**BC/NW 2021№ 2 (37)**

**ИССЛЕДОВАНИЕ ХАРАКТЕРИСТИК ПРОТОКОЛА ПЕРЕДАЧИ ДАННЫХ В КОРПОРАТИВНОЙ СЕТИ**

**Бородулин С.В.**

Использование автоматизированных систем контроля (СК) окружающей среды становится популярным в связи с глобальной цифровизацией многих сфер жизни человека. Для того, чтобы обеспечить необходимые условия для функционирования систем контроля активно разрабатываются и применяются устройства и сценарии, способные организовать этот процесс.

***Постановка задачи***

Целью исследования протокола передачи данных *MQTT* и его характеристик путём реализации безопасного соединения и оптимизации взаимодействия между узлами в корпоративной сети, а также сравнения и выбора параметров для достижения необходимого результата передачи данных.

В качестве характеристик протокола, от которых зависят критерии эффективности – производительности и надёжности – рассмотрены:

* Сквозная задержка пакетов (англ. *End-to-end Delay*);
* Количество переданных в ходе одного цикла байт информации.

Задача настоящей работы заключается в исследовании протокола передачи данных, который является компактным, надёжным, масштабируемым и легко внедрялся в существующие сети. Для этой задачи был выбран протокол *MQTT*.

Для решения поставленной задачи потребовалось составить сценарий применения протокола, после чего реализовать его с применением реального сетевого оборудования для проведения анализа и оценки свойств протокола.

Итогом работы явился рабочий стенд для реализации безопасной передачи данных и результаты измерения количества переданных пакетов и байт информации, в связке со значениями сквозной задержки.

Структура работы строится следующим образом:

* В первом разделе работы поставлена цель, сценарий и методы их реализации.
* Во втором разделе представлено техническое обоснование используемых технологий, а именно, описание используемой сети, применяемых протоколов и их взаимодействие между собой.
* В третьем и четвёртом разделах описана реализация стенда и проведение на нём экспериментов.

Актуальной задачей СК является сбор данных с различных устройств и датчиков для соблюдения оптимальных рабочих условий оборудования. Получаемые с датчиков данные позволяют в режиме реального времени отслеживать корректность условий поддержания, а также прочих данных с целью дальнейшего анализа.

Чёткое определение требуемых данных и их соединение с сетевым миром данных являются основой для надёжного обмена данными по сети. При этом, важную роль играет выбор подходящего протокола для реализации передачи данных.

В статье представлена концепция получения данных от датчиков температуры, датчика влажности и специального устройства. Реализация происходит с использованием реального сетевого оборудования, а роль каждого из устройств выполняют настроенные персональные компьютеры. Передаваемые данные поступают в единый центр обработки данных – на выделенный компьютер. Таким образом, объединение устройств в систему сбора и передачи данных позволяет:

* Оперативно получать информацию об изменении ключевых параметров;
* Оптимизировать сбор и обработку данных;
* Фиксировать факты нецелевой работы в случае отклонения параметров от нормы;
* Оперативно реагировать на внештатные или чрезвычайные ситуации.

В ходе исследования происходит подготовка и настройка сетевого оборудования *CISCO*, на персональные компьютеры под управлением операционной системы *Debian* в сети устанавливается программное обеспечение для использования протокола *MQTT*. После подготовки оборудования к работе посредством экспериментов исследуются и оцениваются характеристики протокола в зависимости от установленных параметров протокола, а именно:

* версии протокола,
* качества обслуживания,
* наличия сертификатов для безопасной передачи данных.

***Методы реализации***

Для достижения поставленной цели необходимо выполнить исследование особенностей протокола *MQTT* и его алгоритмов работы, а также разработать полноценный сценарий применения протокола, реализующий взаимодействие устройств в корпоративной сети автоматизированным способом.

Для достижения поставленной цели были поставлены и решены следующие задачи:

* описание сценария взаимодействия устройств в корпоративной сети;
* анализ и сравнение протоколов передачи данных, их оценка;
* развёрнутое описание выбранного протокола;
* описание программного обеспечения (ПО) для тестовой отладки работы протокола в среде виртуализации;
* настройка сетевого оборудования;
* исследование характеристик протокола передачи данных путём проведения экспериментов с изменением его параметров;
* сравнение ожидаемых и полученных результатов.

В статье исследуется корпоративная сеть АС, организованная на базе коммутатора *CISCO Catalyst* серии 3560 [4], реализующая логическом уровне объединение подсетей, состоящих из роутеров и компьютеров. Каждая из подсетей состоит из соединённых между собой роутера и компьютера за исключением одного компьютера в центре сети, единственной функцией которого является маршрутизация трафика.

На рисунке 1. представлена логическая схема корпоративной сети.

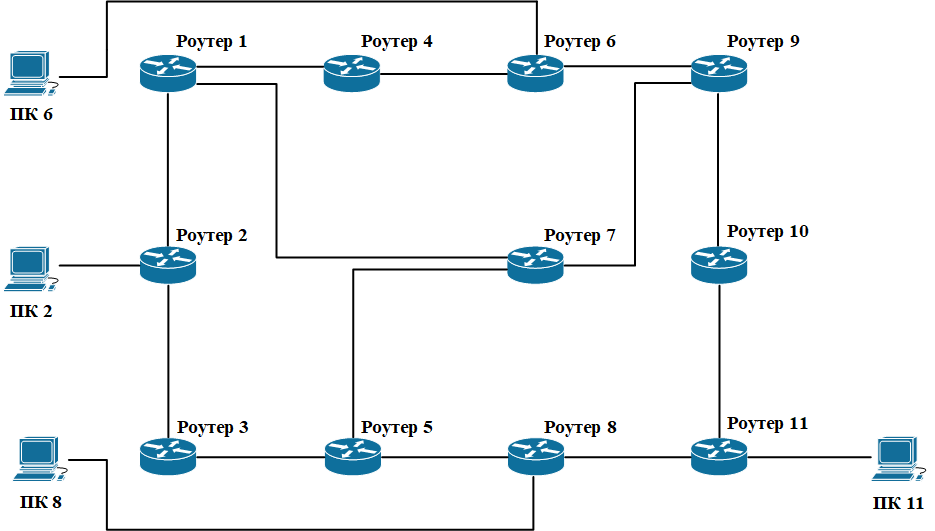


Рис. 1. Логическая схема корпоративной сети

Представленная схема сети является базой для применения протокола в большой сети.

***Модель стека протоколов TCP / IP***

*TCP / IP* – это стандарт для построения глобальных сетей, включающий в себя совокупность основных протоколов сети Интернет для связи компьютеризированных устройств, известный также как стек *TCP / IP*. Он используется и в корпоративных сетях. Структура сетевых протоколов организована таким образом, что каждый протокол принадлежит определённому уровню [5], каждый из которых может быть реализован в программном или аппаратном обеспечении. Также возможна и их комбинация.

На прикладном уровне обеспечивается обслуживание множества конечных систем при помощи порций данных, называемых сообщением [5]. Протоколами прикладного уровня являются: *HTTP, MQTT, Telnet, SMTP, FTP* и другие.

На транспортном уровне осуществляется непосредственно передача порций данных, образующих сообщения, называемых сегментами. Обмен данными происходит между конечными приложениями. Протоколами транспортного уровня являются: *TCP* и *UDP*.

Передача блоков информации, называемых дейтаграммами, между хостами в сети происходит на сетевом уровне. Кроме *IP*, другими протоколами сетевого уровня являются*: ICMP, ARP, RIP* и другие.

На нижнем уровне – сетевых интерфейсов – обеспечивается поддержка всех популярных стандартов физического и канального уровней для осуществления устройствам доступа к сети. К протоколам уровня сетевых интерфейсов относятся: *Ethernet, FDDI, ATM, PPP, Wi-Fi* и прочие.

*Криптографические протоколы SSL и TLS*

*SSL (Secure Sockets Layer)* и *TLS (Transport Layer Security)* – это семейство протоколов, которые созданы для того, чтобы обеспечивать безопасные соединение и передачу данных в небезопасной сети [9], к примеру, в сети *Internet*.

*SSL* – это протокол, работающий на уровне защищённых сокетов, разработанный *Netscape* в 1994 году, однако выпущенный только в 1995 году [11] для безопасного обмена информацией между *WEB*-сервером и браузером по незащищённым сетям.

Протокол *TLS* был построен на базе протокола *SSL* версии 3.0, однако в связи с внесёнными изменениями, протоколы *SSL* и *TLS* стали несовместимыми. Именно из-за этой несовместимости, большинство устройств прикладного уровня применяют оба протокола. Протокол *TLS* способен преобразовываться в протокол *SSL*, и наоборот. По этой причине возможно обозначение как *TLS* / *SSL*, так и *SSL* / *TLS*.

На сегодняшний день действующими являются версии *TLS* 1.2 и *TLS* 1.3. Версии *TLS* 1.1, *TLS* 1.0 и все версии *SSL* считаются устаревшими. В исследовании используется *TLS* 1.2.

*TLS* – это протокол для защиты передаваемых данных, работающий на транспортном уровне модели *TCP* / *IP*. Для передачи данных по незащищённой сети *TLS* и *SSL* используют протокол *TCP*. То есть, если в классической схеме протоколы прикладного уровня работают напрямую с протоколами транспортного уровня *TCP*, то протокол *TLS* поверх *TCP* обеспечивает защищённое соединение. Именно с таким защищённым соединением работают протоколы прикладного уровня, например протоколы *HTTP* и *MQTT*. Также на прикладном уровне поверх *TLS / SSL* работает большое количество других протоколов.

Задача *SSL* заключается в поддержке шифрования во время сессии после установки защищённого соединения.

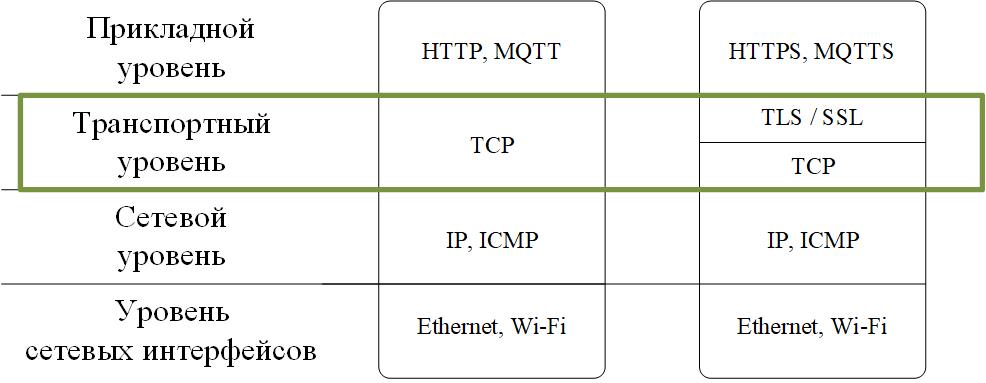


Рис. 2. Протоколы *TLS / SSL* в модели *TCP / IP* [15]

В свою очередь, *TLS* также состоит из двух уровней. Нижний уровень – протокол записей (англ. *Record Protocol*) – задаёт формат, в котором данные будут передаваться по сети. Верхний уровень содержит 4 протокола определяющих передаваемое содержание. Им являются протокол установки *TLS*-соединения между клиентом и сервером (англ. *Handshake Protocol*), протокол оповещения в случае возникновения ошибки или некорректной работы (англ. *Alert Protocol*), протокол смены шифра для перехода к передаче данных в зашифрованном виде (англ. *Change Cipher Protocol*) и протокол передачи данных в зашифрованном виде (англ. *Application Data Protocol*) от вышестоящего протокола, к примеру, протокола *MQTT*.

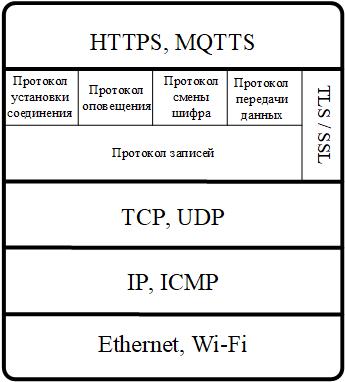


Рис. 2.Уровни *TLS* [15]

Протокол записей является основой *TLS*. В нём реализована большая часть операций по защите данных – шифрование и проверка целостности. В его записи вкладываются сообщения вышестоящих протоколов *TLS*, таких как *Handshake*-протокол или протокол передачи данных. Записи *Record*-протокола вкладываются в сегменты *TCP* для передачи по сети.

Формат сообщения протокола записи *TLS* состоит из нешифруемого заголовка, включающего три поля – тип сообщения, версию протокола и длину сообщения, и зашифрованных передаваемых данных.



Рис. 3. Формат сообщения протокола записи *TLS* [15]

Для сравнения, относительно модели взаимодействия открытых систем *OSI*, расположение *TLS / SSL* соответствует трём уровням – транспортному, сеансовому и представления.

