Содержание

Введение

В настоящее время системы обмена мгновенными сообщениями (Instant Messaging Systems, IM-системы) приобрели огромную популярность. Обмен мгновенными сообщениями является основным способом общения для многих пользователей сети Интернет. IM-системы используются уже не только для частной переписки, но и для ведения бизнеса: для проведения деловых переговоров, заключения сделок, общения с заказчиками, внутрикорпоративного общения и т.д. Можно смело утверждать, что современный бизнес, особенно в сфере информационных технологий, в значительной степени зависит от этого вида коммуникаций. Соответственно, все более значащую роль начинает играть надежность систем обмена мгновенными сообщениями. Популярные в настоящее время IM-системы в большинстве своем являются централизованными, т.е. зависят от одного или нескольких центральных серверов, обычно контролируемых одной организацией. Из такого способа организации следует ряд недостатков. Самый главный недостаток - это зависимость системы от центрального сервера. При выведении его из строя прекращается функционирование всей системы. Поэтому к серверу предъявляются высокие требования в плане надежности и отказоустойчивости программного и аппаратного обеспечения, обеспечения защиты от хакерских атак. Функционирование централизованной системы также может быть легко прекращено административными методами, например, по указанию правительства, путем изъятия сервера. Таким образом, сервер любой централизованной системы является самым уязвимым местом. Еще одним недостатком централизованности является зависимость пользователей системы от политики контролирующего систему субъекта (организации или физического лица). Контролирующий субъект может по своему усмотрению отключать отдельных пользователей, ограничивать их коммуникационные возможности, использовать их персональные данные и содержимое переписки в своих целях. Ярким примером являются недавние отключения альтернативных клиентов системы ICQ, осуществленные компанией AOL.

Разрабатываемая в данной работе система является распределенной

и децентрализованной и, благодаря своим организационным принципам, не может контролироваться какой-либо организацией или группой лиц. Функционирование системы обеспечивается за счет множества серверов-узлов, связанных между собой и обслуживающих пользователей. Пользователи системы не привязаны к какому-либо определенному узлу. Если какой-то узел прекращает работу, пользователь будет автоматически соединен с другим доступным узлом. Другая важная характеристика IM-систем – защищенность от перехвата содержимого сообщений. Защита передаваемой информации осуществляется путем шифрования. Многие популярные в настоящее время IM-системы, например ICQ, передают информацию в открытом виде. Защищенность содержимого переписки определяется не только криптостойкостью применяемых алгоритмов шифрования, но и способом реализации шифрования, а также организацией обмена ключами шифрования. Достаточно распространена такая схема шифрования, когда сообщение шифруются отправителем одним ключом и передается на сервер, затем сервер расшифровывает сообщение, зашифровывает его другим ключом и пересылает получателю. В частности, такая схема используется в IM-системах, основанных на протоколе Jabber. Недостатком данной схемы является возможность доступа к содержимому сообщений, проходящих через сервер, субъектом, контролирующим данный сервер. В разрабатываемой системе шифрование выполняется в канале клиент-клиент. Цель данной работы – спроектировать и разработать распределенную защищенную систему обмена мгновенными сообщениями. Надежность и отказоустойчивость системы должна обеспечиваться за счет наличия большого количества независимых серверов-узлов, осуществляющих функции авторизации пользователей, поиска пользователей и пересылки данных. Выход узлов из строя не должен нарушать функционирования системы в целом, система должна функционировать, пока действует хотя бы один узел.

1 Обзор существующих систем обмена мгновенными сообщениями

В настоящее время на принципах распределенности и децентрализованности построены многие распространенные файлообменные сети. Однако в сфере коммуникаций преобладают централизованные системы. Из относительно популярных распределенных и децентрализованных систем можно выделить Skype и Jabber. Также можно упомянуть DAPIM (Distributed, Anonymous and Private Instant Messenger) – разработку студентов из университета Кент, Великобритания [1]. Данная разработка весьма близка по принципам построения разрабатываемой в данной работе системе, однако она, по всей видимости, является незавершенной, т.к. никаких дальнейших упоминаний о ней в сети Интернет обнаружить не удалось.

1.1 Обзор Skype

Система Skype преимущественно предназначена для совершения звонков через Интернет. По принципу организации взаимосвязей между пользователями среди распространенных систем Skype наиболее близка к рассматриваемой в данной работе системе. Skype является закрытой разработкой, поэтому ни спецификации протокола, ни принципы взаимодействия узлов сети, ни исходный код не являются публично доступными. Общие принципы организации взаимодействия компонентов сети Skype известны благодаря независимым исследованиям, в частности [2]. Рассмотрим ключевые моменты. Каждый клиент Skype может соединяться со своим собеседником напрямую либо посредством т.н супер-узлов (super nodes). Супер-узлом может стать любой клиент Skype, имеющий публичный IP-адрес в сети Интернет, достаточную пропускную способность канала связи и достаточные вычислительные ресурсы. Проблема в том, что это происходит без ведома пользователя. Если его клиентское приложение перешло в режим супер-узла, то оно потребляет большие объемы сетевого трафика, т.к. пропускает через себя звонки большого количества других пользователей. Регистрация но-

вых пользователей и авторизация зарегистрированных осуществляется через единственный сервер авторизации. Вследствие этого Skype нельзя считать полноценной децентрализованной системой. Прекращение работы сервера авторизации ведет к прекращению работы всей сети Skype. Такой случай уже имел место 16 августа 2007 года [3]. Skype нельзя считать защищенной системой в плане обеспечения приватности передаваемой информации, т.к. программное обеспечение Skype является закрытым, что не гарантирует отсутствие тайных закладок в программном коде. В частности, правоохранительные органы Австрии сообщали об успешном прослушивании разговоров Skype [4]. Компания Skype отказалась от официальных комментариев по этому поводу.

1.2 Обзор Jabber

В отличие от Skype, Jabber является открытой системой, спецификации протокола Jabber – XMPP доступны всем желающим. Также существует множество открытых реализаций серверов и клиентов Jabber. Распределенные сети на основе Jabber функционируют путем взаимодействия многих Jabber-серверов. Принцип функционирования сетей и адресации пользователей Jabber во многом соответствует принципам электронной почты (e-mail). Однако из такой схемы вытекает серьезный недостаток, заключающийся в том, что учетные записи пользователей привязаны к определенному серверу. При выходе этого сервера из строя функционирование всей системы не будет нарушено, но пользователи, хранящие свои учетные записи на данном сервере, не смогут выйти в сеть. Поэтому Jabber не может считаться полноценной распределенной и децентрализованной системой.

1.3 Обзор DAPIM

Система DAPIM задумана как децентрализованная сеть распределенных узлов, осуществляющих анонимную маршрутизацию сообщений. В системе применяется шифрование. Каждый клиент является узлом сети и может осуществлять пересылку сообщений для других узлов. При запуске кли-

ентское приложение считывает список адресов узлов DAPIM и осуществляет попытку подключения к ним. При успешном подключении приложение получает актуальный список активных узлов, однако список содержит не более 100 записей, т.е. введено искусственное ограничение. Каких-либо конкретных принципов авторизации пользователей в описании обнаружено не было. Прототип клиентского приложения DAPIM разработан на языке Java.

2 Задача разработки распределенной защищенной системы обмена мгновенными сообщениями

2.1 Цель и назначение системы

Назначение системы – осуществлять надежный и защищенный обмен текстовыми сообщениями и иными видами информации в реальном времени. Цель – создание в сети Интернет децентрализованной распределенной системы обмена сообщениями, которая исключает возможность перехвата передаваемой по сети информации третьими лицами. Функционирование системы обеспечивается множеством серверов-узлов, взаимодействующих между собой. Пользователи не должны зависеть от какого-то определенного узла. Выход узлов из строя не должен нарушать работоспособность сети, система должна функционировать, пока работает хотя бы один узел. Система должна быть устойчива к попыткам прекращения ее функционирования путем выведения из строя отдельных узлов системы либо иными способами, возможности по ограничению доступа пользователей к системе должны быть максимально затруднены технически.

2.2 Архитектура системы

Программное обеспечение системы должно состоять из серверной и клиентской части. Клиентская часть (клиент) является приложением, которое содержит графический интерфейс пользователя и обеспечивает подключение к системе, авторизацию в системе, ввод отправляемых сообщений, отображение полученных сообщений, хранение контактов собеседников, хранение списка адресов серверов-узлов системы и поддержание данного списка в актуальном состоянии. Сообщения должны шифроваться перед отправкой и дешифроваться при получении. Шифрование и дешифрование сообщений должно выполняться исключительно в клиентских приложениях. Следует разделить функции графического интерфейса, хранения контактов, настроек

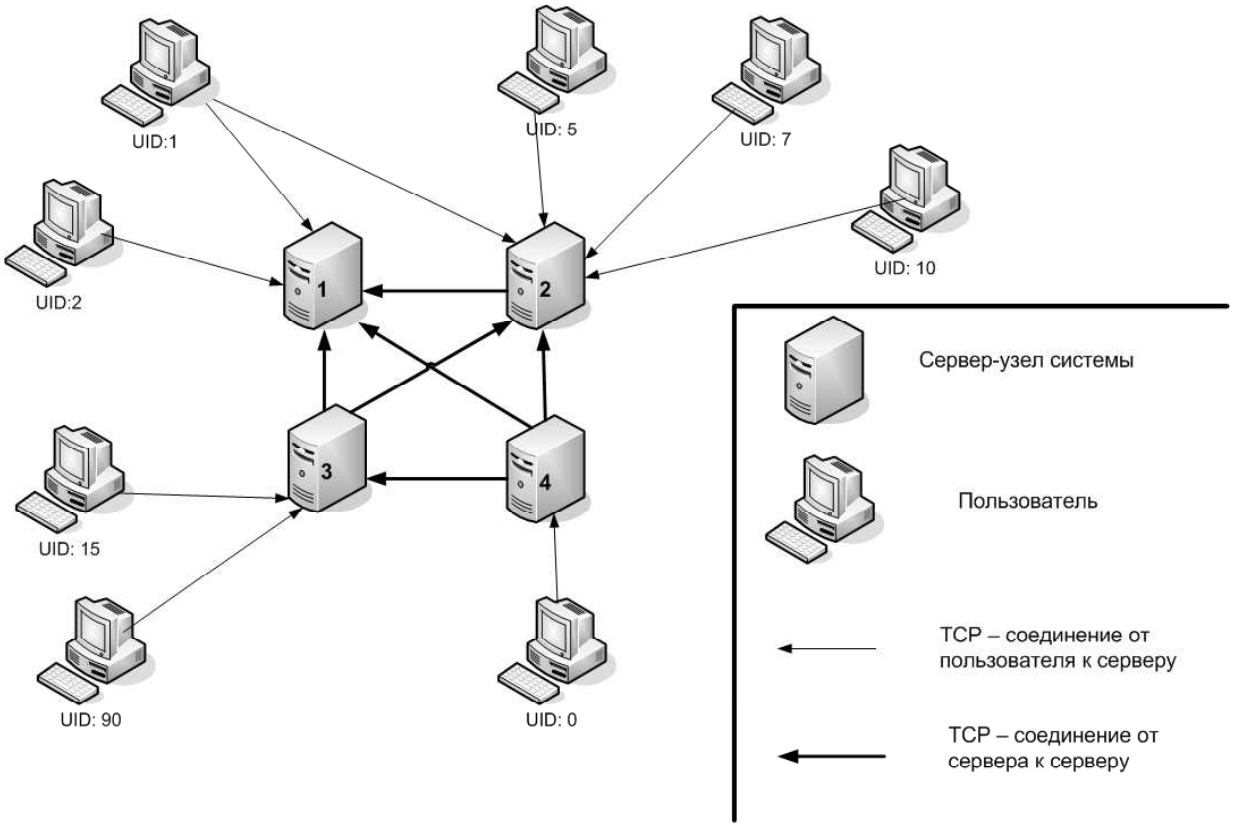

Рисунок 1— Схема взаимодействия компонентов системы

от функций работы с сетью, протоколом системы, авторизации. Последние нужно выделить в библиотеку. Данная библиотека должна облегчить разработку новых IM-клиентов, работающих с разрабатываемой системой, либо модификацию уже существующих. Серверная часть является приложением, которое обеспечивает передачу сообщений между пользователями и функционирование системы в целом. Функционирование системы обеспечивается взаимодействием множества узлов. Под узлом понимается запущенный экземпляр серверного приложения либо компьютер, подключенный к сети и выполняющий серверное приложение. Каждый клиент в процессе работы должен быть подключен к одному или нескольким узлам. Каждый узел должен иметь соединение со всеми остальными узлами. Соединения должны выполняться посредством протокола TCP. Схема взаимосвязей узлов и клиентов показана на рисунке 1. Модель данных системы в виде диаграммы "сущность-связь"отображена на рисунке 2

