SACK TCP

Selective Acknowledgment (SACK)
Transmission Control Protocol (TCP)

Mathis, M., Mahdavi, J., Floyd, S. and A.Romanow,
"TCP Selective Acknowledgement Options",
RFC 2018, October 1996.

Что плохо в версии TCP до `October 1996`?

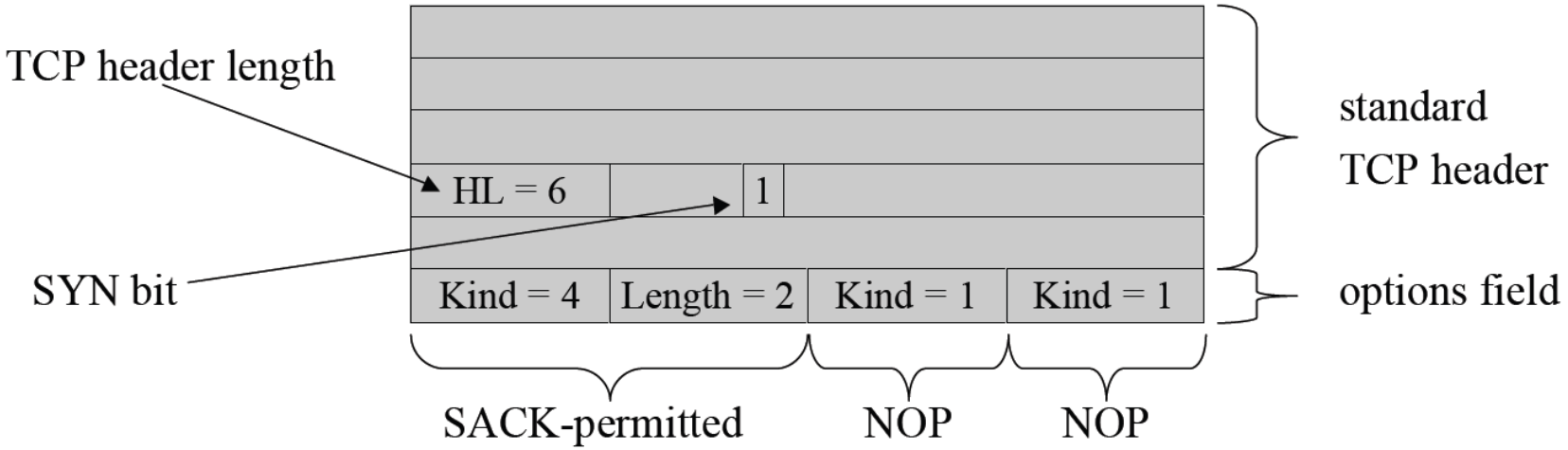
- TCP использует кумулятивную схему подтверждения, в которой приемник идентифицирует последний успешно полученный байт данных
- Полученные сегменты, которые не являются левой границей окна, не признаются (отбрасываются получателем)
- Такая логика исправления ошибок передачи заставляет отправителя
  - или подождать время ретрансляции (RTT), чтобы узнать был ли потерян сегмент,
  - или подождать срабатывание таймера RTO, чтобы сделать повторную передачу сегментов, которые были правильно доставлены но отброшены получателем
- Результат - значительное сниженной общей пропускной способности

Селективное подтверждение TCP

- Выборочное (селективное) подтверждение (SACK - Selective Acknowledgment)
  - Позволяет получателю информировать отправителя обо всех успешно полученных сегментах
  - Позволяет отправителю повторно передавать только те сегменты, которые были потеряны
- SACK реализован с помощью двух различных TCP опций:
  - The SACK-Permitted Option
  - The SACK Option

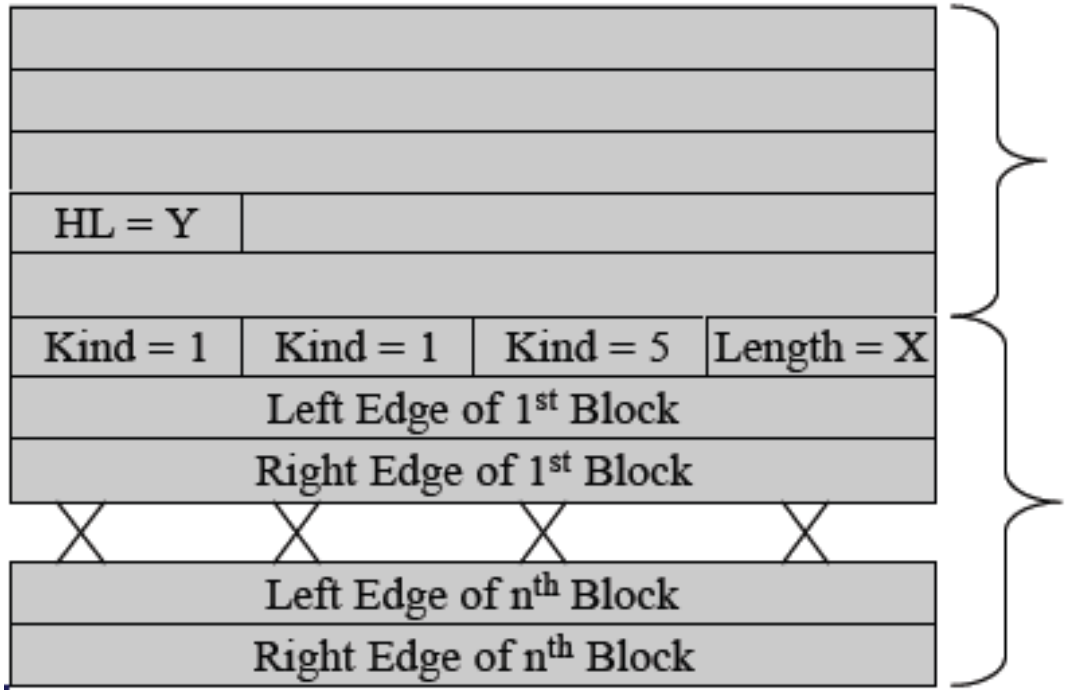
SACK-Permitted Option

- Первая TCP опция
  - это активирующая опция “SACK-permitted”, формируемая отправителем в фазе установления соединения в сегменте с флагом SYN
  - эта опция означает, что отправитель может обрабатывать данные SACK, и получатель должен отправить их, если это возможно. (Обе стороны могут включить SACK, но каждое направление TCP соединения обрабатывается независимо)



SACK option

- Вторая TCP опция
  - “SACK option” содержит левую и правую границы успешно полученных байт
  - Только получив опцию “SACK-Permitted”, можно отправить “SACK option”



Формула для длины поля опции SACK (основанную на n, количество блоков в опции)?

(2 + 8 * n) байт

Какое максимальное количество блоков SACK возможно? Почему?

Максимальный размер поля опции составляет 40 байт, что дает максимум SACK блоков - 4 (за исключением других опций TCP).

