HTTP/2 для IoT

Камбаров Д.К.

Московский Государственный Университет им. М.В.Ломоносова,

***Аннотация* — HTTP/2 - Вторая крупная версия сетевого протокола HTTP, используемая для доступа ко Всемирной сети. Данный протокол был разработан разработчиками HTTP working group и основан на SPDY(протокол прикладного уровня, разработанный Google). HTTP/2 был разработан для решения проблем и недостатков HTTP/1. К тому же, данный протокол позволяет улучшить производительность и реализацию в системах IoT(Internet of Things). Данная статья является обзором возможностей HTTP/2 для систем IoT.**

***Ключевые слова*—HTTP/2, IoT.**

I. Введение

На сегодняшний день все виды физических и виртуальных объектов объединены в Интернете.
Появилась новая модель, названная Интернетом Вещи (IoT). Данная модель объединяет устройства в компьютерную сеть и позволяет им собирать, анализировать, обрабатывать и передавать данные другим объектам через программное обеспечение, приложения или технические устройства. По оценкам Intel, около 26 умных объектов для каждого человека будут подключены в Интернет к 2020 г.

Интеграция огромного количества устройств с разнородными функциями и возможностями в Интернете требует полностью взаимосвязанной и масштабируемой структуры. Основной проблемой для IoT является интеграция маломощных устройств и сетей с низкой пропускной способностью. Важно отметить необходимость поддержки стека протоколов, который поддерживает разнородные подключенные устройства и требовании в приложениях. Рис. 1 показывает стек протоколов для IoT, который включает различные стандартизированные протоколы на транспортном, сетевом и канальном уровнях, где каждый уровень способствует решению хотя бы одной проблемы таких как масштабируемость, надежность и безопасность.

Обычно в системах Интернета вещей используются IP-совместимые открытые протоколы, которые стандартизированы для широкого деплоя. Несмотря на то, что протоколы на основе IP требуют большого объема памяти, подход на основе IP получает все большее распространение в крупномасштабных деплоях в контексте интеллектуальных сервисов. В дополнение к протоколу передачи гипертекста (HTTP), который де-факто является транспортным протоколом Интернета, были предложены различные протоколы, такие как протокол ограниченного приложения (CoAP) [1] и передача телеметрии очереди сообщений (MQTT) [2], направленные на удовлетворение требований, возникающих в результате интеграции устройств с ограниченными ресурсами.

17 февраля 2015 года утвердили протокол HTTP/2, разработанный HTTP working group, который эффективнее использует ресурсы сети и снижает задержку сети с помощью механизма сжатия полей заголовка и разрешения нескольких одновременных обменов по одному и тому же соединению.

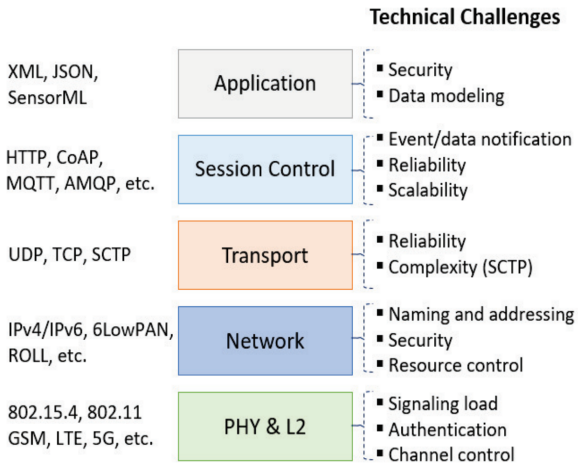

Рис. 1, Проблемы на каждом уровне стека протоколов.

Данная статья является обзором использования протокола HTTP/2 для IoT. Обзор был выполнен на основе изучения статей, связанных с данной темой.

II. HTTP/2 и другие протоколы.

A. Классификация IoT протоколов

После появления и развития Интернета вещей стало понятно, что не все существующие протоколы способны обеспечивать выполнение задач, возложенных на технологии распределенных вычислений, встроенные датчики, современные беспроводные технологии.
Существуют различные подходы к классификации протоколов Интернета вещей. Один из них предполагает объединение протоколов по месту применения в клиент-серверной архитектуре сети:

- D2D (Device to Device) – протоколы взаимодействия оконечных устройств между собой;
- D2S (Device to Server) – для передачи данных, собранных устройствами на серверы для обработки;
- S2S (Server to Server) – протоколы взаимодействия серверов друг с другом.Один из них предполагает объединение протоколов по месту применения в клиент-серверной архитектуре сети:
- D2D (Device to Device) – протоколы взаимодействия оконечных устройств между собой;
- D2S (Device to Server) – для передачи данных, собранных устройствами на серверы для обработки;
- S2S (Server to Server) – протоколы взаимодействия серверов друг с другом.

