UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

HTTP/2 PARA INTERNET DE LAS COSAS: ANÁLISIS, EVALUACIÓN Y ADAPTACIÓN DEL CONTROL DE FLUJO

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA

DIEGO FERNANDO LONDOÑO DELGADO

PROFESORA GUÍA:
SANDRA CÉSPEDES UMAÑA
PROFESOR CO-GUÍA:
JAVIER BUSTOS JIMÉNEZ

MIEMBROS DE LA COMISIÓN:
CLAUDIO ESTÉVEZ MONTERO
NICOLÁS HIDALGO CASTILLO

Este trabajo ha sido parcialmente financiado por Niclabs Chile y Fondecyt a través del proyecto 11140045

SANTIAGO DE CHILE
2020

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE MAGÍSTER EN CIENCIAS DE LA INGENIERÍA, MENCIÓN ELÉCTRICA
POR: DIEGO FERNANDO LONDOÑO DELGADO
FECHA: 2020
PROF. GUÍA: SANDRA CÉSPEDES UMAÑA
PROF. CO-GUÍA: JAVIER BUSTOS JIMÉNEZ

HTTP/2 PARA INTERNET DE LAS COSAS: ANÁLISIS, EVALUACIÓN Y ADAPTACIÓN DEL CONTROL DE FLUJO

El término *Internet of Things* (IoT) apareció en 1999 y poco tiempo después diferentes entidades y consorcios empezaron a crear estándares para este campo. En contraste con el modelo tradicional en el que personas se comunican entre sí, en IoT las máquinas se comunican entre ellas (*Machine to Machine Communications*, M2M); sin embargo, esas suelen tener limitaciones en términos de batería, procesamiento, almacenamiento y potencia de transmisión. Tales limitaciones han llevado a replantear el uso de protocolos ampliamente utilizados en Internet y crear nuevos protocolos especialmente diseñados para escenarios con restricciones.

Actualmente, hay una propuesta en la *Internet Engineering Task Force* (IETF) que busca adaptar la segunda versión del *Hypertext Transfer Protocol* (HTTP/2) a IoT, dado que es un protocolo compacto y configurable. Dicha propuesta es soportada por cuatro argumentos principales: seguridad, interoperabilidad, conocimiento de HTTP y el amplio uso del mismo. HTTP/2 provee más agilidad a las aplicaciones web actuales, ya que introduce nuevos parámetros, funcionalidades y reduce las cabeceras al adoptar un formato binario.

Esta tesis plantea una propuesta de adaptación de HTTP/2 a IoT. Parte de la propuesta que está en la IETF, donde plantean un perfil de configuración para los parámetros del protocolo. Se propone un algoritmo de control de flujo para escenarios restringidos que busca simplificar el tráfico, reduciendo el *overhead*, sin sacrificar el *throughput*. La metodología está basada en los principios de diseño de protocolos y enfocada en las pruebas, simulaciones y los análisis de resultados que ofrecen el rigor necesario.

A mi familia.

Agradecimientos

Al momento de escribir esto se me vienen a la cabeza todas las personas que de una u otra forma me ayudaron para que, al fín, esta tesis, y esta etapa, fuera concluida. Debo agradecer, primero, el apoyo incansable de mis padres, mis hermanos y mi familia. Desde mi primer día en la U de Chile hasta el último.

Agradezco a la profesora Sandra, de no ser por su apoyo, a veces inmerecido, este trabajo no hubiera sido posible. Agradezco su orientación y paciencia no sólo en estos años de magíster, sino desde que yo era un estudiante de pregrado en Icesi, en Colombia. Agradezco también a Javier, mi profesor co-guía, y al NIC Labs, por apoyarme, acogerme y brindarme un espacio.

Cuando decidí venir a estudiar a Chile, en medio de las expectativas y temores, normales supongo, no imaginaba que estos serían algunos de los mejores años de mi vida. Debo agradecer a quienes estuvieron ahí para que esto fuera así: Pablo, Adri, Jeisson, gracias por todo el apoyo tanto en las tareas académicas como por fuera de ellas, gracias por la compañía, las charlas, las risas, las cervezas, los asados, las caminatas al cerro... Y obviamente gracias a Sergio, parce, llegué a tu departamento buscando una habitación y encontré un amigo. Gracias a vos y a tu familia por acogerme y hacerme sentir como en casa, gracias por todo el aprecio, por los carretes, las piscolas, los partidos de tenis, las fiestas patrias, el viaje al sur y tantas experiencias que me hicieron querer esta ciudad y este país.

Gracias a Pauli por su apoyo, por estar pendiente de mi, darme ánimo, visitarme y compartir tiempo conmigo. Fue una prueba dura, pero salimos fortalecidos :*

Gracias a mis compañeros del NIC Labs, especialmente a Maite; gracias a Paty, por su amabilidad y estar ahí siempre dispuesta a ayudar cada vez que lo necesité, a mis demás amigos chilenos, Diego, Fer y Thamara, por la acogida y hacerme sentir como un chileno más. Finalmente gracias a Nico y Mancho por llevarme a la U a trabajar en esto, ya en Cali, casi todos los días.

Tabla de Contenido

1. Introducción **5**
- 1.1. Motivación . . . 6
- 1.2. Definición del problema . . . 8
- 1.3. Hipótesis . . . 8
- 1.4. Solución propuesta . . . 8
- 1.5. Objetivos . . . 9
  - 1.5.1. Objetivo general . . . 9
  - 1.5.2. Objetivos específicos . . . 9
- 1.6. Metodología y herramientas . . . 9
  - 1.6.1. Metodología . . . 9
  - 1.6.2. Herramientas . . . 11

2. Marco Teórico **14**
- 2.1. Dispositivos restringidos . . . 14
- 2.2. Modelos de servicio . . . 15
  - 2.2.1. Representationtal State Transfer . . . 15
  - 2.2.2. Publish/Subscribe . . . 15
- 2.3. IoT y Web of Things . . . 16
  - 2.3.1. Arquitectura de IoT . . . 16
  - 2.3.2. Pila de protocolos en IoT . . . 18
- 2.4. Tecnologías LoWPAN . . . 18
  - 2.4.1. IEEE 802.15.4 . . . 19
  - 2.4.2. ZigBee . . . 19
  - 2.4.3. Bluetooth Low Energy . . . 20
  - 2.4.4. WiFi . . . 21
- 2.5. Tecnologías LPWAN . . . 22
  - 2.5.1. LoRaWAN . . . 22
  - 2.5.2. SIGFOX . . . 23
  - 2.5.3. NB-IoT . . . 24
  - 2.5.4. Wi-SUN . . . 24
- 2.6. Capa de adaptación . . . 25
  - 2.6.1. 6LoWPAN . . . 25
  - 2.6.2. Grupos de trabajo 6lo y lpwan . . . 26
- 2.7. Capa de enrutamiento y transporte . . . 27
  - 2.7.1. RPL . . . 27
  - 2.7.2. TCP . . . 27

2.7.3. UDP . . . 27
2.8. Capa de seguridad . . . 28
2.8.1. TLS . . . 28
2.8.2. DTLS . . . 28
2.9. Protocolos de Aplicación . . . 29
2.9.1. CoAP . . . 29
2.9.2. MQTT . . . 31
2.9.3. DDS . . . 34
2.9.4. AMQP . . . 34
2.9.5. XMPP . . . 35
2.9.6. HTTP/1.1 . . . 36
2.9.7. SPDY . . . 37
2.9.8. HTTP/2 . . . 39

3. Revisión del estado del arte **41**
3.1. Evaluación de protocolos de aplicación en ambientes IoT . . . 41
3.2. Comparación de protocolos de aplicación vía simulaciones y experimentos . . . 43
3.3. Diseño de protocolos . . . 45

4. Estudio de parámetros de HTTP/2 en IoT **49**
4.1. Diseño de los experimentos . . . 49
4.2. Discusión de resultados . . . 50

5. Adaptación de HTTP/2 a entornos IoT **54**
5.1. Control de flujo . . . 54
5.2. Propuesta de algoritmo de control de flujo . . . 54
5.3. Análisis teórico de desempeño . . . 58
5.4. Evaluación mediante simulaciones . . . 62
5.4.1. Algoritmo de referencia . . . 62
5.4.2. Simulador . . . 64
5.4.3. Simulaciones . . . 64
5.4.4. Comparación de resultados teóricos y simulados . . . 73
5.4.5. Otros parámetros a considerar . . . 80

6. Conclusiones y trabajo futuro **81**
6.1. Conclusiones . . . 81
6.2. Trabajo futuro . . . 82

A. Anexos **83**
A.1. Códigos de MATLAB para algoritmo propuesto . . . 83
A.1.1. Código del algoritmo propuesto usado cuando el receptor es más rápido 83
A.1.2. Codigo del algoritmo popuesto usado cuando el receptor es más lento 84
A.1.3. Código del algoritmo de NGHTTP2 cuando hay cuello de botella . . . 85
A.2. Código de MATLAB para el algoritmo de NGHTTP . . . 85
A.3. Resultados de simulaciones . . . 86

Bibliografía **89**

Índice de Tablas

2.1. Pila de protocolos en IoT. Adaptado de [14] 18
2.2. Formato de la trama de la capa de red en ZigBee. 20
2.3. Formato de la trama de soporte de aplicación en ZigBee. 20
2.4. Resumen de tecnologías LPWAN 25
2.5. Estructura de un paquete MQTT. Adaptado de [74] 33
2.6. Arquitectura de capas de AMQP. *El protocolo de transporte opera sobre TCP 34
2.7. Formato de encabezado del paquete. Adaptador de [43] 35
2.8. Estructura del frame en AMQP. Adaptado de [43] *DOFF = Data Offset . . 35

3.1. Resultados de evaluación de seguridad de IPSec y DTLS utilizando X.805. Adaptado de [56]. *PKI = Public Key Infrastructure 42
3.2. A. Resumen de trabajos de evaluación y comparación de protocolos 46
3.3. B. Resumen de trabajos de evaluación y comparación de protocolos 47

4.1. Resumen de configuración de experimentos CoAP vs HTTP/2 50
4.2. Resumen de configuración de experimentos en Raspberry Pi 51
4.3. Resultados ante petición GET 51
4.4. Resultados ante petición POST 51

5.1. Resumen de tiempos implicados en la transmisión. 55
5.2. Resumen de parámetros evaluados en las simulaciones. 64

A.1. Throughput en Bytes variando tamaño velocidades de envío y lectura, tamaño de ventanas y delay. 87
A.2. Overhead en Bytes usado al variar velocidades de envío y lectura, tamaño de la ventana y delay. 88

Índice de Ilustraciones

1.1. Raspberry Pi 3 Modelo B 12

2.1. Arquitectura simplicada de IoT. Adaptado de [5][6][7] 18
2.2. Capas de CoAP. Tomado de [86] 30
2.3. Estructura de paquete CoAP. Tomado de [86] 30
2.4. Flujo MQTT utilizando QoS de nivel 0 32
2.5. Flujo MQTT utilizando QoS de nivel 1 33
2.6. Flujo MQTT utilizando QoS de nivel 2 33
2.7. Trama de control en SPDY. Tomado de [8] 37
2.8. Trama de datos en SPDY. Tomado de [8] 38
2.9. Trama de HTTP/2. Tomado de [25] 39

4.1. Escenarios implementados en las pruebas 50
4.2. Tiempo de respuesta, Tiempo de CPU y Trabajo versus Tamaño de ventana en el cliente 52
4.3. Porcentaje CPU versus Tamaño de ventana en cliente y servidor 52

5.1. Esquema simplicado de un algoritmo de control de flujo 55
5.2. Flujo de paquetes en el algoritmo propuesto 56
5.3. Representación de una cola M/M/1/K. 60
5.4. Flujo de paquetes en el algoritmo de referencia. 62
5.5. *Throughput* cuando el receptor es más lento en relación al emisor 65
5.6. *Overhead* cuando el receptor es más lento en relación al emisor 67
5.7. *Throughput* cuando el receptor es más lento en relación al emisor y el tiempo de propagación es 200 ms 69
5.8. *Overhead* cuando el receptor es más lento en relación al emisor y el tiempo de propagación es 200 ms 70
5.9. *Throughput* cuando el receptor y emisor tienen velocidades similares 71
5.10. *Overhead* cuando el receptor y emisor tienen velocidades similares 72
5.11. *Throughput* cuando el receptor y emisor tienen velocidades similares y el tiempo de propagación es 200 ms 74
5.12. *Overhead* cuando el receptor y emisor tienen velocidades similares y el tiempo de propagación es 200 ms 75
5.13. Resultados cuando el receptor es más rápido que el emisor 76
5.14. Resultados cuando el receptor es más rápido que el emisor y el tiempo de propagación es 200 ms. 77
5.15. Comparación de resultados simulados y teóricos cuando el receptor es más rápido 78

5.16. Comparación de resultados simulados y teóricos cuando el receptor y emisor tienen velocidades similares. 78
5.17. Comparación de resultados simulados y teóricos cuando el emisor es más rápido. 79

Acrónimos

3GPP 3rd Generation Partnership Project

ACK Acknowledgment

AMQP Advanced Message Queuing Protocol

AP Algoritmo propuesto

ASCII American Standard Code for Information Interchange

AS Access Stratum

BLE Bluetooth Low Energy

BS Base Station

CBOR Concise Binary Object Representation

CoAP Constrained Application Protocol

CoRE Constrained RESTful Environments

CRC Cyclic Redundancy Check

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

CSS Chirp Spread Spectrum

DAG Directed Acyclic Graph

DCPS Data-Centric Publish-Subscribe

DDS Data Distribution Service

DECT Digital Enhanced Cordless Telecommunications

DNS Domain Name Server

DTLS Datagram Transport Layer Security

EUA Estados Unidos de América

FAN Field Area Network

FDD Frecuency-Division Duplexing

GFSK Gaussian Frequency Shift Keying

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IEEE Institute of Electric and Electronics Engineers

IETF Internet Engineering Task Force

IoT Internet of Things

IPSec Internet Protocol Security

IPv6 Internet Protocol version 6

IP Internet Protocol

ISM Industrial, Scientific and Medical

ITU International Telecommunications Union

LAN Local Area Network

LLC Logical Link Control

LLN Low Power and Lossy Network

LoRa Long Range

LoWPAN Low-Power Wireless Personal Area Networks

LPWAN Low-Power Wide Area Network

LR-WPAN Low-Rate Wireless Personal Area Network

LTE Long Term Evolution

M2M Machine to Machine

MAC Media Access Control

MHDS Multi-Hop Delivery Service

MIMO Multiple Input Multiple Output

MQTT Message Queue Telemetry Transport

MS/TP Master-Slave/Token-Passing

MU-MIMO Multi-User MIMO

NAS Non-Access Stratum

NAT Network Address Translation

NB-IoT Narrowband Internet of Things

OFDMA Ortogonal Frequency Division Multiple Access

OSCOAP Object Security of CoAP

OSI Open System Interconnection

P2P Peer-to-Peer

PAN Personal Area Network

PER Packet Error Rate

PHY Physical layer

PKI Public Key Infrastructure

QAM Quadrature Amplitude Modulation

QoS Quality of Service

RAM Random Access Memory

RA Registration Authority

REST Representational State Transfer

RFC Request For Comments

RFID Radio Frequency Identification

ROM Read Only Memory

RPL Routing Protocol for Low-Power and Lossy Networks

RTT Round Trip Time

SASL Simple Authentication and Security Layer

SC-FDMA Single Carrier Frequency Division Multiple Access

SC Service Center

SoAP Simple Object Access Protocol

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

UE Unión Europea

ULE Ultra Low Energy

UNB Ultra Narrow Band

URI Universal Resource Identifier

URL Uniform Resource Locator

W3C World Wide Web Consortium

WiFi Wireless Fidelity

WLAN Wireless Local Area Network

WoT Web of Things

WSN Wireless Sensor Network

XML Extensible Markup Language

XMPP Extensible Messaging and Presence Protocol

ZDO ZigBee Device Object

Capítulo 1

Introducción

Internet of Things (IoT) es un paradigma cuya idea básica plantea la conexión a Internet de una nueva variedad de "cosas" u "objetos", como sensores, motores, *tags* RFID, electrodomésticos, máquinas, vehículos, etc. De manera que puedan enviar y recibir información, interactuar y cooperar entre ellos. Este nuevo campo abre muchas posibilidades de aplicaciones en sectores tales como: viviendas y edificios, industria, minería, el sector agrícola y forestal, infraestructura urbana y transporte.

IoT implica que el número de conexiones a Internet originadas por cosas sea mucho mayor a las conexiones originadas por humanos; así, las comunicaciones y el tráfico *Machine-to-Machine* toman mayor protagonismo, pero al mismo tiempo, se generan retos en cuanto a *big data*, protocolos de comunicación, seguridad, etc.

Los dispositivos de IoT son típicamente restringidos, es decir, tienen poca memoria, poca capacidad de procesamiento, poca potencia de transmisión y se alimentan por medio de baterías, de las que se espera que no requieran ser cambiadas en meses o incluso años. Las anteriores restricciones han llevado a replantear el uso de los protocolos más difundidos en Internet, como lo son HTTP/1.1 y TCP, porque al no ser diseñados en principio para ese tipo de dispositivos, pueden resultar costosos en términos de exigencia de cómputo para ellos. Es por esto que diferentes entidades y consorcios iniciaron hace ya más de una década esfuerzos por crear estándares para IoT; de esta manera surgieron por ejemplo: IEEE 802.15.4 en la capa física y de acceso, 6LoWPAN como una capa de adaptación entre la capa de enlace y la de red y *Constrained Application Protocol* (CoAP) y *Message Queue Telemetry Transport* (MQTT) en la capa de aplicación, entre otras tecnologías y protocolos.

Algunas funcionalidades de los protocolos recientemente creados para IoT ya existían en protocolos más antiguos, con la ventaja de estos últimos de ser ampliamente conocidos y probados. Por este motivo, algunos autores han señalado la viabilidad de adaptar HTTP/2 a IoT. Las razones para hacerlo pasan por la interoperabilidad, seguridad, conocimiento y amplio uso del protocolo. El presente trabajo aborda esta discusión y busca generar una propuesta de adaptación de HTTP/2 a IoT, partiendo de un recorrido por el marco teórico de lo concerniente a IoT, las diferentes pilas de

protocolos que se manejan en este campo y los trabajos relacionados.

Este trabajo se centra en diseñar una extensión para HTTP/2, con el objetivo que opere en dispositivos restringidos. Por lo tanto, se parte de los principios de arquitectura para este fin establecidos en [32], donde señalan tener en cuenta la operatividad, interoperabilidad, seguridad, desempeño, etc. al momento de hacer una extensión a un protocolo. La extensión propuesta es probada en esta tesis, para lo cual se utiliza un esquema de pruebas ágiles basada en un modelo A/B dónde según determinadas métricas se identifica el mejor desempeño de dos variantes del protocolo. De esta manera se espera llegar a una propuesta sólida, que comprende los valores específicos de los parámetros y cambios estructurales y funcionales de HTTP/2 para tener un buen funcionamiento en ambientes restringidos para IoT.

1.1. Motivación

Internet of Things (IoT) es definido por la ITU-T [55] como la red de objetos físicos y virtuales embebidos con electrónica, software, sensores, actuadores, y conectividad de red, que permiten a tales objetos recolectar e intercambiar datos. Esto sugiere un significativo cambio de paradigma, ya que hasta hace poco las comunicaciones en Internet se hacían entre computadoras o dispositivos manejados por humanos. Lo anterior abre un amplio campo de acción con aplicaciones en diversos sectores, como transporte, domótica, agricultura, gestión de recursos naturales, industria, etc.

El número de dispositivos conectados a Internet crece día a día. Pronósticos indican que entre el 2020 y el 2021 habrán alrededor de 30 mil millones de cosas conectadas a Internet y para el año 2025 se estima que este número supere los 70 mil millones [73] [88]. Esto evidencia el crecimiento y potencial que tiene IoT, pronto la mayor parte de dispositivos conectados no serán computadores, *tablets* o *smartphones* controlados por humanos, sino otras cosas que tradicionalmente no habían estado conectadas. Una gran cantidad de dispositivos conectados se traduce también en una gran cantidad de información generada, es por esto que IoT tiene una estrecha relación con *Big Data* y computación en la nube. De esta manera, hay un cambio estructural de las comunicaciones, y así lo han entendido los fabricantes quienes desde hace algunos años ya han definido sus estrategias para estar en el negocio.

El futuro es promisorio, hay tantas aplicaciones como se pueda imaginar, sin embargo, aún hay retos pendientes, algunos de ellos surgen de las características de los dispositivos. En *Internet of Things* se utilizan típicamente dispositivos con capacidades reducidas, es decir, alimentados con baterías, con RAM inferior a 50 kB, ROM inferior a 250 kB y poca capacidad de procesamiento [27]. Esto ha llevado a repensar el uso de protocolos ampliamente utilizados en Internet, como TCP y HTTP, en IoT. Inicialmente se ha descartado el uso de este tipo de protocolos en escenarios restringidos por ser pesados para las capacidades de los dispositivos, y se ha buscado crear nuevos protocolos a lo largo de las diferentes capas del modelo OSI. Así, gracias a los esfuerzos de diferentes entidades de estandarización, fueron creadas tecnologías

como IEEE 802.15.4 en la capa física y de acceso, protocolos como 6LoWPAN entre la capa de red y acceso, y más recientemente han sido creados nuevos protocolos de capa de aplicación como: *Constrained Application Protocol* (CoAP) [86] y *Message Queue Telemetry Transport* (MQTT) [74].

CoAP fue estandarizado por la IETF en junio de 2014. Es un protocolo sencillo y liviano que funciona sobre UDP con el fin de minimizar la carga a los dispositivos. Está basado en *Representational State Transfer* (REST), al igual que HTTP. En cuanto a funcionalidades, CoAP es un protocolo similar a HTTP y desde un principio se ha diseñado para que pueda convivir fácilmente con este protocolo por medio de *proxies* [86]. Hay que resaltar, entonces, que se haya tenido que crear un protocolo liviano que incluye funcionalidades que HTTP ya las realizaba, y aunque HTTP no fue creado para IoT, es el protocolo de capa de aplicación más utilizado en este campo según la encuesta *IoT Developer Survey 2019* [43].

