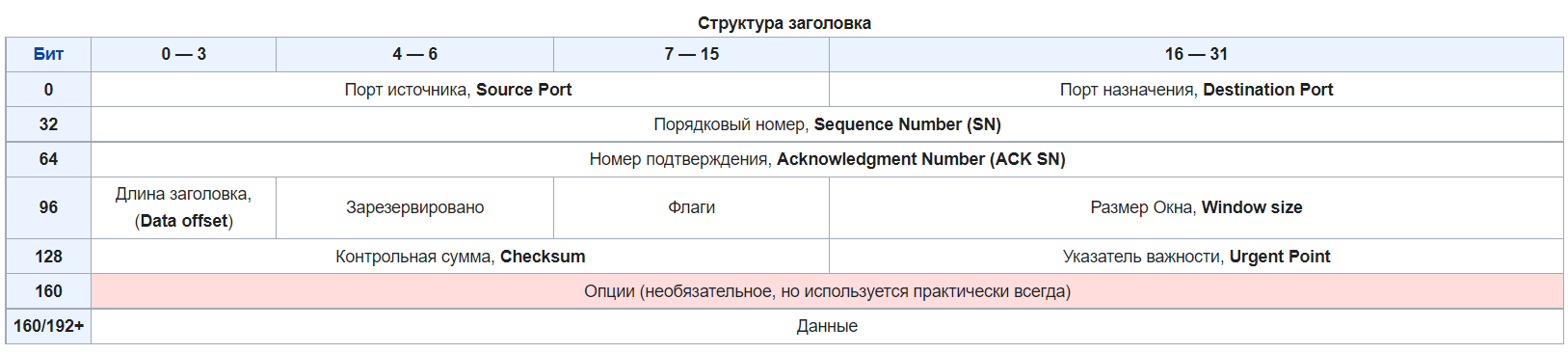
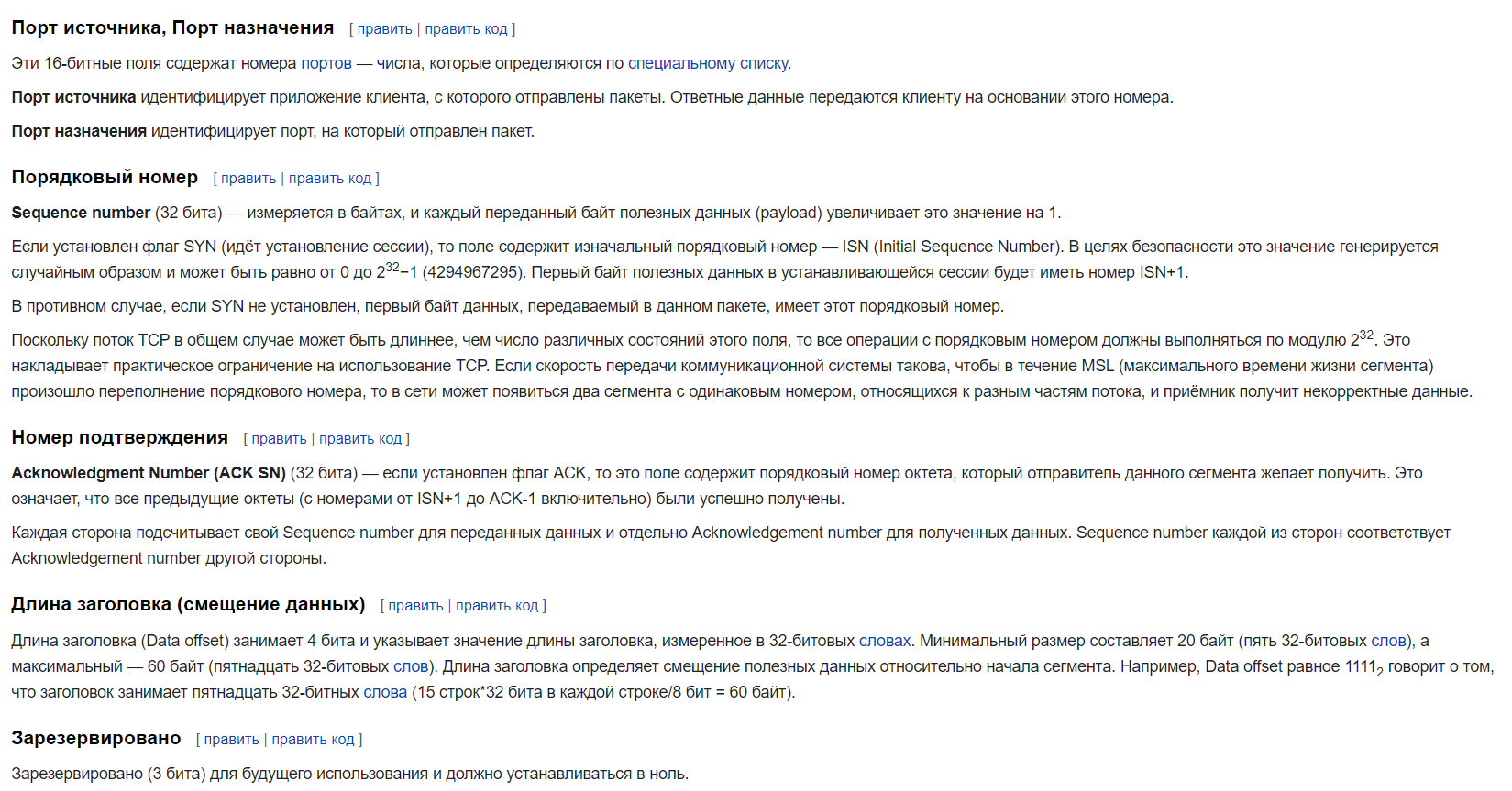
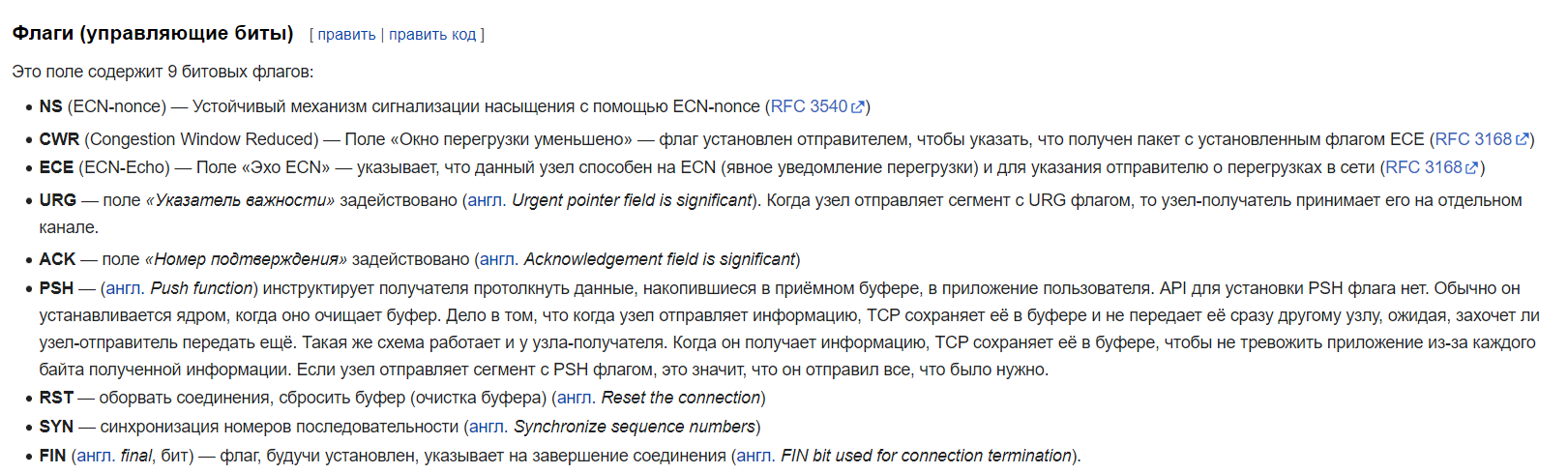
**Transmission Control Protocol** (TCP, протокол управления передачей) — один из основных [протоколов передачи данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) интернета. Предназначен для управления [передачей данных интернета](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Пакеты в TCP называются *сегментами***.**

