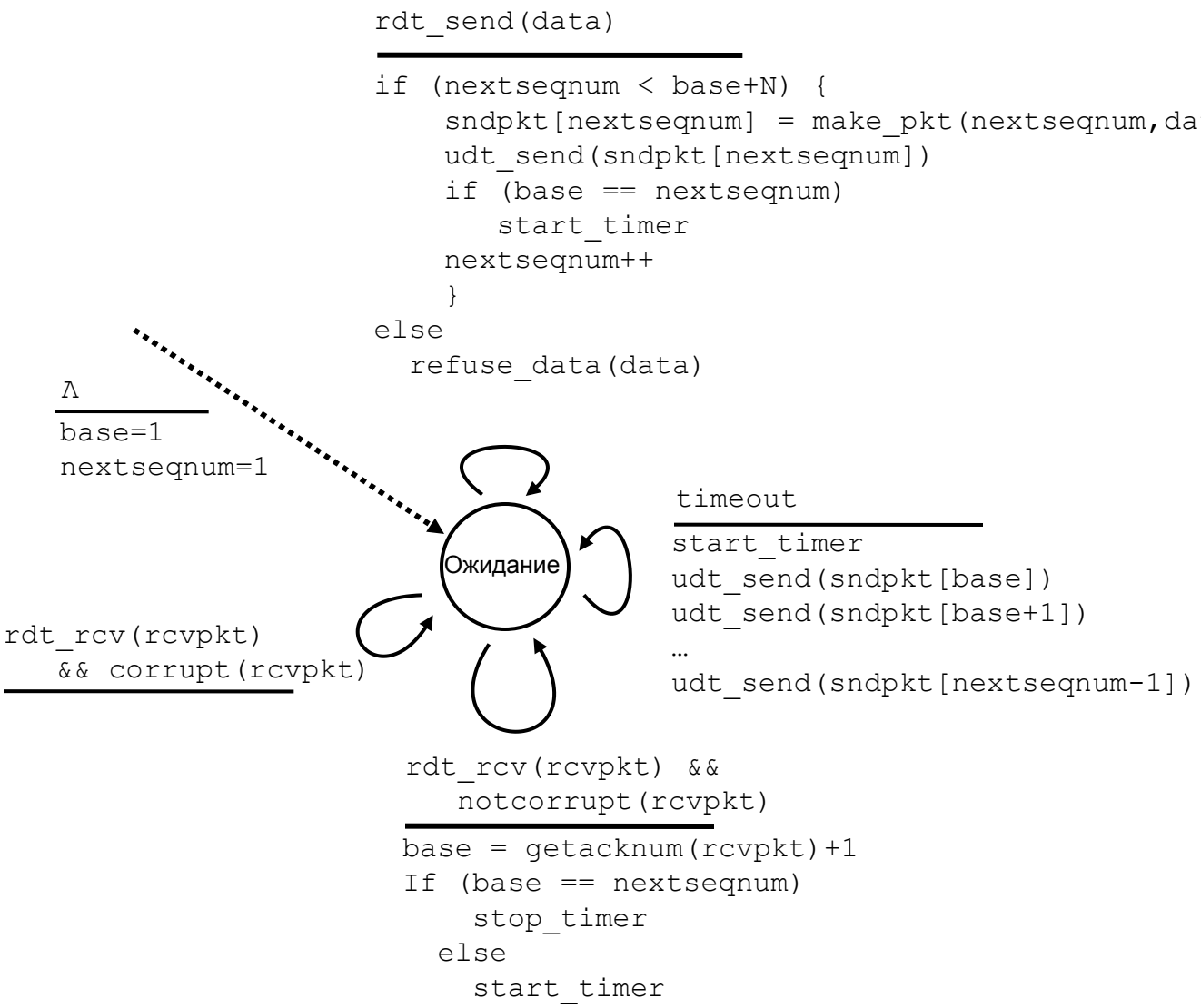
Протокол Go-Back-N, отправитель

- k-разрядный порядковый номер в заголовке пакета
- «окно», вмещающее до N последовательных неподтвержденных пакетов

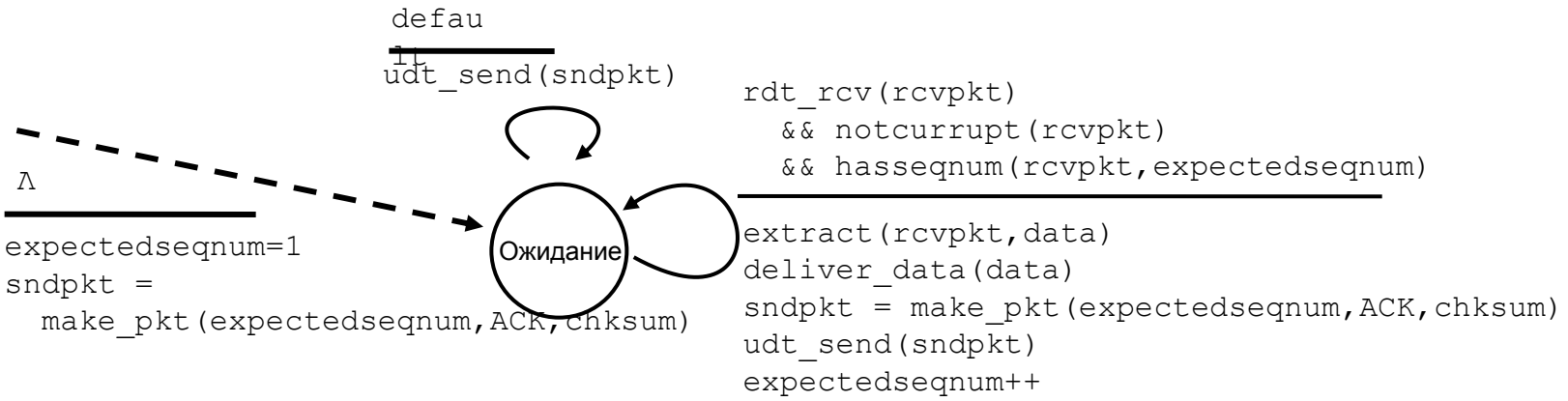


- ACK(n): получены ACK-пакеты на все пакеты данных до пакета с порядковым номером n – *«общий ACK-пакет»*
  - Может получать дублирующие ACK-пакеты (см. описание получателя)
- Таймер для самого раннего передаваемого пакета
- *timeout(n):* повторная передача пакета n и всех пакетов с большими порядковыми номерами в окне

GBN: расширенная FSM-схема отправителя



GBN: расширенная FSM-схема получателя

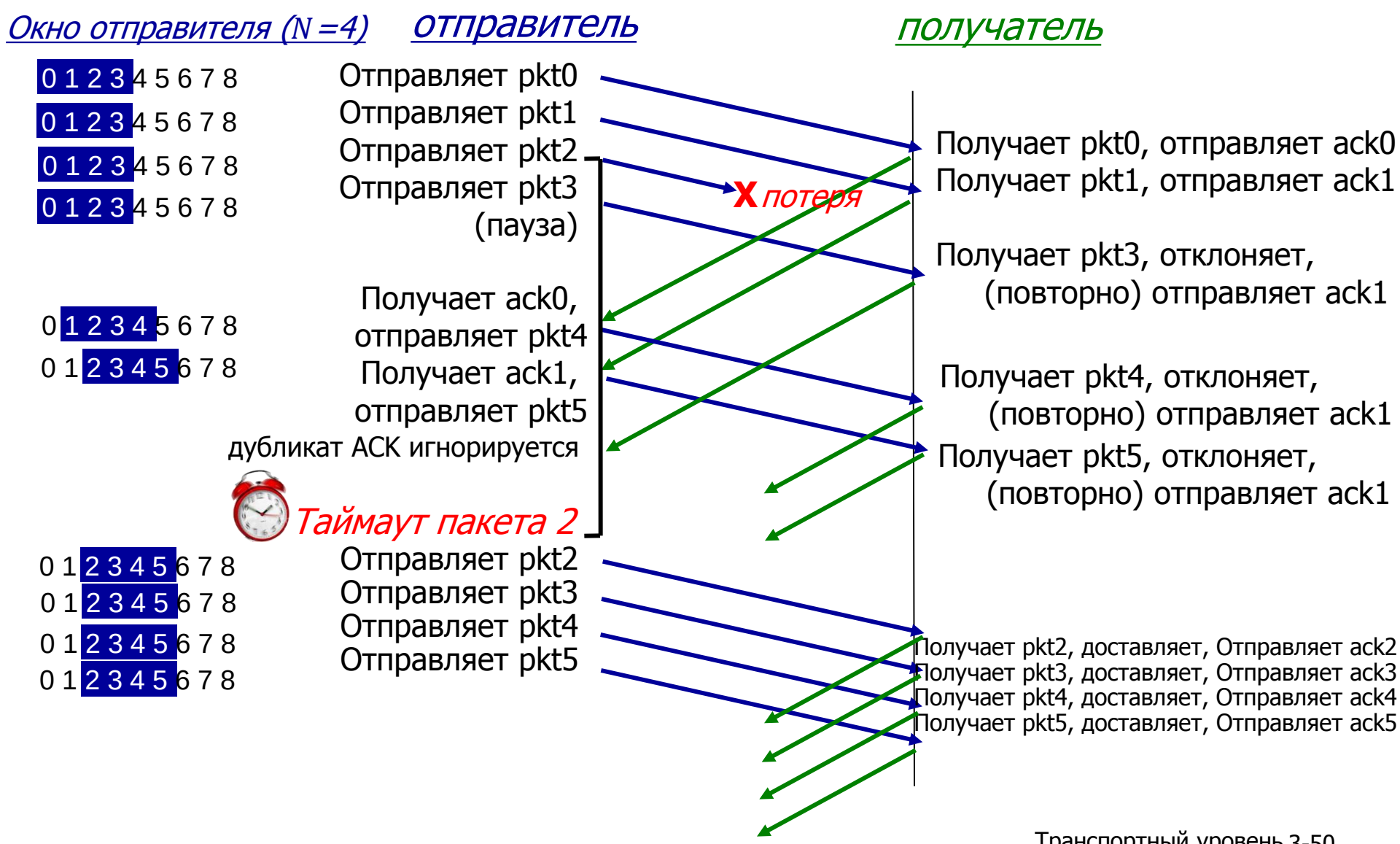


Только ACK: всегда отправляет ACK-пакет в ответ на корректно полученный пакет с наибольшим *идущим по порядку* порядковым номером

- Может создавать дублирующие ACK-пакеты
- Должен хранить только **`expectedseqnum`**

- Идущие не по порядку пакеты:
  - отклоняются (нет буфера): *нет буферизации на стороне получателя!*
  - Повторное подтверждение пакета данных с наибольшим порядковым номером

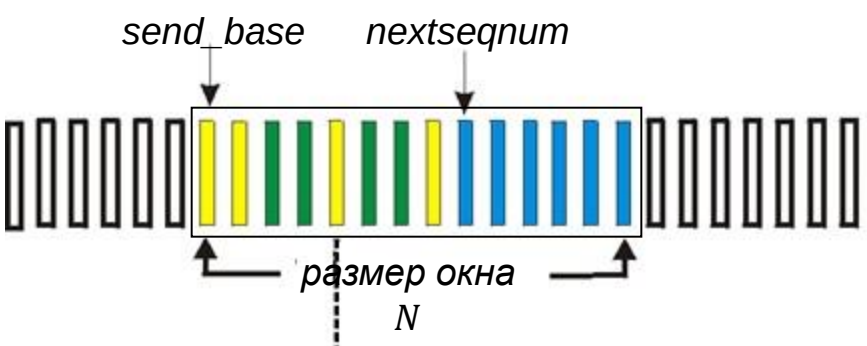
GBN в действии



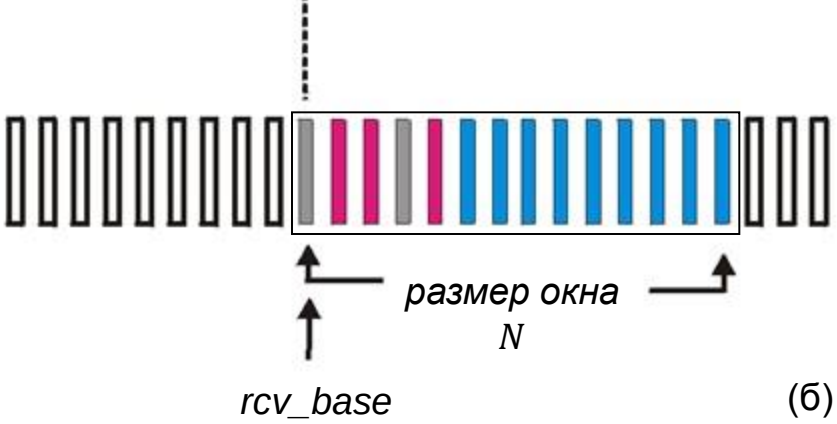
Выборочное повторение

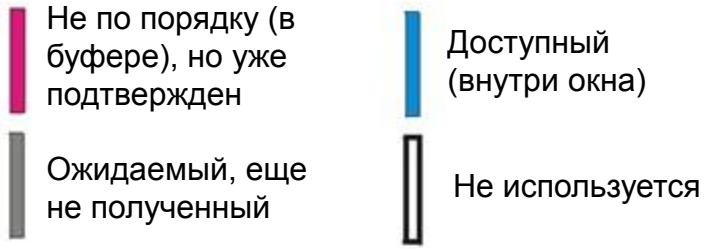
- Получатель *отдельно* подтверждает все корректно полученные пакеты
  - Буферы пакетов необходимы для упорядоченной доставки данных на верхний уровень
  - Отправитель повторно отправляет только пакеты, на которые не были получены ACK-подтверждения
  - Отправитель использует таймер для каждого неподтвержденного пакета
- Окно получателя
  - *N* последовательных порядковых номеров
  - Ограничение количества порядковых номеров отправленных, но не подтвержденных пакетов

Выборочное повторение: окна отправителя и получателя



(а) Представление порядковых номеров отправителем





(б) Представление порядковых номеров получателем

Выборочное повторение

отправитель

Данные сверху:

- Если следующий доступный порядковый номер попадает в окно, отправить пакет

таймаут(ы):

- Переслать пакет n, перезапустить таймер

ACK-пакет в диапазоне $[sendbase, sendbase + N]$:

- Отметить пакет n, как полученный
- если n – наименьший номер подтвержденного пакета, сдвинуть начало окна до следующего неподтвержденного пакета

получатель

пакет n в диапазоне $[rcvbase, rcvbase + N - 1]$

- отправить ACK(n)
- Не по порядку: в буфер
- По порядку: доставить (также доставить все идущие по порядку пакеты из буфера), сдвинуть окно до следующего еще не подтвержденного пакета

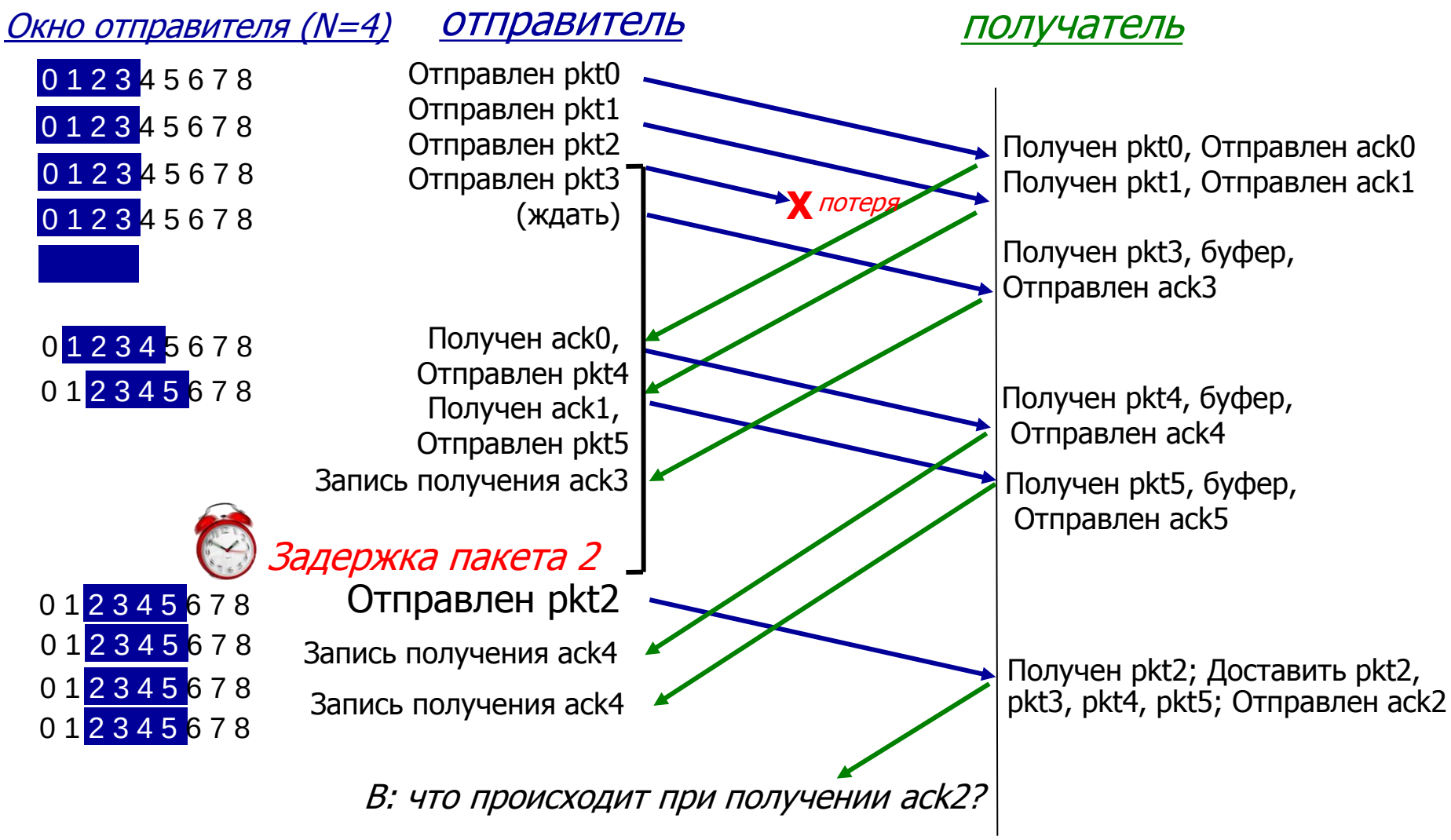
пакет n в диапазоне $[rcvbase - N, rcvbase - 1]$

