Санкт-Петербургский Политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Отчёт по курсовой работе

Реализация TCP-клиента для протокола Tox

по предмету "Сети и телекоммуникации"

Выполнил студент гр. 43501/3
Намаконов Егор
Преподаватель
Зозуля А.В.

18 апреля 2017 г.

Содержание

1. Назначение и особенности протокола

Tox — открытый протокол для текстовой, голосовой и видеосвязи в интернете. Отличительная особенность - полная децентрализация и шифрование всего трафика. Эталонная реализация включает следующие функции:

- голосовая и видеосвязь
- режим конференции с несколькими участниками
- указание и смена сетевого статуса
- поддержка эмотиконов
- демонстрация экрана
- возможность отправлять мгновенные сообщения и передавать файлы
- и т.д.

Позиционируется как открытая, свободная, лишённая бэкдоров и не шпионящая за пользователями альтернатива Skype. Так как протокол является открытым, а узлы сети организуются всеми желающими, реклама отсутствует.

Одна из особенностей протокола - 32-битные публичные ключи, которые генерируются локально у каждого участника и которые служат для идентификации во всей сети. После установки Tox автоматически создаётся пара ключей. Публичный ключ можно передавать кому угодно — он служит как уникальный идентификатор для поиска собеседника. Секретный ключ хранится только у владельца и подтверждает его подлинность не раскрывая персональные данные. Центральный сервер отсутствует, поиск собеседников происходит через DHT.

Криптографические функции выполняются с помощью библиотеки NaCl.

2. Алгоритмы работы протокола

2.1. Организация сети

Сеть участников Tox образует т.н. **DHT** - Distributed Hash Table. Хеш-таблица может представлять собой ассоциативный массив, содержащий пары (ключ-значение). Особенностью распределённой таблицы является возможность распределить информацию среди некоторого набора узлов-хранителей таким образом, что каждый участвующий узел смог бы найти значение, ассоциированное с данным ключом. Ответственность за поддержание связи между именем и значением распределяется между узлами, в силу чего изменение набора участников является причиной минимального количества разрывов. Это позволяет легко масштабировать DHT, а также постоянно отслеживать добавление и удаление узлов и ошибки в их работе.[1]

Хэш-таблица Tox содержит зашифрованные сетевые адреса участников, которые расшифровываются только на двух конечных узлах, которые желают установить соединение друг с другом. Таким образом, Tox предполагает, что сетевой адрес узла нельзя узнать, не установив соединение с ним.

Для того, чтобы подключиться к сети, участник посылает запрос к одному из существующих узлов, запрашивает у него список других клиентов, к которым можно подключиться и продолжает этот процесс, расширяя список известных участников. Обычно для начала этого процесса необходим **bootstrap-узел** . Это узел, к которому клиент подключается, если не знает других участников. Очевидно, что bootstrap-узлы должны работать стабильно и иметь известные адреса. Актуальный список bootstrap-узлов находится на https://nodes.tox.chat/.

2.2. Соединение между узлами

Для создания соединения между 2 узлами (**friend connection**) используются **friend requests** . Для его отправки необходимы **Tox ID** участников, которые имеют следующую структуру:

1. 32 байта - публичный ключ
2. 4 байта - **nospam** - дополнительный идентификатор, назначение которого - гарантировать, что friend request может отправить только тот, кто видел Tox ID целиком, а не только публичный ключ. Nospam можно сменить при необходимости
3. 2 байта - контрольная сумма, которая образуется XORом всех последовательных пар байт Tox ID ($0, 1 \oplus 2, 3 \oplus 4, 5 ... \oplus 34, 35$).

После установления соединения узлы могут начать обмен сообщениями.

2.3. TCP-соединения

По умолчанию Tox использует UDP для обмена данными. Однако если участник находится в локальной сети за NAT, то такой трафик может быть заблокирован. Поэтому при работе из локальной сети участник должен пользоваться **TCP relays** . Эти узлы позволяют подключаться к сети тем участникам, которые не могут общаться напрямую.

Такой сервер ведёт себя как посредник между двумя узлами. После подключения к серверу клиент указывает, к какому собеседнику он хочет подключиться. Соединение будет установлено только тогда, когда оба участника отправят такие запросы. Это позволяет избежать проверок на то, подключён ли тот или иной узел к сети.

[1] https://en.wikipedia.org/wiki/Distributed_hash_table

2.4. Процедура подключения к TCP-серверу

Наиболее абстрактно последовательность действий представляется так:

1. Клиент, зная публичный ключ сервера, отправляет тому handshake-пакет
2. Сервер отвечает handshake-ответом
3. Клиент отправляет любой пакет протокола, чтобы сервер убедился в корректности установления соединения
4. Сервер обрабатывает соответствующий пакет (ответ на пинг и т.д.)

Весь трафик при обмене шифруется. При этом ключи, которые в дальнейшем будут использоваться при обмене данными с сервером в рамках этого соединения, не равны постоянным ключам клиента и сервера - они генерируются на время соединения.