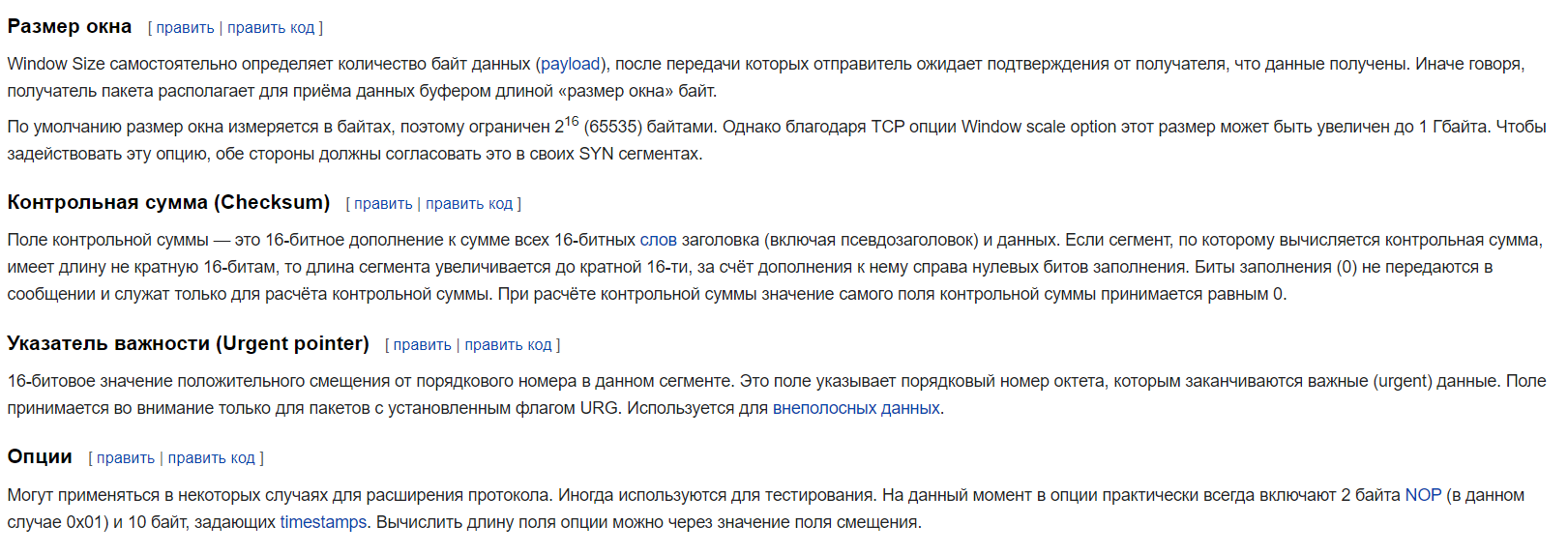
В [стеке протоколов TCP/IP](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82-%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D0%BE%D0%B2_%D1%82%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D0%BE%D0%B3%D0%BE_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D1%8F) выполняет функции [транспортного уровня](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BF%D0%BE%D1%80%D1%82%D0%BD%D1%8B%D0%B9_%D1%83%D1%80%D0%BE%D0%B2%D0%B5%D0%BD%D1%8C) [модели OSI](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_OSI).

Механизм TCP предоставляет [поток данных](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым (в отличие от [UDP](https://ru.wikipedia.org/wiki/UDP)) целостность передаваемых данных и уведомление отправителя о результатах передачи.