- ACK(n)

иначе:

- игнорировать

Выборочное повторение в действии

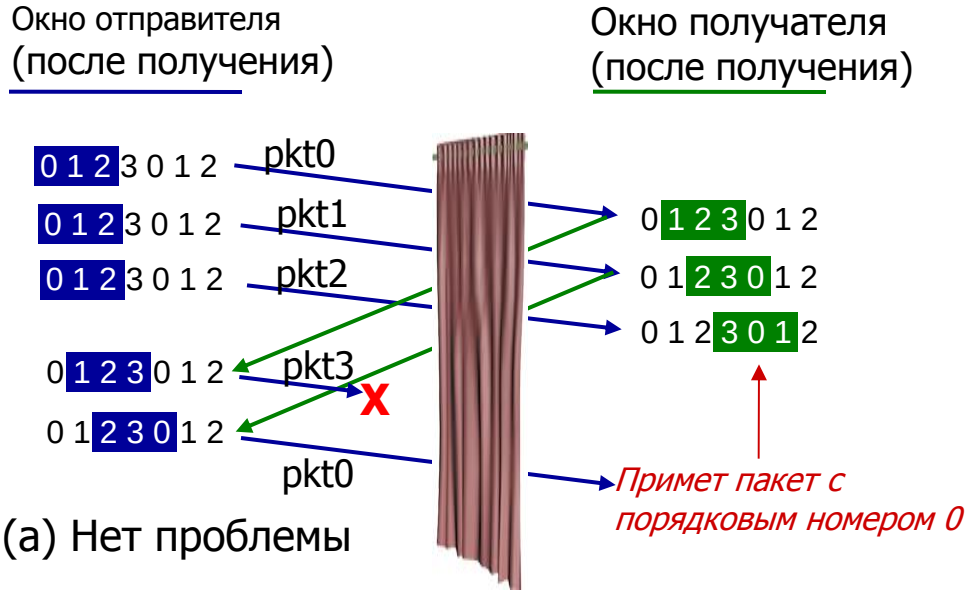


Выборочное повторение: дилемма

пример:

- Порядковые номера: 0, 1, 2, 3
- Размер окна=3
- Получатель не видит разницы двух сценариев!
- Дублированные данные воспринимаются как новые (б)

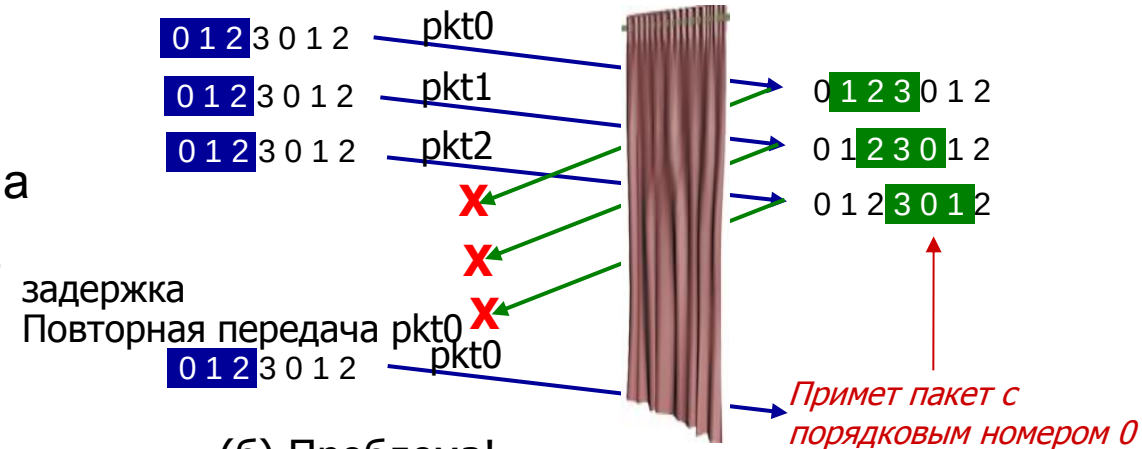
В: как отношение между величиной порядкового номера и размером окна приводит к проблеме, показанной на рис. (б)?



(а) Нет проблемы

Получатель не может видеть сторону отправителя. Поведение получателя одинаково в каждом случае!

Что-то (очень) плохо!



(б) Проблема!

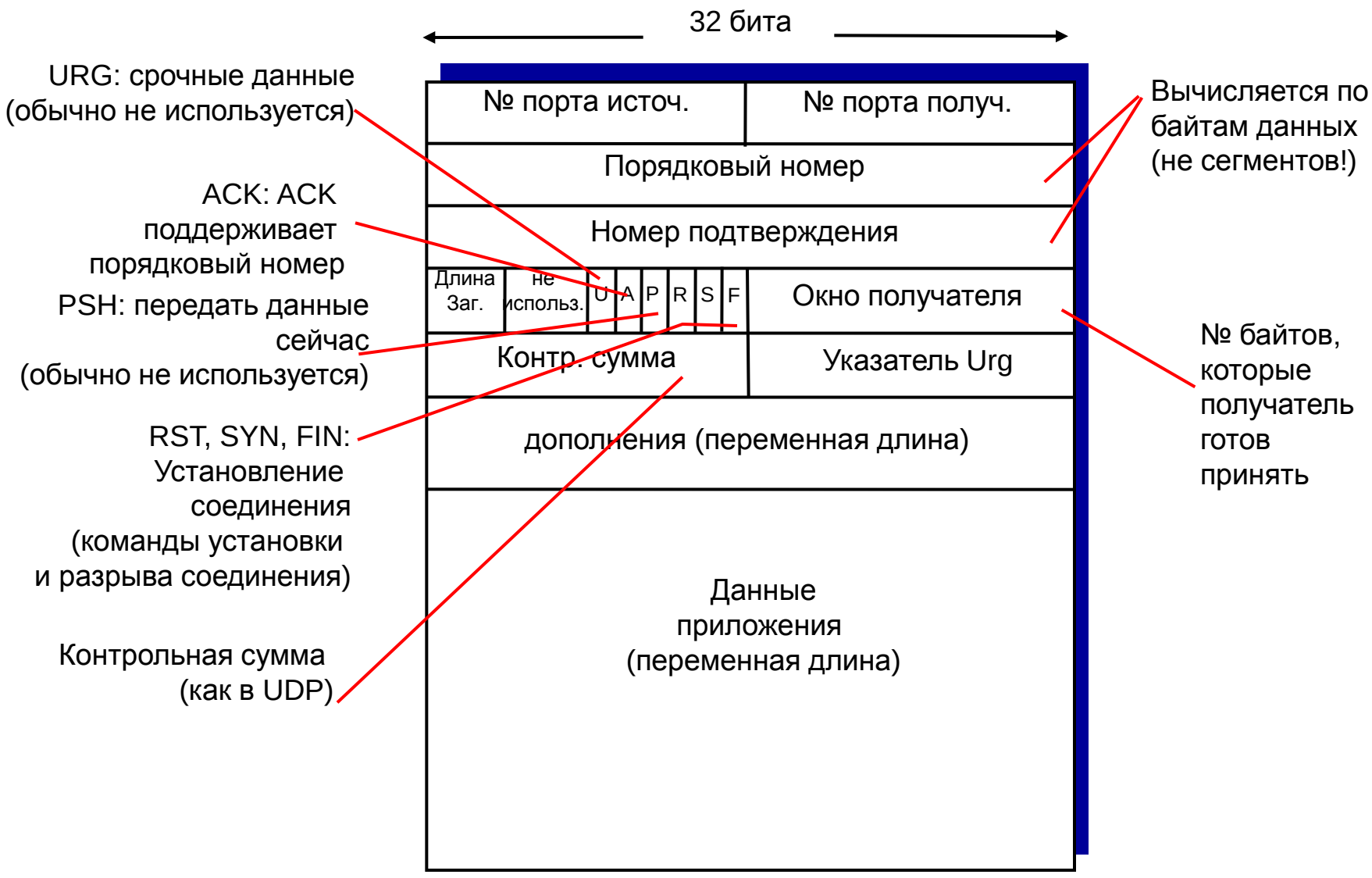
Глава 3: План

3.1 Службы транспортного уровня
3.2 Мультиплексирование и демультиплексирование
3.3 Передача данных без установления логического соединения: протокол UDP
3.4 Принципы надежной передачи данных
3.5 Передача данных с установлением логического соединения: TCP
3.6 Принципы управления перегрузками
3.7 Механизм управления перегрузками протокола TCP

Протокол TCP, обзор RFC: 793,1122,1323, 2018, 2581

- Соединение точка-точка:
  - Один отправитель, один получатель
- Надежный упорядоченный *поток байт:*
  - нет «границ сообщений»
- конвейеризация:
  - Управление перегрузкой и потоком TCP устанавливают размер окна
- Полнодуплексная передача данных:
  - Двунаправленный поток данных в одном соединении
  - MSS: максимальный размер сегмента
- С установлением логического соединения:
  - Установление соединения (обмен управляющими сообщениями) инициируется отправителем, состояние получателя до обмена данными
- Управляемый поток:
  - Отправитель не вызывает переполнение у получателя

Структура TCP-сегмента



Протокол TCP: порядковые номера, ACK-пакеты

Порядковые номера:

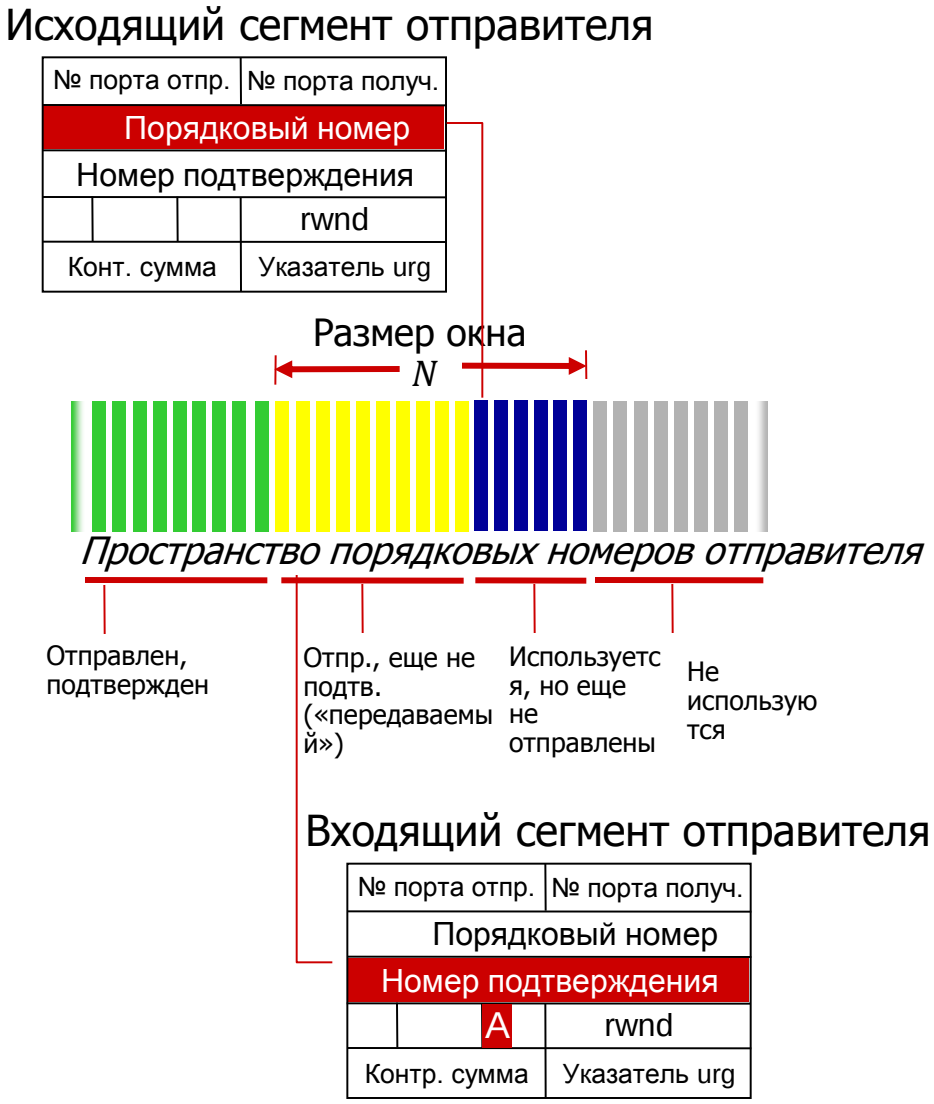
- «номер» первого байта данных сегмента в потоке байт

подтверждения:

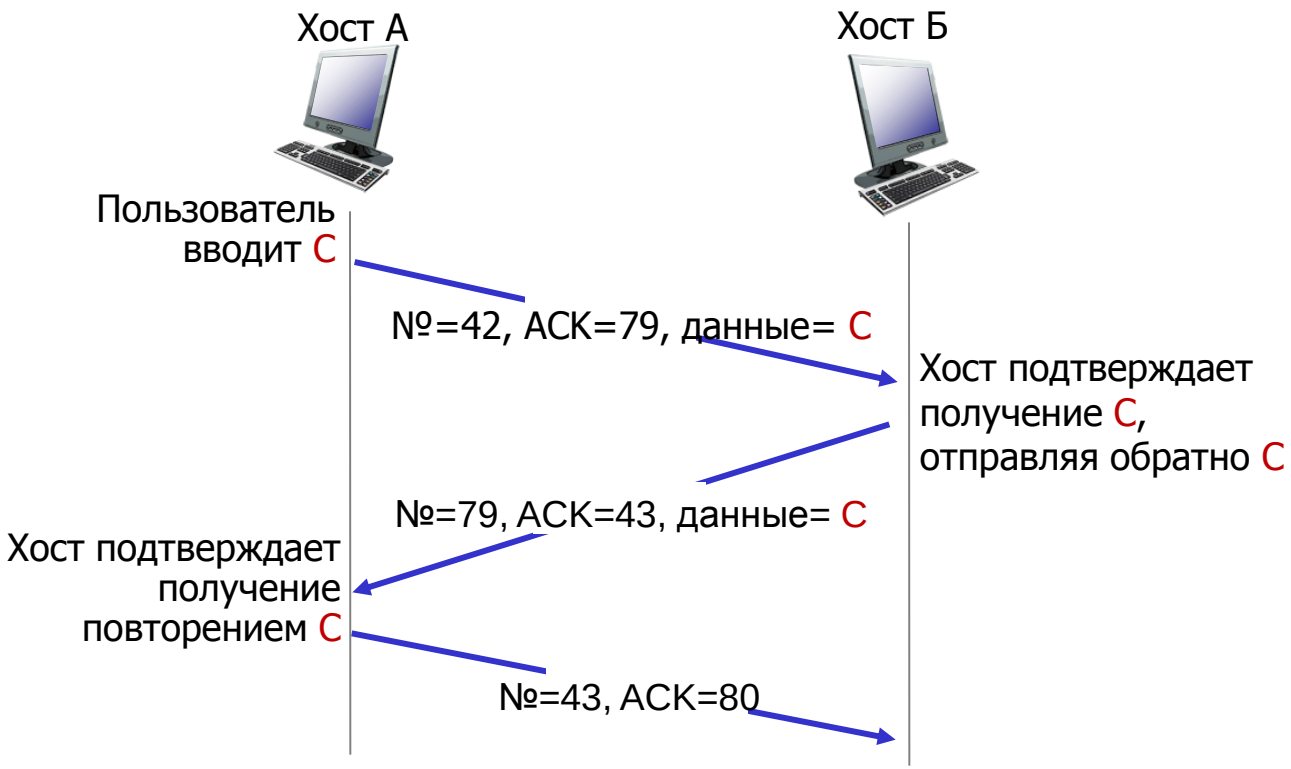
- порядковый номер следующего байта ожидаемого от второй стороны
- Общее подтверждение

B: как получатель обрабатывает полученные не по порядку сегменты

- *O*: спецификация протокола TCP не регламентирует, ответ зависит от конкретной реализации



Протокол TCP: порядковые номера, ACK-пакеты



Простой пример telnet

Протокол TCP: время оборота, таймаут

В: как устанавливается значение таймаута в TCP?