Según el RFC 5218 [90], cuando un protocolo es tan exitoso que excede las áreas para las que fue originalmente creado, es un buen momento para revisarlo con el fin de adaptarlo a las nuevas áreas de uso. En ese sentido, es lógico pensar en una adaptación de HTTP a IoT y es precisamente esa la propuesta que se ha recopilado en el *Internet-draft* [69]. Las razones que sustentan estas propuestas son cuatro: seguridad, interoperabilidad, conocimiento del protocolo y el amplio uso del mismo. En cuanto a la seguridad, la creación de nuevos protocolos y pilas implica también nuevas formas de ataques, con el agravante de que IoT es un escenario de alto riesgo, debido a las restricciones de los dispositivos y a su amplia distribución geográfica que posibilita el acceso de atacantes. Respecto a la interoperabilidad, tener una amplia variedad de pilas y protocolos puede dificultar la comunicación de las partes. Por otro lado, utilizar la tecnología mejor conocida tiene beneficios en cuanto a que hay más personas con las habilidades para trabajar con ella, además de más implementaciones, algunas de ellas *Open Source*, hechas y probadas. Finalmente, HTTP es un protocolo ampliamente usado en la web y se espera que esa tendencia se mantenga en los próximos años.

Una eventual adaptación de HTTP a IoT se haría partiendo de la segunda versión, HTTP/2, al ser una versión más reciente, compacta, configurable, ágil y con significativas mejoras sobre su versión anterior, HTTP/1.1. Entre esas mejoras se encuentran: cabeceras binarias en lugar de ASCII; multiplexión de *streams*, lo que ahorra conexiones TCP; priorización de *streams*; control de flujo en la capa de aplicación y nueva funcionalidades como Push Promise, que permite ahorrar peticiones del cliente al servidor. Lo anterior hace a HTTP/2 un protocolo más ágil y liviano que su antecesor, pero aún así no es un protocolo diseñado para IoT. En el *Internet-draft "HTTP/2 Configuration Profile for the Internet of Things"* [69] se propone la redefinición de los valores por defecto de algunos parámetros, sin embargo, estos cambios necesitan ser probados y mejor definidos. No obstante, además de los valores de los parámetros, es necesario evaluar otras posibles modificaciones o extensiones que se le puedan hacer a HTTP/2 para lograr su operación en ambientes restringidos para IoT. El impacto potencial que podría tener una eventual estandarización de HTTP/2 para IoT es grande. HTTP es un protocolo que los desarrolladores conocen y que tienden a usar,

muestra de ello son las estadísticas actuales [43], donde se refleja, incluso, un alto uso de HTTP/1.1, siendo este un protocolo menos eficiente que HTTP/2. Finalmente, las primeras pruebas de HTTP/2 sobre dispositivos restringidos han arrojado buenos resultados, lo que es un motivante para continuar indagando en este campo.

1.2. Definición del problema

Ante las restricciones de los dispositivos de IoT, se han creado nuevos protocolos, algunos de ellos con funcionalidades que ya tenían otros protocolos existentes, como HTTP/2. Actualmente ya se han hecho estudios donde se evalúan funcionalidades de HTTP/2 con objeto de usar este protocolo en escenarios restringidos. Aunque dichos estudios han dado algunos indicios del funcionamiento de HTTP/2 en IoT, aún no se conoce concretamente cuáles son los cambios en estructura, operación y parámetros que debería tener el protocolo para un correcto funcionamiento en escenarios restringidos.

1.3. Hipótesis

A partir del problema identificado donde se manifiesta que actualmente no está definido el funcionamiento del control de flujo de HTTP/2 para escenarios restringidos, se hace necesario establecer un algoritmo que reduzca el *overhead* sin sacrificar el desempeño en términos de *throughput*. En ese sentido, esta tesis pretende probar que con un algoritmo de control de flujo sencillo y orientado a escenarios restringidos se puede reducir el *overhead* en un 30 % sin un gran sacrificio en términos de *throughput*.

1.4. Solución propuesta

HTTP/2 puede ser adaptado para que logre un desempeño adecuado en dispositivos con capacidades restringidas de IoT. Para esto es necesario identificar y evaluar los valores de los parámetros del protocolo, además de cambios adicionales en su estructura y funcionamiento. Lo anterior será realizado bajo principios de buenas prácticas de diseño de protocolos y ejecutando las pruebas necesarias para no comprometer la interoperabilidad, seguridad y desempeño de la versión de HTTP/2 actual.

1.5. Objetivos

1.5.1. Objetivo general

Diseñar, implementar y evaluar una adaptación del control de flujo de HTTP/2 para IoT, donde se redefinan algunos valores de los parámetros del protocolo y se sugieran extensiones adicionales a su estructura y operación, reduciendo así el overhead del protocolo y conservando un *throughput* aceptable para los escenarios restringidos.

1.5.2. Objetivos específicos

- Documentar un estado del arte de lo concerniente a protocolos para dispositivos restringidos, especialmente HTTP/2 y su control de flujo, para determinar si existe o no un mecanismo orientado a escenarios restringidos.
- Establecer y evaluar mediante mecanismos experimentales los valores óptimos de la ventana de control de flujo y algunos otros parámetros de HTTP/2 que permitan su funcionamiento adecuado sobre dispositivos restringidos.
- Proponer y modelar un algoritmo de control de flujo de HTTP/2 que se adapte a las necesidades de los escenarios restringidos.
- Evaluar comparativamente el algoritmo propuesto con un algoritmo de control de flujo que esté actualmente funcionando en una implementación de HTTP/2.

1.6. Metodología y herramientas

1.6.1. Metodología

HTTP es un protocolo ampliamente utilizado en Internet, es por esto que cualquier modificación que se le haga debe tener un cuidado especial, ya que podría crear problemas de interoperabilidad, seguridad, compatibilidad, etc. que podrían llegar a comprometer la infraestructura de toda la Internet [32]. Dado que esta tesis busca una propuesta de adaptación de HTTP/2 a IoT, es necesario tener en cuenta algunas consideraciones de diseño y extensión de protocolos, métricas de evaluación, pruebas y análisis de datos. Permitiendo de esta manera una propuesta robusta que no suponga riesgos para el funcionamiento del protocolo.

Según el RFC 5218 [90], cuando un protocolo es tan exitoso que excede los escenarios para los que fue pensado, es un buen momento para revisar el protocolo y ajustarlo a los nuevos escenarios en los que se usa. Al plantear de un rediseño o extensión de un protocolo, es necesario tener en cuenta las consideraciones de arquitectura consignadas en el RFC 6709 [32]:

- La extensibilidad debe ser limitada: el protocolo no se va a extender más allá de lo necesario.
- Interoperabilidad global: un protocolo abierto a toda la Internet.
- Compatibilidad arquitectural: una extensión a HTTP/2 deber ser compatible con el protocolo base.
- Evitar variaciones de protocolos: protocolos que son muy similares al original pero no son compatibles deben ser evitados.
- Validar propuesta: para determinar si la extensión ha sido propiamente implementada se deben hacer las pruebas correspondientes.

El mismo RFC hace referencia a otras consideraciones como: no comprometer la interoperabilidad con protocolos subyacentes, evitar problemas de compatibilidad por cambios de la sintaxis, no generar vulnerabilidades al variar el modelo de seguridad, no impactar las implementaciones actuales y no desmejorar el desempeño actual del protocolo.

Partiendo de las anteriores consideraciones de extensibilidad de protocolos y considerando que la etapa de pruebas y simulaciones es esencial en el presente trabajo, se definen las siguientes métricas para la evaluación de desempeño del protocolo:

- Trabajo que implica la utilización del protocolo medido en *Joules* (J).
- Tiempos de respuesta ante solicitudes medido en segundos (s).
- Throughput para determinar el volumen de información intercambiada medido en bits por segundo (bps).
- Goodput para determinar el volumen de la carga útil medido en bps.
- Tasa de paquetes perdidos.
- Porcentaje de uso de CPU. Esta métrica depende de cada dispositivo, y está sujeta a que esté disponible en el mismo.

Las pruebas constan de las siguientes etapas: documentación y requerimientos, diseño de las pruebas, ejecución, análisis de resultados y documentación de estos. Se hacen siguiendo un modelo de pruebas ágiles, que es iterativo entre las etapas, permitiendo que se lleven a cabo de manera más rápida [57]. Las pruebas consisten principalmente de un modelo A/B, utilizado normalmente en páginas web [54]. Este tipo de pruebas consiste, originalmente, en comparar dos versiones de página web, con diferencias puntuales y establecidas previamente, al mismo tiempo y con igual número y tipos de usuarios, con el fin de determinar cuál tiene mejor desempeño dadas las métricas elegidas. En el caso de una prueba de protocolo se utiliza un conjunto de peticiones y operaciones como tráfico de entrada. Se utiliza una configuración A del protocolo y posteriormente el mismo tráfico es utilizado en la configuración B, que es la misma configuración excepto por algún valor de parámetro del protocolo u otra variación según sea el caso. Siguiendo las recomendaciones de [26], se hace un análisis de confiabilidad consistente en una prueba exhaustiva de todas las iteraciones posibles del protocolo. Esta prueba se hace a la versión final de la propuesta.

Para la propuesta de algoritmo de control de flujo, dicho algoritmo se modela

matemáticamente y luego es probado en simulación por medio de un código hecho en Java. Con el objetivo de hacer una evaluación comparativa, también se modela matemáticamente un algoritmo de referencia que está en uso en una implementación de HTTP/2, dicho algoritmo también es simulado. Las simulaciones y modelamientos permiten variar los distintos parámetros que intervienen en la transmisión, por otro lado, las métricas usadas para evaluar el desempeño son *throughput* y *overhead*.

1.6.2. Herramientas

En el marco del desarrollo de la presente tesis se hace uso de diferentes herramientas software y hardware. Las herramientas hardware son utilizadas para correr las pruebas de protocolos y para labores complementarias a las mismas. En cuanto al software, la situación es similar, se utilizan implementaciones de HTTP/2 y otras orientadas a labores complementarias como monitoreo de paquetes y de otras variables.

Software:

En el software se puede hacer la distinción de implementaciones de HTTP/2 y software para labores complementarias. En el primer grupo se tienen:

- Nghttp2: Es una implementación *Open Source* de HTTP/2 hecha en lenguaje C [71]; cuenta con la implementación del cliente, servidor y un *proxy*. Esta implementación es utilizada para realizar pruebas de cliente y servidor sobre Raspberry Pi. Permite la configuración de los parámetros del protocolo, tales como: tamaño de ventanas, número de streams concurrentes, *push promise*, entre otras.
- Apache Server: Implementación *Open Source* de un servidor HTTP/2 [18], desarrollado por el *Apache HTTP Server Project*. Es una implementación robusta del protocolo que es utilizada en algunas pruebas sobre computadores sin limitaciones de hardware.
- Mozilla Firefox: Navegador web utilizado sobre computadores sin limitaciones hardware para probar las respuestas del servidor implementado. Tienen la limitación de sólo funcionar con TLS.
- Simulador de control de flujo: implmentación de elaboración propia hecha en Java, donde se simula el comportamiento de dos algoritmos de control de flujo diferentes. Permite editar variables como cantidad de Bytes a enviar, tamaño inicial de la ventana, tamaño de paquetes, tiempos de propagación y velocidad de transmisión y lectura.

En cuanto al software de soporte se utiliza:

- Wireshark: Es un software para el análisis de protocolos. Con este se captura tráfico mientras las pruebas se ejecutan, para posteriormente hacer análisis detallado de los paquetes. Permite el análisis de paquetes de HTTP/2, 802.15.4, ZigBee, además de la pila tradicional de protocolos. En el caso de HTTP/2,

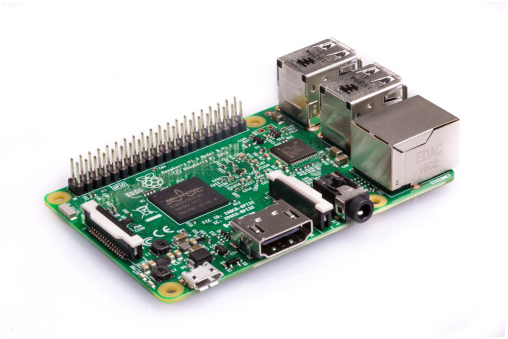


Figura 1.1: Raspberry Pi 3 Modelo B

Wireshark muestra estadísticas del número de paquetes intercambiados en cada stream y su respectivo tipo.

- Eclipse: Es una plataforma de software de código abierto que ofrece distintas herramientas para programación. En el caso del presente trabajo, se usa el IDE de Java para el desarrollo de las simulaciones de los algoritmos de control de flujo.
- Microsoft Excel: Software de hojas de cálculo usado para el análisis de los resultados de las pruebas y simulaciones.

Hardware:

- Raspberry Pi 3 Modelo B: La Raspberry Pi es un minicomputador orientado al desarrollo de proyectos y a la enseñanza de las ciencias de la computación. Es elaborada por la Fundación Raspberry Pi del Reino Unido. Esta placa permite la conexión de periféricos como mouse, teclado a través de puertos USB y pantalla a través de un puerto HDMI, además corre el sistema operativo Linux Raspbian. El modelo utilizado es la Raspberry Pi 3 modelo B, que tiene como principales características:
  - Procesador Quad-core de 1.2 GHz.
  - 1 GB de RAM LPDDR2.
  - Ranura para tarjeta Micro SD donde se instala el sistema operativo.
  - Conexión inalámbrica 802.11 b/g/n.
  - Conexión cableada vía FastEthernet.
  - Bluetooth.
  - 40 pines GPIO.
  - 5V de alimentación vía micro USB.

Tipo de Red

Las redes en las que se probará HTTP/2 en dispositivos restringidos, serán principalmente redes donde se privilegie la comunicación entre dispositivos finales. Debido

a que si bien IoT tiene un componente importante en la nube, el interés de este trabajo es el funcionamiento de HTTP/2 en dispositivos restringidos y la compatibilidad con las implementaciones actuales. La topología usada es estrella, haciendo uso de tecnología WiFi.

Capítulo 2

Marco Teórico

En el marco de IoT es importante tener en cuenta algunas definiciones y conceptos, a continuación se enumeran algunos de los más relevantes.

2.1. Dispositivos restringidos

Según la RFC 7228 [27] los dispositivos restringidos son aquellos que tienen limitaciones en cuanto a la capacidad de cómputo, la batería, la capacidad de comunicarse y la memoria disponible. A menudo son utilizados como sensores o actuadores, objetos inteligentes o dispositivos inteligentes. Los dispositivos restringidos se clasifican en:

- Clase 0: Dispositivos severamente restringidos en memoria y capacidad de cómputo. No pueden conectarse a Internet de manera segura y para esto necesitan otros equipos como *gateways, proxys* o servidores.
- Clase 1: Son dispositivos que no pueden trabajar fácilmente con los protocolos tradicionales de Internet como HTTP, TLS y TCP. Sin embargo, sí lo pueden hacer con protocolos que han sido diseñados para dispositivos restringidos, tales como CoAP sobre UDP.
- Clase 2: Dispositivos con menos restricciones que pueden correr la pila de protocolos tradicional, sin embargo, es deseable que sean más eficientes en el consumo energético y ancho de banda.

Los dispositivos restringidos pueden usar diferentes estrategias de uso energético según sus necesidades, una de ellas sería que el dispositivo esté siempre encendido, otra que esté normalmente apagado y otra de bajo consumo. La primera estrategia puede ser usada cuando no hay requerimientos muy exigentes de ahorro de energía, en ella el dispositivo estará siempre conectado a la red. Cuando el dispositivo usa la estrategia de estar normalmente apagado, estará largos periodos de tiempo dormido y se reconecta sólo cuando lo necesita. Por último, la estrategia de bajo poder es aplicable a dispositivos que requieren operar con una pequeña cantidad de energía, pero

que tienen requerimientos altos de conectividad, haciendo uso de ella. El dispositivo dormirá en periodos cortos entre transmisiones pero conservando de alguna forma la conexión a la red.

2.2. Modelos de servicio

Cliente/Servidor

La arquitectura Cliente/Servidor es la más común en Internet, y ha sido crucial para la expansión de la *World Wide Web*. Por definición, un cliente es un sistema o programa que requiere de uno o más servicios que se ejecutan en otro sistema o programa llamado servidor. De esta manera, el cliente envía peticiones al servidor, este las procesa y envía de vuelta el resultado. Las peticiones/respuesta pueden transmitirse al interior de una Red de Área Local (LAN), o a través de Internet si es el caso [52].

2.2.1. Representationtal State Transfer

Representational State Transfer (REST) es un estilo de arquitectura de software que es a menudo utilizado en el desarrollo de servicios web. Usualmente es preferido sobre SoAP *(Simple Object Access Protocol)*, debido a que REST utiliza menos ancho de banda lo que lo hace más adecuado para Internet. REST destaca las siguientes características [3]:

- Cliente/Servidor sin estado: Se separan los roles, una de las partes presta el servicio y la otra hace uso de él. Cada petición debe contener toda la información necesaria para ser procesada ya que no se guardan estados previos.
- Conjunto de operaciones: En REST la interacción entre clientes y servidores se da a través de un número limitado de operaciones (Verbos) únicas sobre los recursos disponibles en el servicio. A los recursos se les asigna un *Universal Resource Identifier* (URI) para su identificación.

2.2.2. Publish/Subscribe

Las comunicaciones síncronas y punto a punto se orientan a aplicaciones rígidas que añaden dificultad si se requiere un desarrollo de servicios dinámicos a gran escala. En este sentido la arquitectura *Publish/Subscribe* es una opción atractiva. En ella la interacción del emisor y receptor se da a través de un *Middleware* llamado, usualmente, *Broker* al que los suscriptores expresan interés en un evento o patrones de eventos. Cuando el editor o *Publisher* emite un evento, los suscriptores que hayan expresado interés en el evento son notificados por el *Broker*. La ventaja que tiene esta

arquitectura es que relaja en tiempo, espacio y sincronización la comunicación entre editor y suscriptor [45].

2.3. IoT y Web of Things

El término *Internet of Things* se remonta al año 1999, cuando en el Massachusetts Institute of Technology se inician trabajos de identificación por radiofrecuencia. Con el paso del tiempo, se fueron encontrando más aplicaciones en las que "cosas inteligentes" se conectaban a Internet. Entre los objetos se pueden encontrar *pallets*, cajas, vehículos, maquinaria, electrodomésticos, etc. Por otro lado, existen "entidades de interés" como habitaciones, edificios, ríos o lagos, en los que se pueden tomar mediciones de alguna variable o ejercer alguna acción específica mediante sensores y actuadores [20].

Cuando los objetos inteligentes se integran a la web por medio de una conexión a Internet alimentándola de datos del mundo físico, prestando y consumiendo servicios web conectados a través de Internet, aparece el término *Web of Things* (WoT). Cabe recordar que el *World Wide Web Consortium* (W3C) define un servicio web como un sistema de software que soporta comunicaciones *Machine-to-Machine* (M2M) sobre una red.

La conexión a Internet de los objetos puede ser directa o indirecta. En el primer caso, los objetos tienen la capacidad de soportar los estándares de la Web y pueden interactuar con sus servicios, mediante una conexión directa a Internet. En el caso de la conexión indirecta, los objetos no tienen la capacidad de conectarse directamente a Internet, y su interacción con los servicios web se da mediante un intermediario, un ejemplo de este caso pueden ser las etiquetas RFID, que necesitan un lector para extraer y comunicar su información [98].

2.3.1. Arquitectura de IoT

En [22] sugieren que la arquitectura de IoT no es muy diferente a la de la Internet tradicional, en IoT se tiende a "relajar" el rigor de algunos principios, dado también por el mismo dinamismo del avance tecnológico. En el mencionado documento se explora la arquitectura de la Internet tradicional plasmados en la RFC 1958 [33] y la RFC 3439 [31] y su relación con el escenario de IoT. En términos generales la RFC 1958 establece tres principios:

- Conectividad como objetivo.
- *Internet Protocol* (IP) como herramienta.
- Inteligencia es *end-to-end*

En IoT se sigue considerando la conectividad como un objetivo e IP sigue siendo una herramienta. El tercer principio relaja su rigor, debido a que en un ambiente de

IoT, las "cosas" comúnmente tienen capacidades restringidas y esto limita la complejidad de las tareas que pueden realizar.

Las características generales de Internet que se conservan en IoT son:

- Soporte para tecnologías heterogéneas, puesto que toman más relevancia en IoT.
- Se debe evitar el duplicado de protocolos, salvo una buena razón técnica.
- Diseño escalable.
- El desempeño, el costo y la funcionalidad deben ser considerados.
- Simplicidad.
- Modularidad.
- Los parámetros no deberían ser configurados manualmente, sino negociados automáticamente. En IoT, dadas las restricciones de cómputo de los dispositivos, toma relevancia el costo de la negociación y aumenta la probabilidad de romper este principio.
- Ser estricto para enviar y tolerante para recibir.
- Ser mesurado con paquetes no solicitados.
- Se deben evitar dependencias circulares.
- Los objetos deben estar auto-descritos, esto en IoT se cumple parcialmente, debido a que los protocolos buscan ahorrar en tráfico.
- La terminología debe ser consistente.
- No se estandariza nada hasta que hayan múltiples instancias de código corriendo.

La movilidad es un factor importante en el desarrollo de IoT. Actualmente con el desarrollo de *smartphones*, redes vehiculares, drones, etc. se tienen muchos dispositivos que se pueden conectar a Internet, pero su movimiento no permite que lo hagan utilizando una misma infraestructura fija. Debido a esto, los protocolos y aplicaciones deben tener en cuenta que la conexión se hará con múltiples saltos, donde los mismos nodos pueden funcionar también como enrutadores y presentar alta latencia.