Опция SACK

- Каждый блок в SACK представляет успешно полученные байты, которые являются непрерывными и изолированными
  - Для SACK=6000-6500 байты непосредственно слева (5999) и последний справа (6500) еще не были получены

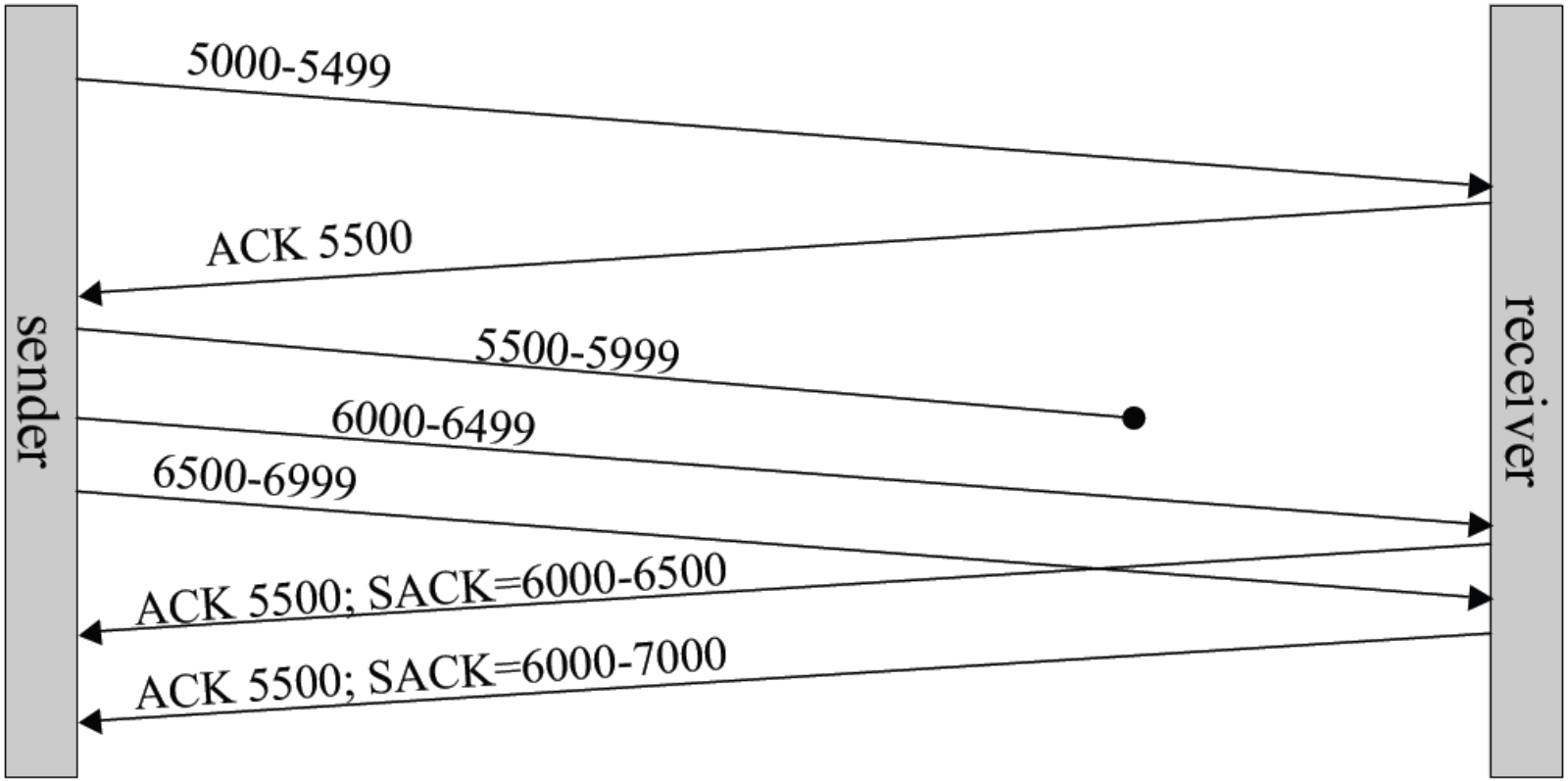
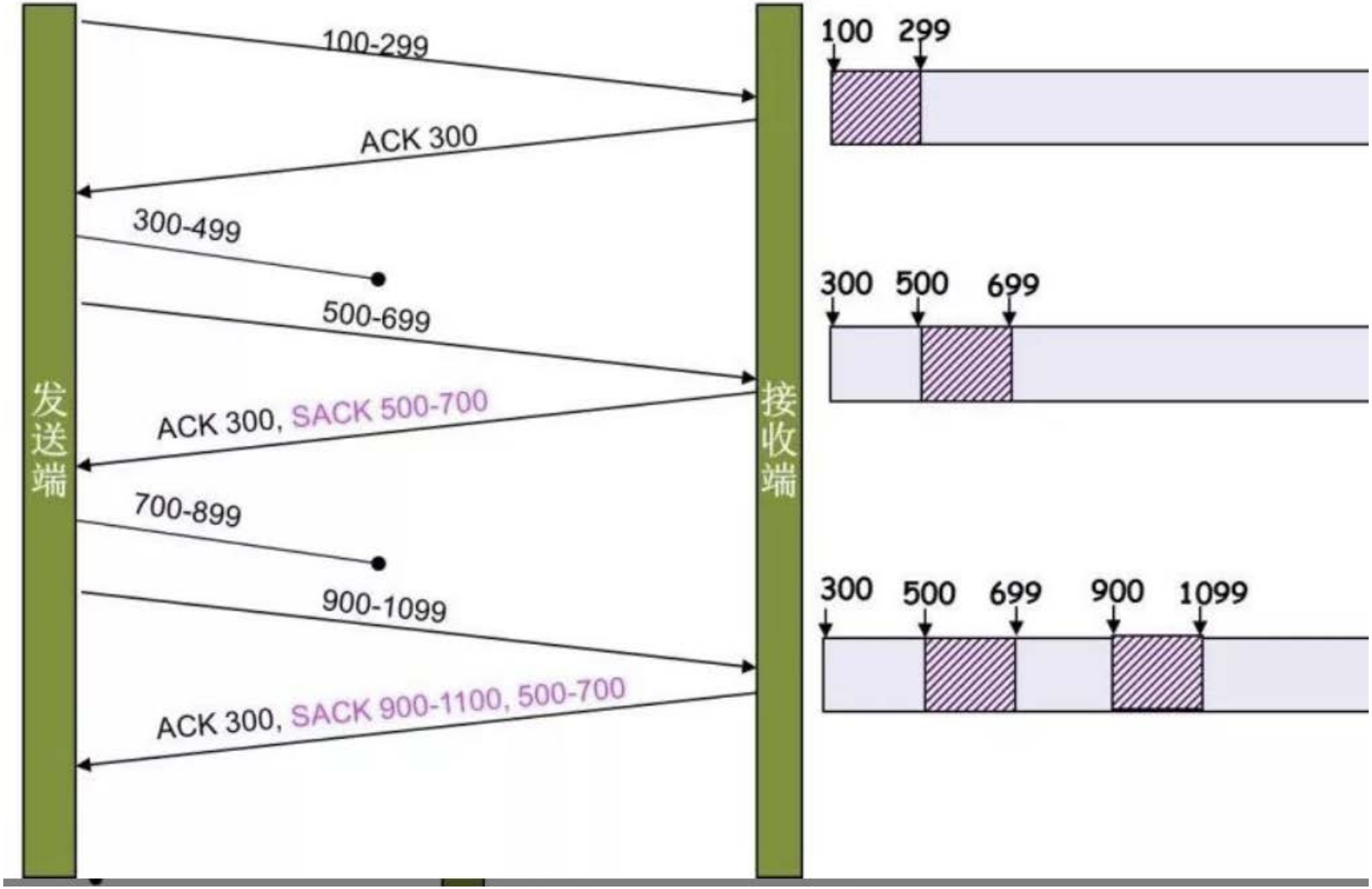