- длиннее, чем RTT
  - но RTT изменяется
- *Слишком короткий:* преждевременный таймаут, ненужные повторные передачи
- *Слишком долгий:* медленная реакция на потерю сегментов

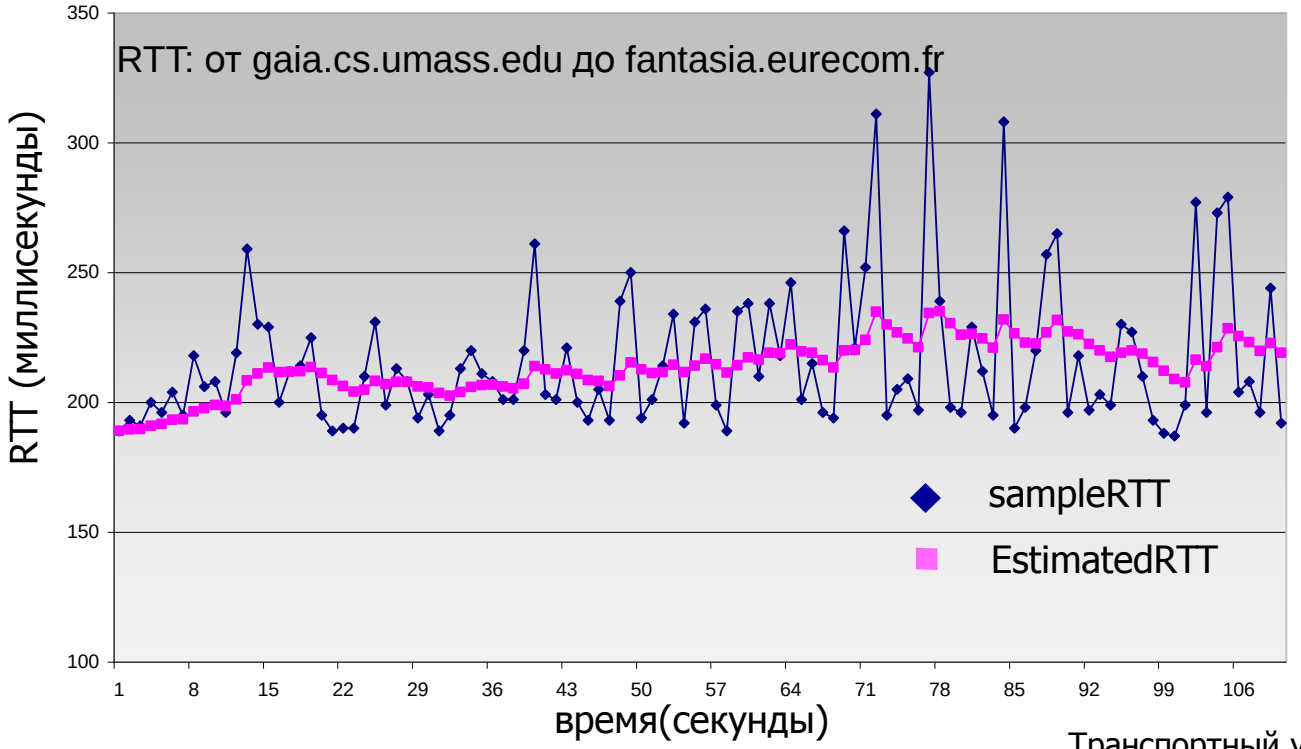
В: как вычисляется RTT?

- **`SampleRTT`**: измеряет время с момента передачи сегмента до получения ACK-пакета
  - Игнорирует повторные передачи
- **`SampleRTT`** будет меняться, желательно «сглаженное» измерение RTT
  - Среднее нескольких *недавних* измерений, а не только текущего **`SampleRTT`**

Протокол TCP: время оборота, таймаут

$$\boldsymbol{EstimatedRTT} = (1 - \alpha) \times \boldsymbol{EstimatedRTT} + \alpha \times \boldsymbol{SampleRTT}$$

- экспоненциальное взвешенное скользящее среднее
- Влияние предыдущего значения уменьшается экспоненциально быстро
- Стандартное значение $\alpha = 0{,}125$



Протокол TCP: время оборота, таймаут

- Интервал таймаута: **`EstimatedRTT`** плюс резерв
  - Большие изменения **`EstimatedRTT ->`** больший резерв
- Оценка отклонения SampleRTT от EstimatedRTT:

$$DevRTT = (1-\beta) \times DevRTT + \beta \times |SampleRTT - EstimatedRTT|$$

`(как правило,` β **`=0,25)`**

$$TimeoutInterval = EstimatedRTT + 4 \times DevRTT$$

Измеренное *RTT*

«резерв»

Протокол TCP: надежная передача данных

- Протокол TCP создает надежную службу поверх ненадежной службы протокола IP
  - Конвейеризация отправки сегментов
  - Общие подтверждения
  - Единственный таймер повторной передачи
- Повторная передача может быть вызвана:
  - Событиями таймаута
  - Дублирующими подтверждениями

Для начала рассмотрим упрощенную версию TCP-отправителя:

- Игнорирует дублирующие подтверждения
- Игнорирует управление потоком и перегрузкой

Протокол TCP: события на стороне отправителя

Данные полученные от приложения:

- создать сегмент с порядковым номером
- порядковый номер – номер первого байта данных в сегменте
- запустить таймер, если еще не был запущен
  - таймер для самого раннего неподтвержденного сегмента
  - интервал таймаута: **`TimeOutInterval`**

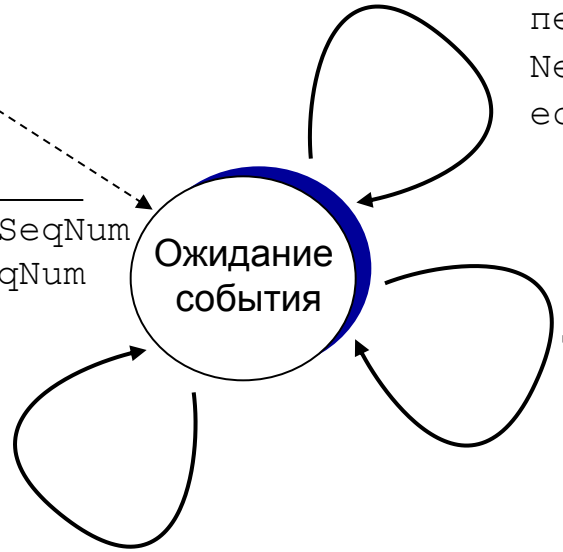
таймаут:

- повторно передать сегмент, который вызвал таймаут
- перезапустить таймер

получено подтверждение:

- если ACK-пакет подтверждает ранее неподтвержденные сегменты
  - обновить то, что уже подтверждено
  - запустить таймер, если еще есть неподтвержденные сегменты

Упрощенная FSM-схема TCP-отправителя



данные получены сверху от приложения

создать сегмент, пор. №: NextSeqNum
передать сегмент IP (т.е. "отправить")
NextSeqNum = NextSeqNum + length(data)
если (еще не был запущен таймер)
запустить таймер

Λ

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

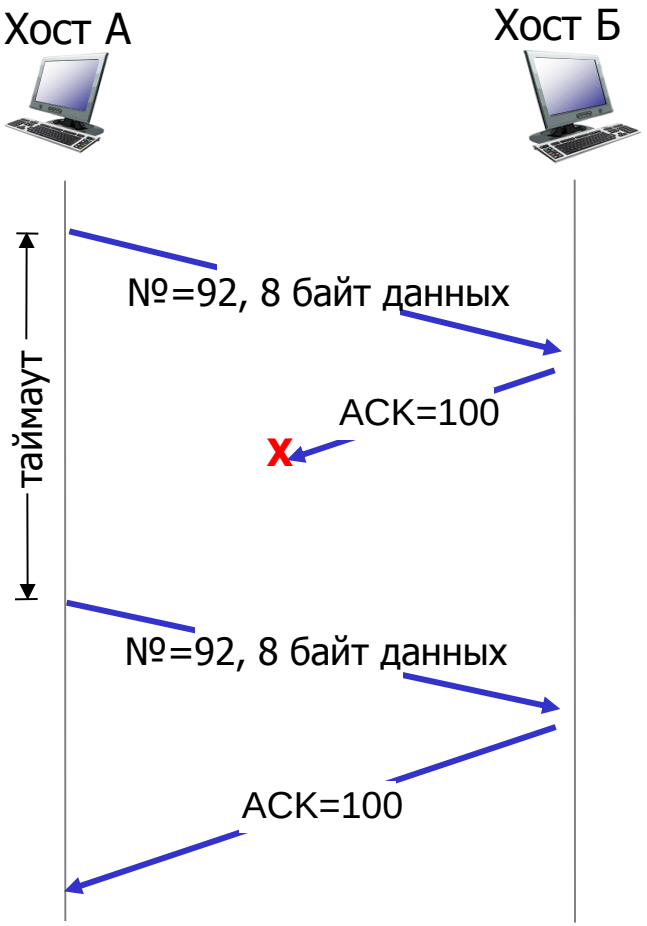
таймаут

повторно передать еще не
подтвержденный сегмент с
наименьшим порядковым номером
запустить таймер

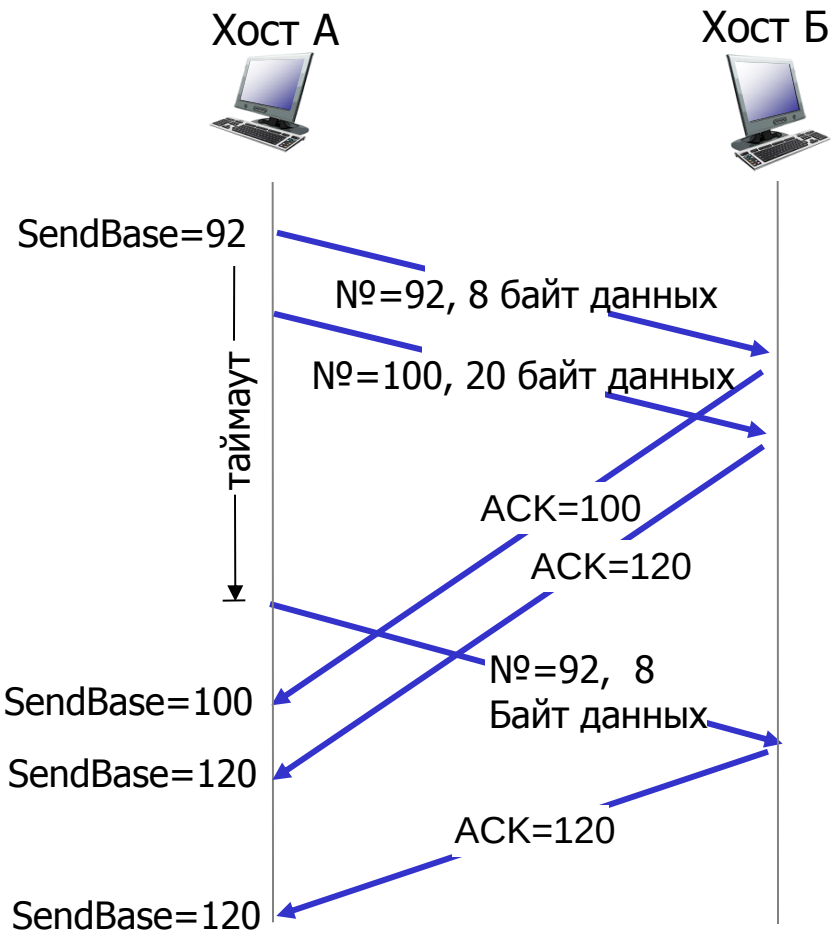
получен ACK, со значением y поля ACK

```
если (y > SendBase) {
    SendBase = y
    /* SendBase-1: байт последнего общего ACK-пакета */
    если (существуют еще не подтвержденные сегменты)
        запустить таймер
      иначе остановить таймер
     }
```

Протокол TCP: сценарии повторной передачи

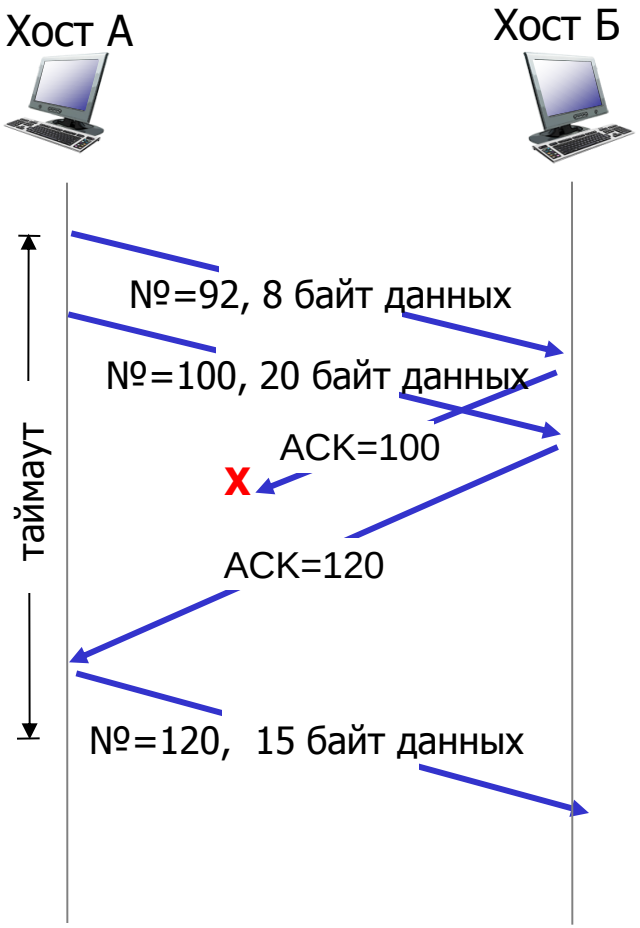


Сценарий потери ACK-пакета



Преждевременный таймаут

Протокол TCP: сценарии повторной передачи



Общий ACK-пакет

Протокол TCP формирование ACK-пакета [RFC 1122, RFC 2581]

Событие получателя	*Действие TCP-получателя*
поступил сегмент в верном порядке с ожидаемым порядковым номером. Все данные до этого номера подтверждены	Задерживает ACK-пакет. Ожидает 500мс до следующего сегмента. Если следующего сегмента нет, отправляет ACK-пакет.
поступил сегмент в верном порядке с ожидаемым порядковым номером. Еще один сегмент ожидает отправку ACK-пакета	Незамедлительно отправляет общий ACK-пакет, подтверждая оба сегмента
Поступил сегмент в неверном порядке с порядковым номером превышающим Ожидаемый. Обнаружен разрыв	незамедлительно отправляет *дублирующий ACK*, указывает порядковый номер следующего ожидаемого сегмента
поступил сегмент, частично или полностью покрывающий разрыв	незамедлительно отправляет ACK-пакет, при условии, что сегмент соответствует нижнему краю разрыва

Протокол TCP: ускоренная повторная передача

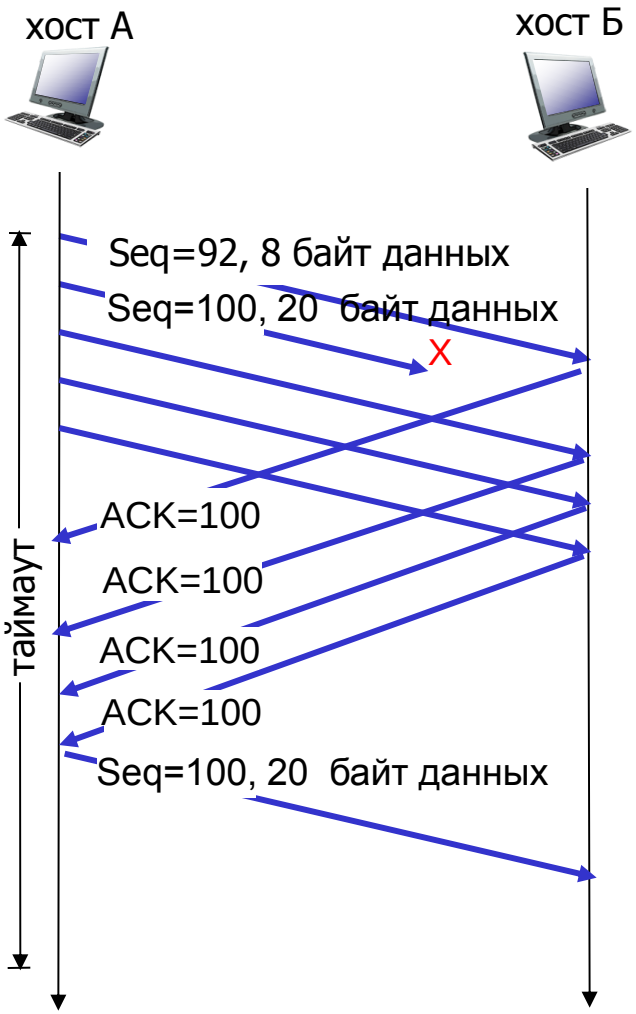
- Интервал ожидания часто относительно длинный:
  - Длительная задержка до повторной передачи потерянного пакета
- Обнаружение потери сегмента по дублирующим ACK-пакетам.
  - Отправитель часто последовательно передает несколько сегментов
  - Если сегмент потерян, это вызовет получение множества дублирующих ACK-пакетов.