Самые широко используемые протоколы прикладного уровня для устройств и приложений IoT являются MQTT (сетевой протокол, работающий поверх TCP/IP, ориентированный на обмен сообщениями между устройствами по принципу издатель-подписчик.), CoAP (веб-протокол передачи, использующий протокол UDP), AMQP.

B. Протокол HTTP/2

Протокол HTTP/2 был разработан HTTP working group и основан на Google SPDY протоколе. Вторая версия протокола HTTP направлена на обеспечение более эффективного использования сетевых и серверных ресурсов и уменьшения задержки за счет введения средства сжатия полей заголовка и обеспечения возможности одновременного обмена множественных запросов между браузерами и веб-сайтом по одному соединению. HTTP/2 используется на 47.2% всех веб-сайтов [3]. Также, существуют работы, анализирующие влияние особенностей HTTP/2 на веб производительность. Результаты, представленные в [3], показывают, что механизмы, используемые в HTTP/2, могут привести к явному повышению производительности во время загрузки веб-страницы. Более того, тесты, представленные в [4], показывают, что HTTP/2 имеет лучшие показатели энергопотребления, чем HTTP/1.1.

Основные нововведения вошедшие в HTTP/2, которые внесли вклад для IoT решений:

1) Сжатие заголовка и бинарное кодирование

Каждая HTTP передача содержит набор заголовков, которые описывают переданный ресурс и его свойства. В HTTP/1.x эти метаданные всегда отправляются в виде простого текста и добавляют от 500 до 800 байтов служебных данных на передачу, а иногда и больше килобайт, если используются файлы cookie HTTP. Чтобы уменьшить эти издержки и повысить производительность, HTTP/2 сжимает метаданные заголовка запроса и ответа, используя формат сжатия HPACK.

2) Потоки и мультиплексирование

Мультиплексирование – это процесс передачи списка HTTP запросов в рамках одного TCP соединения. В HTTP/1.1 для каждого запроса требуется устанавливать отдельное TCP-соединение. HTTP/2 вводит понятие потока: каждый клиентский запрос назначается конкретному потоку, и все потоки мультиплексируются через одно TCP-соединение. Как следствие, запросы не влияют друг на друга, и сервер может одновременно отвечать на них. Следовательно, HTTP / 2 решает проблему блокировки заголовка строки, в то же время уменьшая количество TCP-соединений, обрабатываемых на стороне сервера. Кроме того, это улучшает время задержки по сравнению с функцией конвейерной обработки, которая была представлена в HTTP 1.1.

3) Серверный Push

Еще одна особенность HTTP/2 – Серверный Push, который позволяет серверу отправлять множественные ответы на один клиентский запрос. К тому же сервер может отправить дополнительные ресурсы клиенту без надобности клиенту запрашивать ресурсы по отдельности.

III. HTTP/2 для IoT

В обозримом будущем для IoT-ландшафта потребуется несколько стеков. Очевидно, что одного стека будет недостаточно приводит к размышлению о каноническом стеке, основанном на основных протоколах. Это один из вариантов обслуживания IoT системы в будущем.

Цели создания канонического стека, следующие:

- Максимальное использование элементов, основанных на стандартах, в различных технологиях и стеках IoT
- Уменьшение особенностей и специфичности протокола IoT
- Сокращение количества трансляторов, необходимых в IoT хабе.

Канонический стек должен быть должным образом профилирован и оптимизирован. Это требует оптимизации таких аспектов, как:

- Структура аутентификации и авторизации путем адаптации OAUTH вместо изобретения новой системы
- Управление устройствами и объектная модель / описания (в настоящее время определяется).
- Транспорт приложений на основе HTTP/2

В этой статье рассматривается только транспорт прикладного уровня на основе HTTP/2.

