# Распределенные информационно-аналитические системы

## Лекция № 5.

## Протоколы связи

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

## *Учебные цели:*

*Изучить основы научных знаний по сбалансированному протоколу скользящего окна; протоколу, основанному на таймере.*

## *Учебные вопросы:*

***1. Сбалансированный протокол скользящего окна.***

***2. Протокол, основанный на таймере.***

В лекции рассматриваются два протокола, которые используются для надежного обмена данными между двумя ЭВМ. В идеальном случае данными обмениваются, посылая и получая сообщения. К сожалению, не всегда можно игнорировать возможность ошибок связи; сообщения должны передаваться через физическую среду, которая может терять, дублировать, переупорядочивать или искажать сообщения. Эти ошибки должны быть обнаружены и исправлены дополнительными механизмами, выполняющимися на ЭВМ, которые традиционно называются ***протоколами***.

Основная функция таких протоколов – передача данных, то есть, принятие информации на одной ЭВМ и получение ее на другой ЭВМ. Надежная передача данных включает повторную посылку потерянных сообщений, отклонение или исправление искаженных сообщений и игнорирование дублирующихся сообщений. Для выполнения этих функций протокол содержит информацию о состоянии, записывая, какие данные уже был посланы, какие данные считаются полученными и т.д. Необходимость использования информации о состоянии обусловливает проблему управления соединением, то есть, инициализации и отбрасывания информации о состоянии.

***Инициализацией*** называется открытие соединения, а ***отбрасыванием*** называется закрытие соединения. Трудности управления соединением возникают из-за того, что сообщение может оставаться в каналах связи, когда соединение закрыто. Такое сообщение может бы быть получено в отсутствие соединения или в течение более позднего соединения, и получение не должно нарушать правильную операцию текущего соединения.

Протоколы, обсуждаемые в лекции разработаны для различных уровней в иерархии протокола, типа модели OSI. Первый протокол полностью асинхронный, в то время как второй протокол полагается на правильное использование таймеров. В обоих случаях заострим внимание на требуемом свойстве безопасности, а именно на том, что приемник получит только правильные данные.

Первый протокол разработан для обмена данными между двумя ЭВМ, которые имеют прямое физическое соединение (типа телефонной линии), и, следовательно, принадлежит канальному уровню модели OSI.

Второй протокол разработан для двух ЭВМ, которые связываются через промежуточную сеть (возможно содержащую другие ЭВМ и соединяющую ЭВМ различными путями), и этот протокол, следовательно, принадлежит к транспортному уровню OSI модели.

Это различие отражается на функциональных возможностях, требуемых от протоколов, следующим образом.

***Рассматриваемые ошибки.*** Для двух протоколов будут рассматриваться различные классы ошибок передачи данных. При физическом соединении сообщения не могут пересекаться, и они не могут быть продублированы; таким образом, в первом учебном вопросе рассматривается только потеря сообщений. В сети сообщения могут передаваться различными путями, и, следовательно, пересекаться; также, из-за отказов промежуточных ЭВМ сообщения могут быть продублированы или потеряны. Поэтому во втором учебном вопросе будут рассматриваться потеря, дублирование и переупорядочение сообщений.

***Управление соединением.*** Управление соединением не будет рассматриваться для первого протокола, но будет для второго. Предполагается, что физическое соединение функционирует непрерывно в течение очень длительного времени, а не открывается и закрывается неоднократно. Для соединений с удаленными ЭВМ это не так. Такое соединение может быть необходимо временно для обмена некоторыми данными, но обычно слишком дорого поддерживать соединение с каждой удаленной ЭВМ неопределенно долго. Следовательно, для второго протокола будет рассмотрена способность открывать и закрывать соединение.

При рассмотрении первого протокола показано, что не только механизмы, основанные на таймерах, могут обеспечить требуемые свойства безопасности протоколов передачи данных. Первый протокол служит примером доказательства реализации безопасности с помощью инструментальных средств. Правильное использование таймеров и ограничение на время, в течение которого сообщение может передаваться, также необходимы для безопасного управления соединением. Таким образом, для того, чтобы доказать безопасность протоколов, нужно принимать во внимание роль таймеров в управлении соединением. При рассмотрении второго протокола показано, как модель распределенных систем может быть расширена до процессов, использующих таймеры, и рассмотрен пример такого расширения.

***Искажение сообщений.*** Естественно принять во внимание возможность того, что сообщения могут быть искажены в течение передачи. Содержание сообщения, переданного через физическое соединение, может быть повреждено из-за атмосферных шумов, плохо функционирующих модулей памяти и т.д. Однако можно предположить, что искажение сообщения может быть обнаружено процессом-получателем, например, посредством контроля четности или более общих механизмов контрольной суммы. Получение искаженного сообщения затем представляется так, как будто не было получено никакого сообщения, и таким образом, искажение сообщения фактически вызывает его потерю. По этой причине искажение не обрабатывается явно; вместо этого всегда рассматривается возможность потери сообщения.

# 1. Сбалансированный протокол скользящего окна

Рассмотрим симметричный протокол, который обеспечивает надежный обмен информацией в обоих направлениях. Поскольку он используется для обмена информацией между ЭВМ, которые непосредственно соединены линией связи, можно предположить, что каналы имеют дисциплину FIFO.

Два процесса связи обозначаются как p и q. Предположения, требования и протокол абсолютно симметричны.

Вход p состоит из информации, которую он должен послать q, и моделируется неограниченным массивом слов $in_p$.

Выход p состоит из информации, которую он получает от q, и также моделируется неограниченным массивом слов, $out_p$.

Предполагается, что p имеет случайный доступ по чтению к $in_p$ и случайный доступ по записи к $out_p$. Первоначально значение $out_p[i]$ не определено и представлено как udef для всех i.

Вход и выход процесса q моделируется массивами $in_q$ и $out_q$, соответственно.

Эти массивы нумеруются натуральными числами, то есть они начинаются со слова с номером 0. Далее будет показано, что произвольный доступ может быть ограничен доступом к «окну» конечной длины, передвигающемуся вдоль массива. Поэтому протокол называется ***протоколом «скользящего окна»***.

Процесс p содержит переменную $s_p$, показывающую наименьшее нумерованное слово, которое p все еще ожидает от q. Таким образом, в любой момент времени, p уже записал слова от $out_p[0]$ до $out_p[s_p - 1]$. Значение $s_p$ никогда не уменьшается. Аналогично q содержит переменную $s_q$.

Теперь могут быть установлены ***требуемые свойства протокола***.

Свойство безопасности говорит о том, что каждый процесс передает только корректные данные; свойство живучести говорит о том, что все данные когда-либо будут доставлены.

***Свойство безопасности.*** В каждой достижимой конфигурации протокола

$$out_p[0..s_p - 1] = in_q[0..S_p - 1] \text{ и } out_q[0..s_q - 1] = in_p[0...s_q - 1].$$

***Свойство живучести.*** Для каждого целого $k \geq 0$ конфигурации с $s_p \geq k$ и $s_q \geq k$ когда-либо достигаются.

**Представление протокола.** Протоколы передачи обычно полагаются на использование сообщений подтверждения. Сообщение подтверждения посылается процессом-получателем, чтобы сообщить отправителю о данных, которые он получил корректно. Если отправитель данных не получает подтверждение, то он предполагает, что данные (или подтверждение) потеряно, и повторно передает те же самые данные. В рассматриваемом протоколе, однако, не используются явные сообщения подтверждения. В этом протоколе обе ЭВМ имеют сообщения, которые нужно послать другой ЭВМ; сообщения каждой ЭВМ служат, в свою очередь, подтверждениями сообщений другой ЭВМ.

Сообщения, которыми обмениваются процессы, называют ***пакетами***, и они имеют форму <**pack**, w, i>, где w – слово данных, а i – натуральное число (называемое порядковым номером пакета). Этот пакет, посылаемый процессом p (к q), передает слово = $in_p[i]$ для q, но также, как было отмечено, подтверждает получение некоторого количества пакетов от q. Процесс p может быть «впереди» q не более, чем на $l_p$ пакетов, если необходимо, чтобы пакет данных <**pack**, w, i>, посланный p, подтверждал получение слов с номерами 0.. $i - l_p$ от q (q посылает аналогичные пакеты.) Константы $l_p$ и $l_q$ неотрицательны и известны обоим процессам p и q. Использование пакета данных в качестве подтверждения имеет два последствия для протокола:

(1) Процесс p может послать слово $in_p[i]$ (в виде пакета <**pack**, $in_p[i]$, i>) только после того, как запишет все слова от $out_p[0]$ до $out_p[i - l_p]$, то есть если $i < s_p + l_p$.

(2) Когда p принимает <**pack**, w, i>, повторная передача слов с $in_p[0]$ до $in_p[i - l_q]$ уже не нужна.