Иллюстрация SACK

Иллюстрация «дыр» в буфере получателя (300-499, 700-899)

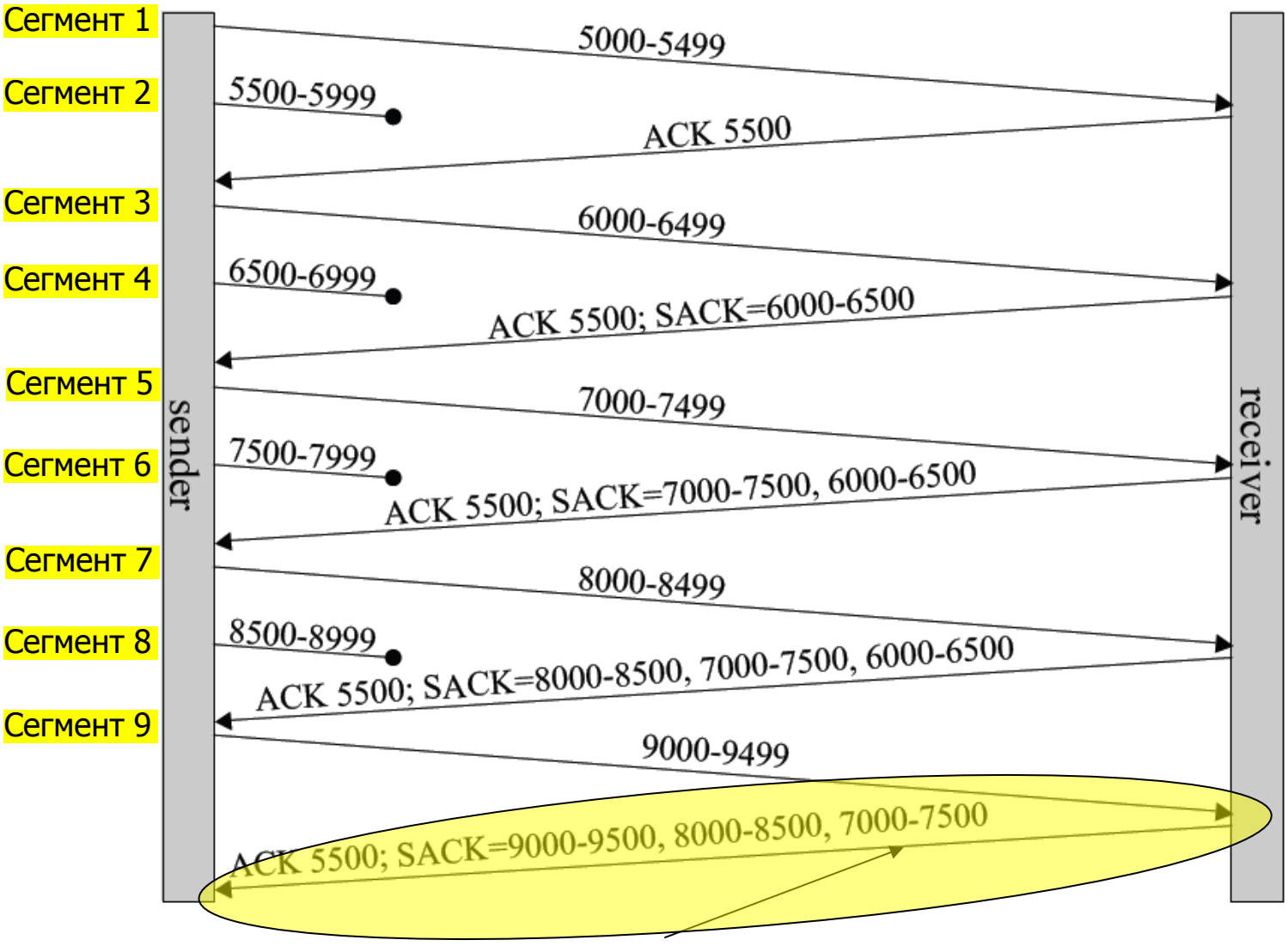


SACK правила получателя

- SACK не может быть отправлен, если опция SACK-permitted не была получена (в SYN)
- Если получатель решил отправить SACK, он должен отправлять их всякий раз, когда он имеет данные для SACK во время ACK
- Получатель должен
  - отправить ACK для каждого полученного действительного сегмента, содержащего новые данные (стандартное поведение TCP)
  - каждый из этих ACK должен содержать SACK, при условии, что есть данные для SACK
  - первый блок SACK должен содержать самый последний правильно полученный сегмент, который должен быть SACK
  - второй блок SACK должен содержать второй из последних полученных сегментов, который должен быть SACK и т.д.
- Ремарка. Некоторые данные в буферах приемника должны быть SACK, но не будут таковыми, если сегментов для SACK больше чем свободного места в заголовке TCP.