En cuanto a los protocolos de aplicación, es conocido que deben estar adaptados a un escenario con restricciones. En [22] plantean que, dado el dominio que tiene HTTP en Internet, es razonable que se cree una versión adaptada para IoT. En la capa de transporte es importante el control de tráfico debido al gran número de dispositivos aunque su tráfico sea poco. En la capa de red 6LoWPAN es un estándar estable. Por el lado comercial, diferentes empresas de tecnología han planteado modelos de infraestructura para IoT [5][6][7], estos modelos son más prácticos y están orientados según los intereses de sus respectivos negocios. En la Figura 2.1 se muestra una versión adaptada de estos.

Cabe resaltar que se pueden tener dos enfoques desde el punto de vista de la arquitectura, uno orientado a Internet, donde toman más relevancia los servicios que se prestan y la manera como se prestan [51], y otro orientado a los objetos inteligentes. El presentado en la Figura 2.1 está orientado a Internet.

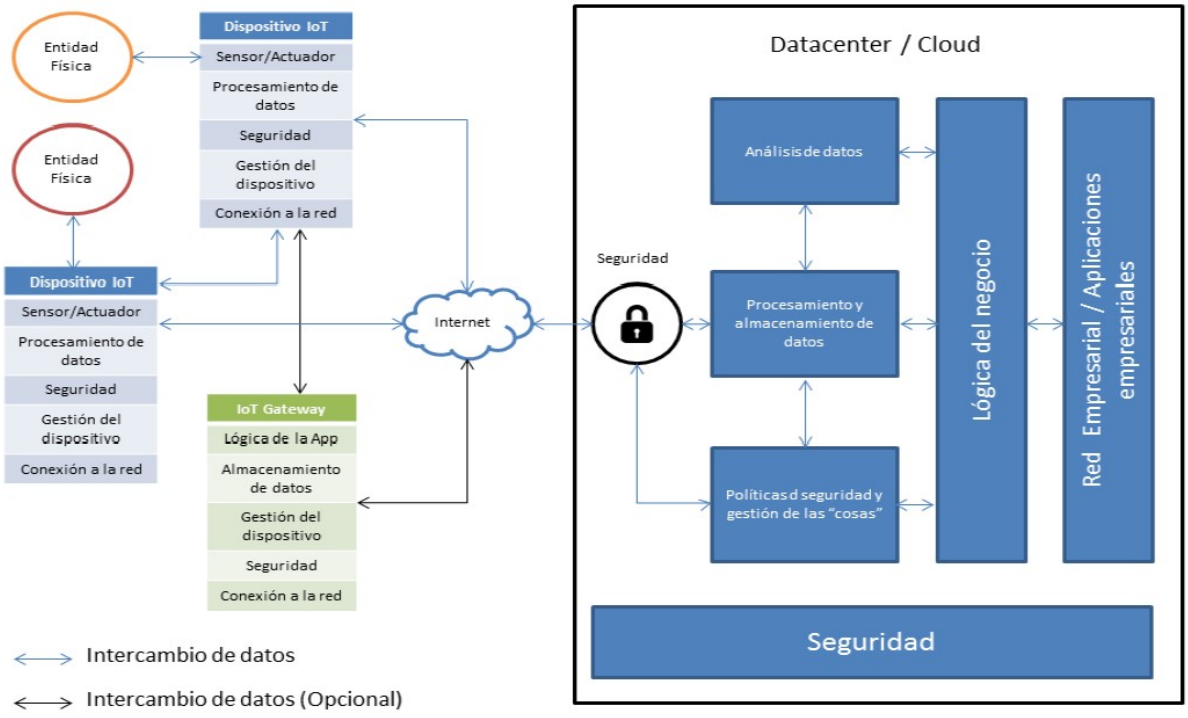


Figura 2.1: Arquitectura simplicada de IoT. Adaptado de [5][6][7]

Tabla 2.1: Pila de protocolos en IoT. Adaptado de [14]

Aplicación	CoAP	DDS	AMQP	MQTT	XMPP	HTTP/1.1	HTTP/2	ZigBee Application
	DTLS	TLS						
Transporte	UDP	UDP/TCP	TCP					ZigBee Network
Red	IPv6	IPv6/IPv4	IPv6	IPv6 / IPv4				
	6LoWPAN / LPWAN							
Enlace	6top / LPWAN							
	IEEE 802.15.4 / LPWAN							
Física	IEEE 802.15.4 PHY / LPWAN							

2.3.2. Pila de protocolos en IoT

En la tabla 2.1 se clasifican por capas algunas de las tecnologías y protocolos más importantes que se utilizan actualmente en IoT.

2.4. Tecnologías LoWPAN

Las redes inalámbricas de área personal y baja potencia (LoWPAN, por su nombre en inglés) son las tecnologías de transmisión inalámbricas para escenarios restringidos más antiguas y usadas. Surgen de la necesidad de tener una tecnología inalámbrica de bajo costo, baja potencia, con cobertura para un área personal o local y una transferencia de datos limitada. Dentro de estas redes se encuentra variedad de topologías y al día de hoy se pueden considerar tecnologías maduras.

A continuación se presentan algunas de las tecnologías LoWPAN más usadas, como lo son: IEEE 802.15.4 y Zigbee (Ambas basadas en 802.15.4) y Bluetooth Low Energy (BLE). Además se presenta IEEE 802.11 (WiFi) debido a que suele ser usada también en aplicaciones de IoT.

2.4.1. IEEE 802.15.4

Es una tecnología que define la capa de Control de Acceso al Medio (MAC) y la Capa Física (PHY) para redes privadas de área local con baja tasa de transmisión (LR-WPAN). Se caracteriza por ser una tecnología simple y flexible, de bajo consumo de potencia, baja tasa de transmisión y bajo costo. Se define dos tipos de dispositivos: *Full Function Device*, los cuales son capaces de funcionar como coordinadores de la red de área personal (PAN); y los *Reduced Function Device*, que no pueden servir como coordinadores y son utilizados en aplicaciones simples haciendo uso de pocos recursos [13]

Dependiendo los requerimientos de la aplicación, 802.15.4 puede operar en dos topologías: en estrella y en *peer-to-peer* (P2P).

- Estrella: El coordinador es el controlador principal de la PAN y los demás dispositivos se conectan a él. El coordinador puede, a menudo, estar alimentado energéticamente de manera constante, los demás con baterías.
- P2P: Aún se conserva el coordinador de la PAN, pero en este caso todos los dispositivos se pueden conectar entre si mientras estén en el rango.

El coordinador es el encargado de la creación, el control y mantenimiento de la red. Cuando se crea una red PAN, se selecciona un identificador único que permite la comunicación entre dispositivos dentro de la red utilizando una dirección corta. Las tasas de transmisión dependen de la banda de frecuencia a la que funcione, de esta manera se tiene: 250 kbps a 2.4 GHz, 40 kbps a 915 MHz y 20 kbps a 868 MHz. En 802.15.4 se utiliza CSMA/CA con el fin de reducir el riesgo de colisión.

2.4.2. ZigBee

Zigbee [100] es un estándar de comunicaciones inalámbricas, de bajo costo y bajo consumo energético, creado por la Alianza Zigbee. Su arquitectura está compuesta por bloques, conocidos como capas. En el estándar IEEE 802.15.4-2003 están definidas las capas inferiores, que son la capa física (PHY) y la subcapa de control de acceso al medio (MAC). La Alianza Zigbee se encargó de definir la capa de red y el *framework* para la capa de aplicación. Este *framework* está conformado por la subcapa de soporte de aplicación (APS) y los objetos del dispositivo Zigbee (ZDO). Cabe destacar que lo más importante en la comunicación entre dispositivos a través de una red Zigbee es el acuerdo sobre un perfil, que define acciones comunes entre los dispositivos.

Tabla 2.2: Formato de la trama de la capa de red en ZigBee.

Campo	Frame de control	Dirección de destino	Dirección de fuente	Radio de Transmisión	Número de Secuencia	Dirección de destino IEEE	Dirección de fuente IEEE	Control de Multicast	Subframe con ruta de fuente	Frame del payload
Tamaño	2 Bytes	2 Bytes	2 Bytes	1 Byte	1 Byte	0/8 Bytes	0/8 Bytes	0/1 Byte	variable	variable
	Encabezado de la capa de red									***Payload***

Tabla 2.3: Formato de la trama de soporte de aplicación en ZigBee.

Campo	Frame de control	Direccionamiento					Contador de la APS	Encabezado extendido	Frame del payload
		Dispositivo destino	Dirección de grupo	Identificador de clúster	Identificador del perfil	Dispositivo origen			
Tamaño	1 Byte	0/1 Byte	0/2 Bytes	0/2 Bytes	0/2 Bytes	0/1 Byte	1 Byte	0/variable	variable
	Encabezado de la sub-capa de soporte de aplicación (APS)								**Payload de la APS**

La capa de red se encarga de asegurar que la subcapa MAC funcione correctamente para interactuar con la capa de aplicación por medio de los servicios de datos y de gestión, que forman la interfaz entre ambas capas. Esta capa admite topologías en estrella, árbol y *mesh*. Cuando se usa la topología de estrella, el responsable de controlar la red, inicializar y mantener los dispositivos de la red es un dispositivo llamado coordinador Zigbee. En las topologías de árbol y *mesh*, el coordinador Zigbee tiene menos responsabilidades y sólo debe iniciar la red y escoger los parámetros clave para esta. Por otra parte, los frames de la capa de red, que se pueden ver en la Figura 2.2, se caracterizan por su encabezado, que contiene el frame de control y la información de direccionamiento y de secuencia, y por su payload variable, porque es el que lleva la información.

La subcapa de soporte de aplicación proporciona un conjunto de servicios, que soportan la interfaz entre la capa de red y la capa de aplicación. Al igual que en la capa de red, dichos servicios son manejados por una entidad de datos y otra de gestión. Un frame de esta subcapa contiene un encabezado, con el frame de control y la información de direccionamiento, y un payload, como se puede observar en la Tabla 2.3.

Los objetos del dispositivo Zigbee se encuentran entre el framework de la aplicación y la subcapa de soporte y proveen una interfaz entre los objetos de aplicación, el perfil del dispositivo y la APS. Sus funciones son inicializar la subcapa de soporte de aplicación, la capa de red y el proveedor de seguridad y reunir la información de configuración de las aplicaciones finales para gestionar la red y la seguridad.

2.4.3. Bluetooth Low Energy

Bluetooth es una tecnología inalámbrica de corto alcance. A partir de la aparición de Bluetooth 4.0 en 2010, también se estandarizó Bluetooth Low Energy (BLE), una versión diseñada para aplicaciones de bajo consumo energético [50]. Puede ser utilizado en múltiples topologías de red como: *point-to-point*, *Broadcast* y *Mesh*. En BLE, como en Bluetooth clásico, existen los roles de maestro y esclavo. El maestro es el encargado de crear la conexión y gestionarla, mientras que el esclavo acepta la conexión a la red y sigue al maestro.

Al igual que el Bluetooth clásico, BLE opera en la banda ISM de 2.4 GHz, sin embargo, tiene algunas características particulares: BLE utiliza 40 canales de 2 MHz, durante los periodos de inactividad se "duerme", reduce el overhead, utiliza *Adaptative frequency hopping* para reducir la interferencia con otras redes, en la modulación usa *Gaussian Frequency Shift Keying* (GFSK) con una tasa de modulación de 1 Mbps y para detectar errores usa un *Cyclic Redundancy Check* (CRC) de 24 bits.

2.4.4. WiFi

El conjunto de estándares IEEE 802.11 definen las capas física (PHY) y de enlace (MAC) para redes de área local inalámbricas (WLAN) que funcionan sobre algunas bandas ISM, principalmente la de 2.4 GHz y la de 5 GHz. La primera versión fue publicada en 1997 y la última versión en el mercado es la IEEE 802.11ac-2013. Pasando así de una tasa de transmisión de 1 o 2 Mbps en la primera versión a tasas del orden de Gbps en las modulaciones más densas de IEEE 802.11ac. Actualmente se trabaja en el estándar IEEE 802.11ax y se espera que esté en el mercado en 2019. En particular, la versión IEEE 802.11ac-2013 se destaca por [56]:

- Opera sólo en la banda de 5 GHz ya que la banda de 2.4 GHz es actualmente susceptible de interferencia debido al amplio despliegue de las versiones anteriores del estándar que funcionan en dicha banda. El funcionamiento en la banda de 2.4 GHz está determinado por la versión 802.11n.
- En IEEE 802.11ac se añaden canales de 80 MHz y 160 MHz, además de los de 20 MHz y 40 MHz ya definidos en la versión 802.11n, incrementando así las tasas de transmisión.
- 802.11ac incrementa el número de streams MIMO hasta 8. Esto quiere decir que un máximo de 8 antenas pueden ser usadas simultáneamente.
- Introduce el Multi-User MIMO (MU-MIMO) donde un transmisor con múltiples antenas puede transmitir datos a varios receptores simultáneamente. Cada receptor puede tener una o varias antenas.
- En 802.11n la modulación más densa es 64 QAM. En 802.11ac se pasa a 256 QAM, esto aumenta la tasa de transmisión en un 33
- IEEE 802.11ac como las versiones anteriores utiliza CSMA/CA como mecanismo de acceso al medio.

La versión IEEE 802.11ax aún está en desarrollo, sin embargo, se espera que introduzca algunas características que permitan elevar la tasa de transmisión alrededor de los 10 Gbps. Se espera que funcione con *Ortogonal Frequency Division Multiple Access* (OFDMA) permitiendo así que múltiples usuarios compartan bloques de radio centrados en un sólo *carrier*. Se espera que continúe con utilizando, como en 802.11ac. MIMO y MU-MIMO. También se introducirá el concepto de reúso espacial, debido a que dentro de escenarios densos el uso de CSMA/CA y una alta potencia de transmisión puede resultar en escenarios con un limitado reúso espacial del espectro. De esta manera es necesario hablar de una adaptación dinámica de la potencia de transmisió que permita el reúso espacial [40] [24].

2.5. Tecnologías LPWAN

Tecnologías como IEEE 802.15.4 y 6LoWPAN cumplen bien la función en redes de área personal, de corto alcance, sin embargo, cuando se habla de redes de área amplia con alcance de varios kilómetros estas tecnologías ya no son suficientes. La tecnología celular aparece, en principio, como una opción en casos donde se requiera una red de gran cobertura, no obstante, esta tecnología es compleja y costosa, además no es ideal cuando los dispositivos son restringidos. Es en esta situación que aparecen las tecnologías *Low Power Wide Area Network* (LPWAN) [46], que permiten formar redes con cobertura de varios kilómetros en dispositivos restringidos, donde el *delay* se pueda tolerar y no se necesiten altas tasas de transmisión. Entre los objetivos y las técnicas de diseño están:

- Largo rango: Las tecnologías LPWAN permiten alcanzar unos pocos kilómetros en entornos urbanos y 10 o más kilómetros en zonas rurales. Para lograr esto se utilizan bandas por debajo de 1 GHz y técnicas de banda angosta y espectro ensanchado.
- Operación de baja potencia: Es una de las principales características de las LPWAN, se espera que los dispositivos funcionen con baterías y que estas duren varios años. Utilizar topologías en estrella y no en *mesh* puede ayudar a reducir el consumo.
- Bajo costo: Se logra que la tecnología sea barata reduciendo la complejidad del hardware, utilizando bandas de licencia libre y trasladando la complejidad a estaciones base, que permiten conectar un gran número de dispositivos.
- Escalabilidad: Técnicas de diversidad, densificación y selección adaptativa del canal.

Entre las tecnologías LPWAN más destacadas se encuentra: LoRaWAN, SIGFOX, NB-IoT y Wi-SUN.

2.5.1. LoRaWAN

LoRaWAN [12] es una tecnología inalámbrica de amplio rango, bajo poder y baja tasa de datos. Normalmente, las redes LoRaWAN se configuran en una topología en estrella, en las cuales los *gateways* o estaciones base retransmiten los mensajes entre los dispositivos finales y un servidor central de red que está en el *backend*. La comunicación entre los dispositivos finales y el *gateway* se da usando la comunicación *single-hop* de LoRaWAN, mientras que la comunicación entre el *gateway* y el servidor se da por medio de enlaces IP. De esta manera, en una implementación LoRaWAN interactuan:

- *End-device*: actúa como cliente y se comunica con las estaciones base.
- *Gateway*: también llamado estación base o concentrador. Es el intermediario entre el servidor de red y los clientes.

- Servidor de red: es el centro de la topología estrella y además quien provee la lógica y control de la red.
- Servidor de aplicación: servidores que están detrás del servidor de red, sobre estos corren las aplicaciones.

LoRaWAN utiliza las bandas ISM 433 MHz y 868 MHz en Europa y 915 MHz en las Américas y utiliza la modulación *Chirp Spread Spectrum* (CSS). Para hacer la red robusta frente a interferencias los dispositivos finales saltan de frecuencia de manera pseudoaleatoria.Además cuenta con seguridad por medio de un mecanismo de llave compartida basado en una clave de 128 bits y alcanza rangos de hasta 15 km en zonas rurales.

2.5.2. SIGFOX

SIGFOX [46] es una tecnología LPWAN propietaria con cobertura en más de 30 países. SIGFOX utiliza tecnología de banda estrecha y opera sobre bandas ISM. Los dispositivos están diseñados para enviar unos pocos Bytes por día, por lo que las baterías pueden durar de 10 a 15 años sin ser reemplazadas. La topología que usa es estrella, donde la capacidad de la estación base no depende del número de dispositivos que conecta, sino la cantidad de mensajes que estos envían. El protocolo de radio está optimizado para el *Uplink* y la cobertura puede variar entre 10 km y 50 km según el entorno (urbano o rural).

La interfaz de radio está basada en *Ultra Narrow Band* (UNB), que permite incrementear el rango de transmisión gastando una cantidad limitada de energía en el dispositivo, además de la coexistencia de un gran número de dispositivos. SIGFOX tiene un modelo de *One-contact-one-network* que permite a los dispositivos conectarse desde cualquier país con cobertura sin necesidad de *roaming* o *handover*. La arquitectura consta de una red de *core* basada en la nube, que permite conectividad global buscando ser lo más transperente posible al dispositivo final y red de acceso. En la red *core* se ubican:

- *Service Center* (SC): es el encargado de la conectividad, la gestión de los dispositivos finales y las estaciones base, además de la conectividad entre estas e Internet.
- Autoridad de registro (RA): es el encargado del control de acceso de los dispositivos finales.

La red de acceso de SIGFOX está comprendida por las estación base (BS) y SC. Las estaciones base cumplen funciones de nivel 1 y 2, mientras que el SC cumple algunas otras de nivel 2 y las de nivel 3. Los dispositivos están asociados al SC y no a una estación base específica, lo que facilita la movilidad de estos. SIGFOX autentica y asegura la integridad del mensaje utilizando un código de autenticación basado en AES-128.

2.5.3. NB-IoT

Narrowband Internet of Things (NB-IoT) [46] es una tecnología LPWAN desarrollada y estandarizada en junio de 2016 por 3GPP. La arquitectura del protocolo de radio se divide en dos planos: el de control y el de usuario. El primero es el encargado de controlar la conexión entre el equipo del usuario y la red. En la parte superior del plano de control está la capa *Non-Access Stratum* (NAS), que es la encargada de la autenticación, el control de seguridad y la gestión de portadoras. En la parte inferior de esta capa está la capa *Access Stratum* (AS), la cual establece la conexión, las portadoras, gestiona el sistema de información y reconfigura. En cuanto al plano de usuario, este es el encargado de transferir los datos del usuario a través de la capa AS. NB-IoT puede ser desarrollado de tres maneras:

- *In-band*: significa que la banda estrecha es desarrollada en la banda de LTE y los recursos son compartidos de manera flexible entre NB-IoT y LTE.
- Guarda banda: utiliza la banda que no es utilizada entre dos portadoras LTE.
- *Standalone*: la banda estrecha se ubica en un espectro dedicado, por ejemplo en la banda GSM de 850/900 MHz.

NB-IoT soporta *half-duplex* con *Frequency-Division Duplexing* (FDD), con una tasa máxima de 60 kbps en *uplink* y 30 kbps en *downlink*. NB-IoT utiliza bandas estrechas de 180 kHz, OFDMA con 15 kHz de espacio entre bandas en el *downlink* y en el *uplink* utiliza SC-FDMA. Para ahorrar energía tiene un modo de ahorro energético en el que los dispositivos emisores permanecen dormidos y sólo se despiertan cuando necesitan enviar una señal.

2.5.4. Wi-SUN

La alianza Wi-SUN [95] [46] es una alianza industrial para *smart city, smart grid* y otras aplicaciones de IoT. Una FAN *(Field Area Network)* Wi-SUN está basada en estándares abiertos y fue desarrollada para aplicaciones como monitoreo y gestión de infraestructura de *smart city,* infraestructura de vehículos eléctricos, infraestructura de medición avanzada, gestión y/o protección de datos entre otras aplicaciones de IoT. Una FAN se caracteriza por ser una red IPv6 inalámbrica en malla con soporte de seguridad.

Algunas de las características principales de Wi-SUN son:

- Rango de cobertura entre 2-3 km en línea de vista, que puede ser extendido por medio de redes multi-salto.
- Ancho de banda de hasta 300 kbps.
- Enlaces de baja latencia (0.02 s) y comunicación bidireccional.
- Bajo consumo de poder en los dispositivos, cuando están durmiendo es de 2 μA y cuando están escuchando es de 8 mA.
- Escalabilidad.

