

# Современные алгоритмы управления перегрузкой ТСП

м.н.с. Степанов Евгений Павлович

# Программа курса

## Подходы:

- 1. Управление перегрузкой**
  - Современные протоколы управления перегрузкой TCP
- 2. Демультимплексирование/мультиплексирование**
  - Многопоточные транспортные протоколы
  - Маршрутизация на уровне интернет провайдеров
  - Network Coding
- 3. Сегментация**
  - TCP Proxy
- 4. Балансировка**
  - Балансировка нагрузки и управление трафиком

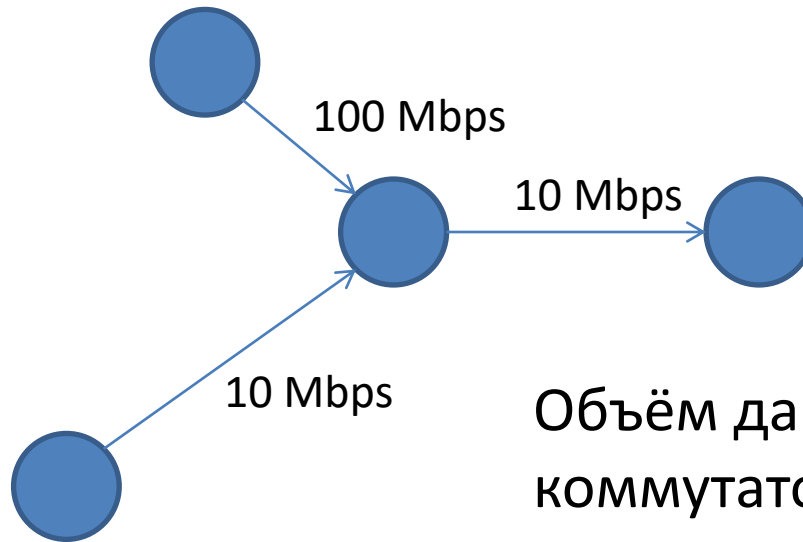
## Модели:

- Сетевое исчисление: математический подход к качеству сервиса
- NS3: моделирование поведения сети с высокой точностью

## Примеры:

- HTTP3/QUIC
- Управление сетевыми ресурсами в Центрах Обработки Данных
- Обеспечение качества сервиса в сетях доставки контента
- Пропускная способность по требованию

# Перегрузки в сети



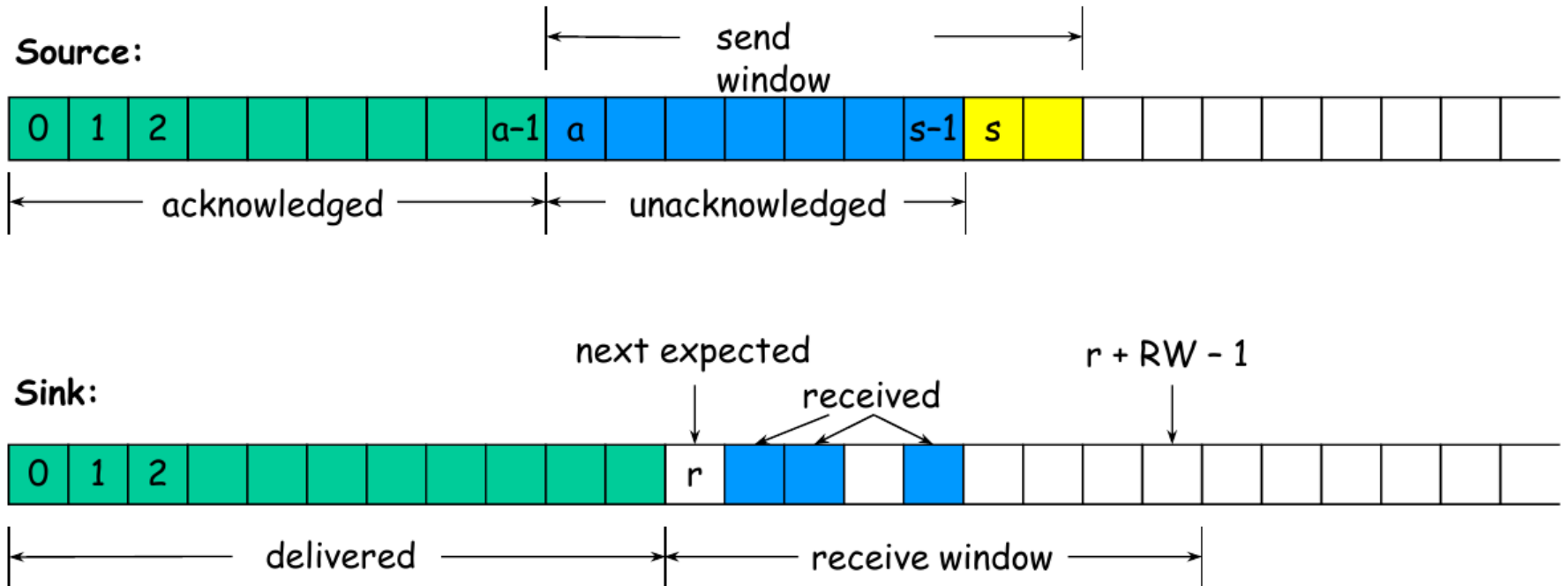
Объём данных, поступающих на коммутатор, превышает объём данных, которые он может передать

Коммутаторы буферизируют проходящий через них трафик:

- При кратковременной перегрузке увеличивается задержка
- При достаточно долгой перегрузке пакеты сбрасываются

# Базовое устройство ТСР

## Алгоритм скользящего окна (sliding window)



- Позволяет восстанавливать порядок пакетов при отправке множества пакетов с использованием буфера фиксированного размера

# Управление ТСП трафиком

*какими бывают “окна”*

- Rate control (traffic policing)
- Receiver-side flow control
  - Цель: избежать перегрузки получателя
  - Параметр: RWND – окно получателя
  - Получатель отправляет свой RWND отправителю
- Network congestion control
  - Цель – избежать перегрузки сети
  - Параметр: CWND – окно перегрузки
  - Отправитель вычисляет свой CWND с помощью алгоритма управления перегрузкой

# Cumulative ACK

- Получатель сообщает о последнем успешно полученном пакете – пакеты с меньшими номерами считаются полученными автоматически
- NACK (negative ACK) – получатель явно перечисляет пакеты, которые он не получил
- SACK (selective ACK) – получатель сообщает диапазоны успешно полученных пакетов

# Небольшая история Интернет

- 1974 год – первый драфт TCP/IP
- 1983 год – ARPANET переходит на TCP/IP
- 1986 год – ARPANET испытывает сильнейшую перегрузку (congestion collapse)
- 1988 год – появляется управление перегрузкой для TCP

## *TCP design principles:*

- **Van Jacobson** "Congestion avoidance and control" // Proceedings of SIGCOMM '88, Stanford, CA. Aug. 1988.