***Алгоритм протокола.*** После выбора модели нетрудно разработать алгоритм протокола. Для процесса p введена переменная $a_p$ (и $a_q$ для q), в которой хранится самое первое слово, для которого p (или q, соответственно) еще не получил подтверждение.

***Протокол скользящего окна (для p):***

var $s_p$, $a_p$ : integer  init 0, 0 ;
$in_p$ : array of word  (* Посылаемые данные *) ;
$out_p$ : array of word  init udef, udef, ...,
$S_p$: {$a_p$ £ i < $S_p$ + $l_p$}
begin send <**pack**, $in_p$[i], i> to q end
$R_p$: { <**pack**, w, i> ∈ $Q_p$ }
begin receive <**pack**, w, i> ;
if $out_p$[i] = udef then
begin $out_p$[i] := w ;
$a_p$ := max($a_p$, i – $l_p$ + 1) ;
$S_p$ := min{j|$out_p$[j]= udef}
end
(* else игнорируем, пакет передавался повторно *)
end
$L_p$: {<**pack**,w, i> ∈ $Q_p$}
begin $Q_p$ := $Q_p$ \ {<**pack**, w, i>} end

В алгоритме протокола скользящего окна действие $S_p$ – посылка i-го слова процессом p, действие $R_p$ – принятие слова процессом p, и действие $L_p$ – потеря пакета с местом назначения p. Процесс p может послать любое слово, индекс которого попадает в указанные ранее границы. Когда сообщение принято, в первую очередь делается проверка – было ли идентичное сообщение принято ранее (на случай повторной передачи). Если нет, слово, содержащееся в нем, записывается в выход, $a_p$ и $s_p$ корректируются. Также вводятся действия $S_q$, $R_q$ и $L_q$, где p и q поменяны ролями.

***Инвариант протокола.*** Подсистема связи представляется двумя очередями: $Q_p$ – для пакетов с адресатом p и $Q_q$ – для пакетов с адресатом q. Заметим, что перевычисление $s_p$ в $R_p$ никогда не дает значение меньше предыдущего, поэтому $s_p$ никогда не уменьшается. Чтобы показать, что алгоритм удовлетворяет заданным ранее требованиям, сначала покажем, что утверждение P – инвариант. (В этом и других утверждениях i – натуральное число.)

$P \equiv \forall i < s_p : out_p[i] \neq udef$ (0p)

$\wedge \forall i < s_q, : out_q[i] \neq udef$ (0q)

$\wedge <\mathbf{pack}, w, i> \in Q_p \Rightarrow w = in_q[i] \wedge (i < s_q + l_q)$ (lp)

$\wedge <\mathbf{pack}, w, i> \in Q_q \Rightarrow w = in_p[i] \wedge (i < s_p + l_p)$ (lq)

$\wedge out_p[i] \neq udef \Rightarrow out_p[i] = in_q[i] \wedge (a_p > i - l_q)$ (2p)

$\wedge out_q[i] \neq udef \Rightarrow out_q[i] = in_p[i] \wedge (a_q > i - l_p)$ (2q)

$\wedge a_p \leq s_q,$ (3p)

$\wedge a_q \leq s_p$ (3q)

**Лемма 1.** P – инвариант алгоритма протокола скользящего окна.

*Доказательство.* В любой начальной конфигурации $Q_p$ и $Q_q$ – пустые, для всех i, $out_p[i]$ и $out_q[i]$ равны udef, и $a_p$, $a_p$, $s_p$ и $s_q$ равны нулю 0; из этого следует, что P = true. Перемещения протокола рассмотрим с точки зрения сохранения значения P.

Во-первых, заметим, что значения $in_p$ и $in_q$, никогда не меняются.

$S_p$: Чтобы показать, что $S_p$ сохраняет (0p), заметим, что $S_p$ не увеличивает $s_p$ и не делает ни один из $out_p[i]$ равным udef.

Чтобы показать, что $S_p$ сохраняет (0q), заметим, что $S_p$ не увеличивает $s_q$, и не делает ни один из $out_q[i]$ равным udef.

Чтобы показать, что $S_p$ сохраняет (1p), заметим, что $S_p$ не добавляет пакеты в $Q_p$ и не уменьшает $s_p$.

Чтобы показать, что $S_p$ сохраняет (lq), заметим $S_p$ добавляет <**pack**, w, i> в $Q_p$ с $w = in_p[i]$ и $i < s_p + l_p$, и не изменяет значение $s_p$.

Чтобы показать, что $S_p$ сохраняет (2p) и (2q), заметим, что $S_p$ не изменяет значения $out_p$, $out_q$, $a_p$ или $a_q$.

Чтобы показать, что $S_p$ сохраняет (3p) и (3q), заметим, что $S_p$ не меняет значения $a_p$, $a_q$, $s_q$ или $s_p$.

$R_p$: Чтобы показать, что $R_p$ сохраняет (0p), заметим, что $R_p$ не делает ни одно $out_p[i]$ равным udef, и если он перевычисляет $s_p$, то оно впоследствии также удовлетворяет (0p).

Чтобы показать, что $R_p$ сохраняет (0q), заметим, что $R_p$ не меняет $out_q$ или $s_q$.

Чтобы показать, что $R_p$ сохраняет (lp), заметим, что $R_p$ не добавляет пакеты в $Q_p$ и не уменьшает $s_q$.

Чтобы показать, что $R_p$ сохраняет (lq), заметим, что $R_p$ не добавляет пакеты в $Q_q$ и не уменьшает $s_p$.

Чтобы показать, что $R_p$ сохраняет (2p), заметим, что $R_p$ изменяет значение $out_p[i]$ на w при принятии <**pack**, w, i>. Так как Qp содержала этот пакет до того, как выполнился $R_p$, из (1p) следует, что $w = in_p[i]$. Присваивание $a_p := \max(a_p, i - l_q + 1)$ гарантирует, что $a_p > i - l_q$ сохраняется после выполнения. Чтобы показать, что $R_p$ сохраняет (2q), заметим, что $R_p$ не меняет значения $out_q$ или $a_q$.

Чтобы показать, что $R_p$ сохраняет (3p), заметим, что когда $R_p$ присваивает $a_p := \max(a_p, i - l_q + 1)$ (при принятии <**pack**, w, i>), из (lp) следует, что $i < s_q + l_q$, следовательно $a_p \le s_q$ сохраняется после присваивания. $R_p$ не меняет $s_q$. Чтобы показать, что $R_p$ сохраняет (3q), заметим, что $s_p$ может быть увеличен только при выполнении $R_p$.

$L_p$: Чтобы показать, что $L_p$ сохраняет (0p), (0q), (2p), (2q), (3p), и (3q), достаточно заметить, что $L_p$ не меняет состояния процессов. (lp) и (lq) сохраняются потому, что $L_p$ только удаляет пакеты (а не порождает или искажает их). Процессы $S_q$, $R_q$ и $L_q$ сохраняют P, что следует из симметрии. □

**Доказательство правильности протокола.** Продемонстрируем, что алгоритм протокола скользящего окна гарантирует безопасную и окончательную доставку. Безопасность следует из инварианта, а живучесть продемонстрировать труднее.

**Теорема 1.** Алгоритм протокола скользящего окна удовлетворяет требованию безопасной доставки.

*Доказательство.* Из (0p) и (2p) следует, что $out_p[0..s_p - 1] = in_q[0..s_p - 1]$, а из (0q) и (2q) следует $out_p[0..S_q - 1] = in_p[0..S_q - 1]$. □

Чтобы доказать живучесть протокола, необходимо сделать предположения относительно $l_p$ и $l_q$. Без этих предположений протокол не удовлетворяет свойству живучести, что может быть показано следующим образом. Неотрицательные константы $l_p$ и $l_q$ еще не определены; если их выбрать равными нулю, начальная конфигурация протокола окажется тупиковой. Поэтому предполагается, что $l_p + l_q > 0$.

Конфигурация протокола может быть обозначена $g = (c_p, c_q, Q_p, Q_q)$, где $c_p$ и $c_q$ – состояния p и q. Пусть g будет конфигурацией, в которой применим $S_p$ (для некоторого i). Пусть

$$d = S_p(g) = (c_p, c_q, Q_p, (Q_q \cup \{m\})),$$

и отметим, что действие $L_q$ применимо в d. Если $L_q$ удаляет m, $L_q(d) = g$. Отношение $L_q(S_p(g)) = g$ дает начало неограниченным вычислениям, в которых ни $s_p$ , ни $s_q$ не уменьшаются.

Протокол удовлетворяет требованию «окончательной доставки», если удовлетворяются ***два следующих справедливых предположения***:

F1. Если посылка пакета возможна в течение бесконечно долгого временно, пакет посылается бесконечно часто.