На транспортном уровне осуществляется передача данных непосредственно между процессами различных хостов в сети. На сеансовом уровне происходит восстановление сеанса шифрования, который был установлен ранее для повышения производительности. В связи с тем, что установка соединения и обмен ключами шифрования занимает достаточно длительное время и требует высоких вычислительных ресурсов, в протоколе *TLS* предусмотрена возможность использовать ключи из этого соединения. На уровне представления реализуется непосредственно шифрование данных. Таким образом, в сеть передаются данные не в том виде, в котором они поступают от протокола прикладного уровня, а в зашифрованном. Это шифрование обеспечивает именно протокол *TLS* / *SSL*.



Рис. 4. Протоколы *TLS* / *SSL* в эталонной модели *OSI* *[]*

*Безопасность*

В настоящее время безопасность передачи данных является очень актуальным вопросом. Важным фактором является не только передача зашифрованных сообщений, которые во время пересылки практически невозможно расшифровать посторонним, но и удостовериться, что данные отправляются нужному адресату. Таким образом, безопасность передачи данных включает в себя 3 аспекта:

* приватность данных,
* целостность,
* аутентификация.

Для осуществления каждого из этих аспектов протоколы *TLS* / *SSL* используют такие технологии как симметричное и асимметричное шифрование, криптографические хэш-функции, электронную подпись совместно с сертификатами открытого ключа.

Протокол *TLS* комбинирует технологии для обеспечения безопасной передачи данных по сети. При гибридном шифровании в *TLS* / *SSL* асимметричное шифрование используется для передачи симметричного ключа, а симметричное шифрование – непосредственно, для передачи данных.

Для алгоритма обмена ключами в *TLS* применяются шифры Диффи-Хеллмана и *RSA*. Наборы шифров *AES*, 3*DES* и другие используются для реализации симмеричного шифрования.

Во время установки соединения *TLS* происходит вначале выбор поддерживаемого набора шифров *TLS*, после чего происходит обмен ключами для симметричного шифрования.

Именно за счёт применения гибридного шифрования обеспечивается соврешенная прямая секретность, которая гарантирует невозможность расшифровать данные при получении доступа к серверу.

*Целостность*

Не смотря на то, что изменение передаваемых зашифрованных данных сложно, оно всё же возможно. Оно возможно как при попытке подключения злоумышленника к каналу связи между клинтом и сервером и изменнием им этих данных, так и при возникновении ошибок в канале связи. Для предотвращения каких-либо изменений в процессе передачи, необходимо обеспечить целостность данных.

Для обеспечения целостности в *TLS* / *SSL* используются криптографические хэш-функции. Они предназначены для преобразования массива данных в строку фиксированной длины, называемой хэш. Криптографическими хэш-функциями, реализуемыми в протоколе являются *MD*5 (*Message* *Digest* 5) и алгоритм *SHA* (*Secure* *Hash* *Algoritm*) с различной длиной хэша, к примеру, *SHA*-384 и *SHA*-512.

Таким образом, при передаче данных, клиентом рассчитывается хэш при помощи криптографической функции, который передаётся вместе с этими данными. При получении сервер извлекает данные, рассчитвает хэш и сверяет его с хэшем из присланного пакета. Совпадение хэшей означает целостную передачу, а разное значаение, соответственно, говорит о том, что данные в процессе передачи были изменены. При этом у клиента и сервера имеется разделяемый ключ, который служит для исключения предумышленного изменения данных в процессе передачи.

***Установка соединения***

Для того, чтобы в *TLS* можно было передавать данные с помощью протокола записи, необходима установка сессии. Она происходит, как правило, между клиентом и сервером. Им необходимо назвать о наборе используемых шифров в процессе передачи данных по соединению *TLS*, иметь одинаковые разделяемые ключи для симметричного шифрования и для *MAC*.

Так как процесс установки соединения достаточно ресурсозатратный, в протоколе предусмотрена возможность возобновления ранее установленного соединения по идентификатору сессии с использованием уже имеющихся разделяемых ключей.

На рисунке 5. представлен этап установки защищённого соединения – рукопожатие *TLS / SSL (*англ*. Handshake TLS / SSL)*

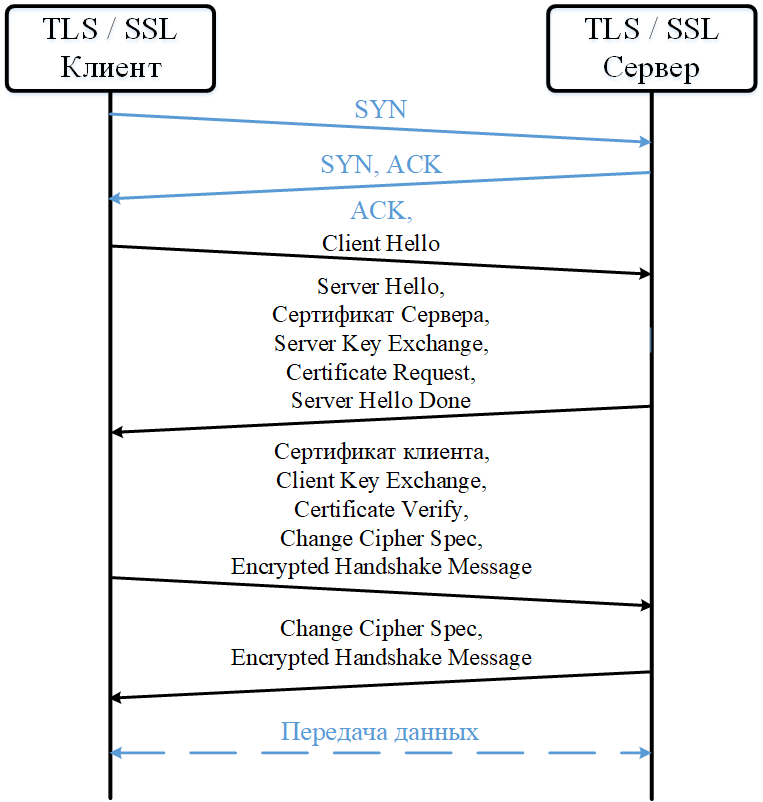


Рис. 5. Установка защищённого соединения *TLS / SSL*

*Аутентификация*

Аутентификация, или, другими словами, процедура установления подлинности, с применением цифровых сертификатов является альтернативой использованию паролей. При использовании сертификатов информация о пользователях не видна в сети передачи данных, она предоставляется в запросах в виде сертификатов, удостоверяющих личность пользователей. Причём к пользователю может относиться как клиент, так и сервер. Цифровой сертификат необходим для того, чтобы установить доверенное соединение между клиентом и сервером и обеспечить аутентификацию.

Аутентификация предполагает использование электронной подписи или сертификата. В то время как электронная подпись подразумевает и использование закрытого ключа для подписи информации и открытого ключа для доступа к данной информации. Для идентификации открытого ключа используется инфраструктура открытых ключей, распространяющая их с помощью удостоверяющих центров.

Удостоверяющий центр выдаёт узлам сети сертификаты, которые представляют собой файлы специального вида, содержащие несколько компонентов, одним из которых является открытый ключ. Для подтверждения подлинности ключ в сертификате, удостоверяющий центр подписывает сертификат своим приватным ключом. Если клиент доверяет удостоверяющему центру, то у него есть открытый ключ этого центра. Таким образом, при передаче сервером клиенту сертификата, последний может расшифровать электронную подпись своим открытым ключом.

Сертификатом является файл, в котором содержится следующая информация:

* открытый ключ владельца данного сертификата;
* сведения о владельце сертификата – сервере либо пользователе,
* наименование сертифицирующей организации, выдавшей данный сертификат.

Сертификаты выдаются, или, другими словами, подписываются, специальными уполномоченными организациями — центрами сертификации (англ. *Certificate Authority, СА*). Центр сертификации – это доверенная обоими клиентами – сервером и пользователем – сторона, которая выдаёт каждой из них сертификаты. Поэтому задача хранения секретной информации, такой как приватные ключи, возлагается на самих пользователей. Это решение является гораздо более масштабируемым, чем использование централизованной базы паролей.

В связи с тем, что для обеспечения огромным количеством сертификатов как пользователей в сети *Internet*, так и в других менее масштабных сетях, одного удостоверяющего центра мало, и используется инфраструктура сертификатов и открытых ключей, состоящая из удостоверяющих центров, объединённых в иерархию. На вершине иерархии находится корневой удостоверяющий центр, который не выдаёт сертификаты узлам, однако с ним работают другие удостоверяющие центры, которых может быть несколько. Клиенты для получения сертификатов обращаются именно к ним. Иерархия может иметь глубину более одного уровня. На рисунке 6 приведена обобщённая иерархическая инфраструктура.

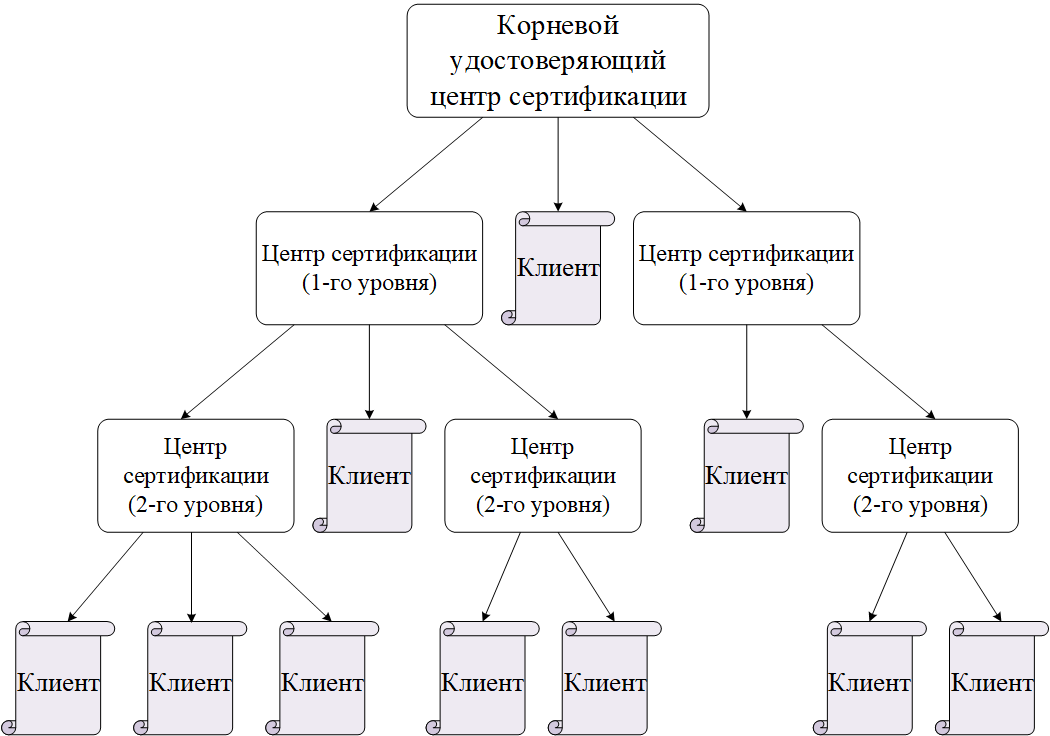


Рис. 6. Обобщенная схема иерархической инфраструктура управления сертификатами с глубиной иерархии = 2

Для подписывания сертификатов используется цепочка доверия или, другими словами, путь сертификации. Для проверки надёжности сертификата клиенту необходимо пройти по всей цепочке доверия. Для обеспечения взаимодействия всех конечных клиентов должен быть только один корневой доверенный центр.

Форматом сертификата является *X*.509. При этом срок действия сертификата любого уровня из соображений безопасности является конечным.

*Виды аутентификации*

Аутентификация данных означает доказательство их подлинности, то есть их неизменность и подтверждённость адресата. Существует односторонняя и двусторонняя аутентификация. Для односторонней аутентификации предусматривается, что если клиент доверяет серверу и получает сертификат с его стороны, то с помощью полученного сертификата появляется возможность установить контакт. Другими словами, клиент сообщает серверу свой идентификатор и посредством сертификата проходит аутентификацию для входа в защищённую систему.