# Принцип инвариантности количества отправляемых пакетов

*packet conserving principle*

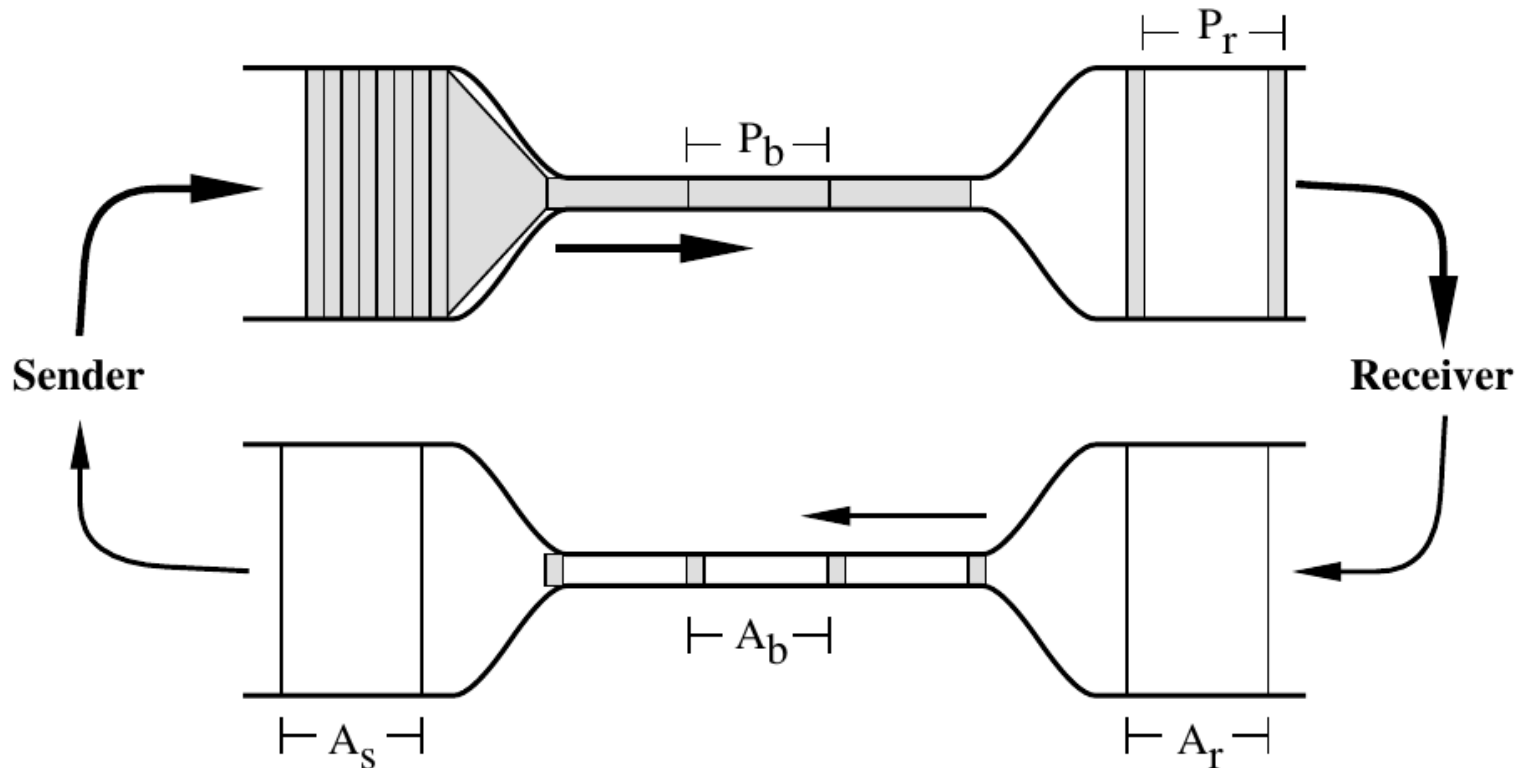
Каждое соединение должно стремиться к *устойчивому состоянию*, в котором отправка каждого следующего пакета соединения происходит сразу после того, как завершилась передача одного из его предыдущих пакетов:

- Каналы не должны простаивать
- Передача пакетов должна быть равномерной, а её итоговая скорость должна соответствовать пропускной способности канала



# Как найти состояние равновесия?

*self-clocking principle*



Скорость отправки пакетов должна соответствовать пропускной способности канала в узком месте

# Slow start

*подгонка часов*

- На отправителе появляется новый параметр – congestion window (CWND)
- Значение CWND устанавливается в 1 в начале соединения и при каждой потере
- При получении каждого подтверждения о доставке CWND увеличивается на 1
- Отправитель передаёт  $\max(\text{CWND}, \text{RWND})$  где RWND – окно получателя

Медленный старт совсем не медленный – количество отправленных пакетов растёт в геометрической прогрессии

# Как остаться в состоянии равновесия?

## *Retransmit timer*

- Если пакет теряется, то отправитель должен отправить в сеть новый пакет в нужный момент времени
- Нужен надёжный расчёт для RTT и RTO

$$RTT_{avg} = \alpha RTT_{avg} + (1 - \alpha) R_{curr}$$

$$RTO = \beta RTT_{avg}$$

# Выбор таймера при повторении ошибок

Перегруженная система обретает стабильность, если нагрузка падает по экспоненциальному закону:

- Exponential Backoff
- RTO, 2RTO, 4RTO, 8RTO, 16RTO, 32RTO

# Как остаться в состоянии равновесия?

## *Congestion avoidance*

- Вероятность потери из-за ошибки на физическом уровне обычно мала ( $\sim 10^{-10}$ )
- Потери возникают из-за перегрузок в сети
- Стратегия:
  - Сеть подаёт сигнал о потере пакета
  - Отправитель снижает нагрузку

**If packet loss is (almost) always due to congestion and if a timeout is (almost) always due to a lost packet, we have a good candidate for the 'network is congested' signal.**

**- Congestion Avoidance and Control**

*+ Не нужно изменять стек протоколов на устройствах*

# Как остаться в состоянии равновесия?

*Additive Increase Multiple Decrease (AIMD)*

Политика изменения окна при перегрузке:

- Пусть  $L_i$  – загрузка очередей на интервале  $i$
- В состоянии равновесия  $L_i = N$
- В состоянии перегрузки  $L_i = N + \gamma L_{i-1}$
- Поскольку загрузка очередей растёт экспоненциально, то размер окна перегрузки тоже должен меняться по экспоненте  $W_i = dW_{i-1}$  ( $d < 1$ )

# Как остаться в состоянии равновесия?

*Additive Increase Multiple Decrease (AIMD)*

Изменение окна при нормальной работе:

- Скорость изменения окна должна быть меньше, чем при перегрузке, иначе система будет нестабильной
- Можно построить такую смесь трафика, что экспоненциальное увеличение (умножение) размера окна приведёт к тому, что пакеты будут отправляться в сеть быстрее, чем она сможет их обрабатывать

