Санкт-Петербургский государственный университет

Кафедра системного программирования

Клещин Антон Сергеевич

# Реализация транспортного протокола сетевого вычислителя

Курсовая работа

Научный руководитель:
ст. преп. Баклановский М. В.

Санкт-Петербург
2019

# Оглавление

# Введение

Не секрет, что каждый год объём обрабатываемой информации растёт стремительными темпами. В связи с этим серверам всё сложнее справиться с увеличивающейся нагрузкой, а наращивание мощностей обходится всё дороже и дороже. Обслуживание высоконагруженных серверов — также непростая задача, которая требует больших финансовых затрат. Есть задачи, которые решаются на этих серверах (то есть централизованным образом), но выполнение которых можно естественным образом перенести на распределённую вычислительную систему. Такой системой может являться совокупность вычислительных устройств организации, используемых не в полную силу.

Наиболее существенную выгоду от перехода на распределённую систему могут получить независимые последовательные задачи. Их работа зависит только либо от данных, полученных в результате выполнения самой программы, либо от данных, которые были созданы заранее в результате выполнения других программ и уже не будут меняться. Первый тип данных переносится от устройства к устройству непосредственно вместе с процессом исполнения задачи. Второй тип данных может быть доступен из некоторого хранилища данных, которое тоже может быть распределённым. Таким образом, вопрос согласованности данных решается автоматически. В случае, если следующий узел для исполняемой задачи недоступен, текущей узел может сохранить пересылаемые данные и повторить попытку позже.

Несмотря на то, что существуют многие peer-to-peer системы, все они для взаимодействия входящих в них программ используют принцип соединения точка-точка. С ростом таких систем связи между приложениями становятся всё запутаннее. Также отсутствует учёт маршрута данных, который мог бы привести к более оптимальным использованиям ресурсов всей системы, а значит и повышению быстродействия. Поэтому создание протокола, который учитывает маршруты целиком и убирает необходимость заботы о них с приложений, обрабатывающих данные, является актуальной задачей.

## 0.1. Сетевой вычислитель

Сетевой вычислитель — это распределённая одноранговая система, которая нацелена на выполнение множества независимых последовательных задач в соответствии с их логикой движения между устройствами сети (так называемый маршрут задачи).

Рассмотрим эту общую концепцию на примере. Пусть есть задача — подписание документа, который должен быть рассмотрен в строгой очерёдности некоторой группой людей. В клиент-серверной архитектуре это после прохождения очередного этапа документ отправляется на выделенный сервер, который решает, куда дальше его отправить. Таким образом возникает большое количество накладных расходов, которых можно избежать с помощью сетевого вычислителя. А именно: документ после рассмотрения на очередном узле сети отправляется непосредственно на следующее устройство, ассоциированное со следующим человеком из группы. Видно, что удаётся избежать дополнительной нагрузки на узлы, которые не участвуют в логике движения этого документа.

# 1. Постановка задачи

Целью данной работы является разработка и реализация транспортного протокола сетевого вычислителя для Linux. Для её достижения были выделены следующие задачи:

- Обзор существующих подходов;
- Разработка протокола;
- Разработка модуля ядра, реализующего транспортный протокол;
- Тестирование с целью проверки функциональности.

# 2. Изучение предметной области

Разрабатываемый протокол должен определять правила маршрутизации при получении данных от приложений, правила пересылки данных в надёжных ненагруженных сетях, правила обмена служебной информацией. Протокол должен быть бинарным. Поэтому в данной главе будут рассмотрены принципы работы нескольких самых популярных сетевых протоколов, которые частично удовлетворяют данным требованиям.

## 2.1. UDP

UDP[5] — User Datagram Protocol — протокол транспортного уровня в стеке TCP/IP, предполагает, что на сетевом уровне используется IP. Формат его заголовка состоит всего из четырёх полей: порт отправителя, порт получателя, длина и контрольная сумма. Используется для передачи данных между приложениями в местах, где скорость пересылки пакетов важнее их целостности.

После установки адреса получателя, пакет отправляется сразу, без установки предварительного соединения. Этот протокол не заботится о порядке доставки данных, не устраняет дублирования, нет защиты от потерь пакетов при пересылке: считается, что подобные ошибки не так часты, а попытка их исправить слишком дорога. Несмотря на наличие поля с контрольной суммой, часто она не проверяется в силу всё тех же причин. Таким образом, вся ответственность за целостность данных перекладывается на приложения прикладного уровня.