Tabla 2.4: Resumen de tecnologías LPWAN

	LoRAWAN	**SIGFOX**	**NB-IoT**	**Wi-SUN**
Espectro	ISM UE: 433MHz, 868 MHz, EUA: 915 MHz Asia 433 MHz	ISM UE: 868 MHz EUA: 902 MHz	<1GHz (Licenciados)	EUA: 902-928 MHz Japón: 920 MHz UE:868.3 MHz
Rango	Urbano: 5 km Rural: 15 km	Urbano: 10 km Rural: 50 km	<15km	2-3 km
Tasa de datos	0.3-50 kbps (Adaptativo)	100 bps (Up) 600 bps (Down)	150 kbps	Hasta 300 kbps
No. de canales	UE: 10 EUA: 64 (Up), 8 (Down)	360		
Forward Error Correction	Si	No		Opcional
Acceso al medio	Unslotted ALOHA	Unslotted ALOHA	NPRACH	CSMA/CA
Batería	10 años	10 años	10 años	<15 años
Topología	Estrella	Estrella de estrellas	Estrella	Mesh
Tamaño de payload	250 Bytes (Max)	12 Bytes (Up) 8 Bytes (Down)	200 Bytes (Max)	2047 Bytes (Max)
Modulación	CSS	UNB, DBPSK	OFDMA (Down) SCFDMA (Up)	MR-FSK MR-OFDM MR-O-QPSK
Encripción	AES 128b	En capa de aplicación	AKA 128 bits	AES-CCM
Fragmentación		Si, en capa de aplicación		Si, a nivel 6LoWPAN

- Métodos de seguridad como control de acceso a la red por la capa de Enlace de datos, autenticación mutua y establecimiento de un vínculo de seguro por parejas.

La especificación de una FAN provee dos métodos posibles para el enrutamiento de paquetes IPv6: RPL *(Routing Protocol for Low-Power and Lossy Networks)*, usado en la capa de red, y MHDS *(Multi-Hop Delivery Service)*, usado en la capa de enlace. Se afirma también que para usar IPv6 en la capa de red se usa la adaptación definida por 6LoWPAN. En la capa de transporte es posible usar UDP o TCP. En la subcapa *Logical Link Control* (LLC) de la capa de enlace se usa un servicio que sea capaz de soportar 6LoWPAN y en la subcapa MAC se usa el estándar IEEE 802.15.4-2015, donde se define la especificación de SUN FSK. Por tanto, en la capa física se utiliza el esquema de modulación 2-FSK, con un rango de canal de 200-600 kHz, para brindar tasas de datos de 50-300 kbps.

La Tabla 2.4 resume y compara las tecnologías LPWAN mencionadas.

2.6. Capa de adaptación

2.6.1. 6LoWPAN

Utilizar IP ofrece grandes ventajas en términos de compatibilidad con la infraestructura existente: cantidad de tecnologías conocidas y de bajo costo, estándares y

la existencia de herramientas de gestión para este protocolo. Esto hace pensar que cualquier protocolo de otras capas busque ser compatible con IP (actualmente IPv6); en el caso puntual de IEEE 802.15.4 había un problema: entre sus principales características está el reducido tamaño de los paquetes: máximo 127 Bytes. En principio esto implica un problema debido a que contando los Bytes de cabeceras del mismo protocolo más los de protocolos de capas superiores como IPv6, práctimanente no dejan espacio para la carga útil. Esto llevó a la IETF a crear 6LoWPAN con el fin de lograr compatibilidad entre IPv6 e IEEE 802.15.4.

6LoWPAN [68] especifica una capa de adaptación que permite el uso de IPv6 sobre IEEE 802.15.4 mediante la compresión y fragmentación de paquetes IPv6 de hasta 1280 Bytes, logrando así que se ajusten a los 127 Bytes que permite 802.15.4. 6LoWPAN comprime los encabezados, eliminando campos innecesarios y repetitivos, y comprime también las direcciones IPv6 por medio de inferencia a partir de las direcciones MAC. Adicionalmente, define las reglas de fragmentación que permiten que un paquete IPv6 se distribuya en varios de 802.15.4. Los paquetes 6LoWPAN buscan tener lo mínimo necesario para llevar un paquete IPv6 desde el origen al destino, es por esto que las cabeceras de estos pueden variar, por ejemplo, si además de la compresión, requieren información para enrutamiento *Mesh* o fragmentación.

2.6.2. Grupos de trabajo 6lo y lpwan

Ante la diversidad de tecnologías de acceso en IoT, la IETF se ha dado a la tarea de crear estándares que permitan el uso de IPv6 y protocolos de niveles superiores sobre tecnologías ya desarrolladas y que actualmente no lo permiten. Es por esto que se han creado los grupos de trabajo *IPv6 over Networks of Resource-constrained Nodes* (6lo) e *IPv6 over Low Power Wide-Area Networks* (lpwan).

El grupo de trabajo 6lo se enfoca en facilitar la conectividad de IPv6 en dispositivos restringidos, partiendo de la adaptación existente para IEEE 802.15.4, 6LoWPAN, y extendiendola a dispositivos que utilicen diversas tecnologías, tales como: Bluetooth LE [72], transceptores de radiocomunicación digital de corto alcance y banda estrecha, ITU-T G.9959, *rfc7428*, Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE) [67] y sobre redes Master-Slave/Token-Passing (MS/TP) [64].

Las tecnologías LPWAN han aparecido como una opción atractiva al momento desplegar redes en IoT. La amplia cobertura que ofrecen, el muy bajo consumo energético, las topologías en estrella, entre otros, permite crear redes de bajo costo que escalan fácilmente. Sin embargo, esto tiene un precio: cada tecnología define de manera optimizada sus propias capas de enlace, no tienen capa de red y en ocasiones la capa de aplicación se conecta directamente a la capa 2. Es por esto que, no se puede usar IPv6, pero además, las diferencias con IEEE 802.15.4 hace que una adaptación a partir de 6LoWPAN o los trabajos realizados el grupo de 6lo no sea trivial. En consecuencia, el grupo de trabajo lpwan se ha dado a la tarea de facilitar por medio de una capa de adaptación, el uso de IPv6 sobre tecnologías LPWAN como: LoRa, SIGFOX, WI-SUN y NB-IoT.

2.7. Capa de enrutamiento y transporte

2.7.1. RPL

En IoT, las redes de bajo poder y alta tasa de pérdida de paquetes (LLN) son comunes. Estas se caracterizan por utilizar dispositivos restringidos y por la baja transferencia de datos. En el RFC 6550 [96] se define el *IPv6 Routing Protocol for Low-Power and Lossy Networks* (RPL), un protocolo de enrutamiento de vector distancia, diseñado para LLN, donde tales redes comprendan miles de nodos enrutadores. Este protocolo acomoda la red como un Grafo Acíclico Dirigido (DAG), donde la raiz es el nodo receptor, buscando que los caminos para llegar a él desde cualquier otro nodo sean los más cortos. RPL soporta tres patrones de tráfico: punto a punto, punto a multipunto y multipunto a punto [16].

2.7.2. TCP

Transmission Control Protocol (TCP) es un protocolo de la capa de transporte diseñado para proveer un canal *end-to-end* confiable sobre una red no confiable de Internet comprendiendo diversas topologías, anchos de banda, retrasos, tamaños de paquetes y otros parámetros. Formalmente definido en el RFC 793 [78] en septiembre de 1981. Es un protocolo orientado a conexión que garantiza la entrega de los paquetes enviados por medio de *Checksums* y *Acknowledgements*. Provee control de flujo por medio de ventanas deslizantes y tiene la capacidad de reorganizar paquetes que llegan en desorden, ya que cada paquete lleva un número de secuencia. Para establecer una conexión TCP se hace un *handshake*, es decir, un intercambio de paquetes inicial donde se establece el puerto TCP a utilizar. Una vez establecida la conexión, TCP permite multiplexar datos de varias aplicaciones en una misma conexión.

2.7.3. UDP

El *User Datagram Protocol* (UDP) es un protocolo de transporte definido por la IETF en el RCF 768 [77] en 1980. A diferencia de TCP, UDP no es orientado a conexión y no garantiza la entrega de los paquetes o datagramas. Tampoco requiere el establecimiento de una conexión previa ni hace control de flujo. UDP utiliza cabeceras livianas, por lo que añade poca carga a la red.

2.8. Capa de seguridad

2.8.1. TLS

Transport Layer Security (TLS) es un protocolo definido en la RFC5246 [42] que tiene como objetivo establecer una comunicación segura entre dos aplicaciones de comunicación. TLS está compuesto por dos capas: *TLS Record Protocol* y *TLS Handshake Protocol*. El primero está en la parte baja, sobre el protocolo de transporte (Usualmente UDP o TCP), y provee seguridad en las conexiones utilizando dos propiedades básicas:

- La conexión es privada: Se utiliza criptografía simétrica para la encripción de la información. Las llaves son únicamente creadas y mantenidas en secreto para cada conexión y están basadas en la negociación que se hace con otro protocolo (como *TLS Handshake*). El *TLS Record Protocol* puede ser usado también sin encriptación.
- La conexión es confiable: El transporte del mensaje hace chequeo de la integridad del mensaje utilizando la MAC del equipo. Funciones de seguridad HASH son utilizadas para computar las MAC.

TLS Record Protocol es utilizado para encapsular protocolos de capas superiores. Una vez encapsulado el protocolo, el *TLS Handshake Protocol* permite al cliente y el servidor autenticarse y negociar el algoritmo de encriptación y las llaves criptográficas antes de que el protocolo de aplicación transmita el primer byte de información. El *TLS Handshake protocol* tiene tres propiedades básicas:

- La identidad de las partes puede ser autenticada utilizando criptografía simétrica o llaves públicas.
- La negociación del secreto compartido es segura: El secreto negociado es resistente a escuchas de terceros. Para una conexión autenticada el secreto no puede ser obtenido, ni siquiera por un ataque de hombre en el medio.
- La negociación es confiable: Ningún atacante puede modificar la comunicación de la negociación sin ser detectado por las partes.

Una ventaja de TLS es que es independiente del protocolo de aplicación.

2.8.2. DTLS

Datagram Transport Layer Security (DTLS) es un protocolo definido en la RFC6347 [81] que provee privacidad en las comunicaciones de protocolos basados en datagramas. El protocolo permite comunicaciones en un esquema cliente/servidor sin escuchas de terceros, manipulación y falsificación de mensajes. DTLS está basado en TLS y provee garantías de seguridad equivalentes.

La necesidad de este protocolo se da porque TLS requiere un canal orientado a conexión, es decir, un canal que funcione con TCP, sin embargo las aplicaciones que usan UDP han ido creciendo con el tiempo y en un principio no se encontraban soluciones de seguridad que satisficieran esta necesidad.

El principio de DTLS es ofrecer TLS para comunicaciones de datagramas. La razón por la que no se puede usar directamente TLS en datagramas es porque los paquetes pueden perderse o cambiar de orden.

La desconfianza del canal provee en TLS dos problemas que son corregidos en DTLS, el primero es que TLS no permite desencriptación individual de registros, porque el chequeo de integridad depende de un número de secuencia. En DTLS se añade un número explícito de secuencia para suplir la necesidad. Por otro lado, el *handshake* de TLS asume que la comunicación se da en un canal confiable, por lo tanto si un paquete se pierde, fallará. En DTLS se corrige este "problema" activando contadores para hacer el reenvío del mensaje en caso de que no reciban la respuesta esperada.

2.9. Protocolos de Aplicación

2.9.1. CoAP

Constrained Application Protocol (CoAP) es un protocolo de transferencia web especializado para el uso en nodos restringidos en un esquema de comunicación *Machine to Machine* (M2M). Está definido en la RFC 7252 [86]. Estos nodos a menudo tienen microcontroladores de ocho bits y memoria RAM y ROM reducidas.

Las características más relevantes de CoAP son:

- Es un protocolo Web orientado a requerimientos propios de los ambientes restringidos de M2M.
- Funciona sobre UDP, sin embargo, tiene entrega garantizada opcional y soporta peticiones unicast y multicast.
- Permite el intercambio de mensajes asíncronos.
- Tiene una cabecera simple y de baja complejidad.
- Da la posibilidad de proveer seguridad a través de Datagram Transport Layer Security, DTLS, e Internet Protocol Security, IPSec.
- Haciendo uso de un proxy, puede convivir con HTTP.
- Tiene la posibilidad de hacer descubrimiento de recursos alojados en otros nodos.

CoAP añade dos subcapas a la pila de protocolos, la de petición/respuesta y la de mensaje, como lo muestra la Figura 2.2.

El modelo de interacción de CoAP es similar al modelo cliente/servidor de HTTP,

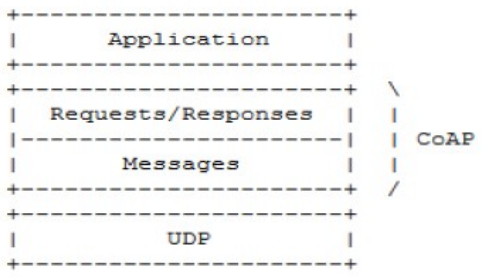


Figura 2.2: Capas de CoAP. Tomado de [86]

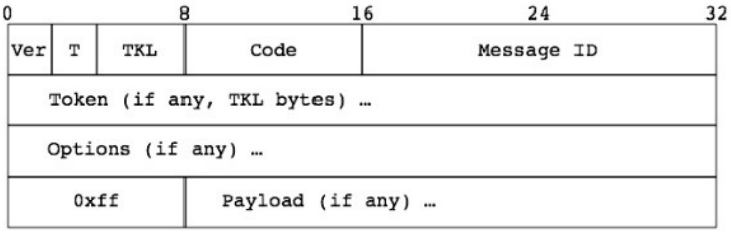


Figura 2.3: Estructura de paquete CoAP. Tomado de [86]

sin embargo, en CoAP los *endpoints* pueden asumir ambos roles. CoAP está basado en REST, por lo que una petición en este protocolo es similar a una en HTTP. Se puede solicitar una acción con un método como GET, POST, PUT, DELETE, o un recurso en un servidor con una URI.

CoAP utiliza una cabecera binaria fija de 4 Bytes, que puede ser seguida por las opciones binarias compactas y la carga útil. Este formato se usa tanto para peticiones como para respuestas. Cada mensaje usa un ID (16 bits, permite 250 mensajes por segundo), para evitar duplicados y otras opciones de confiabilidad. El paquete CoAP se muestra en la Figura 2.3.

CoAP define cuatro tipos de mensajes: Confirmables (CON), No confirmables (NON), *Acknowledgment* (ACK) y *Reset* (RST). Un mensaje CON es retransmitido utilizando *timeout* y *back-off* exponencial entre retransmisiones, hasta que el receptor envíe un ACK con el mismo ID del mensaje. Por otro lado, en un mensaje NON se envían transmisiones que no requieren garantizar entrega (sin ACK), sin embargo, sí se envía con un número de ID para detección de duplicados. Cuando el receptor no puede procesar el mensaje NON envía un RST.

Con el fin de extender las funcionalidades de CoAP y seguir con el proceso de estandarización, en el *Working Group* de CoRE en el primer trimestre del 2017 estaban los siguientes trabajos en progreso:

- *Publish-Subscribe Broker for the Constrained Application Protocol* (CoAP): CoAP funciona originalmente con una arquitectura cliente/servidor, sin embargo, es común que los dispositivos sobre los que funciona tengan largos periodos en los que están “dormidos”, con el objetivo de ahorrar batería. Este comportamiento

supone dificultades para un esquema cliente/servidor dado que necesita sincronización entre las partes. Con el fin de dar solución a esta dificultad, el draft en [58] define la utilización de CoAP con una arquitectura *Publish/Subscriber* mediante la utilización de un *broker*, facilitando así la comunicación muchos-a-muchos de dispositivos con restricciones.

- *CoAP over TCP, TLS, and WebSockets*: CoAP fue diseñado para funcionar sobre UDP, sin embargo, en algunos ambientes UDP no es bien recibido, y puede, por ejemplo, tener problemas de bloqueos o incompatibilidades por parte de *firewalls, proxies* o NATs. Para solucionar este tipo de problemas, el draft en [30] define la utilización de CoAP sobre TCP, TLS y websockets. Este tipo de problemas podrían ser solucionados utilizando HTTP/2, pero a cambio de un costo adicional en los *delays*.
- *CoAP Simple Congestion Control/Advanced*: CoAP está diseñado para funcionar en redes sin congestión, es por esto que en el *draft* en [29] proponen un sistema de control de congestión simple para ambientes restringidos.
- *Patch and Fetch Methods for CoAP*: En algunos casos donde hay datos complejos o muy grandes no se requiere solicitar el recurso completo, sino una parte de él, el draft en [94] define tres nuevos métodos para CoAP: FETCH, PATCH y iPATCH.
- *Guidelines for HTTP-to-CoAP Mapping Implementations*: En el RFC 7252 se define de manera básica el proceso de mapeo para pasar de HTTP a CoAP y viceversa, dejando para el futuro la definición de mayores detalles. El *draft* en [34] pretende dar respuesta a esto, la ventaja de tener una guía definida es que los *proxies* tendrán implementaciones más consistentes, evitando problemas de interpretaciones y compatibilidad.
- *Object Security of CoAP (OSCOAP)*: Este *draft* [85] especifica el *Object Security of CoAP*, (OSCOAP), un método para la protección del intercambio de mensajes CoAP utilizando el formato *CBOR Object Signing and Encryption* [84]. OSCOAP provee encripción *end-to-end* y protección de la integridad de los campos de *payloads*, opciones y cabeceras. Además de una unión segura entre los mensajes de petición y respuesta.

2.9.2. MQTT

Message Queue Telemetry Transport (MQTT) es un protocolo ISO abierto, liviano, con arquitectura *publish/subscriber* diseñado para sensores remotos y dispositivos de control comunicados a través de redes intermitentes con bajo ancho de banda [74].

Una de las características relevantes de MQTT es que utiliza TCP e IP, como HTTP, con la diferencia que su *header* es significativamente más pequeño.

MQTT tiene las siguientes calidades de servicio de entrega de mensajes:

- Máximo una vez: donde los mensajes son enviados de acuerdo a la regla del mejor esfuerzo del ambiente operativo. La pérdida de mensajes puede ocurrir. Puede

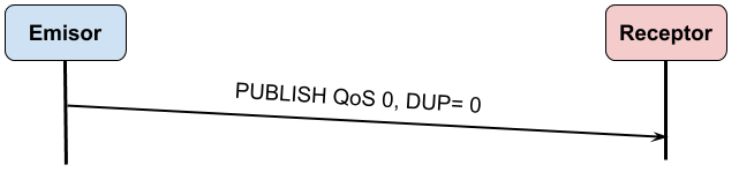


Figura 2.4: Flujo MQTT utilizando QoS de nivel 0

ser usado en ambientes donde no importa si un paquete se pierde.

- Al menos una vez: se asegura la llegada del mensaje, sin embargo, pueden ocurrir duplicados.
- Exactamente una vez: se asegura que el mensaje llega exactamente una vez, es decir, se evitan duplicados o pérdidas de paquetes.

El funcionamiento de MQTT se define a continuación: inicialmente, el cliente envía al Servidor un paquete CONNECT una y sólo una vez, esto siempre y cuando se haya establecido previamente una conexión con la red, ya que, si se envía otro, esto tendrá el efecto contrario y el servidor lo desconectará. Luego, el servidor responde con un paquete CONNACK. Después de esto, se puede enviar un mensaje de aplicación llamado PUBLISH, que puede ir del cliente al servidor o viceversa, el cual no tiene respuesta, si se usa QoS de nivel 0, como se ve en la Figura 2.4, o puede tener como respuesta un PUBACK, cuando se usa QoS de nivel 1, como en la Figura 2.5, o PUBREC, que corresponde a QoS de nivel 2 y es el segundo paquete que se intercambia en este nivel de QoS, reflejado en la Figura 2.6. En los casos en que se recibe un PUBREC, el siguiente paquete es un PUBREL, correspondiente al tercer paquete que se intercambia en QoS de nivel 2, que a su vez se ve seguido por un PUBCOMP, que es el último paquete durante el intercambio de QoS de nivel 2. Otro tipo de mensaje que es posible enviar del cliente al servidor, una vez realizada la conexión, es SUBSCRIBE, la cual se usa para que el cliente le informe al servidor los intereses que tiene y así este le envíe paquetes PUBLISH al Cliente con mensajes de aplicación de acuerdo a la suscripción. La confirmación del paquete SUBSCRIBE es un SUBACK. Para cancelar las suscripciones hechas se usa un UNSUBSCRIBE y su confirmación de recibido es UNSUBACK. Además, existen dos paquetes que se usan para verificar si una conexión está activa, que son PINGREQ y PINGRESP. Finalmente, para terminar la conexión se usa el paquete DISCONNECT.

Los paquetes tienen un encabezado fijo, y algunos, según el tipo de paquete, tienen además un encabezado variable y el *payload*. En la Tabla 2.5 se muestra un resumen de la estructura de los paquetes en MQTT.

En cuanto a la seguridad, MQTT puede usarse con o sin TLS, es por ello que los puertos TCP registrados para MQTT TLS y sin TLS son 8883 y 1883 respectivamente. Cabe destacar que, aunque TLS ayuda a mejorar la seguridad en la parte de autenticación hecha por el servidor al cliente, MQTT es un protocolo que no es confiable simétricamente, es decir, no hay un mecanismo para que el Cliente autentique al Servidor. Adicionalmente, algunas implementaciones MQTT podrían hacer uso de túneles seguros alternativos (por ejemplo, SSH) mediante el uso de SOCKS.

