

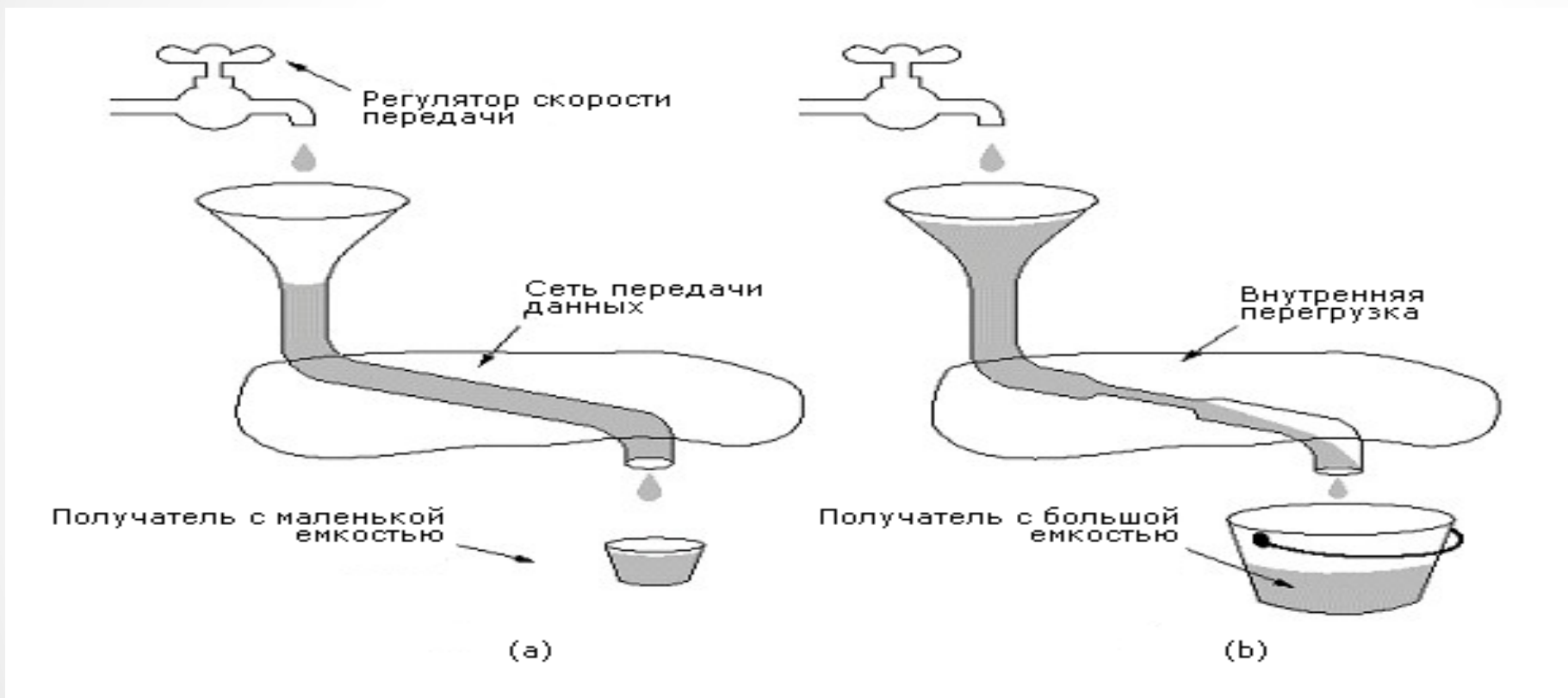


# Перегрузка в сетях передачи данных с пакетной коммутацией

Введение в компьютерные сети  
чл.-корр. РАН Смелянский Р.Л.  
Кафедра АСВК

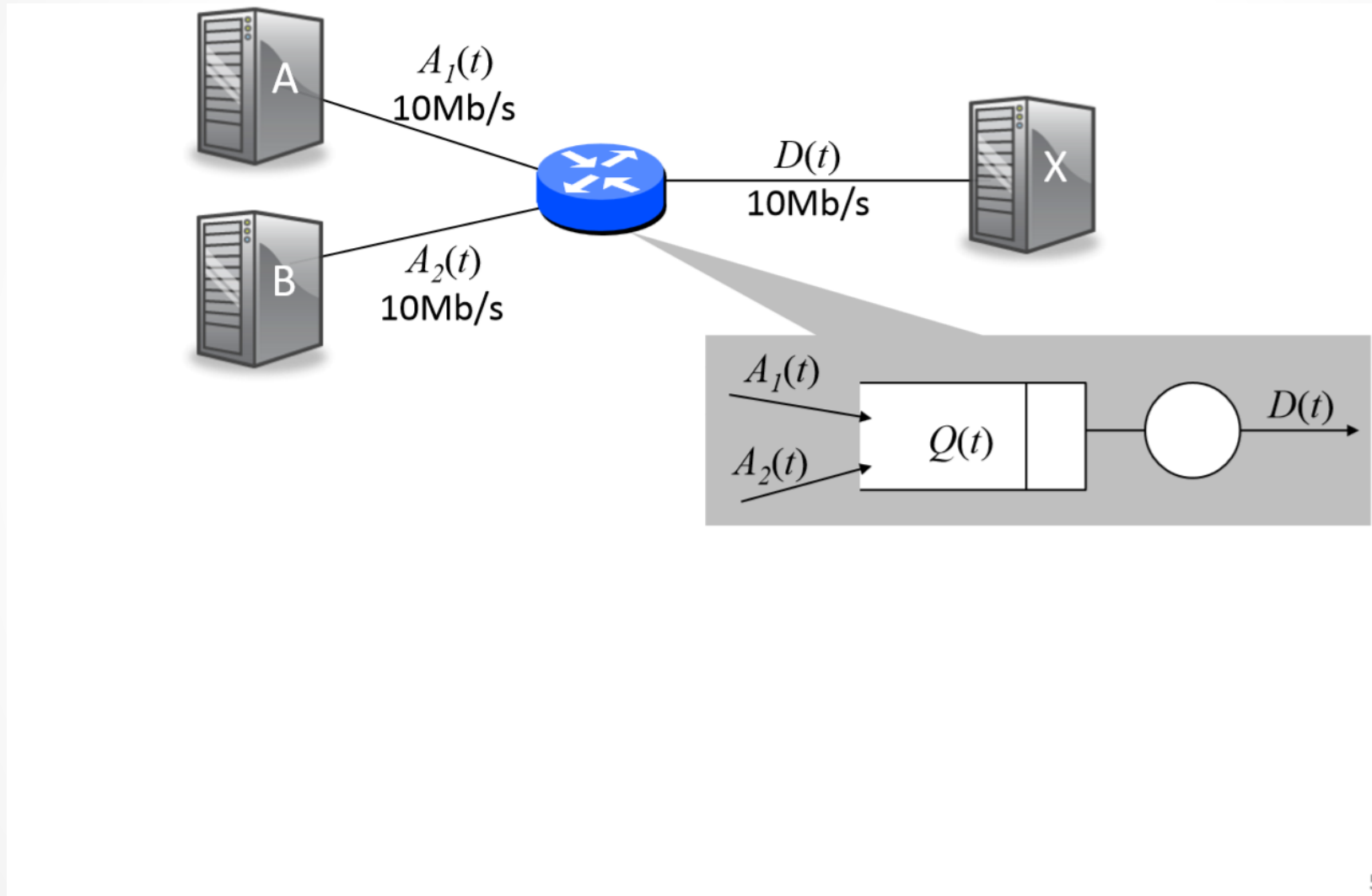


# Управление потоком и управление перегрузкой



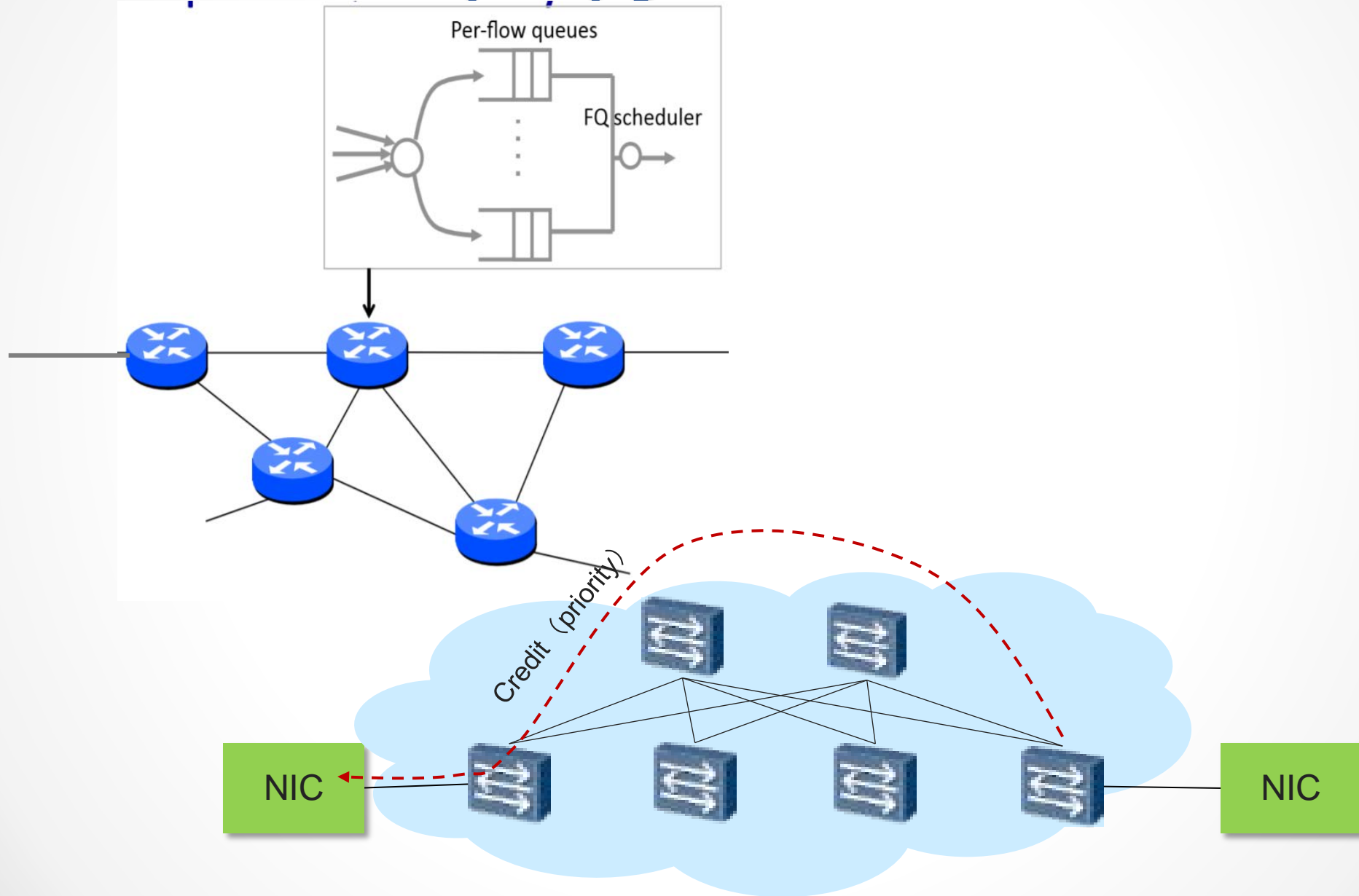


# Что вызывает перегрузку?





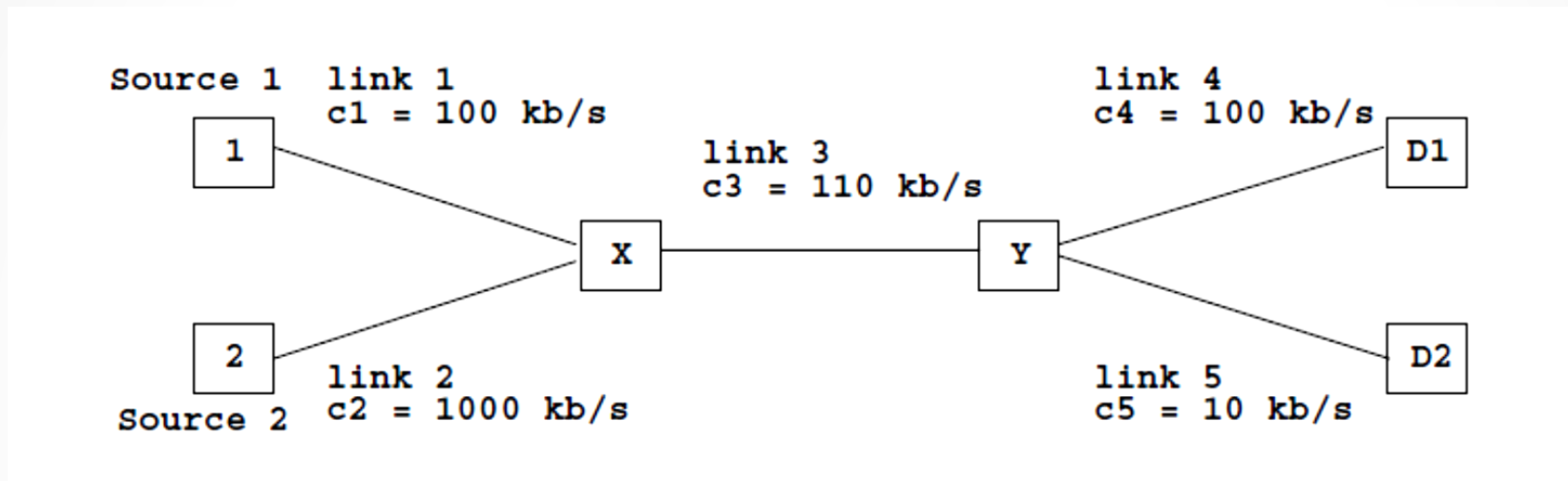
# Перегрузки в сети





# Что вызывает перегрузку?

Источники ничего не знают о распределении пропускных способностей каналов



$$\lambda_1 = 100 \text{ kb/s}$$

$$\lambda_2 = 1000 \text{ kb/s}$$

**Буфер у X будет переполняться!  
Интенсивность трафиков не соответствует  
распределению пропускных способностей каналов .**



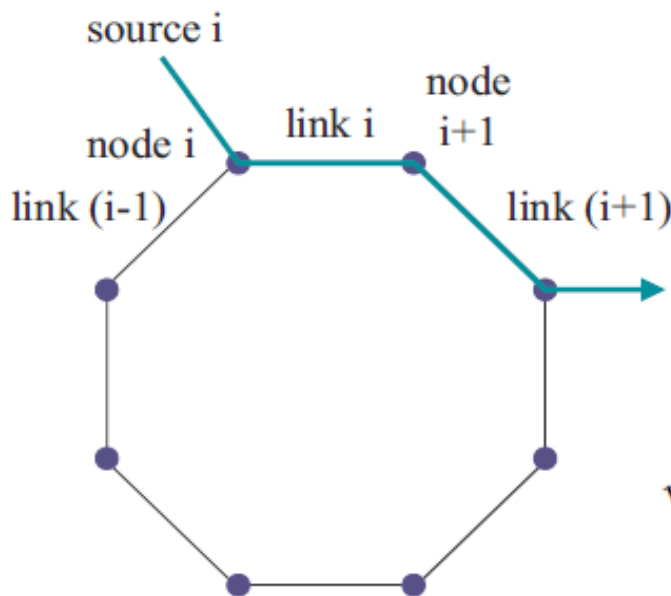
# Пример

$$\begin{cases} \lambda'_i = \min \left( \lambda_i, \frac{c_i}{\lambda_i + \lambda'_{i-1}} \lambda_i \right) \\ \lambda''_i = \min \left( \lambda'_i, \frac{c_{i+1}}{\lambda'_i + \lambda_{i+1}} \lambda'_i \right) \end{cases}$$