F2. Если один и тот же пакет посылается бесконечно часто, то он принимается бесконечно часто.

Предположение F1 гарантирует, что пакет посылается снова и снова, если не получено подтверждение; F2 исключает вычисления, подобные описанному выше, когда повторная передача никогда не принимается.

Ни один из двух процессов не может быть намного впереди другого: разница между $s_p$ и $s_q$ остается ограниченной. Поэтому протокол называется ***сбалансированным***, а также из этого следует, что если требование окончательной доставки удовлетворяется для $s_p$, тогда оно также удовлетворяется для $s_q$, и наоборот. Понятно, что протокол не следует использовать в ситуации, когда один процесс имеет намного больше слов для пересылки, чем другой.

**Лемма 2.** Из P следует $s_p - l_q \le a_p \le s_q \le a_q + l_p \le s_p + l_p$.

*Доказательство.* Из (0p) и (2p) следует $s_p - l_q \le a_p$, из (3p) следует $a_p \le s_p$. Из (0q) и (2q) следует $s_p \le a_p + l_p$. Из (3q) следует $a_p + l_p \le s_p + l_p$. □

**Теорема 2.** Алгоритм протокола скользящего окна удовлетворяет требованию окончательной доставки.

*Доказательство.* Сначала продемонстрируем, что в протоколе невозможны тупики. Из инварианта следует, что один из двух процессов может послать пакет, содержащий слово с номером, меньшим, чем ожидается другим процессом.

**Утверждение 1.** Из P следует, что посылка <**pack**, $in_p[s_q]$, $s_q$> процессом p или посылка <**pack**, $in_q[s_p]$, $s_p$> процессом q возможна.

*Доказательство.* Так как $l_p + l_q > 0$, хотя бы одно из неравенств Леммы 2 строгое, то есть $s_q < s_p + l_p \vee s_p < s_q + l_q$.

Из P следует, что $a_p \le s_q$ (3p) и $a_q \le s_p$ (3q), а также следует, что $(a_p \le s_q < s_p + l_p) \vee (a_q \le s_p < s_q + l_q)$. Это значит, что $S_p$ применим с $i = s_q$ или $S_q$ применим с $i = s_p$. □

Теперь мы можем показать, что в каждом из вычислений $s_p$ и $s_q$ увеличиваются бесконечно часто. Согласно Утверждению 1 протокол не имеет терминальных конфигураций, следовательно каждое вычисление неограниченно. Пусть C – вычисление, в котором $s_p$ и $s_q$ увеличиваются ограниченное число раз, и пусть $s_p$ и $s_q$ – максимальные значения, которые эти переменные принимают в C. Согласно утверждению, посылка <**pack**, $in_p[s_q]$, $s_q$> процессом p или посылка <**pack**, $in_q[s_p]$, $s_p$> процессом q применима всегда после того, как $s_p$, $s_q$, $a_p$ и $a_q$ достигли своих окончательных значений. Таким образом, согласно F1, один из этих пакетов посылается бесконечно часто, и согласно F2, он принимается бесконечно часто. Но, так как принятие пакета с порядковым номером $s_p$ процессом p приводит к увеличению $s_p$ (и наоборот для q), это противоречит допущению, что ни $s_p$, ни $s_q$ не увеличиваются более. Таким образом Теорема 2 доказана. □

Предположение F2 – минимальное требование, которому должен удовлетворять канал, соединяющий p и q для того, чтобы он мог передавать данные. Очевидно, если некоторое слово $in_p[i]$ никогда не проходит через канал, то невозможно достичь окончательной доставки слова.

Предположение F1 обычно реализуется в протоколе с помощью условия превышения времени: если $a_p$ не увеличилось в течение определенного промежутка времени, $in_p[a_p]$ передается опять. Для этого протокола безопасная доставка может быть доказана без принятия во внимания проблем времени (тайминга).

**Анализ протокола.**

***Ограничение памяти в процессах.*** Алгоритм протокола скользящего окна не годится для реализации в компьютерной сети, так как в каждом процессе хранится бесконечное количество информации (массивы in и out) и так как он использует неограниченные порядковые номера. Покажем, что достаточно хранить только ограниченное число слов в каждый момент времени. Пусть $L = l_p + l_q$.

**Лемма 3.** Из P следует, что отправление <**pack**, w, i> процессом p применимо только для $i < a_p + L$.

*Доказательство.* Действие $S_p$ требует $i < s_p + l_p$, значит согласно Лемме 2 $i < a_p + L$. □

**Лемма 4.** Из P следует, что если $out_p[i] \neq udef$, то $i < s_p + L$.

*Доказательство.* Из (2p), $a_p > i - l_q$, значит $i < a_p + l_q$, и $i < s_p + L$ (из Леммы 2). □

Последствия этих двух лемм отображены на Рисунке 1.

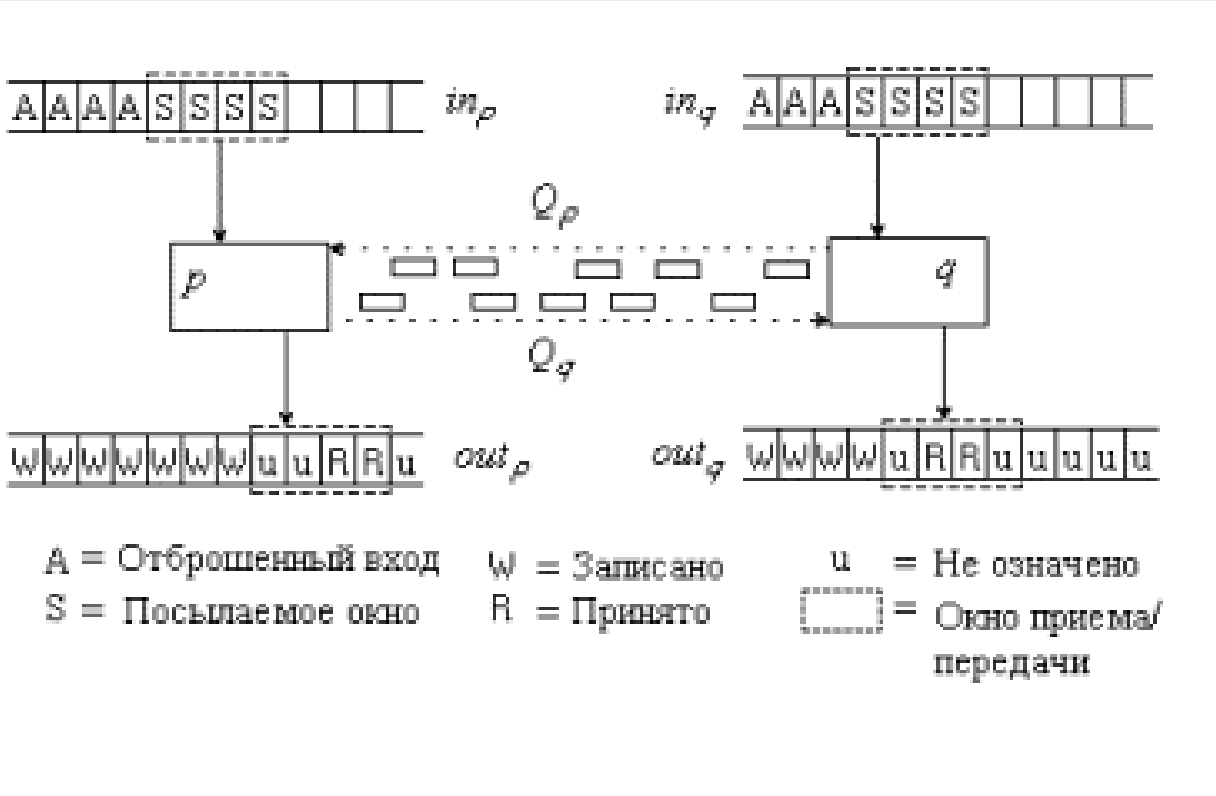


**Рисунок 1 – Скользящие окна протокола**

Процессу p необходимо хранить только слова $in_p[a_p..s_p + l_p - 1]$, потому что это слова, которые p может послать. Назовем их как ***посылаемое окно*** p (представлено как S на рисунке 1).

Каждый раз, когда $a_p$ увеличивается, p отбрасывает слова, которые больше не попадают в посылаемое окно (они представлены как A на рисунке 1). Каждый раз, когда $s_p$ увеличивается, p считывает следующее слово из посылаемого окна от источника, который производит слова. Согласно Лемме 3, посылаемое окно процесса p содержит не более L слов.