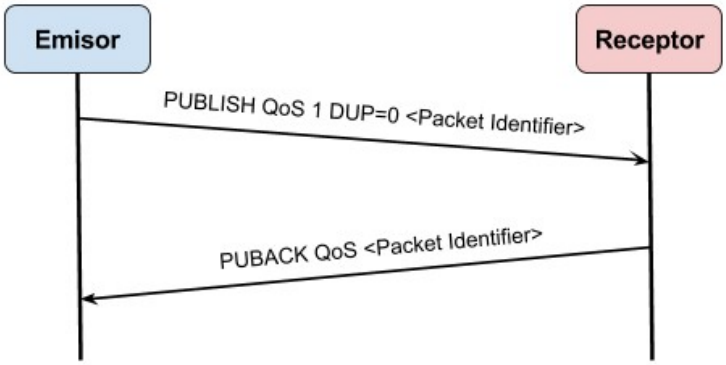


Figura 2.5: Flujo MQTT utilizando QoS de nivel 1

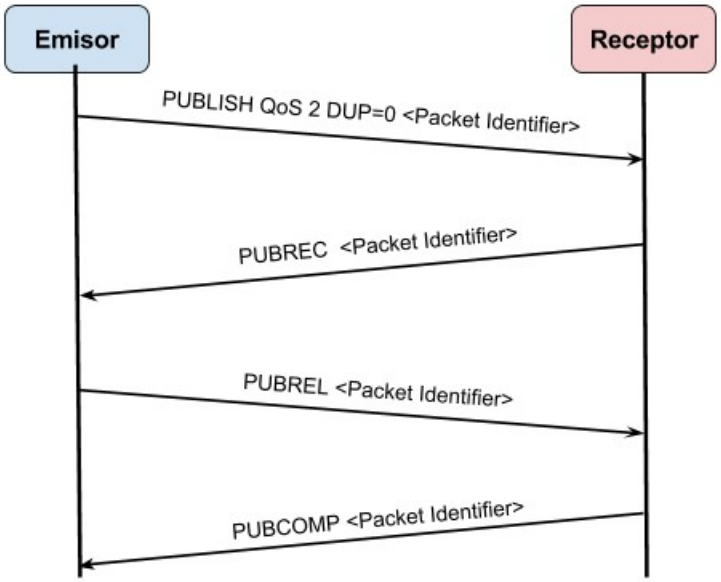


Figura 2.6: Flujo MQTT utilizando QoS de nivel 2

Tabla 2.5: Estructura de un paquete MQTT. Adaptado de [74]

Encabezado fijo	**Encabezado variable**	**Payload**
En todos los paquetes	En algunos paquetes	En algunos paquetes
Formato		
Byte 1: Tipo de paquete de control (7-4) || flags (3-0)	Byte 1: Identificador de paquete. Most Significant Byte	El largo depende del Tipo de paquete de control
Byte 2: Longitud Restante	Byte 2: Identificador de paque. Less Significant Byte	

Tabla 2.6: Arquitectura de capas de AMQP. *El protocolo de transporte opera sobre TCP

Parte 5	Capas de seguridad
Parte 4	Mensajería transaccional
Parte 3	Formato del mensaje
Parte 2	Protocolo de transporte (peer-to-peer*)
Parte 1	Tipo de sistema y codificación

2.9.3. DDS

DDS consiste en un modelo *Data-Centric Publish-Subscribe* (DCPS) para la comunicación e integración de aplicaciones distribuidas. Esto, debido a que hay muchas aplicaciones en tiempo real que necesitan publicar datos para que estén disponibles para otras aplicaciones remotas interesadas en estos. DDS se caracteriza por brindar calidad del servicio (QoS) asociada a las entidades, la cual debe ser compatible tanto con el que publica como con el suscriptor [75]. Los contratos de QoS facilitan el rendimiento y el control de los recursos que necesitan los sistemas embebidos en tiempo real, manteniendo la modularidad, escalabilidad y robustez propias del modelo *Publish/Subscribe* [76].

Este estándar de DDS se encarga de unificar las mejores prácticas que se han desplegado de forma correcta para la distribución de datos en tiempo real [76].

DDS forma una especie de espacio global de datos con objetos de datos, a los que tienen acceso nodos distribuidos, por medio de operaciones simples de lectura y escritura, es por esto que se usa el modelo *Publish-Subscribe* para datos distribuidos.

2.9.4. AMQP

Advanced Message Queuing Protocol, AMQP, es un protocolo de Internet abierto para mensajería en los negocios. Entre sus características están las colas confiables, mensajes basados en publish/subscribe, enrutamiento flexible, transacciones, seguridad y entrega confiable garantizada por TCP en la capa de transporte. Este protocolo nace en el mundo empresarial, pensando en grandes compañías que dependen de la mensajería para integrar aplicaciones y mover datos en la organización [11].

AMQP consta de una arquitectura por capas que se describen en la Tabla 2.6.

Una red AMQP está compuesta por nodos, llamados entidades, conectados por medio de enlaces unidireccionales. Dichas entidades son las encargadas del almacenamiento y entrega segura de los mensajes, tales como las colas, y se encuentran alojadas en contenedores.

Antes del intercambio de paquetes en una conexión, cada nodo debe enviar un encabezado de protocolo, indicando la versión del protocolo que va a usar. El nodo que actuará como cliente será el primero en enviarlo al establecer la conexión y pos-

Tabla 2.7: Formato de encabezado del paquete. Adaptador de [43]

1er byte	2do byte	3er byte	4to byte	5to byte	6to byte	7mo byte	8vo byte
Nombre protocolo				ID			
A	M	Q	P	%d0	*Major*	*Minor*	*Revision*

Tabla 2.8: Estructura del frame en AMQP. Adaptado de [43] *DOFF = Data Offset

Byte 0	SIZE	FRAME HEADER
Byte 1		
Byte 2		
Byte 3		
Byte 4	DOFF	
Byte 5	TYPE	
Byte 6	CANAL	
Byte 7		
Byte 8		EXTENDED HEADER
Byte 4*DOFF-1		
Byte 4*DOFF	PERFORMATIVE	FRAME BODY
Byte SIZE-1	PAYLOAD	

teriormente lo hará el servidor. El formato para ello es el que se muestra en la Tabla 2.7.

Las tramas se componen por tres partes, un encabezado fijo, de 8 Bytes, un encabezado extendido variable y un cuerpo variable como se describe en la Tabla 2.8.

2.9.5. XMPP

Extensible Messaging and Presence Protocol, XMPP, es un protocolo de aplicación abierto desarrollado por la comunidad de Jabber y plasmado en la RFC 6120 [83], que utiliza XML para la comunicación y funciona sobre TCP/IP. Se usa principalmente para mensajería instantánea, chats múltiples, llamadas de voz, video llamadas, multiconferencia y distribución de contenido. Entre las características del protocolo están el cifrado del canal, la autenticación, el control de errores, la disponibilidad de la red ("presencia") y las interacciones petición respuesta [9].

XMPP trabaja con una arquitectura distribuida cliente-servidor, que provee una tecnología asíncrona y punto a punto para el intercambio de datos, es decir, hay un proceso por el cual un cliente se conecta a un servidor para poder intercambiar mensajes, el cual consiste en primero, determinar la dirección IP y el puerto al cual conectarse, luego, se abre una conexión TCP y sobre esta a su vez un flujo XML. En lo posible después, por razones de seguridad, se debe hacer una negociación TLS para encriptar el canal y proceder a realizar una autenticación simple. Ahora, se pasa a la fase de intercambio de mensajes en la red. Finalmente, se cierra el flujo XML y la conexión TCP.

XMPP ha considerado una comunicación segura para aquellos casos en los que el flujo sea entre dos servidores o entre un cliente y un servidor, lo cual equivale a una debilidad cuando se comuniquen dos clientes.

Para la seguridad es muy importante el orden de las capas en las que deben apilarse los protocolos, esto es XMPP la superior, SASL es la que está inmediatamente inferior, después está TLS y la última es TCP.

2.9.6. HTTP/1.1

Hypertext Transfer Protocol (HTTP) es un protocolo web de capa de aplicación que funciona con arquitectura cliente/servidor. La versión 1.1 de este protocolo fue publicada en 1999 y está descrita en la RFC 2616 [47].

HTTP es un protocolo sencillo que funciona sobre TCP y se basa en solicitudes y respuestas. Para esto, un cliente inicia una conexión TCP con el servidor y envía un mensaje con la información de la solicitud, el servidor responde con un mensaje que contiene el estado de la operación y su resultado. Las características más relevantes de HTTP/1.1 son:

- Toda la comunicación entre los clientes y servidor se realiza basada en los caracteres US-ASCII de 7 bits.
- Permite la transferencia de multimedia.
- Existen una serie de verbos que representan las solicitudes que pueden hacer los clientes a los servidores. Los más conocidos son GET, POST, DELETE, PUT, HEAD, OPTIONS y CONNECT.
- Cada operación HTTP implica una conexión TCP con el servidor, que es finalizada al obtener la respuesta.
- No mantiene estado, cada operación es independiente de las anteriores.
- Cada objeto o recurso sobre el que se aplican las operaciones está identificado con una *Uniform Resource Locator* (URL).
- HTTP/1.1 puede funcionar opcionalmente con seguridad. En la RFC2818 [80] se define el uso de HTTP en modo seguro haciendo uso de TLS.

Cuando el cliente realiza una solicitud al servidor se ejecutan los siguientes pasos:

1. El usuario accede a una URL.

2. El cliente web decodifica la URL, identificando el protocolo de acceso, dirección de *Domain Name Server* (DNS), IP y el recurso solicitado al servidor.

3. Se abre una conexión TCP entre el cliente y el servidor.

4. El cliente envía la petición utilizando uno de los comandos definidos y la dirección del objeto requerido.

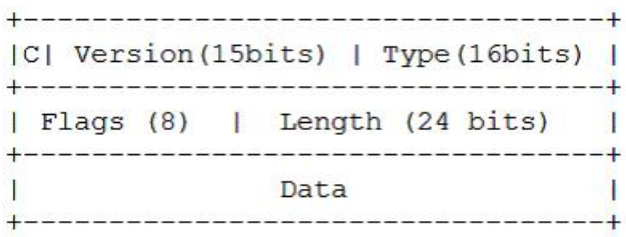


Figura 2.7: Trama de control en SPDY. Tomado de [8]

5. El servidor responde al cliente, indicando el estado de la operación y el recurso o respuesta que corresponda.

6. Se cierra la conexión TCP.

2.9.7. SPDY

Con el objetivo de buscar reducciones en la latencia de las páginas web, en 2009 Google publicó el primer *draft* de SPDY, un protocolo de nivel de sesión que busca agilizar la carga de contenidos de páginas web en comparación con la pila tradicional HTTP/TCP. Las pruebas iniciales de laboratorio desarrolladas por la compañía arrojaron una reducción de hasta el 64 % en la carga de páginas web [92]. SPDY se diseñó para ser compatible con las aplicaciones web existentes, para esto se debían conservar la integridad y las funcionalidades de HTTP en la capa de aplicación. Entre las funcionalidades que introduce SPDY están:

- *Stream* multiplexados: SPDY permite *streams* concurrentes ilimitados sobre una conexión TCP. Esto reduce significativamente el número de conexiones TCP y hace más eficiente el intercambio de carga útil en relación con los paquetes transmitidos.
- Priorización de peticiones: Los *streams* multiplexados resuelven el problema de la serialización, pero introduce otro: si se tiene ancho de banda limitado ¿Cuáles peticiones atender primero? En SPDY el cliente puede hacer peticiones ilimitadas al servidor, y a cada una asignarle una prioridad, solucionando así este problema.
- Compresión del *header* de HTTP: SPDY comprime las cabeceras HTTP, resultando así menos paquetes y menos Bytes transmitidos.
- *Server push*: El servidor identifica contenidos asociados a una petición hecha por el cliente y le anuncia que va a enviar estos contenidos antes de que el cliente los solicite.

Una vez se establece la conexión, SPDY utiliza dos tipos de tramas, una de control y otra de datos [8].

En la Figura 2.7 se muestra la estructura de la trama de control, cuyos campos son los siguientes:

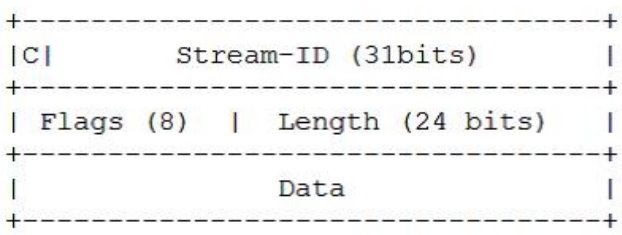


Figura 2.8: Trama de datos en SPDY. Tomado de [8]

- C: Un bit que indica que la trama es de control.
- *Version*: El número de la versión del protocolo. En 15 bits.
- *Type*: Tipo de la trama de control. Estas pueden ser:
  - SYN_STREAM: Permite al emisor crear asíncronamente un *stream* nuevo.
  - SYN_REPLY: Indica que se acepta la creación del *stream*.
  - RST_STREAM: Permite la terminación anormal de un *stream*.
  - SETTINGS: Contiene un conjunto de valores que determinan cómo los *endpoints* se van a comunicar
  - PING: Permite medir el mínimo *Round Trip Time* (RTT) entre el emisor y receptor
  - GOAWAY: Indica al otro *endpoint* que deje de crear *streams* en la conexión.
  - HEADERS: Aumenta la trama con headers adicionales.
  - WINDOW_UPDATE: Es utilizada para hacer control de flujo en la capa de sesión.
- *Flags*: Banderas relacionadas a la trama.
- *Length*: 24 bits que anuncian el tamaño del segmento siguiente, *data*.
- *Data*: Los datos relacionados con el tipo de la trama.

En la Figura 2.8 se muestra la estructura de la trama de datos, cuyos campos son los siguientes:

- C: Un bit que en las tramas de datos siempre es 0.
- *Stream ID*: 31 bits que identifican el *stream* al que pertenece la trama.
- *Flags*: Banderas relacionadas a la trama. En la trama de datos la bandera válida es 0x01, que significa la finalización del *stream*.
- *Length*: 24 bits que anuncia el tamaño del segmento siguiente, *data*.
- *Data*: Un campo variable de carga útil.

SPDY sirvió de base para HTTP/2, plasmado y estandarizado en la RFC7540 [25]. En 2016 Google anunció que no dará más soporte a SPDY.

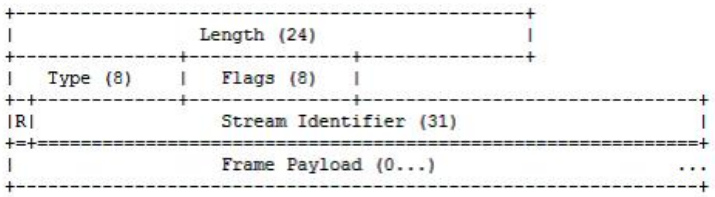


Figura 2.9: Trama de HTTP/2. Tomado de [25]

2.9.8. HTTP/2

Esta nueva versión de HTTP está plasmada en el RFC7540 [25] e introduce ajustes y modificaciones con el fin de tener un protocolo robusto y ágil, que se adapte de mejor manera a las aplicaciones de la actualidad.

Una de las características de las versiones anteriores de HTTP, que pueden verse como una desventaja, es el tamaño de los *headers* y que su información en muchos casos es repetitiva, esto genera tráfico innecesario y congestión debido a que la ventana TCP se llena rápidamente.

En HTTP/2 se mitigan estos "problemas", optimizando la semántica de los *headers*, que permiten la intercalación de peticiones y respuestas en una misma conexión. Se permite también la priorización de peticiones, además de añadir nuevas formas de interacción, como el *server push*, que permite al servidor enviar información anticipada al cliente, asumiendo que la va a necesitar [25].

El resultado es un protocolo más liviano sin perder robustez, debido a que requiere menos conexiones TCP y las cabeceras son más reducidas, por otro lado el usuario percibirá mayor agilidad en las aplicaciones.

A continuación se identifican las características más relevantes de HTTP/2 [1]:

- Trama de HTTP/2: Después de establecida la conexión, todas las tramas inician con 9 octetos seguidos del *payload*, de longitud variable, como se muestra en la Figura 2.9.
- Los primeros tres octetos *(Length)* indican la longitud del *payload*, por defecto este valor no será mayor a 2048 Bytes, a menos que se especifique otra longitud máxima. El octeto *Type*, determina el formato y la semántica de la trama. El octeto *Flags* consiste en 8 bits reservados para banderas booleanas específicas de cada tipo de trama. El bit reservado, R, no está definido aún. El identificador del *stream* está expresado por un entero de 31 bits, donde la 0x0 está reservada para las tramas asociadas al establecimiento de la conexión. Por último está el *payload*.
- Tramas binarias: En HTTP/1.x las tramas están en texto plano, esto hace que inspeccionarlas sea relativamente sencillo y se incrementa el tamaño de las mismas. En HTTP/2, las tramas son binarias, de esta manera se reduce su tamaño.
- Compresión del *header*: En HTTP/1.x el campo del header contiene metadatos

en texto sin formato, esto puede añadir en ocasiones más de un kilobyte a la trama. En HTTP/2 los metadatos del *header* son comprimidos utilizando el algoritmo de Huffman para reducir la carga y mejorar el rendimiento.

- Multiplexión: Una de las desventajas de HTTP/1.x es que si el cliente requiere hacer múltiples peticiones paralelas al mismo servidor, se deben crear varias conexiones TCP. Esto incrementa el tráfico y los tiempos de respuesta significativamente, debido al *handshake* que debe hacer cada conexión. En HTTP/2 se pueden enviar tramas de diferentes *streams* por una misma conexión TCP.
- Priorización de *streams*: Si un cliente tiene varios *streams* con el mismo servidor, el cliente puede asignar una prioridad para un nuevo *stream*, de esta manera el cliente puede expresar cómo va a preferir que el servidor maneje los recursos.
- Control de flujo: Adicional al control de flujo de TCP, HTTP/2 define una ventana para control de flujo en la capa de aplicación. Esto se hace con el fin de proteger a los dispositivos finales que operan bajo recursos restringidos, asegurando que los *streams* de una misma conexión no interfieran de manera destructiva sobre los demás. El control de flujo es usado para cada stream de manera individual o también para la conexión como un todo.
- *Server push*: El servidor envía anticipadamente respuestas a un cliente en asociación con una petición previa iniciada por el cliente. Esto sirve cuando el servidor conoce que el cliente necesitará esas respuestas porque están asociadas a la solicitud original. Con esta funcionalidad se ahorra tiempo y tráfico.

Capítulo 3

Revisión del estado del arte

En el capítulo anterior se mostraron algunas tecnologías y protocolos que se han creado para IoT, o que se utilizan en este campo. En el presente capítulo se muestran trabajos recientes relacionados con el diseño, comparación y evaluación de protocolos de capa de aplicación para IoT. Dichos trabajos son agrupados en tres grupos: evaluación en ambientes de IoT, comparación vía experimentos y simulaciones, y diseño de nuevos protocolos para escenarios restringidos.

3.1. Evaluación de protocolos de aplicación en ambientes IoT

En [93] se evalúa el desempeño de CoAP teniendo en cuenta el número de saltos entre nodos, para esto hacen una simulación utilizando Contiki, donde sensores de humedad y temperatura son requeridos periódicamente por un cliente CoAP. Como resultado se evidencia lo ligero que es CoAP y la ventaja de usar UDP, se resalta, también, el ahorro de energía y que mejora los tiempos de respuesta de los dispositivos restringidos. Se muestra el efecto en las variables evaluadas de aumentar el número de saltos en la transmisión.

Uno de los argumentos por los que se busca utilizar HTTP/2 en IoT son los riesgos de seguridad que sugiere la creación de nuevas pilas de protocolos, como en el caso de CoAP. Referente a esto, en [79] se analizan los aspectos de seguridad de toda la pila de protocolos utilizada por CoAP en un ambiente restringido: IEEE 802.15.4, 6LoWPAN, RPL y CoAP. Las discusiones de seguridad en CoAP se han centrado principalmente en el uso de DTLS, el pesado costo de computación y el *handshake* que causa la fragmentación de los mensajes. Como solución se ha planteado comprimir DTLS. Además, DTLS tiene la desventaja de no soportar mensajes *multicast* en grupos de comunicaciones. El documento concluye que hay áreas en las que DTLS se queda corto, convirtiéndose en una amenaza de seguridad.

Tabla 3.1: Resultados de evaluación de seguridad de IPSec y DTLS utilizando X.805. Adaptado de [56]. *PKI = Public Key Infrastructure

Aspectos de Seguridad	IPSec	DTLS
Control de Acceso	No	No
Autenticación	Si	Parcialmente - Sólo en el Servidor
No Repudio	Sí/No; depende del método de autenticación. PKI* no es soportado por dispositivos restringidos.	Sí/No; depende del método de autenticación. PKI* no es soportado por dispositivos restringidos.
Confidencialidad	Si	Si
Seguridad de la comunicación	Si	Si
Integridad	Si	Si
Disponibilidad	Mitigación - No hay protección completa	Sí - Cookie sin estado
Privacidad	No	No

DTLS es la propuesta para hacer CoAP seguro. Además, en el *Working Group* de CoRE ha habido propuestas para habilitar seguridad en capas inferiores, como por ejemplo haciendo uso de IPSec [28]. En [15] usan el estándar de seguridad X.805 [10] para analizar los aspectos de seguridad de CoAP. El estándar X.805 define tres capas de seguridad: aplicaciones, servicios e infraestructura; tres planos de seguridad: usuario final, control y gestión; además de ocho dimensiones de seguridad: control de acceso, autenticación, no repudio, confidencialidad de datos, seguridad en la comunicación, integridad de los datos, disponibilidad y privacidad. Las conclusiones a las que llegan es que IPSec y DTLS fallan en algunos requerimientos de seguridad. Adicionalmente, hay un problema de usabilidad cuando se utiliza DTLS y IPSec en dispositivos restringidos. El documento argumenta la necesidad de un mecanismo de seguridad ligero e integrado para una versión segura de CoAP. Los resultados más relevantes se muestran en la Tabla 3.1.

En [38] evalúan CoAP y 6LoWPAN en un escenario de hogar inteligente. Se evalúan dos escenarios utilizando medidores de energía, uno de ellos infectado con código malicioso. En un escenario, un medidor hace espionaje de la red, y en el otro introduce solicitudes falsas. Los resultados muestran que la mayoría de aplicaciones pueden beneficiarse del cifrado AES contra ataques de espionaje. Para evitar ataques de *Denial-of-Service* (DoS) se puede delegar la tarea de descifrado a un hardware especializado.