Пусть для любого  $i$

$$\lambda' = \frac{c\lambda}{\lambda + \lambda'}$$

$$\lambda'' = \frac{c\lambda'}{\lambda + \lambda'}$$



$$\lambda' = \frac{\lambda}{2} \left( -1 + \sqrt{1 + 4\frac{c}{\lambda}} \right)$$

$$\lambda'' = c - \frac{\lambda}{2} \left( \sqrt{1 + 4\frac{c}{\lambda}} - 1 \right)$$

$$\sqrt{1 + u} = 1 + \frac{1}{2}u - \frac{1}{8}u^2 + o(u^2)$$

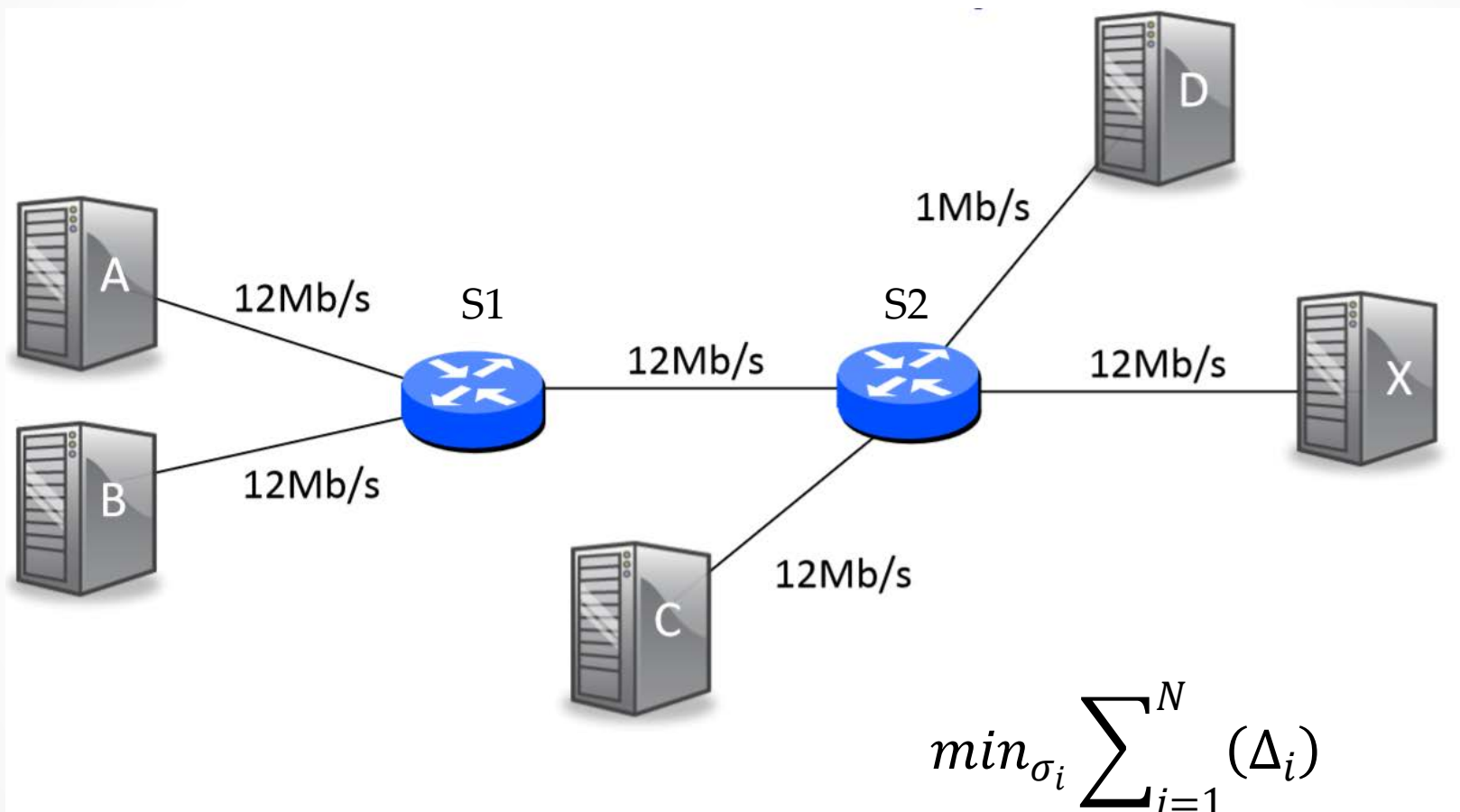
при  $u \rightarrow 0$

$$\lambda'' = \frac{c^2}{\lambda} + o\left(\frac{1}{\lambda}\right) \quad \text{При } \lambda \rightarrow \infty, \lambda'' \rightarrow 0$$

**Вывод: в сети с коммутацией пакетов источник должен регулировать свою скорость вброса пакетов в сеть в зависимости от состояния сети. В противном случае наступит перегрузка.**



# Попробуем ограничить источники







# Перегрузки неизбежны!

(может быть это и хорошо)

- Коммутацию пакетов используют потому, что она позволяет эффективно использовать пропускную способность каналов. Поэтому буферы в маршрутизаторах стараются держать заполненными.
- Если буферы не заполнены, то задержки малы, но интенсивность использования сети низкая.
- Если буферы постоянно заполнены, задержки возрастают, но интенсивность использования сети также возрастает

**Перегрузки неизбежны и даже желательны!**



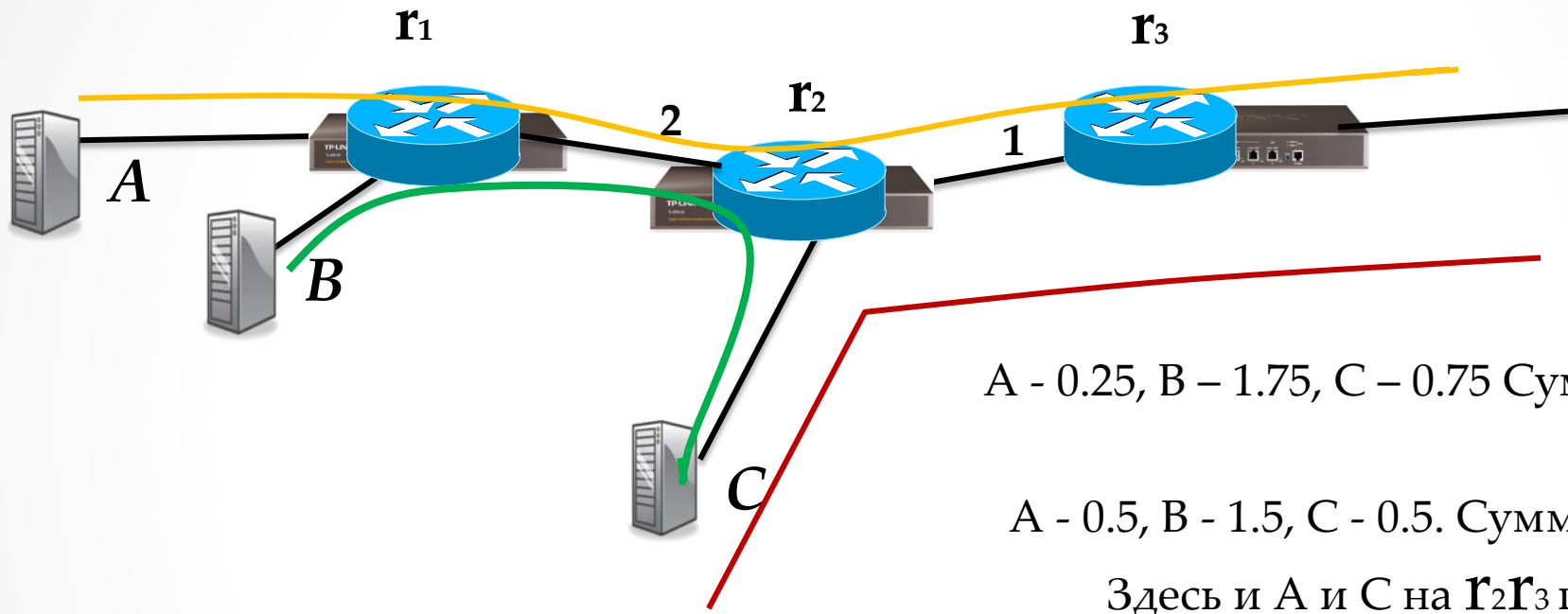


# Промежуточный итог

1. Перегрузка снижает производительность сети
2. Для повышения производительности сети Отправитель должен иметь обратную связь с Получателем
3. Перегрузки неизбежны и даже желательны - позволяют измерить пределы возможного
4. Перегрузки возникают из-за несбалансированности распределения ресурсов между Отправителями
5. Сброс пакета - *waste of resources*, *retransmission* может усугубить перегрузку (*backward restoration* - затратно!)
6. Введение справедливой политики распределения ресурсов помогает сократить риск возникновения перегрузки



# Понятие справедливости



Что такое справедливость и как ее измерить ?

максимум удовлетворенных —  $\min_{\sigma_i} \sum_{i=1}^N (\Delta_i)$ , где  $\sigma_i$  — выделенная доля для источника  $i$ ,

минимизация разрыва —  $\min_j (\bar{B}_j - \min_i \sigma_i)$ , где  $\bar{B}_j$  — пропускная способность каналов в среднем



# Max-min справедливость

Определение:

Распределение *max-min справедливо* если нельзя увеличить скорость какого-нибудь потока, не понизив скорости другого, потока с меньшей скоростью

A - 0.25, B – 1.75, C – 0.75  
Суммарно = 2.75

A - 0.5, B - 1.5, C - 0.5. Суммарно - 2.5  
Здесь и A и C на  $r_2 r_3$  по 0.5



# Необходимое и достаточное условие min-max справедливости

Пусть есть:

- $s = 1, \dots, S$  - множество источников
- $l = 1, \dots, L$  - множество линий
- $A_{l,s}$  – доля потока от  $s$  на линии  $l$
- $\{c_l\}_{l=1}^L$  - пропускная способность  $c_l$

Модель сети  $M = (x, A)$ , где

$x$  – распределение потоков от  $s$  (вектор),

$A$  – матрица  $S \times L$

*Распределение потоков допустимо*

- $\forall s: 1 \leq s \leq S \Rightarrow x_s \geq 0$
- $\forall l: 1 \div L: \sum_{i=1}^S A_{l,i} x_i \leq c_l$

*Линия  $l$  – насыщена если  $c_l = \sum_{i=1}^S A_{l,i} x_i$*

*Линия  $l$  – критична для  $s$  тогда и только тогда*

- $l$  – насыщена
- $s$  имеет максимальный поток среди всех источников, использующих  $l$ :
- $\forall s': A_{l,s} \geq 0 \rightarrow x_s \geq x_{s'}$

**Определение:**  $x$  является max-min распределением, если

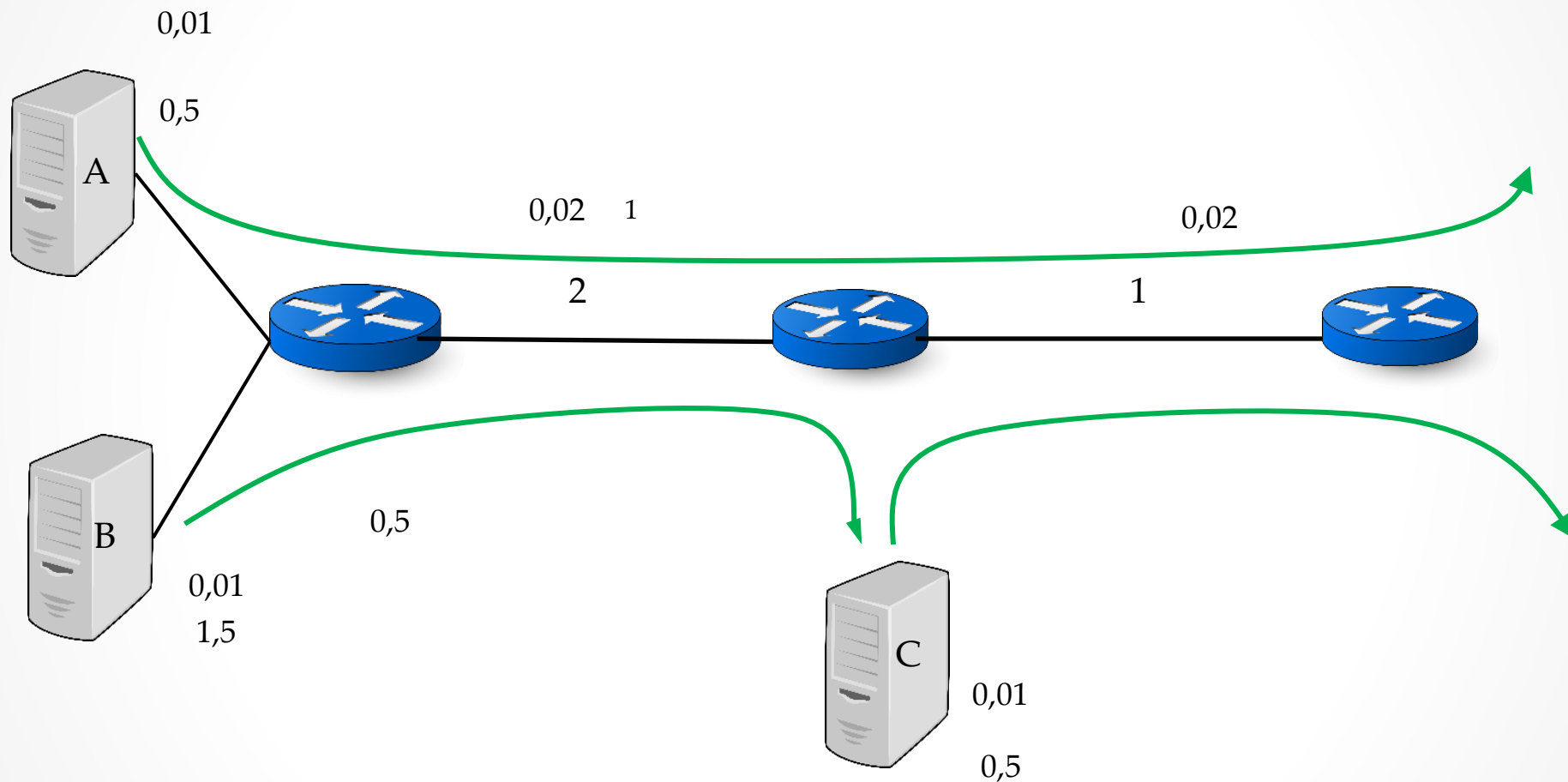
1.  $x$  допустимое распределение потоков

2.  $\forall y$  - допустимое:  $\exists s: y_s > x_s$  то  $\exists s': \forall s: (s \neq s') \& (x_{s'} \leq x_s) \& (y_{s'} < x_{s'})$

**Теорема 1:**  $x$  есть max-min распределение потоков тогда и только тогда когда у каждого источника есть критичная линия