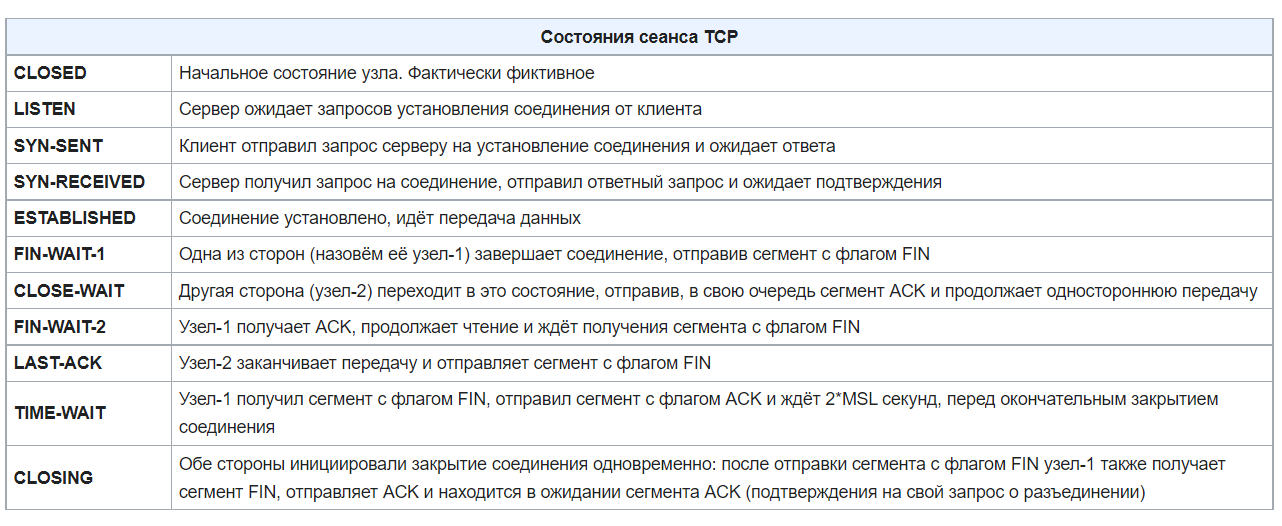


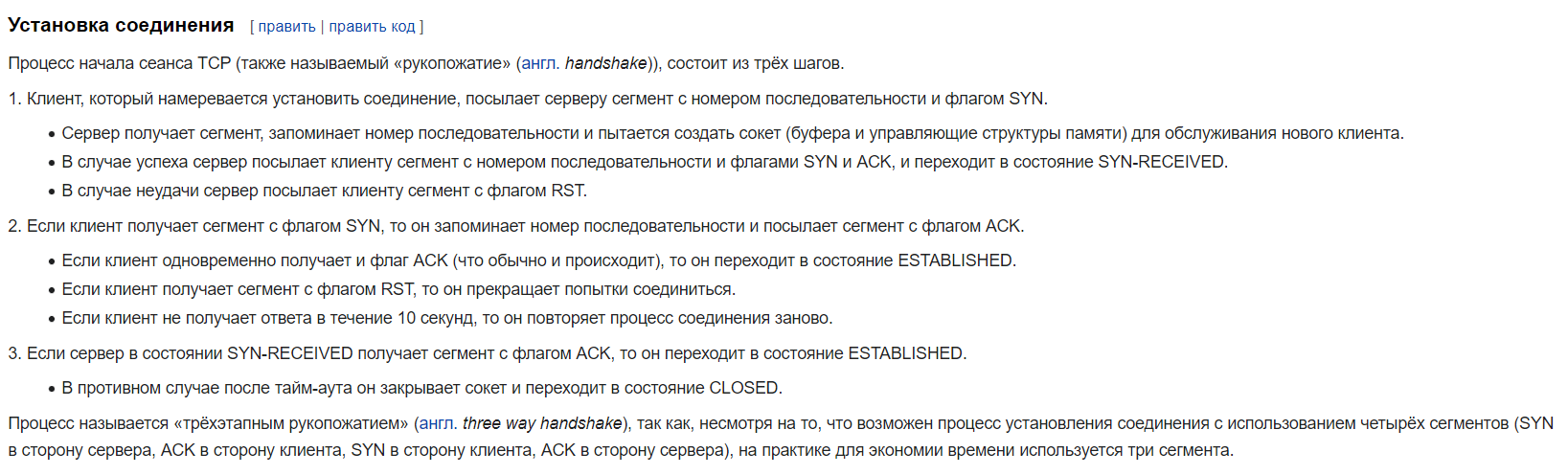


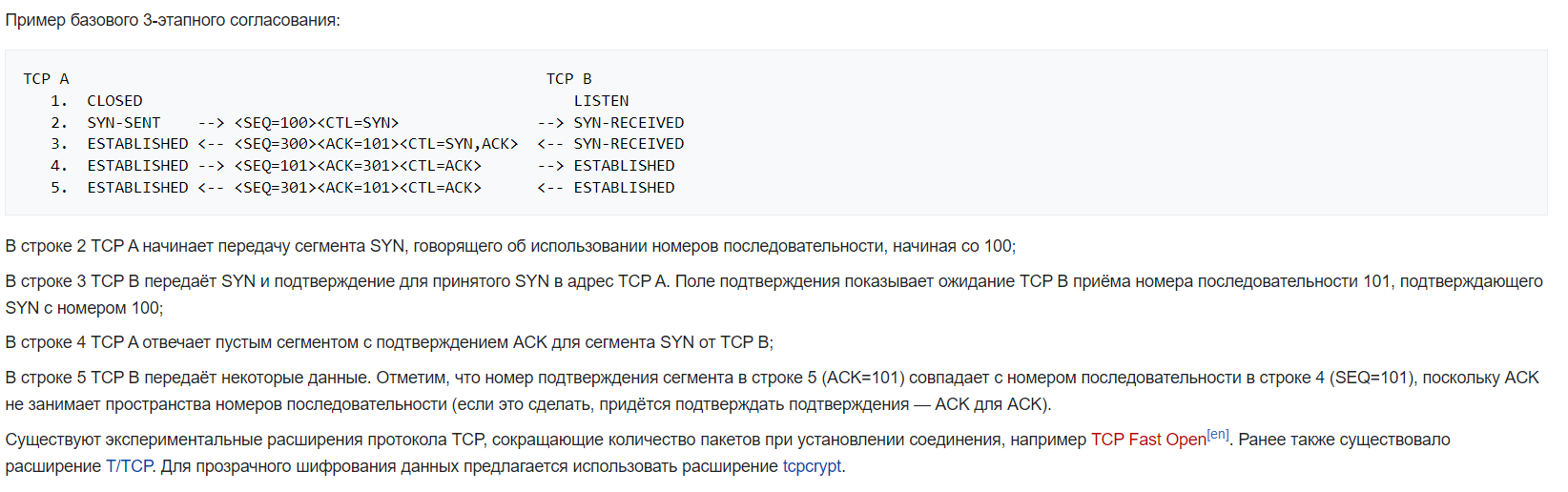
**Механизм действия протокола**

В отличие от традиционной альтернативы — UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP-соединение можно разделить на 3 стадии:

* Установка соединения
* Передача данных
* Завершение соединения





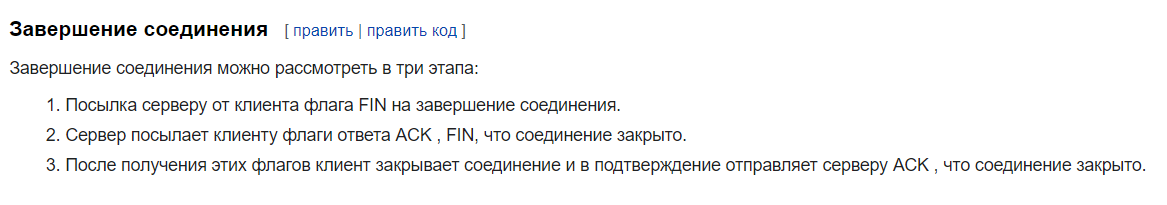


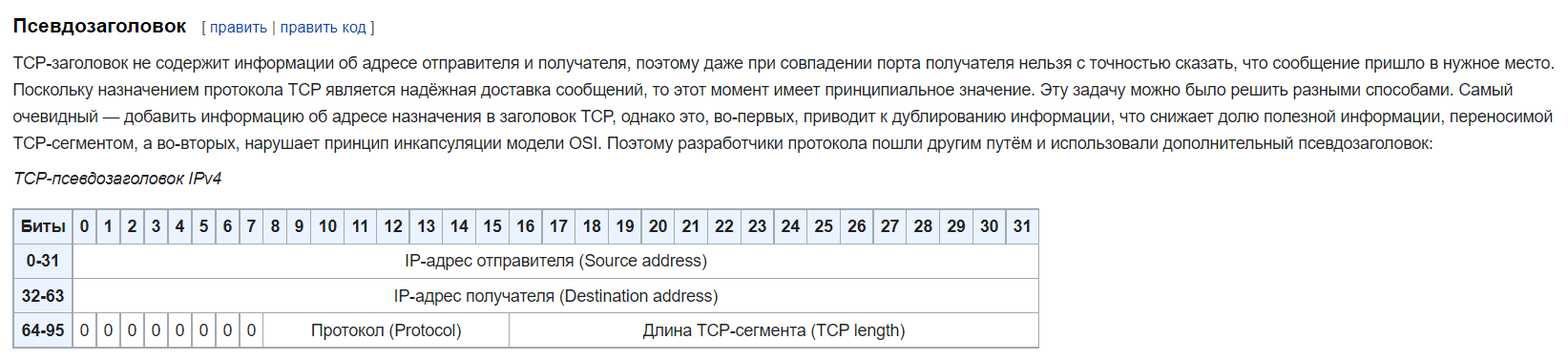
**Передача данных**

При обмене данными приёмник использует номер последовательности, содержащийся в получаемых сегментах, для восстановления их исходного порядка. Приёмник уведомляет передающую сторону о номере последовательности, до которой он успешно получил данные, включая его в поле «номер подтверждения». Все получаемые данные, относящиеся к промежутку подтверждённых последовательностей, игнорируются. Если полученный сегмент содержит номер последовательности больший, чем ожидаемый, то данные из сегмента буферизируются, но номер подтверждённой последовательности не изменяется. Если впоследствии будет принят сегмент, относящийся к ожидаемому номеру последовательности, то порядок данных будет автоматически восстановлен исходя из номеров последовательностей в сегментах.

Для того, чтобы передающая сторона не отправляла данные интенсивнее, чем их может обработать приёмник, TCP содержит средства управления потоком. Для этого используется поле «окно». В сегментах, направляемых от приёмника передающей стороне, в поле «окно» указывается текущий размер приёмного буфера. Передающая сторона сохраняет размер окна и отправляет данных не более, чем указал приёмник. Если приёмник указал нулевой размер окна, то передачи данных в направлении этого узла не происходит, пока приёмник не сообщит о большем размере окна.

В некоторых случаях передающее приложение может явно затребовать передать данные до некоторой последовательности принимающему приложению, не буферизируя их. Для этого используется флаг PSH. Если в полученном сегменте обнаруживается флаг PSH, то реализация TCP отдаёт все буферизированные на текущий момент данные принимающему приложению. «Проталкивание» используется, например, в интерактивных приложениях. В сетевых терминалах нет смысла ожидать ввода пользователя после того, как он закончил набирать команду. Поэтому последний сегмент, содержащий команду, обязан содержать флаг PSH, чтобы приложение на принимающей стороне смогло начать её выполнение.



****

**Cкользящее окно.** В этом случае отправитель передает сразу несколько порций данных не дожидаясь подтверждения. Получатель отправляет одно подтверждение которое называется кумулятивное. Это означает, что получатель получил последнюю порцию данных и все предыдущие.

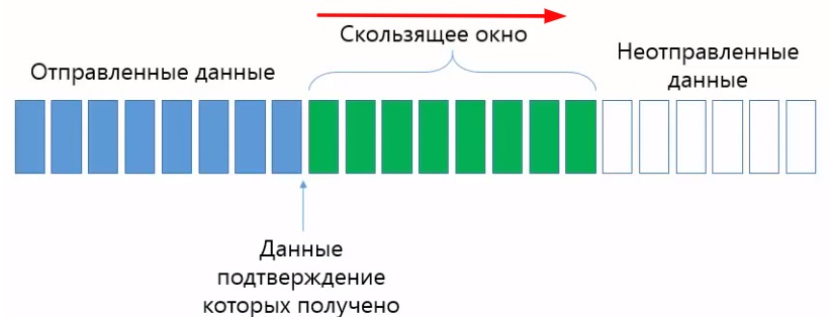
**Скользящее окно**

Почему термин называется скользящее окно? Удобно представлять себе окно, которое скользит по потоку байт получаемых от приложений. У есть поток байт, разделенный на отдельные сегменты, часть сегментов уже передана, часть еще не отправлены. Для некоторых сегментов, которые уже переданы, получено подтверждение. И отправлено некоторое количество сегментов соответствующие размеру окна, для которых подтверждение не получено.



***Размер окна*** — это количество байтов данных, которые могут быть переданы без получения подтверждения.

В примере размер окна 8 сегментов. Что происходит, если мы получили очередное подтверждение? Мы можем передвинуть окно дальше по данным, в него попадает новая порция не отправленных данных. Можно отправить эти данные получателю, после этого отправитель останавливается и дожидаются подтверждения получения следующей порции данных. Таким образом, окно скользит вдоль нашего потока байт от приложения.



**Синдром «Глупого окна»** — это проблема, которая возникает из-за плохой реализации [**TCP**](http://espressocode.top/computer-network-tcp-congestion-control/) . Это снижает производительность TCP и делает передачу данных крайне неэффективной. Проблема называется так, потому что:

1. Это приводит к уменьшению размера окна отправителя до глупого значения.
2. Размер окна уменьшается до такой степени, когда передаваемые данные меньше, чем заголовок TCP.

**Каковы причины?**  
Две основные причины этого синдрома:

1. Окно отправителя повторно передает один байт данных.
2. Окно получателя, принимающее один байт данных несколько раз.

**Причина-1:** Окно отправителя повторно передает один байт данных —  
Предположим, что приложение генерирует только один байт данных. Плохая реализация TCP приводит к передаче этого небольшого сегмента данных. Каждый раз, когда приложение генерирует байт данных, окно передает их. Это делает процесс передачи медленным и неэффективным. Проблема решается алгоритмом Нейгла.

**Алгоритм Нэгла предполагает:**

1. Отправитель должен отправлять только первый байт при получении однобайтовых данных из приложения.
2. Отправитель должен буферизовать все остальные байты, пока не будет подтвержден выдающийся байт.
3. Другими словами, отправитель должен ждать 1 RTT (время прохождения в оба конца).

После получения подтверждения отправитель должен отправить буферизованные данные в одном сегменте TCP. Затем отправитель должен снова буферизовать данные, пока ранее отправленные данные не будут подтверждены.

**Причина-2:** Окно получателя, принимающее один байт данных многократно —  
Предположим, рассмотрим случай, когда получатель не может обработать все входящие данные. В таком случае получатель будет объявлять окно небольшого размера. Процесс продолжается, и размер окна становится все меньше и меньше. Приходит, когда он неоднократно объявляет окно размер 1 байт. Это делает процесс приема медленным и неэффективным. Решением этой проблемы является решение Кларка.

**Решение Кларка предполагает:**

1. Приемник не должен отправлять обновление окна на 1 байт.
2. Приемник должен подождать, пока у него не появится достаточно места.
3. Получатель должен затем сообщить об этом размере окна отправителю.