Клиентское и серверное приложения должны поддерживать работу

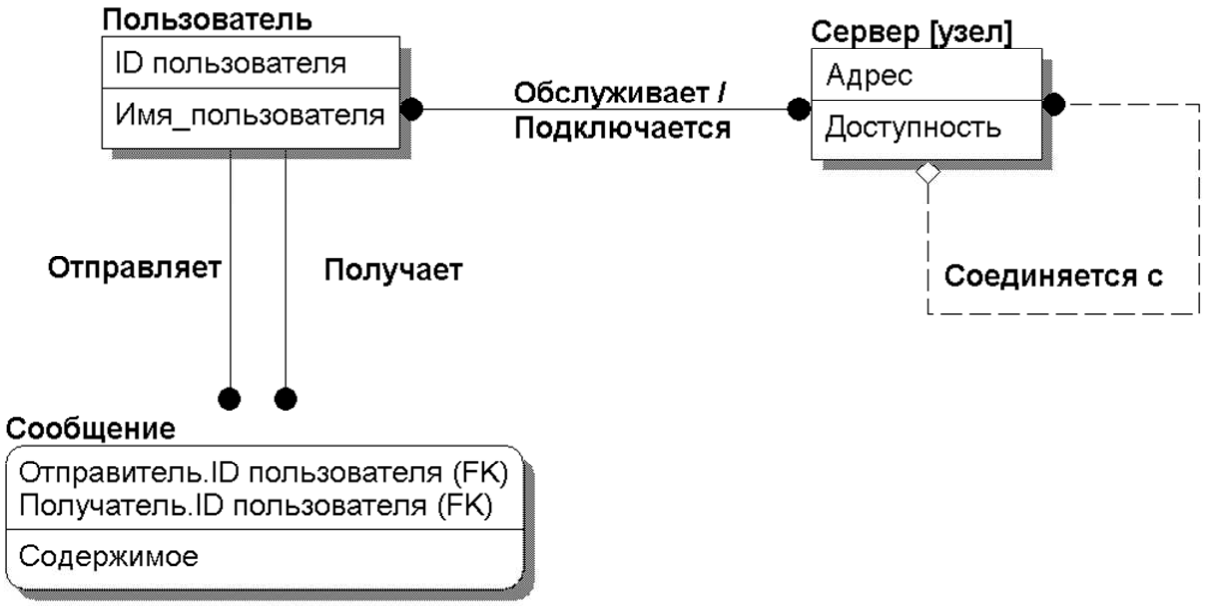

Рисунок 2— Модель данных системы

с протоколами сетевого уровня IPv4 и IPv6, а также должны иметь возможность добавления поддержки других протоколов сетевого уровня. Также клиентское и серверное приложения должны поддерживать адресацию посредством системы DNS.

2.3 Процессы, протекающие в системе

Процесс функционирования системы во времени можно разделить на отдельные процессы. Из этих процессов наиболее важными являются:

- Процессы подключения новых узлов (серверов).
- Процессы подключения и авторизации пользователей.
- Процессы поиска пользователей.
- Процессы создания и поддержания сеансов (каналов) связи между пользователями.
- Процесс поддержания в актуальном состоянии списка узлов.

Неотъемлемой частью задачи разработки рассматриваемой системы является реализация перечисленных выше процессов в виде программных алгоритмов.

2.4 Требования универсальности и переносимости

Серверное приложение должно быть кроссплатформенным на уровне исходных кодов, т.е. оно должно компилироваться и собираться без дополнительных модификаций на как можно большем количестве систем. Кроссплатформенность должна быть обеспечена, как минимум, для операционных систем на базе Linux и Windows 2000/XP/Vista.

Система должна поддерживать передачу сообщений на любых языках. Для обеспечения этой поддержки должны применяться стандарты Unicode. Текстовые сообщения, передаваемые системой, должны быть в кодировке UTF8. Также система должна поддерживать передачу информации произвольного характера посредством туннелирования. Ввод-вывод и кодирование данной информации должны обеспечиваться сторонними приложениями на компьютере пользователя.

3 Авторизация пользователей в системе

3.1 Преимущества авторизации с помощью цифровых сертификатов перед другими способами авторизации

В настоящее время во всех популярных IM-системах используется схема авторизации пользователей, основанная на использовании идентификатора (логина) и пароля. Реализация такой схемы требует хранения записей, сопоставляющих логины и пароли. Применение такой схемы в распределенной системе связано с определенными трудностями. Требуется либо наличие центрального сервера, хранящего учетные данные (при этом система уже не будет полностью децентрализованной), либо реализация распределенной базы данных с учетными записями, что является технически сложной задачей и к тому же увеличивает вероятность похищения учетных данных злоумышленниками.

Помимо использования пароля и логина существуют и другие способы авторизации, например, с использованием смарт-карт, на основе биометрических данных (отпечатков пальцев, изображения радужной оболочки глаза и т.д.). Однако эти способы требуют наличия дополнительного аппаратного обеспечения, что для рассматриваемой системы неприемлемо по причине резкого уменьшения круга потенциальных пользователей.

Поэтому для разрабатываемой в данной работе системы был выбран метод авторизации, основанный на применении цифровых сертификатов. Данный метод набирает популярность в последнее время. Для авторизации пользователи предъявляют свой цифровой сертификат, также в процессе авторизации используется закрытый ключ, соответствующий открытому ключу в сертификате. Цифровой сертификат представляет собой электронный документ, сопоставляющий реквизиты какого-либо субъекта с его открытым ключом. Сертификат подписывается открытым ключом центра серти-

фикации. Механизм данного способа авторизации основан на доверии центру сертификации. Для данной системы достаточно быть уверенным в том, что центр сертификации не будет подписывать новый сертификат, полностью дублирующий реквизиты в уже выданном сертификате. Авторизацию пользователя может осуществить любой узел системы, имеющий подлинную копию корневого сертификата центра сертификации.

Важным преимуществом использования сертификатов для авторизации является то, что при такой схеме процессы регистрации новых пользователей и авторизации существующих пользователей полностью независимы друг от друга. Регистрацию новых пользователей весьма проблематично осуществлять децентрализованно, т.к. необходимо обеспечивать уникальность выдаваемых идентификаторов (в данном случае номеров). Авторизация по цифровым сертификатам избавляет от необходимости иметь базу данных с учетными записями пользователей.

3.2 Реализация процесса авторизации

Авторизация в системе осуществляется при помощи цифровых сертификатов, соответствующих стандарту X.509 [5][6]. Пользователи однозначно идентифицируются номером, записанным в персональном цифровом сертификате. Выдачу пользователям персональных сертификатов осуществляет центр сертификации. Центр сертификации, обслуживающий систему, должен выдавать пользователям персональные сертификаты. Персональный сертификат пользователя должен содержать в текстовом виде уникальный идентификационный номер пользователя и псевдоним пользователя, разделенные символом . Данный номер представляет собой целое число в диапазоне $0 \ldots 2^{32}-1$. Персональный сертификат пользователя может также содержать другие данные пользователя. Все персональные сертификаты пользователей должны быть подписаны непосредственно открытым ключом корневого сертификата центра сертификации. Каждый узел должен иметь при себе копию корневого сертификата центра сертификации. Серверы-узлы системы никак не связаны с центром сертификации, достаточно, чтобы каждый узел имел

копию корневого сертификата центра сертификации. Авторизация пользователей осуществляется путем проверки соответствия сертификата субъекта, подписавшего персональный сертификат пользователя, копии корневого сертификата центра сертификации. Также пользователи могут осуществлять дополнительную взаимную проверку сертификатов. Для этого пользователи должны сохранять сертификаты всех своих собеседников. При повторном сеансе общения с тем же пользователем можно сопоставить предъявляемый собеседником сертификат с сохраненным. Если они различаются, то это свидетельствует о попытке выдать себя за другое лицо. Такая ситуация возможна, если был скомпрометирован (был похищен) закрытый ключ центра сертификации.

4 Разработка сетевого протокола

Для осуществления обмена информацией между компонентами системы (клиентами и узлами) через сеть необходим протокол, который определяет правила и способы этого обмена. Данный раздел посвящен разработке сетевого протокола распределенной защищенной системы обмена мгновенными сообщениями.

4.1 Текстовый и бинарный подход к проектированию протоколов

Существуют разные подходы к разработке и реализации сетевых протоколов. Один из таких подходов - представление команд и полей данных протокола в текстовом виде. Типичные примеры протоколов такого рода - это HTTP, FTP, POP3, SMTP. Например, в простейшем случае для протокола HTTP запрос на получение главной страницы сайта будет выглядеть так:

GET / HTTP/1.1
host: www.site.com

Обычно такой запрос содержит значительное количество дополнительных текстовых полей. Очевидным недостатком "текстовых"протоколов является их явная избыточность. Приверженцы текстового подхода называют такое преимущество, как легкость восприятия для человека. Однако данное преимущество является весьма сомнительным, если учитывать, что с протоколом в конечном итоге работает компьютер, а не человек. К тому же обработка текстовых полей требует больших вычислительных затрат и занимает больше времени программиста на написание соответствующих функций, по сравнению с полями, представленными в двоичном коде. Тем не менее, текстовый подход в разработке сетевых протоколов пользуется популярностью и получил дальнейшее развитие в форме использования языка разметки XML для описания полей протокола. Например, на XML основан протокол XAMPP, использующийся в системе Jabber. Приведем пример [7]:

<message from='juliet@example.com'
to='romeo@example.net'
xml:lang='en'>
<body>Art thou not Romeo, and a Montague?</body>
</message>
<message from='romeo@example.net'
to='juliet@example.com'
xml:lang='en'>
<body>Neither, fair saint, if either thee dislike.</body>
</message>

Помимо повышенной избыточности, такой протокол обладает еще одним недостатком - в каждом сообщении содержатся одни и те же последовательности символов, такие как "message "body"и т.д. Это способствует снижению криптографической стойкости к дешифрованию зашифрованных сообщений. Очевидно, что для защищенной системы обмена сообщениями это недопустимо. В свете изложенных выше соображений было решено, что сетевой протокол для разрабатываемой системы будет основан на "бинарном"подходе. При таком подходе команды и поля данных стремятся кодировать минимально необходимым количеством битов. Желательно при этом соблюдать компромисс между минимизацией избыточности в передаваемых сообщениях и удобством работы с протоколом.