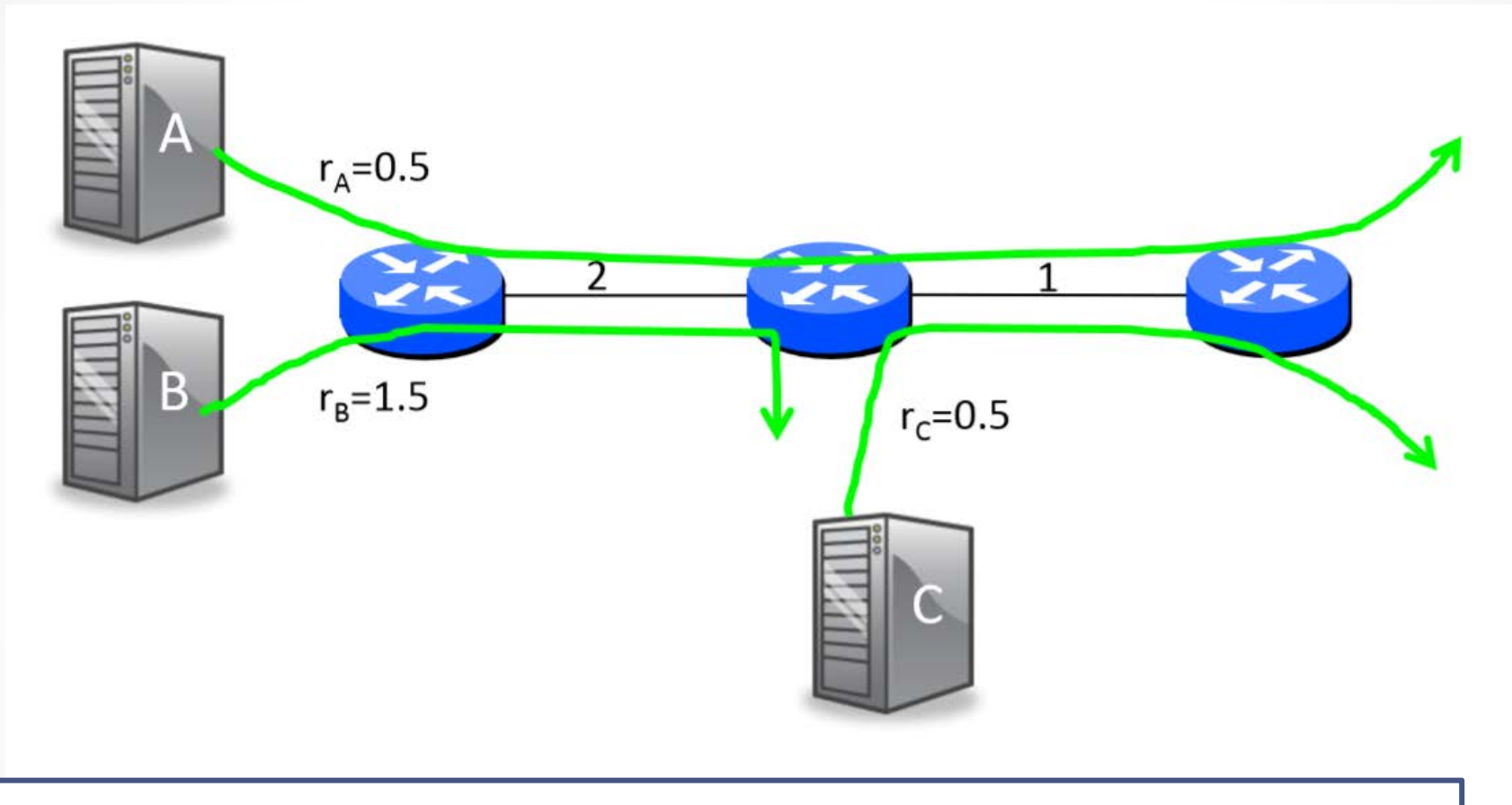


# Алгоритм построения max-min распределения (постепенного заполнения)





# Max-min справедливое распределение



**Если max-min справедливое распределение существует, то оно единственное для заданной ТОПОЛОГИИ.**



# Где и как управлять перегрузкой?

Введение в компьютерные сети  
чл.-корр. РАН Смелянский Р.Л.  
Кафедра АСВК ф-т ВМК МГУ





# План

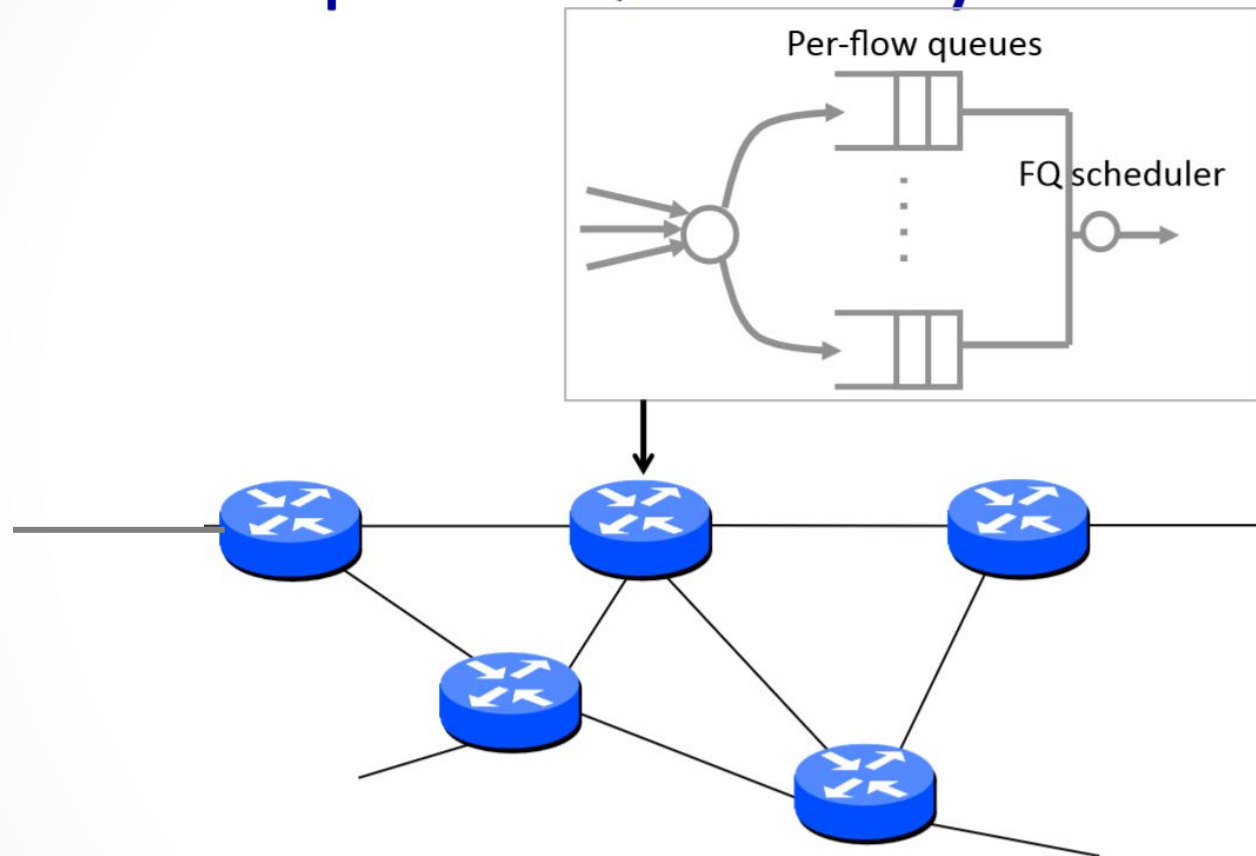
Мы хотим построить алгоритм управления перегрузками такой, чтобы обеспечить:

1. **Высокую пропускную способность:** каналы загружены, скорость потоков высокая
2. **Быстрая реакция на изменения** состояния сети (возникновение новых потоков)

- **Где надо размещать управление перегрузкой?**
  - В сети каждом маршрутизаторе (WFQ)
  - На отправителе и получателе
- **Скользящее окно и AIMD**



# Пример: FQ на каждом маршрутизаторе





# Управление перегрузками в ТСП

## (управление на конечных точках)

В ТСП управление перегрузкой размещается на конечном хосте

- Реакция на события, наблюдаемые на конечном хосте (например, потеря пакета).
- Использование скользящего окна, предназначенного для управления потоком в ТСП
- Постараться оценить сколько пакетов можно безопасно отправить в сеть одновременно.





# Скользящее окно перегрузки в ТСР

ТСР варьирует число пакетов отправляемых в сеть, изменяя размер скользящего окна:

$$\text{размер окна} = \min\{ \underbrace{\text{Объявленное окно}}_{\text{получатель}}, \underbrace{\text{Окно перегрузки}}_{\text{отправитель (cwnd)}} \}$$

Как определить размер cwnd?



# AIMD управление перегрузкой для одного потока

Введение в компьютерные сети

проф. Смелянский Р.Л.  
Лаборатория Вычислительных комплексов  
ф-т ВМК МГУ



# AIMD

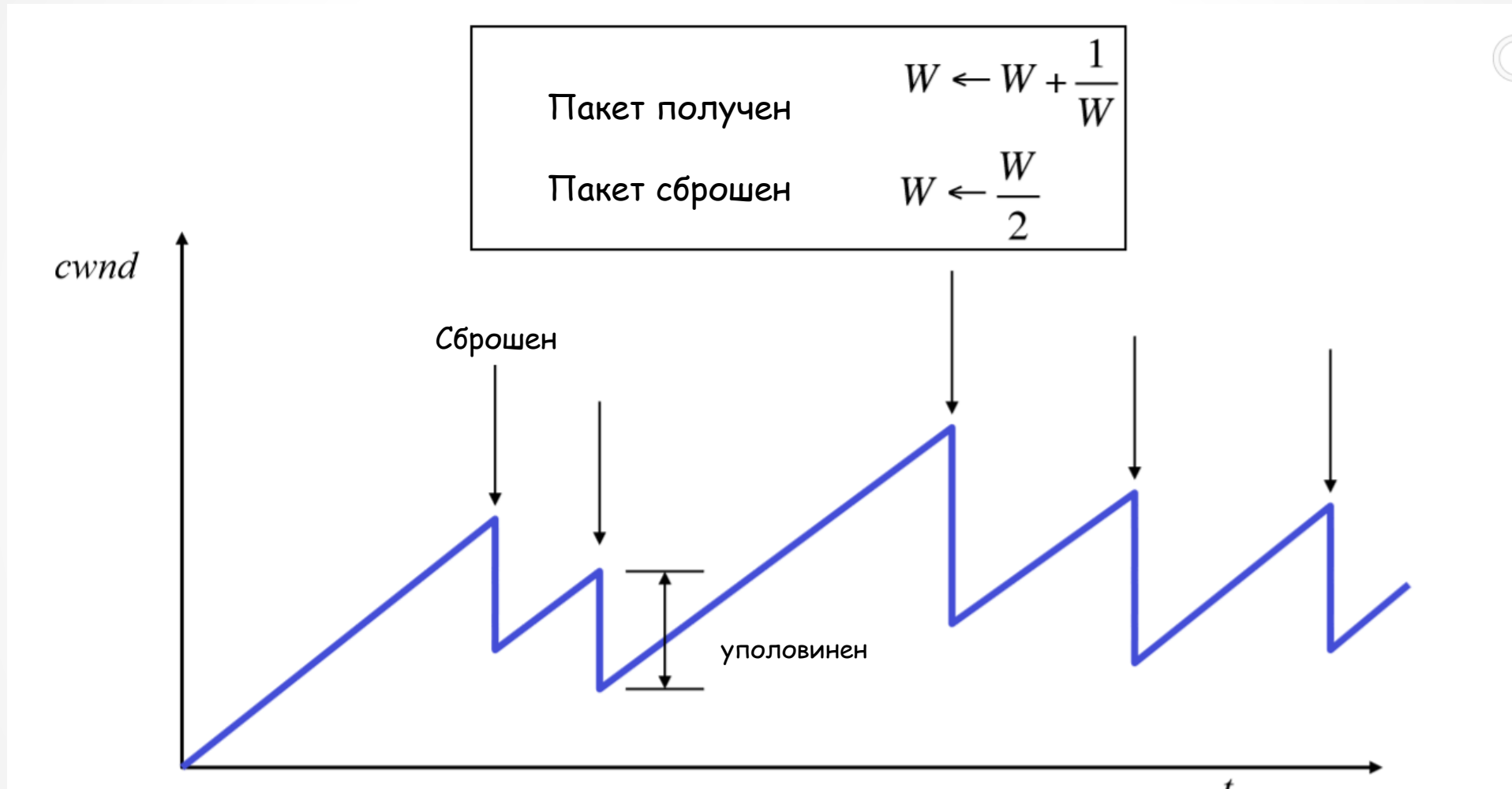
(Additive Increase Multiple Decrease)

- Если пакет получен успешно:  $W \leftarrow W + \frac{1}{W}$
- Если пакет был сброшен:  $W \leftarrow \frac{W}{2}$