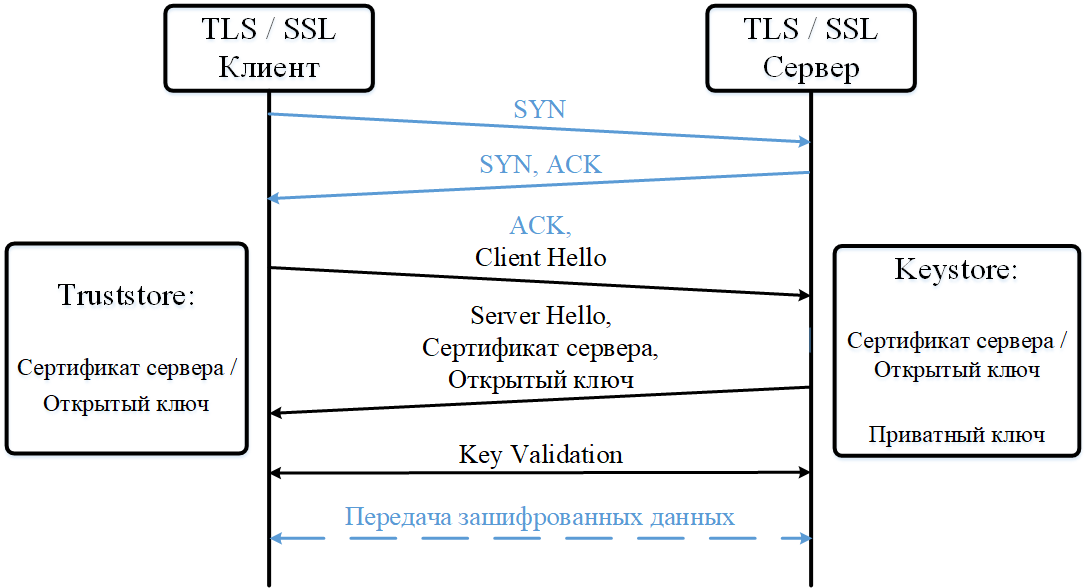


Рис. 7 Односторонняя аутентификация [17]

Сертификаты и ключи хранятся в виде файлов в хранилищах обеих сторон. Существует два различных типа хранилищ – *Keystore* и *Truststore* [19]. В *Keystore* содержится более чувствительная информация, а именно приватный ключ собственного сертификата. В *Truststore* хранится информация о доверенных сертификатах, которые в том числе могут присутствовать в операционной системе

В случаях, когда односторонней аутентификации недостаточно, используется двусторонняя. При двусторонней аутентификации помимо установления доверенности сервера со стороны клиента, устанавливается также и достоверность клиента со стороны сервера. То есть сервер отправляет свой сертификат, подписанный центром сертификации, с целью установления аутентификационных данных клиента, а клиент отправляет свой сертификат, также подписанный центром сертификации, с целью установления аутентификационных данных сервера.

Когда пользователь хочет подтвердить свою личность, он предъявляет свой сертификат в двух формах — открытой, в которой он получил его в сертифицирующей организации, и зашифрованной с применением своего закрытого ключа. Сторона, проводящая аутентификацию, берет из открытого сертификата открытый ключ пользователя и расшифровывает с помощью него зашифрованный сертификат. Совпадение результата с открытым сертификатом подтверждает факт, что предъявитель действительно является владельцем закрытого ключа, парного с указанным открытым.

Оба сертификата выдаёт доверенный центр.

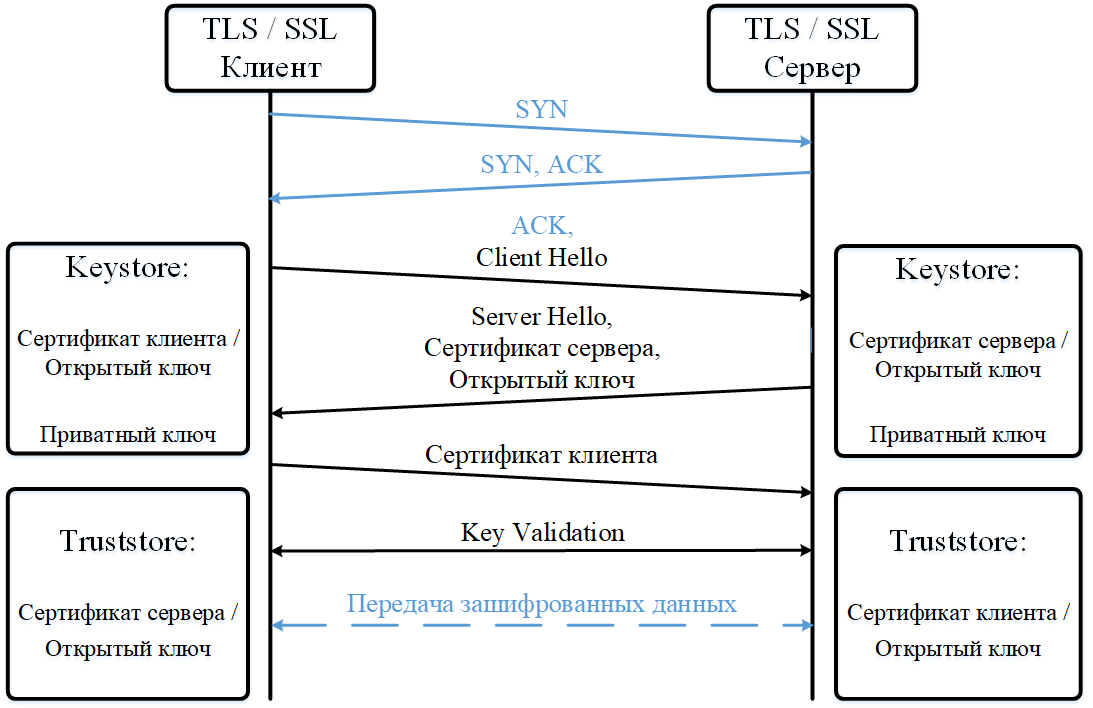


Рис. 8. Двусторонняя аутентификация [17]

Сертификаты могут быть приобретёнными либо самоподписанными, то есть созданными самостоятельно. Для более гибкой настройки в исследовательской диссертации используется самоподписанные сертификаты.

*Самоподписанные сертификаты*

Самоподписанный сертификат – это сертификат открытого ключа, подписанный собственным же приватным ключом самого пользователя, который его издал, без обращения для подписи к удостоверяющему центру. Такие сертификаты могут издаваться, к примеру, на стороне сервера.

В то же время все сертификаты корневых удостоверяющих центров являются самоподписанными. Никакой из других удостоверяющих центров не может подписать эти сертификаты, потому что корневой центр находится на вершине иерархии. Однако эти сертификаты установлены часто в хранилище доверенных сертификатов операционных систем и поэтому являются доверенными.

Существуют различные способы создания самоподписанных сертификатов. Наиболее популярными являются инструменты *OpenSSL* и *LibreSSL*.

Процесс создания включает в себя генерирование следующей серии сертификатов и ключей:

* корневые (центра сертификации) сертификат и ключ,
* сертификат и ключ сервера,
* сертификат и ключ клиентов.

Корневой сертификат является отправной точкой всей цепочки доверия. Если эмитенту сертификата каждого уровня и эмитенту корневого сертификата доверяют, то этот сертификат является доверенным.

Вначале необходимо создать корневой сертификат. Для этого генерируется приватный ключ, который далее используется для создания корневого сертификата. При создании также необходимо указать срок действия сертификата.

Далее серверу необходимо сгенерировать собственный приватный ключ для обеспечения контроля над своими сертификатами. При помощи ключа будет создан запрос в корневой центр на создание сертификата для сервера с указанием контактных данных создателя сертификата сервера. После подтверждения центром сертификации, им будет создан сертификат для сервера также на указанный срок.

Процесс создания сертификата для клиента с использованием центра сертификации аналогичен созданию сертификата для сервера. После генерации клиентом приватного ключа, создаётся запрос в центр сертификации для создания сертификата. В нём указываются контактные данные создателя сертификата клиента. Далее центр сертификации подтверждает создание сертификата и создаёт его на указанный срок.

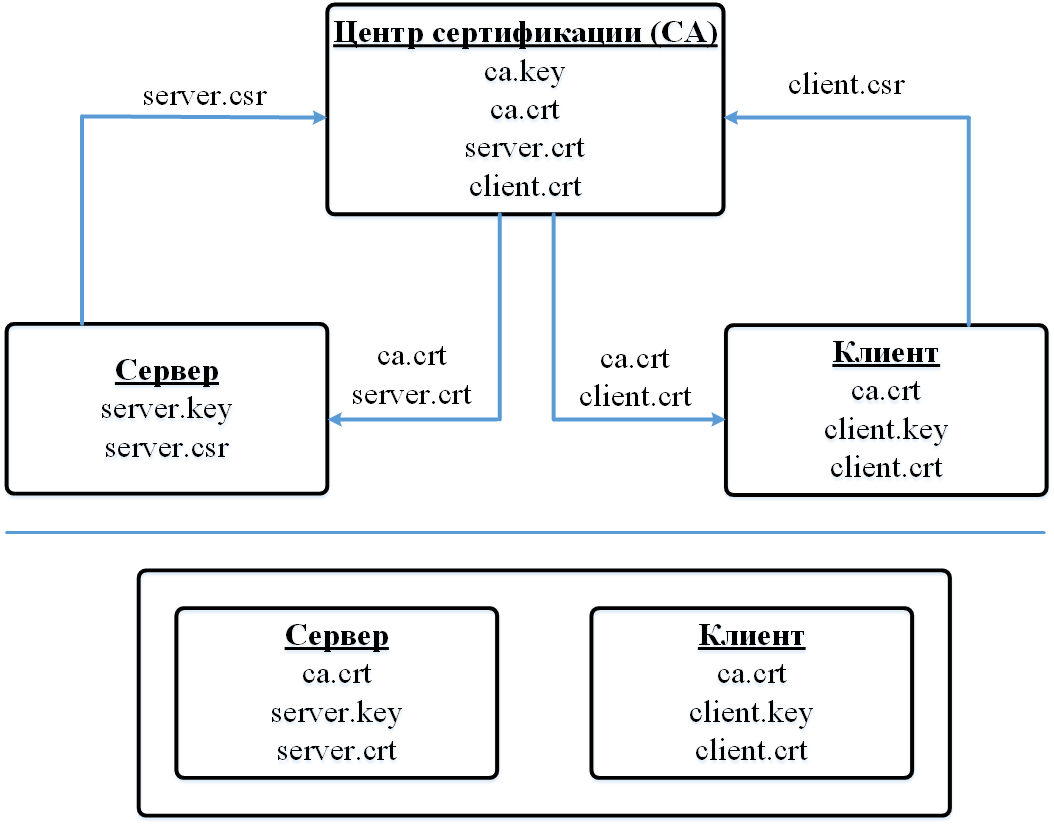


Рис. 9. Создание и распределение самоподписанных сертификатов

Таким образом, у сервера и у клиента имеется набор из собственных приватного ключа и сертификата, подписанных в одном центре сертификации, которому доверяют обе стороны.

Во время установки защищённого *TLS*-рукопожатия и дальнейшей коммуникации клиент и сервер устанавливают связь, используя полученные сертификаты.

*Протокол* *MQTT*

На сегодняшний день существует огромное количество протоколов, позволяющих подключать различные устройства к сети *Internet*, а также коммуницировать между собой. Для постоянной коммуникации устройств между собой и шлюзами часто используются специальные протоколы, обеспечивающие надёжность передачи данных, масштабируемость, а также низкое энергопотребление. Основываясь на научной работе автора [1], для реализации сценария был выбран протокол передачи данных *MQTT*.

*MQTT* (*Message* *Queuing* *Telemetry* *Transport*) – это простой и легковесный протокол обмена сообщениями, функционирующий согласно модели передачи сообщений «издатель-подписчик» (англ. *Publish*-*Subscribe*). При этом протоколом определяется формат сообщения, но не его содержимое. Он предназначен для обмена информацией между разными компьютеризированными устройствами, такими как датчики и модули, подключёнными к локальной или глобальной сети.

Простота в использовании, невысокая нагрузка на каналы связи за счёт работы поверх *TCP / IP*, поддержка работы в условиях постоянной потери связи и лёгкая встраиваемость в любую систему сыграли решающую роль при выборе протокола коммуникации.

*Модель «издатель-подписчик»*

Поведенческий шаблон проектирования «издатель-подписчик», лежащий в основе работы протокола *MQTT,* подразумевает под собой передачу сообщений между клиентами, которые, в свою очередь, подразделяются на издателей и подписчиков. В задачу издателей сообщений входит отправка данных, а подписчиков – принятие этих данных. Коммуникация между устройствами-клиентами происходит не напрямую, а с использованием третьего узла – брокера сообщений, выполняющего роль сервера. Брокер может представлять собой одиночный узел или может состоять из нескольких узлов, образующих кластер брокеров, который на логическом уровне для издателей и подписчиков будет являться единым центром.

В названной модели подписчики обычно получают некоторое подмножество всех опубликованных сообщений. Процесс отбора сообщений для получения и их обработка называется фильтрацией. Существуют две основных формы фильтрации: основанная на теме (англ. «*topic*») и основанная на содержимом.

Модель, применяемая в *MQTT*, реализует фильтрацию по темам и осуществляет передачу сообщений по следующему принципу: издатель публикует сообщение, в котором указывает тему, или, другими словами, топик, и оно отправляется на брокера. Если подписчики посредством предварительного обращения к брокеру подписаны на этот топик, то они получают опубликованное сообщение. гибкая настройка тем позволяет одновременно подписываться на несколько тем и повысить гибкость тем, упрощая управление ими.

Схема описанного паттерна проектирования без учёта иерархии тем представлена на рисунке 10.