Ускоренная повторная передача TCP

Если отправитель получил 3 ACK-пакета на одни данные ("утроенный ACK-пакет"), повторно отправить неподтвержденный сегмент с меньшим порядковым номером

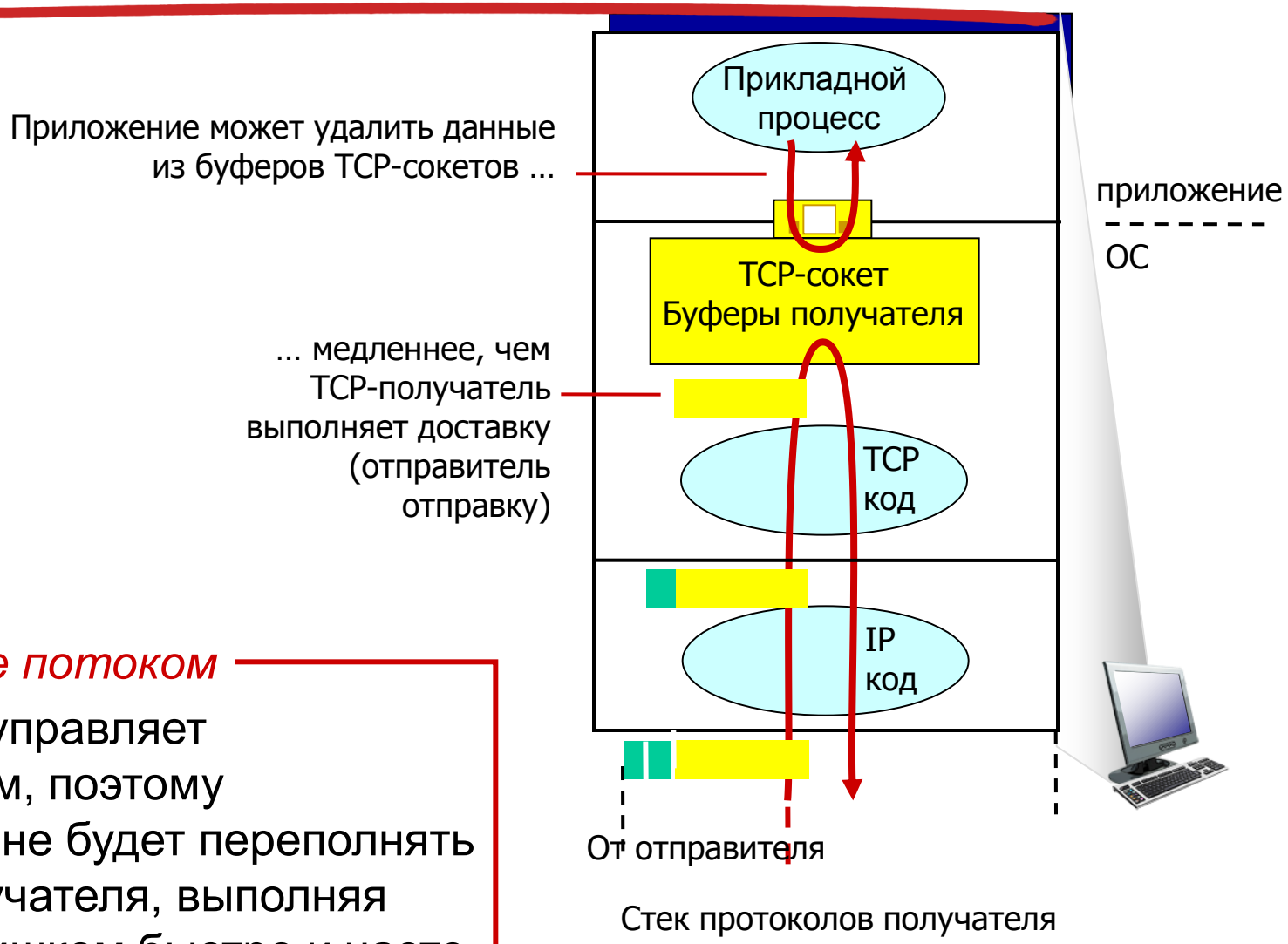
- То есть не дожидаться таймаута до повторной отправки потерянного сегмента

Протокол TCP: ускоренная повторная передача



Ускоренная повторная передача после получения отправителем трех дублирующих ACK-пакетов

Управление потоком TCP

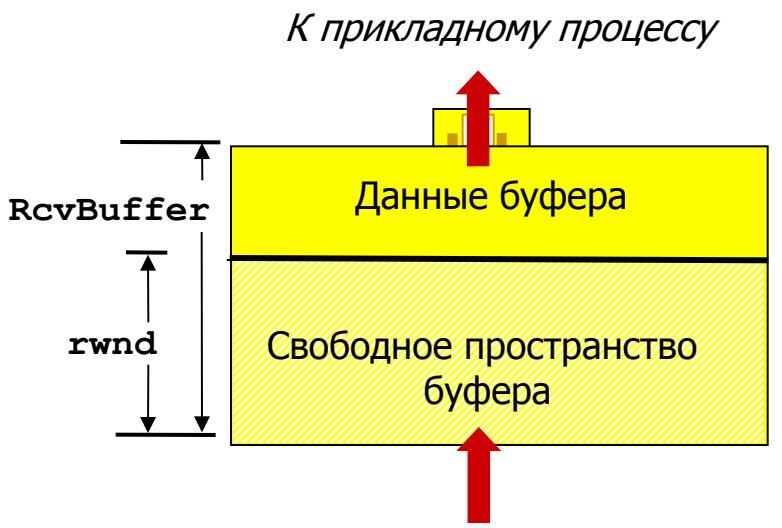


Управление потоком

Получатель управляет отправителем, поэтому отправитель не будет переполнять буферы получателя, выполняя передачу слишком быстро и часто

Управление потоком TCP

- Получатель «объявляет» свободное пространство буфера включая значение **`rwnd`** в заголовок TCP-сегмента направленного отправителю
  - **`Размер RcvBuffer`** устанавливается с помощью параметров сокета (по умолчанию обычно 4096 байт)
  - Многие операционные системы автоматически корректируют **`RcvBuffer`**
- Отправитель ограничивает количество неподтвержденных (передаваемых) данных значением получателя **`rwnd`**
- Гарантированное отсутствие переполнения буфера

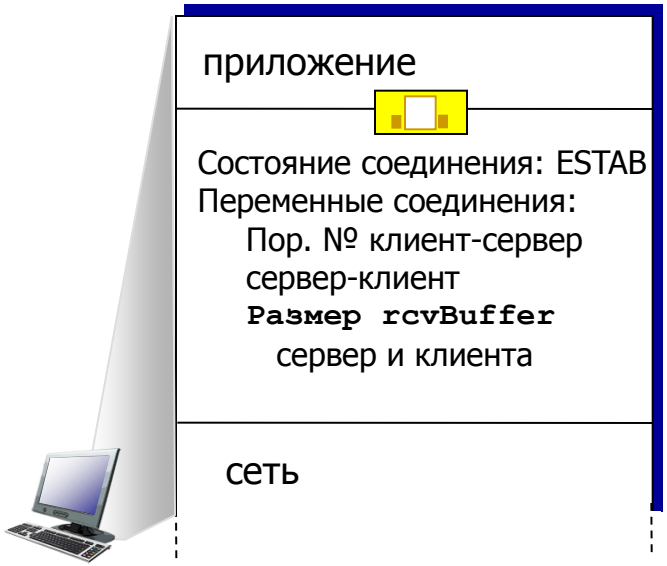


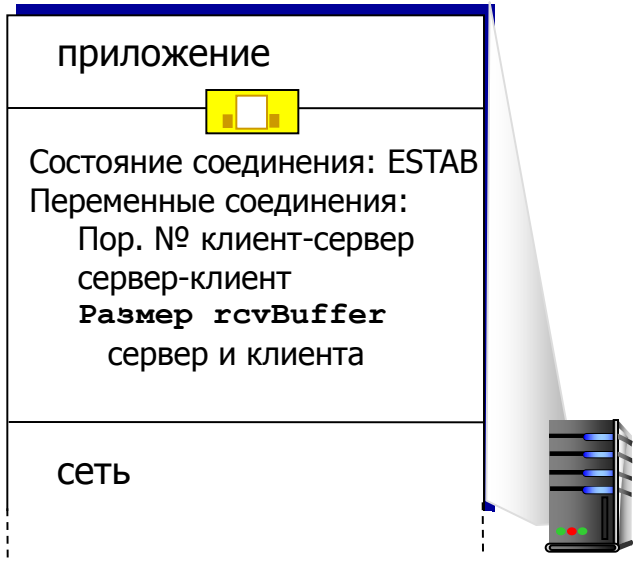
Буферизация на стороне получателя

Управление соединением

Установление соединения перед обменом данными:

- Рукопожатие (каждая сторона знает, что другая хочет установить соединение)
- Согласование параметров соединения



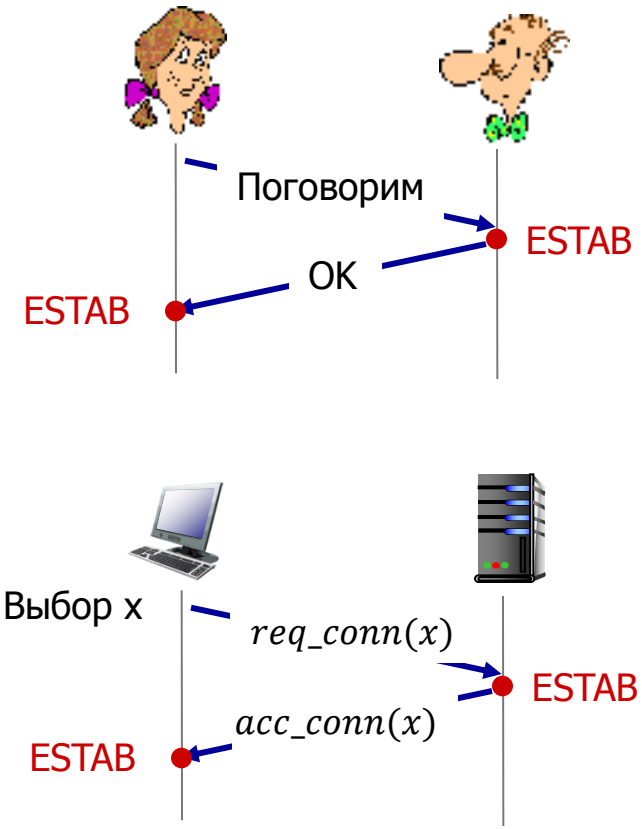



```
Socket clientSocket =
  newSocket("имя_хоста", "номер_порта");
```

```
Socket connectionSocket =
  welcomeSocket.accept();
```

Согласование установления соединения

2-этапное рукопожатие:

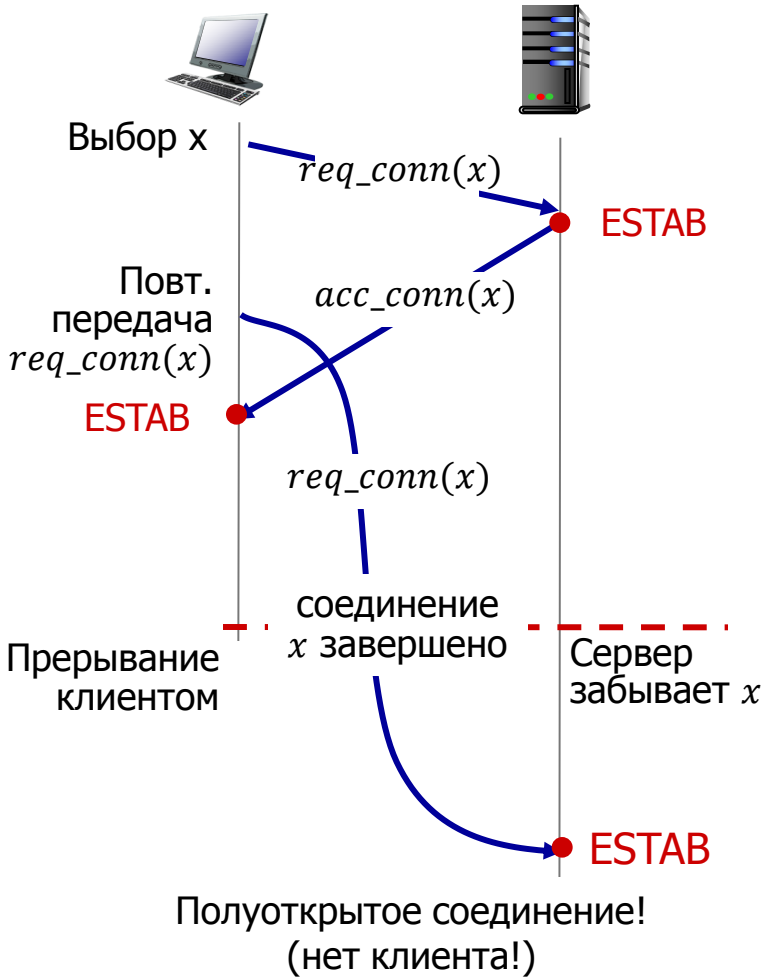


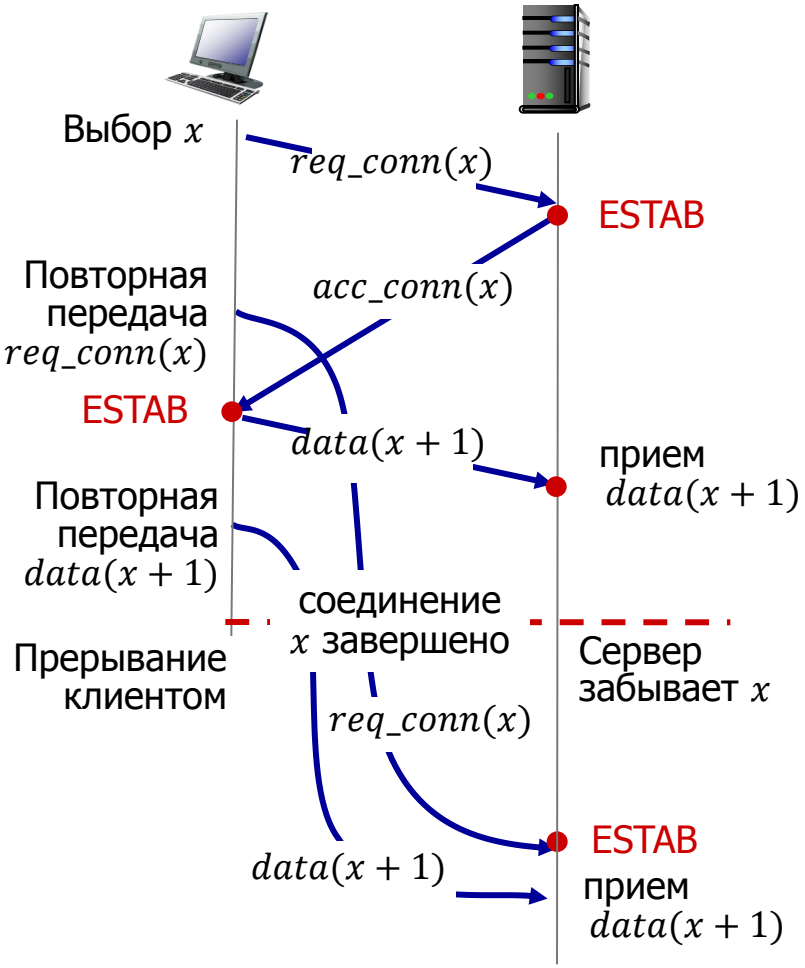
B: всегда ли будет работать двухшаговое рукопожатие в сети?