Handshake-запрос имеет следующую структуру:

1. 32 байта - постоянный публичный ключ клиента, который он анонсирует серверу и всей сети
2. 24 байта - временный т.н. **nonce** - одноразовый код, который выбирается в данном пакете случайно и используется для шифрования-дешифрования сообщений вместе с ключом. В дальнейшем для его генерации будут использоваться различные алгоритмы
3. зашифрованный с помощью публичного ключа сервера, приватного ключа клиента и nonce фрагмент (длина в зашифрованном виде - 72 байта):
   (a) 32 байта - временный публичный ключ, который будет использоваться для шифрования данных, отправляемых клиенту
   (b) 24 байта - базовый Nonce, который будет использоваться вместе с временным ключом для шифрования

Handshake-ответ имеет следующую структуру:

1. 24 байта - nonce
2. зашифрованный с помощью публичного ключа клиента, приватного ключа сервера и nonce фрагмент (длина в зашифрованном виде - 72 байта):
   (a) 32 байта - временный публичный ключ, который будет использоваться для шифрования данных, отправляемых серверу
   (b) 24 байта - базовый Nonce, который будет использоваться вместе с временным ключом для шифрования

Публичный ключ сервера в обмене не указывается, т.к. предполагается, что клиенту он известен вместе с сетевым адресом сервера.

Далее клиент должен отправить любой пакет протокола, чтобы сервер убедился в корректности установления соединения. Обычно отправляется ping-запрос, на который сервер даёт ping-ответ.

2.5. Обмен данными с сервером

После handshake шифрование производится симметричным алгоритмом с использованием секретного ключа. Он вычисляется на основе временных ключей данного соединения и одинаков для обеих сторон.

При этом nonce различается для каждого пакета, отправленного одной из сторон. Он вычисляется так: base nonce (отправленный в handshake соответствующей стороной), представленный как 24-битное беззнаковое целое + номер текущего сообщения в данном направлении (начиная с 0 и не считая handshake). Документация в данном случае не даёт чёткого объяснения, какой из nonce соответствует направлению обмена. Экспериментально установлено, что nonce устанавливает сторона, которая будет отправлять пакеты в данном направлении, т.е., например, клиент будет использовать base nonce, который он отправил в handshake request.

Зашифрованные пакеты выглядят в TCP-потоке так:

1. 2 байта - длина зашифрованного сообщения
2. зашифрованное сообщение соответствующей длины (до 2048, согласно документации)

2.6. Установление соединения между клиентами посредством TCP-сервера

При установлении соединения между клиентами из разных локальных сетей с помощью TCP-сервера происходит следущее:

1. клиенты подключаются к серверу
2. клиенты отправляют запросы на установление соединения друг с другом, используя публичные ключи. При этом сервер отправляет им номер подключения, который соответствует паре этих клиентов
3. после получения обоих запросов сервер отправляет клиентам уведомление о том, что данное подключение активно
4. после отключения одного из клиентов сервер уведомляет другого о деактивации соединения

Пакет для установления соединения с другим клиентом - Routing Request и Response; уведомления о статусе соединения - Connect и Disconnect Notification. Эти пакеты имеют простую структуру и хорошо описаны в спецификации. Отметим лишь следующие моменты, относящиеся к Routing Response:

- в ответе дублируется публичный ключ целевого клиента, к которому производится попытка подключения. Это позволяет отправить сразу несколько запросов на подключение к разным клиентам и обработать ответы на них по отдельности
- Единственная причина (на настоящий момент) не создать номер подключения - превышение максимального количества одновременных подключений к серверу. В этом случае вместо номера подключения отправляется 0

2.7. Обмен данными между клиентами с использованием TCP-сервера

Этот аспект в рамках курсовой работы 2017 года изучен не был. По-видимому, реализуется следующий алгоритм[2]:

[2] https://habrahabr.ru/post/217289/

1) Алиса пользуется Tox на соединении которое допускает только TCP подключения, она генерирует временные публичные ключи и подключается к узлам для инициализации.

2) После получения данных о сети с узлов инициализации, она выбирает некоторое количество случайных узлов которые являются супер-нодами

3) Она, используя луковичный роутинг через TCP супер-ноду может отправлять запросы на авторизацию или сообщить своему контакт-списку о своём он-лайн статусе, используя временный публичный ключ.

4) Боб получает пакет через луковичный роутинг от Алисы, которая сообщает ему, к каким узлам она подключена по DHT с использованием временного публичного ключа.

5) Боб подключается к тем же узлам что и Алиса.

6) Данное соединение используется для передачи как сообщений так и A/V трафика

7) Если какой-либо узел отключается, Боб и Алиса переходят на любой другой узел, к которому они оба подключены.

Супер нода — это такой узел, у которого имеется внешний IP адрес и проброшены порты Tox.