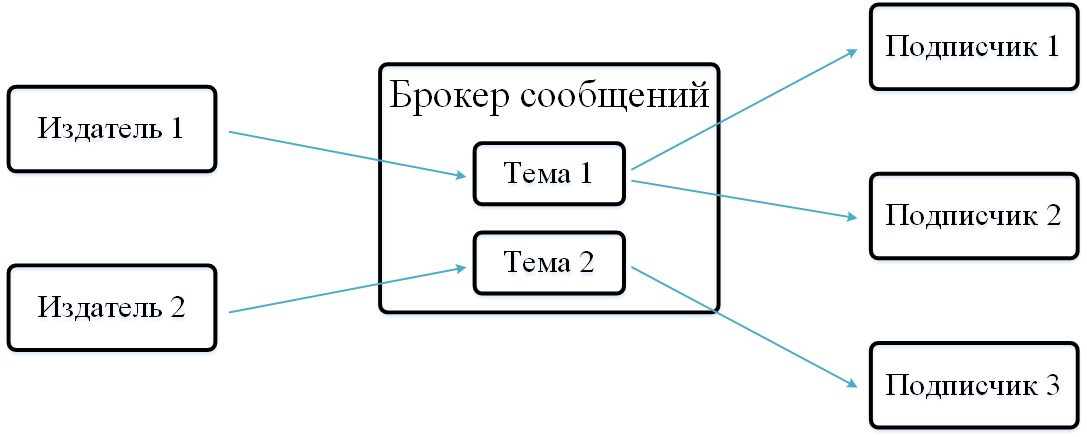


Рис. 10. Модель «издатель-подписчик»

*Брокер сообщений*

Брокер сообщений представляет собой узел, обеспечивающий взаимодействие клиентов, а именно, на который приходят сообщения от издателя и отправляются подписчикам. Узел может представлять собой одиночный брокер или кластер брокеров. В задачи брокера входит:

* получение данных от клиентов,
* обработка и хранение данных,
* доставка данных клиентам,
* контроль за доставкой сообщений.

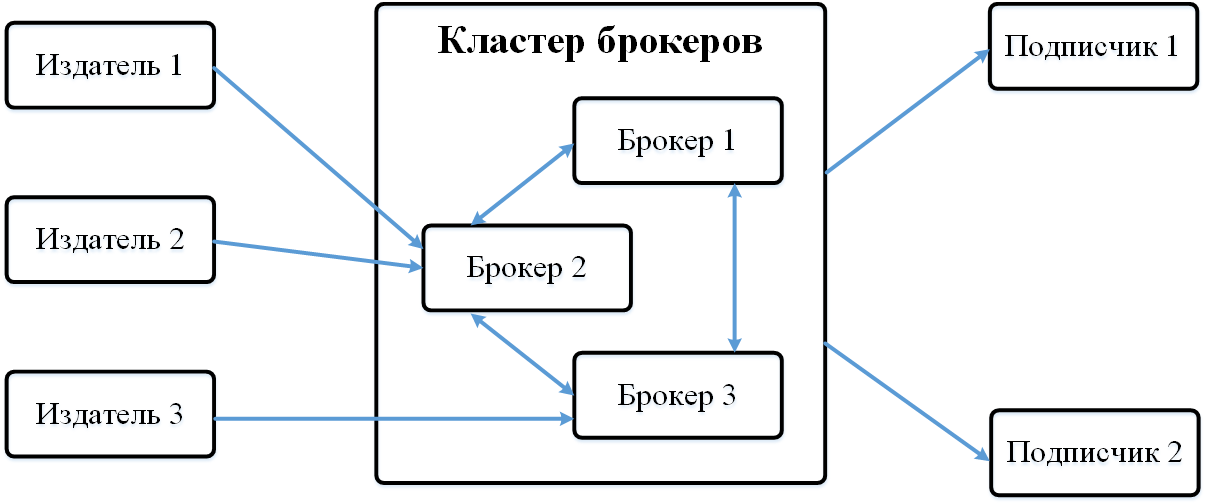


Рис. 11. Коммуникация между издателями и подписчиками через кластер брокеров

Существуют различные реализации для брокера сообщений. Самыми распространёнными являются брокер для локальной сети и облачное решение. Обе реализации зарекомендовали себя, однако они задействуют разные технологии. Для передачи данных в локальной сети применяется, в свою очередь, протокол *MQTT*, а для облачных решений передача происходит посредством протокола *WebSocket*. В настоящей работе используется первый вариант.

При стандартной настройке *MQTT*-брокер для коммуникации использует порт 1883. При использовании защищенного *SSL*-подключения задействуется порт 8883.

*Параметры протокола* *MQTT*

Протокол *MQTT* разрабатывался максимально лёгким для маломощных устройств с ограниченными ресурсами и сетей с ограниченной пропускной способностью. Поэтому в нём присутствуют механизмы по обеспечению гарантированной доставки сообщений даже в случае потери пакетов и обрывов связи.

К главным особенностям протокола *MQTT* относятся [1]:

* Асинхронность, позволяющая обслуживать большое количество устройств и не зависеть от сетевых задержек;
* Компактность и легковесность сообщений, позволяющие минимизировать расходы на пересылку данных;
* Устойчивость к потерям данных во время пересылки, что позволяет гарантировать доставку в условиях нестабильных сетевых подключений;
* Поддержка разных уровней качества обслуживаний – *Quality of Service* *(QoS)*, что добавляет возможность управлять приоритетом сообщений и реализовать гибкую гарантию доставки сообщений в зависимости от требований;
* Лёгкая интеграция, не требующая больших затрат и позволяющая развернуть систему практически в любой сети.

Для организации коммуникации необходимо, чтобы издатель и подписчик знали, какая информация отправляется и получается.

Согласно ценности и приоритету доставки, за счёт флагов протокол *MQTT* поддерживает 3 уровня *QoS*:

* *QoS* = 0 – сообщение будет отправлено один раз без подтверждения доставки;

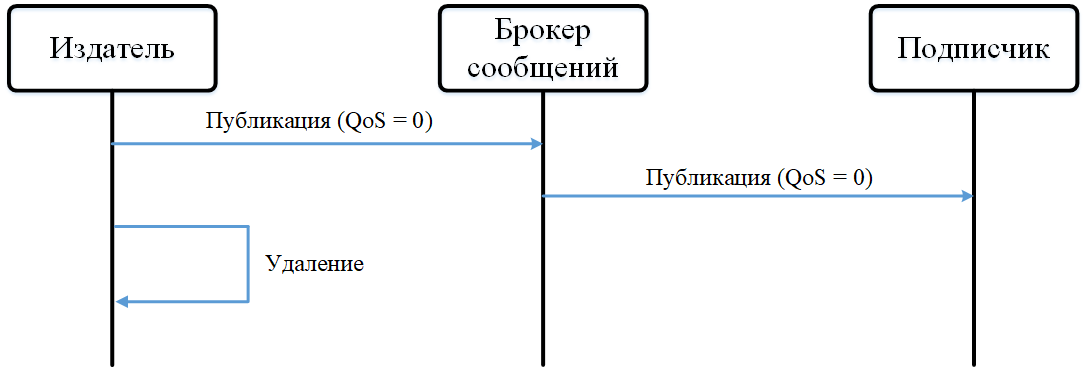


Рис. 12. Доставка сообщение при уровне *QoS* = 0 [1]

* *QoS* = 1 – сообщение будет отправлено минимум один раз с подтверждением о доставке;

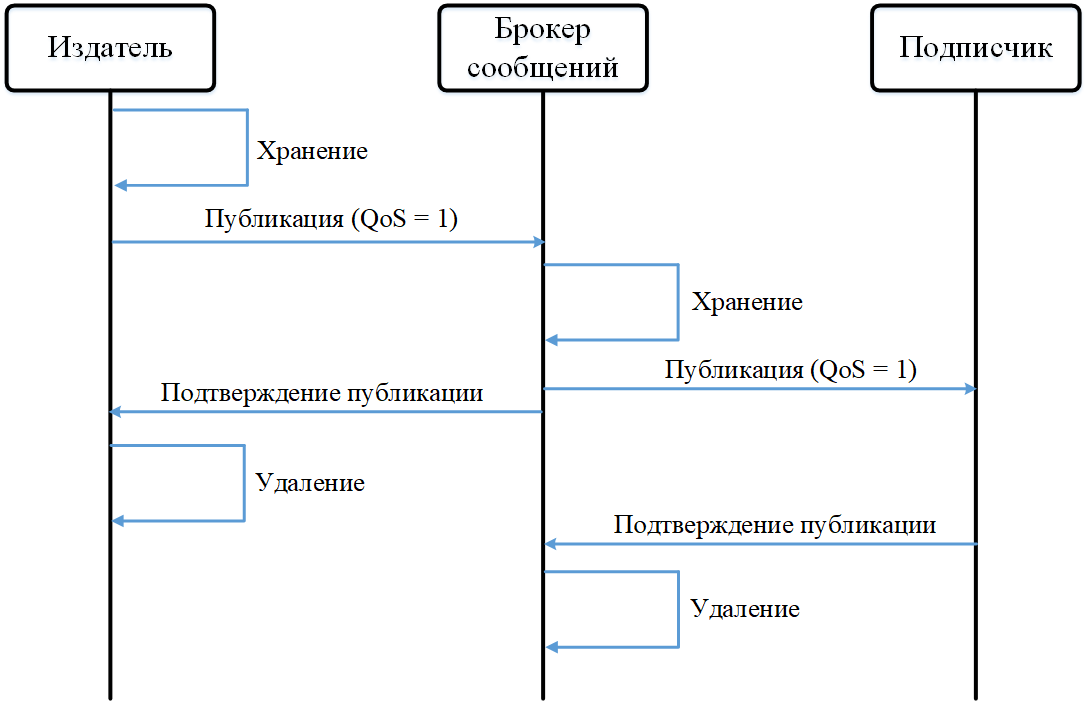


Рис. 13. Доставка сообщение при уровне *QoS* = 1 [1]

* *QoS* = 2 – используя подтверждение, основанное на четырёхступенчатом рукопожатии, сообщение будет доставлено ровно один раз.

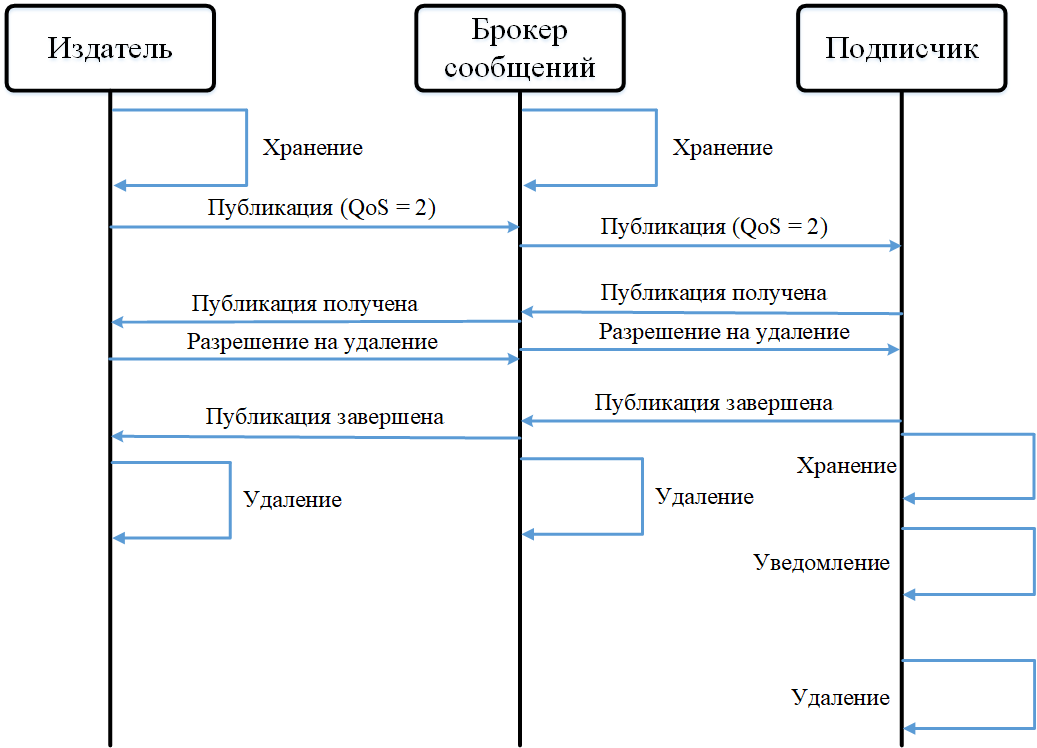


Рис. 14. Доставка сообщение при уровне *QoS* = 2 с использованием четырёхступенчатого рукопожатия [1]

Использование предполагает работу с любыми данными, которые можно представить в байтовой форме размером не более 260 Мбайт [21], однако на практике сообщения таких размеров не используются. Основная сфера применения – это доставка небольших сообщений, к примеру, содержащих телеметрические данные или любые другие, подразумевающие использование различных датчиков и устройств, способных передавать необходимые сведения между устройствами или в центральный узел.