- Непостоянная задержка
- Повторно передаваемые сообщения (например, $req\_conn(x)$) из-за потери сообщений
- Нарушение порядка сообщений
- Нет возможности «видеть» вторую сторону

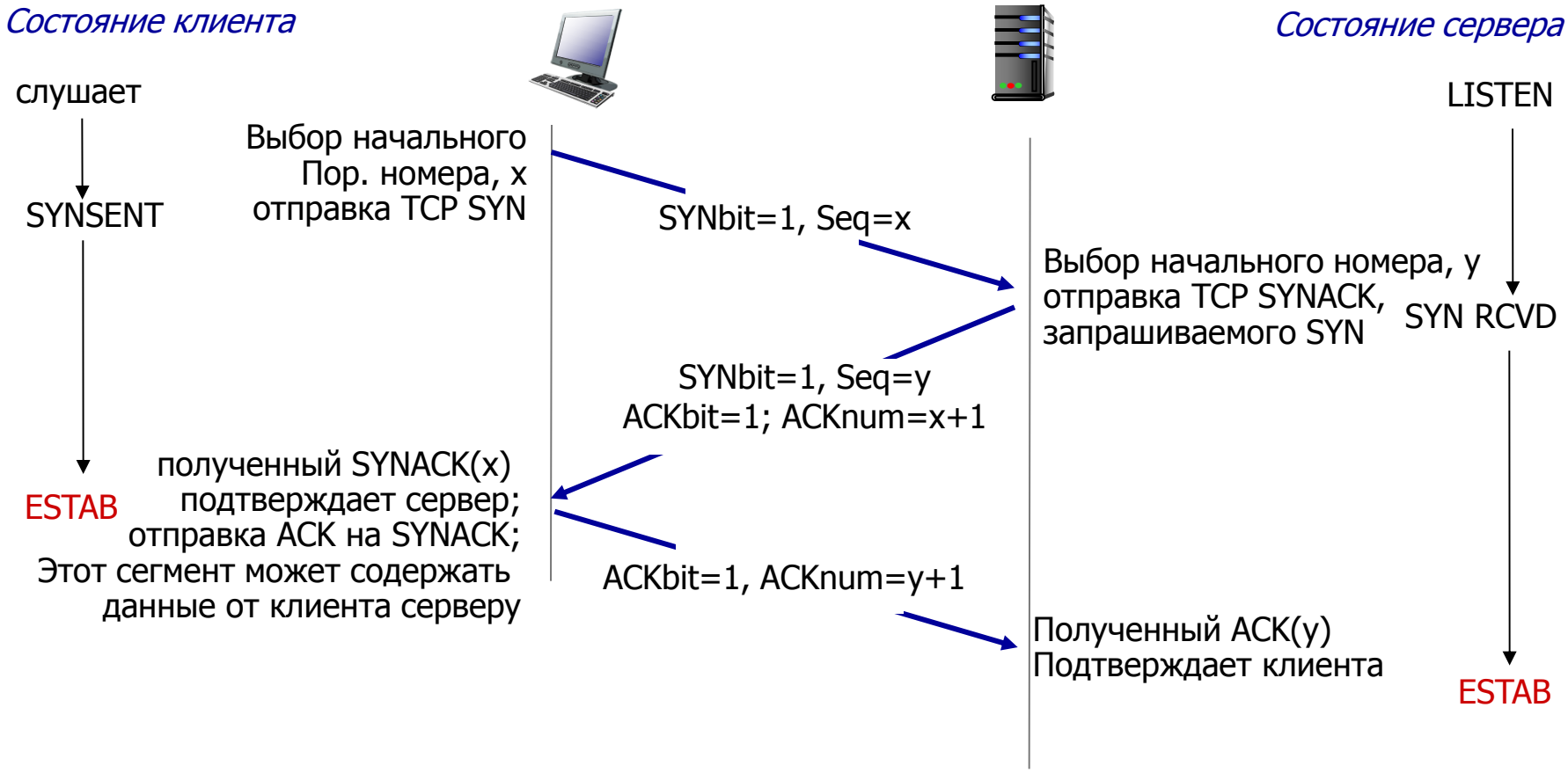
Согласование установления соединения

Двухшаговое рукопожатие, сценарий отказа:

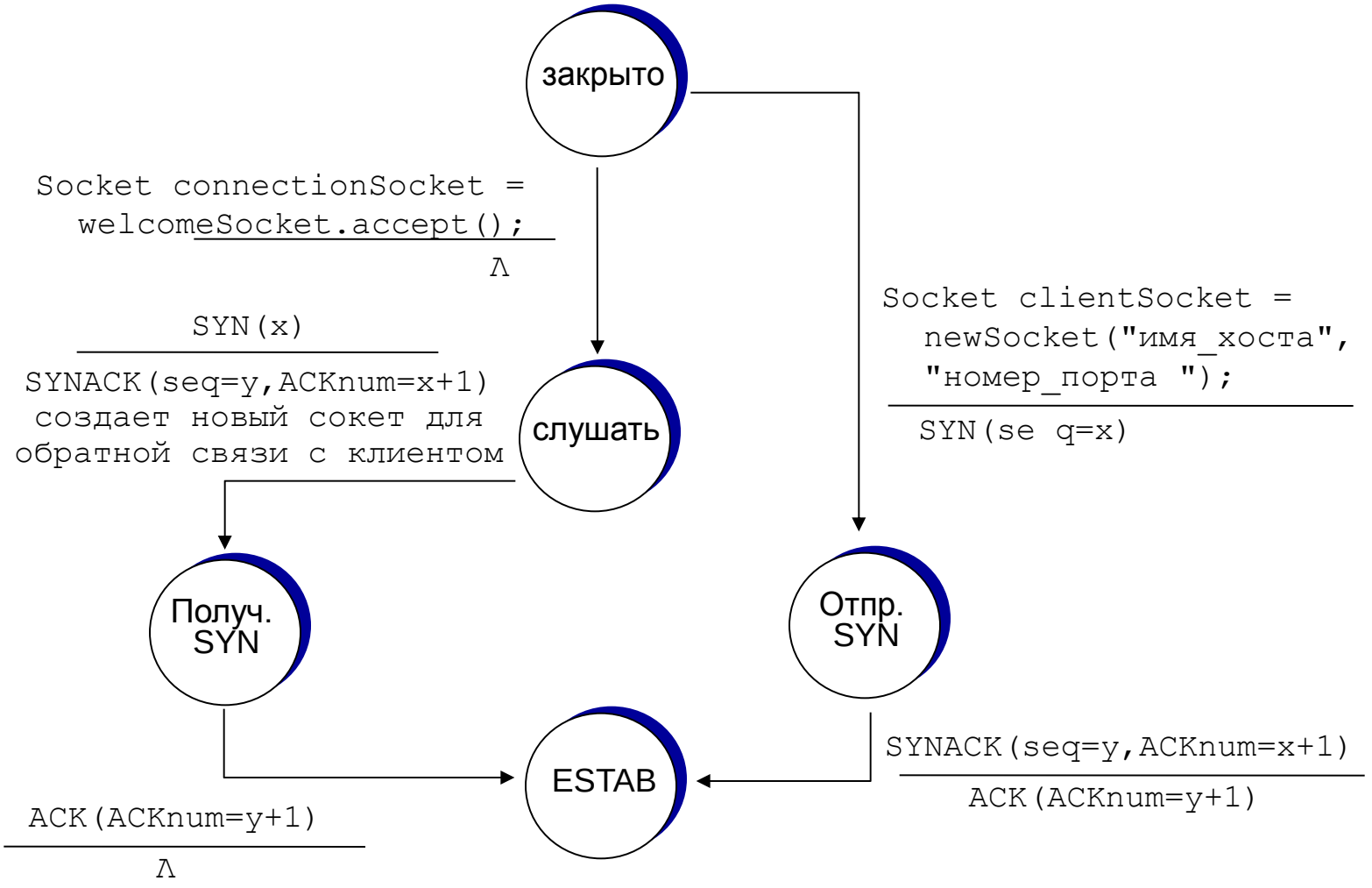




TCP: трехшаговое рукопожатие



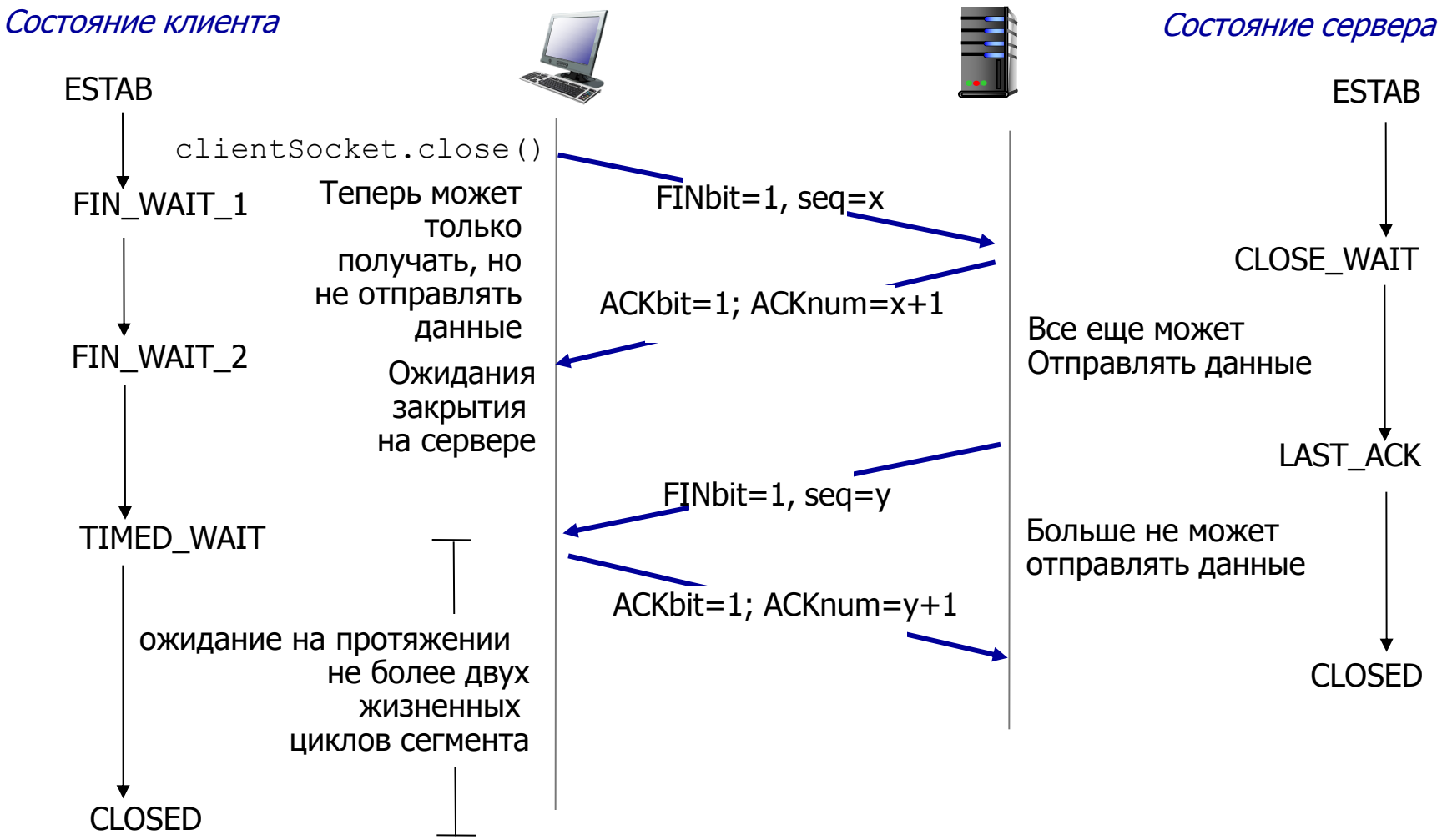
FSM-схема трехшагового установления TCP-соединения



TCP: закрытие соединения

- Клиент и сервер закрывают соединение со своей стороны
  - отправка TCP-сегмента с FIN битом равным 1
- Подтверждение получения FIN получением ACK-пакета
  - При получении FIN, ACK-пакет может быть объединен с собственным FIN
- Одновременный обмен FIN может быть обработан

TCP: закрытие соединения



Глава 3: План

3.1 Службы транспортного уровня

3.2 Мультиплексирование и демультиплексирование

3.3 Передача данных без установления логического соединения: протокол UDP

3.4 Принципы надежной передачи данных

3.5 Передача данных с установлением логического соединения: TCP

- Структура сегмента
- Надежная передача данных
- Управление потоком
- Управление соединением

3.6 Принципы управления перегрузками

3.7 Механизм управления перегрузками протокола TCP

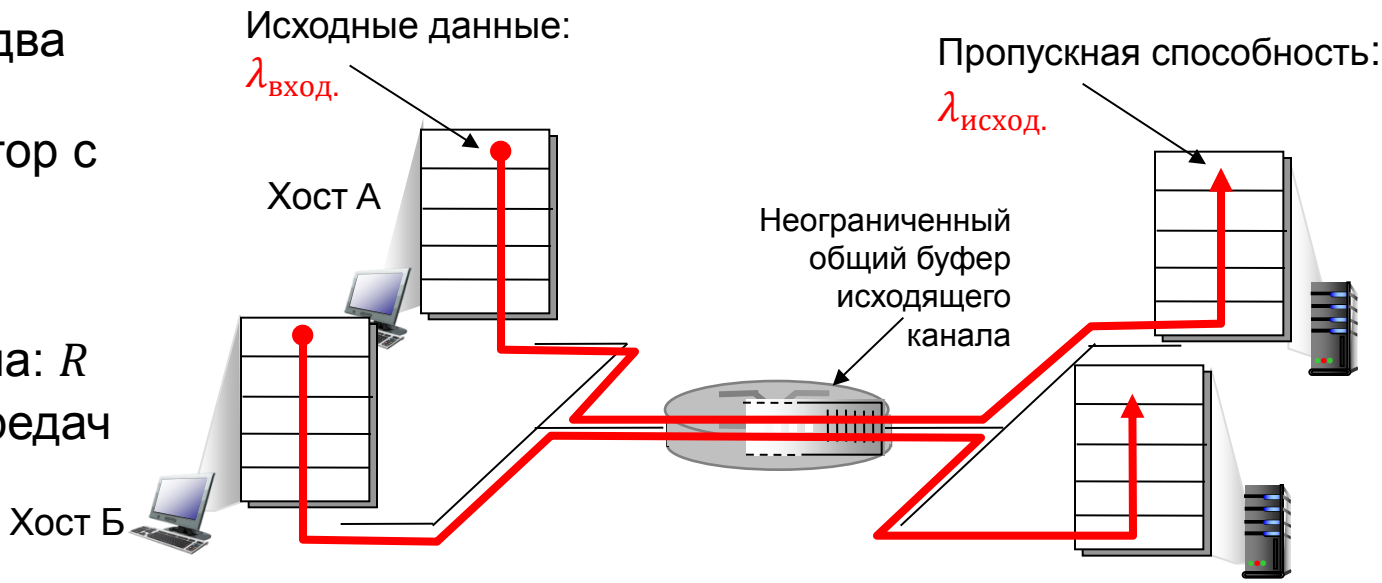
Принципы управления перегрузкой

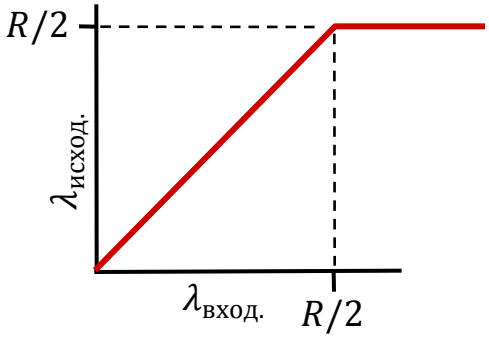
перегрузка:

- упрощенно: «слишком много источников слишком быстро отправляют слишком много данных для обработки в *сети*»
- Различные формы управления потоком!
- Возможные варианты:
  - Потеря пакетов (переполнение буферов маршрутизаторов)
  - Длительные задержки (очереди в буферах маршрутизаторов)
- Проблема из числа наиважнейших!

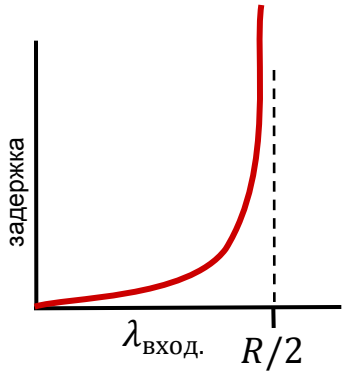
Причины и следствия перегрузки: сценарий 1

- Два отправителя, два получателя
- Один маршрутизатор с неограниченным буфером
- Пропускная способность канала: R
- Нет повторных передач





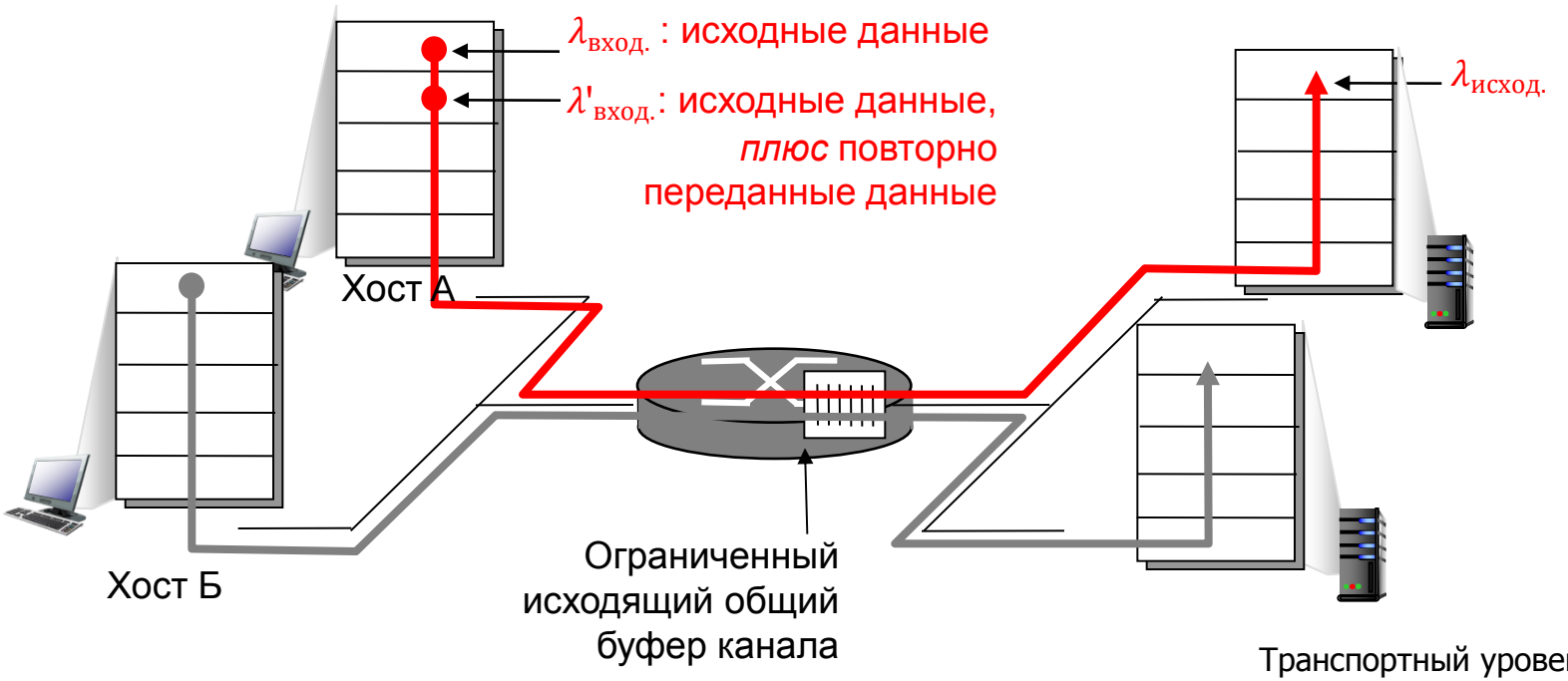
- Максимальная пропускная способность соединения: $R/2$