SACK TCP пример

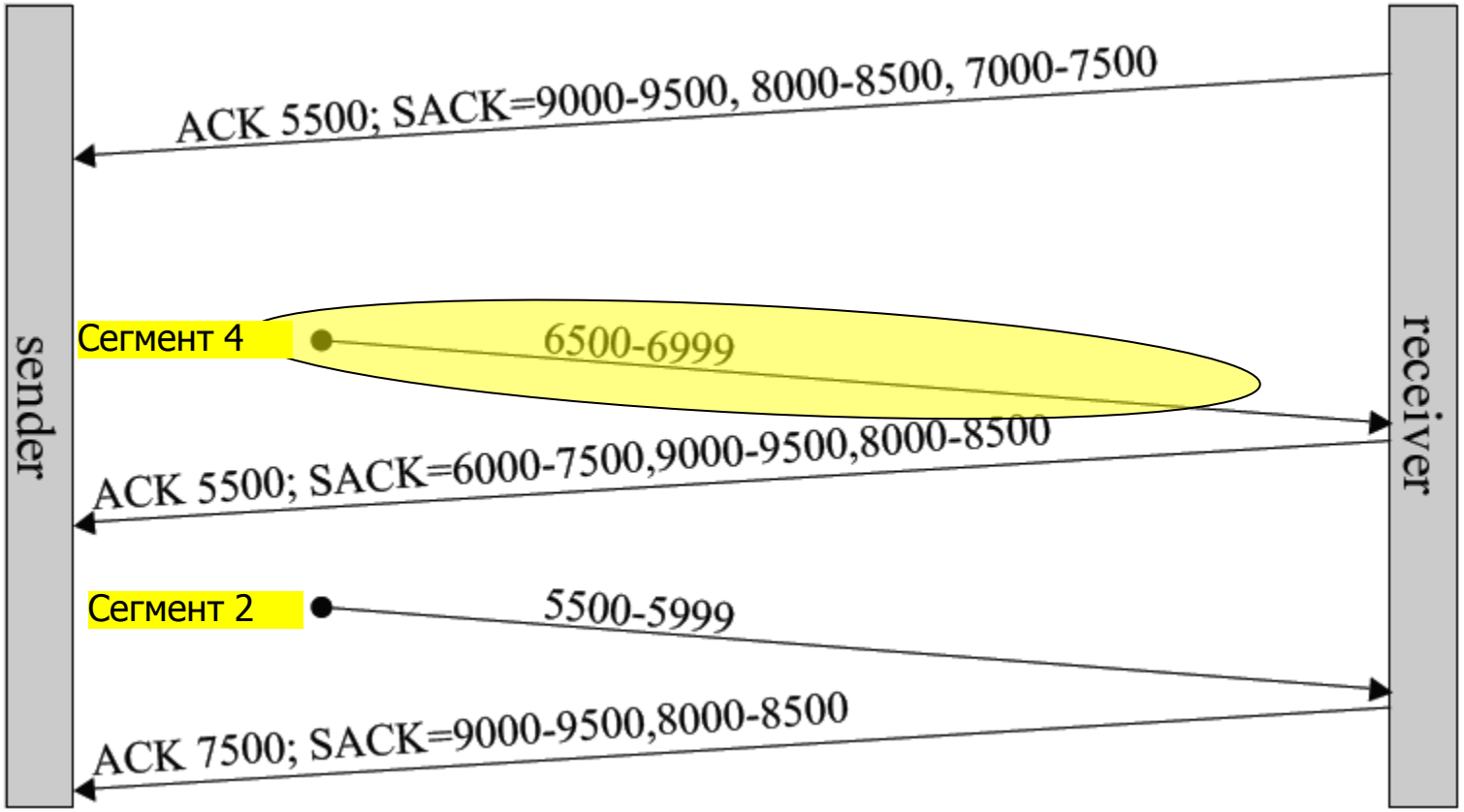
При условии максимума из 3-х блоков SACK



SACK=6000-6500 не вошел в поле опций TCP

SACK TCP пример

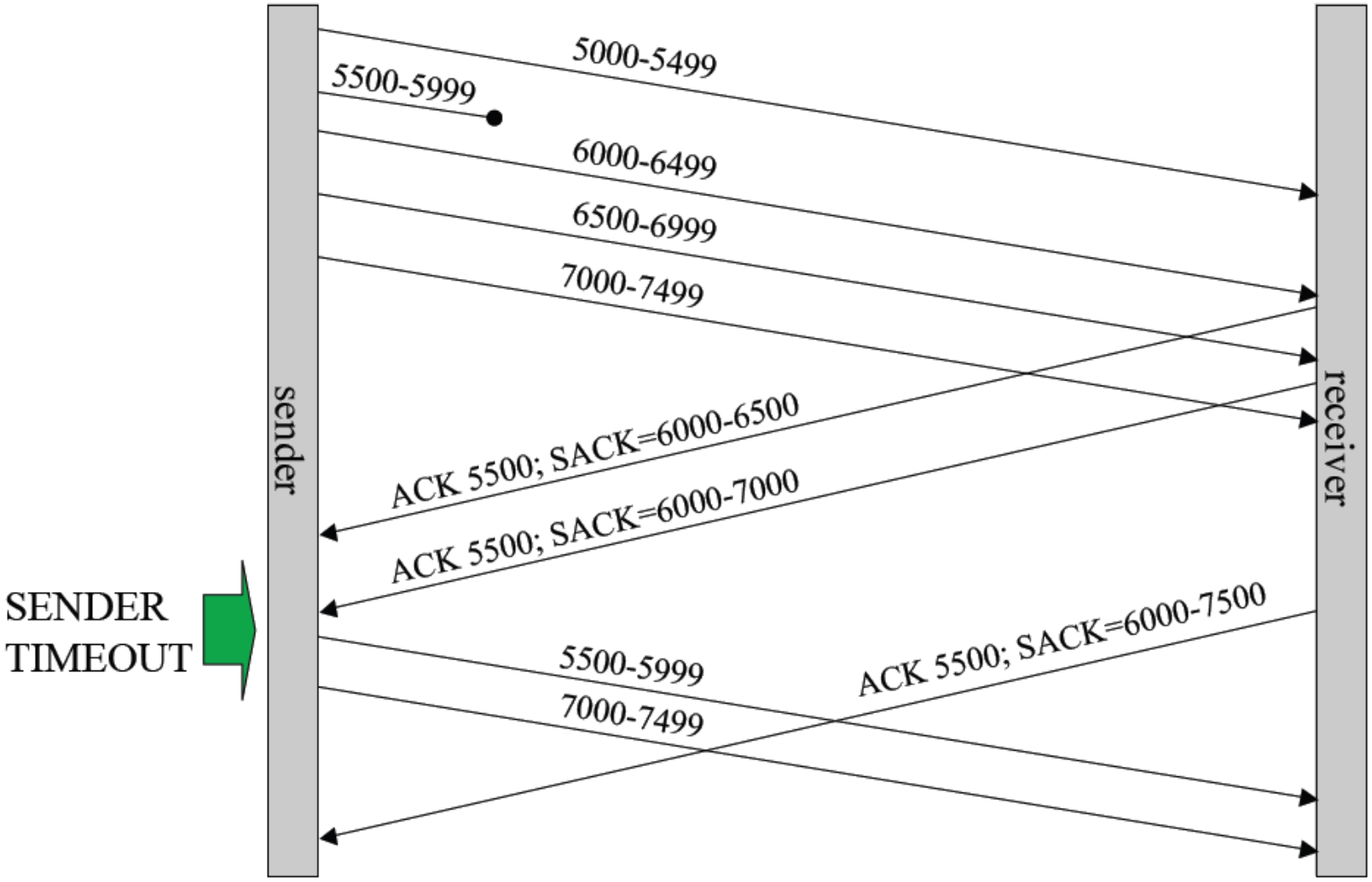
- Пусть придет 4-й сегмент (6500-6999). Получатель подтвердит этот прием (SACK=6000-7500)
- После подтверждения приема далее пусть придет 2-й сегмент (5500-5999)