*Безопасное соединение*

В связи с тем, что между устройствами по сети возможен обмен конфиденциальными данными, безопасная передача играет большую роль. В протоколе *MQTT* для этого реализованы следующие методы защиты:

* Аутентификация клиентов при помощи полей с именем клиента и паролем,
* Контроль доступа клиентов путём использования *Client ID* или *MAC*-адреса,
* Аутентификация клиентов через *TLS / SSL*-подключение с использованием сертификатов.

В связи с тем, что первые два метода не обеспечивают полную безопасность через шифрование данных в процессе передачи, в исследовательской работе решено использовать коммуникацию через порт 8883 с использованием протоколов *TLS / SSL* и аутентификацией на основе самоподписанных сертификатов.

*Шифрование в* *MQTT*

Согласно функциям протоколов безопасности при использовании протокола *TLS* версий 1.2 и 1.3 гарантируется безопасность передачи данных в сетях связи.

Используемое в исследовательской работе программное обеспечение для брокера сообщений *EMQ* *X*, имеет встроенную поддержку *TLS* / *SSL*. Таким образом, безопасная передача может быть обеспечена путём односторонней либо двусторонней аутентификации, применением сертификата *X*.509, как в настоящей работе, и других сертификатов безопасности.

Преимущества, обеспечиваемые применением данных протоколов:

* аутентификация с использованием сертификатов для обмена открытыми ключами, позволяющая при установлении соединения проверить личности каждой из сторон;
* конфиденциальность, обеспечиваемая путём шифрования сеанса посредством ключа сеанса;
* целостность данных, обеспечиваемая за счёт предотвращения изменения данных в процессе передачи.

Связь клиента и сервера состоит из двух этапов. Вначале происходит идентификация и установление безопасного канала – *TLS* *Handshake*, после чего происходит обмен набором паролей и сеансовым ключом. Вторым этапом происходит непосредственный обмен информацией, но уже в зашифрованном виде посредством протокола записи.

***Аппаратные средства***

*Описание сценария*

Прежде чем приступить к непосредственному описанию аппаратных средств, использующихся в настоящем исследовании, необходимо подробно представить сценарий, согласно которому устройства взаимодействуют в сети.

Для реализации сбора данных с устройств организована сеть, на логическом уровне представляющая неполносвязную ячеистую топологию, состоящая из 11-ти роутеров и 4-х персональных компьютеров (ПК). ПК 11 представляет собой центр сбора и обработки данных, получаемых от трёх других ПК, играющих роль датчика температуры (ПК 8), датчика влажности (ПК 6) и электронного турникета (ПК 2). Согласно шаблону проектирования «издатель-подписчик», по которому функционирует протокол *MQTT*, последние являются издателями, а ПК 11 – подписчиком. Роутеры 1, 5 и 9 являются брокерами, объединёнными в кластер. Логическая схема сети представлена на рисунке 15.

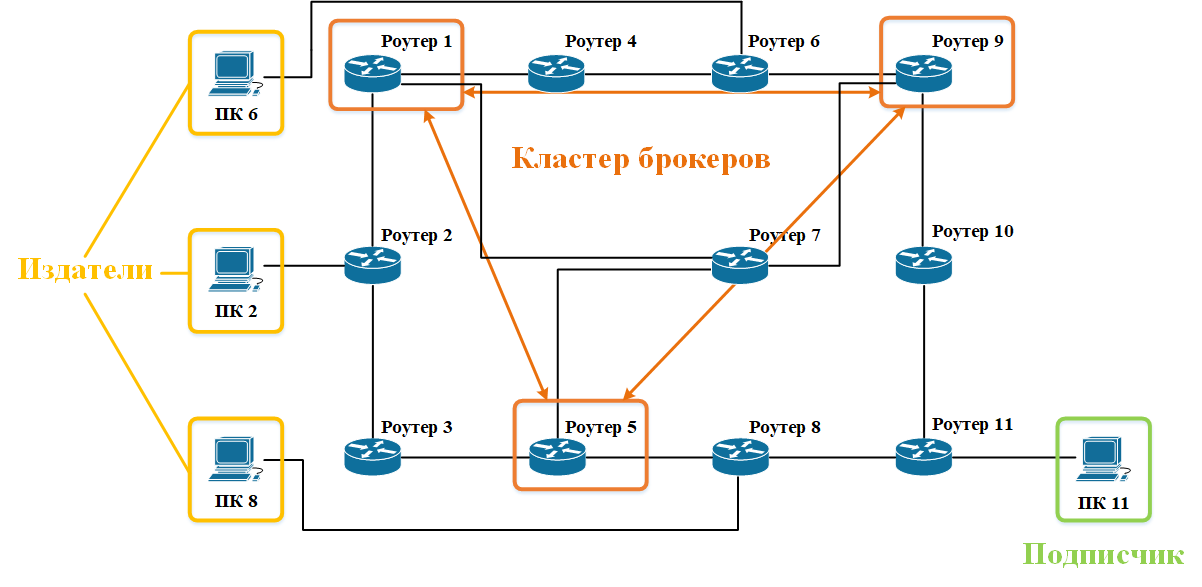


Рис. 15. Логическая схема сети с выделенными согласно сценарию клиентами и кластером брокеров

Для получения отправленного издателем на определённую тему сообщения подписчику необходимо быть подписанным на эту тему. Так как в схеме имеется три издателя и один подписчик, последнему нужно быть подписанным на каждую из тем. Темы и сообщения издателей представлены ниже:

* *mpei/main-building/room1/temp* – для датчика температуры:
  + 28.51,
  + 28.53,
  + 28.54,
  + 28.50,
  + 27.49,
* *mpei/main-building/room2/humi* – для датчика влажности:
  + 36.1,
  + 36.3,
  + 36.2,
  + 36.1,
  + 36.0,

Издатели публикуют запрограммированные на отправку сообщения с интервалом 15 секунд. Данные отражают одно из устройств, каждое из которых публикует, в общей сложности, 500 сообщений.

***Методы проектирования сети***

Для реализации разработанного сценария, необходимо вначале спроектировать проект в виртуальной среде.

Существует два основных метода проектирования сетей с целью их исследования: в модельной вычислительной среде и реальной. Модельный метод, в свою очередь, подразделяется на симуляционный и эмуляционный. Реальный метод включает в себя натурное моделирование и эталонное.

Симуляция представляет собой подход, при котором смоделированный процесс выполняется в рамках модели вычислительной среды.

Эмуляция – это метод, в котором реальный процесс выполняется в модели вычислительной среды. Его идея заключается в том, что создаются синтетические условия для работы реального процесса. Примером средств эмуляции является, к примеру, проект *Mininet* [25].

Эталонное тестирование – это метод, при котором смоделированный процесс выполняется в реальной среде, при этом оценивается эффективность её функционирования. Процесс должен охватывать следующие характеристики среды: её мощность, скорость обработки и ввода/вывода данных, пропускную способность сети и т.д. В отличии от натурального моделирования, в эталонном тестировании не важен результат выполнения процесса, внимание обращается на функционирование системы в ходе работы.

Натурное моделирование представляет собой метод, при котором реальный процесс выполняется в реальной вычислительной среде. В качестве процесса могут выступать сервис, приложение и т.д., а в качестве среды – сервер, операционная система и т.д. При этом собирается информация о работе процесса и состоянии системы, в результате формируется статистика.

В рамках исследовательской работы сеть вначале эмулируется в ПО *Mininet* [1], [25] с целью отладки работоспособности сценария, после чего проект переносится на реальное оборудование и проводится эталонное тестирование.

ПО *EMQ X*

*EMQ X Broker* – это программное обеспечение с открытым исходным кодом и работающее под лицензией *Apache License Version 2.0*, являющееся масштабируемым брокером сообщений *MQTT*, которое написано на языке *Erlang* [27]. ПО разработано для массового доступа клиентов и реализует быструю маршрутизацию сообщений с малой задержкой между устройствами. Может применяться как в масштабах экспериментальной установки, так и для масштабного коммерческого развертывания.

*EMQ X Broker* поддерживает протокол *MQTT* версий 3.1, 3.1.1 и 5.0, две новейшие из которых используются в настоящей работе.

В исследовании применяется ПО *EMQ X* версии 4.2.5, реализующее на компьютерах брокеров, объединённых в кластер.

*Команды Mosquitto*

*Mosquitto* – это проект, реализующий ПО с открытым исходным кодом для работы брокера сообщений, работающее под лицензией *EPL/EDL* и обеспечивающее взаимодействие с протоколом *MQTT* [29]*.* Также *Mosquitto* предоставляет библиотеку Си для реализации *MQTT*-клиентов, в частности, клиентов командной строки *mosquitto\_pub* и *mosquitto\_sub*. Названные утилиты используются для отправки и принятия данных посредством команд с набором необходимых параметров. Проект *Mosquitto* поддерживает все версии протокола *MQTT* – версии 3.1, 3.1.1 и 5.0.

В настоящей работе для публикации сообщений используется команда *mosquitto\_pub* следующего вида [29]:

*mosquitto\_pub { [ -h hostname ] [ -p port-number] -t message-topic [ -q message-QoS ] [ -V protocol-version ] -m message [ --cafile file } [ --cert file ] [ --key file ] }*

Соответственно, для сообщения MQTT версии 3.1.1, параметром QoS = 1 без установления безопасного TLS / SSL-соединения команда следующая:

*mosquitto\_pub -h 44.44.65.5 -t "mpei/main-building/server-room2.humi" -m "36.3" -q 1*

Для публикации MQTT-сообщения версии 5.0, с уровнем QoS = 2 и установлением безопасного соединения через порт 8883 с использованием сертификатов и приватного ключа команда выглядит следующим образом:

*mosquitto\_pub -h 44.44.65.5 -p 8883 --cafile ca.pem --cert client.pem --key client.key -t "mpei/main-building/server-room1/temp" -m "28.51" -V mqttv5 -q 2*

Для получения сообщений необходимо быть подписанным на определённую тему, на которую сообщения публикуются. Так для получения данных о влажности в комнате server-room2 в главном здании по незащищённому каналу через порт 1883 с уровнем QoS = 1 команда подписки следующая:

*mosquitto\_sub -h 44.44.22.11 -t "mpei/a-building/server-room2" -q 1*

Для подписки на тему температуры из серверной комнаты №1, с параметром QoS = 2 по версии протокола MQTT 5.0 команда выглядит следующим образом:

*mosquitto\_sub -h 44.44.65.5 -p 8883 --cafile ca.pem --cert client.pem --key client.key -t "mpei/a-building/server-room1" -V mqttv5 -q 2*

*Эталонное тестирование*

После создания сценария, в среде эмулирования *Mininet* для отладки проекта и успешного выполнения тестирования в виртуальной среде следующим этапом является перенос конфигураций на реальное оборудование с целью дальнейшего выполнения измерений.

Построение стенда производится на базе компьютерного класса *CISCO* в НИУ «МЭИ». Применяемое в ходе исследования оборудование *CISCO* является мировым лидером в сегменте безопасного сетевого оборудования [11].

*Использование самоподписанных сертификатов*

Создание самоподписанных сертификатов подразумевает самостоятельное генерирование приватного ключа пользователем для дальнейшей подписи им корневого сертификата. Далее рассмотрен способ, предполагающий использование ПО *OpenSSl*. После установки названного ПО можно приступить к созданию сертификатов.

На первом этапе будет создан корневой сертификат центра сертификации. Для его создания вначале происходит генерация приватного ключа *ca.key* длины 2048 бит алгоритмом *RSA*:

*openssl genrsa -out ca.key 2048*

Далее этот ключ используется для создания корневого сертификата *ca.pem*. Следующей командой это реализуется алгоритмом *SHA*256 на срок действия сертификата 120 дней:

*openssl req -x509 -new -nodes -key ca.key -sha256 -days 120 -out ca.pem*

На втором этапе создаётся сертификат сервера. Для этого для сервера также генерируется свой приватный ключ для обеспечения контроля над своими сертификатами. Процесс создания приватного ключа *server*.*key* аналогичен приведенному выше для создания приватного ключа центра сертификации:

*openssl genrsa -out server.key 2048*

Далее сгенерированный приватный ключ используется для создания запроса *server.csr* на создание сертификата для сервера:

*openssl req -new -key ./server.key -out server.csr*

В процессе создания необходимо будет указать контактные данные владельца сертификата. Этот запрос будет направлен в центр сертификации для создания сертификата.