- Большие задержки прибытия сегментов, $\lambda_{\text{вход.}}$, приближения потенциала

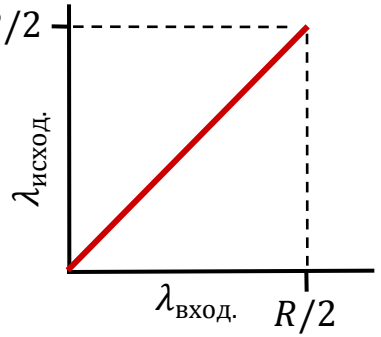
Причины и следствия перегрузки: сценарий 2

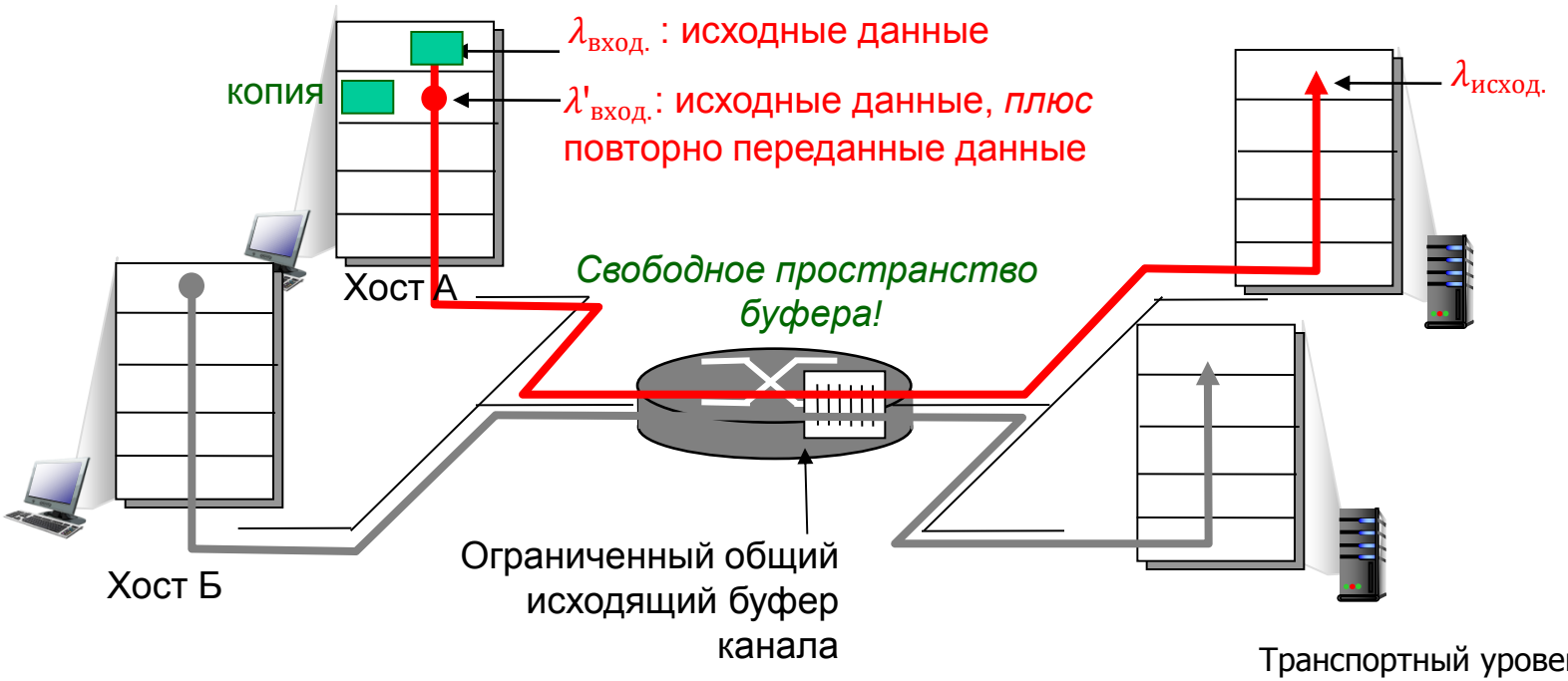
- Один маршрутизатор, *ограниченные* буферы
- Отправитель повторно передает пакеты, для которых истекло время
  - Вход уровня приложений = выход уровня приложений: $\lambda_{\text{вход.}} = \lambda_{\text{исход.}}$
  - Входные данные транспортного уровня включают *повторные передачи*: $\lambda'_{\text{вход.}} \geq \lambda_{\text{вход.}}$



Причины и следствия перегрузки: сценарий 2

Идеализация:
- исчерпывающие сведения
- Отправитель отправляет, только когда доступны буферы маршрутизатора

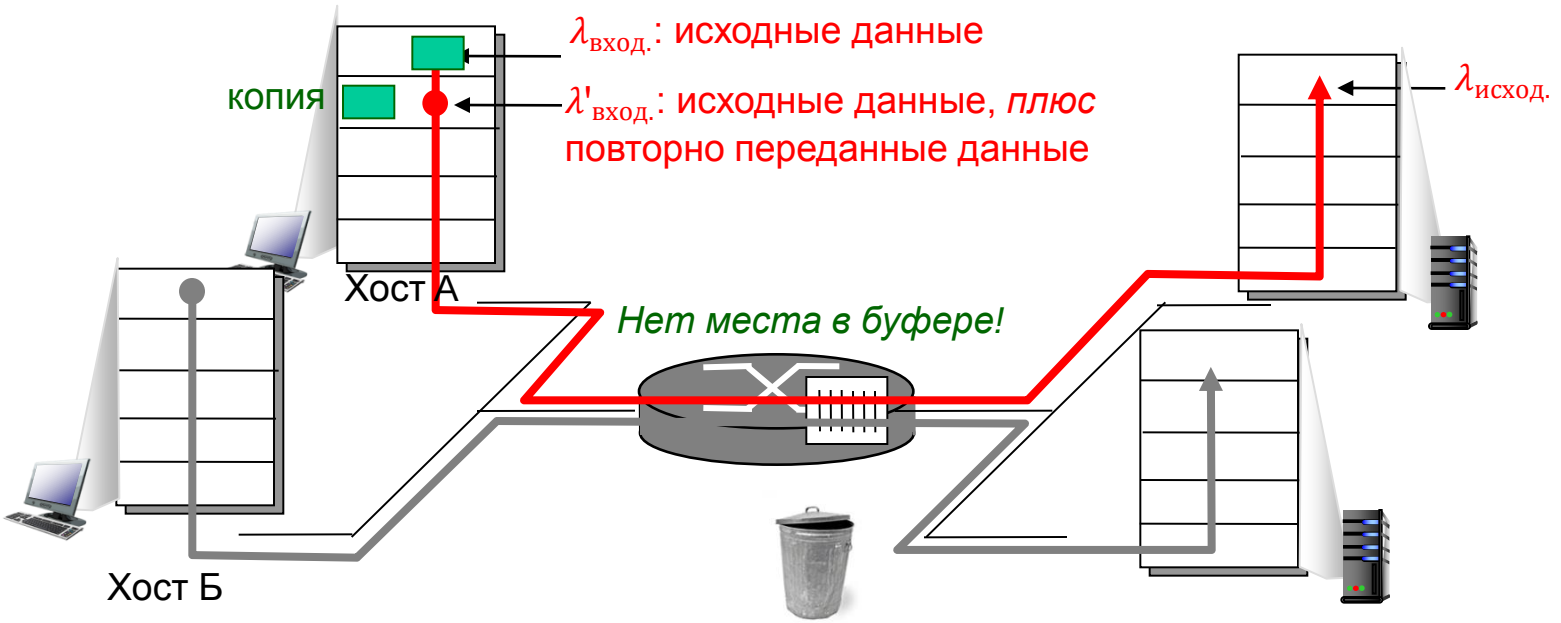




Причины и следствия перегрузки: сценарий 2

Идеализация: известны потери, пакеты могут быть потеряны, пропущены маршрутизатором и-за переполненного буфера

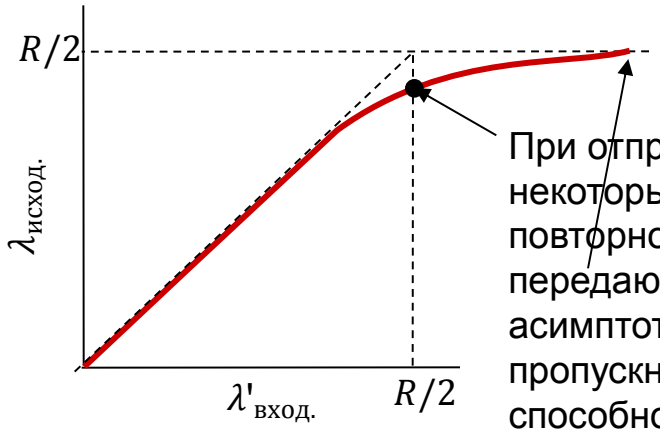
- Отправитель отправляет пакет повторно, только если знает, что он потерян



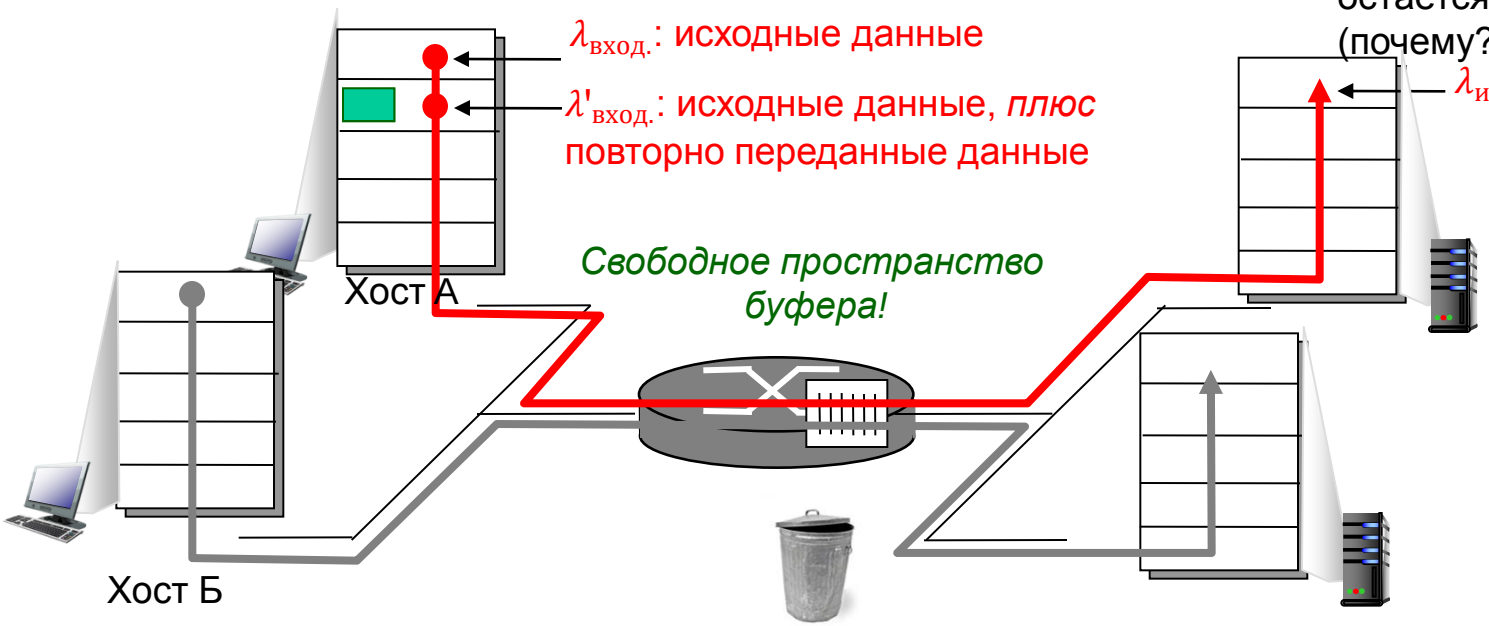
Причины и следствия перегрузки: сценарий 2

Идеализация: известны потери, пакеты могут быть потеряны, пропущены маршрутизатором и-за переполненного буфера

- Отправитель отправляет пакет повторно, только если знает, что он потерян



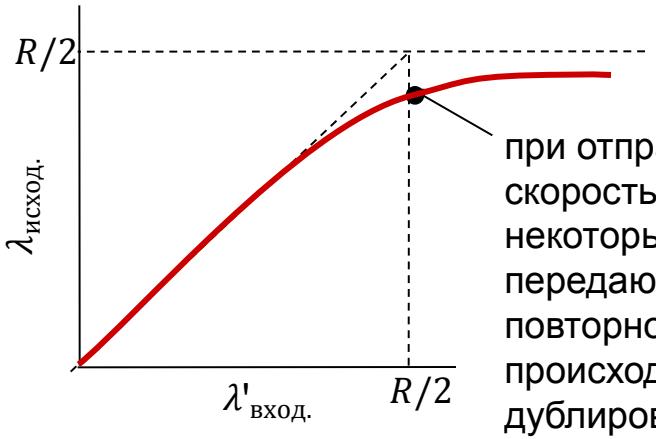
При отправке в $R/2$, некоторые пакеты повторно передаются, но асимптотически пропускная способность остается $R/2$ (почему?)



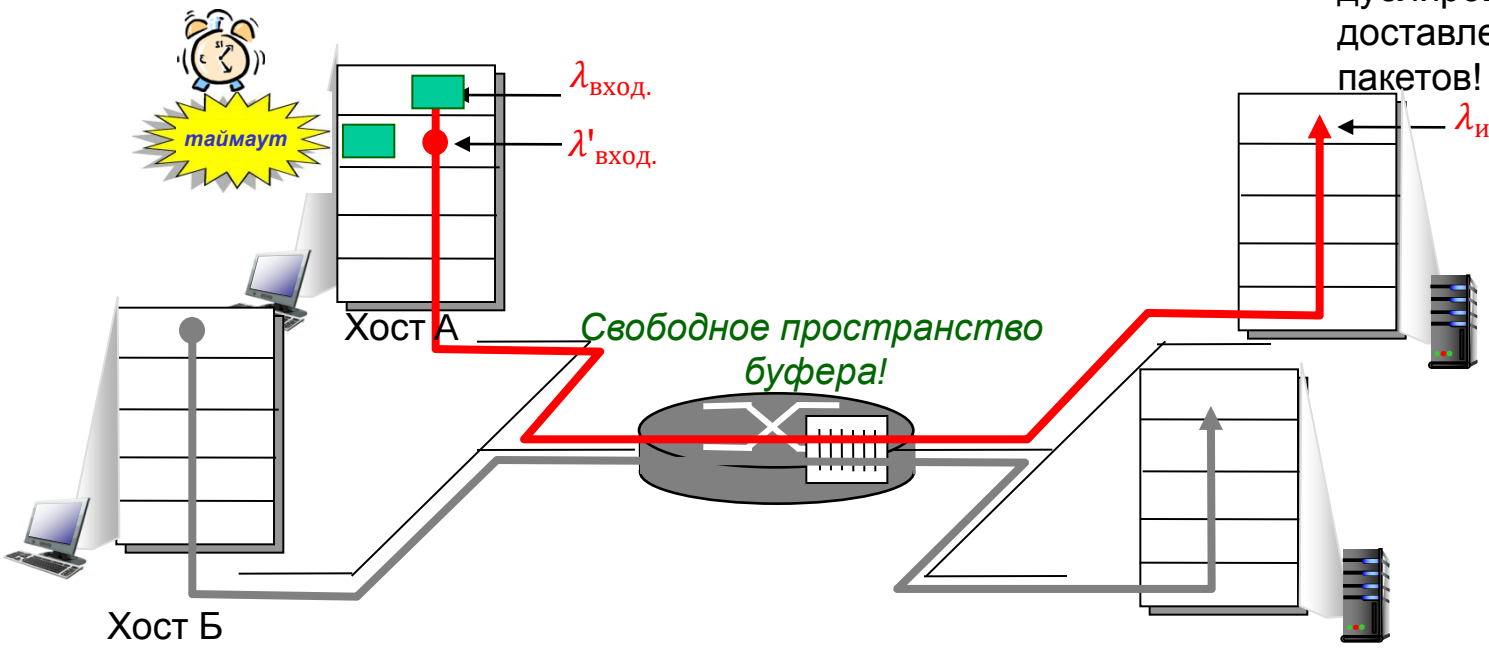
Причины и следствия перегрузки: сценарий 2

Реалистичная картина: дублирование

- Пакеты могут быть потеряны, пропущены маршрутизатором из-за переполнения буферов
- Преждевременный таймаут отправителя, отправка *двух* копий, каждая из которых доставляется



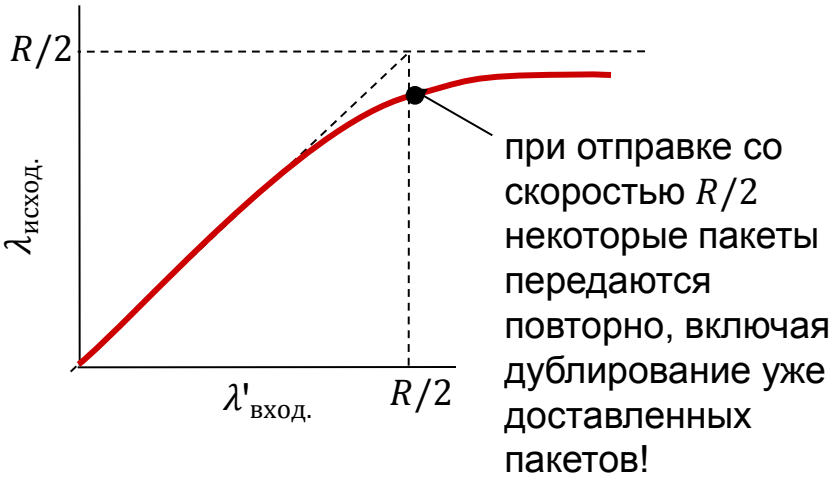
при отправке со скоростью $R/2$ некоторые пакеты передаются повторно, происходит дублирование уже доставленных пакетов!