4.2 Уровни протоколов системы

Разрабатываемый в данном разделе протокол,согласно модели OSI соответствует 7-му уровню - уровню приложения (см. рисунок 3). Уровнем ниже находится протокол TLS (Transport Level Security). Этот протокол обеспечивает шифрование передаваемых сообщений. Он, в свою очередь, пользуется услугами протокола TCP. Протоколы TLS и TCP не вписываются полностью в пределы одного уровня модели OSI, т.к. TLS выполняет функции, которые можно отнести не только к уровню представления, но и к сеансовому, аналогично протокол TCP, помимо транспортного уровня, реализует

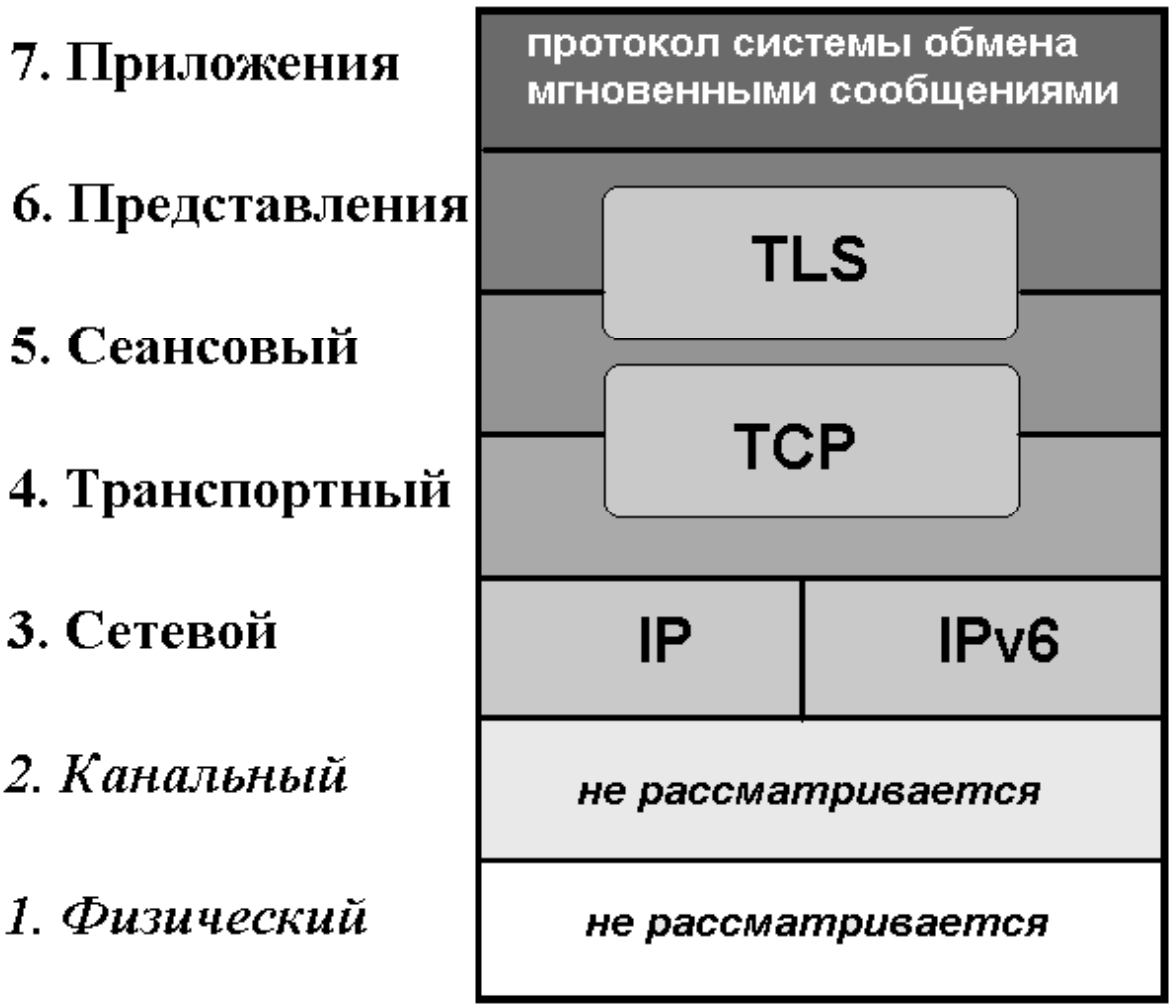

Рисунок 3— Уровни протоколов системы согласно модели OSI

часть функций сеансового уровня.

Разрабатываемая система поддерживает работу с протоколами сетевого уровня IP и IPv6. Выбор протоколов канального и физического уровня абсолютно независим от разрабатываемой системы.

4.3 Структура сообщения

Протокол системы базируется на сообщениях (пакетах), отправляемых друг другу клиентами и серверами. Каждое сообщение состоит из идентифицирующего заголовка и остальной части-хвоста (см. рисунок 4). Идентифицирующий заголовок является последовательностью из двух байт (октетов), имеющих одинаковое значение. Это значение определяет тип со-

Рисунок 4— Структура сообщения протокола

общения. Хвост может быть фиксированной или переменной длины. В случае переменной длины сразу после заголовка следует поле размером в 1 или 2 байта, которое должно определяться как беззнаковое целое число, содержащее длину оставшейся части сообщения в байтах (не включая поле размера). Таким образом, общая длина сообщения переменной длины определяется следующим образом: 2 байта заголовка + длина поля размера (1 или 2 байта, в зависимости от типа сообщения) + значение поля размера. Из сказанного следует, что максимальная длина сообщения составляет 65539 байт.

4.4 Виды сетевых соединений

Сообщения протокола можно разделить по видам соединений, по которым они передаются. Всего в системе возможны 3 вида соединений: клиент-сервер,клиент-клиент, сервер-сервер.

Соединение клиент-сервер - это управляющее соединение, по которому передается служебная информация. Для данного соединения выбран TCP порт 51914.

Соединение клиент-клиент осуществляет передачу между пользователями полезной информации - текстовых сообщений. Данное соединение является не прямым TCP-соединением между компьютерами пользователей (хотя оно может быть организовано и таким образом), а реализованным поверх цепочки соединений клиент1->сервер1->сервер2<-клиент2 в общем случае. В частном случае оба клиента могут быть подключены к одному и

тому же серверу. Каждый из клиентов создает отдельное TCP соединение к серверу,на котором он авторизован, на порт 51915. Данные, отправленые клиентом своему серверу по этому соединенению, пересылаются серверу собеседника, тот, в свою очередь, пересылает их клиенту - получателю данных. Взаимодействие клиентов осуществляется путем установления виртуальных соединений-сессий, наподобие телефонного звонка.

Соединение сервер-сервер использует TCP порт 5196. По нему передаются потоки данных клиентов и списки адресов узлов.

4.5 Спецификация протокола

В данном подразделе описаны все сообщения протокола. Поля, размер которых не задан явно, имеют длину 1 байт. В полях длиной больше одного байта порядок байт – little-endian (младший байт по младшему адресу), за исключением полей, содержащих IP или IPv6 адрес – в таких полях порядок байт сетевой, т.е. big-endian (старший байт по младшему адресу).

– Запрос на подключение клиента

0xCA	0xCA	старший номер версии протокола	младший номер версии протокола

– Положительный ответ сервера

0x01	0x01	0x4F (символ 'O')	0x4B (символ 'K')

– Различные версии протокола

0x02	0x02	старший номер версии протокола	младший номер версии протокола

– Запрос на установление соединения с собеседником

0x03	0x03	UID вызываемого пользователя (4 байта)

– Запрос на разрешение соединения

0x04	0x04	UID вызывающего пользователя (4 байта)

– Пользователь с данным UID не найден

0x05	0x05	UID вызываемого пользователя (4 байта)

– Разрешение соединения от вызываемого пользователя

0x06	0x06	UID вызывающего пользователя (4 байта)

– Пользователь с данным UID найден и готов общаться

0x07	0x07	UID вызываемого пользователя (4 байта)

– Отказ от общения

0x08	0x08	UID вызывающего пользователя (4 байта)

– Идентификация нового соединения

0x20	0x20	UID вызывающего пользователя (4 байта)	UID вызываемого пользователя (4 байта)

– Сообщение - текст в формате UTF8

0x30	0x30	длина сообщения в байтах (2 байта)	текст в формате UTF8

– Отчет о доставке предыдущего сообщения

0x40	0x40	случайное число (2 байта)

– Запрос на подключение сервера

0xCB	0xCB	старший номер версии протокола	младший номер версии протокола

– Запрос списка узлов

0x90	0x90	случайное число (2 байта)

– Анонс нового узла - IPv4 адрес

0xA0	0xA0	IP-адрес нового узла (4 байта)

– Анонс нового узла - IPv6 адрес

0xA1	0xA1	IPv6-адрес нового узла (16 байт)

– Анонс нового узла - DNS адрес

0xA2	0xA2	длина адреса	DNS адрес нового узла (0 - 255 байт)

– Список узлов

0xA4	0xA4	размер списка адресов в байтах (2 байта)	список адресов

Список адресов представляет собой последовательность структур, описывающих адрес узла. Каждая структура может быть одного из 3 видов:

0xA0	IP-адрес (4 байта)

или

0xA1	IPv6-адрес (16 байт)

или

0xA2	длина адреса	DNS адрес (0 - 255 байт)

4.6 Сценарий взаимодействия клиента с сервером

Клиент выполняет подключение к серверу на порт 51914 по протоколу TCP и посылает **запрос на подключение клиента**(0xCA) (далее все названия сообщений протокола, приведенных в предыдущем подразделе, выделены жирным шрифтом и отмечены типом сообщения в скобках в виде числа в шестнадцатеричной записи). Запрос содержит номер версии протокола клиента. Если версия протокола сервера отличается от указанной в запросе, сервер отвечает сообщением **различные версии протокола**(0x02) и разрывает соединение. Если же версии протокола совпадают,

сервер присылает **положительный ответ сервера**(0x01). После этого происходит установка защищенного TLS соединения. Этот процесс описан в стандарте RFC2246 [8]. При этом сервер выступает в роли TLS-клиента, а клиент - в роли TLS-сервера. Это обусловлено тем, что для установления TLS соединения TLS-сервер должен иметь сертификат, для TLS-клиента это не обязательно.

После того, как TLS соединение установлено, сервер присылает **список узлов** (0xA4). Если весь список не помещается в одном сообщении, то присылается несколько сообщений.

Клиент может создавать исходящие соединения с другими клиентами и принимать входящие. Для открытия соединения клиент отправляет **запрос на установление соединения с собеседником** (0x03). В этом запросе указывается уникальный идентифицирующий номер (UID) вызываемого пользователя. Сервер выполняет поиск пользователя с данным UID: сначала в списке своих (подключенных к данному серверу) клиентов, затем в кэше, затем выполняет последовательный опрос всех активных узлов. Если клиент найден, то ему посылается **запрос на разрешение соединения** (0x04). Вызываемый клиент может отказаться от общения, отправив **отказ от общения** (0x08). Если же клиент согласен на установление соединения, то он отправляет **разрешение соединения от вызываемого пользователя** (0x06), после чего выполняет подключение к серверу на порт 51915. Вызывающему клиенту отправляется **пользователь с данным UID найден и готов общаться** (0x07). Получив данное сообщение, вызывающий клиент также создает новое TCP соединение со своим сервером на порт 51915. Оба клиента отправляют по новому созданному соединению **идентификация нового соединения** (0x20), после проверки которых серверы организуют логический канал связи между двумя клиентами. Теперь данные, отправленные серверу по созданному TCP соединению одним клиентом, будут получены другим клиентом, и наоборот. Клиенты устанавливают через канал связи TLS соединение, сервером TLS при этом выступает вызывающий клиент. После того, как защищенное соединение установлено, клиенты могут посылать друг другу **текстовое сообщение** (0x30). Получив такое

сообщение, клиент должен отправить **отчет о доставке** (0x40).