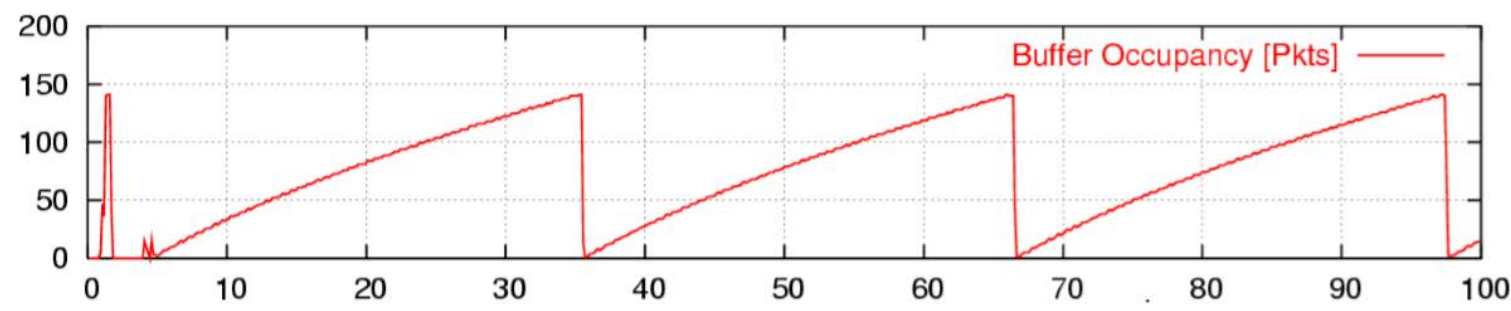
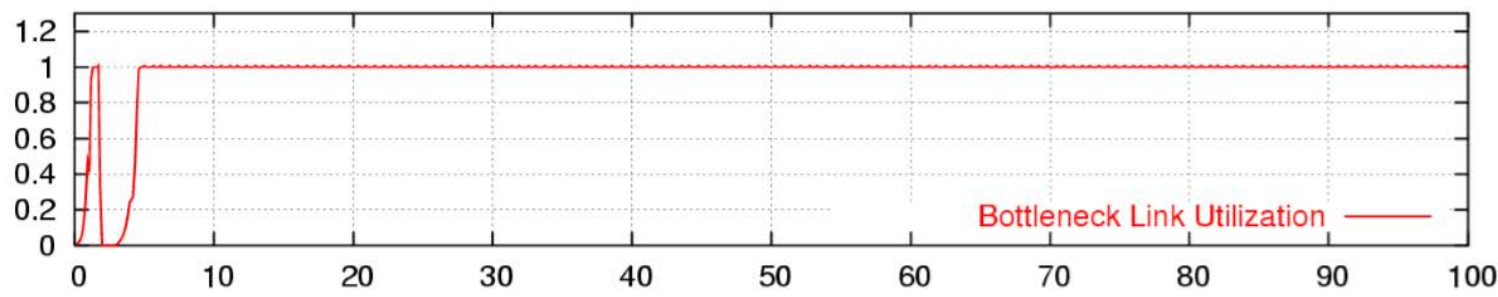
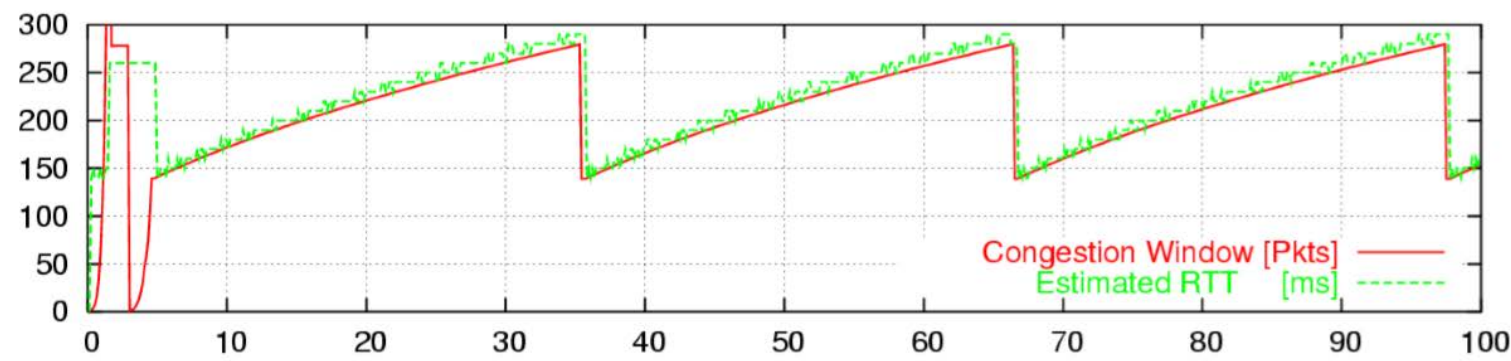


# Пила AIMD



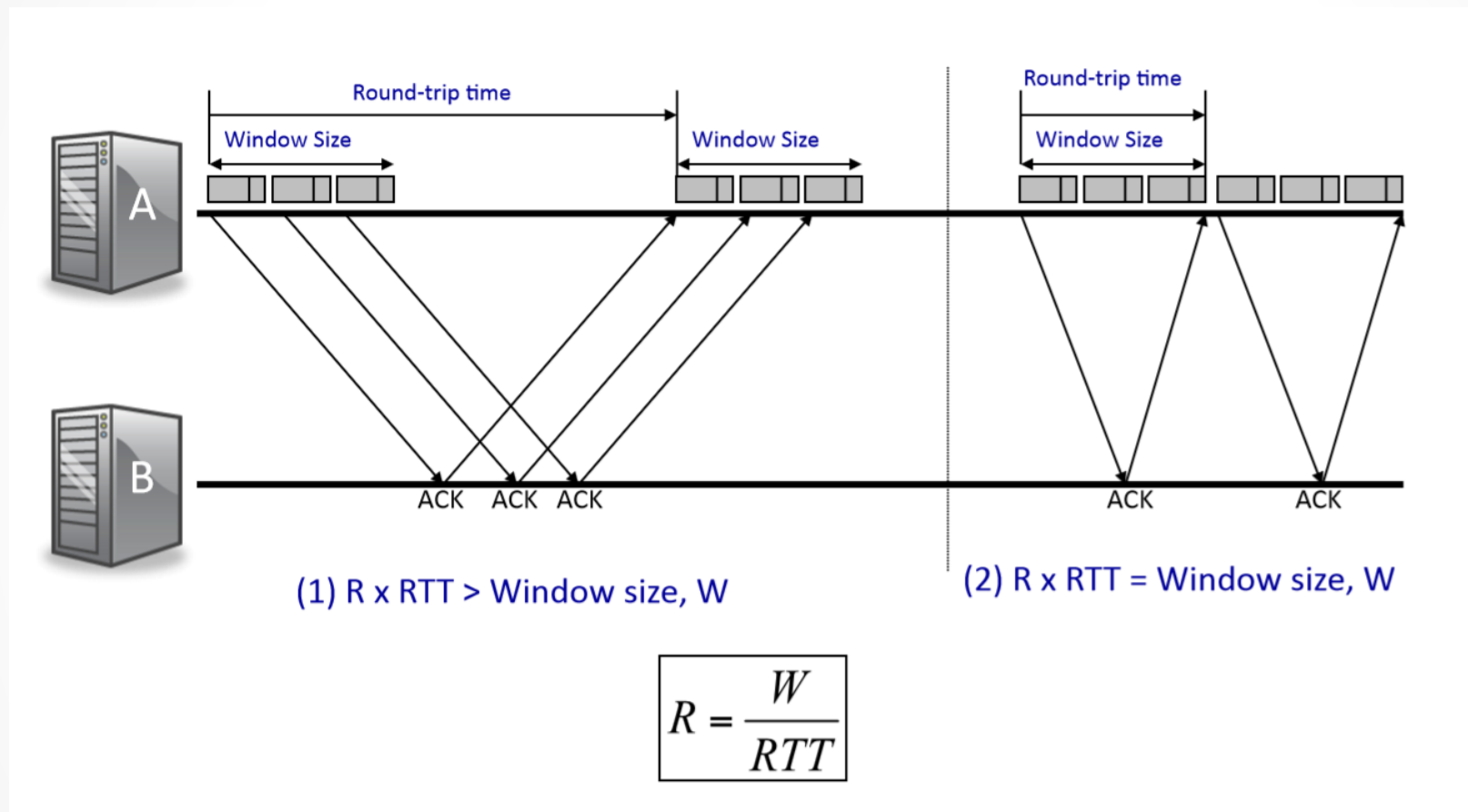


# Примеры динамики одиночного потока



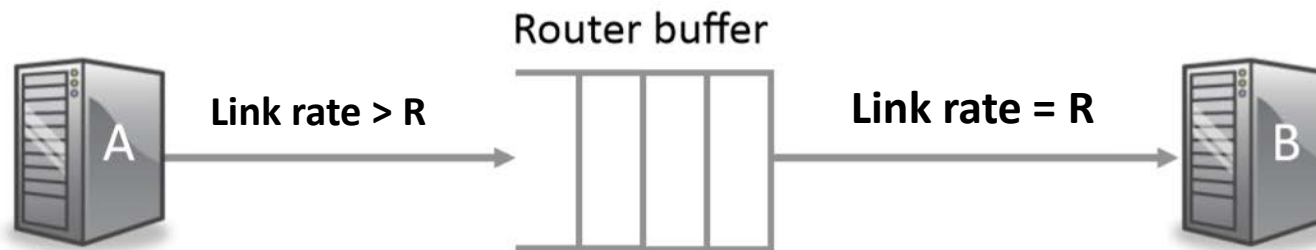
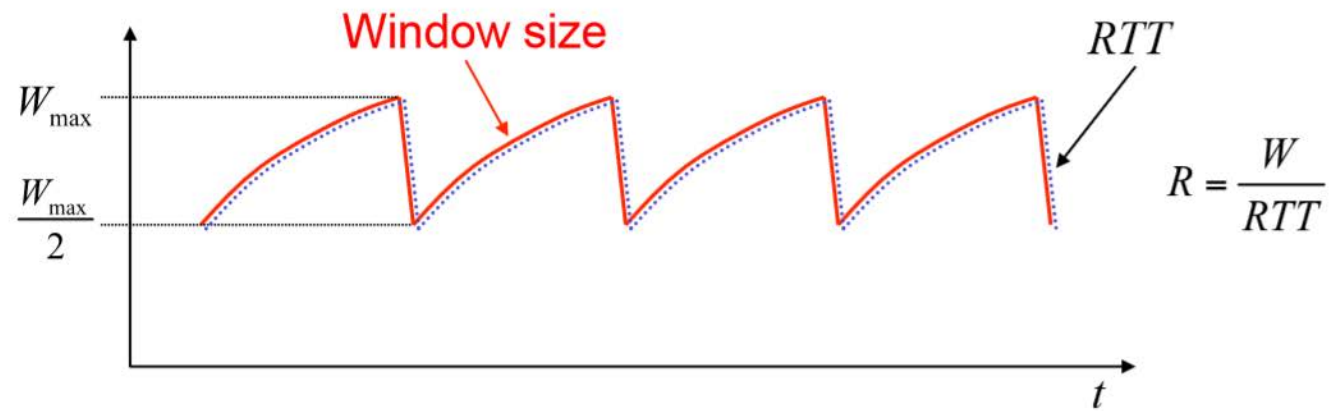


# Скорость отправки для одиночного потока





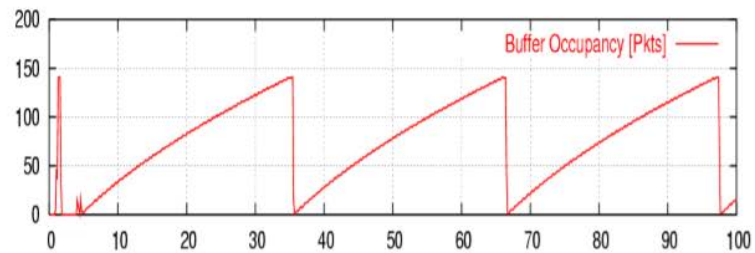
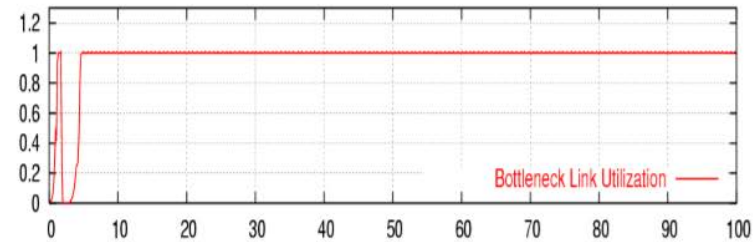
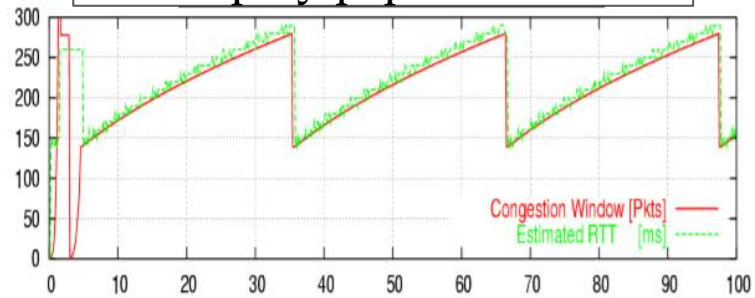
# Скорость отправки для одиночного потока



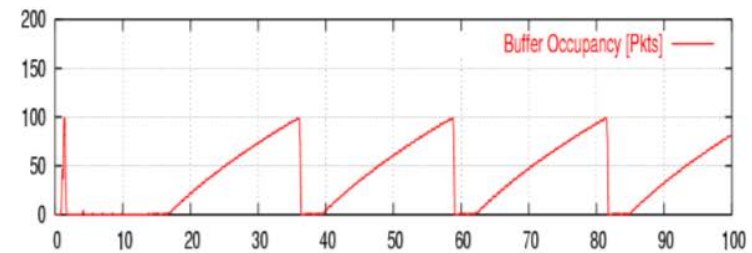
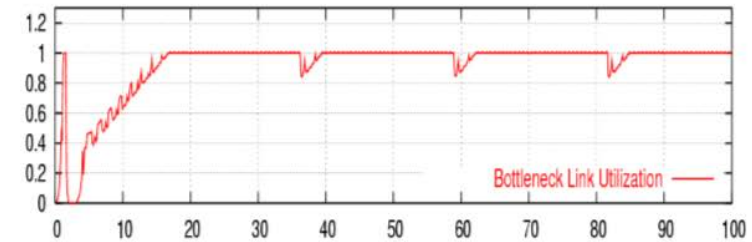
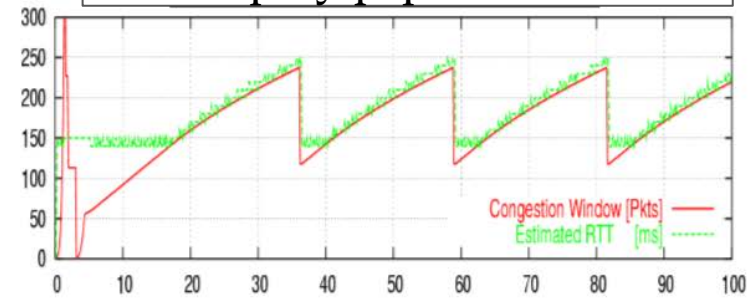


# Насколько большим должен быть буфер?

Размер буфера  $W = RTT * R$



Размер буфера  $W < RTT * R$





# Промежуточные выводы

- В TCP управление перегрузкой располагается на хосте
- Реакция на события, наблюдаемые на конечном хосте (например, потеря пакета).
  - Использование скользящего окна, предназначенного для управления потоком в TCP
  - Стараться как можно быстрее оценить сколько пакетов можно безопасно отправить в сеть одновременно.
  - Изменять размер окна в соответствии с алгоритмом AIMD





# Комментарии для одиночного потока

1. Окно увеличивают, сокращают в соответствии с AIMD
2. ... пробировать как много байт канал еще может вместить
3. Пилообразное поведение - нормальная форма динамики
4. Скорость отправки постоянная
5. Размер буферного пространства определяет соотношение -  $RTT \times R$





# Управление перегрузкой: AIMD с несколькими потоками

Введение в компьютерные сети

чл.-корр. РАН Смелянский Р.Л.