Подобным же образом можно ограничить память для хранения процессом p массива $out_p$. Так как $out_p[i]$ не меняется для $i < s_p$, можно предположить, что p выводит эти слова окончательно и более не хранит их (они представлены как W на рисунке 1). Так как $out_p[i] = udef$ для всех $i \geq s_p + L$, эти значения $out_p[i]$ также не нужно хранить. Подмассив $out_p[s_p..s_p + L - 1]$ назовем ***принимаемым окном*** p. Принимаемое окно представлено на рисунке 1 как u для неозначенных слов и R для слов, которые были приняты. Только слова, которые попадают в это окно, хранятся процессом. Леммы 3 и 4 показывают, что не более 2L слов хранятся процессом в любой момент времени.

***Ограничение чисел последовательности.*** Числа последовательности могут быть ограничены, если используются FIFO-каналы. При использовании FIFO-предположения можно показать, что порядковый номер пакета, который получен процессом p всегда внутри 2L-окрестности, $s_p$. Обратим внимание, что FIFO предположение используется первый раз.

**Лемма 5.** Утверждение P', определяемое как

$P' \equiv P$

$\wedge$ <**pack**, w, i> is behind <**pack**, w', i'> in $Q_p \Rightarrow i > i' - L$ (4p)

$\wedge$ <**pack**, w, i> is behind <**pack**, w', i'> in $Q_q \Rightarrow i > i' - L$ (4q)

$\wedge$ <**pack**, w, i> $\in Q_p \Rightarrow i \geq a_p - l_p$ (5p)

$\wedge$ <**pack**, w, i> $\in Q_q \Rightarrow i \geq a_q - l_q$ (5q)

является инвариантом алгоритма протокола скользящего окна.

*Доказательство.* Так как уже было показано, что P – инвариант, мы можем ограничиться доказательством того, что (4p), (4q), (5p) и (5q) выполняются изначально и сохраняются при любом перемещении. Заметим, что в начальном состоянии очереди пусты, следовательно (4p), (4q), (5p) и (5q) выполняются. Покажем, что перемещения сохраняют истинность этих утверждений.

$S_p$: Чтобы показать, что $S_p$ сохраняет (4p) и (5p), заметим, что $S_p$ не добавляет пакетов в $Q_p$ и не меняет $a_p$.

Чтобы показать, что $S_p$ сохраняет (5q), заметим, что если $S_p$ добавляет пакет <**pack**, w, i> в $Q_q$, то $i \geq a_p$, откуда следует, что $i \geq a_q - l_q$ (из Леммы 2).

Чтобы показать, что $S_p$ сохраняет (4q), заметим, что если <**pack**, w', i'> в $Q_q$, тогда из (lq) $i' < s_p + l_p$, следовательно, если $S_p$ добавляет пакет <**pack**, w, i> с $i \geq a_p$, то из Леммы 2 следует $i' < a_p + L \leq i + L$.

$R_p$: Чтобы показать, что $R_p$ сохраняет (4p) и (4q), заметим, что $R_p$ не добавляет пакеты в $Q_p$ или $Q_q$.

Чтобы показать, что $R_p$ сохраняет (5p), заметим, что когда $a_p$ увеличивается (при принятии <**pack**, w', i'>) до $i' - l_q + 1$, тогда для любого из оставшихся пакетов <**pack**, w, i> в $Q_p$ мы имеем $i > i' - L$ (из 4p). Значит неравенство $i \geq a_p - l_p$ сохраняется после увеличения $a_p$.

Чтобы показать, что $R_p$ сохраняет (5q), заметим, что $R_p$ не меняет $Q_q$ и $a_q$.

$L_p$: Действие $L_p$ не добавляет пакетов в $Q_p$ или $Q_q$, и не меняет значения $a_p$ или $a_q$; значит оно сохраняет (4p), (4q), (5p) и (5q).

Из симметрии протокола следует, что $S_q$, $R_q$ и $L_q$ тоже сохраняет P'. □

**Лемма 6.** Из P' следует, что <**pack**, w, i> $\in Q_p \Rightarrow s_p - L \le i < s_p + L$ и <**pack**, w, i> $\in Q_q \Rightarrow s_q - L \le i < s_q + L$.

*Доказательство.* Пусть <**pack**, w, i> $\in Q_p$. Из (1p): $i < s_q + l_q$, и из Леммы 2: $i < s_p + L$. Из (5p): $i \ge a_p - l_p$, и из Леммы 2: $i \ge s_p - L$. Утверждение относительно пакетов в $Q_q$ доказывается так же. □

Согласно Лемме 6 достаточно посылать пакеты с порядковыми номерами modulo k, где $k \ge 2L$. В самом деле, имея $s_p$ и i mod k, p может вычислить i.

***Выбор параметров.*** Значения констант $l_p$ и $l_q$ сильно влияют на эффективность протокола. Оптимальные значения зависят от числа системно зависимых параметров, таких как:

- время связи (время между двумя последовательными операциями процесса);
- время задержки на обмен (среднее время на передачу пакета от p к q и получение ответа от q к p);
- вероятность ошибки (вероятность того, что конкретный пакет потерян).

***Протокол с поочередными битами.*** Интересный случай протокола скользящего окна получается, когда L = 1, то есть, $l_p = 1$ и $l_q = 0$ (или наоборот). Переменные $a_p$ и $a_q$, инициализируются значениями $-l_p$ и $-l_q$, а не 0. Можно показать, что $a_p + l_q = s_p$ и $a_q + l_p = s_q$ всегда выполняются, значит только одно $a_p$ и $s_p$ (и $a_q$ и $s_q$) нужно хранить в протоколе. Протокол с поочередными битами получается, если использование таймеров ограничивается, чтобы гарантировать, что ЭВМ посылают сообщения в ответ.

# 2. Протокол, основанный на таймере

Рассмотрим роль таймеров в проектировании и проверке протоколов связи, анализируя упрощенную форму Dt-протокола Флэтчера и Уотсона для сквозной передачи сообщений. Этот протокол обеспечивает не только механизм для передачи данных (как сбалансированный протокол скользящего окна), но также открытие и закрытие соединений. Он устойчив к потерям, дублированию и переупорядочению сообщений.

Информация о состоянии (передачи данных) протокола хранится в структуре данных, называемой ***запись соединения***. Запись соединения может быть создана и удалена при открытии и закрытии соединения. Таким образом, говорят, что соединение открыто (на одной из ЭВМ), если существует запись соединения.

Чтобы сконцентрироваться на релевантных аспектах протокола (а именно, на механизме управления соединением и роли таймеров в этом механизме), будем рассматривать упрощенную версию протокола. ***В рассматриваемом протоколе сделаны следующие упрощения:***

*Одно направление.* Подразумевается, что данные передаются в одном направлении, скажем от p к q. Иногда будем называть p ***отправителем***, а q – ***адресатом*** (***приемником***). Однако, следует отметить, что протокол использует сообщения подтверждения, которые посылаются в обратном направлении, то есть от q к p.

Обычно данные нужно передавать в двух направлениях. Чтобы предусмотреть подобную ситуацию, дополнительно выполняется второй протокол, в котором p и q поменяны ролями. Тогда можно ввести комбинированные data/ack (данные/подтверждения) сообщения, содержащие как данные (с соответствующим порядковым номером), так и информацию, содержащуюся в пакете подтверждения протокола, основанного на таймере.

*Окно приема из одного слова.* Приемник не хранит пакеты данных с номером, более высоким, чем тот, который он ожидает. Только если следующий пакет, который придет – ожидаемый, он немедленно принимается. Более интеллектуальные версии протокола хранили бы прибывающие пакеты с более высоким порядковым номером и принимали бы их после того, как пришли и были приняты пакеты с меньшими порядковыми номерами.

*Предположения, упрощающие синхронизацию.* Протокол рассмотрен с использованием минимального числа таймеров. Например, предполагается, что подтверждение может быть послано процессом-получателем в любое время, пока соединение открыто со стороны приемника. Также возможен случай, когда подтверждение может быть послано только в течение определенного интервала времени, но это делает протокол более сложным.

Также из описания протокола опущены таймерные механизмы, используемые для повторной передачи пакетов данных. Включен только механизм, гарантирующий безопасность протокола.

*Однословные пакеты.* Отправитель может помещать только одиночное слово в каждый пакет данных. Протокол был бы более эффективным, если бы пакеты данных могли содержать блоки последовательных слов.

Протокол основан на таймере, то есть процессы имеют доступ к физическим часовым устройствам. ***По отношению ко времени и таймерам в системе сделаны следующие предположения:***

*Глобальное время.* Глобальная мера времени простирается над всеми процессами системы, то есть каждое событие происходит в некоторое время. Предполагается, что каждое событие имеет продолжительность 0, и время, в которое происходит событие, не доступно процессам.

*Ограниченное время жизни пакета.* Время жизни пакета ограничено константой m (максимальное время жизни пакета). Таким образом, если пакет посылается во время s и принимается во время t, то $s < t < s + m$.