4.7 Сценарий взаимодействия серверов между собой

При запуске сервер пытается подключиться к каждому из имеющихся у него в списке серверов. В случае успешного подключения сервер отправляет **запрос на подключение сервера** (0xCB), содержащий версию протокола. Ответ другого сервера аналогичен случаю с подключением клиента - в ответ отправляется либо **положительный ответ сервера**(0x01), либо **различные версии протокола**(0x02) и соединение разрывается. После успешной проверки версий протокола новый сервер отправляет анонс своего адреса - сообщение (0xA0), (0xA1) или (0xA2), в зависимости от типа сетевого адреса (IPv4, IPv6 или DNS). Сервер, получивший анонс, проверяет соответствие адреса, указанного в сообщении, с реальным адресом, с которого осуществлено TCP соединение. При несоответствии соединение разрывается. После отправки анонса адреса новый узел посылает **запрос списка узлов**(0x90). В ответ он получает полный список адресов всех действующих на данный момент узлов в виде одного или нескольких сообщений **список узлов** (0xA4). Далее новый узел заменяет свой локальный список адресов узлов полученным новым и выполняет подключение к каждому узлу из списка. Подключившись к очередному узлу, новый узел отправляет анонс своего адреса. После завершения этого процесса новый узел можно считать успешно включенным в распределенную сеть обмена сообщениями.

5 Разработка серверной части

Серверная часть системы представляет собой приложение, работающее в фоновом режиме и не взаимодействующее с пользователем посредством пользовательского интерфейса. В терминологии Windows - это сервис (служба), в терминологии Unix - daemon. Каждый сервер (запущенный экземпляр серверного приложения) является узлом распределенной системы. Каждый узел взаимодействует с другими узлами, а также обслуживает пользователей. Узел отвечает за организацию соединений между клиентами, осуществляет передачу данных по этим соединениям, обеспечивает поиск авторизованных в системе пользователей, а также хранит и поддерживает в актуальном состоянии список адресов всех узлов системы.

5.1 Выбор инструментов разработки.

В качестве языка разработки для серверного приложения выбран C. Выбор обусловлен гибкостью данного языка, возможностью выполнения низкоуровневых операций, кроссплатформенностью, наличием инструментариев разработки для огромного количества программно-аппаратных платформ. В качестве основного компилятора выбран GCC - GNU Compilers Collection. GCC портирован на множество платформ, поддерживает кросс-компиляцию. Разработка ведется в среде дистрибутива Linux Mandriva. Для сборки используются утилиты из пакета binutils - ld, make и т.д. Для сборки под ОС Windows применяется среда разработки DevCpp, которая использует MinGW - порт GCC на Windows. Для работы с протоколом TLS применяется библиотека OpenSSL. Данная библиотека является весьма популярной и широко распространенной, она обеспечивает работу с протоколами защищенного соединения SSLv2, SSLv3,TLSv1 а также работу с цифровыми сертификатами. Все перечисленные программные продукты являются свободно распространяемыми.

5.2 Алгоритм подключения нового узла

При запуске серверное приложение считывает из файла список адресов узлов системы. Данный файл поставляется вместе с приложением, содержимое данного файла периодически обновляется в процессе работы серверного приложения. Таким образом, при запуске файл содержит актуальный список адресов действующих узлов либо на момент выхода дистрибутива серверного приложения, либо на момент последнего сохранения списка в файл. Далее сервер пытается подключиться к каждому из узлов. В случае успешного подключения сервер запрашивает актуальный список адресов всех узлов в системе. Получив этот список, сервер соединяется с каждым из узлов в данном списке. После этого сервер становится равноправным узлом распределенной системы. Рассмотренный алгоритм представлен в виде блок-схемы на рисунке 5.

5.3 Алгоритм поиска пользователей

Реализация поиска в распределенных системах представляет собой нетривиальную задачу. Причиной этого является то, что набор данных, в котором осуществляется поиск, распределен между большим числом компьютеров.

Для рассматриваемой системы необходим алгоритм поиска активных пользователей, т.е. пользователей, подключенных и авторизовавшихся на каком-либо сервере. Любой пользователь данной системы может быть авторизован на более чем одном сервере. Результатом поиска пользователя должен быть адрес узла, на котором он авторизован. Теоретически возможны два способа учета активных пользователей со стороны сервера – анонсирование каждого вновь авторизовавшегося пользователя всем остальным серверам-узлам и опрос каждого узла на предмет наличия у него информации о заданном пользователе. Для реализации первого способа каждый сервер должен хранить в памяти информацию обо всех активных пользователях в сети. При максимально возможном количестве активных пользователей (4294967296) и минимальном размере структуры, отведенной в памяти

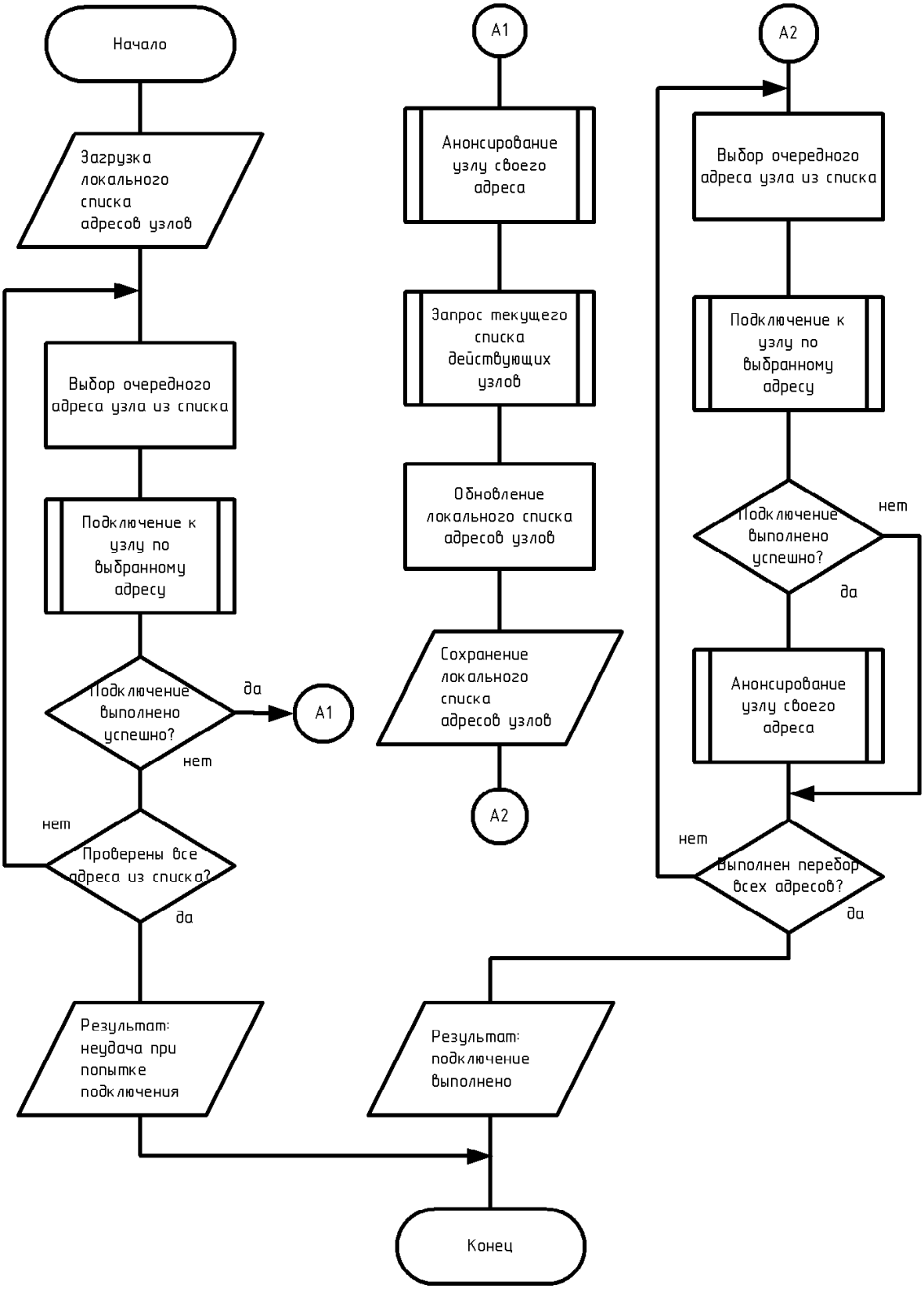

Рисунок 5— Алгоритм подключения нового узла

на каждого пользователя (8 байт: 4 байта на номер пользователя и 4 байта на IP-адрес узла, на котором авторизован пользователь), небходимый объем памяти составит 32 Гигабайта. Достоинством данного способа является малое время поиска заданного пользователя. Опрос каждого узла для поиска заданного пользователя требует гораздо больших затрат времени, однако при этом требования по памяти минимальны. Время поиска можно уменьшить, если после отправки запроса одному узлу посылать запрос следующему, не дожидась ответа от предыдущего узла. Также среднее время поиска можно уменьшить путем введения кэширования. Исходя из этих соображений, был выбран второй вариант алгоритма поиска пользователей – опрос каждого узла.

Поиск пользователей в системе реализован следущим образом. Пользователь присылает запрос, содержащий номер (UID) искомого пользователя. Сервер в первую очередь выполняет поиск по заданному UID среди подключенных к нему пользователей. Каждая структура, описывающая подключенного и авторизованного пользователя, включена в бинарное дерево поиска, упорядоченное по UID пользователей. Таким образом, среднее время поиска среди локальных пользователей пропорционально $log_2 N$, где N – количество локальных пользователей. Если пользователь с заданным UID обнаружен, процесс поиска прекращается. Если нет, то поиск продолжается в локальном кэше. Кэш содержит записи с результатами предыдущих запросов на поиск в виде сочетаний UID-адрес узла. Если пользователь найден в кэше, то соответствующему узлу посылается запрос. Если узел отвечает на данный запрос, то поиск прекращается. Если узел не отвечает, либо запись с данным UID в кэше не обнаружена, то начинается последовательный опрос всех узлов. Если какой-либо узел присылает утвердительный ответ, то поиск прекращается и в кэш заносится результат поиска. Если же в течение заданного времени ни один узел не пришлет ответа, то пользователь с заданным UID считается не активным (находящимся в "оффлайне") и запрашивающему пользователю отправляется соответствующее сообщение. Рассмотренный алгоритм представлен в виде блок-схемы на рисунке 6.