Кафедра АСВК ф-т ВМК МГУ

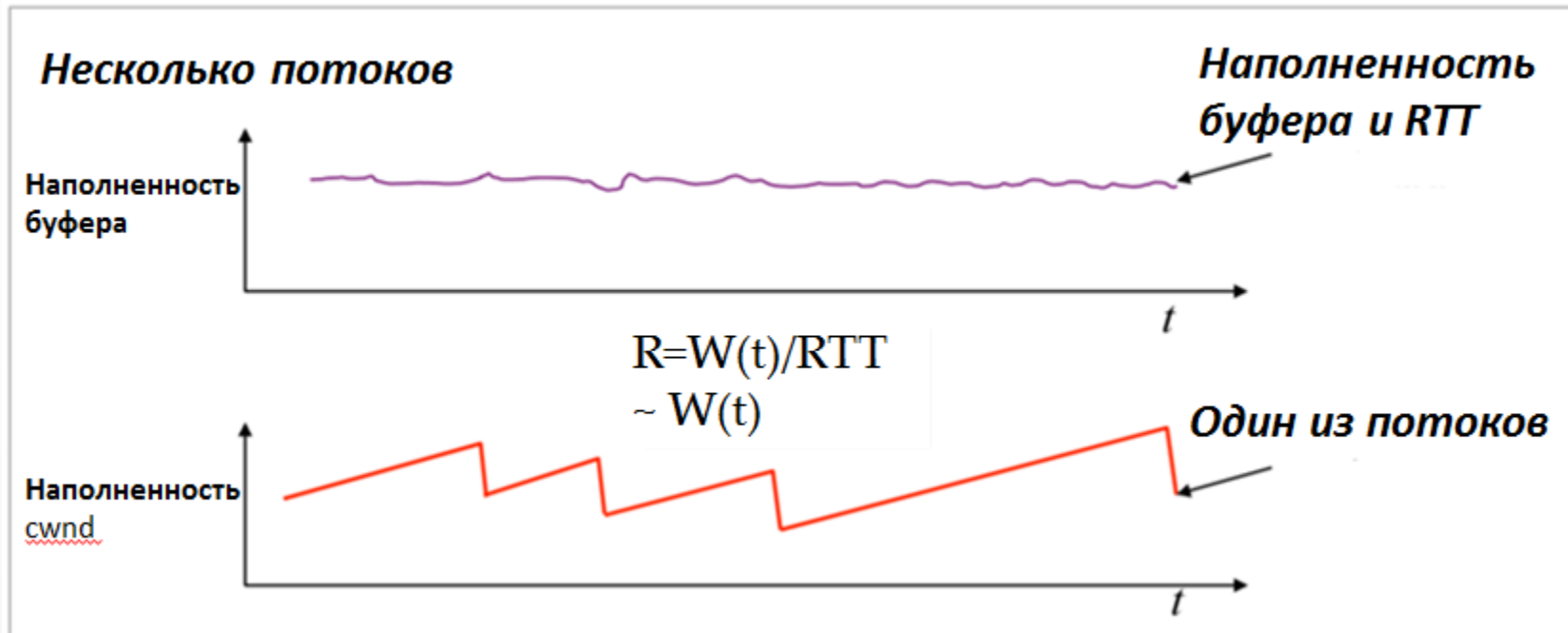
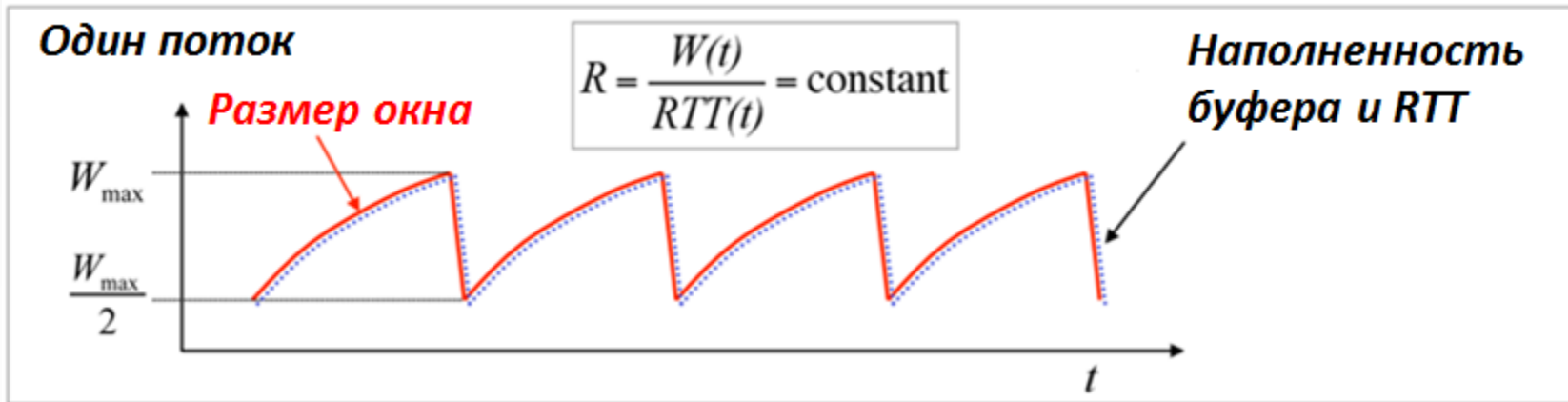


## Буфер маршрутизатора



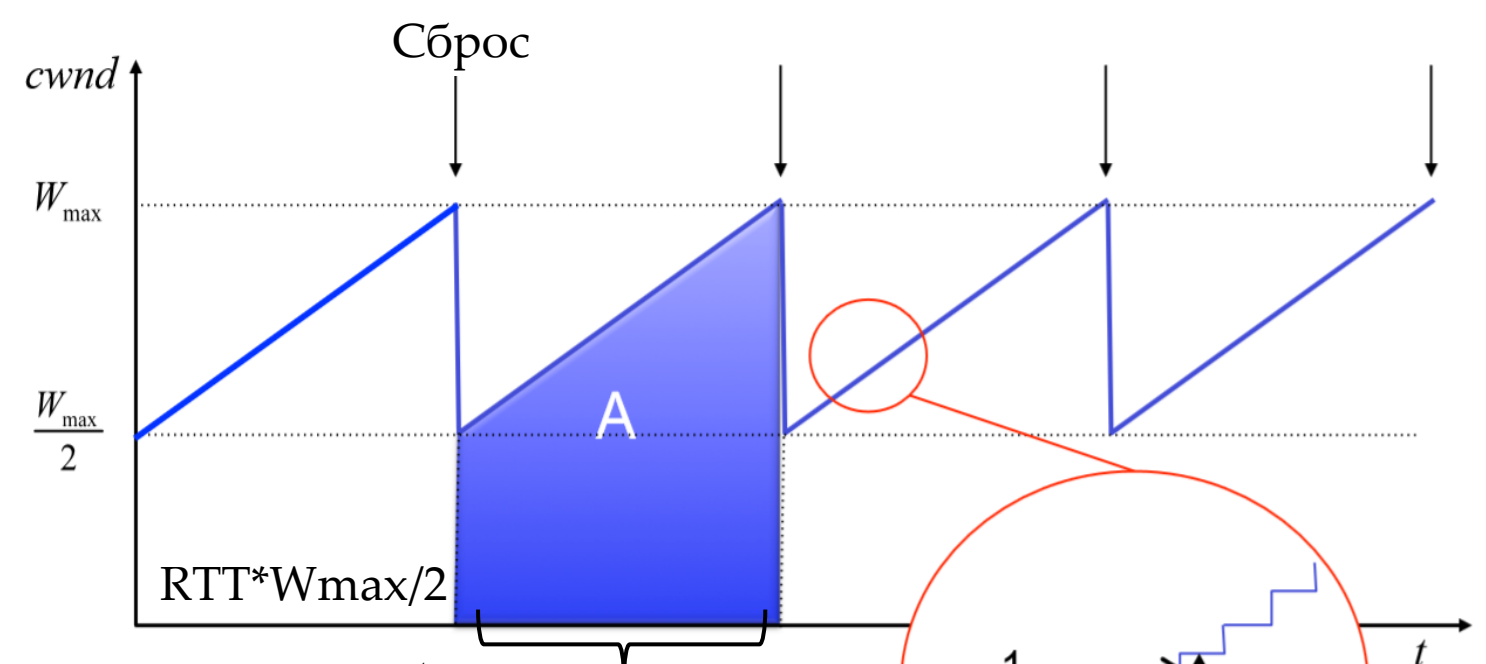


# Один поток vs много потоков





# Зависимость скорости потока от RTT и вероятности сброса пакета



Вероятность сброса пакета

$$p \approx 1/A, \text{ где } A = \frac{3}{8} W_{max}^2$$

**Пропускная способность**

$$R = \frac{A}{\left(\frac{W_{max}}{2}\right) RTT} = \sqrt{\frac{3}{2}} \frac{1}{RTT \sqrt{p}}$$



# Интерпретация уравнения скорости

$$R = \sqrt{\frac{3}{2}} \frac{1}{RTT \sqrt{p}}$$

$$RTT \rightarrow 0 \Rightarrow R \rightarrow \infty$$

$$p \rightarrow 0 \Rightarrow R \rightarrow \infty$$

$$\frac{R_1}{R_2} = \sqrt{\frac{p_2}{p_1}}$$



# Комментарии для нескольких потоков

1. Окно увеличивают/сокращают в соответствии с AIMD
2. ... пробировать как много байт канал еще может вместить
3. В «узком месте» будут скапливаться пакеты разных потоков
4. Скорость отправки меняется в зависимости от размера окна
5. AIMD очень чувствителен к вероятности потери пакетов
6. AIMD ущемляет потоки с большим RTT



# Заключение:

## Управление перегрузками в ТСП

В ТСП управление перегрузкой размещается на  
**конечном хосте**

- Реакция на события, наблюдаемые на конечном хосте (например, потеря пакета).
- Использование скользящего окна, предназначенного для управления потоком в ТСП
- Механизм AIMD позволяет оперативно оценить сколько пакетов можно безопасно отправить в сеть одновременно.





# Реализация управления перегрузкой на практике (базовые алгоритмы)

Введение в компьютерные сети  
чл.-корр. РАН Смелянский Р.Л.  
Кафедра АСВК



# Три основных вопроса

1. *Когда следует посылать новые данные?*
2. *Когда следует посылать данные повторно?*
3. *Когда надо отправлять подтверждения?*



# TCP Pre-Tahoe



# Немного истории

- *1983 - ARPANET переходит на TCP/IP*
- *1986 - Интернет страдает от перегрузок*
- *1987 - Ван Якобсон предлагает TCP Tahoe*
- *1990 - Добавляются режимы быстрого восстановления и быстрой повторной передачи (Reno)*



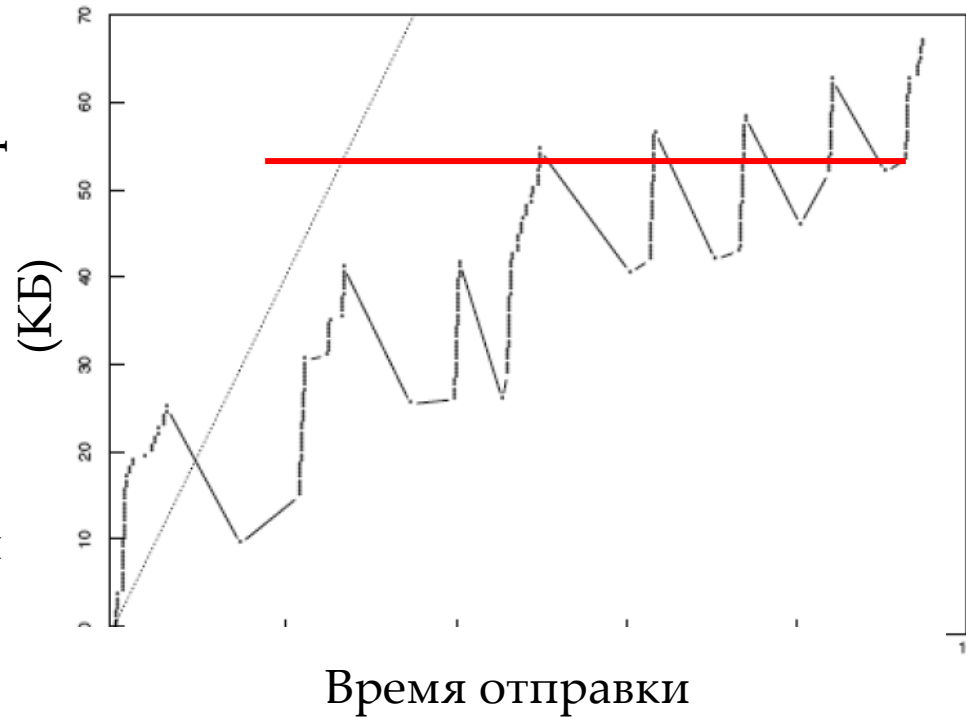
# TCP Pre-Tahoe

- *Получатель устанавливает размер окна управления потоком (размер скользящего окна)*
- *Отправитель шлет пакеты, число которых полностью соответствует размеру скользящего окна*
- *На каждый пакет устанавливают таймер*
- *Проблема: что будет если размер окна превышает пропускную способность сети?*



# TSP образца 1986

Последовательные номера пакетов



Переотправка  
одного пакета 4  
раза!

Рисунок из статьи ван Якобсона и Карела



# TCP Tahoe





# Три усовершенствования ТСР

- *Окно перегрузки (CWND)*
- *Оценка Time\_out*
- *Саморегулирование (Self-clocking)*



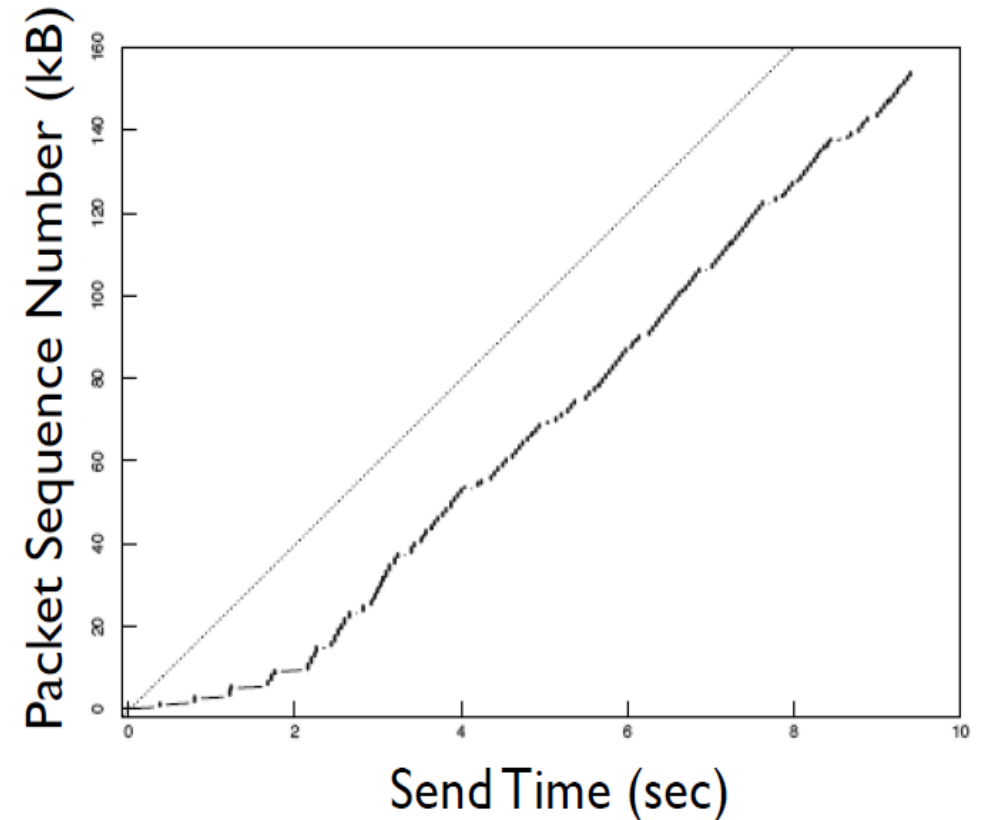
# Окно перегрузки (ТСР Tahoe)

- Отправитель
  - узнает от получателя *FCWND* (*Flow Control WND* - окно управления потоком)
  - оценивает размер *CWND*
- Окно отправителя =  $\min(\text{FCWND}, \text{CWND})$
- Разделение фазы управления перегрузкой на две
- Медленный старт
- Предотвращение перегрузки - стабилизация



# Медленный старт

- Медленный старт
- $CWND = MSS$
- На каждый ACK увеличиваем окно на MSS
- Экспоненциально увеличиваем (удваиваем) размер окна перегрузки, прощупывая возможность сети
- «медленный» по сравнению с изначальным алгоритмом