**Замечания:**

1. Алгоритм Nagle отключен для тех приложений, которым требуется немедленная отправка данных. Алгоритм Nagle может вводить задержку, так как он отправляет только один сегмент данных за одну поездку.
2. И алгоритм Нейгла, и алгоритм Кларка могут работать вместе. Оба дополняют друг друга.

**Алгоритм Нэгла** - это средство повышения эффективности сетей [TCP / IP](https://translated.turbopages.org/proxy_u/en-ru.ru.a18279a0-63addbf6-cc7116eb-74722d776562/https/en.wikipedia.org/wiki/TCP/IP) за счет уменьшения количества пакетов, которые необходимо отправлять по сети.

RFC описывает то, что он назвал "проблемой малых пакетов", когда приложение повторно отправляет данные небольшими порциями, часто размером всего 1 [байт](https://translated.turbopages.org/proxy_u/en-ru.ru.a18279a0-63addbf6-cc7116eb-74722d776562/https/en.wikipedia.org/wiki/Byte). Поскольку [TCP](https://translated.turbopages.org/proxy_u/en-ru.ru.a18279a0-63addbf6-cc7116eb-74722d776562/https/en.wikipedia.org/wiki/Transmission_Control_Protocol)-пакеты имеют 40-байтовый заголовок (20 байт для TCP, 20 байт для [IPv4](https://translated.turbopages.org/proxy_u/en-ru.ru.a18279a0-63addbf6-cc7116eb-74722d776562/https/en.wikipedia.org/wiki/IPv4)), это приводит к 41-байтовому пакету на 1 байт полезной информации, что приводит к огромным накладным расходам. Такая ситуация часто возникает во время сеансов [Telnet](https://translated.turbopages.org/proxy_u/en-ru.ru.a18279a0-63addbf6-cc7116eb-74722d776562/https/en.wikipedia.org/wiki/Telnet), когда большинство нажатий клавиш генерируют один байт данных, которые передаются немедленно. Хуже того, по медленным каналам связи одновременно может проходить много таких пакетов, что потенциально может привести к [коллапсу перегрузки](https://translated.turbopages.org/proxy_u/en-ru.ru.a18279a0-63addbf6-cc7116eb-74722d776562/https/en.wikipedia.org/wiki/Congestion_collapse).

Алгоритм Нэгла работает путем объединения нескольких небольших исходящих сообщений и отправки их всех одновременно. В частности, до тех пор, пока существует отправленный пакет, для которого отправитель не получил подтверждения, отправитель должен продолжать буферизацию своих выходных данных до тех пор, пока не будет получен полный выходной пакет, что позволяет отправлять все выходные данные одновременно.

**Ограничения алгоритма Нейгла**

Алгоритм Nagle можно использовать только с TCP. Другие протоколы, включая UDP, не поддерживают его.

Приложения TCP, требующие быстрого сетевого ответа, такие как телефонные звонки через Интернет или игры-шутеры от первого лица, могут не работать должным образом, если включен Nagle. Задержки, вызванные тем, что алгоритму требуется дополнительное время, чтобы собрать меньшие порции данных, могут вызвать заметную задержку визуально на экране или в цифровом аудиопотоке. Эти приложения обычно отключают Nagle.

Этот алгоритм был изначально разработан в то время, когда компьютерные сети поддерживали гораздо меньшую пропускную способность, чем сегодня. Описанный выше пример был основан на опыте Джона Нейгла в Ford Aerospace в начале 1980-х годов, когда разумные компромиссы в их медленной, сильно загруженной сети дальней связи имели смысл. Сегодня все меньше и больше ситуаций, когда сетевые приложения могут извлечь выгоду из его алгоритма.

**Срочные данные**

Модель пересылки данных приложением предполагает применение упорядоченного потока байтов, следующего к точке назначения. Снова обратившись к примеру интерактивного сеанса, предположим, что пользователь нажал клавишу*attention* (внимание) или*break* (прерывание). Удаленное приложение должно быть способно пропустить мешающие байты и отреагировать на нажатие клавиши как можно скорее.

Механизм*срочных данных* (urgent data) маркирует специальную информацию в сегменте как*срочную.*Этим TCP сообщает своему партнеру, что сегмент содержит срочные данные, и может указать, где они находятся. Партнер должен переслать эту информацию в приложение назначения как можно скорее.

**Медленный старт TCP** - это алгоритм, который уравновешивает скорость сети при подключение. Медленный старт постепенно увеличивает объем передаваемых данных, пока не будет найдена максимальная пропускная способность сети.

**Обзор**

Одним из наиболее распространенных способов оптимизации скорости соединения является   увеличение скорости канала (то есть увеличить пропускную способность).   Однако любая ссылка может стать перегруженной, если устройство пытается отправить слишком много данных. Пересыщение ссылки известно как перегрузка, и это может привести к медленной связи или даже потери данных.

Медленный старт предотвращает перегрузку сети, регулируя количество данных, отправленных по нему. Он согласовывает связь между отправителем и получателем, определяя количество данных, которые могут быть переданы с каждым пакетом, и медленно увеличивает объем данных, пока пропускная способность сети не достигнута. Это обеспечивает передачу как можно большего объема данных без засорения сети.

**Как работает медленный старт TCP**

Медленный старт TCP является одним из первых шагов в процессе контроля перегрузки.   Он уравновешивает объем данных, которые может передать отправитель (известный как окно перегрузки), с объемом данных, которые может принять получатель (известный как окно получателя). Нижнее из этих двух значений становится максимальным объемом данных, которые отправителю разрешено передавать до получения подтверждения от получателя.

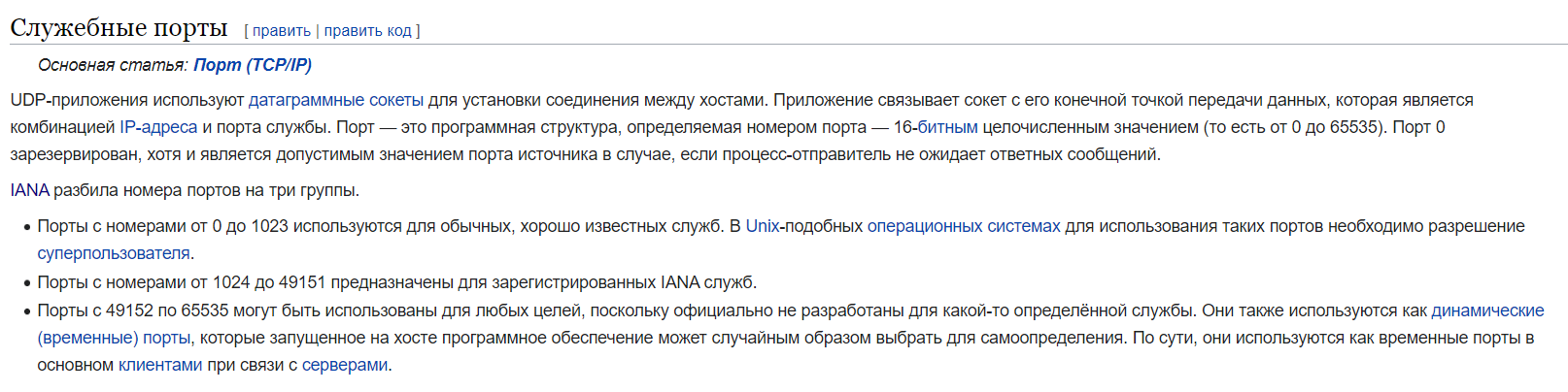
Шаг за шагом, вот как работает медленный старт:

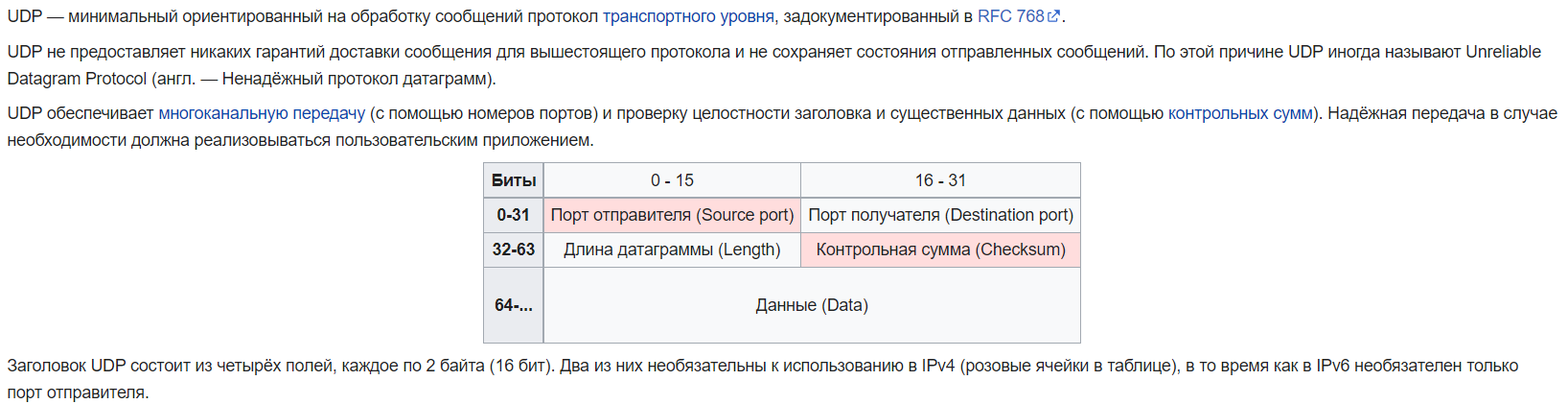
1. Отправитель пытается связаться с получателем. Исходный пакет отправителя содержит небольшое окно перегрузки, которое определяется на основе максимального окна отправителя.;
2. Получатель подтверждает пакет и отвечает своим собственным размером окна. Если получатель не отвечает, отправитель знает, что не нужно продолжать ли отправку данных;
3. После получения подтверждения отправитель увеличивает размер окна следующего пакета. Размер окна постепенно увеличивается до тех пор, пока получатель не сможет больше не подтверждать каждый пакет или пока не будет достигнут предел окна отправителя или получателя.

Как только лимит был определен, работа медленного старта завершена. Другие алгоритмы управления перегрузкой используются для поддержания скорости соединения.

**UDP** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *User Datagram Protocol* — протокол пользовательских [датаграмм](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D1%82%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)) — один из ключевых элементов набора [сетевых протоколов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB) для [Интернета](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). С UDP компьютерные приложения могут посылать сообщения (в данном случае называемые [датаграммами](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D1%82%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)) другим [хостам](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82) по [IP-сети](https://ru.wikipedia.org/wiki/IP) без необходимости предварительного сообщения для установки специальных [каналов передачи](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BD%D0%B0%D0%BB_%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) или путей данных.

UDP использует простую модель передачи, без явных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа, но гарантируется, что если они придут, то в целостном состоянии. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты, что может оказаться невозможным в [системах реального времени](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8). При необходимости исправления ошибок на сетевом уровне интерфейса приложение может задействовать [TCP](https://ru.wikipedia.org/wiki/TCP) или [SCTP](https://ru.wikipedia.org/wiki/SCTP), разработанные для этой цели.





**Порт отправителя**

В этом поле указывается номер порта отправителя. Предполагается, что это значение задаёт порт, на который при необходимости будет посылаться ответ. В противном же случае значение должно быть равным 0. Если хостом-источником является клиент, то номер порта будет, скорее всего, динамическим. Если источником является сервер, то его порт будет одним из «хорошо известных».

**Порт получателя**

Это поле обязательно и содержит порт получателя. Аналогично порту отправителя, если хостом-получателем является клиент, то номер порта динамический, если получатель — сервер, то это будет «хорошо известный» порт.

**Длина датаграммы**

Поле, задающее длину всей датаграммы (заголовка и данных) в байтах. Минимальная длина равна длине заголовка — 8 байт. Теоретически, максимальный размер поля — 65535 байт для UDP-датаграммы (8 байт на заголовок и 65527 на данные). Фактический предел для длины данных при использовании IPv4 — 65507 (помимо 8 байт на UDP-заголовок требуется ещё 20 на IP-заголовок).

На практике также следует учитывать, что если длина IPv4 пакета с UDP будет превышать [MTU](https://ru.wikipedia.org/wiki/Maximum_transmission_unit) (для [Ethernet](https://ru.wikipedia.org/wiki/Ethernet) по умолчанию 1500 байт), то отправка такого пакета может вызвать его фрагментацию, что может привести к тому, что он вообще не сможет быть доставлен, если промежуточные маршрутизаторы или конечный хост не будут поддерживать фрагментированные IP пакеты. Также в [RFC 791](https://tools.ietf.org/html/rfc791) указывается минимальная длина IP пакета 576 байт, которую должны поддерживать все участники IPv4, и рекомендуется отправлять IP пакеты большего размера только в том случае если вы уверены, что принимающая сторона может принять пакеты такого размера. Следовательно, чтобы избежать фрагментации UDP пакетов (и возможной их потери), размер данных в UDP не должен превышать: MTU — (Max IP Header Size) — (UDP Header Size) = 1500 — 60 — 8 = 1432 байт. Для того чтобы быть уверенным, что пакет будет принят любым хостом, размер данных в UDP не должен превышать: (минимальная длина IP пакета) — (Max IP Header Size) — (UDP Header Size) = 576 — 60 — 8 = 508 байт[[2]](https://ru.wikipedia.org/wiki/UDP#cite_note-2).

В Jumbogram’мах IPv6 пакеты UDP могут иметь больший размер. Максимальное значение составляет 4 294 967 295 байт (232 — 1), из которых 8 байт соответствуют заголовку, а остальные 4 294 967 287 байт — данным.