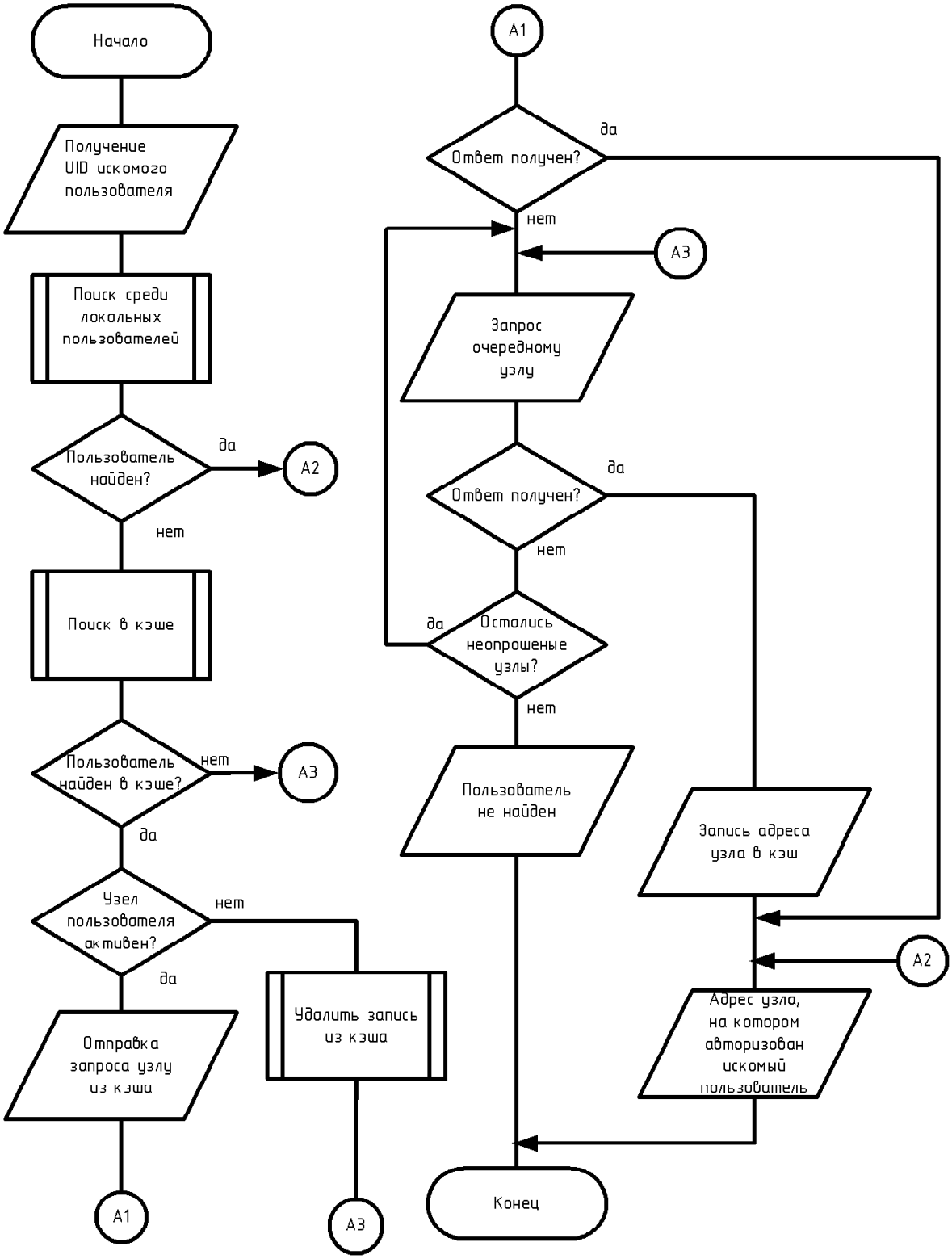

Рисунок 6— Алгоритм поиска клиентов

6 Разработка клиентской части

Клиентская часть представляет собой приложение, предназначенное для обмена сообщениями и хранения контактов. Многие современные IM-клиенты поддерживают большое количество протоколов различных систем обмена сообщениями, поддержка каждого протокола обычно выполняется в виде отдельного плагина. В настоящее время уже сформировался более-менее привычный пользовательский интерфейс IM-клиентов, и никаких новшеств в этом плане в данной работе создавать не планируется. Исходя из этого вполне логично рассмотреть создание не какого-то конкретного приложения - клиента системы обмена мгновенными сообщениями, а клиентской библиотеки. Данная библиотека реализует функции работы с протоколом, авторизации и отправки-приема сообщений. С помощью данной библиотеки можно легко разработать готовое приложение - IM-клиент, либо плагин, добавляющий возможность работы в разрабатываемой системе в уже существующий IM-клиент. Возможные кандидаты - это QIP Infium, Miranda, Pidgin и другие.

6.1 Выбор инструментов разработки

Клиентская библиотека разрабатывается на языке C. Для разработки клиентской библиотеки применяются те же инструменты, что и для серверного приложения. В качестве основного компилятора выбран GCC - GNU Compilers Collection. Разработка ведется в среде дистрибутива Linux Mandriva. Для сборки используются утилиты из пакета binutils - ld, make и т.д. Для сборки под ОС Windows применяется среда разработки DevCpp, которая использует MinGW - порт GCC на Windows. Для работы с протоколом TLS применяется библиотека OpenSSL.

6.2 Программный интерфейс клиентской библиотеки

Программный интерфейс клиентской библиотеки на данный момент состоит из 9 функций, вызываемых клиентским приложением и предназначенных для инициализации библиотеки, подключения к заданному узлу, создания соединения с указанным пользователем, обработки сообщений сервера, отправки сообщений и закрытия соединений. Также в библиотеке объявлены 6 функций обратного вызова. Эти функции определяются клиентским приложением. Они предназначены для задания реакции приложения на различные события. Ниже приведено описание всех функций из программного интерфейса библиотеки.

int DPIM_client_init(struct DPIM_client_callbacks *clcbs);

Осуществляет инициализацию клиентской библиотеки. Принимает единственный аргумент ***clcbs*** *– указатель на структуру* ***DPIM_client_callbacks****, содержащую адреса функций обратного вызова, которые должны быть определены в приложении. Описание структуры* ***DPIM_client_callbacks*** *приведено ниже. Функция возвращает значение* ***1*** *при удачном завершении или* ***0*** *в случае возникновения ошибки.*

```
struct DPIM_client_callbacks
{
void (*newmsg_cb)(unsigned int peer_id,char* msgbuf,unsigned
int nb,int type);
void (*msgok_cb)(unsigned int peer_id);
void (*channel_established_cb)(unsigned int peer_id);
void (*channel_failed)(unsigned int peer_id);
int (*accept_cb)(unsigned int caller_id);
void (*notfound_cb)(unsigned int callee_id);
};
```

void (*newmsg_cb)(unsigned int peer_id,char* msgbuf,unsigned int nb,int type);

функция вызывается при получении текстового сообщения от удаленного пользователя. Аргумент ***peer_id*** *содержит идентификатор пользователя, приславшего сообщения,* ***msgbuf*** *содержит адрес буфера, содержащего полученное сообщение,* ***nb*** *– размер сообщения в байтах,* ***type*** *– тип сообщения. На данный момент поддерживается только один тип сообщения* ***DPIM_MSGUTF8*** *– текстовое сообщение в кодировке UTF8. Следует отметить, что сообщение, хранящееся в буфере, не заканчивается завершающим 0. Это нужно учитывать при реализации данной функции.*

void (*msgok_cb)(unsigned int peer_id);

функция вызывается при получении подтверждения о доставке сообщения собеседнику. Аргумент ***peer_id*** *содержит идентификатор пользователя, которому отправлено сообщение.*

void (*channel_established_cb)(unsigned int peer_id);

функция вызывается после успешного создания соединения с пользователем, идентификатор которого передается в аргументе ***peer_id****.*

void (*channel_failed)(unsigned int peer_id);

функция вызывается в случае, если попытка установить соединение с пользователем, идентификатор которого передается в аргументе ***peer_id****, оказалась неуспешной.*

Вызов данной функции происходит, когда заданный пользователь найден, т.е. находится в сети и согласился на установление соединения с собой, но в процессе создания соединения возник сбой: разрыв сетевого соединения, сертификат пользователя оказался поддельным, либо сбой по иным причинам.

int (*accept_cb)(unsigned int caller_id);

функция вызывается при получении запроса на установление соединения от удаленного пользователя. Идентификатор данного пользователя передается в аргументе ***caller_id****. Данная функция должна вернуть ненулевое значение, если нужно согласиться на создание соединения с данным пользователем, либо значение* ***0****, если нужно отказать данному пользователю. В случае возврата нулевого значения серверу отправляется соответствующее сообщение протокола, в результате чего запрашивающий пользователь получит сообщение о том, что такой пользователь не найден. Данная функция может быть использована для реализации т.н. "черных списков", т.е. списков нежелательных пользователей.*

void (*notfound_cb)(unsigned int callee_id);

функция вызывается при получении сообщения о том, что вызываемый пользователь не найден. Это свидетельствует о том, что пользователь, указанный аргументом ***callee_id****, либо действительно не найден, т.е. находится в "оффлайне", либо отказался от создания соединения.*

int DPIM_new_connection(struct sockaddr *addr,char* cert_prkey,char* password,struct DPIM_conninfo **ci,char* root_cert);

Осуществляет подключение к серверу и авторизацию. Адрес сервера задается аргументом ***addr****. Поля структуры, на которую указывает* ***addr****, должны быть заполнены до вызова функции. Аргумент* ***cert_prkey*** *содержит имя файла, содержащего персональный сертификат пользователя и закрытый ключ, закодированные в формате PEM.* ***password*** *содержит указатель на строку, хранящую пароль для закрытого ключа. Если закрытый ключ не зашифрован, то пароль может быть произвольным. Аргумент* ***ci*** *представляет собой указатель на переменную, в которую сохраняется адрес структуры* ***DPIM_conninfo****, описывающей соединение с сервером. Аргумент* ***root_cert*** *задает имя файла, содержащего корневой сертификат центра сертификации в формате PEM. В случае успешного выполнения функция возвращает значение* ***DPIM_OK****. Остальные возможные значения (коды ошибок) перечислены в заголовочном файле в приложении.*

int DPIM_create_channel(struct DPIM_conninfo *ci,unsigned int uid,struct DPIM_chaninfo **chn);

Данная функция предназначена для создания канала связи с указанным пользователем. Аргумент ***ci*** *задает указатель на структуру, которая должна быть заполнена после успешного вызова функции* ***DPIM_new_connection****. Указатель на данную структуру идентифицирует управляющее соединение с сервером. Аргумент* ***uid*** *задает идентификационный номер пользователя, с которым нужно установить канал связи. Аргумент* ***chn*** *представляет собой указатель на переменную, которая будет проиниция-*

лизирована в случае успешного выполнения функции. Переменная будет идентифицировать данный канал связи. В случае успешного выполнения функция вернет значение ***DPIM_OK****. Однако успешное завершение функции не означает успешного создания канала. О результате попытки создания канала библиотека уведомит приложение посредством соответствующих функций обратного вызова.*

int DPIM_read(struct DPIM_conninfo *ci,int timeout);

Функция осуществляет обработку всех входящих сообщений, полученных по соединению с сервером, указанному аргументом ***ci****, а также полученных по установленным каналам связи с удаленными пользователями, ассоциированными с данным подключением к серверу. Аргумент* ***timeout*** *задает время ожидания в микросекундах. Эту функцию следует периодически вызывать в процессе работы приложения для каждого установленного соединения с сервером. При получении сообщений данная функция инициирует вызов необходимых функций обратного вызова. В случае успешного выполнения функция возвращает значение* ***DPIM_OK****. Остальные возможные значения (коды ошибок) перечислены в заголовочном файле в приложении.*

int DPIM_send_msg(struct DPIM_chaninfo *chn, char *buf,unsigned int len);

Функция осуществляет отправку текстового сообщения по установленному каналу связи ***chn****. Аргумент* ***buf*** *задает адрес буфера с сообщением. Сообщение должно быть в кодировке UTF8. Аргумент* ***len*** *определяет длину сообщения в байтах. В случае успешного выполнения функция возвращает значение* ***DPIM_OK****.*

void DPIM_close_channel(struct DPIM_chaninfo *chn);

Функция осуществляет закрытие канала связи ***chn****. Функцию следует вызывать только для установленного канала. При вызове функции для идентификатора канала, попытка создания которого была неудачной, произойдет аварийное закрытие приложения. Также не следует вызывать функцию повторно для уже закрытого канала.*

void DPIM_close_connection(struct DPIM_conninfo *cid);

Функция осуществляет закрытие соединения с сервером ***cid****. Данная функция закрывает также все связанные с указанным соединением каналы. Функцию не следует вызывать повторно для уже закрытого соединения.*

Исходный код клиентской библиотеки приведен в приложении.