En [89] diseñan, implementan y evalúan un *proxy* HTTP-CoAP, para pasar de HTTP sobre IPv4 a CoAP sobre una red 6LoWPAN. El software de este *proxy* se compone por un módulo para un servidor web Apache, otro encargado del mapeo de HTTP a CoAP y un último que es el módulo CoAP. El *proxy* cuenta con un mecanismo de *caching* que se comporta como si fuera directamente el servidor CoAP, con el fin de mejorar el desempeño del sistema. Para la medición del rendimiento se utilizó la herramienta Apache JMeter que permite simular clientes múltiples concurrentes. Como resultado se obtuvo que la latencia aumenta a medida que aumentan los clientes; el throughput indicó que el proxy es capaz de soportar 25 solicitudes GET por segundo y se comprobó la efectividad del mecanismo de caché, ya que logró reducir la latencia.

3.2. Comparación de protocolos de aplicación vía simulaciones y experimentos

En [35] se busca evaluar el desempeño de diferentes protocolos de aplicación para escenarios restringidos: MQTT, CoAP, *Data Distribution System*, DDS, y un protocolo propio basado en UDP. Las pruebas se hicieron con una serie de sensores médicos que envían información a un servidor. Como resultados relevantes referentes al desempeño de los protocolos, se tiene que los protocolos basados en TCP no experimentan pérdidas de paquetes en condiciones de red difíciles, con tasas de paquetes perdidos de hasta 25 % y latencia de 400ms. Sin embargo, DDS supera significativamente a MQTT en lo que respecta a la latencia de telemetría en malas condiciones de la red; así DDS tenga un consumo mayor de ancho de banda que MQTT, éste es mejor candidato para redes que requieren menos pérdidas.

En cuanto a la implementación de ambientes donde conviven CoAP y HTTP, previamente se han realizado trabajos comparativos. En [60] se hace una implementación donde interactúan un servidor CoAP, un cliente HTTP, un cliente CoAP y un servidor HTTP. El servidor CoAP utilizado es un Erbium [2] en una tarjeta TelosB, que tiene un microcontrolador que opera a 8 MHz, tiene 10 kB de RAM y 48 kB de memoria flash. Como cliente HTTP se utiliza un computador con Firefox, como cliente CoAP se usa el complemento Copper de Firefox y como servidor HTTP se utiliza un Apache Tomcat. Para la comunicación HTTP-CoAP se necesita un *proxy*, en este caso el utilizado es CoAP-Proxy. Como resultados relevantes de este estudio, se destaca en primer lugar, que el protocolo y las componentes desplegados trabajan correctamente. Por otro lado, se obtienen datos del *caching* que hace el *proxy*, el cual reduce las solicitudes al servidor entre un 66 % a un 95 % según el número de usuarios, agilizando de esta manera la comunicación.

En [91] se hace una comparación de desempeño entre CoAP y MQTT. En este caso utilizan las implementaciones libcoap y Mosquitto, respectivamente. Los resultados muestran que el desempeño de MQTT y CoAP dependen de las condiciones de la red. Para bajos valores de pérdidas de paquetes, MQTT muestra menor *delay*, sin embargo, si la pérdida de paquetes aumenta, CoAP muestra mejor desempeño en términos de *delay*. Esto debido a que las cabeceras de TCP para la retransmisión de paquetes son más grandes que las de UDP en las transmisiones CoAP. En el estudio también se identifica que el desempeño de los protocolos depende del tamaño de los paquetes. Por ejemplo, cuando el tamaño de este es pequeño y la tasa de pérdidas es de 25 % o menos, CoAP genera menos tráfico extra que MQTT para asegurar la transmisión confiable.

Por otro lado, en [37] llegan a la conclusión de que MQTT tiene más *throughput* y menor latencia que CoAP en un escenario con alto tráfico y alta pérdida de paquetes. En lo que coinciden es que la elección de uno u otro protocolo dependerán de las condiciones de la red. Recomiendan usar MQTT en escenarios con bajo *throughput* y alto *delay*.

En [97] se hace una comparación de desempeño entre HTTP y MQTT en cuanto al tamaños de cabeceras. En términos generales, concluyen que MQTT tiene mejor desempeño que HTTP, sin embargo, cuando los *topics* de MQTT sobrepasan los 680 Bytes supera las cabeceras de HTTP. Para mejorar lo anterior proponen la compresión de los topics de MQTT. Cabe resaltar que en MQTT un *topic* es un string que es usado por el *broker* para clasificar mensajes, que están organizados en jerarquías de manera similar a una carpeta.

En [49] plantean el acceso a un servidor CoAP, desde un cliente CoAP que llaman SCoAP. Este último es la combinación de una implementación de CoAP en Javascript llamada JSCoAP y un *proxy* CoAP especial que usa un web *socket* en HTML5 llamado WSCoAP, el cual permite una comunicación CoAP/CoAP entre el cliente y el servidor. Por otro lado, se usa una comunicación entre cliente y servidor, donde hay un *proxy* HTTP/CoAP en medio. Al comparar los resultados, se encontró que el *proxy* WSCoAP tiene una ventaja significativa sobre el *proxy* HTTP/CoAP en cuanto a tiempos de respuesta.

En cuanto a las evaluaciones de desempeño de HTTP/2, en [39] comparan el desempeño de HTTP/1.1 contra la nueva versión del protocolo. Aclarando que es un trabajo preliminar y que aún quedan funcionalidades por probar, como era de esperarse, HTTP/2 obtuvo mejor desempeño que su versión anterior. Se evaluaron los tiempos de respuesta en páginas web, donde HTTP/2, gracias a la multiplexión y compresión, fue más rápido. Sin embargo, también se concluye que HTTP/2 es vulnerable a la pérdida de paquetes, que combinado con el control de congestión de TCP, terminan por afectar el desempeño, lo que sugiere que una mejora puede consistir en tener una capa de transporte con un protocolo optimizado para redes con pérdidas de paquetes.

En [61] se preguntan si HTTP/2 realmente ayuda en el desempeño web de teléfonos inteligentes. Se llega a la conclusión de que las pérdidas de paquetes afectan de manera significativa a HTTP/2, pero es útil cuando las páginas web tienen muchos objetos pequeños y hay una alta latencia, puesto que se aprovecha la multiplexión. Esto indica que la conveniencia de HTTP/2 tiene cierta dependencia de las condiciones del canal y del tipo de tráfico con el que se va a trabajar.

En [70] se realizan cuatro experimentos que examinan el uso de SPDY y HTTP/2 en reducir la latencia de la web y ser adecuados para la WoT. Los resultados indican que la implementación de estos protocolos en los servidores mejora el rendimiento y reducen la latencia. Sin embargo, se concluye que para usar estos protocolos en la WoT se deben mejorar o reemplazar por protocolos más ligeros como CoAP, MQTT y AMQP.

En [36] hacen una evaluación de desempeño energético de HTTP/2 respecto a HTTP/1.1 utilizando *Green Miner* [53] como metodología de evaluación. Los resultados muestran que HTTP/2 ahorra energía cuando los *Round Trip Time* (RTT) son superiores a 30ms y cuando, además, se usa TLS. También se logró determinar que la implementación de Mozilla Firefox Nightly para HTTP/2 consume menos energía que la implementación de HTTP/1.1 haciendo las mismas tareas. Entre las razones está que HTTP/1.1 se vuelve costoso debido al alto número de conexiones TCP que

se deben hacer. Por último, se evidencia que adoptar HTTPS en clientes móviles es costoso energéticamente.

Mas recientemente, en [82] se evalúan diferentes parámetros de HTTP/2 usando IoT-Lab [4] y Rapsberry Pi con NGHTTP2. Además del tamaño de la ventana, se evalúan parámetros tales como: el tamaño máximo del frame, tamaño de la tabla y lista de HEADERS , server push y número de streams concurrentes, midiendo la ocupación de CPU, memoria y potencia utilizada. A pesar del amplio número de pruebas, las conclusiones no llegan a sugerir cambios puntuales en los parámetros o en el funcionamiento del protocolo.

En [44] evalúan comparativamente HTTP/2 frente HTTP/1.1 utilizando Raspberry Pi como clientes y un servidor Ubuntu como servidor, conectados a través de una red WiFi. En este trabajo se utilizan métricas como throughput y tiempos de respuesta promedio. Después de las pruebas se concluye que HTTP/2 y HTTP/1.1 tienen desempeños similares para peticiones simples, sin embargo, para peticiones simultaneas, HTTP/2 obtiene mejores resultados. En este trabajo se hace una evaluación de desempeño, sin embargo, no se propone ningún cambio al protocolo.

3.3. Diseño de protocolos

En secciones anteriores se ha tratado el tema de diseño de protocolos en cuanto a principios y metodologías. En esta sección se muestran algunos trabajos en los que se ha trabajado este tema de manera aplicada.

En [21] señalan que algunos protocolos que son utilizados en redes de sensores inalámbricos han sido creados con consideraciones para redes cableadas. Este trabajo recoge y analiza requerimientos y restricciones que pueden afectar el diseño del enlace de datos y las distintas capas de red. Por ejemplo, si los nodo-sensores están ante la presencia de vibraciones, movimiento, fenómenos atmosféricos, térmicos, energéticos, etc.

En [65] diseñan un protocolo privado P2P de aplicación que busca cumplir con los requerimientos propios de este tipo de comunicación funcionando sobre TCP. Describen la estructura de la cabecera del paquete y los comandos de petición y respuesta que utiliza. Sin embargo, este trabajo no hace referencia a alguna metodología formal de diseño de protocolos, tampoco describen la interacción con TCP y al ser un protocolo privado no se asegura la interoperabilidad global.

En [66] proponen una nueva noción de diseño de protocolos de aplicación para *Wireless Sensor Network* (WSN), ya que los enfoques tradicionales tienden a desarrollar el protocolo primero y después usarlo en las diferentes topologías. Este trabajo propone considerar la topología lógica de la WSN antes de diseñar el protocolo, de esta manera el protocolo tiene mejor desempeño. Identifican cuatro escenarios de uso particular de las WSN: el primero donde el usuario necesita recolectar información

Tabla 3.2: A. Resumen de trabajos de evaluación y comparación de protocolos

Estudio	Protocolo(s)	Qué compara / Qué mide	Condiciones	Resultado
[93]	MQTT, CoAP, DDS y un protocolo propio	Packet loss	Pruebas realizadas en sensores médicos que envían información a un servidor. Variación en la tasa de pérdida de paquetes	Protocolos, basados en TCP son más resistentes a pérdidas de paquetes. DDS resulta más adecuado que MQTT en telemetría.
[79]	CoAP	*Throughput, delay y packet loss* según el número de saltos y la tasa de generación de paquetes.	Simulación en Cooja (Contiki), cliente utiliza Californium. Se utilizan sensores de humedad y temperatura que son requeridos periódicamente por un cliente CoAP.	Se evidencia lo ligero que es CoAP y la ventaja de usar UDP. Ahorra energía y mejora los tiempos de respuesta de las motes. Se muestra el efecto en las variables evaluadas de aumentar el número de saltos en la transmisión.
[27][10]	CoAP	Seguridad	Se analizan aspectos de seguridad de CoAP. En [57] se usa el estándar X.805 para la evaluación de seguridad.	Hay áreas de seguridad en las que DTLS y IPSec se quedan cortos. Convirtiéndose en una amenaza de seguridad. Hay problemas de usabilidad con DTLS y IPSec en dispositivos restringidos. Se requiere una versión integrada y ligera para una versión segura de CoAP.
[89]	*Proxy* HTTP-CoAP	Latencia y *throughput*	HTTP sobre IPv4 a CoAP sobre 6LoWPAN. El *proxy* usa *caching*. Utilizan Apache JMeter para la medición de rendimiento.	La latencia aumenta a medida que aumentan los clientes; el *throughput* indicó que el *proxy* es capaz de soportar 25 solicitudes GET por segundo y se comprobó la efectividad del mecanismo de caché, ya que logró reducir la latencia
[35]	CoAP, 6LoWPAN	Aspectos de seguridad	Se evalúan aspectos de seguridad respecto a un posible ataque de espionaje e introducción de código malicioso. Escenario utilizado: Medidores eléctricos	El cifrado AES puede ser beneficioso ante ataques de espionaje. Para evitar ataques DoS se puede delegar la tarea de descifrado a un hardware especializado.
[60]	HTTP, CoAP	Proxy HTTP-CoAP	Cliente CoAP: Copper, Servidor CoAP: Erbium Proxy: JCoAP, Cliente HTTP: Firefox Servidor HTTP: Apache Tomcat	Las implementaciones funcionan. El *Caching* reduce entre un 65 % y 95 % las solicitudes al servidor.
[91]	CoAP	MQTT, packet loss y tamaño de paquetes	CoAP: libcoap, MQTT: Mosquitto, Variación del packet loss y el tamaño de los paquetes.	Para bajos valores de packet loss, MQTT muestra menor *delay*, Si el *packet loss* aumenta CoAP tiene mejor desempeño. En el estudio también se identifica que el desempeño de los protocolos depende del tamaño de los paquetes.
[82]	HTTP/2	Diferentes parámetros del protocolo	Variación de cada parámetro medido. Pruebas en distintos dispositivos.	Se hace una evaluación amplia. Resultados mixtos, no concluye cambios puntuales en el funcionamiento del protocolo.
[44]	HTTP/2	HTTP/1.1, miden *throughput* y tiempos de respuesta promedio	Usan raspberry como cliente, un servidor Ubuntu y WiFi	En peticiones sencillas HTTP/1.1 y HTTP/2 tienen desempeños similares, mientras que para peticiones simultáneas, HTTP/2 logra mejores resultados

Tabla 3.3: B. Resumen de trabajos de evaluación y comparación de protocolos

Estudio	**Protocolo(s)**	**Qué compara / Qué mide**	**Condiciones**	**Resultado**
[37]	CoAP	MQTT, *throughput* y delay	Variación del tráfico del canal y pérdida de paquetes.	La elección del protocolo más adecuado depende de las condiciones del canal. Es recomendable usar MQTT en escenarios con bajo *throughput* y alto *delay*
[97]	HTTP	MQTT, tamaño de las ventanas	Se compara respecto al tamaño de cabeceras.	En términos generales concluyen que MQTT tiene mejor desempeño que HTTP, sin embargo cuando los *topics* de MQTT superan los 680 Bytes supera las cabeceras de HTTP. Para mejorar esto proponen la compresión de los *topics*.
[49]	CoAP y HTTP	Comunicación CoAP/CoAP vs HTTP/CoAP utilizando proxy	Un *proxy* CoAP/CoAP llamado JSCoAP se compara con un *proxy* HTTP/CoAP.	La comunicación CoAP/CoAP es mucho más rápida que la HTTP/-CoAP.
[39][61]	HTTP/2	HTTP/1.1, tiempos de respuesta	Comparación variando el número de peticiones que se hacen al servidor web. En [61] la evaluación es sobre *smartphones*.	HTTP/2 es más rápido que HTTP/1 en términos generales. HTTP/2 es vulnerable a la pérdida de paquetes. HTTP/2 mejora significativamente el desempeño cuando las páginas web tienen muchos objetos, ya que aprovecha la multiplexión.
[70]	HTTP/2 y SPDY	HTTP/1.1	No se consideró el contenido de las páginas web ni la ubicación del servidor.	Desde el punto de vista del servidor hay una mejoría en el desempeño, sin embargo, no es tan notoria desde el cliente.,HTTP/2 necesitará ajustes para funcionar en WoT, o ser definitivamente reemplazado por protocolos más livianos como MQTT, CoAP y AMQP.
[36]	HTTP/2	HTTP/1, consumo energético	Comparación respecto a consumo energético, utilizando Green Miner.	HTTP/2 ahorra energía respecto a HTTP/1.1, especialmente cuando los RTT son altos.

de los nodo-sensores; otro donde el usuario necesita enviar información a los nodos; el tercero donde un nodo o grupo de nodos necesita comunicarse con otro y el último donde el usuario necesita que los sensores ejecuten una acción. Señalan que la mayor parte de una comunicación en una WSN está relacionada con comunicación entre sensores. Esto es interesante desde el punto de vista de una adaptación de HTTP/2, debido a que su modelo cliente/servidor está diseñado para otro tipo de escenarios.

En [23] explican un protocolo de comunicación, que según el estado del canal transmite determinados paquetes con el objetivo de mantener sensores inalámbricos dormidos el mayor tiempo posible. Así, si el estado del canal es bueno, los paquetes con información son entregados sin errores, por lo que se eliminan los paquetes de ACK; pero si el estado es malo, se utilizan retransmisiones, ya que hay alta probabilidad de que los paquetes se pierdan. Para las pruebas, se han tomado como premisas que los sensores pueden predecir con exactitud el siguiente estado del canal y que el tiempo de duración de una trasmisión es igual al de recepción. Además, es necesario conocer para cada estado del canal la tasa de error de paquetes (PER). En este caso, se espera que un canal en condiciones buenas tenga una PER menor a 20 % y uno en condiciones malas tenga una PER mayor a 80 %. Los resultados arrojaron que cuando las condiciones del canal son buenas, la caída del voltaje de la batería es menor que cuando las condiciones son malas y cuando las condiciones son desconocidas la caída varía entre el comportamiento que se tiene cuando hay buenas y malas condiciones. Con esto, surge la necesidad de tener un canal en buenas condiciones, puesto que así se ahorran la energía que se usa esperando por un ACK y cuando las condiciones no son buenas se puede usar para el envío de dichos paquetes.

En [99] se explora la conversión de CoAP y HTTP/2, especialmente en un esquema publish-subscribe el cual es soportado por CoAP, pero no por HTTP/2. Se construye un framework que mediante sondeo verifica si hay recursos para el cliente HTTP/2 en el *broker*. Posteriormente se hacen evaluaciones de desempeño en términos de *delays* y uso de CPU. Este trabajo explora la extensión de HTTP/2 como lo es su uso en un esquema *Publish-Subscribe*.

Capítulo 4

Estudio de parámetros de HTTP/2 en IoT

HTTP/2 es un protocolo configurable que permite variar los valores de sus parámetros. En el marco del desarrollo de la presente tesis, se hicieron experimentos cuyo objetivo era estudiar el funcionamiento de HTTP/2 frente a un protocolo de referencia y el funcionamiento de HTTP/2 al variar alguno de sus parámetros. En una primera serie de experimentos se evaluó el desempeño de HTTP/2 frente a CoAP. En un segundo trabajo, se realizó una evaluación de desempeño de HTTP/2 ante la variación del tamaño de la ventana de control de tráfico.

4.1. Diseño de los experimentos

En un primer experimento, se realizó una evaluación de desempeño respecto a los tiempos de respuesta y paquetes perdidos de HTTP/2 y CoAP, ante la variación de las condiciones de una red WiFi, utilizando una página web. Para ello, se partió desde un escenario con una red con buenas condiciones de recepción de señal y sin congestión, hasta una con problemas de recepción y con congestión.

Los escenarios implementados en el experimento se muestran en la Figura 4.1. En el primer escenario se prueba el desempeño de CoAP, utilizando un cliente CoAP que se conecta a un proxy CoAP/HTTP/1.1 a través de una red WiFi, y éste a su vez se conecta a un servidor HTTP/1.1. En el segundo escenario un cliente HTTP/2 se conecta con un servidor HTTP/2 a través de una red WiFi. En la Tabla 4.1 se muestra un resumen de las configuraciones utilizadas.

Los experimentos consisten en medir tiempos de respuesta y paquetes perdidos para peticiones GET y POST, mientras se varía la potencia de la señal WiFi y el nivel de congestión, usando equipos adicionales que generan tráfico en la red inalámbrica, lo cual afecta la calidad de recepción. En [62] se presenta un resumen de estas pruebas.

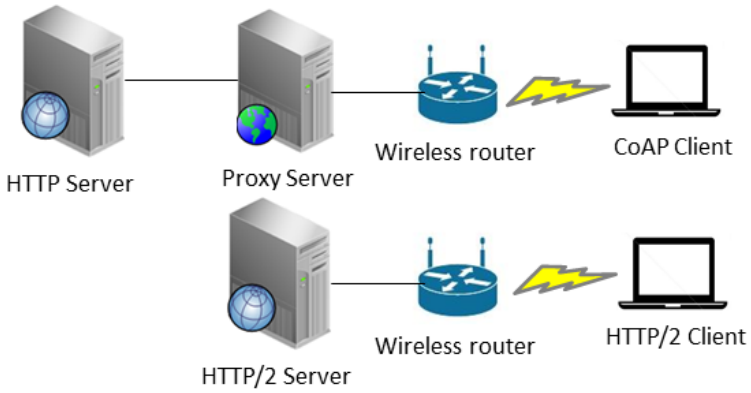


Figura 4.1: Escenarios implementados en las pruebas

Tabla 4.1: Resumen de configuración de experimentos CoAP vs HTTP/2

Componente	**Descripción**
Cliente HTTP/2	Mozilla Firefox 47.0
Cliente CoAP	Copper 0.18.4.1
Proxy CoAP/HTTP/1.1	Crosscoap
Servidor HTTP	Apache 2.4.20
Red WiFi	IEEE 802.11n
Generador de tráfico	Iperf

En una segunda serie de experimentos, se realizó una evaluación de desempeño de HTTP/2 ante la variación del tamaño de la ventana de control de flujo sobre una Raspberry Pi. En este caso, se definieron como métricas los tiempos de respuesta, tiempo de CPU, el trabajo realizado y el porcentaje de CPU utilizado. Para el experimento, se implementó un cliente HTTP/2 que le hacía peticiones repetitivas al servidor mientras las métricas eran medidas. La conectividad entre las Raspberry Pi se hizo mediante FastEthernet con el fin de tener una conexión estable, donde se evidenciara mejor el desempeño del protocolo.

La medición del trabajo realizado por la Raspberry Pi se hizo utilizando un Arduino, en conjunto con un sensor de corriente que se conectaba al cable de alimentación de la Raspberry Pi. En la Tabla 4.2 se muestra un resumen del montaje de este experimento. Un resumen de este trabajo se presenta en [63].