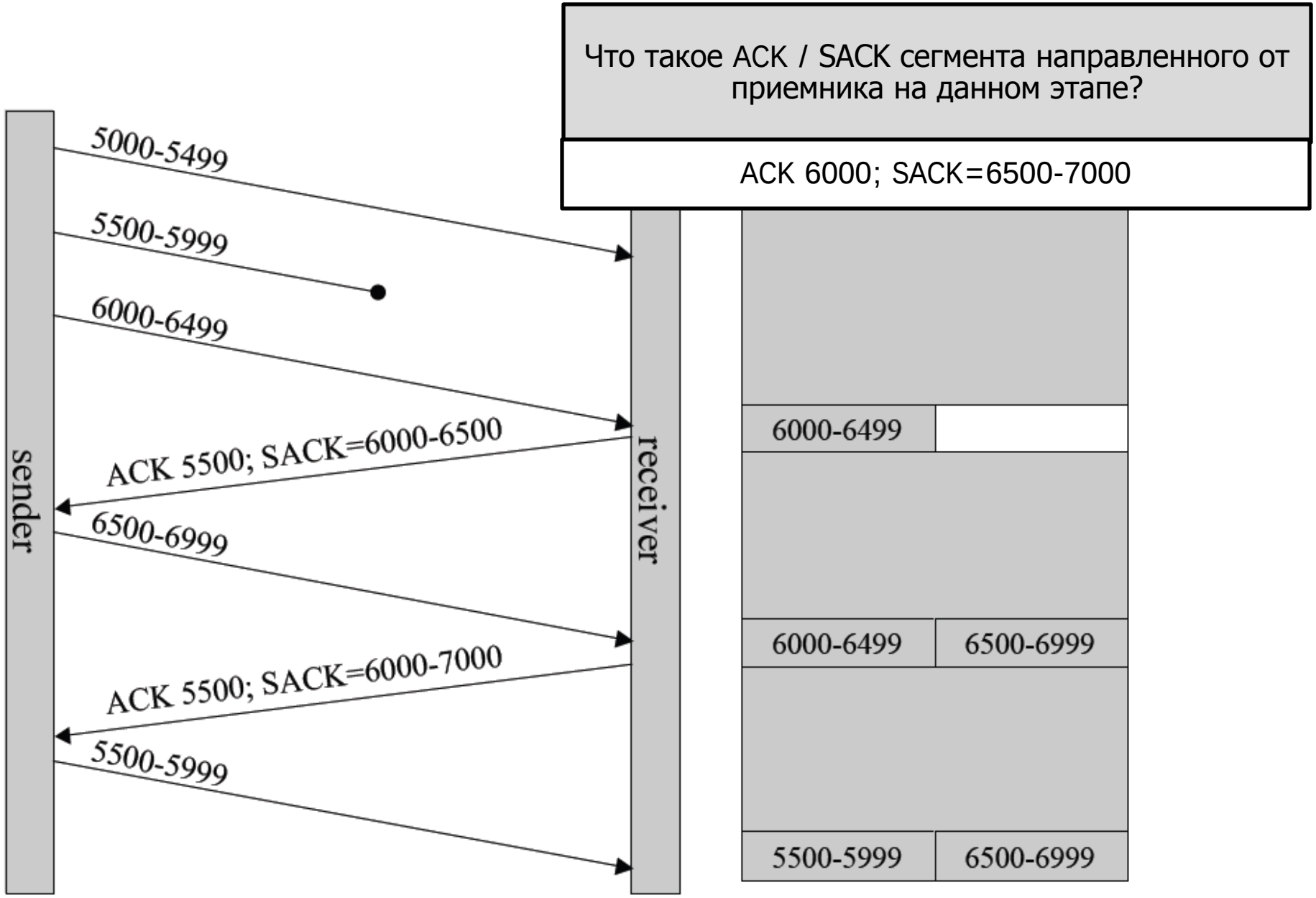
SACK правила отправителя

- Отправитель должен иметь буфер неподтвержденных данных. При получении SACK опции, он должен включить бит SACK-флага для всех сегментов в буфере передачи, которые полностью содержатся в одном из блоков SACK.
- После установки этого бита SACK-флага отправитель должен пропустить этот сегмент при любой последующей повторной передаче.

SACK TCP пример передачи



Получатель имеет двух сегментный буфер (проблема?)



Отказ в SACK TCP (правила получателя)

- Проблема «Reneging получателя»
  - Получатель может подтвердить (SACK) некоторые данные, а затем отбросить эти данные. Это называется отказом (Reneging) получателя. Это не рекомендуется, но разрешено, если получателю не хватает места в буфере.
  - Следовательно, отправитель не может полностью полагаться на SACK, а должен полагаться на поле ACK (Acknowledgment Number) в TCP заголовке и корректно поддерживать таймер повторной передачи (RTO)
- Если «Reneging» произойдет, правила получателя такие:
  - первый блок SACK должен отражать новейший сегмент, т.е. содержать левый и правый края новейшего сегмента, даже если этот сегмент будет отброшен.
  - за исключением новейшего сегмента, все SACK блоки не должны сообщать о каких-либо старых данных которые больше не хранятся в приемнике

Отказ в SACK TCP (правила отправителя)

- Поскольку получатель данных может впоследствии отбросить данные, указанные в опции SACK
  - отправитель НЕ ДОЛЖЕН отбрасывать данные до тех пор, пока они не будут подтверждены полем Acknowledgment Number в заголовке TCP
  - отправитель должен повторно передать сегменты начиная с левой границы окна после тайм-аута повторной передачи, даже если для этого сегмента установлен бит SACK.
- Другими словами, сегмент не может быть удален из буфера передатчика до тех пор, пока левая граница окна не будет передвинута через него, при получение ACK

SACK TCP выводы

- SACK TCP следует стандартному управлению перегрузкой TCP; является дружественным и не должен вредить сети.
- SACK TCP имеет преимущество по сравнению с другими стратегиями управления перегрузкой (Reno, Tahoe, Vegas, и NewReno), поскольку в нем добавлена дополнительная информация благодаря данным SACK.
- SACK информация позволяет отправителю лучше решить, что ему нужно передать повторно, а что нет. Это может помочь только отправителю и не должно негативно влиять на другие TCP
- Функция SACK помогает отправителю определить «пробелы» в буфере получателя. Когда получатель не получает пакеты по порядку, в буфере есть некоторые дыры (отсутствующие байты). SACK помогает отправителю заранее узнать об этих дырах в буфере получателя. Когда отправитель узнает о дырах или потерянных пакетах, он одновременно ретранслирует их. В отличие от Fast Recovery, SACK не слишком поздно информирует отправителя о потерянных пакетах.

SACK TCP выводы

- Несмотря на то, что SACK TCP по-прежнему может без необходимости повторно передавать сегменты, количество таких повторных передач в моделировании оказалось довольно низким по сравнению с Reno и Tahoe TCP.
- В любом случае, *количество ненужных повторных передач* должно быть строго меньше, чем Reno / Tahoe TCP. Поскольку у отправителя есть дополнительная информация, на основе которой можно разработать схему повторной передачи, худшая производительность невозможна (за исключением ошибочной реализации).