При помощи следующей команды, после создания запроса на создание сертификата, посредством центра сертификации будет создан сертификат для сервера *server.pem* сроком на 120 дней:

*openssl x509 -req -in ./emqx.csr -CA ca.pem -CAkey ca.key -CAcreateserial -out server.pem -days 120 -sha256 -extensions v3\_req*

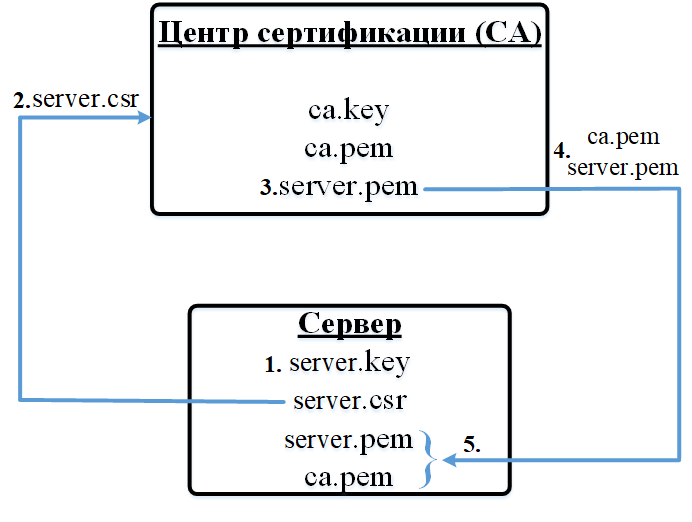


Рис. 16. Процесс создания сертификата сервера

Для проверки корректности созданного сертификата *server.pem* с помощью файла сертификата корневого центра сертификации *ca.pem* используется следующая команда:

*openssl verify -CAfile ca.pem server.pem*

Вывод сообщения «*server.pem: OK»* говорит о том, что создание произошло корректно.

На третьем этапе происходит создание сертификата клиентов. Вначале также генерируется приватный ключ, но уже для обеспечения контроля над клиентскими сертификатами *client*.*key* длиной 2048 бит аналогично приведенным выше для центра сертификации и сервера:

*openssl genrsa -out client.key 2048*

Далее сгенерированный приватный ключ используется для создания запроса *client.csr* на создание сертификата для клиентов:

*openssl req -new -key ./ client.key -out client.csr*

В процессе создания необходимо будет также указать контактные данные владельца сертификата. Этот запрос будет направлен в центр сертификации для создания сертификата.

Далее посредством корневого центра будет создан сертификат для клиентов *client.pem* сроком на 120 дней:

*openssl x509 -req -in ./emqx.csr -CA ca.pem -CAkey ca.key -CAcreateserial -out client.pem -days 120 -sha256 -extensions v3\_req*

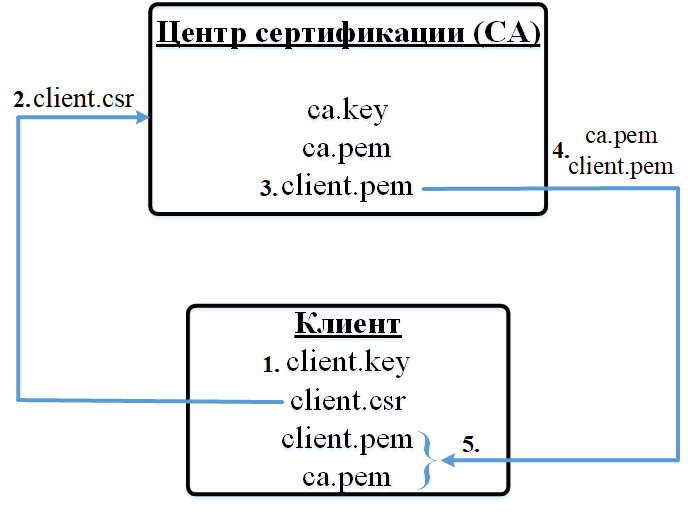


Рис. 17. Процесс создания сертификата клиента

Для проверки корректности созданного сертификата с помощью файла сертификата корневого центра сертификации *ca.pem* используется следующая команда:

*openssl verify -CAfile ca.pem client.pem*

Вывод сообщения «*client.pem: OK»* говорит о том, что создание сертификата для клиентов прошло успешно.

В итоге, на сторонах клиента и сервера набор сертификатов и приватных ключей, которые в дальнейшем будут использоваться протоколом *TLS* для установления безопасного соединения.

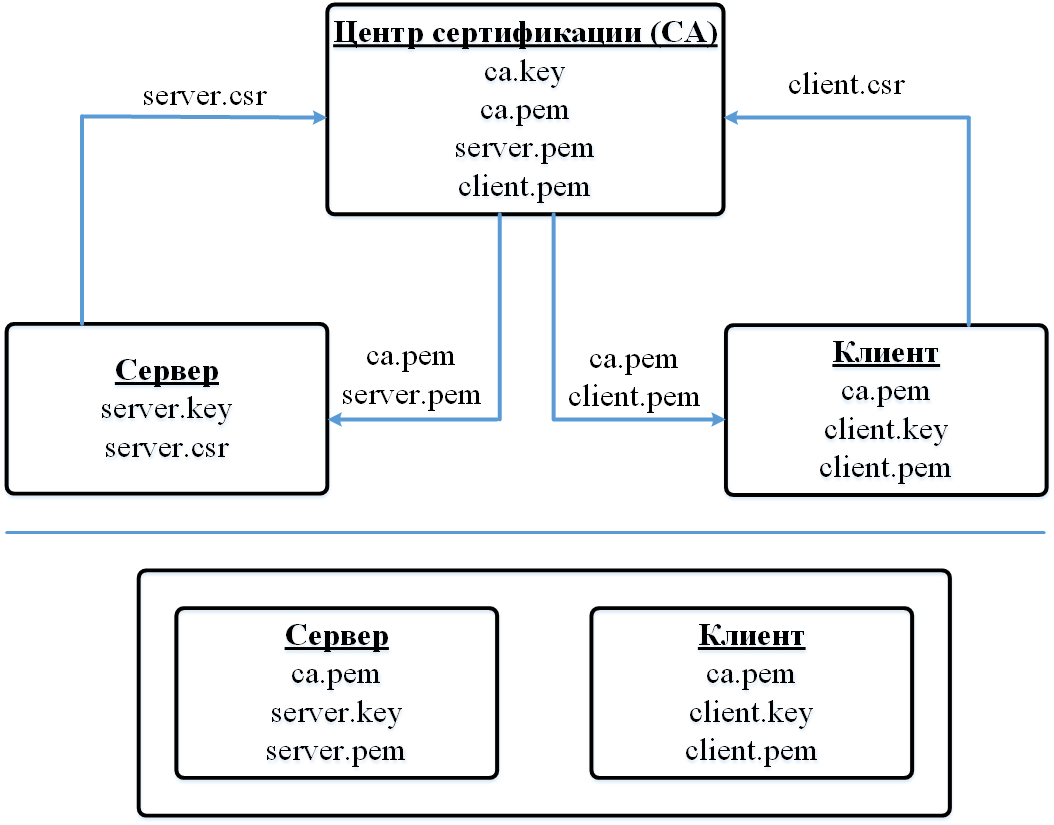


Рис.18. Итоговый набор ключей и сертификатов на сторонах сервера и клиента для дальнейшего взаимодействия

После перехвата и анализа данных при помощи программы *WireShark*, можно выделить обмен сертификатами и ключами из общего трафика. Однако сами данные извлечь не удастся, потому что протоколы *TLS / SSL* шифруют передаваемые сообщения.

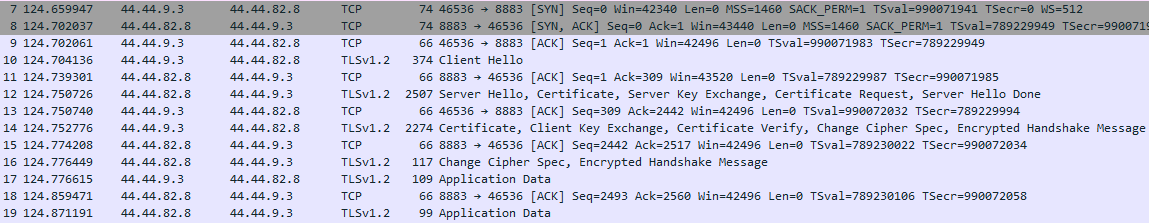


Рис. 19. Фрагмент захваченного сниффером WireShark трафика, реализующего TLS / SSL-рукопожатие, в процессе которого происходит обмен сертификатами и ключами с целью установления безопасного соединения

***Исследование характеристик протокола MQTT***

*Описание параметров корпоративной сети*

Для проведения исследования характеристик протокола *MQTT* автором был разработан и подготовлен стенд, на базе компьютерного класса *CISCO* в НИУ «МЭИ». Стенд включает в себя набор персональных компьютеров, а также сетевое оборудование. Именно, разработанная сеть включает в себя следующее оборудование:

* 14 вычислительных машин с установленным ПО *Debian 10*;
* 1 коммутатор 3-го уровня *Cisco Catalyst 3560 Series PoE-24*;

Анализ предметной области показал:

Для реализации схемы сети, представленной в главе 2.1, необходимо было объединить компьютеры, используя коммутатор третьего уровня. Далее задать сетевые настройки компьютерам, после чего посредством создания на коммутаторе виртуальных локальных сетей – VLAN’ов – и распределения компьютеров по этим сетям, произвести настройку маршрутизации.

*Настройка реального оборудования*

*Cisco IOS* (англ. *Internetwork Operating System* — Межсетевая Операционная Система) — программное обеспечение, которое используется в большинстве коммутаторов и маршрутизаторов *Cisco*. *Cisco IOS* является многозадачной операционной системой, которая включает такие функции как маршрутизация, коммутация, межсетевое взаимодействие и другие [4].

В *Cisco IOS* имеется интерфейс командной строки (англ. *Command Line Interface, CLI*). Интерфейс *IOS* предоставляет фиксированный набор команд, согласно выбранному режиму и уровню привилегий пользователя.

После получения доступа к консоли управления оборудования начинается этап настройки коммутатора. Процесс конфигурации *VLAN* в режиме *Config-vlan*:

1) Выполнить команду "*enable*", чтобы перейти из простого режима в привилегированный.

2) Выполнить команду "*configure terminal*", чтобы перейти в режим конфигурации.

3) Создать *VLAN* командой "*vlan <number>*".

4) Выполнить команду "interface [type] mod/port>", чтобы перейти в режим конфигурации интерфейса.

5) Выполнить команду "switchport mode trunk" чтобы сконфигурировать режим порта.

6) Выполнить команду "switchport trunk encapsulation dot1q", чтобы включить магистральное соединение на интерфейсе.

7) Выполнить команду "switchport trunk allowed <vlan1-id>, <vlan2-id>" чтобы указать перечень VLAN’ов для транкового порта.

8) Чтобы проверить конфигурацию VLAN, выполнить команду "show vlan".

***Проведение экспериментов***

В этом разделе описаны этапы работы, а затем представлены результаты измерения таких характеристик как количество переданных пакетов и байт информации, сквозная задержка.

Раздел включает в себя исследование характеристик протокола *MQTT* версий 3.1.1 и 5.0 с использованием различных параметров протокола.

Для протокола MQTT версии 3.1.1 эксперименты проводились в следующем порядке:

* 1. Передача сообщений *MQTT* с параметром *QoS* = 1:

А) без установления защищённого *TLS / SSL*-соединения,

Б) по защищённому соединению с использованием приватных ключей и сертификатов;

* 1. Передача сообщений *MQTT* с параметром *QoS* = 2:

А) без установления защищённого *TLS / SSL*-соединения,

Б) по защищённому соединению с использованием приватных ключей и сертификатов;

Аналогично проводились эксперименты для протокола *MQTT* версии 5.0:

* 1. Передача сообщений *MQTT* с параметром *QoS* = 1:

А) без установления защищённого *TLS / SSL*-соединения,

Б) по защищённому соединению с использованием приватных ключей и сертификатов;

* 1. Передача сообщений *MQTT* с параметром *QoS* = 2:

А) без установления защищённого *TLS / SSL*-соединения,

Б) по защищённому соединению с использованием приватных ключей и сертификатов;

***Переданный трафик***

Количество переданной информации в Кбайтах

Далее приведена таблица 1 с результатами количества переданного трафика, измеряемого в единицах измерения Кбайт. Меньшее из сравниваемых значение выделено цветом.

Таблица 1. Входящий и исходящий трафик для клиентов 2, 6, 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |  | Входящий трафик, Кбайт | | Исходящий трафик, Кбайт | |
| Версия  Клиент | 3.1.1 | 5.0 | 3.1.1 | 5.0 |
| QoS = 1 | Без TLS / SSL | Клиент 2 | 3125 | 3114 | 4282 | 4294 |
| Клиент 6 | 3126 | 3101 | 4268 | 4282 |
| Клиент 8 | 4097 | 4095 | 4756 | 4761 |
| TLS / SSL | Клиент 2 | 5563 | 5599 | 5501 | 5532 |
| Клиент 6 | 5574 | 5591 | 5509 | 5562 |
| Клиент 8 | 6608 | 6649 | 5988 | 6013 |
| QoS = 2 | Без TLS / SSL | Клиент 2 | 3662 | 3717 | 4886 | 4900 |
| Клиент 6 | 3675 | 3682 | 4868 | 4863 |
| Клиент 8 | 5635 | 5664 | 5881 | 5904 |
| TLS / SSL | Клиент 2 | 6123 | 6113 | 6122 | 6116 |
| Клиент 6 | 6126 | 6148 | 6096 | 6133 |
| Клиент 8 | 8119 | 8158 | 7062 | 7166 |

В ходе проведения экспериментов по подсчёту переданного количества Кбайт информации было установлено, что количество трафика, необходимого для передачи данных протоколом *MQTT* версии 3.1.1 хоть и меньше, чем для версии 5.0, но не значительно, так как в среднем не превышает 0,5%. Объём исходящего трафика также незначительно увеличивается, в среднем на 0,5%.