Следует заметить, что большинство современных сетевых устройств отправляют и принимают пакеты IPv4 длиной до 10000 байт без их разделения на отдельные пакеты. Неофициально такие пакеты называют «Jumbo-пакетами», хотя понятие Jumbo официально относится к IPv6. Тем не менее, «Jumbo-пакеты» поддерживают не все устройства и перед организацией связи с помощью UDP/IP IPv4 посылок с длиной, превышающей 1500 байт, нужно проверять возможность такой связи опытным путём на конкретном оборудовании[[3]](https://ru.wikipedia.org/wiki/UDP#cite_note-3).

**Контрольная сумма**

Поле контрольной суммы используется для проверки заголовка и данных на ошибки. Если сумма не сгенерирована передатчиком, то поле заполняется нулями. Поле не является обязательным для IPv4.

**Расчёт контрольной суммы**

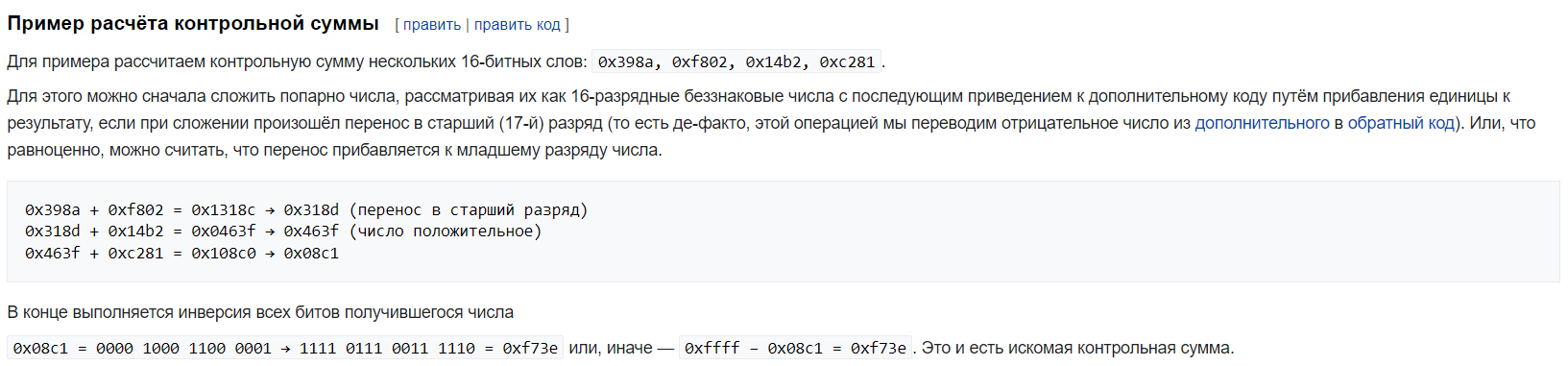
Метод для вычисления контрольной суммы определён в RFC 1071[[4]](https://ru.wikipedia.org/wiki/UDP#cite_note-:0-4).

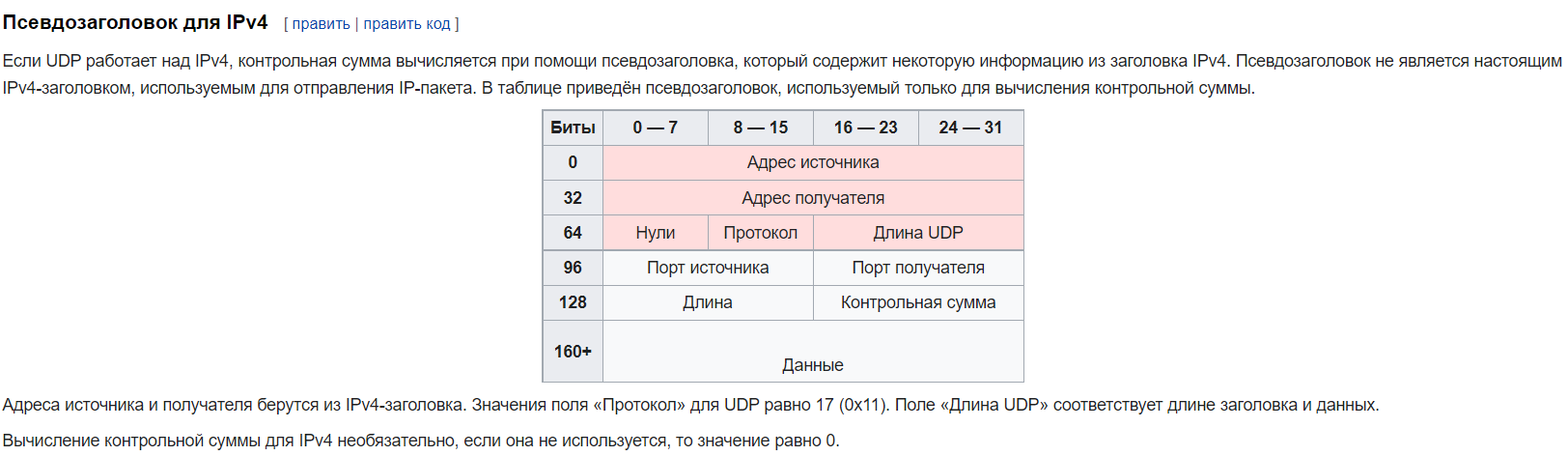
Перед расчётом контрольной суммы, если длина UDP-сообщения в байтах нечётна, то UDP-сообщение дополняется в конце нулевым байтом (псевдозаголовок и добавочный нулевой байт не отправляются вместе с сообщением, они используются только при расчёте контрольной суммы). Поле контрольной суммы в UDP-заголовке во время расчёта контрольной суммы принимается нулевым.