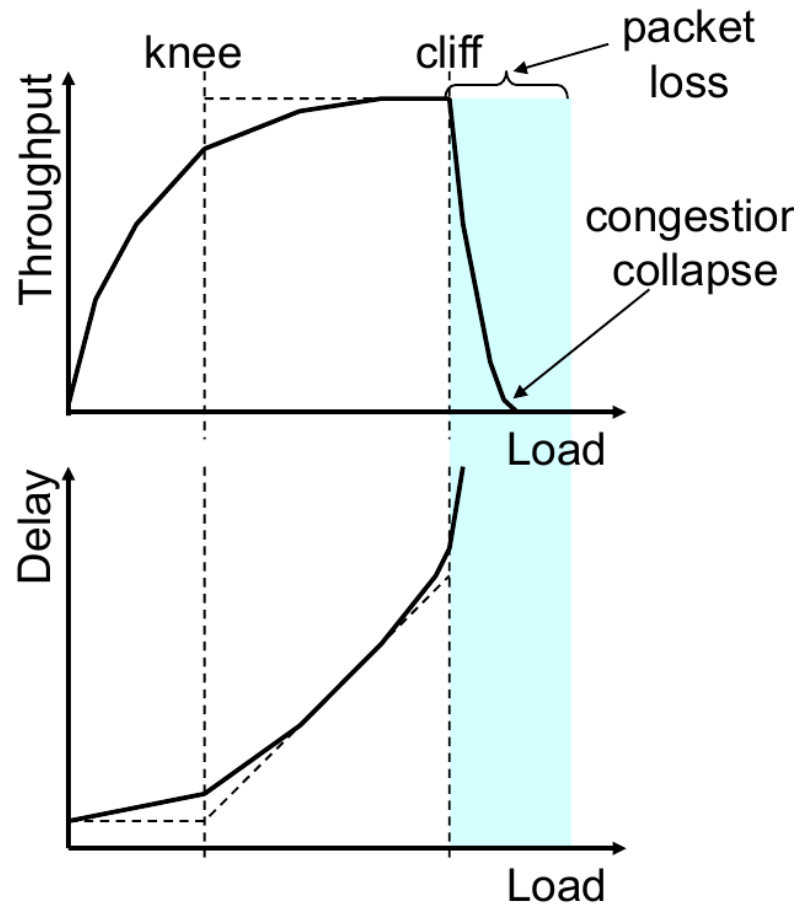
$$W_i = W_i + u \quad (u \ll W_{max})$$

# Управление перегрузкой ТСР: задачи

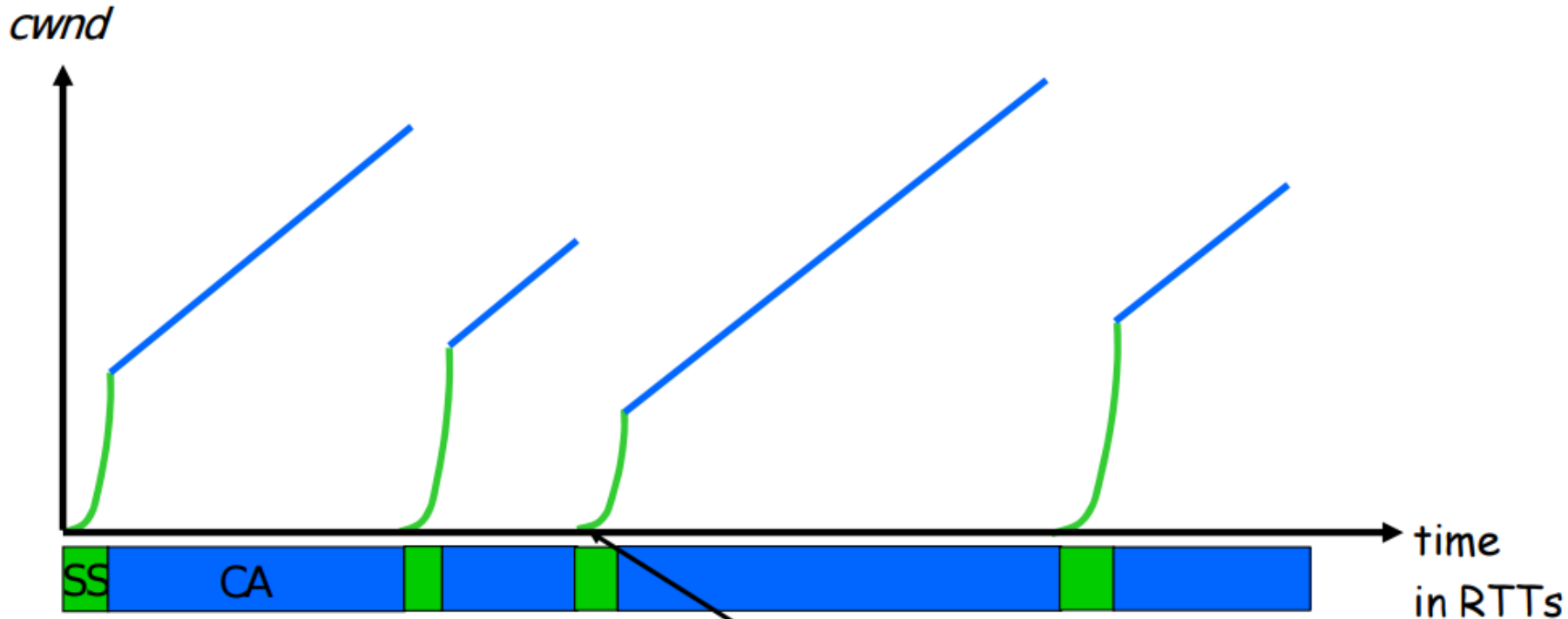
- Соединения должны адаптироваться к качеству предоставленной линии связи и стремиться использовать предоставленные ресурсы максимально *эффективно*
- Соединения должны автоматически распределять пропускную способность разделяемой ими линии связи *справедливым* образом



# Алгоритмы управления перегрузкой протокола TCP



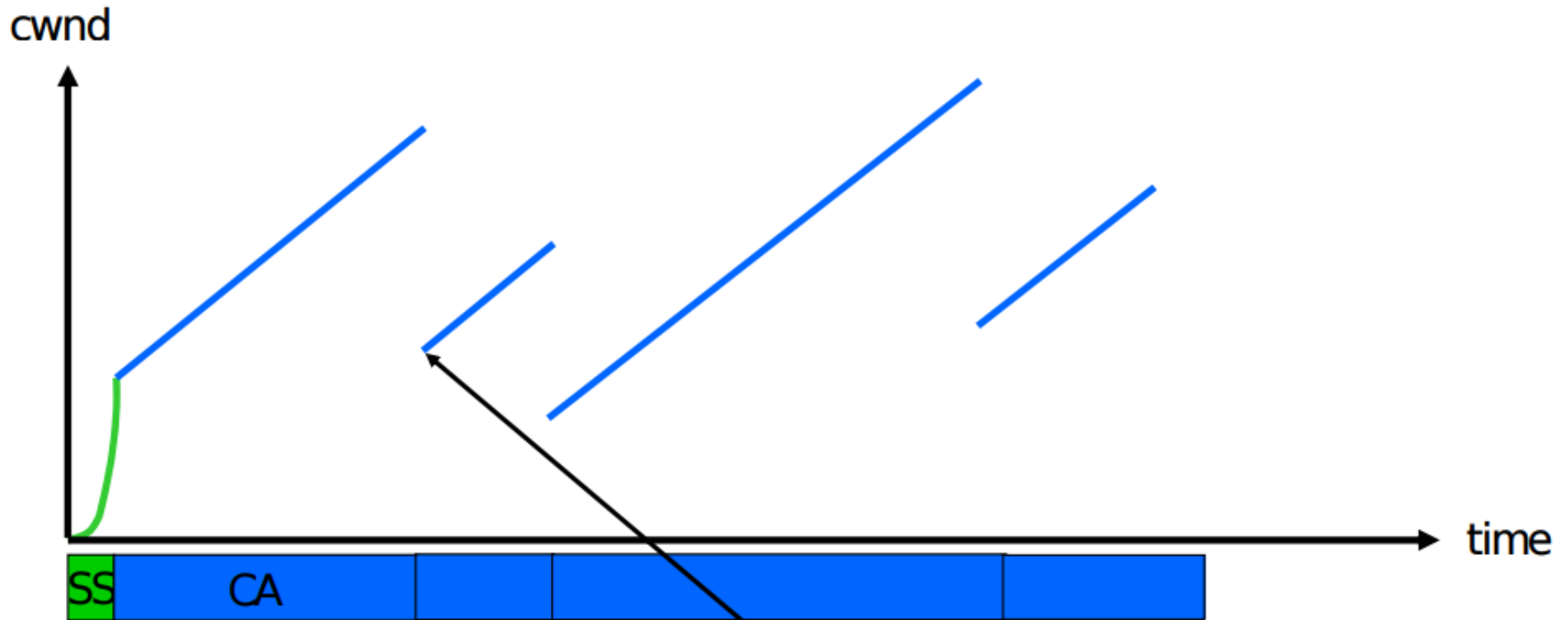
# TCP Tahoe (Jacobson 1988)



SS: Slow Start  
CA: Congestion Avoidance

- Decrease to 1 for either timeout or 3 dupACKs
- Fast retransmit on 3 dupACKs

# TCP Reno (Jacobson 1990)



SS: Slow Start  
CA: Congestion Avoidance

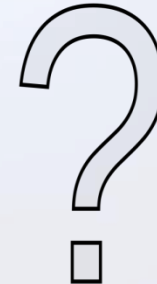
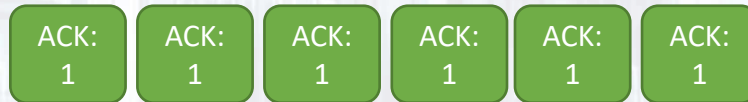
Fast retransmit + **fast recovery** on 3 dupACKs