6.3 Подключение и авторизация пользователя

Для того, чтобы клиентское приложение смогло подключиться к системе и стать частью распределенной сети, необходим список адресов действующих узлов-серверов. Этот список должен включаться в дистрибутив клиентского приложения. При запуске клиентское приложение считывает этот список и пытается подключиться к каждому серверу из списка. Подключение и последующая авторизация осуществляется путем вызова функции клиентской библиотеки **DPIM_new_connection**. В случае успешного подключения и авторизации клиентское приложение получает актуальный на данный момент список адресов всех активных узлов. Приложение должно сохранить этот список в локальное хранилище. Алгоритм процесса подключения и авторизации показан в виде блок-схемы на рисунке 7. Попытка подключения клиента к системе может окончиться неудачно. Если с момента выпуска дистрибутива программы либо с момента последнего обновления списка прошло достаточно много времени, может случиться так, что ни один

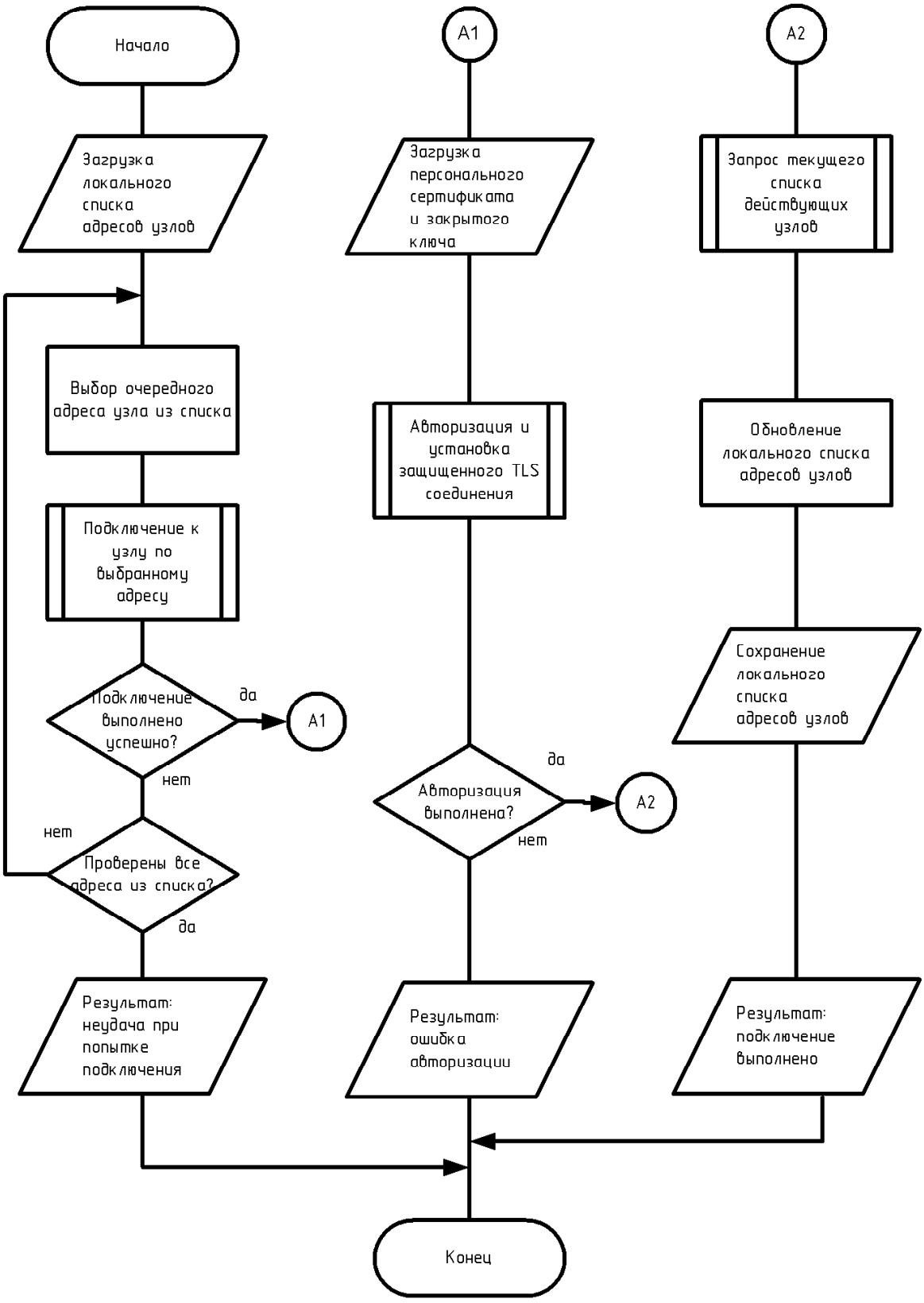

Рисунок 7— Алгоритм подключения и авторизации пользователя

из перечисленных в списке серверов уже не действует, и приложение не сможет подключиться к системе. Универсального способа решения данной проблемы нет, можно лишь привести некоторые рекомендации, позволяющие эту проблему избежать. Например, желательно чтобы приложение работало без перерывов, т.е. круглосуточно, в этом случае оно всегда будет иметь актуальный список узлов. Это требование достаточно очевидно для серверных приложений, однако оно не всегда приемлемо для клиентов. Другой вариант – можно публиковать на каком-либо заранее известном web-сайте (либо каким-нибудь иным способом) постоянно обновляемый список адресов узлов системы.

Следует отметить, что пользователь может быть авторизован на более чем одном узле. Соответственно, он может создавать каналы связи с другими пользователями через разные узлы. Множественные подключения клиентов с одной стороны дают возможность быстрого переключения в случае выхода какого-либо узла из строя, с другой стороны они увеличивают общую нагрузку узлов. Поэтому наиболее оптимальным стоит считать вариант, когда каждый пользователь подключается к не более, чем 2-3 серверам.

Заключение

В этой работе рассмотрена разработка распределенной защищенной системы обмена мгновенными сообщениями. Полноценная программная реализация и внедрение такой системы является достаточно трудоемким процессом, не входящим в рамки данной работы. В данной работе представлены ключевые принципы и особенности системы, продемонстрирована уникальность и эффективность сочетания этих принципов для сферы приложений мгновенного обмена сообщениями, разработаны алгоритмы, необходимые для реализации основных процессов, протекающих в системе.

Проектирование распределенных и децентрализованных приложений в последнее время является объектом повышенного внимания со стороны исследователей. В настоящее время в сети Интернет развернуто большое количество распределенных систем различного назначения, как полностью децентрализованных, так и имеющих централизованные структурные части. Сюда входят файлообменные пиринговые сети, проекты распределенных вычислений, а также различные ботнеты, организованные из зараженных троянскими программами компьютеров,применяемые для рассылки спама и организации DDoS-атак.

Обобщая вышесказанное, можно уверено считать тему данной выпускной квалификационной работы актуальной и весьма интересной.

Список использованных источников

1. Collins A., Blakey P., Streeter R. Distributed, Anonymous and Private Instant Messenger //.– Электрон. дан. (1 файл). – http://www.cs.kent.ac.uk/pubs/ug/2005/co600/dapim/report.pdf
2. Baset Salman A., Schulzrinne Henning G. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol // Department of Computer Science, Columbia University, New York //.– Электрон. дан. (1 файл). – http://www1.cs.columbia.edu/ salman/publications/skype1_4.pdf
3. Skype не работает во всем мире [Электронный ресурс]// http://www.rambler.ru/news/it/0/10977262.html
4. Безопасность системы Skype поставлена под сомнение [Электронный ресурс]// http://www.cybersecurity.ru/crypto/52747.html
5. Housley R., Polk W., Ford W., Solo D. RFC3280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. - 2002. [Электронный ресурс]// http://www.rfc-editor.org
6. Myers M., Ankney R., Adams C. RFC 2560 X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. – 1999. [Электронный ресурс]// http://www.rfc-editor.org
7. Saint-Andre P. RFC3920 Extensible Messaging and Presence Protocol (XMPP): Core. Ed. – 2004. [Электронный ресурс]// http://www.rfc-editor.org
8. Dierks T., Allen C. RFC2246 The TLS Protocol Version 1.0. – 1999. [Электронный ресурс]// http://www.rfc-editor.org

Приложение А

Ниже приведен исходный код клиентской библиотеки.

Файл DPIM_client.h:

```
#ifndef DPIM_CLIENT_H
#define DPIM_CLIENT_H

#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#ifdef WIN32

#include <windows.h>
#include <winsock2.h>

#else

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define SOCKET int
#define SOCKET_ERROR -1

#endif

#define DPIM_MSGOK 3
#define DPIM_MSGUTF8 2
#define DPIM_NEWCHAN 1
#define DPIM_OK 0
#define DPIM_ERRINVARG -1
#define DPIM_ERRNOMEM -2
#define DPIM_ERRSSL -3
#define DPIM_ERRSOCK -4
#define DPIM_ERRSERV -5
#define DPIM_ERRVERDIFFER -6
#define DPIM_ERRNOTFOUND -7
#define DPIM_ERRNOPEERCERT -8
#define DPIM_ERRINVPEERCERT -9
#define DPIM_ERRDISCONNECT -10
#define DPIM_ERRINVCERT -11
#define DPIM_ERRMSGSIZE -12
```

```
#define DPIM_CHNMSGBUFLEN 104

#define DPIM_CHANSTATE_ACTIVE 13

struct DPIM_conninfo;
struct DPIM_chaninfo;

struct DPIM_conninfo
{
#warning "sockaddr!"
struct sockaddr_in srv_addr; // sockaddr !!
unsigned int id;
SSL_CTX* ctx;
SOCKET sck_comm;
SSL* ssl_comm;
//сертификат клиента
unsigned char r_buf[128];
unsigned int r_pos;

struct DPIM_chaninfo *chan_list_head;
struct DPIM_chaninfo *chan_list_end;
};

struct DPIM_chaninfo
{
//соединение с сервером, через которое установлен данный логический канал связи
struct DPIM_conninfo *conn;
unsigned int peer_id;//номер собеседника
SOCKET sck_data;
SSL *ssl;
//1 если создание канала инициировано локальным клиентом, 0 - если собеседником
int outgoing;
X509 *peer_cert;
unsigned char msgbuf[DPIM_CHNMSGBUFLEN];

struct DPIM_chaninfo *next;
struct DPIM_chaninfo *prev;
int state;
};

struct DPIM_client_callbacks
{
//получение сообщения.
void (*newmsg_cb)(unsigned int peer_id,char* msgbuf,unsigned int nb,int type);
//получение подтверждения о доставке сообщения
void (*msgok_cb)(unsigned int peer_id);
//создание канала с собеседником
void (*channel_established_cb)(unsigned int peer_id);
//создать канал не удалось
void (*channel_failed)(unsigned int peer_id);
```

```
//входящий запрос на создание канала от пользователя с номером caller_id
int (*accept_cb)(unsigned int caller_id);
//запрошенный пользователь не найден, в оффлайне или отказался от общения
void (*notfound_cb)(unsigned int callee_id);

};

int DPIM_client_init(struct DPIM_client_callbacks *clcbs);

int DPIM_new_connection
(struct sockaddr *addr,char* cert_prkey,char* password,
struct DPIM_conninfo **ci,char* root_cert);
int DPIM_create_channel
(struct DPIM_conninfo *ci,unsigned int uid,
struct DPIM_chaninfo **chn);
int DPIM_read_conn
(struct DPIM_conninfo *ci /* ,int (*accept_cb)(unsigned int caller_id),
struct DPIM_chaninfo **nchn */);

int DPIM_read(struct DPIM_conninfo *ci,int timeout);

int DPIM_send_msg
(struct DPIM_chaninfo *chn, char *buf,unsigned int len);
int DPIM_read_msg
(struct DPIM_chaninfo *chn,char *buf,unsigned int *n);

void DPIM_close_channel(struct DPIM_chaninfo *chn);
void DPIM_close_connection(struct DPIM_conninfo *cid);

#endif
```