Усовершенствования SACK

- Дублированное выборочное подтверждение
  - **D-SACK TCP** (RFC2883, июль 2000) **Duplicate-SACK**
  - Изменено содержимое опции SACK при отправке подтверждения
- Прямое подтверждение (FACK)
  - **FACK TCP** (RFC …. September 2009) **Forward Acknowledgment**
  - Использует SACK для точного измерения общего количества байтов в сети
- **Алгоритм обнаружения потерь RACK-TLP для TCP**
  - **RACK-TLP для TCP** (RFC 8985, 2021 год) **Recent Acknowledgment**
  - использует метки времени передачи для каждого сегмента и выборочные подтверждения (SACK)

D-SACK TCP (Duplicate-SACK)

Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", **RFC 2883, DOI 10.17487/RFC2883, July 2000,** https://www.rfc-editor.org/info/rfc2883

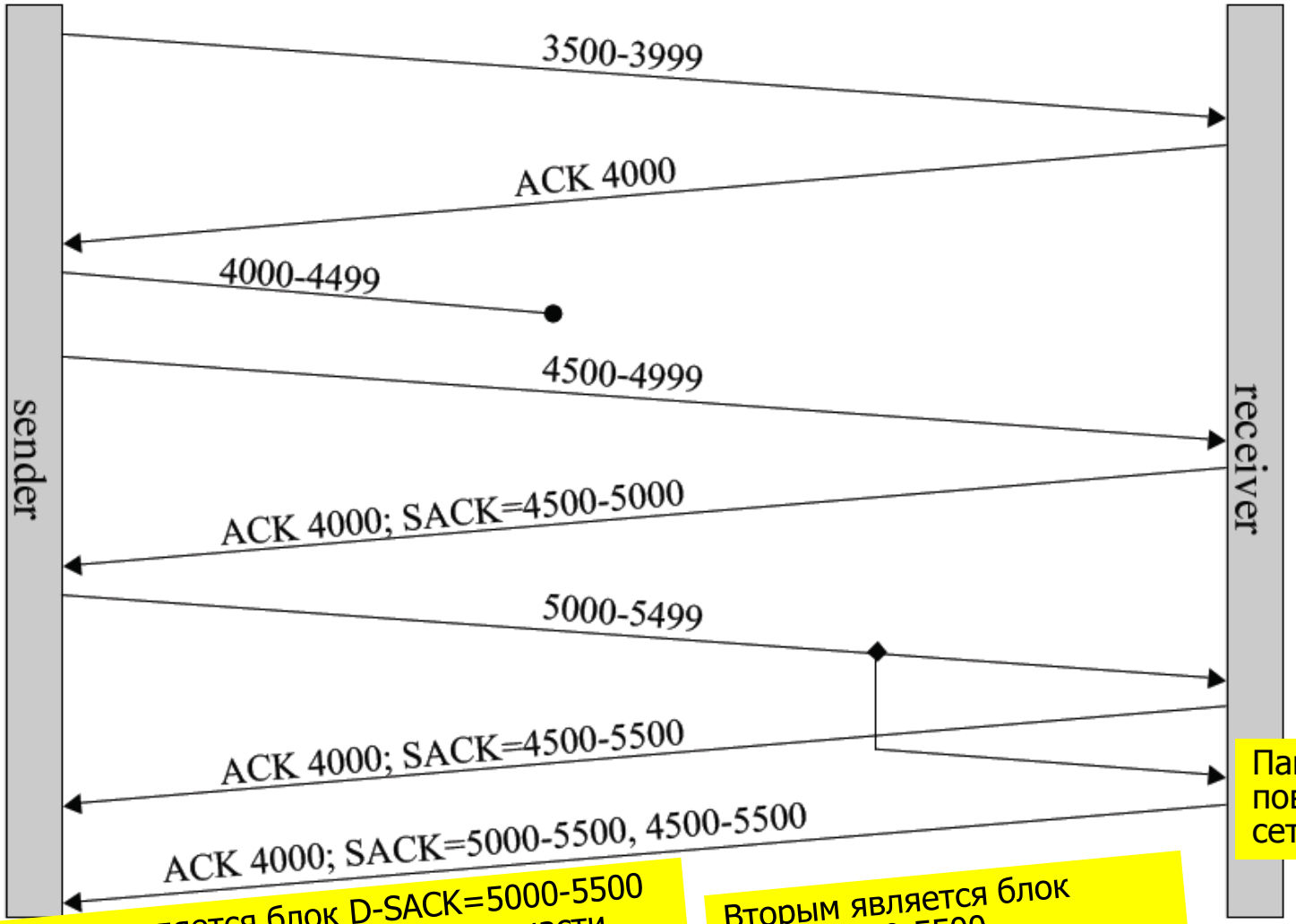
Суть проблемы: обработка псевдо-RTO

- TCP задержка передачи, даже если нет потери пакетов, может привести к срабатыванию тайм-аута повторной передачи (RTO). При срабатывании RTO-таймера TCP настроит ssthresh и установит cwnd на IW, таким образом войдя в состояние медленного старта. Если фактической потери пакетов нет, ACK, который поступает после RTO, приведет к быстрому увеличению cwnd, но все равно будут ненужные повторные передачи до того, как значения cwnd и ssthresh снова стабилизируются, что приводит к трате ресурсов передачи.
- Предлагаются некоторые методы решения этой проблемы: DSACK, Eifel, F-RTO. Любой из этих методов обнаружения может «восстановить» операции TCP с переменными управления перегрузкой, если они сочетаются с соответствующими алгоритмами ответа. Здесь в основном вводится алгоритм ответа Эйфеля.
  - Алгоритм ответа Eifel используется для обработки таймера повторной передачи и операций управления перегрузкой после истечения таймера повторной передачи. Когда повторная передача по таймауту происходит впервые, алгоритм Эйфеля начинает выполняться. Если рассматривается ложная повторная передача, изменение значения ssthresh будет отменено. Если вам нужно изменить значение ssthresh из-за тайм-аута повторной передачи, вам необходимо записать специальную переменную: pipe_prev перед изменением.
  - pipe_prev = min (вне значения данных, ssthresh)

На шаг впереди: D-SACK TCP

- Duplicate-SACK, или D-SACK это расширение SACK TCP, который использует первый блок SACK для того, что бы сообщить какие дубликаты сегментов были получены.
- D-SACK блок используется только для того что бы представить дублированную непрерывную последовательность данных, полученных от приемника в последнем сегменте.
- О каждом дубликате сообщается не более одного раза.
- Это позволяет отправителю TCP определить, когда в ретрансляции нет необходимости. Это может быть не обязательно, если случилось преждевременное истекании таймера или в связи с ложным быстрым повтором (получено три ACK дубликата из-за переупорядочения сегментов в сети).

D-SACK пример (пакет повторен сетью)