Главной целью UDP является высокая скорость доставки данных между двумя программами, которая достигается за счёт минимальных гарантий. Также протокол не хранит какого-либо состояния, что положительно влияет на скорость работы и на нагрузку устройств при большом количестве подключений.

## 2.2. TCP

TCP[8] — Transmission Control Protocol — протокол транспортного уровня в стеке TCP/IP, предполагает, что на сетевом уровне используется IP. Используется для надёжной передачи данных.

Процесс отправки пакета начинается с установки соединения посредством обмена тремя сообщениями (так называемое тройное рукопожатие). Штатное закрытие соединения проходит по аналогичной схеме с точностью до выставленных битов поля флагов. Отправитель каждому пакету выставляет свой номер, являющийся номером первого байта этого пакета в сообщении. Получатель всегда ожидает пакет с номером первого отсутствующего пакета. Если приходят данные с меньшим номером, то они отбрасываются, если с большим, то буферизируются. При получении пакета с ожидаемым номером порядок данных восстанавливается в соответствии с номерами их пакетов, а затем отправляется подтверждение в виде нового значения номера ожидаемого пакета. Таким образом, TCP гарантирует как порядок доставки данных, так и саму доставку как таковую. Эффективность работы протокола главным образом зависит от выбранного алгоритма получения подтверждений. Также для предоставления заявленных гарантий протокол сохраняет некоторое состояние для каждого соединения, что негативно влияет на загруженность серверов при большом количестве подключений.

Основной целью TCP является обеспечение надёжной передачи данных между двумя программами.

## 2.3. DCCP

DCCP[2] — Datagram Congestion Control Protocol — протокол транспортного уровня в стеке TCP/IP, предполагает, что на сетевом уровне используется IP. Создавался, как аналог UDP с контролем плотности трафика.

Процесс передачи пакетов состоит из трёх фаз — инициирование соединения посредством тройного рукопожатия, передача данных и разрыв соединения, также используя тройное рукопожатие. Как и UPD,

не предоставляет гарантий при доставке пакетов, однако использует систему подтверждений, которые позволяют отправителю определить количество теряемых данных и предотвратить возникновение перегрузок в сети. Сам механизм контроля насыщенности может выбираться отправляющей стороной.

DCCP предоставляет возможность выбора между величиной задержки передачи пакетов и надежной доставкой с соблюдением порядка в зависимости от загруженности сети.

## 2.4. SCTP

SCTP[4] — Stream Control Transmission Protocol — протокол транспортного уровня в стеке TCP/IP, предполагает, что на сетевом уровне используется IP. Используется для надёжной передачи данных. Создавался, как замена TCP, решая многие его проблемы и добавляя новую функциональность.

В отличии от TCP, процесс установки соединения состоит из обмена четырьмя сообщениями, а не тремя. Сервер выделяет ресурсы для соединения только при отправке последнего сообщения, а не после получения первого же, как в TCP. Так решается проблема DDoS атак посредством отправки множества запросов на создание соединения.

Непосредственно сама передача данных происходит аналогично TCP за тем лишь исключением, что пакету присваивается теперь его порядковый номер, а отправитель не ожидает какого-то конкретного пакета и отправляет подтверждение в виде массива из всех полученных и пропущенных номеров пакетов. Таким образом, повторно переданы будут только те пакеты, которые действительно потерялись. Также SCTP знает границы сообщений, а значит передаёт данные обработчику ровно сообщениями, в то время как TCP работает с последовательностью байт и передаёт данные приложению по обнаружении пакета со специальным флагом вплоть до актуализированного номера ожидаемого пакета, а значит может передавать обработчику данные не кратные одному сообщению.

Также SCTP поддерживает многопоточность[1] (то есть создание нескольких независимых потоков передачи данных внутри одного соединения) и многодоменность хостов[2] (то есть хосты могут иметь несколько сетевых интерфейсов, а значит и несколько IP-адресов).

## 2.5. IP

IP[6, 1] — Internet Protocol — протокол сетевого уровня стека TCP/IP. Используется для маршрутизации и передачи пакетов между устройствами сети.