Причины и следствия перегрузки: сценарий 2

Реалистично: дублирование

- Пакеты могут быть потеряны, пропущены маршрутизатором из-за переполнения буферов
- Преждевременный таймаут отправителя, отправка *двух* копий, какждая из которых доставляется



«следствия» перегрузки:

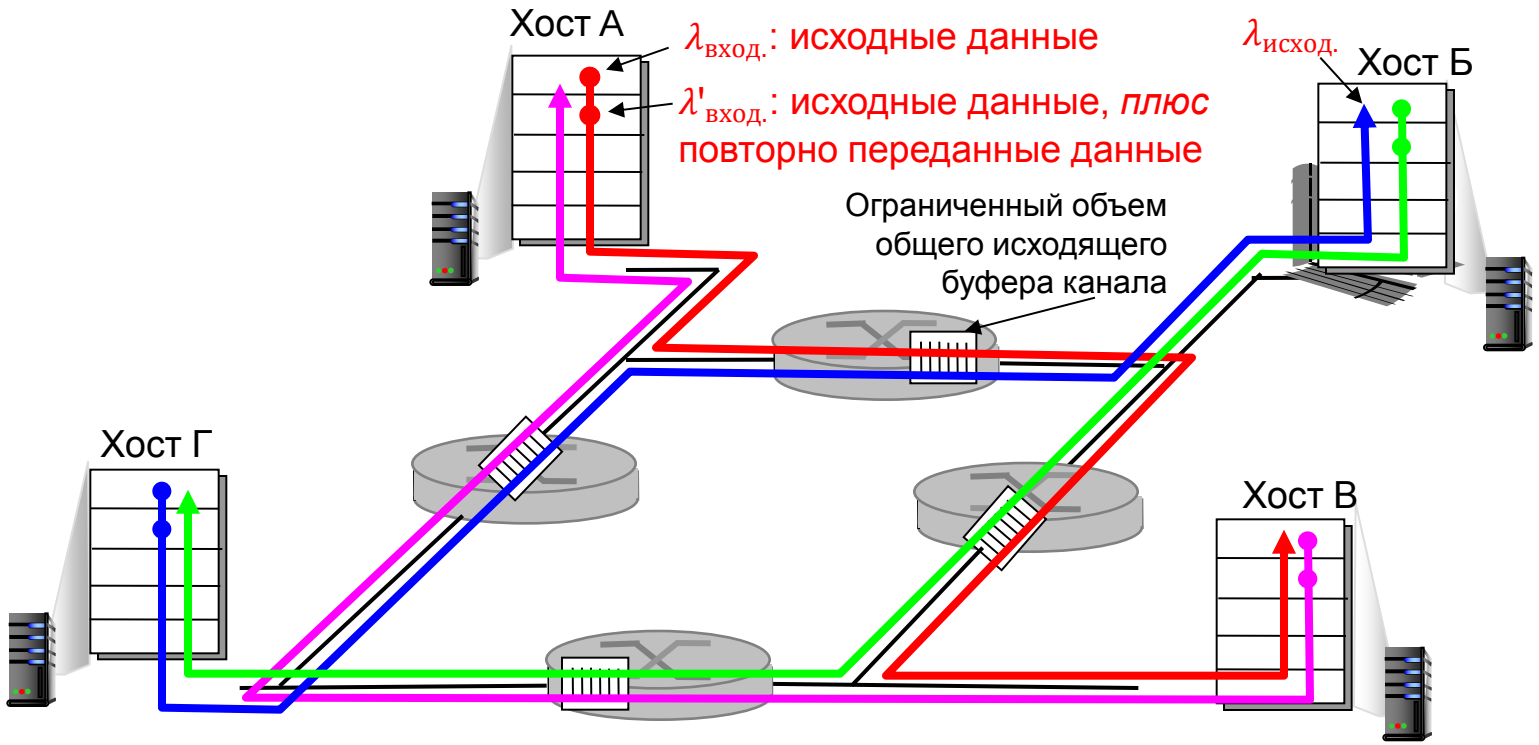
- Больше работы (повторные передачи) для получения хорошей скорости
- Ненужные повторные передачи: канал обрабатывает множественные копии пакетов
  - Снижение пропускной способности

Причины и следствия перегрузки: сценарий 3

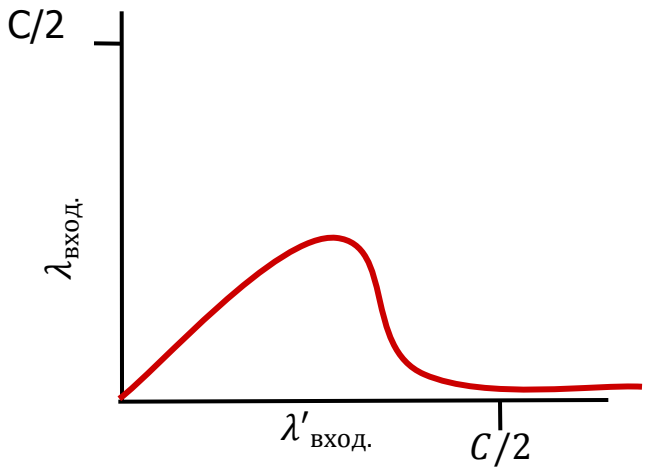
- Четыре отправителя
- Путь в несколько переходов
- таймаут/повторная передача

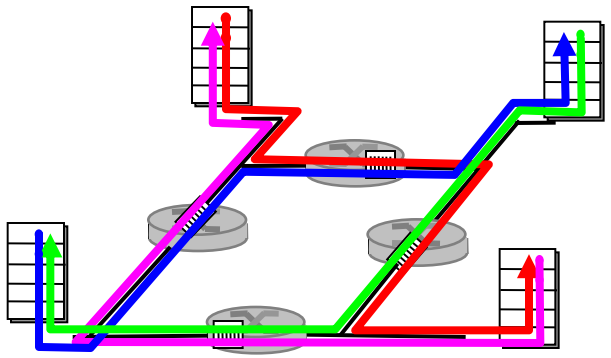
В: что произойдет при увеличении $\lambda_{\text{вход.}}$ и $\lambda'_{\text{вход.}}$?

О: при увеличении красного $\lambda'_{\text{вход.}}$ все поступающие синие пакеты из начала очереди будут пропущены, пропускная способность → 0



Причины и следствия перегрузки: сценарий 3





Другие «следствия» перегрузки:

- При пропуске пакета любая использованная ранее для этого пакета возможность передачи была потрачена впустую!

Подходы к управлению перегрузкой

Два основных подхода к управлению перегрузкой:

Управление перегрузкой на конечных системах:

- Нет явного отклика от сети
- На перегрузку указывает потери и задержки на конечных системах
- Используется в протоколе TCP

Сетевое управление перегрузкой:

- Маршрутизаторы дают отклик конечным системам
  - Одноразрядный индикатор перегрузки (SNA, DECbit, TCP/IP ECN, ATM)
  - Явное указание скорости отправителю

Пример управления перегрузкой ATM ABR

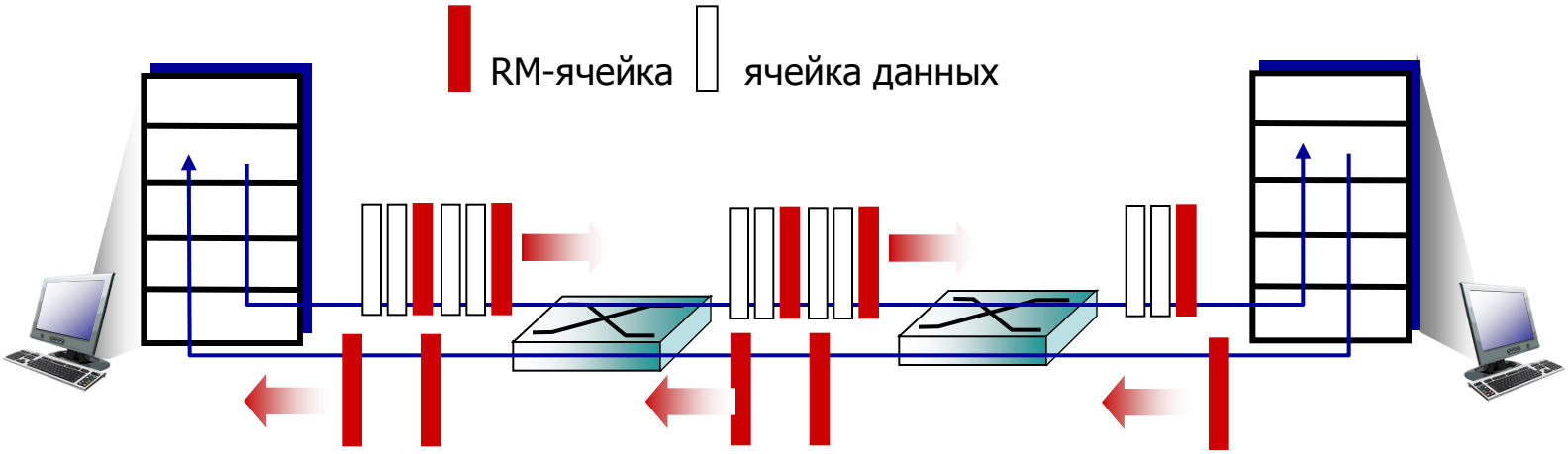
ABR: доступная скорость:

- «эластичный сервис»
- если путь отправителя «недогружен»:
  - Отправитель должен использовать доступную пропускную способность канала
- Если путь отправителя перегружен:
  - Отправитель замедляет скорость до минимальной гарантированной скорости

RM-ячейки (управление ресурсами):

- Передаваемые отправителем, перемежаются с ячейками данных
- Биты в RM ячейке устанавливаются коммутаторами («*поддержка со стороны сети*»)
  - *NI бит:* нет увеличения скорости (средняя перегрузка)
  - *CI бит:* индикатор перегрузки
- RM-ячейки возвращаются отправителю от получателя с неизмененными разрядами

Пример управления перегрузкой ATM ABR



- Двух байтовое поле ER (explicit rate, явная скорость) в RM-ячейке
  - Перегруженный коммутатор может уменьшить значение ER в ячейке
  - Скорость ухода данных от отправителя – максимальная поддерживаемая скорость на пути
- EFCI бит в ячейках данных: получает значение 1 в перегруженном коммутаторе
  - Если ячейка данных предшествующая RM-ячейке имеет установленный бит EFCI, получатель устанавливает CI-бит в возвращаемой RM-ячейке

Глава 3: План

3.1 Службы транспортного уровня

3.2 Мультиплексирование и демультиплексирование

3.3 Передача данных без установления логического соединения: протокол UDP

3.4 Принципы надежной передачи данных

3.5 Передача данных с установлением логического соединения: TCP

- Структура сегмента
- Надежная передача данных
- Управление потоком
- Управление соединением

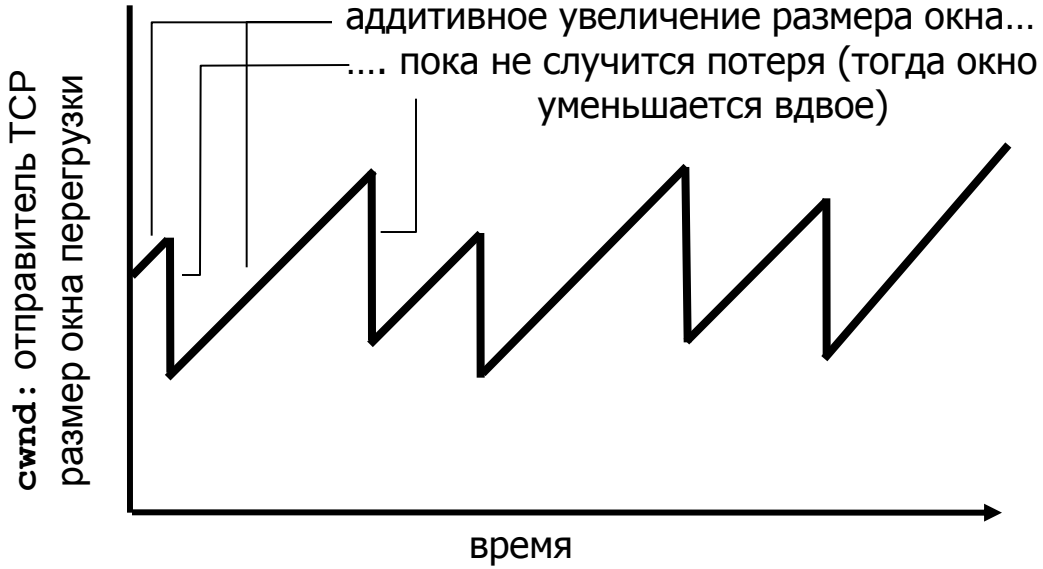
3.6 Принципы управления перегрузками

3.7 Механизм управления перегрузками протокола TCP

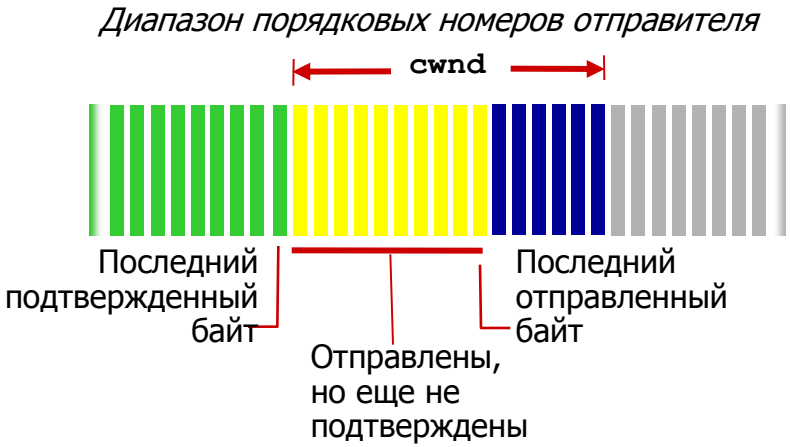
Управление перегрузкой TCP: аддитивное ускорение и мультипликативное замедление

- *Суть:* отправитель увеличивает скорость передачи (размер окна), зондируя доступную ширину канала, пока не случится потеря
  - *Аддитивное ускорение:* увеличение **`cwnd`** на 1 MSS каждые RTT до обнаружения потери
  - *Мультипликативное замедление*: **`cwnd`** уменьшается в половину после потери