# Быстрое восстановление

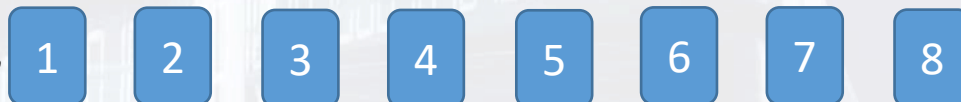


Какая должна быть реакция на большое количество повторных подтверждений ?

receiver



sender



cwnd = 8

# 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: мы можем сдвигать правую границу `sliding window` (в данном случае `congestion window`), увеличивая значение `sliding window`*

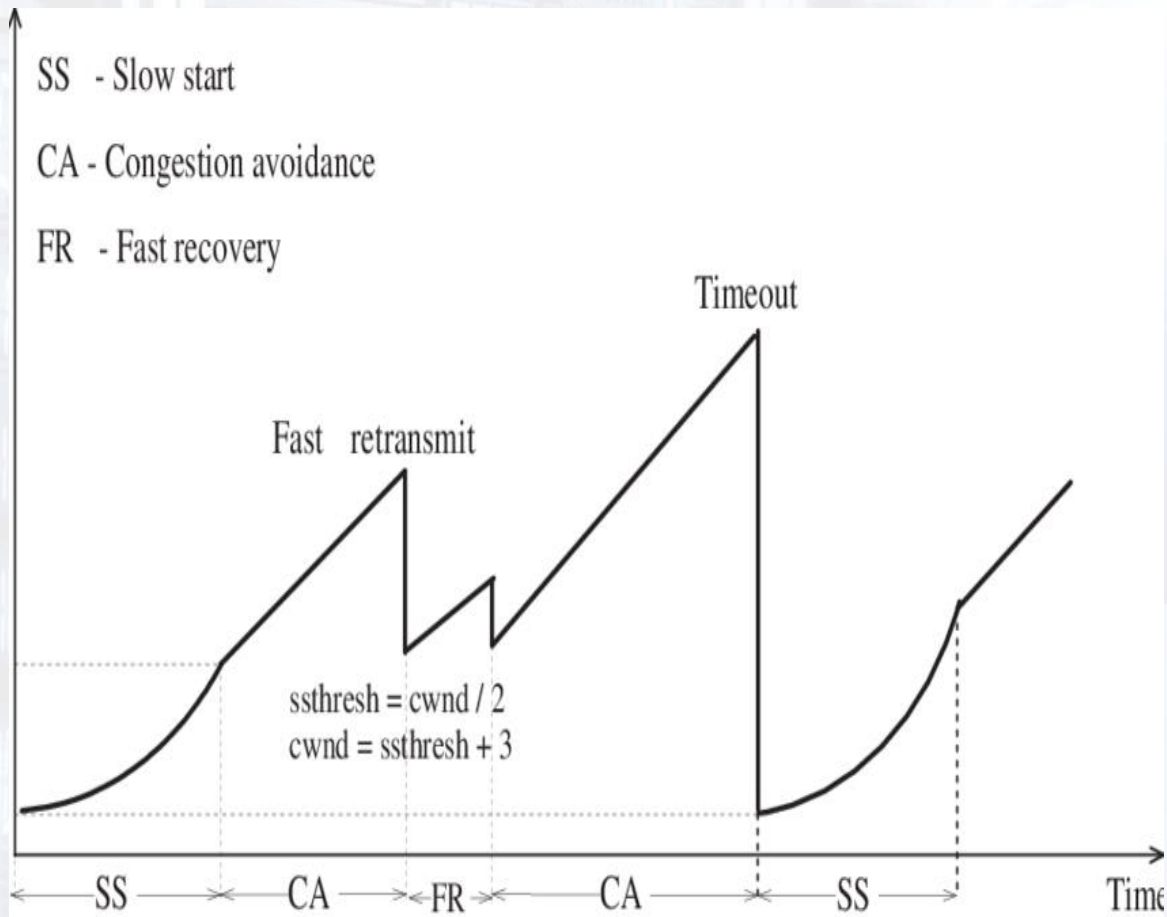
# Быстрое восстановление

---

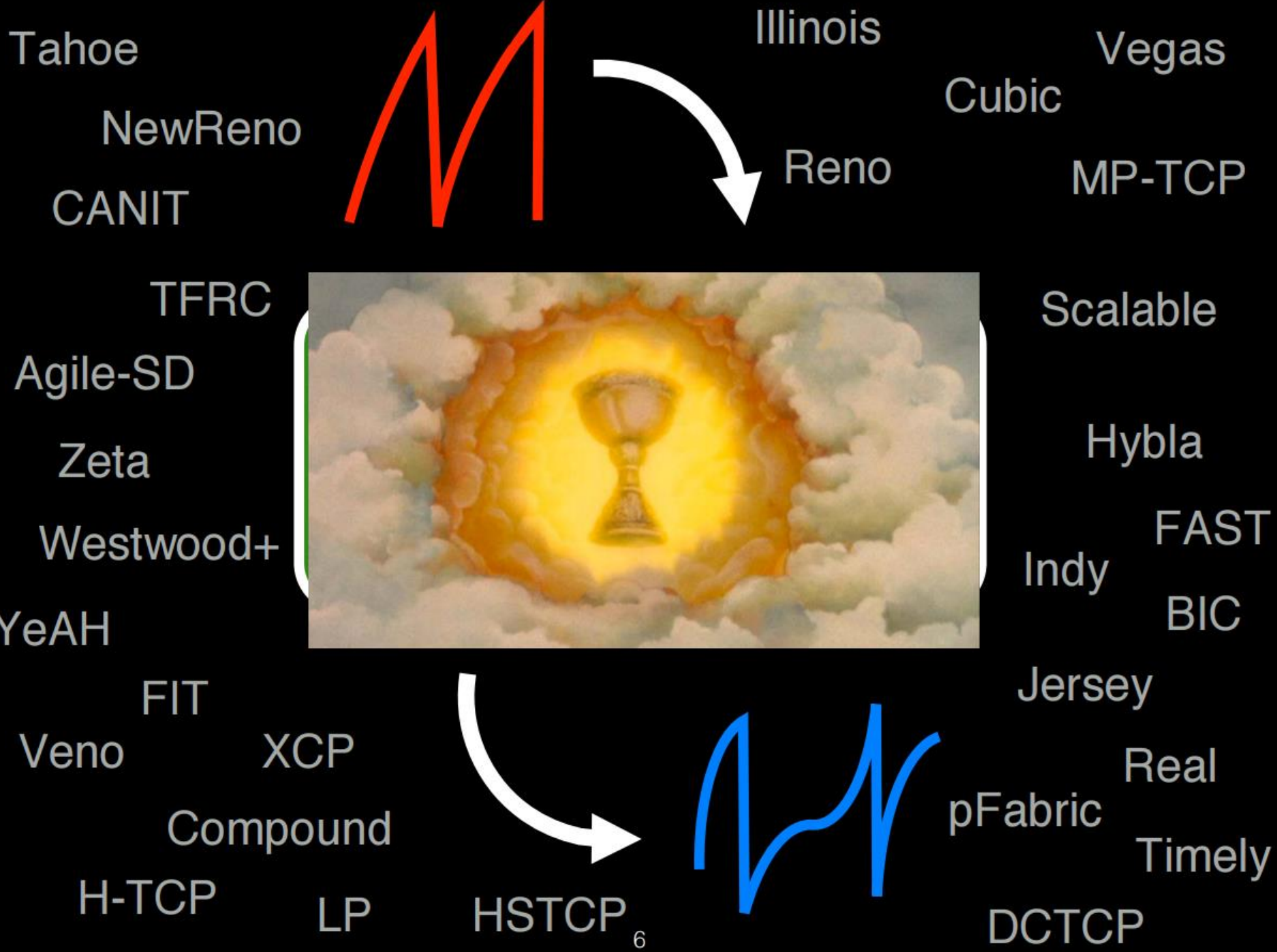


- В начале фазы быстрого восстановления устанавливаем окно в  $swnd/2 + 3$
- Каждое последующее повторное подтверждение означает выход еще одного пакета из сети, поэтому на каждый dup ack увеличиваем  $swnd$  на 1:  $swnd += 1$
- При получении подтверждения на переправленный пакет мы можем двигать левую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