Если пакет дублируется в канале, каждая копия должна быть принята в течение промежутка времени m после отправления оригинала (или стать потерянной).

*Таймеры.* Процессы не могут наблюдать абсолютное время своих действий, но они имеют доступ к таймерам. ***Таймер*** – действительная переменная процесса, чье значение непрерывно уменьшается со временем (если только ей явно не присваивают значение). Точнее, если Xt – таймер, мы обозначаем его значение в момент времени t как Xt(t), и если Xt между $t_1$ и $t_2$ не присвоено иное значение, то $Xt(t_1) - Xt(t_2) = t_2 - t_1$.

Заметим, что эти таймеры работают так: в течение времени d они уменьшаются точно на d.

Входные слова для отправителя моделируются неограниченным массивом $in_p$. Снова этот массив не полностью хранится в p; p в каждый момент времени имеет доступ только к его части. Часть $in_p$, к которой p имеет доступ расширяется (в сторону увеличения индексов), когда p получает следующее слово от процесса, который их генерирует. Эту операцию будем называть как ***принятие слова отправителем***.

В этом протоколе моделирование слов, принятых приемником, отлично от моделирования слов в предыдущем протоколе. Вместо того, чтобы записывать (бесконечный) массив, приемник передает слова процессу потребления операцией, называемой ***доставка слова***. В идеале, каждое слово $in_p$ должно быть доставлено точно один раз, и слова должны быть доставлены в правильном порядке.

Спецификация протокола, однако, слабее, и причина в том, что протокол позволяется обрабатывать каждое слово $in_p$ только в течение ограниченного интервала времени. Не каждый протокол может гарантировать, что слово принимается за ограниченное время потому, что возможно, что все пакеты в это время потеряются. Следовательно, спецификация протокола учитывает возможность сообщенной потери, когда протокол отправителя генерирует отчет об ошибке, указывающий, что слово возможно потеряно. (Если после этого протокол более высокого уровня предлагает это слово p снова, то возможно дублирование; но мы не будем рассматривать эту проблему).

Свойства протокола:

*Нет потерь.* Каждое слово $in_p$ доставляется процессом q или посылается отчет процессом p («возможно потеряно») в течение ограниченного времени после принятия слова процессом p.

*Упорядочение.* Слова, доставляемые q принимаются в строго возрастающем порядке (так же, как они появляются в $in_p$).

**Представление протокола.** Соединение в протоколе открыто, если прежде не существовало никакого соединения, и если (для отправителя) принято следующее слово или (для приемника) прибывает пакет, который может быть доставлен. Таким образом, в этом протоколе, чтобы открыть соединение нет необходимости обмениваться какими-либо сообщениями управления прежде, чем могут быть посланы пакеты данных. Это делает протокол относительно эффективным для прикладных программ, где в каждом соединении передаются только несколько слов (маленькие пакеты связи). Предикат cs (или cr, соответственно) истинен, когда отправитель (или приемник, соответственно) имеет открытое соединение. Это, обычно, не явная булева переменная отправителя (или приемника, соответственно); вместо этого открытое соединение определяется существованием записи соединения. Процесс проверяет открыто ли соединение, пытаясь найти запись соединения в списке открытых соединений.

Когда отправитель открывает новое соединение, он начинает нумеровать принятые слова с 0. Количество уже принятых слов в данном соединении обозначается High, и количество слов, для которых уже было получено подтверждение обозначается через Low. Это подразумевает, что отправитель может передавать пакеты с порядковыми номерами в диапазоне от Low до High – 1, но есть здесь и своя особенность. Отправитель может посылать слово только в течение промежутка времени длиной U, начиная с того момента, когда отправитель принял слово. Для этого с каждым словом $in_p[i]$ ассоциируется таймер Ut[i], он устанавливается в U в момент принятия, и должен быть положительным для передаваемого слова. Таким образом, посылаемое окно p состоит из тех слов с индексами Low ...High – 1, для которых ассоциированный с ними таймер положителен.

***Переменные протокола, основанного на таймере:***

**Сетевые константы:**

```
m : real ;      (* Максимальное время жизни пакета *)
```

**Константы протокола:**

```
U : real ;      (* Длина интервала отправки *)
R : real ;      (* Значение тайм-аута приемника: R ≥ U + m *)
S : real ;      (* Значение тайм-аута отправителя: S ≥ R + 2m *)
```

**Запись соединения отправителя:**

```
Low : integer ;    (* Подтвержденные слова текущего соединения *)
High : integer ;   (* Принятые слова текущего соединения *)
St : timer ;       (* Таймер соединения *)
```

**Запись соединения приемника:**

```
Exp : integer ;          (* Ожидаемый порядковый номер *)
Rt : timer ;             (* Таймер соединения *)
```

**Подсистема связи:**

```
Mq : channel ;         (* Пакеты данных для q *)
Mp : channel ;         (* Пакеты подтверждения для p *)
```

**Вспомогательные переменные:**

```
B : integer init 0 ;           (* Слова в предыдущем соединении *)
cr : boolean init false ;      (* Существование соединения для приемника *)
cs   : boolean init false ;    (*Существование соединения для отправителя *)
```

Протокол посылает пакеты данных, состоящие из: бита начала последовательности, порядкового номера и слова. Для анализа протокола каждый пакет данных содержит четвертое поле, называемое ***оставшееся время жизни пакета***. Оно показывает максимальное время, в течение которого пакет еще может находиться в канале до того, как он должен быть принят или стать потерянным согласно предположению об ограниченном времени жизни. В момент отправления оставшееся время жизни пакета всегда равно m.

Пакеты подтверждения протокола состоят только из порядкового номера, ожидаемого процессом q, но опять для целей анализа каждое подтверждение содержит оставшееся время жизни пакета.

***Протокол отправителя:***

$A_p$: (* Принятие следующего слова *)
begin if not cs then
begin (* Сначала соединение открывается *)
create (St, High, Low) ; (* cs := true *)
Low := High := 0 ; St := S
end;
Ut[B + High] := U, High := High + 1
end
$S_p$: (* Отправление i-го слова текущего соединения *)
{cs /\ Low £ i < High /\ Ut[B + i] > 0}
begin
send <**data**, (i = Low), i, $in_p$[B + i], m> ;
St := S
end
$R_p$: (* Принятие подтверждения *)
{cs /\ <ack, i, r > Î $M_p$ }
begin receive <ack, i, r> ; Low := max (Low, i) end
$E_p$: (* Генерация сообщения об ошибке для возможно потерянного слова *)
{cs /\ Ut[B + Low] £ – 2m – R}
begin error [B + Low] := true ; Low := Low + 1 end
$C_p$: (* Закрытие соединения *)
{cs /\ St < 0 /\ Low = High }
begin B := B + High , delete (St, High, Low) end
(* cs := false *)

Закрытие соединения контролируется таймерами, таймером St для отправителя и таймером Rt для приемника. Ограниченный интервал посылки каждого слова и ограниченное время жизни пакета приводят к тому, что каждое слово может быть найдено в каналах только лишь в течение интервала времени длиной m + U, начиная с момента принятия слова. Это позволяет приемнику отбрасывать информацию о конкретном слове через m + U единиц времени после принятия слова; после этого не могут появиться дубликаты, следовательно невозможна повторная доставка. Таймер Rt устанавливается в R каждый раз, когда слово доставляется, константа R выбирается так, чтобы удовлетворять неравенству $R \geq U + m$. Если следующее слово принимается в течение R единиц времени, то таймер Rt обновляется, иначе соединение закрывается. Значение таймера отправители выбирается так, чтобы невозможно было принять подтверждение при закрытом соединении; для этого, соединение поддерживается в течение по крайней мере S единиц времени после отправления пакета, где S – константа, выбираемая так, чтобы удовлетворять $S \geq R+2m$.

Таймер St устанавливается в S каждый раз, когда посылается пакет, и соединение может быть закрыто только если $St < 0$. Если к этому времени еще остались незавершенные слова (то есть слова, для которых не было получено подтверждение), эти слова объявляются потерянными до закрытия соединения.

***Протокол приемника:***

```
Rq:          (* Принимаем пакет данных *)
{ <data, s, i, w, r > ∈ Mq }
begin receive <data, s, i, w, r > ;
if cr then
if i = Exp then
begin Rt := R ; Exp := i + 1 ; deliver w end
else if s = true then
begin create (Rt, Exp) ; (* cr := true *)
Rt := R ; Exp := i +1 ; deliver w
end
end
Sq:          (* Посылаем подтверждение *)
{cr}
 begin send <ack, Exp, m > end
(* Закрытие соединения по истечении времени Rt, см. В действии Time *)
```

Бит начала последовательности используется приемником, если пакет получен при закрытом соединении, чтобы решить, может ли быть открыто соединение (и доставлено слово в пакете ). Отправитель устанавливает бит в true, если все предыдущие слова были подтверждены или объявлены (как возможно потерянные). Когда q получает пакет при уже открытом соединении, содержащееся слово доставляется тогда и только тогда, когда порядковый номер пакета равен ожидаемому порядковому номеру (хранится в Exp).