Протокол предоставляет три основных функции: адресацию, фрагментирование и сборку пакетов. Маршрутизация выполняется на маршрутизаторах следующим образом: проверяется, принадлежит ли IP-адрес какой-либо локально сети, которая подключена непосредственно к маршрутизатору. Если да, то в неё отправляется пакет с помощью MAC-адреса на канальном уровне. Если нет такой сети, но он знает, какому следующему маршрутизатору из этих сетей его передать, то он делает это. Иначе пакет отбрасывается. IP не имеет каких-либо механизмов надёжности и порядка доставки данных. Имеет ряд опциональных расширений, которые привносят в работу протокола дополнительные функции, например: безопасность, частичное указание отправителем маршрута следования пакета, запись маршрута (для трассировки пути) и другие.

## 2.6. ICMP

ICMP[7, 3] — Internet Control Message Protocol — протокол сетевого уровня стека TCP/IP, используется для отправки служебных сообщений (в основном об исключительных ситуациях). Общий формат заголовка состоит всего из трёх полей: типа сообщения, кода сообщения, который зависит от типа, и контрольной суммы, но при некоторых значениях типа пакета добавляются дополнительные специфичные поля.

[1] multi-streaming

[2] multihoming

## 2.7. Итоги

При учёте маршрутов сетевых приложений становится возможным производить различные виды оптимизаций. Например, можно контролировать доставку сообщений не на уровне отдельных пакетов и не между каждыми двумя устройствами, а на уровне прохождения всем процессом некоторого участка пути. Это требует логики, отличной от TCP и SCTP. Но также нельзя полностью забывать про какие-либо гарантии, как это делает UDP и DCCP.

В плане маршрутизации между двумя устройствами IP полностью подходит под выставленные требования, поэтому в данной работе разрабатываемый протокол будет располагаться на транспортном уровне. Однако возможна реализация протокола многих точек через поля опций (options) IP.

Существует необходимость отправки служебных сообщений. Она возникает, например, при контроле над безопасностью и системой хранения, который может быть одной из функций сетевого вычислителя. Эффективнее будет отвести несколько полей в существующем протоколе, а не создавать новый, как это сделано с IP и ICMP.

# 3. Архитектура протокола

## 3.1. Идентификация сетевых программ

Маршрут сетевой программы — это ориентированный граф с циклами и некоторой выделенной стартовой вершиной. Каждую вершину этого графа назовём состоянием сетевой программы. Разветвления и циклы могут возникать из-за того, что последующий сценарий исполнения может зависеть от результатов обработки данных. На каждом этапе выполнения сетевого процесса должна быть возможность отличить его от других программ, исполняемых на текущем устройстве, от других состояний этой программы, от очередной итерации прохождения по циклу, а также от другого запущенного процесса, который проходит по тому же самому пути и использует те же обработчики (другими словами — от другого экземпляра той же сетевой программы). Для поддержания этой функциональности в протоколе есть следующие поля:

- program_id — идентификатор сетевого процесса;
- state — идентификатор текущего состояния;
- loop_counter — счётчик, который увеличивается каждый раз, когда программа проходит по ребру, конец которого имеет state меньший или равный, чем начало;
- id — аналог значения pid процессов в многозадачных операционных системах. Позволяет различать экземпляры программ. Для простоты имеет размер 8 байт и генерируется случайным образом.

Таким образом, эти четыре поля полностью идентифицируют сетевую программу в любой момент времени.

Объясним выбор размера поля id. Чем больше его размер, тем меньше шанс коллизии. При использовании генератора случайных чисел размера 8 байт с равномерным распределением получаем вероятность

коллизии $2^{-64}$, что примерно равно $5 * 10^{-20}$ и достаточно мало для реальных задач. При размере поля больше 8 байт, оно перестанет помещается в базовые регистры 64-битных процессоров, а значит потребует на обработку больше времени.

## 3.2. Служебные сообщения

Помимо передачи данных сетевых процессов, устройства должны иметь возможность обмениваться служебной информацией. В IP для этих целей ввели дополнительный протокол — ICMP. Его концепция позволяет в точности решить данную задачу, поэтому в разрабатываемый протокол были добавлены аналогичные поля type и code. Однако было решено не выносить их в отдельный протокол, так как сетевой вычислитель должен быть интегрирован с системами хранения и безопасности, которые также требуют информацию об исполняемом процессе. Таким образом, резервирование за ними отдельного значения поля type решает и эту проблему.