HTTP/2 хорошо подходит для IoT по нескольким причинам:

- Двоичный и компактный (заголовок 9 байт)
- Сжатие заголовка
- Обход межсетевых экранов через TLS через порт 443
- Поддержка модели RESTful в основных фреймворках разработки
- Ноу-хау широко доступны

HTTP / 2 имеет множество настраиваемых параметров, которые могут улучшить его производительность для приложений IoT за счет уменьшения требований к пропускной способности, кодовому пространству и оперативной памяти. В частности, следующие настройки и значения оказались полезными в приложениях IoT:

- ***SETTINGS_HEADER_TABLE_SIZE***: этот параметр позволяет хостам ограничивать размер таблицы сжатия заголовка, используемой для декодирования, уменьшая требуемую оперативную память, но потенциально увеличивая требования к пропускной способности. Начальное значение для HTTP/2– 4096. Сценарии IoT могут выиграть от изменения этого значения на меньшее (например, 512), однако, чтобы избежать увеличения использования полосы пропускания, в сценариях IoT следует разумно использовать заголовки HTTP и таблицу динамических заголовков [RFC7541].
- ***SETTINGS_ENABLE_PUSH***: этот параметр позволяет клиентам включать или отключать принудительную отправку сервера. Эта функция может не требоваться в некоторых приложениях Интернета вещей. Начальное значение для HTTP/2 - 1.
- ***SETTINGS_MAX_CONCURRENT_STREAMS***: этот параметр позволяет отправителю ограничивать количество одновременных потоков, которые получатель может создать для соединения. HTTP / 2 рекомендует, чтобы это значение было не меньше 100. IoT сценарии могут захотеть ограничить его гораздо меньшим числом, например 2 или 3.
- ***SETTINGS_INITIAL_WINDOW_SIZE***: этот параметр позволяет хостам ограничивать окно управления потоком, потенциально снижая требования к буферам за счет потенциально недостаточно используемых продуктов задержки полосы пропускания. Для HTTP/2 начальное значение составляет 2^16 – 1 (65 535) октетов. Сценарии IoT могут захотеть ограничить это меньшими значениями в соответствии с ограничениями узла (например, несколькими кило октетами).
- ***SETTINGS_MAX_FRAME_SIZE***: этот параметр позволяет хостам указывать наибольший размер кадра, который они хотят получить. Для HTTP/2 начальное значение составляет 2^14 (16 384) октета. Как это ни парадоксально, хосты IoT могут пожелать оставить это значение большим и полагаться на управление потоком, чтобы избежать ненужных накладных расходов на кадрирование.
- ***SETTINGS_MAX_HEADER_LIST_SIZE***: этот параметр позволяет хостам ограничивать максимальный размер списка заголовков, который они хотят получить. Для HTTP / 2 начальное значение этого параметра не ограничено. Сценарии IoT могут захотеть ограничить это меньшими значениями в соответствии с ограничениями узла (например, несколькими килооктетами).

Распространение приложений и протоколов безопасности в IoT привело к деплою островков устройств IoT, каждый из которых использует один из нескольких доступных протоколов. Однако обычно при деплое IoT необходимо взаимодействовать с другим или, по крайней мере, необходимо взаимодействовать с Интернетом, потому что они должны загружать данные в облако или потому, что обычно они контролируются веб-приложением.

В таких случаях обмен данными обеспечивается межпротокольным прокси-сервером или шлюзом, транслирующим синтаксис и семантику одного протокола в другой. Однако наличие межпротокольных шлюзов или шлюзов приложений имеет как минимум два основных недостатка, которые необходимо тщательно проанализировать и устранить.

- Хотя перевод может быть тривиальным для основных сценариев, во многих случаях перевод может привести к потере информации или несовместимости из-за того, что разные прокси-серверы выполняют перевод по-разному.
- Наличие таких устройств также может стать критическим моментом для безопасности.

IV. Экспериментыи производительность

В данной секции предоставлены результаты и методы выполнения тестирования для сравнения HTTP/2 и HTTP/1 [6]. Результаты, представленные в этом разделе, основаны на следующих предположениях и соображениях:

- Накладные расходы от TCP и TLS игнорируются
- Была предпринята попытка минимизировать используемые заголовки, сохранив при этом соответствие RFC.
- В динамическую таблицу HTTP/2 не делается никаких записей, что устраняет некоторую потенциальную оптимизацию.
- Установление и разрыв соединения игнорируются, хотя очевидно, что это важные соображения для протоколов приложений IoT.
- Отображаются только успешные передачи, поэтому сравнение режимов отказа или повторных передач не проводится.

A. Пример Get запроса

В этом первом примере сравнивается и противопоставляется метод GET ресурсу, содержащему XML-представление простого переключателя, использующего HTTP/1.1 и HTTP/2.