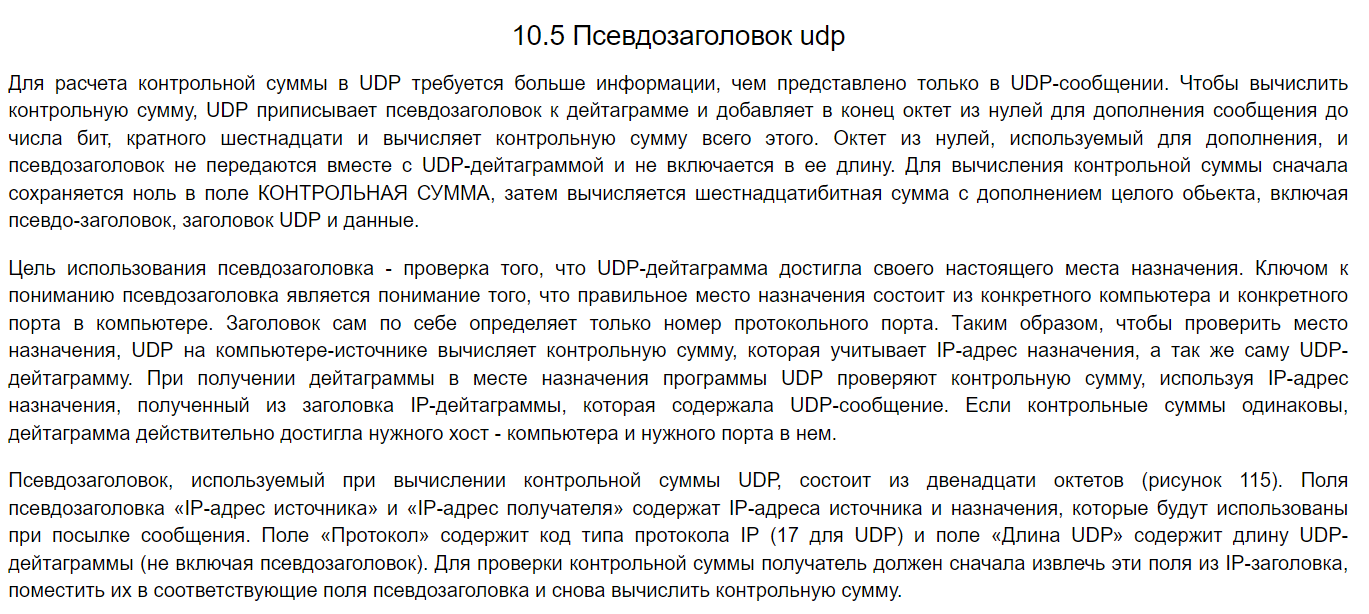
Для расчёта контрольной суммы псевдозаголовок и UDP-сообщение разбивается на двухбайтные слова. Затем рассчитывается сумма всех слов в арифметике [обратного кода](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) (то есть кода, в котором отрицательное число получается из положительного инверсией всех разрядов числа и существует два нуля: 0х0000 (обозначается +0) и 0xffff(обозначается −0)). Результат записывается в соответствующее поле в UDP-заголовке.

Значение контрольной суммы, равное 0х0000 (+0 в [обратном коде](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)), зарезервировано и означает, что для посылки контрольная сумма не вычислялась. В случае, если контрольная сумма вычислялась и получилась равной 0х0000, то в поле контрольной суммы заносят значение 0xffff(-0 в [обратном коде](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4)).

При получении сообщения получатель считает контрольную сумму заново (уже учитывая поле контрольной суммы), и, если в результате получится −0 (то есть 0xffff), то контрольная сумма считается сошедшейся. Если сумма не сходится (данные были повреждены при передаче, либо контрольная сумма неверно посчитана на передающей стороне), то решение о дальнейших действиях принимает принимающая сторона. Как правило, в большинстве современных устройств, работающих с UDP/IP-пакетами имеются настройки, позволяющие либо игнорировать такие пакеты, либо пропускать их на дальнейшую обработку, невзирая на неправильность контрольной суммы.







**Перегрузки UDP**

Более серьёзной потенциальной проблемой является то, что в отличие от TCP, основанные на UDP приложения не обязательно имеют хорошие механизмы контроля и избегания перегрузок. Чувствительные к перегрузкам UDP-приложения, которые потребляют значительную часть доступной пропускной способности, могут поставить под угрозу стабильность в Интернете.

Сетевые механизмы были предназначены для того, чтобы свести к минимуму возможные эффекты от перегрузок при неконтролируемых, высокоскоростных нагрузках. Такие сетевые элементы, как маршрутизаторы, использующие пакетные очереди и техники сброса, часто являются единственным доступным инструментом для замедления избыточного UDP-трафика. [DCCP](https://ru.wikipedia.org/wiki/DCCP) (англ. Datagram Congestion Control Protocol — протокол контроля за перегрузками датаграмм) разработан как частичное решение этой потенциальной проблемы с помощью добавления конечному хосту механизмов для отслеживания перегрузок для высокоскоростных UDP-потоков вроде потоковых медиа.

**Сравнение UDP и TCP**

[TCP](https://ru.wikipedia.org/wiki/TCP) — ориентированный на соединение протокол, что означает необходимость «рукопожатия» для установки соединения между двумя хостами. Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях.

* *Надёжность* — TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших данных, ни (в случае многочисленных тайм-аутов) разорванных соединений.
* *Упорядоченность* — если два сообщения последовательно отправлены, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению.
* *Тяжеловесность* — TCP необходимо три пакета для установки сокет-соединения перед тем, как отправить данные. TCP следит за надёжностью и перегрузками.
* *Потоковость* — данные читаются как поток [байтов](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82), не передается никаких особых обозначений для границ сообщения или сегментов.

UDP — более простой, основанный на сообщениях протокол без установления соединения. Протоколы такого типа не устанавливают выделенного соединения между двумя хостами. Связь достигается путём передачи информации в одном направлении от источника к получателю без проверки готовности или состояния получателя. В приложениях для голосовой связи через интернет-протокол (Voice over IP, TCP/IP) UDP имеет преимущество над TCP, в котором любое «рукопожатие» помешало бы хорошей голосовой связи. В VoIP считается, что конечные пользователи в реальном времени предоставят любое необходимое подтверждение о получении сообщения.

* *Ненадёжный* — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути. Нет таких понятий, как подтверждение, повторная передача, тайм-аут.
* *Неупорядоченность* — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.
* *Легковесность* — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. Это небольшой транспортный уровень, разработанный на IP.
* *Датаграммы* — пакеты посылаются по отдельности и проверяются на целостность только если они прибыли. Пакеты имеют определенные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение целиком, каким оно было изначально послано.
* *Нет контроля перегрузок* — UDP сам по себе не избегает перегрузок. Для приложений с большой пропускной способностью возможно вызвать коллапс перегрузок, если только они не реализуют меры контроля на прикладном уровне.

**Синхронная передача.**

Метод передачи, при котором для управления потоком данных используются тактовые синхросигналы. При синхронной передаче кадры (frame) передаются через равные промежутки времени, причем синхронизм должен жестко контролироваться передающим и принимающим компьютерами. Для начальной синхронизации и контроля за синхронизмом в процессе передачи в поток данных включаются специальные символы, благодаря чему оба взаимодействующих устройства могут обнаруживать и корректировать любые временные отклонения.

**Асинхронная передача.**

Метод передачи данных, при которой интервалы времени между направляемыми блоками данных не являются постоянными (сигнал на линии может появиться в любой момент времени). Для выделения в потоке данных блоков в начале и конце каждого из них записываются старт/стопные биты. При асинхронной передаче передатчик и приемник данных работают не зависимо друг от друга.

**Преимущества асинхронной передачи:**

· несложная, отработанная технология;

· недорогое (по сравнению с синхронным) интерфейсное оборудование;

**Недостатки:** · примерно треть пропускной способности теряется на передачу служебных битов; · при множественной ошибке с помощью бита четности невозможно определить достоверность полученной информации;

·   невысокая (по сравнению с синхронной) скорость передачи.