Остается обсудить значение переменной B в протоколе отправителя. Это вспомогательная переменная, введенная только с целью доказательства правильности протокола. Отправитель нумерует слова в каждом соединении, начиная с 0, но, чтобы различать слова в различных соединениях, все слова индексируются последовательно по возрастанию для анализа протокола. Таким образом, там, где отправитель индексирует слово как i, «абсолютный» номер указанного слова B + i, где B – общее количество пакетов, принятых p в предыдущих соединениях. Соответствие между «внутренними» и «абсолютными» номерами слов показывается на рисунке 2. B реализации протокола B не хранится, и отправитель «забывает» все слова $in_p[0 .. B – 1]$.

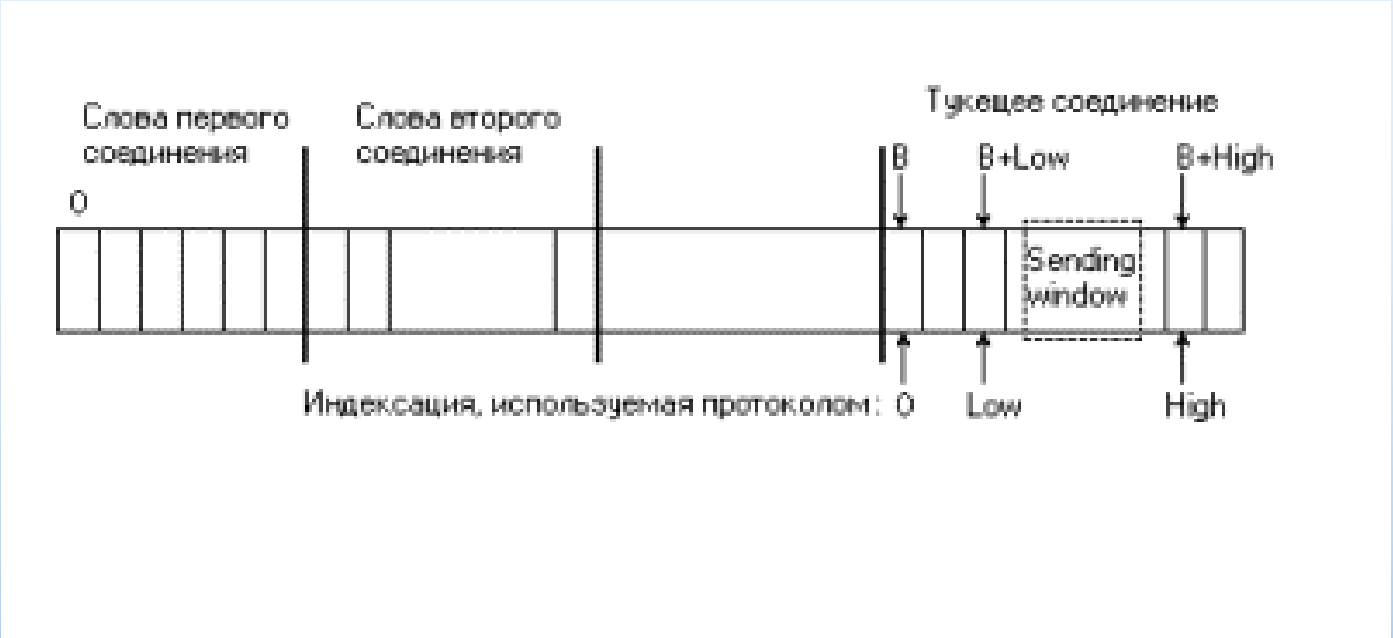


**Рисунок 2 – Порядковые номера протокола**

***Дополнительные переходы протокола:***

Loss: {m ∈ M} (* M – либо $M_p$, либо $M_q$ *)

begin remove m from M end

Dupl: {m ∈ M} (* M – либо $M_p$, либо $M_q$ *)

begin insert m in M end

Time: (* d > 0 *)

begin forall i do Ut[i] := Ut[i] – d ,

St := St – d ; Rt := Rt – d ;

if Rt £ 0 then delete (Rt, Exp) ; (* cr := false *)

forall <.., r> ∈ $M_p$, $M_q$ do

begin r := r – d ;

if r ≤ 0 then remove packet

end

end

Подсистема связи представляется двумя мультимножествами, $M_p$ – для пакетов с адресатом p и $M_q$ – для пакетов с адресатом q. Дополнительные переходы не соответствуют шагам в протоколе процессов. Эти переходы представляют собой отказы канала и изменение времени. В переходах Loss и Dupl M означает или $M_p$, или $M_q$. Действие Time уменьшает все таймеры в системе на величину d, это случается между двумя дискретными событиями, которые отличаются на d единиц времени. Когда таймер приемника достигает значения 0, соединение закрывается.

**Доказательство корректности протокола.** Требуемые свойства протокола будут доказаны в серии лемм и теорем. Утверждение $P_0$, которое определено ниже, показывает, что соединение отправителя остается открытым пока в системе еще есть пакеты, и что порядковые номера этих пакетов имеют корректное значение в текущем соединении.

$P_0 \equiv cs \Rightarrow St \leq S$ (1)

$cr \Rightarrow 0 < Rt \leq R$ (2)

$\forall\ i < B + High : Ut[i] \leq U$ (3)

$\forall\ <... r> \in M_p, M_q : 0 < r \leq m$ (4)

$<\mathbf{data}, s, i, w, r> \in M_q \Rightarrow cs \wedge St \geq r + m + R$ (5)

$cr \Rightarrow cs\ /\backslash\ St \geq Rt + m$ (6)

$<\mathbf{ack}, i, r> \in M_p \Rightarrow cs\ /\backslash\ St > r$ (7)

$<\mathbf{data}, s, i, w, r > \in M_q \Rightarrow (w = in_p[B + i]\ /\backslash\ i < High)$ (8)

Объяснение к (3): значение High предполагается равным нулю во всех конфигурациях, в которых со стороны приемника нет соединения.

**Лемма 7.** $P_0$ – инвариант протокола, основанного на таймере.

*Доказательство.* Первоначально не соединения, нет пакетов, и B = 0, из чего следует, что $P_0$ – true.

$A_p$: (1) справедливо, так как St всегда присваивается значения S (St = S). (3) справедливо, так как перед увеличением High, Ut[B + High] присваивается значение U. (5), (6) и (7) справедливы, так как St может только увеличиваться. (8) справедливо, так как High может только увеличиваться.

$S_p$: (1) справедливо, так как St всегда присваивается значения S. (4) справедливо, так как каждый пакет посылается с оставшимся временем жизни равным m. (5) справедливо, так как пакет <.., m> посылается и St устанавливается в S, и S = R + 2m. (6) и (7) справедливы, так как St может только увеличиться в этом действии. (8) справедливо, так как новый пакет удовлетворяет $w = in_p[B + i]$ и i < High.

$R_p$: Действие $R_p$ не меняет никаких переменных из $P_0$, и удаление пакета справедливо для (4) и (7).

$E_p$: Действие $E_p$ не меняет никаких переменных из P0.

$C_p$: Действие $C_p$ делает ложными выражения (5), (6) и (7). Но выражения (2), (5), (6) и (7) применимы только когда их посылки ложны. $C_p$ также меняет значение B, но, так как пакетов для передачи нет (в соответствии с (5) и (7)), (8) справедливо.

$R_q$: (2) справедливо, так как Rt всегда присваивается значение R (если присваивается). (6) справедливо, так как Rt устанавливается только в R только при принятии пакета <**data**, s, i, w, r>, и из (4) и (5) следует, что $cs \wedge St \geq R + m$, когда это происходит.

$S_q$: (4) справедливо, так как каждый пакет посылается с оставшимся временем жизни, равным m. (7) справедливо, так как пакет <**ack**, i, r> посылается с r = m, когда cr истинно, так что из (2) и (6) $St > m$.

Loss: (4), (5), (7) и (8) справедливы, так как удаление пакета может фальсифицировать только их посылку.

Dupl: (4), (5), (7) и (8) справедливы, так как ввод пакета m применим, только если m уже был в канале, из чего следует, что заключение данного предложения было истинным и перед введением.