Алгоритм AIMD демонстрирует зубчатое поведение: зондирование ширины канала



Управление перегрузкой TCP: подробности



- Предел передачи отправителя:

$$\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \texttt{cwnd}$$

- **cwnd** – это динамическая функция от perceived перегрузки сети

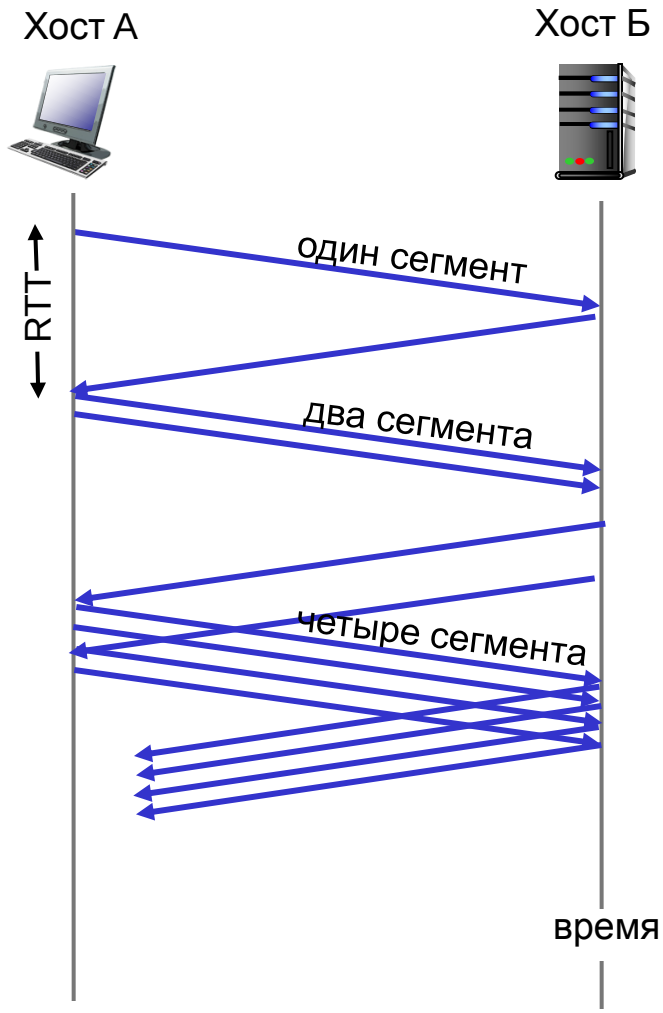
Скорость отправки TCP:

- *приблизительно:* отправить cwnd байт, ждать RTT до получения всех ACK-пакетов, затем отправить еще

$$\text{скорость} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ байт/сек}$$

TCP: Медленный старт

- Скорость увеличивается экспоненциально с начала соединения и до первого события потери:
  - начальное **`cwnd`** = 1 MSS
  - удвоение **`cwnd`** каждые RTT
  - Увеличение **`cwnd`** при получении каждого ACK
- *резюме:* начальная скорость невысока, но возрастает экспоненциально



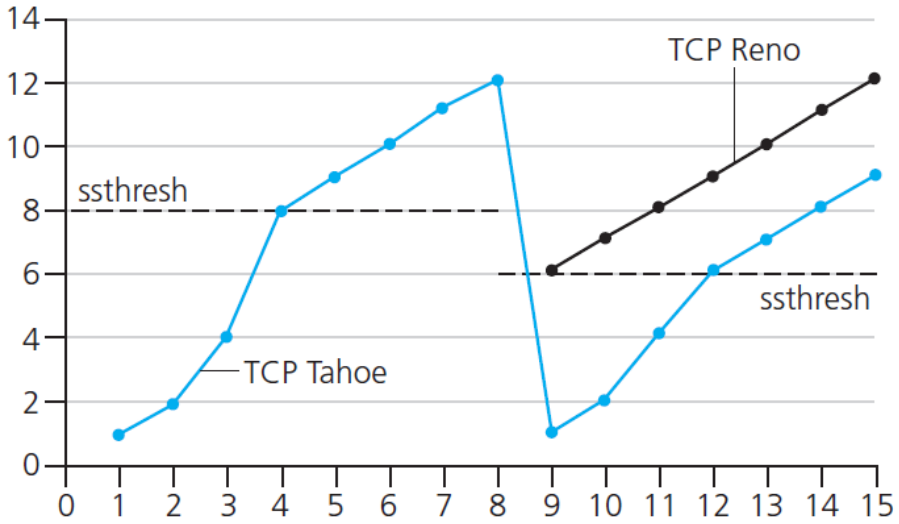
TCP: обнаружение и обработка потерь

- На потери указывает таймаут:
  - **cwnd** становится равным 1 MSS;
  - Затем окно экспоненциально увеличивается (как при медленном старте) до порогового значения, затем увеличивается линейно
- На потери указывают 3 дублирующих ACK-пакета: TCP RENO
  - дублирующие ACK-пакеты указывают на возможность доставки некоторых данных по сети
  - **cwnd** уменьшается вдвое, затем окно линейно увеличивается
- TCP Tahoe всегда устанавливает **cwnd** равным 1 (и при таймауте, и при 3 дублирующих ACK-пакетах)

TCP: переход из медленного старта

В: когда экспоненциальное ускорение должно смениться линейным?

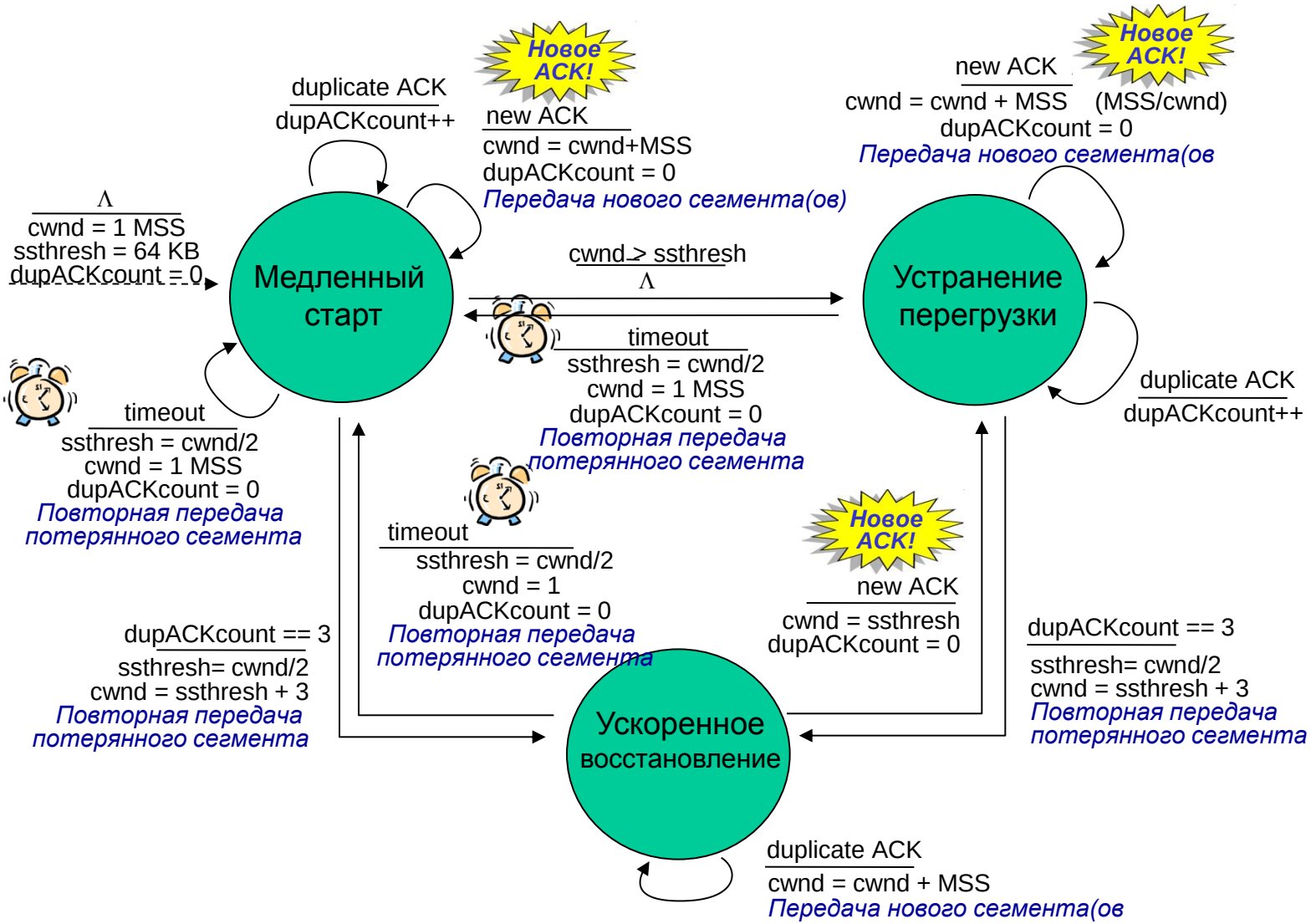
О: когда **`cwnd`** будет равно ½ от собственного значения до таймаута.



Реализация:

- Изменяемое значение **`ssthresh`**
- В случае потери, **`ssthresh`** будет равно 1/2 от **`cwnd`** до события потери

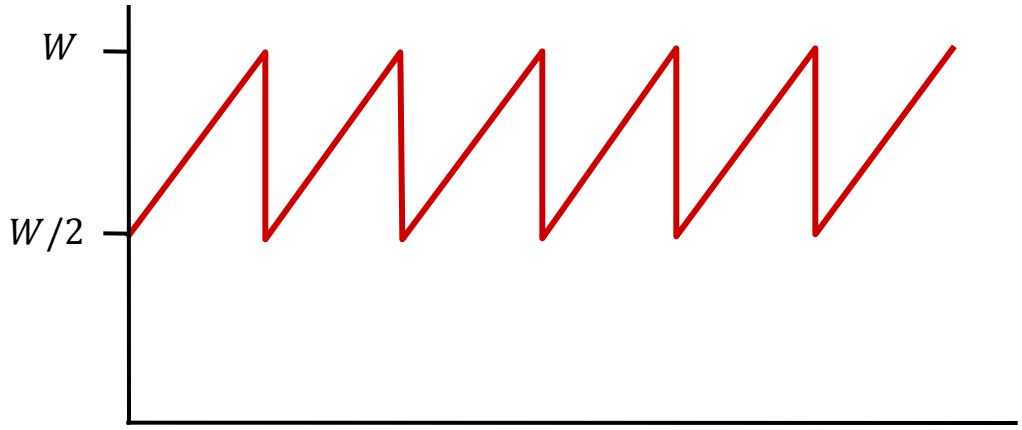
Резюме: управление перегрузкой TCP



Пропускная способность TCP

- Средняя пропускная способность TCP-соединения - это функция от размера окна, RTT?
  - Игнорируем медленный старт, допускаем, что всегда есть данные для отправки
- W: размер окна (в байтах) при потерях
  - Средний размер окна (количество передаваемых байт) равно $¾W$
  - Средняя пропускная способность равна $3/4W$ в RTT

Ср. пропускная способность TCP $= \frac{3}{4}\frac{W}{RTT}$ Байт/сек



Будущее TCP

- пример: сегменты по 1500 байт, 100мс RTT, необходима пропускная способность 10 Гбит/с
- требуется W = 83,333 передаваемых сегментов
- Выраженная в вероятных потерях сегментов L пропускная способность [Mathis 1997]:

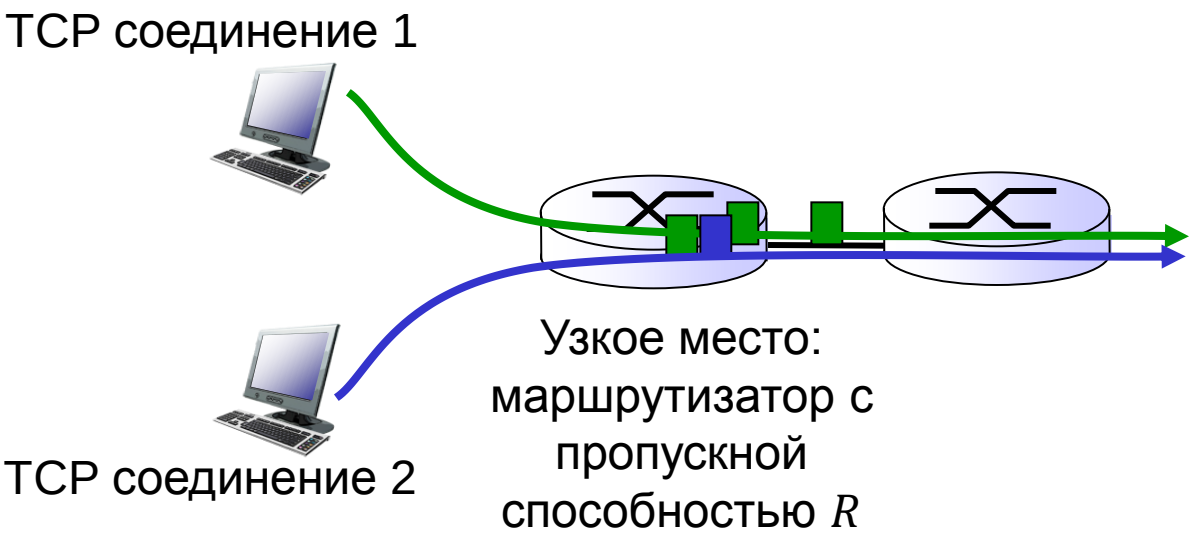
$$\text{Пропускная способностьTCP} = \frac{1{,}22 \times MSS}{RTT\sqrt{L}}$$

➜ для получения пропускной способности10 Гбит/сек, частота потерь должна быть $L = 2 \times 10^{-10}$ *– очень низкая частота!*

- новые версии протокола TCP для высоких скоростей

TCP: выравнивание скорости передачи

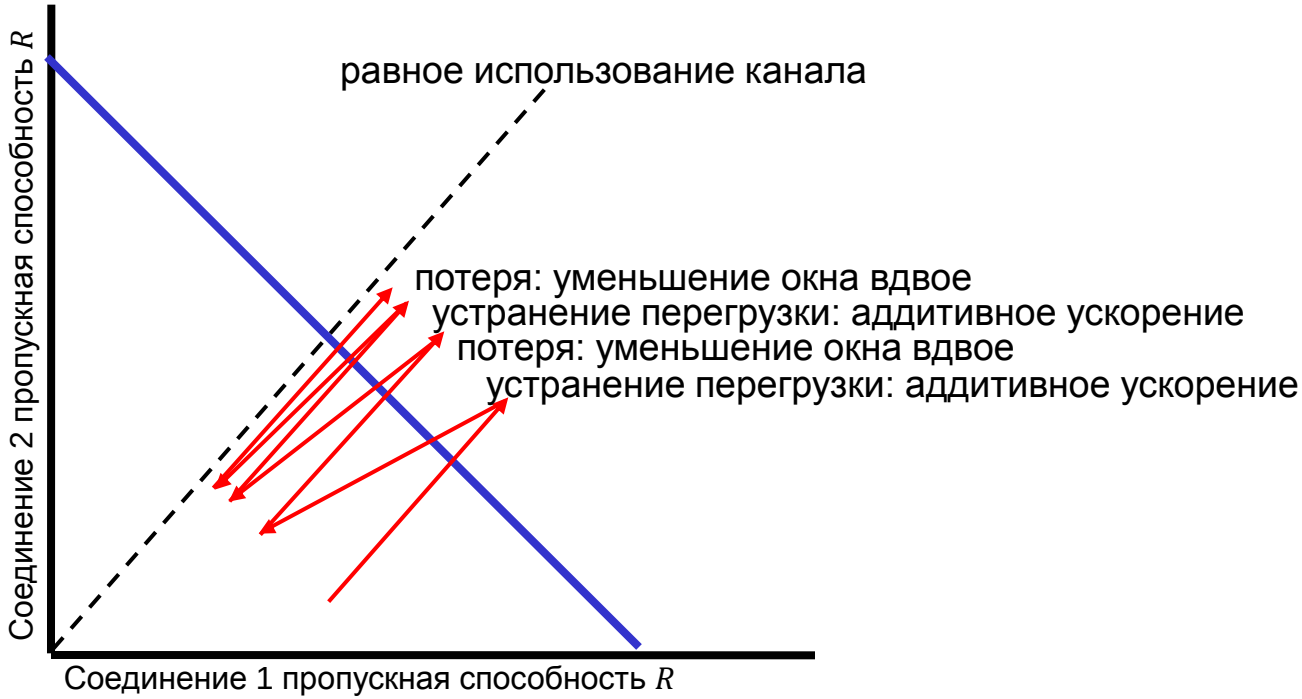
Цель выравнивания: если K TCP-сеансов совместно используют узкий канал ширины R, каждое должно получить среднюю скорость R/K



Почему в TCP выполняется выравнивание скорости передачи?

Два параллельных сеанса:

- Аддитивное ускорение дает уклон 1, при увеличении пропускной способности
- Мультипликативное замедление пропорционально снижению пропускной способности



Выравнивание скорости передачи

Выравнивание скорости и UDP

- Мультимедийные приложения зачастую не используют протокол TCP
  - нежелательное снижение скорости из-за управления перегрузкой
- Напротив используют UDP:
  - Отправляют аудио/видео с постоянной скоростью, не чувствительны к потере пакетов

Выравнивание скорости параллельных TCP соединений

- Приложение может открывать несколько одновременных соединений между двумя хостами
- Так поступают веб-браузеры
- Например, скорость канала R для 9 действующих соединений:
  - Новое приложение, запрашивающее 1 TCP соединение, получает скорость $R/10$
  - Новое приложение, запрашивающее 11 TCP соединения, получает $R/2$

Заключение

- Принципы служб транспортного уровня:
  - Мультиплексирование, демультиплексирование
  - Надежная передача данных
  - Управление потоком
  - Управление перегрузкой
- Установка, реализация в сети Интернет:
  - UDP
  - TCP

далее:

- Покидаем «периферию» сети (прикладной, транспортный уровни)
- Продвигаемся в «ядро» сети