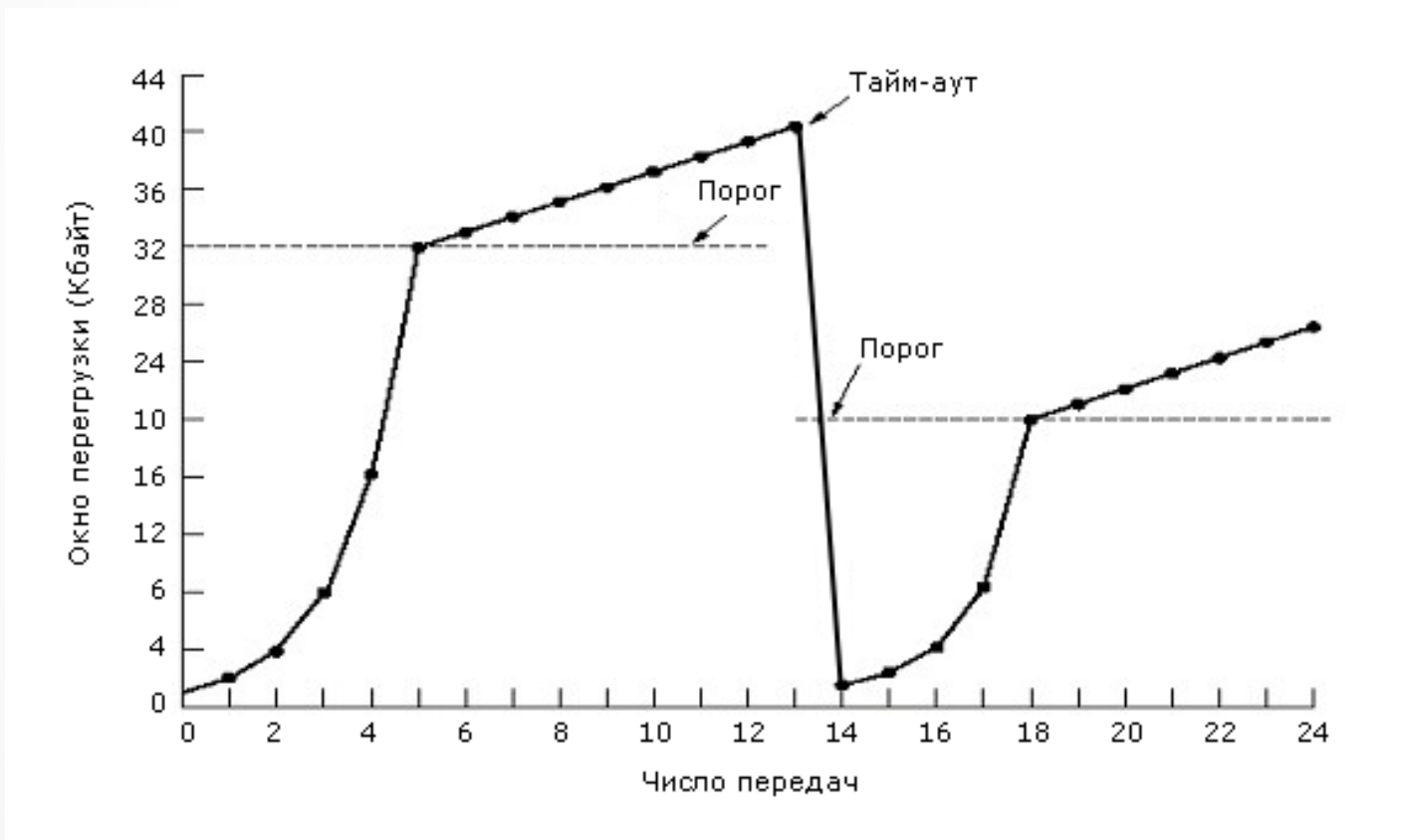


# Предотвращение перегрузки

- Медленный старт (*slow start*)
  - Увеличиваем  $CWND$  на  $MSS$  на каждое подтверждение
  - Экспоненциальный рост (за  $RTT$  удваиваем  $cwnd$ ) пока не достигнем порога ( $cwnd/2$  в предыдущей фазе предотвращения перегрузки) или не обнаружим перегрузку
- Предотвращение перегрузки (*congestion avoidance*)
  - Увеличиваем окно перегрузки только на  $MSS^2 / CWND$  при каждом подтверждении
  - За каждый  $RTT$  ( $cwnd/MSS$ ) увеличиваем  $cwnd$  на  $MSS$
  - Линейный рост

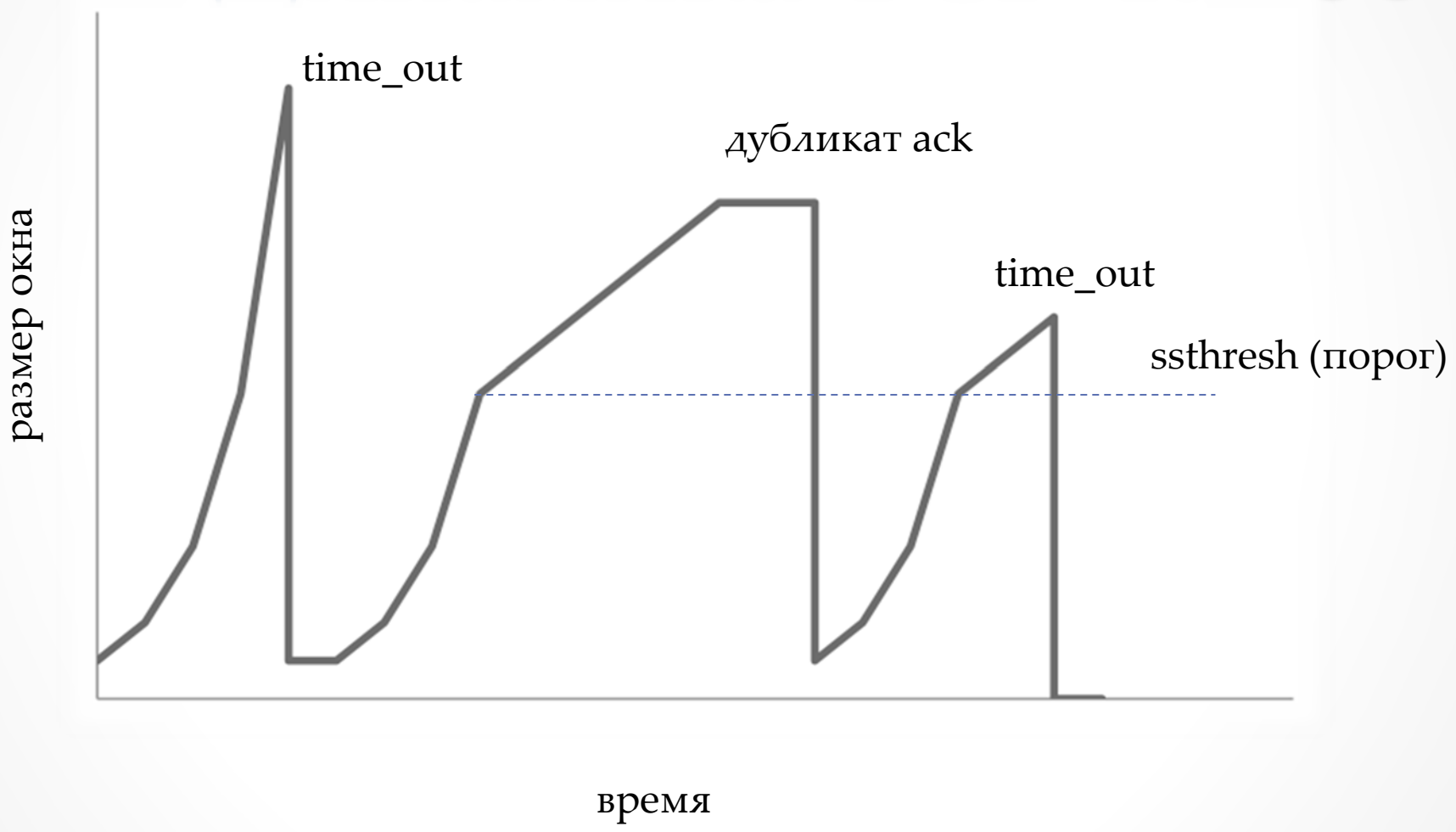


# Пример управления перегрузками





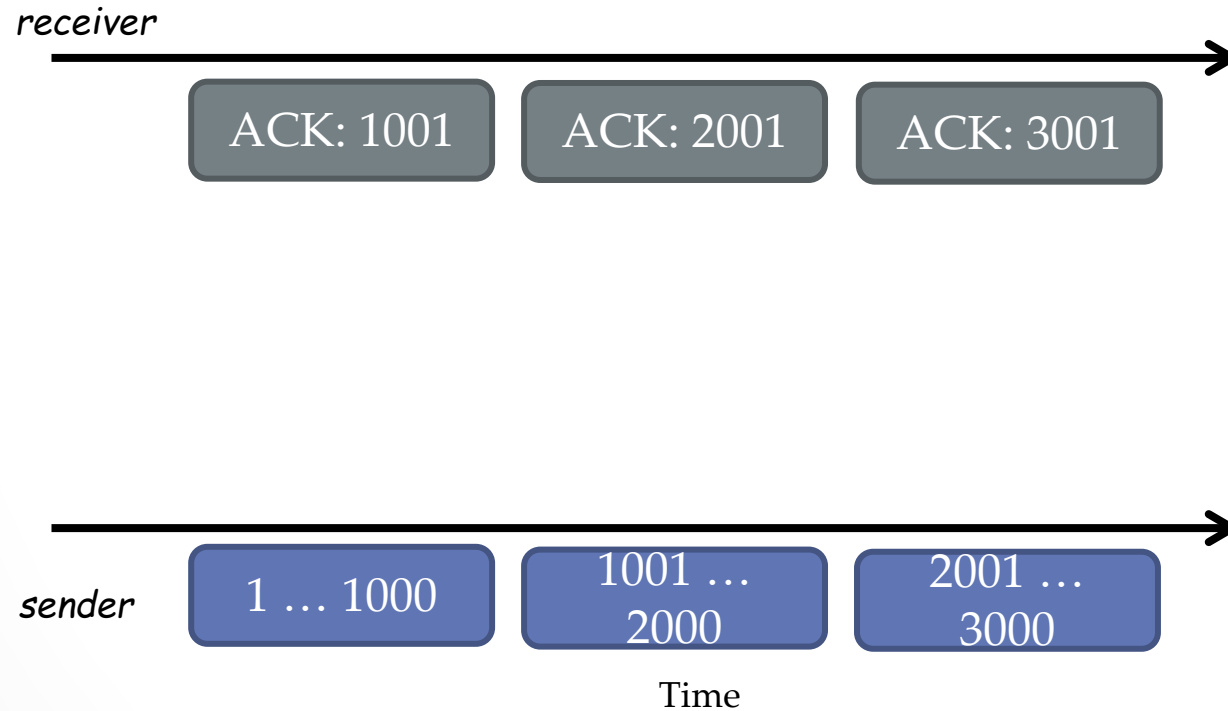
# Динамика ТСР Tahoe





# Механизм подтверждений

- В подтверждении содержится номер следующего ожидаемого байта

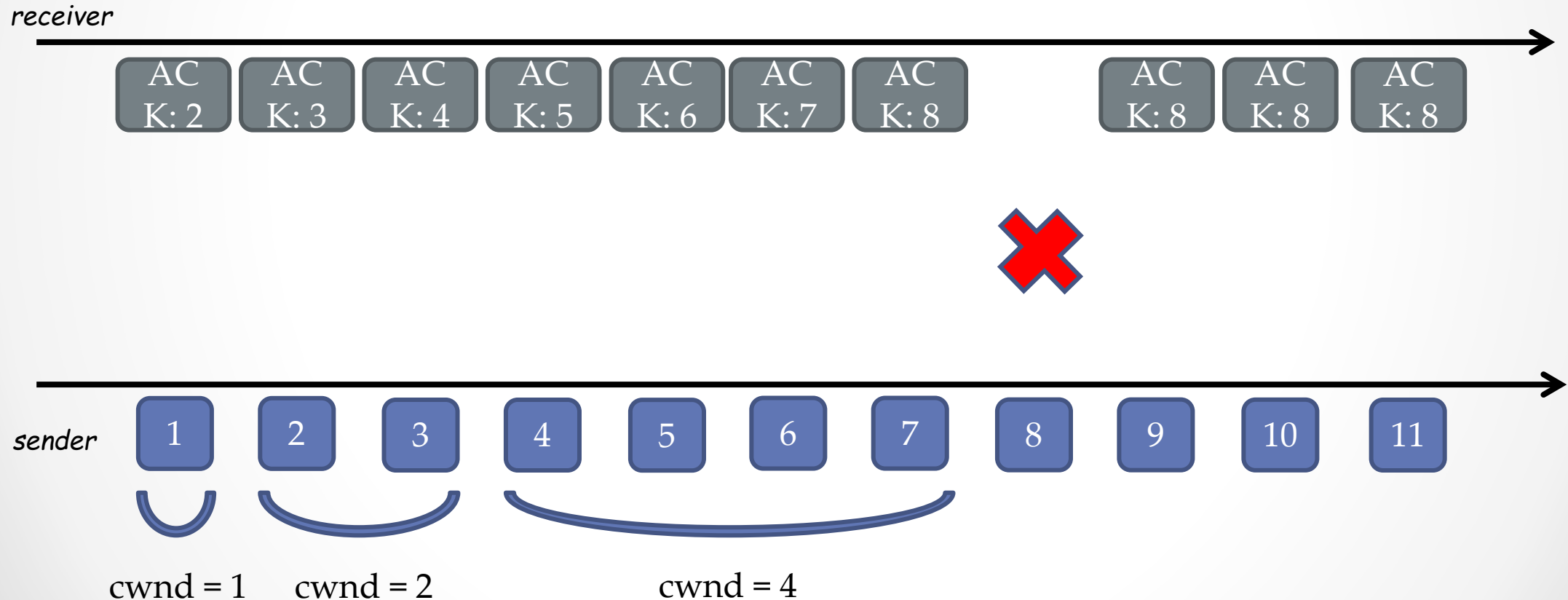






# Пример работы ТСР Tahoe

- Для упрощения будем говорить не о байтах, а о пакетах.