4.2. Discusión de resultados

Los resultados consignados en las Tablas 4.3 y 4.4 muestran que los tiempos de respuesta dependen de las condiciones de la red. Si la red es confiable, CoAP tiene mejor desempeño; por el contrario, si las condiciones de la red son malas, HTTP/2 se puede comportar mejor. Con unas buenas condiciones de red, CoAP es más ágil debido a que es un protocolo sencillo, con cabeceras simples y funciona sobre UDP. Por otro lado, HTTP/2 al ser un protocolo más robusto y sobre todo, al funcionar

Tabla 4.2: Resumen de configuración de experimentos en Raspberry Pi

Componente	**Descripción**
Cliente HTTP/2	nghttp
Servidor HTTP/2	nghttpd
Conexión cliente/servidor	Fast ethernet
Sensor de corriente	ACS712 con Arduino Mega
Tamaño de ventana inicial	256 Bytes a 65536 Bytes
Número de repeticiones	68
Índice de confianza	90 %
Métricas	Uso de CPU, tiempos de respuesta y trabajo

Tabla 4.3: Resultados ante petición GET

Potencia recibida	**Nivel de congestión**	**HTTP/2 (s) [% perdido]**	**CoAP (s) [%perdido]**
-27 dBm	Sin congestión	0,21039 [0 %]	0,00956 [0 %]
	Congestión x1	0,28495 [0 %]	0,04969 [0 %]
	Congestión x2	0,26536 [0 %]	0,06115 [0 %]
	Congestión x3	0,31693 [0 %]	0,09064 [0 %]
-85 dBm	Sin congestión	1,06149 [7 %]	2,03934 [0 %]
	Congestión x3	2,85344 [24 %]	6,44652 [16 %]

sobre TCP, tiene tiempos de respuesta más elevados. El aumento de los tiempos de respuesta de CoAP, cuando la red tiene pérdidas, se da debido a que los tiempos para hacer retransmisión son elevados; sin embargo, esto depende de la implementación y podría ser mejorado si se reducen los tiempos.

Tabla 4.4: Resultados ante petición POST

Calidad de la señal	**Nivel de congestión**	**HTTP/2 (s)**	**CoAP (s)**
-27 dBm	Sin congestión	0,07112	0,0168
	Congestión x3	0,66474	0,1393
-85 dBm	Sin congestión	0,37146	0,3795
	Congestión x3	0,4567	2,1253

En la Figura 4.2 se muestra el trabajo realizado, el tiempo de respuesta y de CPU en el cliente. En ella se ve que cuando la ventana es muy pequeña (256 Bytes), el largo tiempo en el que el servidor recibe paquetes provoca mayor uso de la CPU, lo que se traduce en un mayor trabajo. A medida que el tamaño de la ventana aumenta, los tiempos de respuesta y de CPU disminuyen y a partir de 4096 Bytes como tamaño de ventana, el trabajo requerido se estabiliza alrededor de 120 mJ.

En la figura 4.3 se muestra el porcentaje de uso de CPU tanto en el cliente como en el servidor, ante la variación de la ventana. La gráfica muestra un comportamiento similar en ambos equipos, pero con diferentes magnitudes. Hay una relativa estabilidad hasta 4096 Bytes. Después de esto, el uso aumenta significativamente, siendo mayor en el cliente, donde llega a ocupar el 60 % de capacidad de CPU cuando la ventana es 65.535 Bytes, mientras que en el servidor la ocupación es de aproximadamente 26 %.

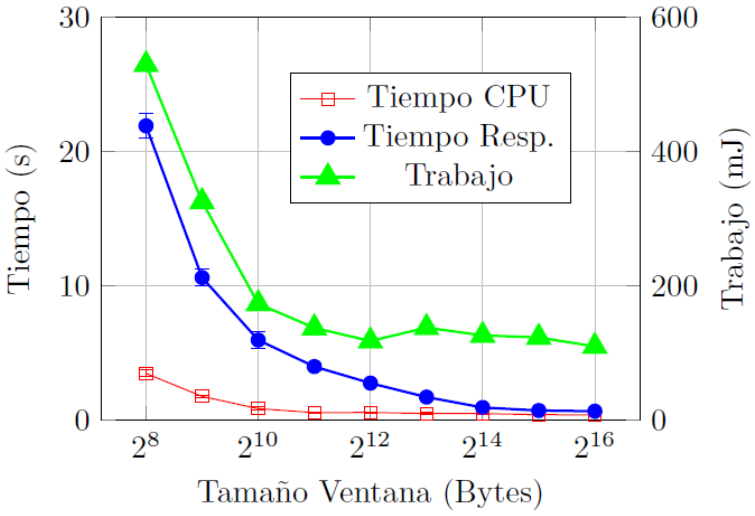


Figura 4.2: Tiempo de respuesta, Tiempo de CPU y Trabajo versus Tamaño de ventana en el cliente

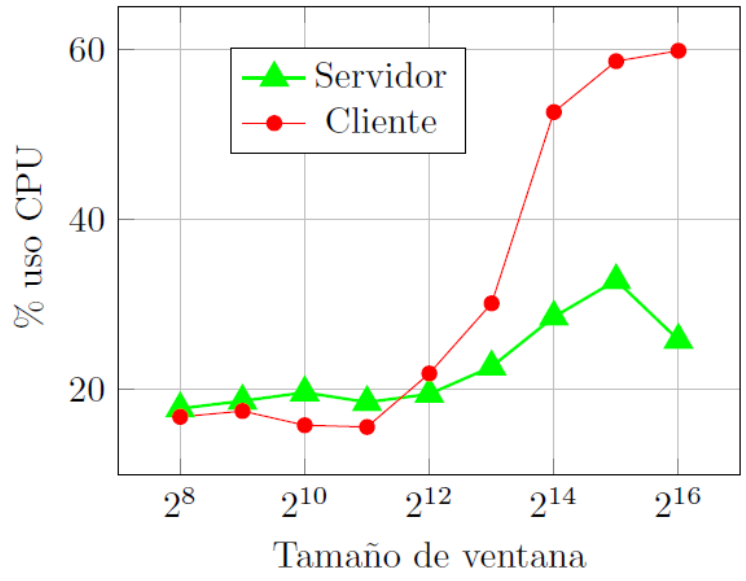


Figura 4.3: Porcentaje CPU versus Tamaño de ventana en cliente y servidor

En las figuras 4.2 y 4.3 se puede observar que el trabajo realizado por el cliente y el porcentaje de uso de CPU en el servidor y cliente tienen un punto de quiebre alrededor de 4096 Bytes. Según los datos del cliente, en ese punto el trabajo es similar al que hace la Raspberry Pi cuando tiene el valor por defecto del protocolo (65 kB), aunque los tiempos de CPU y de respuesta son mayores. Estos valores y de acuerdo al *Internet-draft* [69] llevan a la conclusión que 4096 Bytes es un valor adecuado para el tamaño de la ventana de control de tráfico en una adaptación de HTTP/2, ya que tiene la mejor relación desempeño-carga energética. Cabe destacar que en la capa de transporte no se hace una fragmentación en 4096 Bytes, por lo que el comportamiento observado en este punto podría deberse a la paginación de memoria.

Capítulo 5

Adaptación de HTTP/2 a entornos IoT

5.1. Control de flujo

Un algoritmo de control de flujo tiene como objetivo garantizar que no se vean saturados los recursos de los dispositivos durante una transmisión. Básicamente, el funcionamiento consiste en que el dispositivo receptor le indica, mediante un mensaje, al emisor cuál es el espacio en memoria que ha separado para la tarea de recepción. El emisor no puede exceder la cantidad señalada, puesto que de hacerlo los paquetes podrían ser descartados y tener que hacer retransmisiones más adelante. El espacio disponible en la memoria (o *buffer*) varía durante la transmisión, ya que además de recibir y almacenar los paquetes entrantes, paralelamente está sacando los paquetes que envía a la aplicación. La velocidad con la que recibe y saca paquetes del *buffer* pueden ser diferentes, por lo que corresponde al algoritmo de control de flujo que el emisor conozca esas variaciones en el *búffer* receptor. El comportamiento de los mensajes para el control de flujo depende de la definición de cada algoritmo.

La Figura 5.1 muestra de manera simplificada el fucionamiento de un algoritmo de control de flujo. En un principio, se agregan elementos a una cola de transmisión. Luego, el emisor transmite paquetes de datos, que viajan a través de la red al receptor. En este punto, los paquetes ingresan al *buffer* a una velocidad Vi(t), donde se almacenan hasta que son leídos por la aplicación. La velocidad de lectura se representa como Vo(t).

5.2. Propuesta de algoritmo de control de flujo

Los escenarios de IoT son restringidos, puesto que los dispositivos que interactúan en la red tienen capacidades limitadas. Por esta razón, los protocolos diseñados para

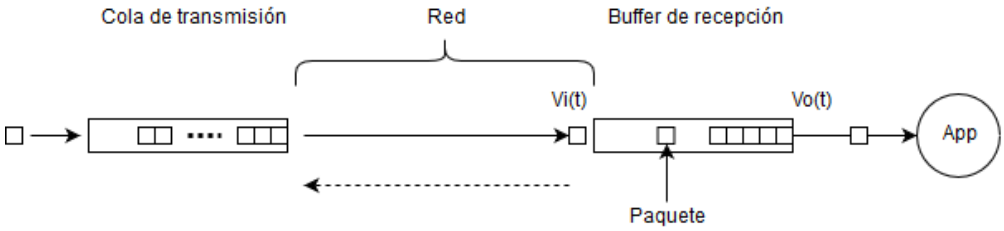


Figura 5.1: Esquema simplicado de un algoritmo de control de flujo

$T_{rec-trans}$	Tiempo de propagación en la red del receptor al emisor. [s]
T_{trans}	Tiempo de procesamiento para la transmisón de un Byte. [s/Byte]
$T_{trans-rec}$	Tiempo de propagación en la red del emisor al receptor. [s]
T_{rec}	Tiempo de lectura de un Byte en el receptor. [s/Byte]
W_{i}	Tamaño de la ventana. [Bytes]

Tabla 5.1: Resumen de tiempos implicados en la transmisión.

IoT tienden a ser sencillos y livianos, a diferencia de protocolos tradicionales, tales como HTTP o TCP, que suelen ser robustos y pesados para un dispositivo restringido. Partiendo de lo anterior, se plantea un algoritmo de control de flujo que simplifique el tráfico de control y reduzca la carga sobre los dispositivos.

El funcionamiento del algoritmo propuesto se ilustra en la Figura 5.2. El intercambio de paquetes comienza cuando el receptor envía un paquete del tipo SETTING_INITIAL_WINDOW_SIZE. Con este paquete el receptor le comunica al emisor el tamaño del *buffer* en Bytes, que ha separado para la recepción de paquetes. Al espacio libre del *buffer* se le conoce como ventana y el tamaño de ésta varía durante la transmisión debido a la entrada y salida de paquetes al *buffer*. La ventana inicial corresponde, entonces, al tamaño del *buffer* y es el valor máximo que esta puede tomar.

Después de recibido el paquete SETTING_INITIAL_WINDOW_SIZE, el emisor empieza a transmitir los paquetes DATA. Se transmiten tantos Bytes como la ventana lo permita, y una vez se transmita ese número, la transmisión se detiene. Paralelamente, cuando el receptor reciba la cantidad de Bytes anunciada en la ventana, envía un paquete WINDOW_UPDATE, donde le indica al emisor cuál es su ventana en ese momento, es decir, cuánto espacio libre tiene en el *buffer*. Tras recibido el paquete WINDOW_UPDATE por el emisor, se reanuda la transmisión y se repite el proceso.

Los valores mostrados en la Figura 5.2 se resumen en la Tabla 5.1.

A continuación se muestra un pseudocódigo del protocolo tanto del lado del emisor como del lado del receptor.

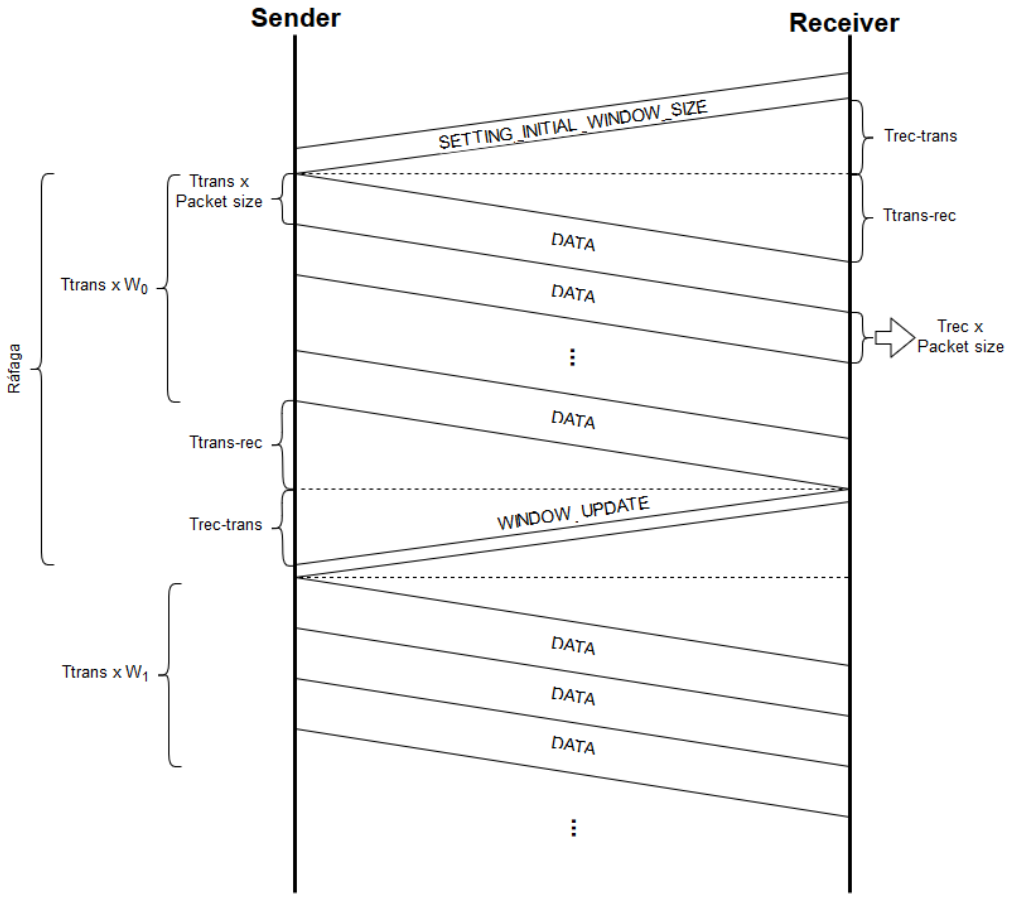


Figura 5.2: Flujo de paquetes en el algoritmo propuesto

```
En el transmisor:

bytes_a_transmitir = x;
Si recibe SETTING_INITIAL_WINDOW_SIZE == y:
ventana = y

    Hacer:
        enviar byte
        bytes_a_transmitir = bytes_a_transmitir-1
        ventana = ventana-1
    Mientras ventana > 0 & bytes_a_transmitir > 0

Si recibe WINDOW_UPDATE == z:
Si z == 0:
    esperar
En caso contrario:
ventana = z

    Hacer:
        enviar byte
        bytes_a_transmitir = bytes_a_transmitir-1
        ventana = ventana-1
    Mientras ventana > 0 & bytes_a_transmitir > 0



En el receptor:

Inicializar buffer de tamaño y
Enviar SETTING_INITIAL_WINDOW_SIZE = y
ventana = SETTING_INITIAL_WINDOW_SIZE
contador_bytes = 0
Hacer:
    escuchar canal
    Si llega un byte:
        encolar byte
        contador_bytes = contador_bytes+1
    Si contador_bytes == ventana
        ventana = espacio libre en el buffer
        WINDOW_UPDATE = ventana
        enviar WINDOW_UPDATE
        contador_bytes = 0
Mientras la conexión esté activa
```

Cuando contador_bytes alcanza el valor de ventana, se debe actualizar el valor de ésta variable. Dicho valor está determinado por el espacio libre que tenga el *buffer* en ese momento. En la práctica esto es una consulta a la cola del *buffer*, debido a que el

espacio libre depende de la tasa de llegada y lectura de paquetes.

5.3. Análisis teórico de desempeño

El algoritmo de control de flujo se puede modelar matemáticamente a través de una métrica como el *throughput*. Basado en [19], el *throughput* máximo en una ráfaga se puede modelar como se muestra en la ecuación 5.1.

$$\lambda_{sup} = min\left(\frac{8}{T_{trans}}, \frac{8}{T_{rec}}, \frac{8W}{W \times T_{trans} + T_{trans-rec} + T_{rec-trans}}\right) \tag{5.1}$$

Cuando el tiempo de transmisión es muy elevado en relación a los demás tiempos, el transmisor se comporta como un cuello de botella. La ecuación 5.2 modela esta situación.

$$\lambda_{sup}\left[\frac{bits}{s}\right] = \frac{8\left[\frac{bits}{Byte}\right]}{T_{trans}\left[\frac{s}{Byte}\right]} \tag{5.2}$$

En el caso en que el receptor actúe como cuello de botella, el modelamiento sería como se muestra en la ecuación 5.3.

$$\lambda_{sup}\left[\frac{bits}{s}\right] = \frac{8\left[\frac{bits}{Byte}\right]}{T_{rec}\left[\frac{s}{Byte}\right]} \tag{5.3}$$

Los anteriores casos podrían presentarse cuando hay bloqueos o colapsos en los dispositivos o sencillamente cuando alguno de estos es muy lento en relación al otro. En el caso en que las velocidades del receptor y transmisor sean similares, el *throughput* será determinado por todos los tiempos que interactúan en la transmisión, como se muestra en la ecuación 5.4.

$$\lambda_{sup}\left[\frac{bits}{s}\right] = \frac{8\left[\frac{bits}{Byte}\right] W\,[Bytes]}{W\,[Bytes] \times T_{trans}\left[\frac{s}{Byte}\right] + T_{trans-rec}\,[s] + T_{rec-trans}\,[s]} \tag{5.4}$$

Las ecuaciones compiladas en la ecuación 5.1 aplican para cada W, es decir para cada ráfaga. Una transmisión normalmente consta de múltiples ráfagas, por lo tanto,

el *throughput* promedio de toda la transmisión debe tener en cuenta la duración de la misma, que está determinada en la ecuación 5.5.

$$T_{total} = \sum_{\mathrm{i}=0}^{n} T_{parcial_{\mathrm{i}}} = \sum_{\mathrm{i}=0}^{n} W_{\mathrm{i}} \times T_{trans_{\mathrm{i}}} + T_{trans-rec_{\mathrm{i}}} + T_{rec-trans_{\mathrm{i}}} \tag{5.5}$$

Finalmente, teniendo en cuenta las anteriores ecuaciones, el *throughput* total se establece en la ecuación 5.6, donde n es el número de ráfagas, que a su vez está determinado por el número de paquetes WINDOW_UPDATE enviados por el receptor. Hay que tener en cuenta que el i = 0 hace referencia a la primera ráfaga, la cual está determinada por la ventana inicial anunciada en el paquete SETTINGS_INITIAL_WINDOW_SIZE.

$$\lambda_{total}\left[\frac{bits}{s}\right] = \sum_{\mathrm{i}=0}^{n} \frac{8\left[\frac{bits}{Byte}\right] W_{\mathrm{i}}\,[Bytes]}{T_{total}\,[s]} \tag{5.6}$$

Por otro lado, el tamaño de la ventana W está determinado por la relación entre las velocidades de entrada y salida al *buffer* de recepción, además del tamaño de la ventana. Se puede expresar W para una ráfaga en función del tiempo, como se muestra en la ecuación 5.7.

$$W(t) = W_{\mathrm{inicial}} - V_{\mathrm{i}}(t) + V_o(t) \tag{5.7}$$

Donde:

$$0 \leq W(t) \leq W_{\mathrm{inicial}}$$

$$V_{\mathrm{i}}(t) = Velocidad\text{ de }entrada\ al\ buffer.$$

$$V_o(t) = Velocidad\text{ de }salida\text{ del }buffer.$$

La ecuación 5.7 se usa para expresar el tamaño de la ventana en función del tiempo, sin embargo, esto puede resultar difícil para el modelamiento. Por ello, es más conveniente hacer dicho modelamiento basándose en teoría de colas. De esta manera, se asume el *buffer* como una cola de capacidad K a la que ingresan elementos y esperan a ser atendidos por un servidor.

Según [17][48][87], existen varios modelos de colas según su tasa de entrada y salida a la misma, el número de servidores que atienden a los elementos de la cola y si esta es finita o infinita. Según el comportamiento del sistema planteado por el algoritmo propuesto, la tasa de llegada de los paquetes a la cola no es constante, puesto que la red por la que son enviados varía sus condiciones. Por lo que, es más realista asumir que la tasa de llegada sigue una distribución de *Poisson*. La tasa de

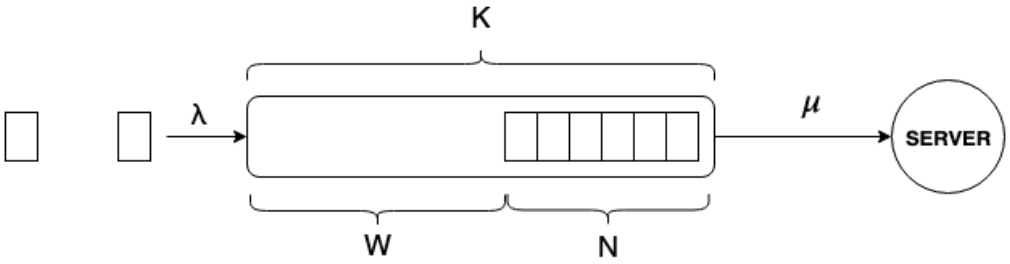


Figura 5.3: Representación de una cola M/M/1/K.

salida tampoco es constante, debido a que pueden haber variaciones en el uso del dispositivo receptor. Para el sistema, existe un sólo servidor que atiende la cola, y como ya se dijo anteriormente, la capacidad máxima es K. Este comportamiento es propio de las colas M/M/1/K, y para este caso la disciplina de entrada y salida es FIFO (*First Input First Output*. La Figura 5.3 muestra el modelo anteriormente descrito. De [48] se obtiene que, la ocupación promedio de la cola es como se expresa en la ecuación 5.8.

$$N = \frac{\rho}{1-\rho} - \frac{(K+1)\rho^{K+1}}{1-\rho^{K+1}} \tag{5.8}$$

Donde:

$$\rho = \frac{\lambda}{\mu} \tag{5.9}$$

A su vez:

$\lambda = la\ tasa\ prom$edio de $llega$d$a\ a\ la\ cola.$

$\mu = la\ tasa\ prom$edio de salid$a$ de $la\ cola.$

Finalmente, la ventana promedio de la cola es:

$$W = K - N \tag{5.10}$$

Cabe destacar que las ecuaciones 5.8 y 5.10, así como la ecuación 5.6, aplican cuando las diferencias entre la tasa de entrada y salida a la cola son pequeñas. Es decir, cuando las velocidades de envío del transmisor y las velocidades de lectura del receptor sean similares, aplican las anteriores ecuaciones. En el caso contrario, cuando estas velocidades tengan diferencias significativas, el *throughput* total estaría determinado por la tasa más lenta, como se considera en la ecuación 5.1.