Параметр качества обслуживания *MQTT* показывает, что при использовании *QoS* = 2, количество трафика увеличивается по сравнению с *QoS* = 1, чему способствуют дополнительные передачи между брокером и клиентом, за счёт которых обеспечивается доставка ровно одного сообщения без повтора. При его использовании количество переданных Кбайт входящего трафика увеличилось на 24,1% по незащищённому каналу и на 14,3% по защищённому для версии 3.1.1 против 25,5% и 13,9% для версии 5.0. Количество исходящего трафика увеличилось в среднем на 17,3% по незащищённому каналу и на 13,3% по защищённому для обеих версий.

С целью обеспечения безопасной передачи данных в зашифрованном виде применялись протоколы *TLS / SSL*. Соответственно, за счёт установления *TLS /* *SSL*-соединения увеличилось количество передач, для аутентификации потребовался обмен дополнительными средствами в виде приватных ключей и сертификатов, а зашифрованные сообщения имеют больший размер, что повлекло значительное увеличение количества трафика – для версии 3.1.1 увеличилось на 72,5% при *QoS* = 1 и на 59,3% при *QoS* = 2 против 74,2% и 58,5%, соответственно, для версии 5.0. Объём исходящего трафика для обеих версий при QoS = 1 и QoS = 2 увеличился в среднем на 28,1% и 23,8%.

В соответствии с ожиданиями, наибольший прирост трафика связан с добавлением шифрования для обеспечения безопасной передачи данных. Обеспечение гарантированной доставки ровно одного сообщений, что означает использование параметра QoS = 2, также, согласно ожиданиям, повлияло на увеличение передаваемой информации. Версия же на данную характеристику не оказала сильного влияния.

Количество переданных пакетов

В таблице 2 приведены результаты подсчёта количества переданных пакетов. Стрелками показано большее или меньшее количество пакетов в зависимости от количества переданных байт информации, относительно таблицы 4.1.

Таблица 2. Количество входящий и исходящих пакетов для клиентов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |  | Входящий трафик | | Исходящий трафик | |
| Версия  Клиент | 3.1.1 | 5.0 | 3.1.1 | 5.0 |
| QoS = 1 | Без TLS / SSL | Клиент 2 | 167 | 201 ↑ | 251 ↓ | 254 |
| Клиент 6 | 168 | 200 ↑ | 248 ↓ | 252 |
| Клиент 8 | 244 | 279 ↑ | 285 ↓ | 288 |
| TLS / SSL | Клиент 2 | 1546 ↓ | 1587 | 1692 ↓ | 1705 |
| Клиент 6 | 1543 ↓ | 1584 | 1693 ↓ | 1708 |
| Клиент 8 | 1645 ↓ | 1688 | 1711 ↓ | 1726 |
| QoS = 2 | Без TLS / SSL | Клиент 2 | 197 ↓ | 234 | 284 ↓ | 287 |
| Клиент 6 | 197 ↓ | 232 | 281 | 283 ↑ |
| Клиент 8 | 327 ↓ | 365 | 346 ↓ | 351 |
| TLS / SSL | Клиент 2 | 1590 | 1625 ↑ | 1732 | 1729 ↓ |
| Клиент 6 | 1587 ↓ | 1628 | 1720 ↓ | 1747 |
| Клиент 8 | 1749 ↓ | 1794 | 1785 ↓ | 1822 |

Согласно данным в таблице 2, относительно количества пакетов соотношения отличаются по сравнению с таблицей 4.1. Так, при использовании *MQTT* версии 5.0 без установления *TLS / SSL*-соединения количество пакетов увеличилось на 15,2% с параметром *QoS* = 1 и на 13,8% с параметром *QoS* = 2 по сравнению с версией 3.1.1 для входящего трафика. С установлением безопасногосоединения количество пакетов увеличилось в среднем всего на 2,5% для обоих случаев. Для исходящего трафика количество пакетов без установления *TLS / SSL*-соединения увеличилось на 1,1%, с установлением – на 1,0%.

При рассмотрении количества переданных пакетов относительно параметра *QoS*, результатом является увеличение их количества на 23,1% для нешифрованного трафика и на 4,0% для зашифрованного для версии 3.1.1 против увеличения на 21,1% и 3,8% для старшей версии во время отправки данных. Во время же получения данных увеличение количества пакетов составило 15,9% по незащищённому каналу и 3,0% по защищённому для версий 3.1.1 и 5.0.

На обеспечение безопасного шифруемого обмена данными с использованием приватных ключей и сертификатов количество пакетов входящего трафика увеличилось на 739,5% с параметром *QoS* = 1 и на 615,9% для младшей версии *MQTT* против увеличения на 628,9% и 529,2% для старшей версии. Увеличение количества пакетов в исходящем трафике для обеих версий протокола при QoS = 1 и QoS = 2 в среднем составило 550,9% и 479,5%, соответственно.

Наибольший прирост количества пакетов, что который отразился на увеличении трафика связан с добавлением шифрования для обеспечения безопасной передачи данных. Использование параметра *QoS* = 2, обеспечивающего гарантированную доставку ровно одного сообщения, также повлияло на увеличение количества передаваемых пакетов, однако при использовании *TLS / SSL*-соединения прирост оказался небольшим и составил в среднем всего 3,9%. Использование же новой версии *MQTT* количество без *TLS / SSL* увеличило число сообщений в среднем на 13,5%.

Связь количества переданных байт информации и пакетов

В итоге получены результатов следующие данные:

Таблица 3. Сравнение входящего трафика по трём параметрам

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *QoS* = 1 | | | | *QoS* = 2 | | | |
| Версия | *no TLS / SSL* | | *TLS / SSL* | | *no TLS / SSL* | | *TLS / SSL* | |
| Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты |
| 3.1.1  vs  5.0 | 0,4% | 15,2% | 0,5% | 2,6% | 0,7% | 13,8% | 0,3% | 2,4% |
|  | Версия 3.1.1 | | | | Версия 5.0 | | | |
| QoS | *no TLS / SSL* | | *TLS / SSL* | | *no TLS / SSL* | | *TLS / SSL* | |
| Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты |
| 1  vs  2 | 24,1% | 23,1% | 14,3% | 4,0% | 25,5% | 21,1% | 13,9% | 3,8% |
|  | Версия 3.1.1 | | | | Версия 5.0 | | | |
| Соеди-нение | *QoS* = 1 | | *QoS* = 2 | | *QoS* = 1 | | *QoS* = 2 | |
| Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты |
| *no TLS / SSL*  *vs*  *TLS / SSL* | 72,5% | 739,5% | 59,3% | 615,9% | 74,2% | 628,9% | 58,5% | 529,2% |

Таблица 4. Сравнение исходящего трафика по трём параметрам

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | *QoS* = 1 | | | | *QoS* = 2 | | | |
| Версия | *no TLS / SSL* | | *TLS / SSL* | | *no TLS / SSL* | | *TLS / SSL* | |
| Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты |
| 3.1.1  vs  5.0 | 0,2% | 1,1% | 0,6% | 0,8% | 0,3% | 1,1% | 0,7% | 1,2% |
|  | Версия 3.1.1 | | | | Версия 5.0 | | | |
| QoS | *no TLS / SSL* | | *TLS / SSL* | | *no TLS / SSL* | | *TLS / SSL* | |
| Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты |
| 1  vs  2 | 17,3% | 16,0% | 13,3% | 2,8% | 17,2% | 15,7% | 13,3% | 3,1% |
|  | Версия 3.1.1 | | | | Версия 5.0 | | | |
| Соеди-нение | *QoS* = 1 | | *QoS* = 2 | | *QoS* = 1 | | *QoS* = 2 | |
| Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты | Кбайты | пакеты |
| *no TLS / SSL*  *vs*  *TLS / SSL* | 27,8% | 552,4% | 23,5% | 479,3% | 28,3% | 549,4% | 24,1% | 479,6% |

На основании полученных результатов из таблиц 3 и 4 и исследования трафика, полученного применением программы WireShark, можно сделать следующие выводы:

1. Количество как входящего, так и исходящего трафиков наибольшим образом зависит от организации безопасного *TLS / SSL*-соединения.
2. При использовании параметра *QoS* = 2 процентное увеличение количества передаваемой и принимаемой информации меньше, чем при значении *QoS* = 1.
3. Версия влияет лишь на количество передаваемых пакетов и практически не оказывает влияния на изменение количества трафика.

Графики измерений для двух крайних случаев – с минимальными параметрами и максимальными – изображены на рисунках 20 и 21.

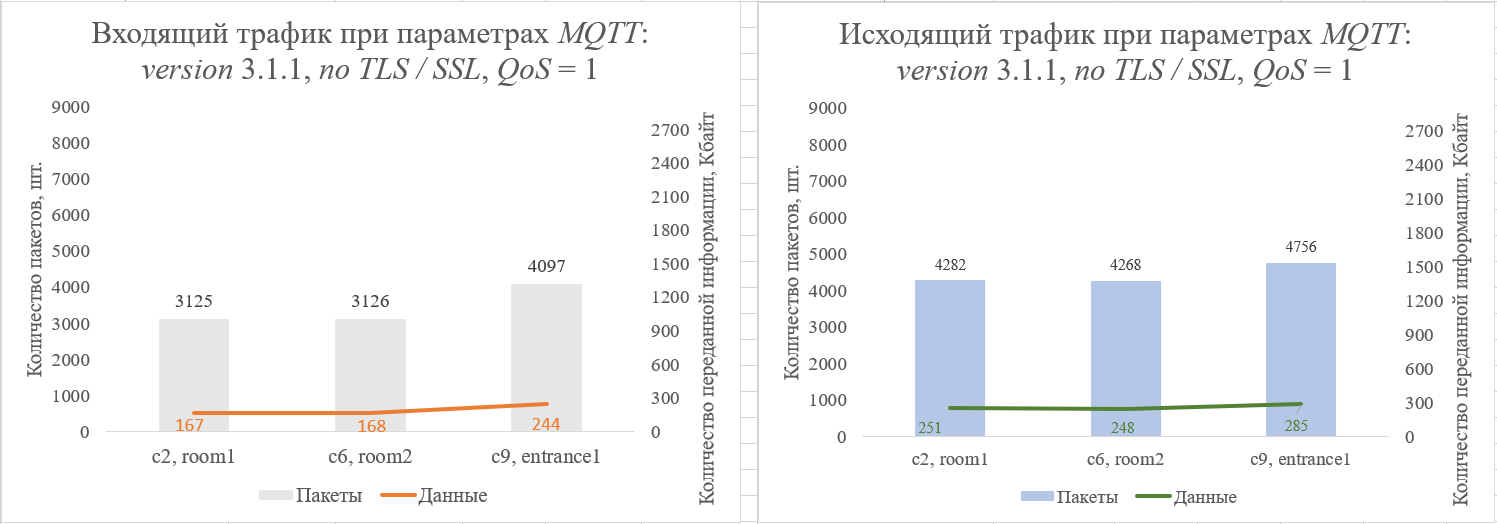
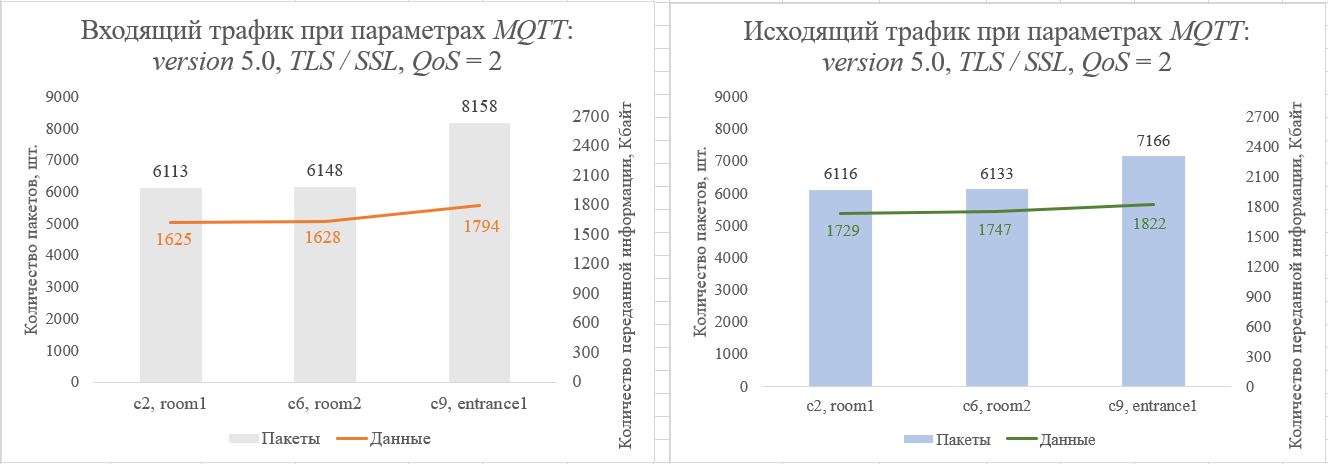


Рис. 20. Графики принятого и отправленного трафика для клиентов 2, 6 и 8 с параметрами без установления защищённого *TLS / SSL*-соединения протокола *MQTT* версии 3.1.1 с параметром *QoS* = 1 (минимальные параметры)

Рис.21. Графики принятого и отправленного трафика для клиентов 2, 6 и 8 с параметрами c установлением защищённого *TLS / SSL*-соединения протокола *MQTT* версии 5.0 с параметром *QoS* = 2 (максимальные параметры)

*Задержки*

Далее приведены результаты измерений и анализ сквозной задержки передачи пакетов. Графики для случаев использования минимальных параметров и максимальных приведены на рисунках 22 и 23.