(1) HTTP/1.1

- Клиент отправляет (47 октетов):
- Сервер отправляет (107 + 36 октетов):

```
4745 5420 2f6f 6e6f 6666 2048 5454 502f 312e 310d
0a48 6f73 743a 2066 6f6f 0d0a 4163 6365 7074 3a20
2a2f 2a0d 0a0d 0a
```

```
GET /onoff HTTP/1.1\r\n
Host: foo\r\n
Accept: */*\r\n
\r\n
```

```
4854 5450 2f31 2e31 2032 3030 204f 4b0d 0a44 6174
653a 204d 6f6e 2c20 3039 204d 6172 2032 3031 3520
3036 3a32 363a 3434 2047 4d54 0d0a 436f 6e74 656e
742d 4c65 6e67 7468 3a20 3336 0d0a 436f 6e74 656e
742d 5479 7065 3a20 6170 706c 6963 6174 696f 6e2f
786d 6c0d 0a0d 0a
3c4f 6e4f 6666 3e0a 093c 7374 6174 653e 6f66 663c
2f73 7461 7465 3e0a 3c2f 4f6e 4f66 663e
```

```
HTTP/1.1 200 OK\r\n
Date: Mon, 09 Mar 2015 06:26:44 GMT\r\n
Content-Length: 36\r\n
Content-Type: application/xml\r\n
\r\n
<OnOff>\n
\t<state>off</state>\n
</OnOff>
```

(2) HTTP/2

- Клиент отправляет (34 октета):
- Сервер отправляет (54 октета):
- Сервер отправляет (45 октетов):

```
0000 1901 0500 0000 01
8286 0585 60f5 1e59 7f01 8294 e70f 0489 f963 e7ef
b401 5c00 07
```

```
:method: GET
:path: /onoff
:scheme: http
:authority: foo
accept: */*
```

```
0000 2d01 0400 0000 01
880f 1296 d07a be94 03ea 681d 8a08 016d 4039 704e
5c69 a531 68df 0f10 8b1d 75d0 620d 263d 4c79 a68f
0f0d 8265 cf
```

```
:status: 200
content-type: application/xml
content-length: 36
date: Mon, 09 Mar 2015 06:26:44 GMT
```

```
0000 2400 0100 0000 01
3c4f 6e4f 6666 3e0a 093c 7374 6174 653e 6f66 663c
2f73 7461 7465 3e0a 3c2f 4f6e 4f66 663e
```

```
<OnOff>
\t<state>off</state>
</OnOff>
```

B. Сравнение

В целом, без учета полезной нагрузки (36 октетов) поток HTTP/2 на 37% меньше потока HTTP / 1.1. Использование дополнительных заголовков, особенно общих заголовков, присутствующих в статической таблице HTTP/2, приводит к большей экономии. Также, способность HTTP / 2 повторно использовать соединения для нескольких потоков снижает накладные расходы на установление соединения, такие как установление соединения TCP и установление сеанса TLS.

V. Заключение

На данный момент многие веб-браузеры поддерживают протокол HTTP/2, и все больше сайтов используют его для улучшения работы пользователя. HTTP/2 может значительно улучшить производительность сети. В данной статье были рассмотрены главные особенности HTTP/2 протокола и его использование в IoT, а также было произведено сравнение HTTP/1.1 и HTTP/2 протоколов.

Библиография

[1] Z. Shelby, K. Hartke, and C. Bormann, “RFC 7252: The Constrained Application Protocol (CoAP).” 2014. W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[2] International Business Machines Corporation (IBM), “MQTT V3.1 Protocol Specification,” 2010.

[3] W3Techs, “Usage Statistics of HTTP/2 for Websites, May 2021.” [Online]. Available: https://w3techs.com/technologies/details/ce-http2

[4] H. de Saxce, I. Oprescu, and Y. Chen, “Is HTTP/2 really faster than HTTP/1.1?,” in 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2015, pp. 293–299.

[5] Asma Elmangoush, “Evaluating the Features of HTTP/2 for the Internet of Things” *IEEE J. Quantum Electron.*, submitted for publication.

[6] G. Montenegro, S. Cespedes, R. Simpson, “H2oT: HTTP/2 for the Internet of Things”

[7] Google HTTP/2 Documentation,

[8] M. Belshe, R. Peon, and M. Thomson, “RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2).” 2015.

[9] H. de Saxce, I. Oprescu, and Y. Chen, “Is HTTP/2 really faster than HTTP/1.1?,” in 2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2015, pp. 293–299

[10] Путь к HTTP/2. Available: https://habr.com/ru/post/308846/