## 3.3. Отладка

Отладка сетевых приложений — отдельный сложный вопрос, который ещй предстоит решить. Для его поддержки, а так же для отладки самого протокола было добавлено опциональное поле debug_server, которое появляется при выставлении соответствующего бита поля flags. Он содержит в себе id устройства во внутренней идентификации, на которое автоматически отправляется вся информация обо всех действиях, связанных с выполнением сетевого процесса.

## 3.4. Доставка пакетов

Во время передачи данных бывают ситуации, когда пакеты теряются. Существует разные подходы к решению данной проблемы: UDP, например, игнорирует её, а TCP борется путём отправки многочисленных

подтверждающий сообщений. С другой стороны, UDP работает неоспоримо быстрее TCP в надёжных незагруженных каналах. Учитывая тот факт, что сети с каждым годом становятся всё надёжнее, хочется получить скорость выше, чем у TCP, но также и иметь гарантии того, что данные не потеряются. Для этого была введена система контрольных точек.

Когда сетевая программа проходит через некоторое состояние, обозначенное контрольным, устройство обменивается тремя сообщениями с предыдущей контрольной точкой, тем самым беря обязанность сохранности данных на себя, сохраняя внутреннее состояние процесса. Сам этот процесс схож идеологически с тройным рукопожатием:

- Новая контрольная точка отправляет сообщение о том, что исполнение программы её достигло;
- Старая контрольная точка отправляет подтверждение о принятии этого сообщения;
- Новая контрольная точка отправляет подтверждение на подтверждение.

Для того, чтобы определить, кто же был последней контрольной точкой, в протокол добавлено поле last_checkpoint_id, которое содержит id устройства во внутренней идентификации.

После того, как сетевой процесс прошёл через контрольную точку, она запускает таймер, в течение которого должно прийти уведомление со следующего контрольного узла. Если по истечении данного времени подобного сообщения не получено, то данные считаются потерянными и отправляются заново. Данных подход применим, когда процесс обработки данных не занимает много времени. В частном случае, если расставить контрольные точки на каждом состоянии сетевой программы, то получится принцип работы TCP, за исключением подтверждений о принятии пакетов внутри пересылки одного сообщения. Для ненадёжных сетей предоставляется возможность добавить алгоритмы TCP для подтверждения каждого пакета в сообщении.

# 4. Особенности реализации

В Linux транспортные протоколы должны быть встроены в ядро системы. Для быстрого добавления новой функциональности в систему есть специальный механизм, который называется модуль ядра — код, который может быть загружен и выгружен во время работы системы без требования её перезагрузки.

## 4.1. Внутреннее представление маршрутов

Данные о маршрутах программы хранятся в памяти ядра и представляются в виде таблицы Handlers, строки которой — program_id, а столбцы — state (это те самые поля, объявленные в формате заголовка протокола). Ячейкой данных является структура следующего вида:

- handler — функция-обработчик данных, расположенная в модуле ядра, если данные могут быть обработаны непосредственно в ядре. Если нет, то ставится обработчик по умолчанию;
- handler_type — тип обработчика в пользовательском пространстве, нужен для обработчика по умолчанию, чтобы знать, какому приложению отдать пришедшие данные;
- таблица, которая отображает коды возврата в пару — новое состояние и id следующего устройства.

## 4.2. Обработка данных

В модуле ядра есть таблица, которая отображает тип обработчика в количество зарегистрированных обработчиков и очередь сообщений, предназначенных для них.

Общение ядра и обработчиков в пользовательском пространстве происходит посредством сокетов. При создании сокета указывается тип обработчика, который сохраняется в структуре сокета, а также происходит его регистрация путём увеличения соответствующего счётчика в таблице. При закрытии сокета этот счётчик уменьшается.

### 4.2.1. Получение сообщения обработчиком

Когда обработчик в пользовательском пространстве пытается прочитать данные из сокета, происходит один из следующих случаев:

- данные в очереди для этого типа обработчиков есть, тогда они перемещаются в структуру сокета вместе с заголовком, а затем отправляются приложению. Сохранение заголовка сообщения в сокете нужно для дальнейшей отправки данных после обработки;

- данных в очереди нет, тогда процесс блокируется либо до их появления, либо до истечения некоторого времени, после которого функция, читающая данные из сокета, возвращает соответствующий код ошибки. Это сделано для того, чтобы через несколько попыток приложение-обработчик могло остановить свою работу и тем самым перестать занимать ресурсы системы. В дальнейшем его работа может быть возобновлена по желанию модуля ядра.