# Быстрое восстановление









# Управления перегрузкой ТСР: принципы работы

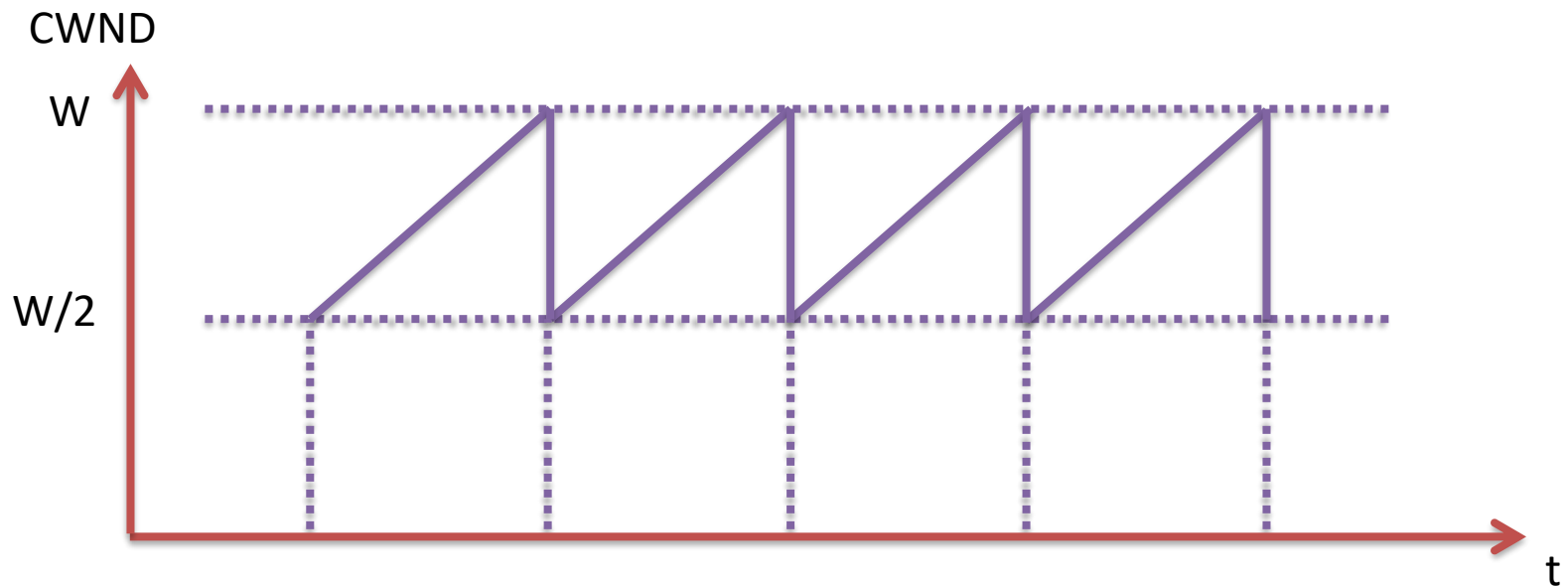
- Взаимодействующие с сетью
  - Сетевые устройства сигнализируют о возникновении перегрузки (ТСР/ECN)
- Без взаимодействия с коммутаторами
  - Перегрузка определяется косвенно (по потере пакета, увеличению задержки и т.д.)
- Реактивные (как правило, loss based)
  - Детектируют возникновение перегрузок по факту
- Проактивные (как правило, delay based)
  - Ограничивают пропускную способность соединения, предчувствуя скорую перегрузку

# Управление перегрузкой: скорость реакции

- При отсутствии дополнительных сервисных пакетов отправитель получает информацию из поступающих к нему АСК-сообщений
- АСК-сообщение, соответствующее отправленному пакету, поступает спустя один Round Trip Time (RTT)
- Хосты способны адаптироваться к состоянию сети не быстрее, чем RTT

# Additive Increase Multiple Decrease (AIMD)

В устойчивом состоянии график зависимости окна перегрузки (CWND) от времени ( $t$ ) для алгоритма TCP Reno имеет вид пилы



# Зависимость размера congestion window от уровня потерь $p$

- Между последовательными потерями:
  - проходит  $W/2$  раундов алгоритма перегрузки, который увеличивает окно от  $W/2$  до  $W$
  - передаётся  $1/p$  пакетов
- $(W/2 + W)/2 * W/2 = 1/p$
- $W = \text{sqrt}(8/3p)$

# Проблемы TCP Reno:

## 1) сети с большим BDP

- ***BDP = Bandwidth Delay Product***
- TCP Reno растёт слишком медленно
  - При использовании канала 10 Gb x 100 ms для увеличения CWND от  $W/2$  до  $W$  потребуется 50000 RTT (более часа)
  - При достаточно частых потерях пакетов алгоритм никогда не сможет добраться до максимальной пропускной способности канала

# Проблемы TCP Reno:

## 2) RTT fairness

- Потоки с меньшим RTT адаптируются к пропускной способности канала быстрее, чем потоки с большим RTT
- Более “быстрые” соединения получают преимущество и используют большее количество сетевых ресурсов

# Проблемы TCP Reno:

## 3) TCP friendliness

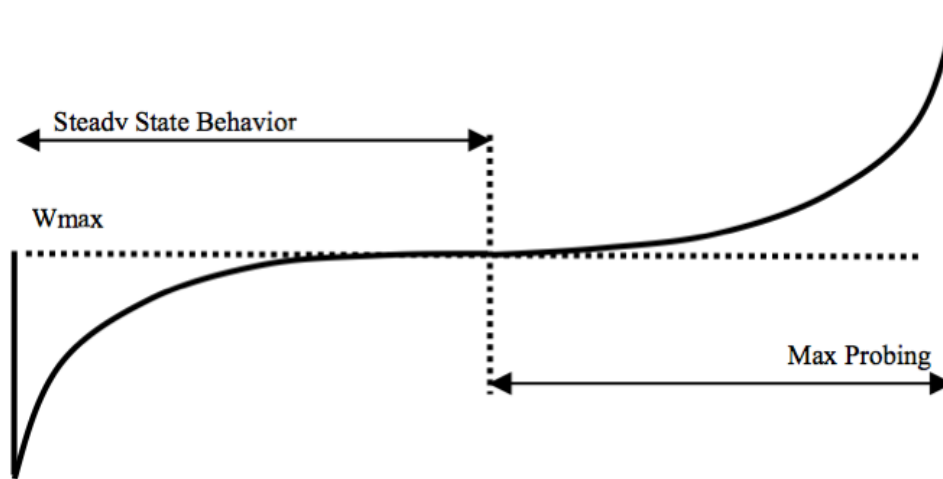
- В теории delay-based алгоритмы управления перегрузкой, как правило, могут работать более эффективно
  - Алгоритмы могут избегать потерь пакетов
  - Значение задержки содержит больше информации о сетевом окружении
- На практике delay-based соединения имеют плохую производительность, потому что передаются вместе с loss-based соединениями
  - Буферы коммутаторов перегружаются вне зависимости от стратегий delay-based соединений

# TCP Cubic

- Идея – предположить, что потери случаются через равные промежутки времени
- Размер окна наращивается таким образом, чтобы он достигал своего максимума в момент следующей потери
  - Размер окна не зависит от RTT!
- Текущий размер окна определяется кубической функцией
  - быстрый рост после перегрузки