Рис. 22. График сквозной задержки пакетов протокола *MQTT*, измеренной от клиентов 2, 6 и 8, с минимальными параметрами



Рис. 23 График сквозной задержки пакетов протокола *MQTT*, измеренной от клиентов 2, 6 и 8, с максимальными параметрами

Рисунки 22 и 23 демонстрируют, что в целом, с увеличением параметра задержка становится более продолжительная и появляется большее количество выбросов. Для наглядности и более детального рассмотрения задержек, были построены диаграммы размаха для каждого из клиентов для всех измерений.

Диаграмма размаха передает шестичастную структуру данных и выделяет следующие значения:

* Минимальное значение,
* Первый квартиль – 25% значений,
* Медиану,
* Среднее арифметическое значение,
* Третий квартиль – 75% выборки,
* Максимальное значение.

Первый квартиль на диаграмме отражается нижней стороной прямоугольника и отражает 25% выборки, а третий квартиль – верхней стороной и показывает значений в 75%. Расстояние между первым и третьим квартилями является межквартильным размахом.

Линией внутри прямоугольника обозначается медиана, а крестиком – среднее арифметическое по выборке.

В то же время диаграмма показывает выбросы. Выбросами являются точки, которые отстают от первого или от третьего квартиля более чем на полтора межквартильных расстояния.

На рисунках 24, 25, и 26 представлены диаграммы размаха для клиентов 2, 6 и 8, соответственно.

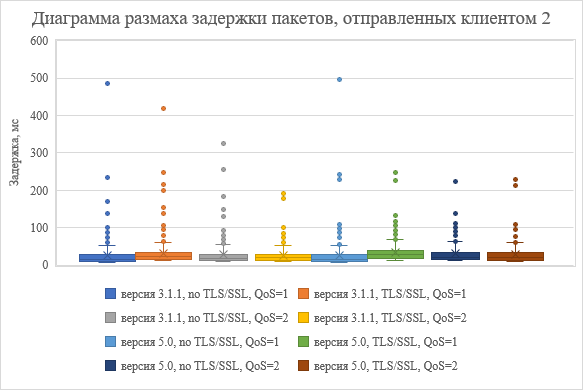


Рис. 24. Диаграмма размаха, построенная для клиента 2 относительно всех измерений

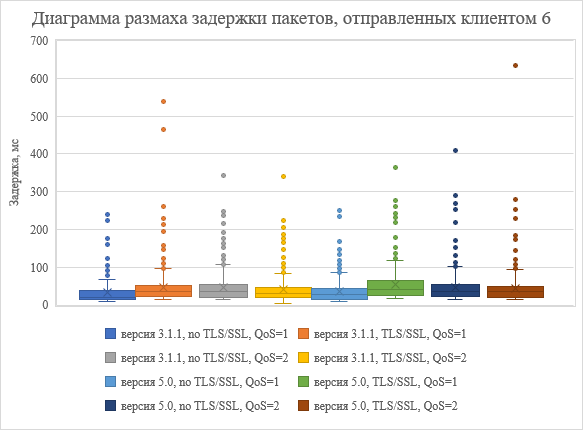


Рис. 25. Диаграмма размаха, построенная для клиента 2 относительно всех измерений

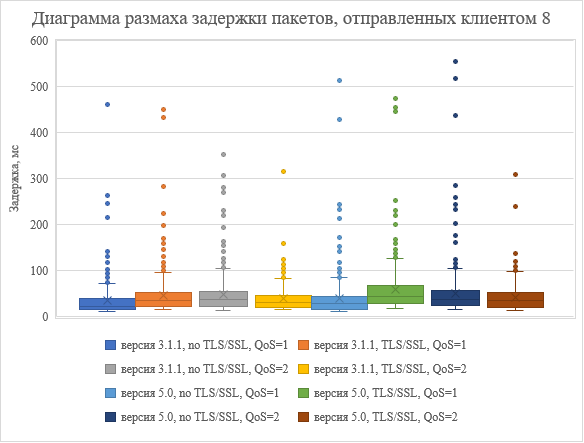


Рис. 26 Диаграмма размаха, построенная для клиента 8 относительно всех измерений

Графики показывают, что для каждого из клиентов при использовании безопасного *TLS / SSL*-соединения значение медианы в 58,3% случаев увеличивается, в 25% остаётся на том же уровне и 16,7% уменьшается.

*Результаты исследования*

В результате экспериментов установлено:

1. Сеть работает без сбоев и исправно доставляет пакеты данных с задержкой менее 70 мс в 75% случаев и с экстремальными выбросами в диапазоне от 120 мс до 620 мс .
2. При использовании безопасного TLS / SSL-соединения количество передаваемых пакетов увеличивается более, чем в полтора раза. Несмотря на это количество передаваемых байт информации увеличилось менее, чем на 70%.
3. В среде проводной сети существует небольшая разница между сквозными задержками при использовании младшей версии 3.1.1 протокола MQTT, значении уровня QoS = 1 без установления TLS / SSL-соединения, что является наиболее «облегчённым» вариантом относительно количества передаваемой информации, и при использовании старшей версии 5.0 с уровнем QoS = 2 и использованием безопасного соединения, являющимся более «нагруженным». В таком случае, за счёт лёгкости протокола, в корпоративной сети предпочтительней обеспечить шифрование трафика для исключения его перехвата и подмены, а также уровень QoS = 2 для исключения дублирования данных при длительных снятиях показаний.
4. С учётом отсутствия обратной совместимости версий 3.1.1 и 5.0 протокола MQTT, предпочтительней использовать старшую версию в связи с расширенным функционалом и по причине его активного развития.

**ЗАКЛЮЧЕНИЕ**

В результате выполненных работ создан стенд, на базе которого исследованы характеристики протокола передачи данных *MQTT*. В том числе была реализована передача зашифрованного трафика с установлением безопасного соединения при помощи протоколов *TLS / SSL*.

В результате произведены измерения количества переданных пакетов и байт информации, которые с учетом значений сквозной задержки позволили обосновать выбор параметров протокола *MQTT* для использования его в качестве протокола передачи данных в корпоративной сети.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Бородулин С.В. *“Service Management in einem Notfallkommunikationsnetz”, TU Ilmenau, Juni*, 2020
2. Таненбаум Э., Уэзеролл Д. Компьютерные сети. – 6-е изд. - СПб.: Питер, 2014.
3. Электронный учебник по курсу: Повышение квалификации руководящих работников, специалистов и преподавателей ВУЗа в области ИКТ *http*://*e*-*ikt*.*uginfo*.*sfedu*.*ru*/
4. Официальный сайт компании *CISCO*. Коммутаторы *CISCO* *Catalyst* серии 3650. - 2021. - URL: *https://www.cisco.com/c/ru\_ru/support/switches/catalyst-3650-series-switches/series.html* (дата обращения 25.06.2021)
5. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. – 5-е изд. - СПб.: Питер, 2016.
6. *Braden R. Internet Engineering Task Force / R. Braden. - USC/Information Sciences Institute, 1989. - 116 p. - URL: https://datatracker.ietf.org/doc/html/rfc1122* (дата обращения: 25.06.2021)
7. *Intuit.ru*: сайт. - 2021*. - URL: https://intuit.ru/studies/courses/2249/52/lecture/1567* (дата обращения: 25.06.2021)
8. Куроуз Д., Росс К. Компьютерные сети. Настольная книга системного администратора. – М.: Издательство «Эксмо», 2016. - с. 912,
9. *Neerc.ifmo.ru:* Протоколы *SSL* и *TLS. -* 2021. *- URL: https://neerc.ifmo.ru/wiki/index.php?title=SSL/TLS* (дата обращения: 25.06.2021)
10. Абросимов Л.И. Базисные методы проектирования и анализа сетей ЭВМ. - М.: Университетская книга, 2015.
11. *Proverkassl.com*: История *TLS* протокола и *SSL* сертификатов. - 2021. - *URL*: https://proverkassl.com/book\_knowledge\_tls.html (дата обращения: 25.06.2021).
12. *Freier A. The Secure Sockets Layer (SSL) Protocol Version 3.0. / A. Freier, P. Karlton. - IETF, 2011. - 67 p. - URL: https://datatracker.ietf.org/doc/html/rfc6101* (дата обращения: 25.06.2021)
13. *Dierks T. The TLS Protocol. Version / T. Dierks, S. Allen. - The Internet Society, 1999. - p. 80. - URL: https://datatracker.ietf.org/doc/html/rfc2246* (дата обращения: 25.06.2021)
14. *Dierks T.* *The Transport Layer Security (TLS) Protocol Version 1.1 - The Internet Society, 2006. - p. 87. - URL: https://datatracker.ietf.org/doc/html/rfc4346* (дата обращения: 25.06.2021)
15. *Dierks T. The Transport Layer Security (TLS) Protocol / T. Dierks. - The Internet Society, 2008. - p. 101. - URL: https://datatracker.ietf.org/doc/html/rfc5246* (дата обращения: 25.06.2021)
16. *Rescorla E. The Transport Layer Security (TLS) Protocol Version 1.3 / E. Rescorla. - Standards Track, 2018. - p. 160. - URL: https://datatracker.ietf.org/doc/html/rfc8446* (дата обращения: 25.06.2021)
17. *Rtfm.co.ua: What is: SSL/TLS* в деталях*. -* 2018*. - URL: https://rtfm.co.ua/what-is-ssl-tls-v-detalyax/* (дата обращения: 25.06.2021)
18. *Apigee.com: About TLS/SSL. - 2021. - URL: https://docs.apigee.com/api-platform/system-administration/about-ssl* (дата обращения: 25.06.2021)
19. Официальный сайт компании *Oracle. Working with Certificates and SSL. - 2021. - URL: https://docs.oracle.com/cd/E19830-01/819-4712/ablqw/index.html* (дата обращения: 25.06.2021)
20. *Tadviser*: Интернет вещей *Internet of Things (IoT)*. - 2021. - *URL:* *https://www.tadviser.ru/index.php/Статья:Интернет\_вещей,\_IoT,\_M2M\_(мировой\_рынок)* (дата обращения: 25.06.2021).
21. *Steve’s Internet Guide: MQTT Client and Mosquitto Broker Message Restrictions With Examples. - 2021. - URL: http://www.steves-internet-guide.com/mqtt-broker-message-restrictions* (дата обращения: 25.06.2021)
22. *MQTT*: официальный сайт проекта. - 2021. - *URL: https://mqtt.org* (дата обращения: 25.06.2021)
23. *EMQ Technologies*: официальный сайт. - 2021. - *URL: https://www.emqx.io/*  (дата обращения: 25.06.2021).
24. *Mininet Project*: сайт. - 2021. - *URL: http://mininet.org*/ (дата обращения: 25.06.2021)
25. *Shelby Z. The Constrained Application Protocol (CoAP) / Z. Shelby, C. Bormann, K. Hartke. - Standards Track, 2014. - p. 100. - URL: https://tools.ietf.org/html/rfc7252* (дата обращения: 25.06.2021)
26. *Kipp E.B. Hickman. The SSL Protocol / Kipp E.B. Hickman. - URL: https://datatracker.ietf.org/doc/html/draft-hickman-netscape-ssl-00* (дата обращения: 25.06.2021)
27. Гойхман В., Савельева А. Аналитический обзор протоколов Интернета вещей. // Технологии и средства связи. - 2016. № 4. С. 32-37
28. *OASIS Standard, 2019. - URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html* (дата обращения: 25.06.2021)
29. Eclipse Mosquitto: официальный сайт проекта https://mosquitto.org/
30. *Wemos*: официальный сайт компании. - 2021. - *URL: https://www.wemos.cc/en/latest/d1/d1\_mini.html* (дата обращения: 25.06.2021)