В приведённой ниже реализации клиента обмен производится с помощью **OOB** -пакетов (Out Of Band). Их можно рассматривать как технические пакеты, которые отправляются по указанному публичному ключу. Номер соединения в этом случае не указывается, однако, если соединение между клиентами на сервере не установлено, этот пакет будет отброшен. Адресату доставляется OOB-ответ, в котором указан публичный ключ отправителя.

3. Реализация TCP-клиента

Для демонстрации алгоритмов работы протокола реализован клиент на Java, который позволяет подключиться к указанному bootstrap-узлу и установить соединение с таким же клиентом. Обмен данными между ними производится с помощью OOB-пакетов.

Репозиторий с исходным кодом находится на Github: https://github.com/fresheed/tox-client.

3.1. Интерфейс пользователя

Программу запускает класс ControlConsole. В параметрах командной строки ему передаётся путь до конфигурационного файла с примерным содержанием:

```
CLIENT_PRIV_KEY=a590f856c785b225035fbe0ace13810fc2047dd02debc87ad4e6d17403967e5e

SERVER_PUB_KEY=53737F6D47FA6BD2808F378E339AF45BF86F39B64E79D6D491C53A1D522E7039
SERVER_HOST=d4rk4.ru
SERVER_PORT=1813
```

Данный класс создаёт объекты, обозначающие локальный и удалённый узлы, создаёт соединение между ними, а затем в цикле считывает пользовательский ввод и выполняет требуемые операции:

- connect PUBLIC_KEY - отправляет серверу запрос на соединение с указанным узлом. В ответном выводе указывается номер соединения
- oob CONNECTION_ID MESSAGE - отправляет по указанному соединению сообщение
- quit - выход

3.2. Работа с TCP

Использование конкретного канала данных (в данном случае - TCP) скрыто за интерфейсом DataChannel с основными методами send и receive. При чтении контролируется число считываемых байт.

Также TCP-сервер представлен подклассом RemotePeer, который обозначает абстрактный удалённый узел. RemotePeer предоставляет метод получения DataChannel, который в данном случае является TCP-соединением по соответствующему IP и порту.

3.3. Криптография

Для выполнения криптографических функций используется биндинг NaCl к Java - Kalium.

Также используются вспомогательные классы, упрощающие получение nonce и шифрование-дешифрование сообщений между конкретными узлами.

3.4. Узлы и соединения

Узлы представляются интерфейсами LocalPeer и RemotePeer. Основное отличие - LocalPeer позволяет получить приватный ключ.

Класс ToxRelayedConnection инкапсулирует логику обмена сообщениями между узлами. Для инициации соединения используется статический метод connect, в котором производится обмен handshake-пакетами. Далее создаётся объект ToxRelayedConnection, в который передаются полученные временные ключи. Полученный объект используется для шифрования входящих и исходящих сообщений.

3.5. Сообщения

В настоящий момент реализованы следующие типы сообщений:

- ping
- routing
- connect/disconnect notifications
- OOB
- handshake

Для унификации интерфейса используются классы ToxIncomingMessage и ToxOutgoingMessage. Основной метод для первого - accept(), который позволяет обработать его. Для этого применяется паттерн Visitor: объект, обрабатывающий сообщение, вызывает на нём accept, передавая себя как аргумент, а объект-сообщение вызывает один из методов visitMessageType. Кроме того, объекты этого класса должны иметь конструктор, принимающий массив байт, чтобы сообщение можно было разобрать при получении.

Для исходящего сообщения основным методом является getContent(), возвращающий набор байт для дальнейшей отправки.

Также существуют классы ToxDataType, которые отражают заданные в спецификации типы данных. В настоящее время используются только числовые типы - 8,16,64-битные числа.

4. Задание для студентов

Необходимо реализовать алгоритм, описанный в разделе "Обмен данными между клиентами с использованием TCP-сервера":

1. разобраться с логикой отправки и обработки onion-пакетов и выбора TCP-серверов для установления соединения
2. подключившись к супер-нодам, осуществить отправку friend request произвольному узлу (использовать в качестве получателя запроса любой GUI-клиент)
3. реализовать отправку текстовых сообщений (тип MESSAGE) узлам-друзьям

После решения этой задачи можно приступить к менее приоритетным:

- реализация других типов сообщений (как для передачи данных, так и статусных - например, "собеседник печатает")
- повышение тестового покрытия
- реализация GUI

5. Ссылки на ресурсы

- https://toktok.ltd/spec - спецификация протокола
- https://github.com/TokTok/c-toxcore - ядро протокола, написанное на C
- https://github.com/TokTok/qTox - один из популярных клиентов
- https://wiki.tox.chat - Wiki проекта
- https://habrahabr.ru/post/217289/ - описание последовательности действий для установления соединения
- irc://irc.freenode.net/#toktok, https://www.reddit.com/r/projecttox/ - сообщества пользователей и разработчиков Tox.

  В частности, https://www.reddit.com/r/projecttox/comments/64e6xv - вопрос, заданный автором курсовой работы 2017 года по поводу алгоритма установления соединения
- https://github.com/fresheed/tox-client - код Java-клиента