# TCP Cubic: CWND

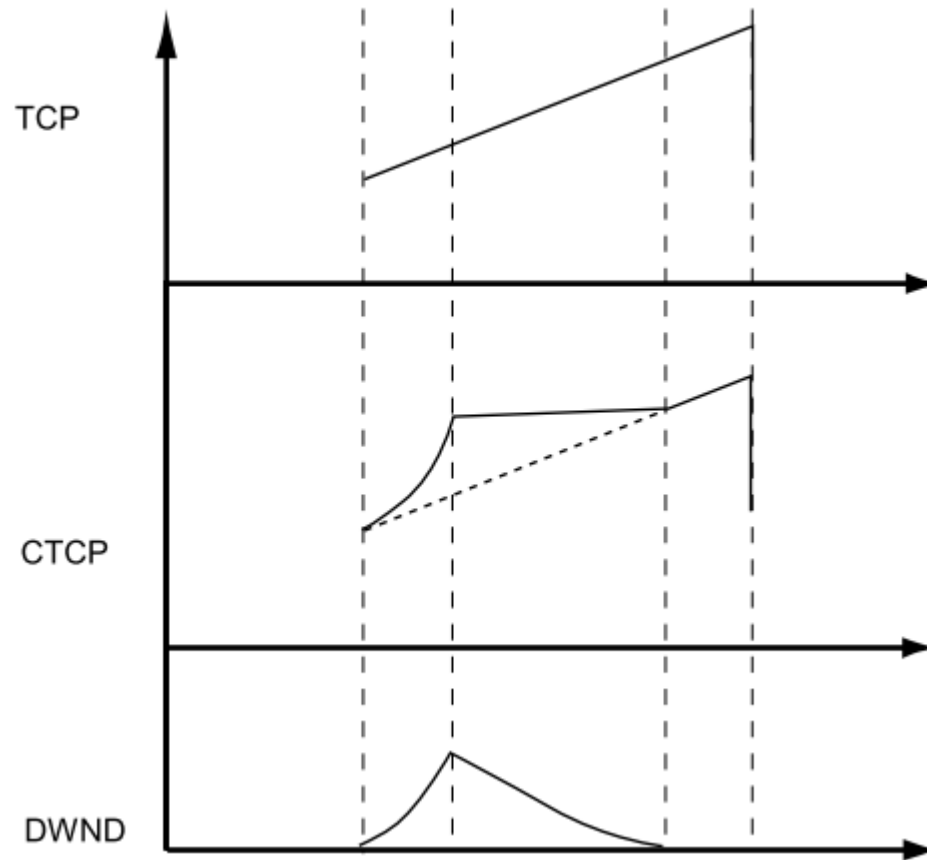


- Изначально CWND растёт быстрее, чем у TCP Reno
- При увеличении вероятности потери рост замедляется
- CWND равен предполагаемому оптимуму максимально долго
- Если потери не происходит – потолок соединения неверен
- Алгоритм начинает быстрый рост, чтобы найти правильную пропускную способность соединения

# TCP Compound

- Идея – эффективная комбинация delay-based и loss-based алгоритмов
  - Скорость delay-based соединений
  - Равная борьба с loss-based соединениями
- Окно перегрузки состоит из loss-based и delay based компонент:
  - Loss-based изменяется по аналогии с TCP Reno
  - Delay-based изменяется экспоненциально в зависимости от текущей задержки: может как расти, так и уменьшаться

# TCP Compound



# Data Center TCP: специализированный протокол управления перегрузкой для ЦОДов

- Горизонтальное масштабирование сервисов: map-reduce
  - Поступающие на сервер запросы отправляются на вспомогательные сервера
  - Каждый из серверов выполняет обработку запроса в своей области и возвращает результат на центральный сервер
  - Центральный сервер комбинирует результаты и формирует окончательный ответ на запрос

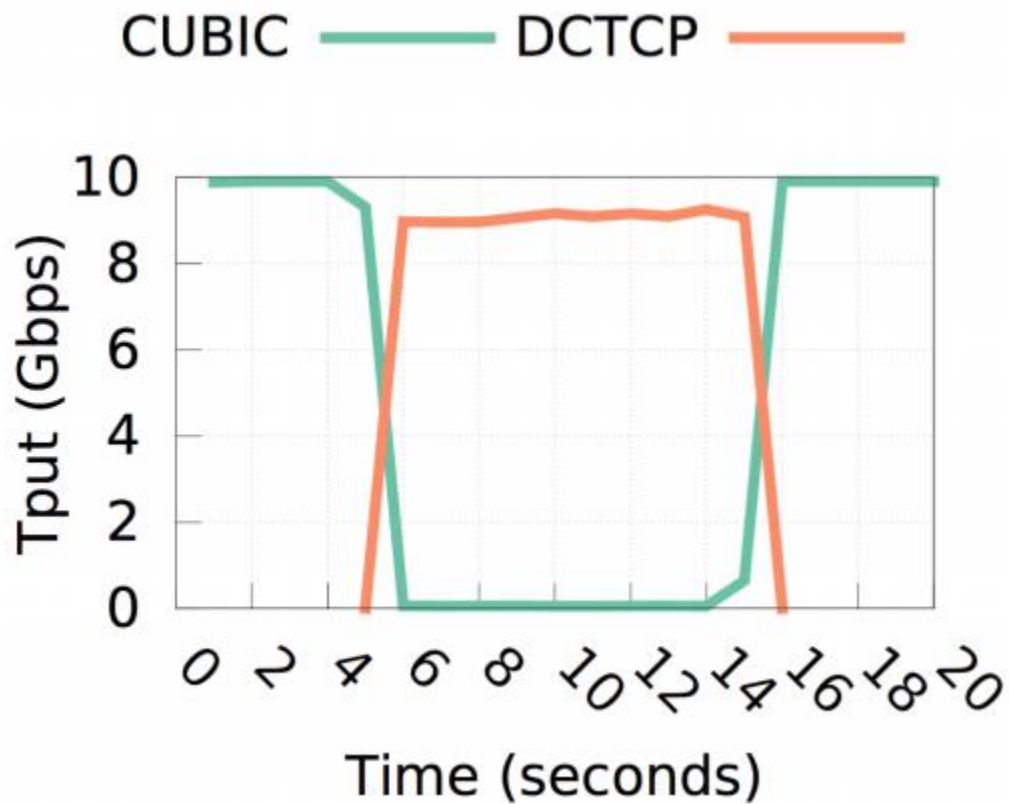
# Data Center TCP: специализированный протокол управления перегрузкой для ЦОДов

- Множество “коротких” соединений конкурирует с долго живущими соединения
- Частый сброс пакетов коротких соединений увеличивает латентность системы
  - В буферах коммутаторов не помещается множество одновременных ответов
  - Буферы коммутаторов заполнены пакетами долго живущих соединений

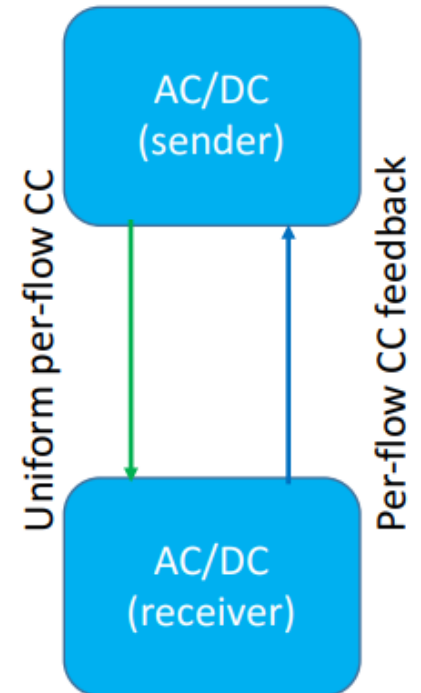
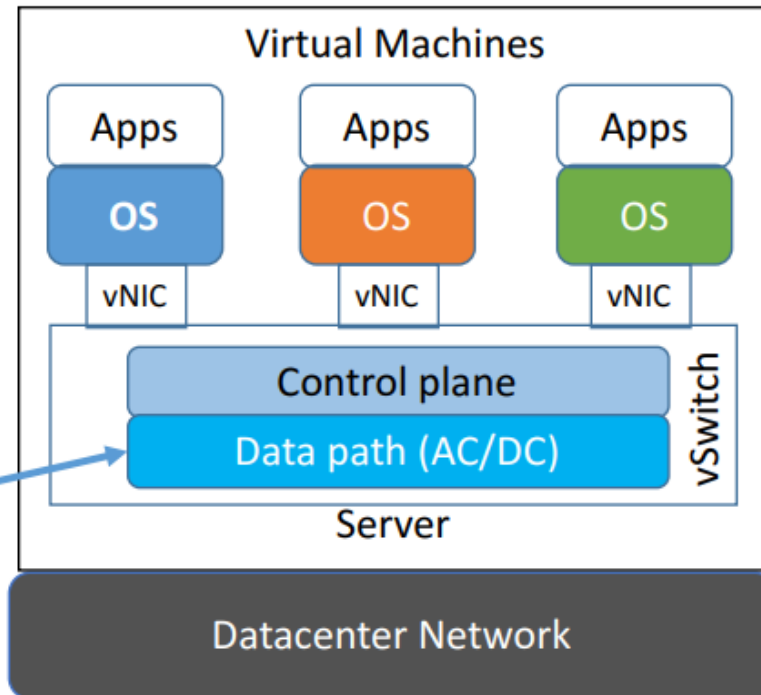
# Data Center TCP: специализированный протокол управления перегрузкой для ЦОДов