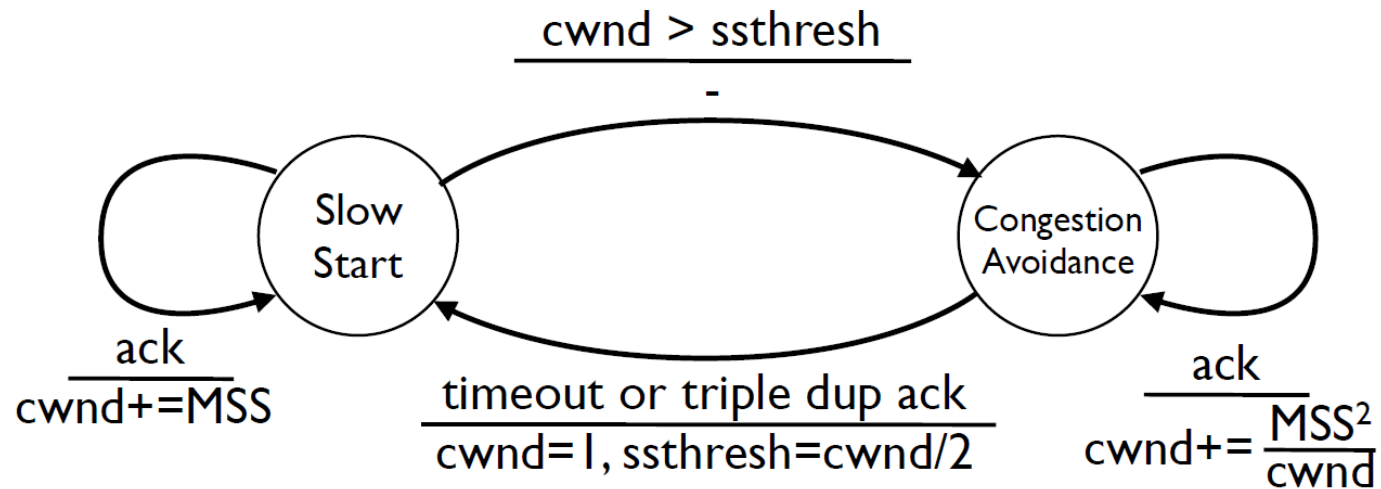


# Стратегия Tahoe

- Стратегия:
  - Используя медленный старт, быстро нащупать доступную пропускную способность сети
  - Приблизившись к насыщению, перейти в режим предотвращения перегрузки, очень осторожно пробирая возможность роста
- Признак перегрузки - Потеря пакета
  - по таймеру;
  - 3 повторных подтверждения (всего 4 подтверждения с одинаковым номером).
- Три сигнала:
  - Рост номеров уведомлений - передача данных идет хорошо
  - Повторные уведомления - где-то произошла задержка/потеря данных, прекратить увеличение окна перегрузки
  - Если наступил `Time_out` или три раза получили `dup ACK` с одним и тем же номером, то устанавливаем порог =  $swnd/2$ ,  $swnd = 1$  и переходим в фазу медленного старта
  - Если пришло ACK с нужным номером, то продолжаем в фазе предотвращения перегрузки



# Диаграмма состояний ТСР Tahoe



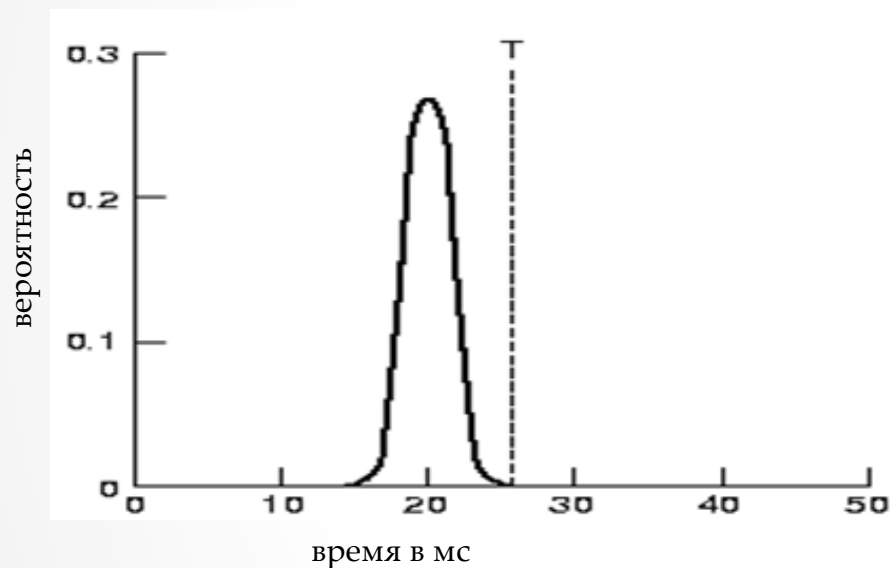


# Оценка time-out

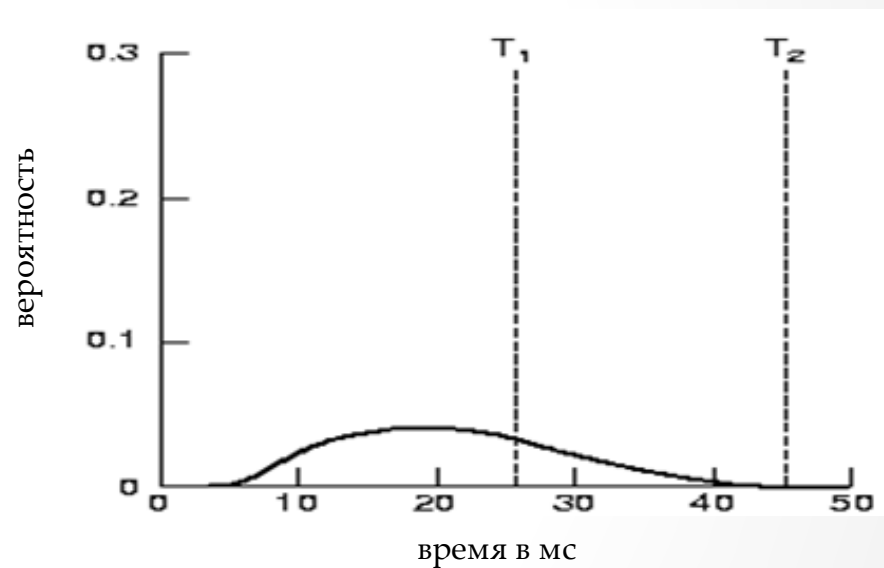
- *RTT измерение критично для оценки time-out*
  - *Если слишком коротко - впустую тратим ресурсы сети на повторные передачи, «ломаем» медленный старт*
  - *Если слишком длинный - зря тратим ресурсы на ожидание*
- *Трудности -*
  - *RTT меняется очень динамично*
  - *RTT сильно зависит от загрузки (load) сети*



# Распределение задержки на канальном и транспортном уровнях



На канальном уровне



На транспортном уровне



# Pre Tahoe time\_out

- $r$  - переменная, начальная оценка RTT
- $m$  - измерение RTT для последнего подтвержденного пакета
- Вычисляем взвешенное среднее -

$$r := a * r + (1 - a) * m, \text{ где } 0 < a < 1 \text{ (обычно } 7/8)$$

- $\text{Time\_out} = \beta * r$ , где  $\beta = 2$
- В чем проблема?  
 $\beta$  - постоянная величина и не учитывает разброс значений  $m$



# TCP Tahoe time-out

(т.2 стр.132-134)

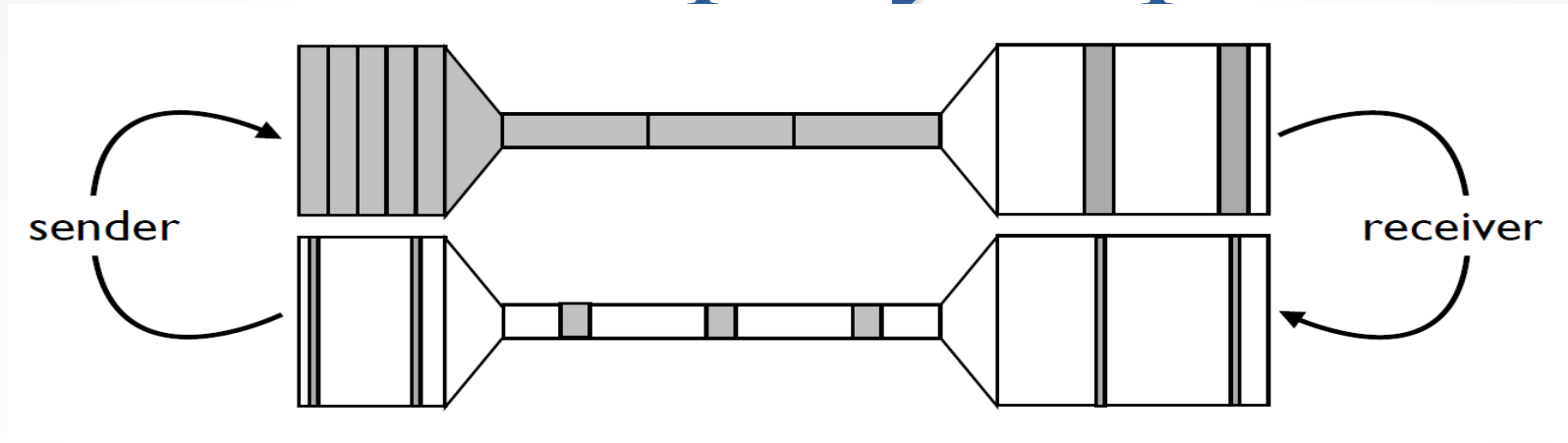
- $r$  - переменная, начальная оценка RTT
- $m$  - измерение RTT для последнего подтвержденного пакета
- Ошибка -  $e := m - r$ .
- Вычисляем  $r := ar + (1-a)m$ , где  $1-a \sim 0.125$
- Измеряем вариацию -  $v := av + (1-a)|e|$  (линейное отклонение)
- $time-out = r + \beta * v$ , где  $\beta = 4$
- В случае повторной передачи  $RTT = \beta * time-out$

**Перед Вами пример алгоритма с машинным обучением!**





# Саморегулировка



*Отправлять данные только после того, как предыдущие покинули сеть*  
*Посылать данные только при получении уведомления*  
*Отправлять уведомления как можно быстрее - это важно!*

*Прочсть «Congestion Avoidance and Control» van Jacobson and Karels*

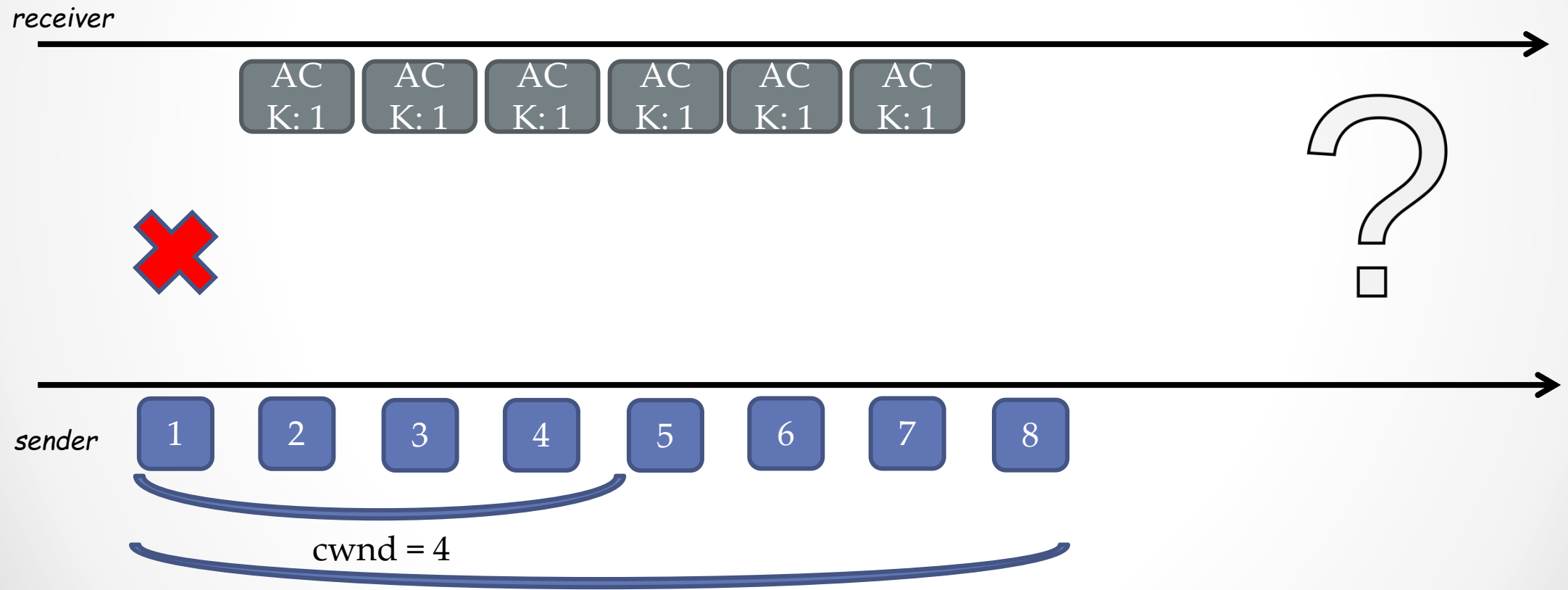


# TCP Reno



# Быстрое восстановление

Какая должна быть реакция на большое количество повторных подтверждений ?





# TCP Tahoe vs TCP Reno

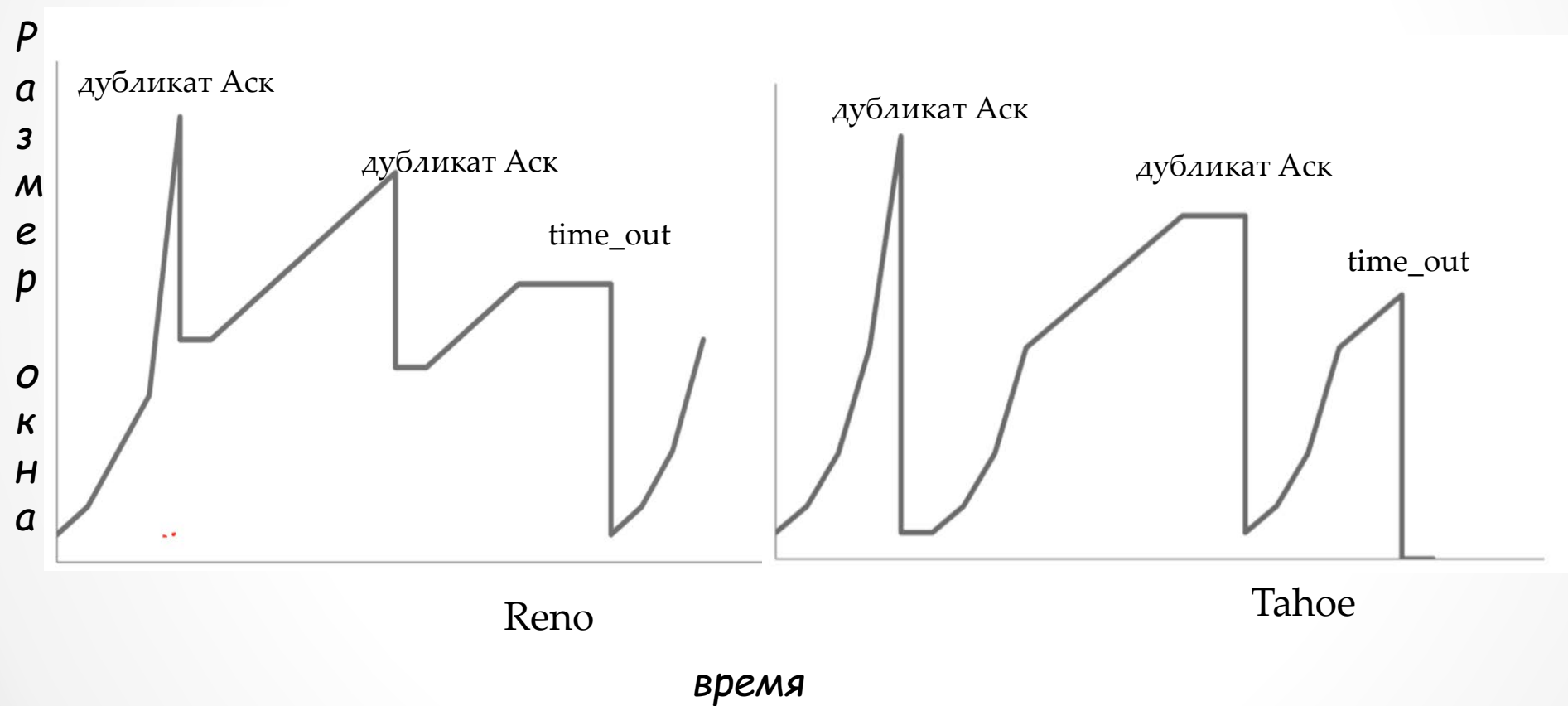
- TCP Tahoe - Не ждать *time-out*, а уже по тройному уведомлению (подразумевается потеря пакета) начинать повторную пересылку и переход в фазу медленного старта
  - установить порог =  $CWND/2$
  - сбросить  $CWND$  в 1
  - начать медленный старт
- TCP Reno - Добавлены фазы быстрой пересылки (*fast retransmit*) и быстрого восстановления (*fast recovery*):
  - по тройному ACK
    - $cwnd = cwnd/2$ , а не 1
    - шлем запрашиваемый сегмент (*fast retransmit*)
    - Переходим в фазу быстрого восстановления (*fast recovery*)
  - По *time-out* схема Tahoe