Файл DPIM_client.c:

```
#include "DPIM_client.h"
#include "DPIM_proto.h"
#include "DPIM_msgproc.h"

#ifdef NO_RAND_DEV
#include "DPIM_rand_seed.h"
#endif

#ifdef WIN32
#define close closesocket
#define SHUT_RDWR SD_BOTH
#endif

#define DPIM_DEBUG // debug
```

```
#ifdef DPIM_DEBUG
#include <stdio.h>
#endif

//static SSL_CTX* ctx;
unsigned char __hellomsg[4]=
{0xCA,0xCA,DPIM_PROTO_VER_HIGH,DPIM_PROTO_VER_LOW};

char _msgbuf[65536];//

static struct DPIM_client_callbacks _cl_cbs;

static unsigned int procmsg_CONNREQINC(unsigned char*,unsigned int,void*);
static unsigned int procmsg_NOTFOUND(unsigned char*,unsigned int,void*);
static unsigned int procmsg_FOUND(unsigned char*,unsigned int,void*);

#define MSGINFO_COMM_MSGSNUM 3
struct DPIM_msginfo
_msginfo_comm[MSGINFO_COMM_MSGSNUM]={
{DPIM_PROTO_CONNREQINC,
DPIM_MSG_FIXED_SIZE,6,procmsg_CONNREQINC},
{DPIM_PROTO_NOTFOUND,
DPIM_MSG_FIXED_SIZE,6,procmsg_NOTFOUND},
{DPIM_PROTO_FOUND,
DPIM_MSG_FIXED_SIZE,6,procmsg_FOUND}
};

static int pem_passwd_cb(char *buf, int size, int rwflag, void *userdata)
{
strncpy(buf, (char *)(userdata), size);
buf[size - 1] = 0;
return(strlen(buf));
}

//функция возвращает номер клиента id в бинарном виде,
//извлекая его из строки,
//которая возвращается функцией X509_NAME_oneline
//и представляет собой DN из сертификата клиента
static int get_id_fromsubjstr(char* DN_str,unsigned int* id)
{
char* tmp1;
char* tmp2;

tmp1=strstr(DN_str,"/CN=");

if (!tmp1)
return 0;

tmp2=strchr(tmp1,'#');
```

```
if (!tmp2)
return 0;

*tmp2=0;

tmp1+=4;

if (tmp1==tmp2)
return 0;

*id=(unsigned int)strtol(tmp1,0,10);

*tmp2='#';
return 1;
}


static int add_chaninfo
(struct DPIM_conninfo *ci,struct DPIM_chaninfo **chn,unsigned int uid,int is_outgoing)
{
struct DPIM_chaninfo *chnl,*ipt;

chnl=*chn=(struct DPIM_chaninfo*)calloc(sizeof(struct DPIM_chaninfo),1);

if (!chnl)
return 0;//ERR_CHECK

chnl->conn=ci;
chnl->outgoing=is_outgoing;
chnl->peer_id=uid;

if (ci->chan_list_head)
{
ci->chan_list_end->next=chnl;
chnl->prev=ci->chan_list_end;
ci->chan_list_end=chnl;
}
else
{
ci->chan_list_head=chnl;
ci->chan_list_end=chnl;
}

return 1;
}

static int establish_data_connection(struct DPIM_chaninfo *chnl)
{
char* DN_str;
unsigned char buf[12];
```

```
int ret,err_ret;
unsigned int tmp_uid;


chnl->sck_data=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

{
struct sockaddr_in addr=chnl->conn->srv_addr;

addr.sin_port = htons(ntohs(addr.sin_port)+1);

ret=connect(chnl->sck_data,(struct sockaddr*)&addr,sizeof(struct sockaddr_in));

if (ret==SOCKET_ERROR)
{
err_ret=DPIM_ERRSOCK;
goto error_ret;
}
}
//а теперь надо отправить DPIM_PROTO_CONNID
buf[0]=buf[1]=DPIM_PROTO_CONNID;

if (chnl->outgoing)
{
*(unsigned int*)(buf+2)=chnl->conn->id;//вызвывающий
*(unsigned int*)(buf+6)=chnl->peer_id;//вызываемый
}
else
{
puts("callee CONNID");//debug
*(unsigned int*)(buf+2)=chnl->peer_id;//вызвывающий
*(unsigned int*)(buf+6)=chnl->conn->id;//вызываемый
}

ret=send(chnl->sck_data,buf,DPIM_PROTO_CONNID_SIZE,0);

if (ret!=DPIM_PROTO_CONNID_SIZE)
{
err_ret=DPIM_ERRSOCK;
goto error_ret;
}

chnl->ssl=SSL_new(chnl->conn->ctx);

if (!chnl->ssl)
{
puts("SSL_new !-  0");//debug
err_ret=DPIM_ERRSSL;
goto error_ret;
}
```

```
ret=SSL_set_fd(chnl->ssl,chnl->sck_data);

if (!ret)
{
puts("SSL_set_fd returned 0");//debug
err_ret=DPIM_ERRSSL;
goto error_ret_;
}

SSL_set_verify(chnl->ssl, SSL_VERIFY_PEER,0);

if (chnl->outgoing)
ret=SSL_connect(chnl->ssl);
else
ret=SSL_accept(chnl->ssl);

printf("verify result:%i\n",SSL_get_verify_result(chnl->ssl));//debug

if (ret!=1)
{
printf("ret=%i of SSL_connect or accept\n",ret);//debug
printf("SSL_get_error says:%i\n",SSL_get_error(chnl->ssl,ret));//debug
err_ret=DPIM_ERRSSL;
goto error_ret_;
}

//проверить сертификат собеседника

chnl->peer_cert=SSL_get_peer_certificate(chnl->ssl);

if (!chnl->peer_cert)
{
err_ret=DPIM_ERRNOPEERCERT;
goto error_ret_;
}

DN_str=X509_NAME_oneline(X509_get_subject_name(chnl->peer_cert), 0, 0);

if (!DN_str)
{
puts("no DN_str");//debug
err_ret=DPIM_ERRSSL;
goto error_ret_;
}

if (!get_id_fromsubjstr(DN_str,&tmp_uid)  || tmp_uid!=chnl->peer_id)
{
err_ret=DPIM_ERRINVPEERCERT;
goto error_ret_;
```

```
}
puts(DN_str);//debug

OPENSSL_free(DN_str);

chnl->state=DPIM_CHANSTATE_ACTIVE;

return DPIM_OK;

error_ret_:
error_ret:
_cl_cbs.channel_failed(chnl->peer_id);
DPIM_close_channel(chnl);
return err_ret;
}

static unsigned int procmsg_CONNREQINC
(unsigned char* buf,unsigned int nb,void* p)
{
int ret;
struct DPIM_conninfo *ci=(struct DPIM_conninfo *)p;


if (_cl_cbs.accept_cb(*(unsigned int*)(buf+2)))
{
struct DPIM_chaninfo *_chnl;

unsigned char sbuf[6]=
{DPIM_PROTO_CONNALLOW,DPIM_PROTO_CONNALLOW};
*(unsigned int*)(sbuf+2)=*(unsigned int*)(buf+2);
ret=SSL_write(ci->ssl_comm,sbuf,6);

if (ret!=6)
{
printf("ret=%i SSL_write ----\n",ret);//debug
return 0;
}
ret=add_chaninfo(ci,&_chnl,*(unsigned int*)(buf+2),0);

if (!ret)
return 0; //ERR_CHECK - кончилась память

ret=establish_data_connection(_chnl);

if (ret!=DPIM_OK)
return 0;

_cl_cbs.channel_established_cb(_chnl->peer_id);//*nchn=_chnl;

return 1;
```

```
}
else
{
unsigned char sbuf[6]=
{DPIM_PROTO_CONNDENY,DPIM_PROTO_CONNDENY};
*(unsigned int*)(sbuf+2)=*(unsigned int*)(buf+2);
ret=SSL_write(ci->ssl_comm,sbuf,6);
if (ret!=6)
return 0;//ERR_CHECK
}
}

static unsigned int procmsg_NOTFOUND
(unsigned char* buf,unsigned int nb,void* p)
{
struct DPIM_conninfo *ci=(struct DPIM_conninfo *)p;
struct DPIM_chaninfo *ipt,*rq=0;

puts("procmsg_NOTFOUND");//debug

ipt=ci->chan_list_end;
while(ipt)
{
if (ipt->peer_id==*(unsigned int*)(buf+2))
{
rq=ipt;
break;
}
ipt=ipt->prev;
}

if (!rq)
return 0;// ERR_CHECK -левое сообщение, такой запрос не посылался

puts("procmsg_NOTFOUND_chkp0");//debug
DPIM_close_channel(rq);

puts("procmsg_NOTFOUND_chkp1");//debug

_cl_cbs.notfound_cb(*(unsigned int*)(buf+2));
return 1;
}

static unsigned int procmsg_FOUND(unsigned char* buf,unsigned int nb,void* p)
{
struct DPIM_conninfo *ci=(struct DPIM_conninfo *)p;
struct DPIM_chaninfo *ipt,*rq=0;
int ret;

ipt=ci->chan_list_end;
```

```
while(ipt)
{
if (ipt->peer_id==*(unsigned int*)(buf+2))
{
rq=ipt;
break;
}
ipt=ipt->prev;
}

if (!rq)
return 0;// ERR_CHECK -левое сообщение, такой запрос не посылался

ret=establish_data_connection(rq);

if (ret!=DPIM_OK)
return 0;

_cl_cbs.channel_established_cb(rq->peer_id);//*nchn=_chnl;

return 1;

}

int DPIM_client_init(struct DPIM_client_callbacks *clcbs)
{
struct sockaddr_in sd;

if (!clcbs)
return 0;

_cl_cbs=*clcbs;

SSL_load_error_strings();
SSL_library_init();

#ifdef WIN32
WSADATA wsd;
WSAStartup(MAKEWORD(2,2),&wsd);
#endif

#ifdef NO_RAND_DEV
DPIM_rand_seed();
#endif

return 1;
}

/*адрес сервера, сертификат клиента и закрытый ключ*/
```

```
//Осуществляет подключение к серверу по указанному адресу.
//Аргументы:
//addr - адрес сервера (номер порта, указанный в структуре,
//относится к управляющему соединению.
//Для соединений, по которым передаются сообщения,
//используется порт с номером на единицу больше)
//cert_pkey - имя файла, который содержит сертификат
//клиента и его закрытый ключ в формате PEM
//password - пароль, которым зашифрован закрытый ключ
//ci -указатель на указатель на структуру DPIM_conninfo,
//который будет проинициализирован при успешном
//выполнении функции
//Память, выделенная под структуру, освобождается при
//вызове DPIM_close_connection
//root_cert - имя файла, содержащего корневой сертификат
//сети, к которой осуществляется подключение
//Возвращаемое значение - код завершения
//DPIM_OK - успешное завершение функции
//....
int DPIM_new_connection
(struct sockaddr_in *addr,char* cert_prkey,char* password,
struct DPIM_conninfo **ci,char* root_cert)
{
int ret,ret_err;
struct DPIM_conninfo *cid;


if (!(addr && cert_prkey && ci))
return DPIM_ERRINVARG;

cid=*ci=(struct DPIM_conninfo*)calloc(sizeof(struct DPIM_conninfo),1);
if (!cid)
return DPIM_ERRNOMEM;

cid->ctx=SSL_CTX_new(TLSv1_method());

if (!cid->ctx)
{
ret_err=DPIM_ERRSSL;
goto error_ret;
}
////<<
SSL_CTX_set_default_passwd_cb(cid->ctx, pem_passwd_cb);
SSL_CTX_set_default_passwd_cb_userdata(cid->ctx,password);

ret=SSL_CTX_use_certificate_file
(cid->ctx,cert_prkey,SSL_FILETYPE_PEM);