- Идея – маркировать пакеты соединений, передающиеся через буферы с высоким уровнем загруженности
- При достаточно частом поступлении маркированных пакетов отправитель уменьшает размер CWND
- При редком поступлении маркированных пакетов размер CWND увеличивается

# Сосуществование разных ТСР в ЦОД



# AC/DC: High Level View



**Case study: DCTCP  
CC in the vSwitch**

\*\* AC/DC TCP: Virtual Congestion Control Enforcement for Datacenter Networks

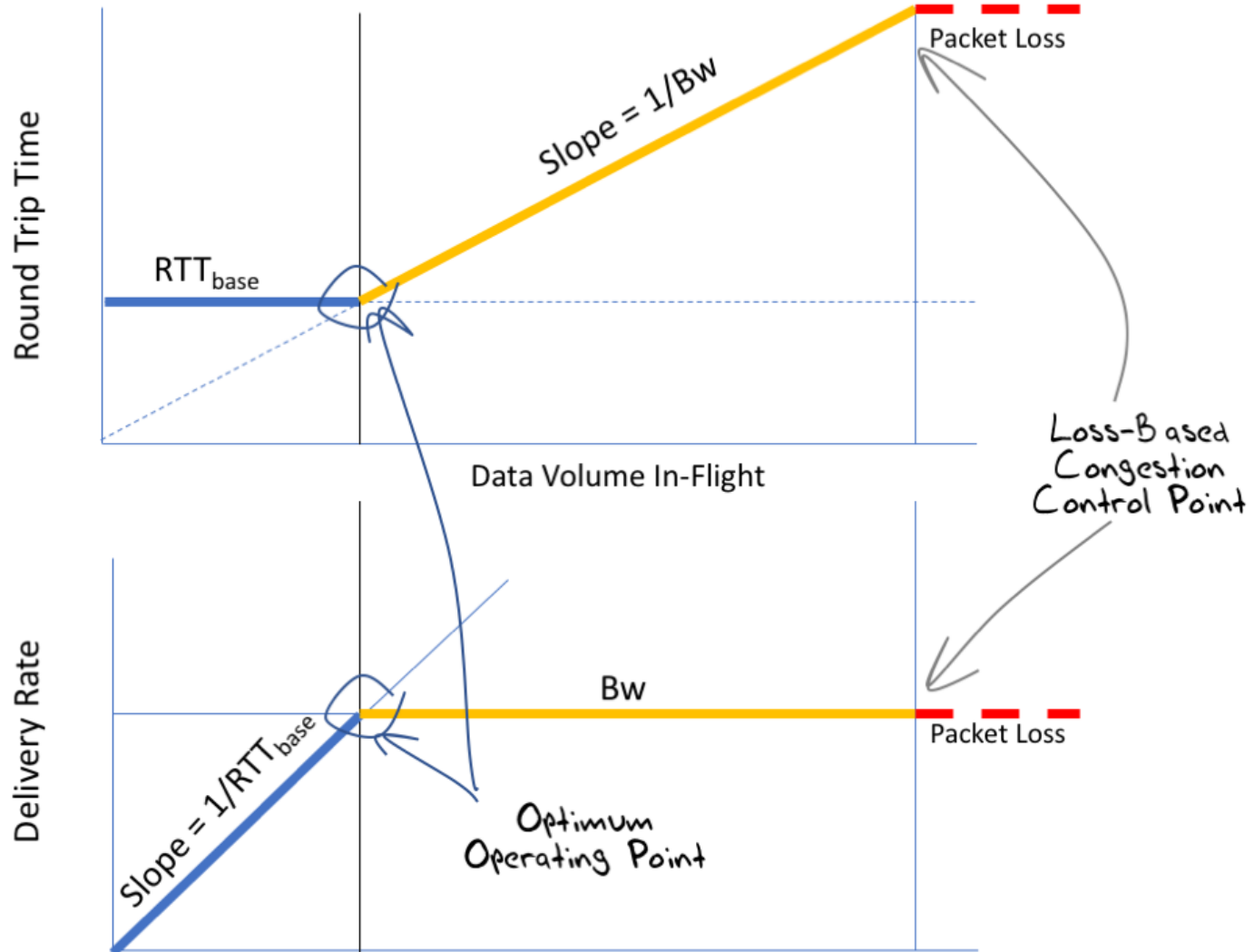
Keqiang He, Eric Rozner, Kanak Agarwal, Yu (Jason) Gu, Wes Felter, John Carter, Aditya Akella



# Классификация современных алгоритмов управления перегрузкой

1. Loss-based алгоритмы (Reno/NewReno, Cubic)
2. Delay-based алгоритмы (Vegas)
3. Гибридные алгоритмы (Compound TCP)
4. Capacity-based алгоритмы (BBR)
5. Алгоритмы, основанные на машинном обучении (QTCP)