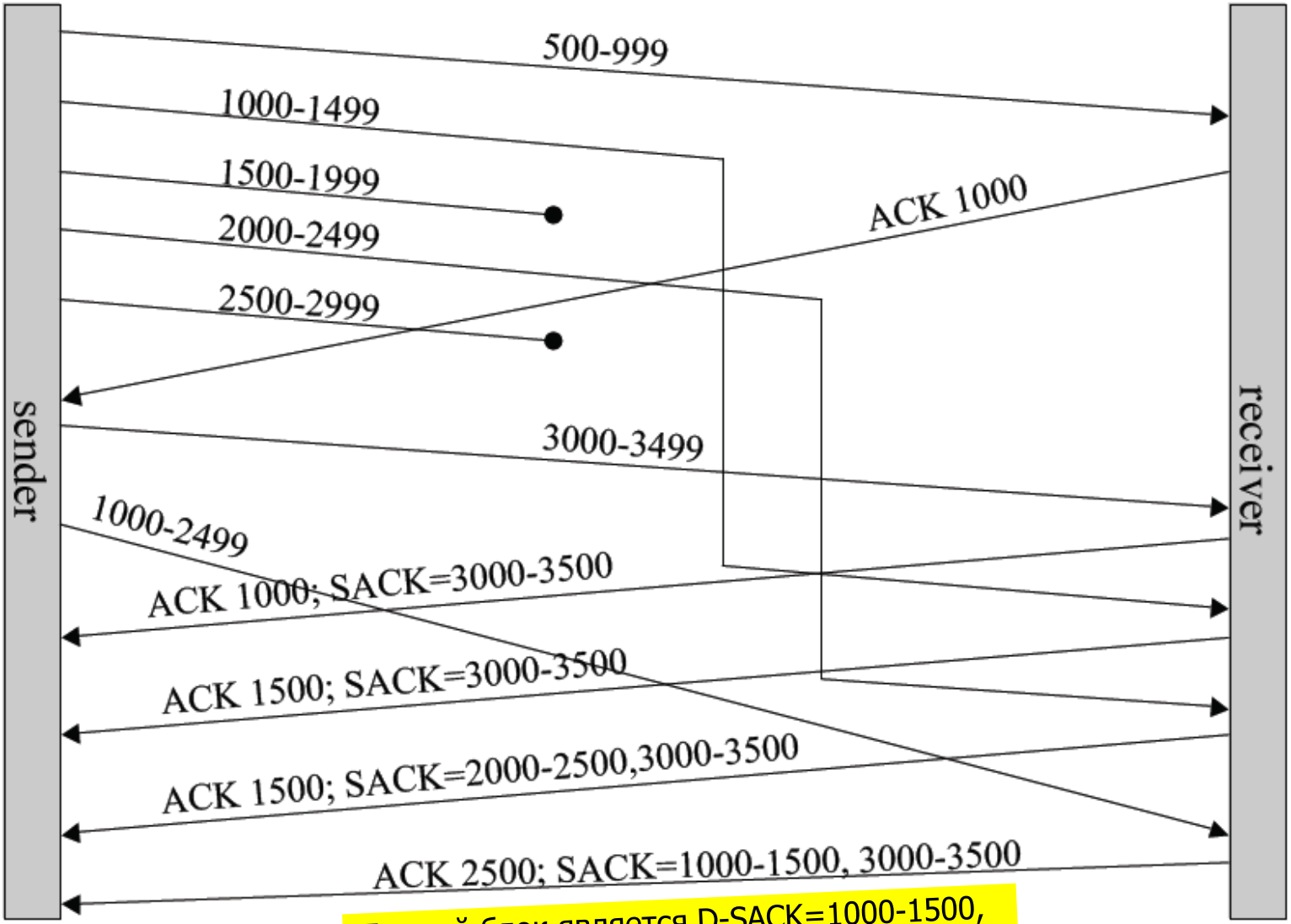


Пакет повторен сетью

Первым является блок D-SACK=5000-5500 сообщает о получении дупликата части ранее подтвержденного блока SACK=4500-5500

Вторым является блок SACK=4500-5500 дупликат которого пришел в D-SACK

D-SACK пример (отправитель изменил размер сегмента)



Первый блок является D-SACK=1000-1500, поскольку сообщает о получении дупликата части диапазона до ACK=2500

D-SACK TCP правила

- Если блок D-SACK сообщает о повторяющейся последовательности из (возможно, большего) блока данных в буфере получателя выше кумулятивного подтверждения, второй блок SACK (первый блок без D-SACK) должен определить этот блок.
- Поскольку только первый блок SACK, как полагают, является блоком D-SACK, если многократные последовательности дублированы, только первое содержится в блоке D-SACK.

D-SACK TCP и повторная передача

- D-SACK позволяет TCP определить, когда ретрансляция не нужна (он получает D-SACK, после повторной передаче сегмента). Когда принимается такое решение, отправитель может "отменить" разбиение окна очереди пополам, так как он будет это делать, когда будет повторно передан сегмент (как это предполагает сетевая очередь).
- D-SACK также позволяет TCP определить, дублирует ли ли сеть пакеты (он получит D-SACK для сегмента, который был отправлен только один раз).
- Минус D-SACK в том, что он не позволяет отправителю определить получены оригинальный и повторный сегмент или оригинал утерян и сегмент дублируется в сети.

Взаимодействие SACK и D-SACK

- Нет никакой разницы между SACK и D-SACK, за исключением того, что первый блок SACK используется, для сообщения о повторяющемся сегменте в D-SACK.
- Для DSACK нет отдельного согласования / опций
- Нет проблем взаимодействия получателя использующего D-SACK и отправителя использующего традиционные SACK. Поскольку дубликат, о котором сообщается, все еще SACKed (во второй или более раз), нет проблем с SACK TCP, использующим это расширение с D-SACK TCP (хотя конкретные данные D-SACK не используются)

FACK (Forward Acknowledgment)

[FACK] Mathis, M. and J. Mahdavi, "Forward acknowledgement: refining TCP congestion control", ACM SIGCOMM Computer Communication Review Volume 26, Issue 4, DOI 10.1145/248157.248181, August 1996, <https://doi.org/10.1145/248157.248181>.

Matthew Mathis http://conferences.sigcomm.org/sigcomm/1996/papers/mathis.pdf

- Алгоритм управления перегрузкой с предварительным подтверждением (FACK) устраняет проблемы производительности
- Алгоритм FACK основан на первых принципах управления перегрузкой и предназначен для использования с предлагаемой опцией TCP SACK.
- Отделяя управление перегрузкой от других алгоритмов, таких как восстановление данных, достигается более точный контроль над потоком данных в сети.
- Вводится два дополнительных алгоритма для улучшения поведения в конкретных ситуациях.

Forward Acknowledgment (FACK) congestion control algorithm

- Алгоритм FACK использует дополнительную информацию, предоставляемую опцией SACK, для точного измерения общего количества байтов данных, находящихся в очереди в сети.
  - Reno и Reno SACK пытаются оценить это, предполагая, что каждый полученный дубликат ACK представляет один сегмент, покинувший сеть.
- Алгоритм FACK вводит две новые переменные состояния snd.fack и retran_data у отправителя, который должен сохранять информацию о блоках данных, хранящихся у получателя, которая требуется для использования информации SACK для правильной повторной передачи данных
  - В дополнение к тому, что необходимо для управления повторной передачей данных, необходимо хранить информацию о повторно передаваемых сегментах, чтобы точно определить, когда они покинули сеть.

FACK congestion control algorithm

- переменная, snd:fack обновляется, чтобы отразить самые передовые данные, хранящиеся в приемнике. В невосстановленных состояниях переменная snd:fack обновляется на основе номера подтверждения в заголовке TCP и совпадает с snd:una. Во время восстановления (пока получатель содержит несмежные данные) отправитель продолжает обновлять snd:una, используя номер подтверждения в заголовке TCP, но использует информацию, содержащуюся в параметрах TCP SACK, для обновления snd:fack.
- Когда получен блок SACK, который подтверждает данные с более высоким порядковым номером, чем текущее значение snd:fack, snd:fack обновляется, чтобы отразить самый высокий известный *порядковый номер*, который был получен, *плюс один*.