# TCP Tahoe vs TCP Reno

## TCP Reno

## TCP Tahoe





# Быстрое восстановление

- *Окно перегрузки определяет количество пакетов, которое может одновременно находиться в сети*
- *Повторное подтверждение означает, что некоторый пакет был доставлен не по порядку -> в сети стало меньше пакетов, чем `swnd`*
- *3 повторных подтверждения означают, что 3 пакета покинуло сеть*
- *Проблема: мы не можем сдвинуть `sliding window` вправо на 3 позиции из-за неподтвержденного пакета*
- *Решение: т.к `sliding window` > `congestion window`, то мы можем продолжить отправку пакетов, не боясь перегрузить получателя.*



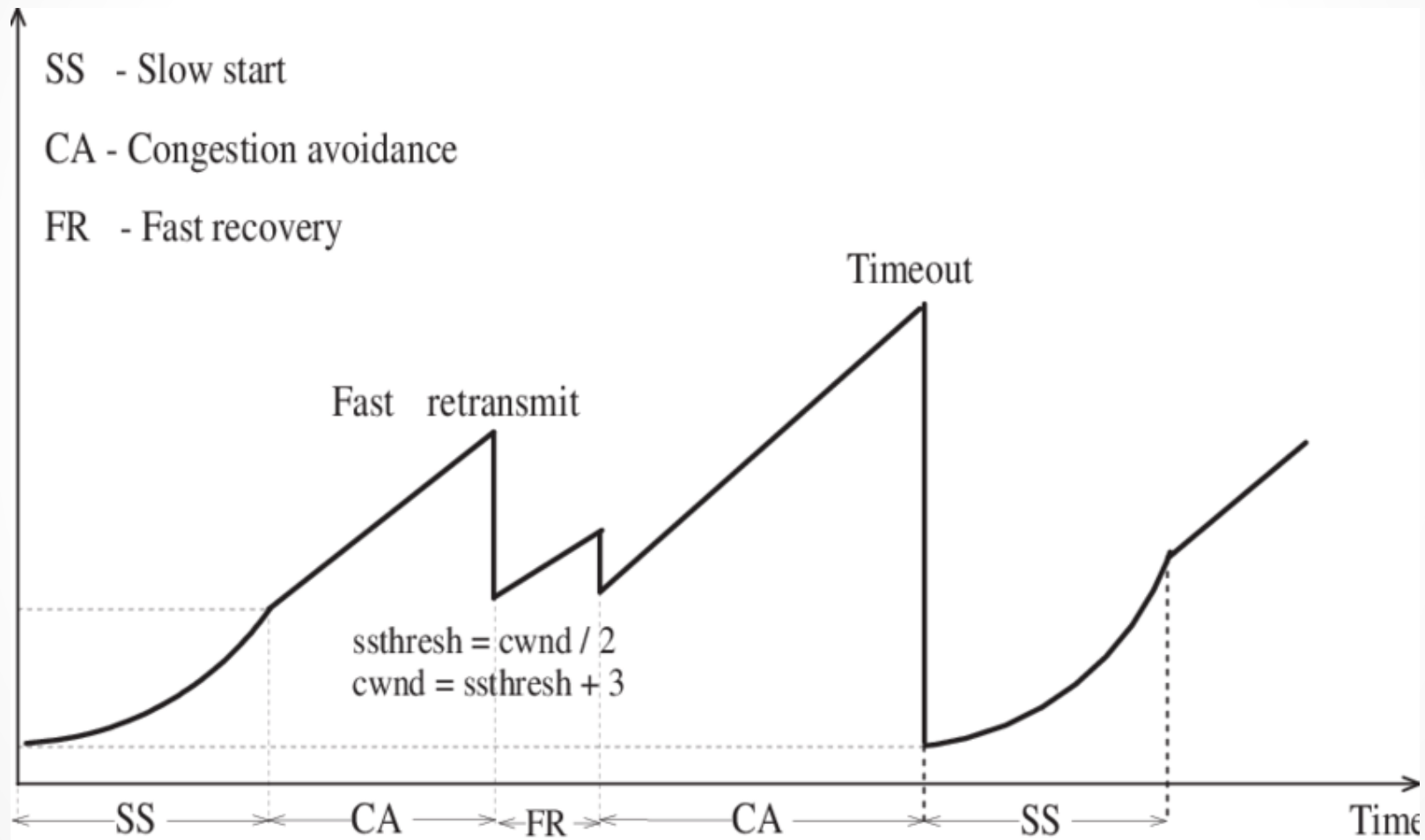
# Быстрое восстановление

- В начале фазы быстрого восстановления устанавливаем окно в  $swnd/2 + 3$
- Каждое последующее повторное подтверждение означает выход еще одного пакета из сети, поэтому на каждый dup ack увеличиваем  $swnd$  на 1:  $swnd += 1$
- При получении подтверждения на переотправленный пакет мы можем двигать левую границу  $swnd$ , поэтому возвращаем значение  $swnd$  в  $swnd/2$ .





# Быстрое восстановление





# Определение перегрузки

- Потеря пакета
  - по таймеру;
  - 3 повторных подтверждения
- По изменению RTT (при перегрузке увеличивается задержка буферизации)
- По уведомлениям из сети (ECN)
- Гибридные методы

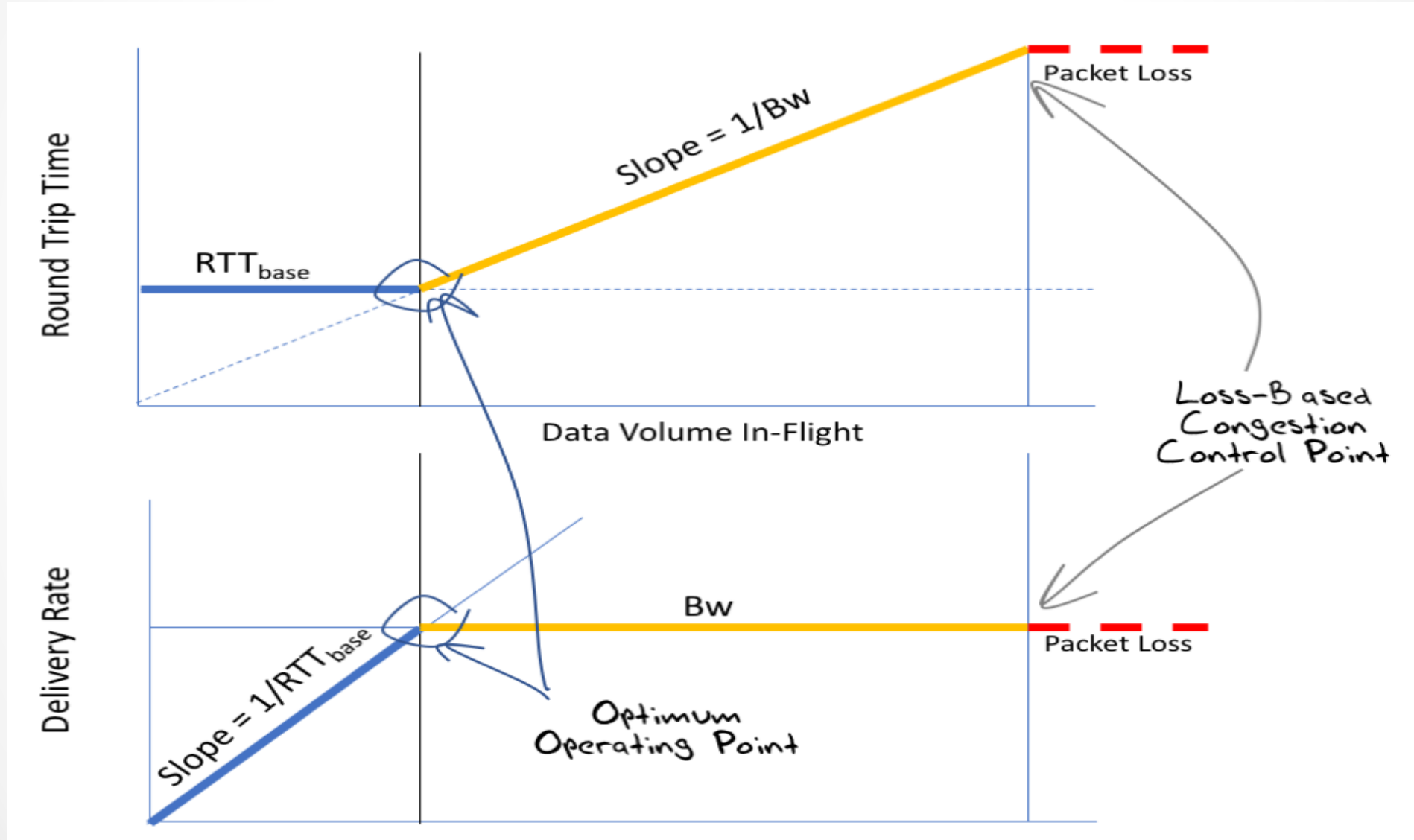


# BBR - Delay Based (2018)

NEAL CARDWELL  
YUCHUNG CHENG  
C. STEPHEN GUNN  
SOHEIL HASSAS YEGANEH  
VAN JACOBSON

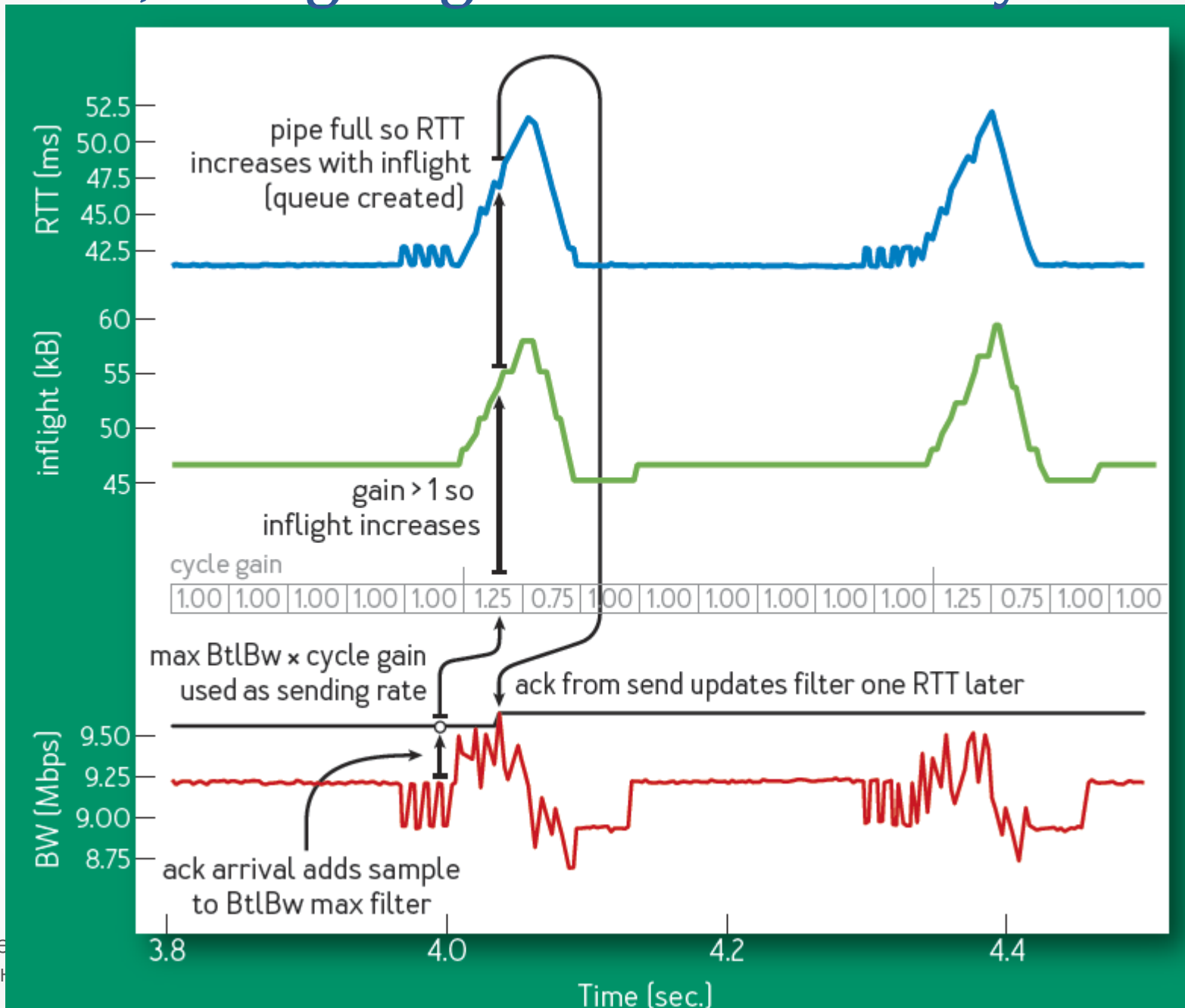


# BBR





# RTT (blue), Inflight (green) and Delivery rate (RED)





# Причины изменения RTT

- Увеличение RTT вызвано
  - Изменением маршрута
  - Изменением пропускной способности узкого места
  - Увеличением задержки буферизации из-за трафика другого соединения
- Измеряем пропускную способность узкого места и минимальный RTT
- Временно увеличиваем скорость отправки:
  - Если измеряемая пропускная способность увеличилась – можем занять больше ресурсов
  - Если увеличилась задержка, то временно замедляем скорость для освобождения буфера
- Не реагируем на потери

## RCP – Rate Control Protocol



# Управление перегрузками: Заключение

- *Одна из сложнейших проблем в сетях передачи данных существенно влияющих на их производительность*
- *Решение этой проблемы - это всегда компромисс между разными характеристиками транспортных соединений*
- *Основные известные подходы: loss based (AIMD (additive increase, multiple decrease)), delay based (BBR), Rate Control*
- *Пути повышения эффективности транспортных соединений - FEC вместо BEC, многопоточная маршрутизация, упорядоченный поток UDP дейтаграмм*
- *Дома построить диаграмму состояний для TCP Reno*