Time: (1), (2) и (3) справедливы, так как St, Rt, и Ut[i] могут только уменьшаться, и соединение приемника закрывается, когда Rt становится равным 0. (4) справедливо, так как r может только уменьшиться, и пакет удаляется, когда его r-поле достигает значения 0. Заметим, что Time уменьшает все таймеры (включая r-поле пакета) на одну и ту же величину, значит сохраняет все утверждения вида Xt > Yt + C, где Xt и Yt – таймеры, а C – константа. Это показывает, что (5), (6) и (7) справедливы. □

Первое требование к протоколу заключается в том, что каждое слово в конце концов доставляется или объявляется потерянным. Определим предикат 0k(i) как 0k(i) – error [i] = true $\vee$ q доставил $in_p[i]$.

Можно показать, что протокол не теряет никаких слов, не объявляя об этом. Определим утверждение $P_1$ как

$P_1 \equiv P_0$

$\wedge \neg$ cs $\Rightarrow \forall$ i < B: 0k(i) (9)

$\wedge$ cs $\Rightarrow \forall$ i < B + Low : 0k(i) (10)

$\wedge$ <**data**, true, I, w, r> $\in M_q \Rightarrow \forall$ i < B + I: 0k(i) (11)

$\wedge$ cr $\Rightarrow \forall$ i < B + Exp : 0k(i) (12)

$\wedge$ <**ack**, I, r> $\in M_p \Rightarrow \forall$ i < B + I: 0k(i) (13)

**Лемма 8.** $P_1$ – инвариант протокола, основанного на таймере.

*Доказательство.* Сначала заметим, что как только 0k(i) стало true для некоторого i, он никогда больше не становится false. Сначала нет соединения, нет пакетов, и B = 0, откуда следует, что P1 выполняется.

$A_p$: Действие $A_p$ может открыть соединение, но при этом справедливо (10), так как соединение открывается с Low = 0 и $\forall$ i < B : 0k(i) выполняется согласно (9).

$S_p$: Действие $S_p$ может послать пакет <**data**, s, I, w, r>, но так как s истинно только при I = Low, то справедливо (11) согласно (10).

$R_p$: Значение Low может быть увеличено, если принят пакет <**ack**, I, r>. Тем не менее, (10) справедливо, так как согласно (13) $\forall$ i < B + I : 0k(i) выполняется, если получено это подтверждение.

$E_p$: Значение Low может быть увеличено, когда применяется действие $E_p$, но генерация сообщения об ошибке гарантирует, что (10) справедливо.

$C_p$: Действие $C_p$ обращает cs в false, но оно применимо только если St < 0 и Low == High. Из (10) следует, что $\forall$ i < B + High: 0k(i) выполняется прежде выполнения $C_p$, следовательно (9) справедливо. Посылка (10) обращается в false в этом действии, и из (5), (6) и (7) следует, что посылки (11), (12) и (13) ложны; следовательно (10), (11), (12) и (13) справедливы.

$R_q$: Сначала рассмотрим случай, когда q принимает <**data**, true, l, w, r> при не существующем соединении (cr – false). Тогда $\forall$ i < B + I: 0k(i) из (11), и w доставляется в действии. Так как w = $in_p$[B + I] согласно (8), присваивание Exp := I + 1 делает справедливым (12).

Теперь рассмотрим случай, когда Exp увеличивается в результате принятия <**data**, s, Exp, w, r> при открытом соединении. Согласно (12), ∀ i < B + Exp: 0k(i) выполнялось перед принятием, и слово w = Wp[B + Exp] доставляется действием, следовательно приращение Exp делает справедливым (12).

$S_q$: Отправление <**ack**, Exp, m> делает справедливым (13) из (12).

Loss: Выполнение Loss может только фальсифицировать посылки предложений.

Dupl: Введение пакета m возможно, только если посылка соответствующего предложения (и, следовательно, заключение) была истинна еще до введения.

Time: Таймеры не упоминались явно в (9)–(13). Выведение пакета или закрытие процессом q может только фальсифицировать посылки (11), (12) или (13). □

Теперь может быть доказана первая часть спецификации протокола, но после дополнительного предположения. Без этого предположения отправитель может быть чрезвычайно ленивым в объявлении слов возможно потерянными; в протоколе отправителя указано только, что это сообщение может и не возникнуть в промежуток времени 2m + R после окончания интервала для отправления слова, но не указано, что оно вообще должно появиться. Итак, можно сделать дополнительное предположение, что действие $E_p$ на самом выполнится процессом p в течение ограниченного времени, а именно прежде, чем Ut[B + Low] = –2m – R – 1.

**Теорема 2. (Нет потерь).** Каждое слово $in_p$ доставляется q или объявляется p как возможно потерянное в течение U + 2m + R + l после принятия слова процессом p.

*Доказательство.* После принятия слова $in_p$[I], B + High > I начинает выполняться. Если соединение закрывается в течение указанного периода после принятия слова $in_p$[I], то B > I, и результат следует из (9). Если соединение не закрывается в этот промежуток времени и B + Low ≤ I, отчет обо всех словах из промежутка B + Low..I возможен ко времени 2m + R после окончания интервала отправления $in_p$[I]. Из этого следует, что этот отчет имел место 2m + R + 1 после окончания интервала отправления, то есть, U+ 2m + R + 1 после принятия. Из этого также следует I < B + Low, и, значит, слово было доставлено или объявлено (согласно (10)). □

Чтобы установить второе требование корректности протокола, покажем, что каждое принимаемое слово имеет больший индекс (в $in_p$), чем ранее принятое слово. Обозначим индекс самого последнего доставленного слова через pr (для удобства запишем, что изначально pr = – 1 и Ut[– 1] = $-\infty$ ). Определим утверждение $P_2$ как:

$P_2 \equiv P_1$

$/\backslash$ <**data**, s, i, w, r> $\in M_q \Rightarrow Ut[B + i] > r - m$ (14)

$/\backslash\ i_1 \le i_2 < B + High \Rightarrow Ut[i_1] \le Ut[i_2]$ (15)

$/\backslash\ cr \Rightarrow Rt \ge Ut[pr] + m$ (16)

$/\backslash\ pr < B + High\ /\backslash\ (Ut[pr] > -m \Rightarrow cr)$ (17)

$/\backslash\ cr \Rightarrow B + Exp = pr + 1$ (18)

**Лемма 9.** $P_2$ – инвариант протокола, основанного на таймере.

*Доказательство.* Изначально $M_q$ пусто, B + High равно нулю, $\neg$ cr выполняется, и Ut[pr] < – m, откуда следуют (14)–(18).

$A_p$: (15) справедливо, так как каждое новое принятое слово получает значение таймера U, что согласно (3) по крайней мере равно значениям таймеров ранее принятых слов.

$S_p$: (14) справедливо, так как Ut[B +i] > 0 и пакет отправляется с r = m .

$C_p$: (14), (16) и (18) справедливы, так как согласно (5) и (6) их посылки ложны, когда $C_p$ применимо. (15) справедливо, так как B принимает значение B + High и таймеры не меняются. (17) справедливо, так как B присваивается значение B + High, pr и cr не меняются.

$R_q$: (16) справедливо, так как, когда Rt устанавливается в R (при принятии слова) Ut[pr] ≤ U согласно (3), и R ≥ 2m + U. (17) справедливо, так как pr < B + High, что следует из (8), и cr становится true. (18) справедливо, так как Exp устанавливается в i + 1 и pr в B + i, откуда следует, что (18) становится true.

Time: (14) справедливо, так как Ut[B + i] и r уменьшаются на одно и то же число (и выведение пакета только делает ложной посылку). (15) справедливо, так как $Ut[i_1]$ и $Ut[i_2]$ уменьшаются на одну и туже величину. (16) справедливо, так как cr не становится истинным в этом действии, Rt и Ut[pr] уменьшаются на одну и ту же величину. (17) справедливо, так как его заключение становится ложным только, если Rt становится ≤ 0, откуда следует (согласно (16)), что Ut[pr] становится < – m. (18) справедливо, так как, если cr не обратился в false, B, Exp и pr не меняются.

Действия $R_p$, $E_p$ и $S_q$ не меняют никаких переменных в (14)–(18). Loss и Dupl делают справедливыми (14)–(18) исходя из тех же соображений, что и в предыдущих доказательствах. □

**Лемма 10.** Из $P_2$ следует, что <**data**, s, $i_1$, w, r> $\in M_q \Rightarrow$ (cr $\vee$ B + $i_1$ > pr).

*Доказательство.* Согласно (14), из <**data**, s, $i_1$, w, r> $\in M_q$ следует, что Ut[B+$i_1$] > r – m > – m.