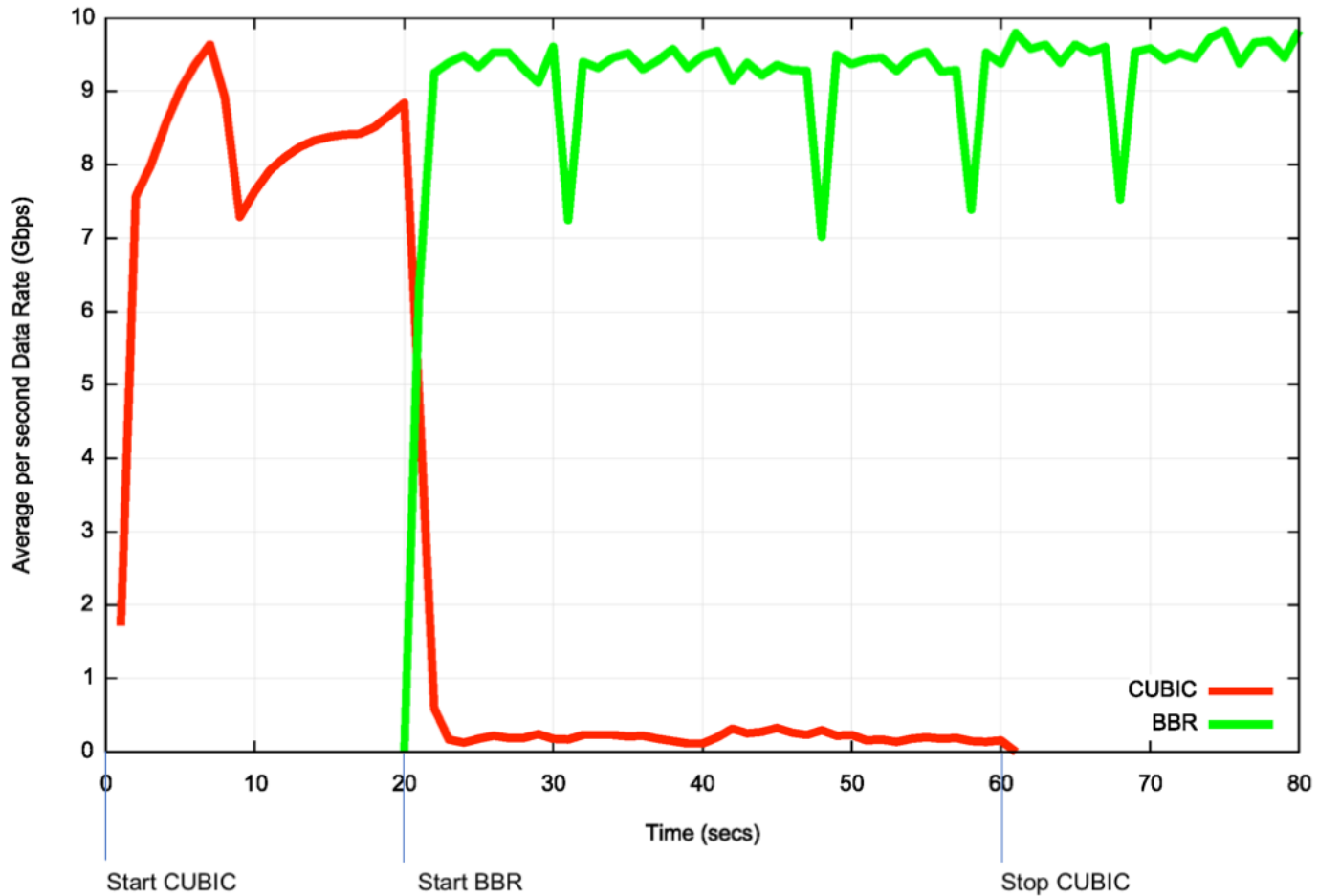
# BBR



# BBR

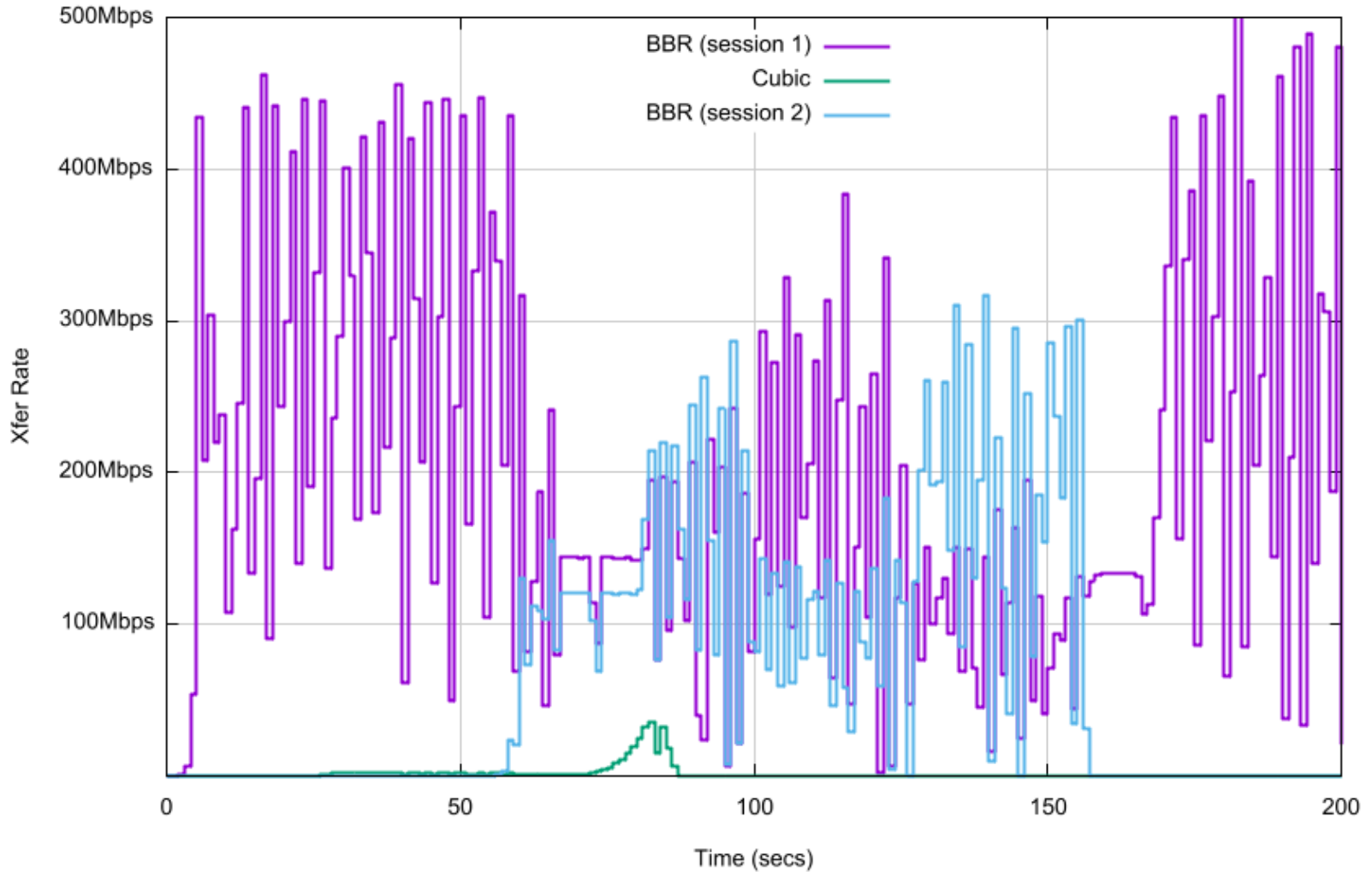
- Увеличение RTT вызвано
  - Изменением маршрута
  - Изменением пропускной способности узкого места
  - Увеличением задержки очередизации из-за трафика другого соединения
- Измеряем пропускную способность узкого места и минимальный RTT
- Временно увеличиваем скорость отправки:
  - Если измеряемая пропускная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

# BBR vs Cubic



# BBR vs Cubic vs BBR

AU - DE (Internet path, 303ms RTT)



# Проблемы BBR

- Aggressive startup operation
- Aggressive bandwidth probing
- RTT unfairness
- Unfairness with loss-based congestion control

# BBRv2

	<b>BBRv1</b>	<b>BBRv2</b>
<b>Model Parameters</b>	BtlBw, RTprop	BtlBw RTprop loss_rate Aggregation
<b>Packet loss as congestion signal</b>	N/A	loss_rate exceeds loss_threshold
<b>ECN</b>	N/A	DCTCP/L4S style ECN
<b>Exit startup</b>	throughput plateaus	throughput plateaus or loss/ECN rate exceeds threshold
<b>cwnd in ProbeRTT</b>	4 packets	BDP/2

Y. -J. Song, G. -H. Kim, I. Mahmud, W. -K. Seo and Y. -Z. Cho, "Understanding of BBRv2: Evaluation and Comparison With BBRv1 Congestion Control Algorithm," in *IEEE Access*, vol. 9, pp. 37131-37145, 2021





# Программа курса

## Подходы:

- 1. Управление перегрузкой**
  - Современные протоколы управления перегрузкой TCP
- 2. Демультимплексирование/мультиплексирование**
  - Многопоточные транспортные протоколы
  - Маршрутизация на уровне интернет провайдеров
  - Network Coding
- 3. Сегментация**
  - TCP Proxy
- 4. Балансировка**
  - Балансировка нагрузки и управление трафиком

## Модели:

- Сетевое исчисление: математический подход к качеству сервиса
- NS3: моделирование поведения сети с высокой точностью

## Примеры:

- HTTP3/QUIC
- Управление сетевыми ресурсами в Центрах Обработки Данных
- Обеспечение качества сервиса в сетях доставки контента
- Пропускная способность по требованию