### 4.2.2. Отправка сообщения обработчиком

Когда обработчик в пользовательском пространстве выполнил необходимые действия над данными, он, если нужно, посылает их дальше по маршруту сетевой программы, отправляя их в сокет, откуда исходное сообщение было прочитано. Вместо параметра флагов функции отправки указывается код возврата. Затем модуль ядра изымает из структуры сокета предыдущее сообщение и использует его заголовок для нового пакета, изменяя лишь состояние, полученное с помощью кода возврата из таблицы Handlers, и наконец отправляет сформированный пакет на следующее устройство.

### 4.2.3. Получение сообщения ядром

Когда устройству приходит новое сообщение, по полям program_id и state из его заголовка находится запись в таблице Handlers и пакет отдаётся указанному там обработчику. Если этой функцией оказался

обработчик по умолчанию, то он добавляет пришедшие данные в очередь, соответствующую типу обработчика (handler_type) из найденной записи таблицы. Если при этом оказывается, что зарегистрированных обработчиков для этого типа нет, то модуль ядра запускает один из известных подходящих.

Несмотря на то, что новые процессы можно создавать из ядра, было выделено отдельное приложение прикладного уровня, которому эти обязанности делегируются. Также оно может выполнять остальные действия, нежелательные для выполнения в модуле ядра. Для этой программы добавлена отдельная очередь сообщений.

# 5. Тестирование

Тестирование проходило на локальном компьютере, для того чтобы исключить влияние среды на результаты. Вместо реальных устройств выступали две виртуальные машины в VirtualBox с операционными системами Arch Linux с ядром 5.1.2-arch1-1-ARCH. Виртуальные машины могли обмениваться данными посредством виртуального сетевого адаптера, также предоставляемого VirtualBox. Для прохождения маршрута требовалось совершить 2000 передачи пакетов. Каждый тест включал в себя прохождение 100 маршрутов. Оценки были сделаны для пакетов с малым (40 байт), средним (1300 байт) и большим (63 Кбайт) количеством полезных данных (то есть без учёта заголовков). Максимальный размер всего пакета в современных сетях считается равным 1500 байт. Дисперсия выражена в мс$^2$, а остальные приведённые данные записаны в мс.

| тип теста | минимум | максимум | среднее | дисперсия |
|---|---|---|---|---|
| TCP, малый | 560 | 633 | 581 | 115 |
| TCP, средний | 575 | 652 | 602 | 197 |
| TCP, большой | 837 | 921 | 860 | 179 |
| Новый, малый | 1 | 2.7 | 1.2 | 0.08 |
| Новый, средний | 1 | 2.9 | 1.2 | 0.08 |
| Новый, большой | 3.9 | 12.9 | 7.3 | 2.5 |

Как видно из таблицы, в надёжных сетях получилось достичь существенного увеличения производительности.

# Заключение

В ходе работы были достигнуты следующие результаты:

- Проведено исследование предметной области;
- Разработан протокол;
- Разработан реализующий транспортный протокол модуль ядра[3];
- Проведено тестирование с целью проверки функциональности.

В дальнейшем планируется провести верификацию протокола, а также расширить его возможности.

[3] `https://github.com/uncerso/MPP`

# Список литературы


[1] RFC 2460: IPv6. — https://tools.ietf.org/html/rfc2460. — 1998. — Декабрь. — Доступ 15.06.2019.

[2] RFC 4340: DCCP. — https://tools.ietf.org/html/rfc4340. — 2006. — Март. — Доступ 15.06.2019.

[3] RFC 4443: ICMPv6. — https://tools.ietf.org/html/rfc4443. — 2006. — Март. — Доступ 15.06.2019.

[4] RFC 4960: SCTP. — https://tools.ietf.org/html/rfc4960. — 2007. — Сентябрь. — Доступ 15.06.2019.

[5] RFC 768: UDP. — https://tools.ietf.org/html/rfc768. — 1980. — Август. — Доступ 15.06.2019.

[6] RFC 791: IP. — https://tools.ietf.org/html/rfc791. — 1981. — Сентябрь. — Доступ 15.06.2019.

[7] RFC 792: ICMP. — https://tools.ietf.org/html/rfc792. — 1981. — Сентябрь. — Доступ 15.06.2019.

[8] RFC 793: TCP. — https://tools.ietf.org/html/rfc793. — 1981. — Сентябрь. — Доступ 15.06.2019.