Если B +$i_1$ ≤ pr то, так как pr < B + High согласно (15), Ut[pr] > – m, так что согласно (17) cr true. □

**Теорема 3. (Упорядочение).** Слова, доставляемые q, появляются в строго возрастающем порядке в массиве $in_p$.

*Доказательство.* Предположим q получает пакет <**data**, s, $i_1$, w, r> и доставляет w. Если перед получением не было соединения, B + $i_1$ > pr (согласно Лемме 10), так что слова w располагается в $in_p$ после позиции pr. Если соединение было, $i_1$ = Exp, значит B + $i_1$ = B + Exp = pr +1 согласно (18), откуда следует, что w = $in_p$[pr + 1]. □

**Анализ протокола.**

***Качество протокола.*** Требования *Нет потерь* и *Упорядочение* являются свойствами безопасности и позволяют получить чрезвычайно простое решение, а именно протокол, который не посылает или не получает никакие пакеты и объявляет каждое слово потерянным. Само собой разумеется, что такой протокол, не производящий никакой транспортировки данных от отправителя к приемнику, не является хорошим решением.

Хорошие решения проблемы не только удовлетворяют требованиям *Нет потерь* и *Упорядочение,* но также объявляют потерянными как можно меньше слов. Для этой цели рассматриваемый протокол может быть дополнен механизмом, который посылает каждое слово неоднократно до тех пор, пока не наступит конец интервала посылки или пока не получит подтверждение. Интервал посылки должен быть достаточно длинным, чтобы можно было повторить передачу некоторого слова несколько раз, и чтобы вероятность, что слово потеряется, стала очень маленькой.

На стороне приемника предусмотрен механизм, который вызывает посылку подтверждения всякий раз, когда пакет доставлен или получен при открытом соединении.

***Ограниченные порядковые номера.*** Порядковые номера, используемые в протоколе, могут быть ограничены. Для этого нужно предположить, что скорость принятия слов (процессом p) ограничена следующим образом: слово может быть принято, только если первое из предыдущих слов имеет возраст по крайней мере U + 2m + R единиц времени. Для этого нужно к действию $A_p$ добавить действие {(High < L) V ( Ut[B + High – L] < – R – 2m)}.

Учитывая это предположение, можно показать, что порядковые номера принимаемых пакетов лежат в 2L-окрестности вокруг Exp, и порядковые номера подтверждений – в L-окрестности вокруг High. Следовательно, можно передавать порядковые номера modulo 2L.

***Форма действий и инвариант.*** Благодаря использованию утверждений, рассуждения относительно протокола связи уменьшены до манипулирования формулами. ***Манипулирование формулами*** – «безопасная» методика потому, что каждый шаг может быть проверен очень подробно, так что возможность сделать ошибку в рассуждениях мала. Но есть риск, что можно потерять идею протокола и его отношение к рассматриваемым формулам. Проблемы проектирования протокола могут быть поняты и с прагматической, и с формальной точки зрения.

Управляющая информация должна быть «защищена» в том смысле, что ее значение не должно изменяться потерей или дублированием пакетов; это – прагматическая точка зрения.

При использовании в проверке утверждений «значение» информации управления отражено в формулировании специфических утверждений в качестве инвариантов. Выбор этих инвариантов и проектирование переходов, сохраняющих их, составляет формальную точку зрения.

Все инвариантные предложения $P_2$ относительно пакетов имеют форму $\forall\ m \in M : A(m)$. И в самом деле, легко видеть, что подобное предложение сохраняется при дублировании и потере пакетов.

Инварианты могут быть представлены в более общей форме, например

$$\sum_{m \in M} f(m) = K$$

или условием $\Rightarrow \exists\ m \in M : A(m)$.

Утверждения, имеющие эту форму, могут быть фальсифицированы потерей или дублированием пакетов, и, следовательно, не могут использоваться в доказательстве корректности алгоритмов, которые должны допускать подобные дефекты.

Подобные же наблюдения применимы к форме инвариантов в действии Time. Уже было отмечено, что это действие сохраняет все утверждения формы $Xt \geq Yt + C$, где Xt и Yt – таймеры, а C – константа.

***Неаккуратные таймеры.*** Действие Time моделирует идеальные таймеры, которые уменьшаются точно на d в течение d единиц времени, но на практике таймеры страдают неточностью, называемой ***отклонением***. Это отклонение всегда предполагается e-ограниченным, где e – известная константа. Это означает, что в течение d единиц времени таймер уменьшается на величину d', которая удовлетворяет условию $d / (1 + e) \le d' \le d \prime (1 + e)$ (e имеет порядок $10^{-5}$ или $10^{-6}$). Такое поведение таймеров моделируется действием Time-e, приведенном в следующем алгоритме:

```
Time-e : {d > 0}
begin (* Таймеры в p уменьшаются на d' *)
d' := ... ; (* ≤ d' ≤ d′ (1 + e) *)
forall i do Ut[i] := Ut[i] – d' ;
St := St – d' ;
(* Таймеры в q уменьшаются на d' *)
d" := ...; (* ≤  d" ≤  d′ (1 + e) *)
Rt := Rt – d" ;
if Rt < 0 then delete (Rt, Exp) ;
(* r – поле передается явно *)
forall (.., r ) ∈ Mp, Mq do
begin r := r – d ,
if r < 0 then remove packet
end
end
```

Было замечено, что Time сохраняет утверждения специальной формы $Xt \ge Yt + C$ потому, что таймеры обеих частей неравенства уменьшаются на в точности одинаковую величину, и из $Xt \ge Yt + C$ следует, что $(Xt – d) \ge ( Yt – d) + C$. Такое же наблюдение может быть сделано для Time-e.

Для действительных чисел Xt, Yt, d, d', d", r, и c, удовлетворяющих $d > 0$ и $r > 1$, из $(Xt \ge r^2\ Yt + C) \wedge (\le d' \le d \prime r) \wedge (\le d" \le d \prime r)$ следует $(Xt – d') \ge r^2(Yt – d") + C$.

Следовательно, Time-e сохраняет утверждение формы $Xt \ge (1 + e)^2\ Yt + C$.

Теперь протокол может быть адаптирован к работе с отклоняющимися таймерами, если соответствующим образом изменить инварианты. Для того, чтобы другие действия тоже сохраняли измененные инварианты, константы R и S протокола должны удовлетворять условиям: $R \geq (1 + e)((1 + e)U + (I + e)^2)$ и $S \geq (1 + e)(2m + (1 + e)R)$.

Исключая измененные константы, протокол остается таким же. Его инвариант:

$P_1' = cs \Rightarrow St \leq S$ (1')

$\wedge\ cr \Rightarrow 0 < Rt \leq R$ (2')

$\wedge\ \forall\ i < B + High : Ut[i] < U$ (3')

$\wedge\ \forall\ \langle .., r\rangle \in M_p, M_q : 0 < r \leq m$ (4')

$\wedge\ \langle$**data**, s, i, w, r$\rangle \in M, \Rightarrow cs \wedge St \geq (1 + e)( r + m + (1+ e)R)$ (5')

$\wedge\ cr \Rightarrow cs \wedge St \geq (l + e)((i + e)Rt + m)$ (6')

$\wedge\ \langle$**ack**, i, r$\rangle \in M_p \Rightarrow cs \wedge St > (1 + e)r$ (7')

$\wedge\ \langle$**data**, s, i, w, r$\rangle \in M_q, \Rightarrow (w = in_p[B + i] \wedge i < High)$ (8')

$\wedge \neg cs \Rightarrow \vee\ i < B: 0k(i)$ (9')

$\wedge\ cs \Rightarrow \vee i < B + Low : 0k(i)$ (10')

$\wedge\ \langle$**data**, true, I, w, r$\rangle \in M_q \Rightarrow \forall\ i < B + I: 0k(i)$ (11')

$\wedge\ cr \Rightarrow \forall\ i < B + Exp : 0k(i)$ (12')

$\wedge\ \langle$**ack**, I, r$\rangle \in M_p \Rightarrow \forall\ i < B + I : 0k(i)$ (13')

$\wedge\ \langle$**data**, s, i, w, r$\rangle \in M_q \Rightarrow Ut[B+i] > (l+ e)(r – m)$ (14')

$\wedge\ i_1 \leq i_2 < B + High \Rightarrow Ut[i_1] < Ut[i_2]$ (15')

$\wedge\ cr \Rightarrow Rt \geq (1 + e)((l + e)\ Ut[pr] + (1 + e)^2\ m\ )$ (16')

$\wedge\ pr < B + High \wedge Ut[pr] > – (1+ e)m \Rightarrow cr$ (17')

$\wedge\ cr \Rightarrow B + Exp = pr + 1$ (18')

**Теорема 4.** $P_1'$ – инвариант протокола, основанного на таймере с e-ограниченным отклонением таймера. Протокол удовлетворяет требованиям *Нет потерь* и *Упорядочение*.

# Выводы

*В ходе лекции рассмотрены следующие вопросы:*

*- сбалансированный протокол скользящего окна;*

*- протокол, основанный на таймере.*

## Задание на самостоятельную работу

*1. Конспект лекций.*

## Вид и тема следующего занятия

***Практическое занятие №5. Основы в ASP.NET Core (ч. 5)***