RACK-TLP для TCP

Алгоритм обнаружения потерь RACK-TLP для TCP

Y. Cheng, N. Cardwel,l N. Dukkipati, P. Jha
«The RACK-TLP Loss Detection Algorithm for TCP»
RFC 8985, February 2021

Алгоритм обнаружения потерь RACK-TLP для TCP

- RACK-TLP:
  - Recent Acknowledgment (RACK) - последний ACK
  - Tail Loss Probe (TLB) – зонд потери хвоста
- RACK-TLP использует метки времени передачи для каждого сегмента и выборочные подтверждения (SACK) и состоит из двух частей.
  1. Функция Recent Acknowledgment (RACK, последнее подтверждение) быстро запускает быстрое восстановление с использованием основанных на времени выводов, полученных из обратной связи подтверждения (ACK),
  2. а датчик потери хвоста (TLP) использует RACK и отправляет тестовый пакет для запуска обратной связи ACK, чтобы избежать событий тайм-аута повторной передачи (RTO).
- По сравнению с широко используемым пороговым подходом с дублирующим подтверждением (DupAck), RACK-TLP более эффективно обнаруживает потери, когда есть потоки данных, ограниченные приложением, потерянные повторные передачи или события переупорядочения пакетов данных. Он задуман как альтернатива пороговому подходу DupAck.

RFC 8985 Fast Recover

3.4. An Example of RACK-TLP in Action: Fast Recovery

The following example in Figure 1 illustrates the RACK-TLP algorithm in action:

```
Event  TCP DATA SENDER                              TCP DATA RECEIVER
-----  ------------------------------------------------------------
  1.   Send P0, P1, P2, P3          -->
       [P1, P2, P3 dropped by network]

  2.                                <--          Receive P0, ACK P0

  3a.  2RTTs after (2), TLP timer fires
  3b.  TLP: retransmits P3          -->

  4.                                <--         Receive P3, SACK P3

  5a.  Receive SACK for P3
  5b.  RACK: marks P1, P2 lost
  5c.  Retransmit P1, P2            -->
       [P1 retransmission dropped by network]

  6.                                <--    Receive P2, SACK P2 & P3

  7a.  RACK: marks P1 retransmission lost
  7b.  Retransmit P1                -->

  8.                                <--          Receive P1, ACK P3
```

Figure 1: RACK-TLP Protocol Example

RFC9885 9. Discussion

9.1. Advantages and Disadvantages

- The biggest advantage of RACK-TLP is that every data segment, whether it is an original data transmission or a retransmission, can be used to detect losses of the segments sent chronologically prior to it. This enables RACK-TLP to use fast recovery in cases with application-limited flights of data, lost retransmissions, or data segment reordering events. Consider the following examples:
- Самым большим преимуществом RACK-TLP является то, что каждый сегмент данных, будь то исходная передача данных или повторная передача, может использоваться для обнаружения потерь сегментов, отправленных в хронологическом порядке до него. Это позволяет RACK-TLP использовать быстрое восстановление (Fast Recover) в случаях ограниченных полетов данных, потерянных повторных передач или событий переупорядочивания сегментов данных. Рассмотрим следующие три примера:

Обсуждение, пример 1

1. ***Пакет отбрасывается в конце потока данных приложения***: рассмотрим отправителя, который передает ограниченный приложением поток из трех сегментов данных (P1, P2, P3), а P1 и P3 потеряны.

Предположим, что передача каждого сегмента не меньше RACK.reo_wnd после передачи предыдущего сегмента. RACK пометит P1 как потерянный, когда будет получен SACK от P2, и это вызовет повторную передачу P1 как R1. Когда R1 будет кумулятивно подтвержден, RACK пометит P3 как потерянный, и отправитель повторно передаст P3 как R3. Этот пример иллюстрирует, как RACK может восстанавливать определенные потери в хвосте транзакции без восстановления по RTO.

Обратите внимание, что ни обычный порог дублирования ACK [RFC5681] , ни алгоритм восстановления после потери [RFC6675], ни алгоритм прямого подтверждения [FACK] не могут обнаружить такие потери из-за требуемого количества сегментов или последовательностей.

Обсуждение, пример 2

2. ***Потеря повторной передачи***: рассмотрим поток трех отправленных сегментов данных (P1, P2, P3); P1 и P2 отбрасываются.

Предположим, что передача каждого сегмента не меньше RACK.reo_wnd после передачи предыдущего сегмента. Когда P3 получает SACK, RACK помечает P1 и P2 как потерянные, и они будут повторно переданы как R1 и R2. Предположим, что R1 снова потерян, но R2 SACKed; RACK пометит R1 как потерянный и снова запустит повторную передачу.

Опять же, ни традиционный подход порога ACK с тремя дубликатами, ни алгоритм восстановления потерь [RFC6675] , ни Forward Acknowledgment [FACK]алгоритм может обнаруживать такие потери. И такая потерянная повторная передача может произойти, когда скорость TCP ограничена, в частности, политиками корзины маркеров с большой глубиной корзины и низким пределом скорости; в таких случаях повторные передачи часто теряются неоднократно, потому что стандартное управление перегрузкой требует нескольких круговых обходов, чтобы снизить скорость ниже контролируемой скорости.

Обсуждение, пример 3

3. ***Переупорядочивание пакетов:*** рассмотрим простое событие переупорядочивания, когда поток сегментов отправляется как (P1, P2, P3). P1 и P2 несут полную полезную нагрузку октетов максимального размера отправителя (MSS), а P3 имеет только 1-октетную полезную нагрузку.

Предположим, что отправитель ранее обнаружил переупорядочение, и поэтому RACK.reo_wnd равно min_RTT/4. Теперь P3 переупорядочивается и доставляется первым, до P1 и P2. Пока P1 и P2 доставляются в течение min_RTT/4, RACK не будет считать P1 и P2 потерянными. Но если P1 и P2 доставлены за пределами окна повторного заказа, RACK все равно будет ложно помечать P1 и P2 как потерянные.

- Приведенные выше примеры показывают, что RACK-TLP особенно полезен, когда отправитель ограничен приложением, что может произойти с интерактивным трафиком или трафиком запроса/ответа. Точно так же RACK по-прежнему работает, когда отправитель ограничен окном приема, что может произойти с приложениями, использующими окно приема для ограничения отправителя.

ЛИТЕРАТУРА

1. Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

2. Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, DOI 10.17487/RFC2883, July 2000, https://www.rfc-editor.org/info/rfc2883

3. Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, DOI 10.17487/RFC6675, August 2012, <https://www.rfc-editor.org/info/rfc6675>.

4. Mathis, M. and J. Mahdavi, "Forward acknowledgement: refining TCP congestion control", ACM SIGCOMM Computer Communication Review Volume 26, Issue 4, DOI 10.1145/248157.248181, August 1996, <https://doi.org/10.1145/248157.248181>.

ЛИТЕРАТУРА (rus)

1. Матис М. , Махдави Дж. , Флойд С. и А. Романов , «Параметры выборочного подтверждения TCP» , RFC 2018 , DOI 10.17487/RFC2018 ,октябрь 1996 г., <https://www.rfc-editor.org/info/rfc2018>.
2. Флойд С. , Махдави Дж. , Матис М. и Подольский М. , «Расширение опции выборочного подтверждения (SACK) для TCP» , RFC 2883 , DOI 10.17487/RFC2883 ,июль 2000 г.,<https://www.rfc-editor.org/info/rfc2883 >