if (ret!=1)
{//ERR_CHECK
ret_err=DPIM_ERRSSL;
```

```
goto error_ret;
}
ret=SSL_CTX_use_PrivateKey_file
(cid->ctx,cert_prkey,SSL_FILETYPE_PEM);
if (ret!=1)
{//ERR_CHECK
ret_err=DPIM_ERRSSL;
goto error_ret;
}
////<<


ret=SSL_CTX_load_verify_locations(cid->ctx,root_cert,0/*"trustedCA/"*/);
printf("loadverifylocations ret=%i\n",ret);//debug
SSL_CTX_set_client_CA_list(cid->ctx,SSL_load_client_CA_file(root_cert));


{
char *DN_str;
X509 *x;
FILE *f;
f=fopen(cert_prkey,"r");
x=PEM_read_X509(f,0,0,0);
fclose(f);

DN_str=X509_NAME_oneline(X509_get_subject_name(x), 0, 0);

if (!get_id_fromsubjstr(DN_str,&cid->id))
{
ret_err=DPIM_ERRINVCERT;
goto error_ret;
}
printf("my id is %u\n",cid->id);//debug
puts(DN_str);//debug

OPENSSL_free(DN_str);

}

cid->ssl_comm=SSL_new(cid->ctx);

printf("1\n");//debug

if (!cid->ssl_comm)
{
ret_err=DPIM_ERRSSL;
goto error_ret;
}

cid->srv_addr=*addr;
```

```
cid->sck_comm=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (cid->sck_comm==SOCKET_ERROR)
{
ret_err=DPIM_ERRSOCK;
goto error_ret;
}

printf("2\ncid->sck_comm=%i\n",cid->sck_comm);//debug

ret=connect(cid->sck_comm,(struct sockaddr*)addr,sizeof(struct sockaddr_in));
printf("ret=%i\n",ret);//debug


if (ret==SOCKET_ERROR)
{
ret_err=DPIM_ERRSOCK;
goto error_ret;
}

printf("3\n");//debug

printf("sck=%i buf=%X \n",cid->sck_comm,__hellomsg);//debug
ret=send(cid->sck_comm,__hellomsg,4,0);

printf("ret=%i\n",ret);//debug

if (ret!=4)
{
free(cid);
*ci=0;
return DPIM_ERRSOCK;
}

printf("4\n");//debug

{//цикл чтения. Вынести в одну общую функцию?
unsigned int rsize=0xFFFFFFFF;

do
{
ret=recv(cid->sck_comm,cid->r_buf+cid->r_pos,128,0);
if (ret>0)
{
cid->r_pos+=ret;
if (rsize==0xFFFFFFFF && cid->r_pos>2)
{
if (cid->r_buf[0]==cid->r_buf[1])
{
switch (cid->r_buf[0])
```

```
{
case DPIM_PROTO_INITRESPOK:
rsize=3+cid->r_buf[2];
break;
case DPIM_PROTO_VERDIFFER:
rsize=4;
break;
default:
{
#warning "close! socket"
free(cid);
*ci=0;
printf("unknown message!\n");//debug
return DPIM_ERRSERV;
}
}
}
else
{
printf("invalid message!\n");//debug
#warning "close! socket"
free(cid);
*ci=0;
return DPIM_ERRSERV;
}
}
}
else
{
#warning "close! socket"
free(cid);
*ci=0;
return DPIM_ERRSOCK;
}
}while(cid->r_pos<rsize);

}
cid->r_pos=0;

printf("5\n");//debug

if (cid->r_buf[0]==DPIM_PROTO_VERDIFFER)
{
shutdown(cid->sck_comm,SHUT_RDWR);
close(cid->sck_comm);
free(cid);
*ci=0;
return DPIM_ERRVERDIFFER;
}
```

```
printf("6\n");//debug

ret=SSL_set_fd(cid->ssl_comm,cid->sck_comm);

printf("SSL_set_fd=%i\n",ret);

ret=SSL_accept(cid->ssl_comm);

if (ret!=1)
{
ret_err=DPIM_ERRSSL;
goto error_ret;
}


return DPIM_OK;

error_ret:
{
printf("SSL_free(%X) 2\n",cid->ssl_comm);//debug
SSL_free(cid->ssl_comm);////---!!!
SSL_CTX_free(cid->ctx);
free(cid);
*ci=0;
return ret_err;
}
}



//создает ИСХОДЯЩИЙ логический зашифрованный канал
//связи с указанным собеседником, если тот доступен
// в данный момент.

int DPIM_create_channel
(struct DPIM_conninfo *ci,unsigned int uid,struct DPIM_chaninfo **chn)
{
struct DPIM_chaninfo *chnl;
char* DN_str;
unsigned char buf[12];
int ret,err_ret;

if (!(ci && chn))
return DPIM_ERRINVARG;


if (!add_chaninfo(ci,chn,uid,1))
```

```
return DPIM_ERRNOMEM;

chnl=*chn;

buf[0]=buf[1]=DPIM_PROTO_CONNREQOUTG;
*(unsigned int*)(buf+2)=uid;

ret=SSL_write
(ci->ssl_comm,buf,DPIM_PROTO_CONNREQOUTG_SIZE);

if (ret!=DPIM_PROTO_CONNREQOUTG_SIZE)
{
//free(chnl);
return DPIM_ERRSSL;
}
ci->r_pos=0;

return DPIM_OK;
}

int DPIM_read(struct DPIM_conninfo *ci,int timeout)
{
fd_set fds;
struct timeval tv={0,1000};
struct DPIM_chaninfo *ipt;
int ret,nfds;

tv.tv_usec=timeout;

FD_ZERO(&fds);

nfds=ci->sck_comm;
FD_SET(ci->sck_comm,&fds);

ipt=ci->chan_list_head;

while(ipt)
{
if (ipt->state==DPIM_CHANSTATE_ACTIVE)
{
FD_SET(ipt->sck_data,&fds);

if (ipt->sck_data>nfds)
nfds=ipt->sck_data;
}

ipt=ipt->next;
}

ret=select(nfds+1,&fds,0,0,&tv);
```

```
if (ret>0)
{

if (FD_ISSET(ci->sck_comm,&fds))
{
ret=DPIM_read_conn(ci);
if (ret!=DPIM_OK)
return ret;
}

ipt=ci->chan_list_head;
while(ipt)
{
if (ipt->state==DPIM_CHANSTATE_ACTIVE)
{
if (FD_ISSET(ipt->sck_data,&fds))
{
unsigned int msglen;
ret=DPIM_read_msg(ipt,_msgbuf,&msglen);

switch(ret)
{
case DPIM_MSGUTF8:
_cl_cbs.newmsg_cb(ipt->peer_id,_msgbuf,msglen,DPIM_MSGUTF8);
break;
case DPIM_MSGOK:
_cl_cbs.msgok_cb(ipt->peer_id);
break;
default:
//ERR_CHECK
break;
}
}
}

ipt=ipt->next;
}

}

return DPIM_OK;
}

//функция читает и обрабатывает данные,
//приходящие от сервера по управляющему соединению.
int DPIM_read_conn(struct DPIM_conninfo *ci)
{
int ret;
```

```
ret=SSL_read(ci->ssl_comm,ci->r_buf+ci->r_pos,128-ci->r_pos);

if (!ret)
return DPIM_ERRSSL;

if (ret<0)
switch(SSL_get_error(ci->ssl_comm,ret))
{
case SSL_ERROR_WANT_WRITE:
case SSL_ERROR_WANT_READ:
break;
default:
return DPIM_ERRSSL;
}

if (ret>0)
{
ci->r_pos+=ret;

while(ci->r_pos)
{
unsigned int uret;
uret=DPIM_msggetsize
(ci->r_buf,ci->r_pos,_msginfo_comm,MSGINFO_COMM_MSGSNUM);

if (!uret)
return DPIM_ERRSERV;

if (uret > ci->r_pos)
break;

if (!DPIM_processmsg(ci->r_buf,ci->r_pos,
ci,_msginfo_comm,MSGINFO_COMM_MSGSNUM))
return DPIM_ERRSERV;

ci->r_pos-=uret;
memcpy(ci->r_buf,ci->r_buf+uret,ci->r_pos);
//сдвиг содержимого буфера к началу
}
}

return DPIM_OK;
}

int DPIM_send_msg
(struct DPIM_chaninfo *chn, char *buf,unsigned int len)
{
unsigned int n,slen=4;
int ret;
```

```
if (len>65535)
return DPIM_ERRMSGSIZE;

chn->msgbuf[0]=chn->msgbuf[1]=DPIM_PROTO_MSGUTF8;
*(unsigned int*)(chn->msgbuf+2)=len;// BIG_ENDIAN !!! ???

n=len;
if (n> (DPIM_CHNMSGBUFLEN-4))
n=DPIM_CHNMSGBUFLEN-4;

memcpy(chn->msgbuf+4,buf,n);

slen+=n;

ret=SSL_write(chn->ssl,chn->msgbuf,slen);

if (ret!=slen)
return DPIM_ERRSSL;

if (n!=len)
{
slen=len-n;
ret=SSL_write(chn->ssl,buf+n,slen);

if (ret!=slen)
return DPIM_ERRSSL;
}
return DPIM_OK;
}

//функция опроса наличия входящего сообщения
//Буфер должен иметь размер 64кб
//Возвращает кол-во полученных байтов сообщения
//через указатель n
// возвращаемое значение - тип полученного
//сообщения или код ошибки
int DPIM_read_msg
(struct DPIM_chaninfo *chn,char *buf,unsigned int *n)
{
int ret,br=0;
unsigned int msglen;

if (!buf)
return DPIM_ERRINVARG;

do
{
ret=SSL_read(chn->ssl,chn->msgbuf+br,4-br);
if (ret>0)
br+=ret;
```

```
else
return DPIM_ERRSSL;
}
while(br<4);

if (chn->msgbuf[0]!=chn->msgbuf[1])
return DPIM_ERRSERV;

puts("n read_msg");//debug

switch(chn->msgbuf[0])
{
case DPIM_PROTO_MSGUTF8:
msglen=(*(unsigned int*)(chn->msgbuf+2)) & 0xFFFF;
br=0;
do
{
ret=SSL_read(chn->ssl,buf+br,msglen-br);
if (ret>0)
br+=ret;
else
return DPIM_ERRSSL;
}
while(br<msglen);

*n=msglen;

chn->msgbuf[0]=chn->msgbuf[1]=DPIM_PROTO_MSGOK;

ret=RAND_bytes(chn->msgbuf+2, 2);

printf("RAND_bytes ret=%i\n",ret);

SSL_write(chn->ssl,chn->msgbuf,4);

return DPIM_MSGUTF8;
break;
case DPIM_PROTO_MSGOK:
return DPIM_MSGOK;
break;
//case ping
}

}

void DPIM_close_channel(struct DPIM_chaninfo *chn)
{
struct DPIM_chaninfo *ipt;

puts("DPIM_close_channel");//debug
```

```
ipt=chn->conn->chan_list_head;
puts("DPIM_close_channel_chkp0");//debug
while(ipt)
{
if (ipt==chn)
{
if (ipt->next)
ipt->next->prev=ipt->prev;
else
chn->conn->chan_list_end=ipt->prev;

if (ipt->prev)
ipt->prev->next=ipt->next;
else
chn->conn->chan_list_head=ipt->next;

break;
}
ipt=ipt->next;
}

puts("DPIM_close_channel_chkp1");//debug

if (chn->peer_cert)
X509_free(chn->peer_cert);

puts("DPIM_close_channel_chkp2");//debug

if (chn->ssl)
SSL_shutdown(chn->ssl);
puts("DPIM_close_channel_chkp3");//debug

if (chn->ssl)
{
printf("3 SSL_free(%X)\n",chn->ssl);//debug
SSL_free(chn->ssl);
}

puts("DPIM_close_channel_chkp4");//debug
shutdown(chn->sck_data,SHUT_RDWR);
puts("DPIM_close_channel_chkp5");//debug
close(chn->sck_data);
puts("DPIM_close_channel_chkp6");//debug
free(chn);
puts("DPIM_close_channel_chkp7");//debug
}

void DPIM_close_connection(struct DPIM_conninfo *cid)
{
```

```
struct DPIM_chaninfo *ipt;

ipt=cid->chan_list_head;

while(ipt)
DPIM_close_channel(ipt);

SSL_shutdown(cid->ssl_comm);
printf("4 SSL_free(%X)\n",cid->ssl_comm);//debug
SSL_free(cid->ssl_comm);
shutdown(cid->sck_comm,SHUT_RDWR);
close(cid->sck_comm);
SSL_CTX_free(cid->ctx);
free(cid);
}
```