La ecuación 5.8 es aplicable cuando la velocidad de llegada a la cola es menor o similar a la velocidad de lectura. Si la velocidad de llegada es mayor, matemáticamente, la cola se va a llenar rápidamente y va a colapsar. Aunque, en realidad, esto

no sucede cuando se usa el algoritmo de control de flujo, ya que su objetivo es precisamente evitar que los dispositivos colapsen. Si la velocidad de lectura es menor a la de llegada, el algoritmo va a regular la velocidad de llegada por medio de las ventanas anunciadas. En otras palabras, va a empezar a anunciar ventanas pequeñas de tal manera que la velocidad de llegada va a ser muy similar a la velocidad de lectura. En este orden de ideas, debido a que la ecuación 5.8 no permite hallar la ocupación media en el caso mencionado, es necesario hallarla de otra forma.

En la Figura 5.2 del lado del receptor se ve que cada ráfaga que llega tiene asociado los tiempos de propagación $T_{rec-trans}$ y $T_{trans-rec}$; además que, cada ráfaga que llega al receptor tiene un tamaño W_{i}, que es directamente proporcional al tiempo que demora el receptor en leer esa ráfaga. Por lo tanto, se puede expresar el tiempo que toma transmitir W_{i} Bytes como se manifiesta en 5.11 y 5.13.

En la primera ráfaga se transmite W_0 Bytes, cuyo valor es el de la ventana inicial. El tiempo total para transmitir la primera ráfaga se puede modelar como se muestra en la ecuación 5.11, y a su vez la primera ventana anunciada en la transmisión se expresa en la ecuación 5.12.

$$T_{t_0}[s] = \frac{8[\frac{bits}{Bytes}] \times W_0[Bytes]}{\lambda_t[\frac{bits}{s}]} \tag{5.11}$$

$$W_1[Bytes] = \left| \frac{\lambda_r[\frac{bits}{s}] \times T_{t_0}[s]}{8[\frac{bits}{Bytes}]} \right| \tag{5.12}$$

Posteriormente T_ti y W_{i} estarán determinadas por las ecuaciones 5.13 y 5.14, respectivamente.

$$T_{t_{\mathrm{i}}}[s] = \frac{8[\frac{bits}{Bytes}] \times W_{(\mathrm{i}-1)}[Bytes]}{\lambda_t[\frac{bits}{s}]} + T_{trans-rec}[s] + T_{rec-trans}[s] \tag{5.13}$$

$$W_{\mathrm{i}}[Bytes] = \left| \frac{\lambda_r[\frac{bits}{s}] \times T_{t_{\mathrm{i}-1}}[s]}{8[\frac{bits}{Bytes}]} \right| \tag{5.14}$$

Cuando el receptor recibe el último Byte de la transmisión, el buffer aún tiene Bytes represados por leer. El tiempo que le llevará al receptor vaciar el buffer se representa como T_{ad} y se modela como en la ecuación 5.15.

$$T_{\mathrm{ad}}[s] = \frac{8 \times (W_0[Bytes] - W_{prom}[Bytes])}{\lambda_r[\frac{bits}{s}]} \tag{5.15}$$

Así, el thorughput total de la transmisión estaría determinado por la ecuación 5.16.

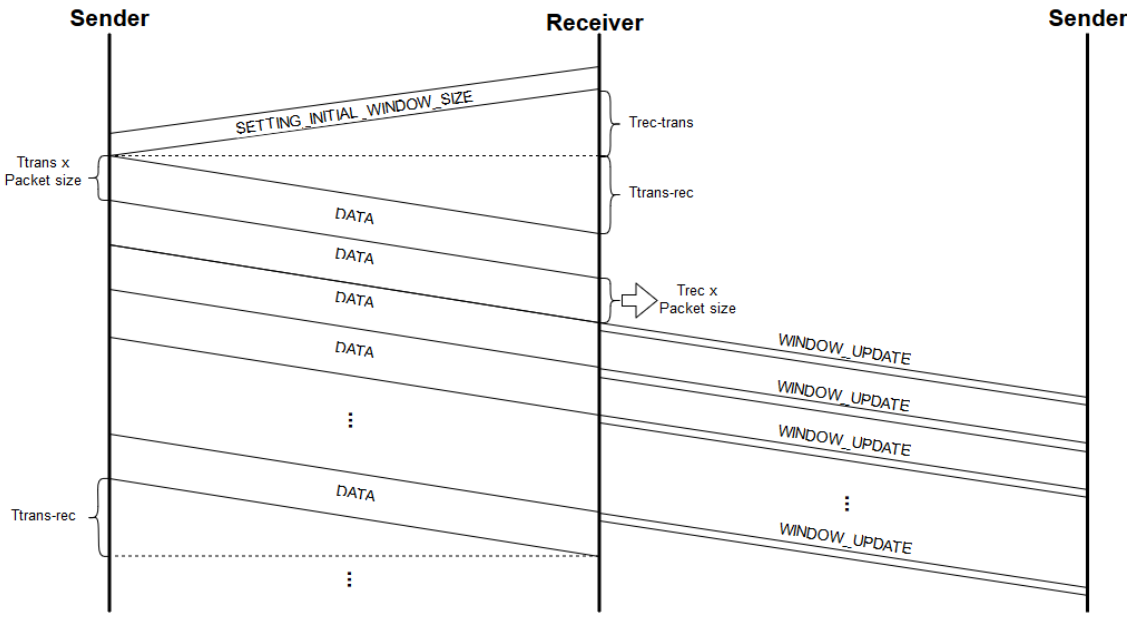


Figura 5.4: Flujo de paquetes en el algoritmo de referencia.

$$\lambda_t[\frac{bits}{s}] = \frac{8[\frac{bits}{Byte}] \times B_{total}[Bytes]}{T_{total}[s] + t_{ad}[s] + T_{trans-rec}[s]} \tag{5.16}$$

En la sección de Anexos se muestran los códigos de Matlab donde se implementan las anteriores ecuaciones y se pueden probar los presentes planteamientos.

5.4. Evaluación mediante simulaciones

5.4.1. Algoritmo de referencia

Con el objetivo de comparar el algoritmo propuesto frente a otro que ya estuviera en uso en alguna implementación de HTTP/2, se decide analizar el algoritmo utilizado por NGHTTP/2 [71]. Esta es una implementación que no está orientada a IoT, por lo que siguiendo los objetivos del protocolo, busca agilidad para aplicaciones web actuales. En ese sentido, el algoritmo de control de flujo que utiliza, se orienta a una transferencia de datos rápida y fluida.

La diferencia fundamental entre el algoritmo propuesto y el algoritmo de referencia, es que este último no espera a recibir la totalidad de datos que anunció en la ventana, sino que cuando el *buffer* está lleno a la mitad o más, envía un paquete WINDOW_UPDATE. En otras palabras, el receptor va a anunciar su tamaño de ventana cada vez que reciba un paquete de datos y su ventana disponible sea la mitad o menos de su tamaño inicial o máximo. El funcionamiento de este algoritmo se muestra en la figura 5.4.

Gráficamente, al comparar las Figuras 5.2 y 5.4, se identifica que el algoritmo de referencia logra una comunicación más fluída y rápida al no hacer pausas, pero también, utiliza más paquetes WINDOW_UPDATE.

El modelamiento matemático del algoritmo de referencia se muestra en la ecuación 5.17.

$$\lambda_{sup} = min\left(\frac{8}{T_{trans}}, \frac{8}{T_{rec}}, \frac{8W}{W \times T_{trans} + T_{trans-rec}}\right) \tag{5.17}$$

Al igual que en el algoritmo propuesto, si el transmisor es el cuello de botella, el *throughput* máximo en una ráfaga estaría determinado por la ecuación 5.2. Por otro lado, si el cuello de botella es el receptor, el *throughput* está determinado por la ecuación 5.3. Para el caso donde las velocidades son similares, el *throughput* de una ráfaga se expresa como en la ecuación 5.18.

$$\lambda_{sup}\left[\frac{bits}{s}\right] = \frac{8\left[\frac{bits}{Byte}\right] W\,[Bytes]}{W\,[Bytes] \times T_{trans}\left[\frac{s}{Byte}\right] + T_{trans-rec}\,[s]} \tag{5.18}$$

A diferencia de la ecuación 5.4, en 5.18 no se tiene en cuenta el tiempo de propagación en la red entre el receptor y el transmisor, debido a que no hay pausas. Por lo tanto, para hallar el *throughput* promedio de toda la transmisión, es necesario obtener el tiempo total de la transmisión, como se muestra en la ecuación 5.19.

$$T_{total} = \sum_{\mathrm{i}=0}^{n} T_{parcial_{\mathrm{i}}} = \sum_{\mathrm{i}=0}^{n} W_{\mathrm{i}} \times T_{trans_{\mathrm{i}}} + T_{trans-rec_{\mathrm{i}}} \tag{5.19}$$

Finalmente, teniendo en cuenta la ecuación 5.19, el *throughput* promedio total se define en la ecuación 5.20.

$$\lambda_{total}\left[\frac{bits}{s}\right] = \sum_{\mathrm{i}=0}^{n} \frac{8\left[\frac{bits}{Byte}\right] W_{\mathrm{i}}\,[Bytes]}{T_{total}\,[s]} \tag{5.20}$$

Para determinar W, aplican las ecuaciones 5.8 y 5.10, debido a que el comportamiento también es el de una cola M/M/1/K, donde K es el tamaño inicial de la ventana. En la sección anexos se encuentra el código de MATLAB donde se implementan los planteamientos realizados.

Tiempos de propagación en ambas direcciones	100 ms y 200 ms
Ventanas iniciales	2048, 4096, 8192, 16384, 32768, 65536
Tamaño de los paquetes	512 B
Bytes a transmitir	500 kB
Velocidades de transmisión	10 kbps, 100 kbps, 500 kbps
Velocidades de lectura en el receptor	10 kbps, 100 kbps, 500 kbps
Algoritmos evaluados	Propuesto y referencia

Tabla 5.2: Resumen de parámetros evaluados en las simulaciones.

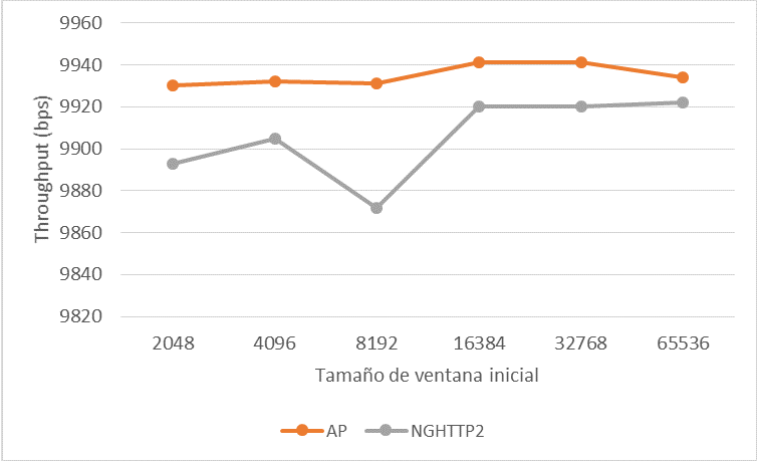
5.4.2. Simulador

Con el objetivo de simular la transferencia de datos entre emisor y receptor, tanto para el algoritmo propuesto como para el algoritmo de NGHTTP2, se construyó un simulador en Java, ya que surge de la necesidad de simular el funcionamiento específico de los protocolos de control de flujo. Su funcionamiento se basa en una cola que simula la ventana y una ejecución con dos hilos que simulan el emisor y el receptor. Este simulador permite modificar variables como: tamaño de paquetes, velocidad de lectura en el receptor, velocidad de envío en el emisor, tiempos de propagación, tamaño de la ventana inicial y cantidad de datos a transmitir. En la Tabla 5.4.2 se resumen los parámetros evaluados en las simulaciones tanto del algoritmo propuesto (AP) como del algoritmo de NGHTTP2.

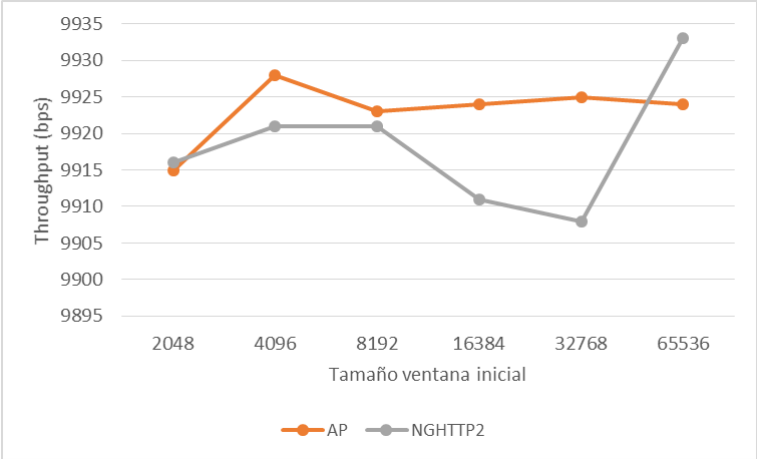
Las métricas definidas para hacer la evaluación son el *throughput* y el tráfico de control u *overhead* necesario para la transferencia. Este último hace referencia a la cantidad de paquetes WINDOW_UPDATE que el receptor envía al emisor. Se definen estas métricas porque en un ambiente de IoT se busca la simplicidad del protocolo, por lo que un bajo número de paquetes de control es deseable, sin que esto signifique un sacrificio muy grande en el *throughput*. El código de los simuladores construídos se encuentra en [41].

5.4.3. Simulaciones

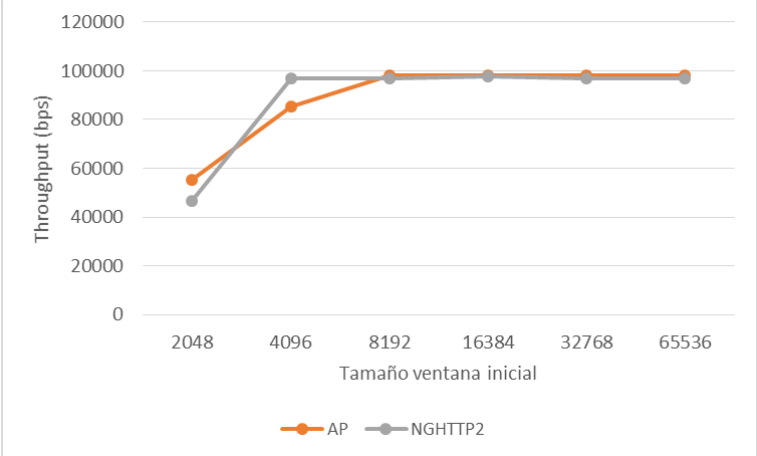
Los resultados de las simulaciones realizadas y los respectivos análisis de los datos se presentan organizados en gráficas, agrupadas en tres escenarios: cuando el receptor es muy demorado en comparación con el emisor, cuando el emisor y receptor tienen velocidades similares y cuando el receptor es mucho más rápido que el emisor. Debido a que las simulaciones están basadas en algoritmos con valores constantes para cada escenario, cada uno de estos fue corrido una vez.



(a) *Throughput* 100 kbps - 10 kbps - *Delay* 100 ms



(b) *Throughput* 500 kbps - 10 kbps - *Delay* 100 ms



(c) *Throughput* 500 kbps - 100 kbps - *Delay* 100 ms

Figura 5.5: *Throughput* cuando el receptor es más lento en relación al emisor

Escenario 1: Receptor más lento

Las figuras 5.5 y 5.6 recogen los resultados cuando los tiempos de lectura son menores a los tiempos de transmisión y el tiempo de propagación es 100 ms. Los resultados en este segmento muestran una significativa reducción del tráfico de control cuando se usa el algoritmo propuesto, en comparación con el algoritmo de NGHTTP2. Dicha reducción es en promedio del 26 % para el caso 100 kbps - 10 kbps, 32 % para 500 kbps - 10 kbps y 65 % para 500 kbps - 100 kbps. Los mayores volúmenes de *overhead* se presentan en el escenario 500 kbps - 10 kbps. Esto se debe al desequilibrio de las velocidades, lo que provoca que el *buffer* esté casi lleno la mayor parte del tiempo y que las ventanas anunciadas sean más pequeñas que en los otros casos.

En términos de *throughput* no hay mayores diferencias, el rendimiento del algoritmo propuesto es igual o incluso ligeramente superior para la mayoría de los casos. Esto quiere decir que el uso del algoritmo propuesto no significa un sacrificio para el desempeño del protocolo, en el escenario en el que el receptor sea muy lento en relación al emisor. La razón para este resultado es que el protocolo de NGHTTP2 al comienzo de la transmisión sufre pérdida de paquetes debido a la relativa lentitud del receptor para leer el *buffer* y el *delay*. La secuencia donde se configura la pérdida de paquetes es:

1. El emisor recibe un paquete SETTING_INITIAL_WINDOW_SIZE, donde el receptor le anuncia el tamaño de su ventana inicial de control de flujo.

2. El emisor empieza a enviar tantos paquetes de datos como se lo permita la ventana anunciada.

3. Mientras tanto el receptor recibe y almacena en el *buffer* los paquetes recibidos.

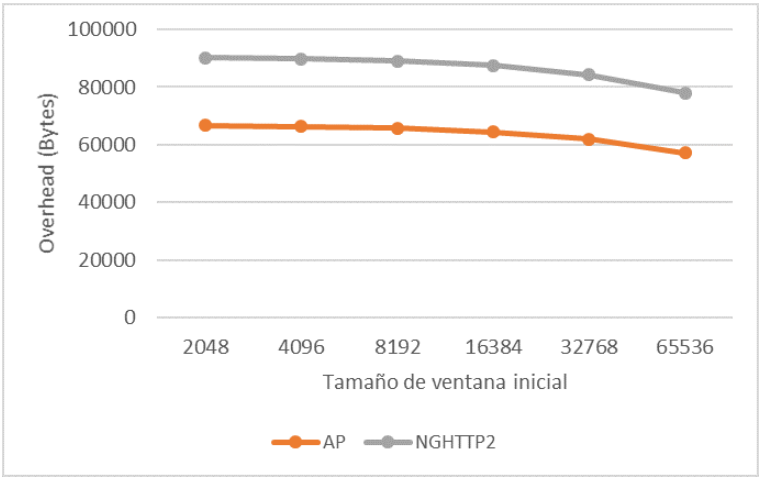
4. Cuando el *buffer* del receptor está lleno hasta la mitad, envía un paquete WINDOW_UPDATE actualizando su tamaño disponible.

5. Por el tiempo de propagación existente entre receptor y emisor, este último no recibe inmediatamente ese paquete WINDOW_UPDATE. Mientras el paquete WINDOW_UPDATE viaja hasta el emisor, este ya ha enviado tantos Bytes como se lo permitió la ventana inicial.

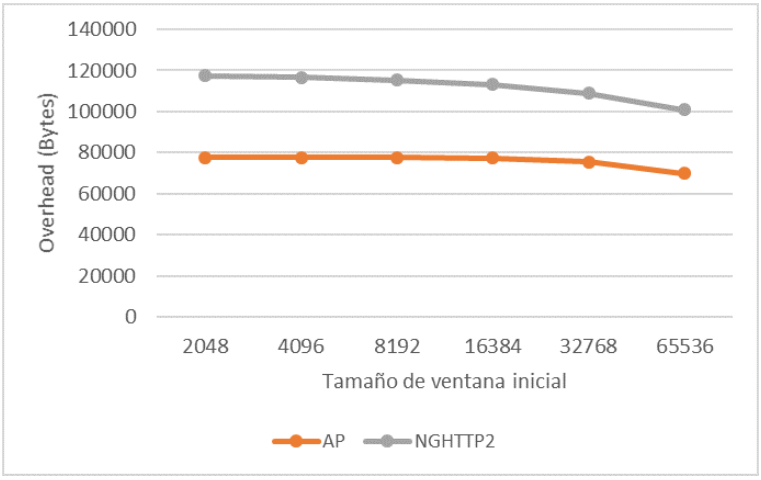
6. El emisor recibe el paquete WINDOW_UPDATE que le autoriza a enviar una cantidad de Bytes que está alrededor de la mitad de la ventana inicial, pero para ese momento el *buffer* del receptor no va a tener disponible ese espacio, debido a que ya recibió todos (o casi todos) los Bytes de la primera ráfaga y a que es muy lento para leerlos.

7. El emisor envía la segunda ráfaga con la cantidad de datos que le indicó el último paquete WINDOW_UPDATE. Como el *buffer* de recepción aún no tiene suficiente espacio para recibirlos, algunos de estos serán descartados.

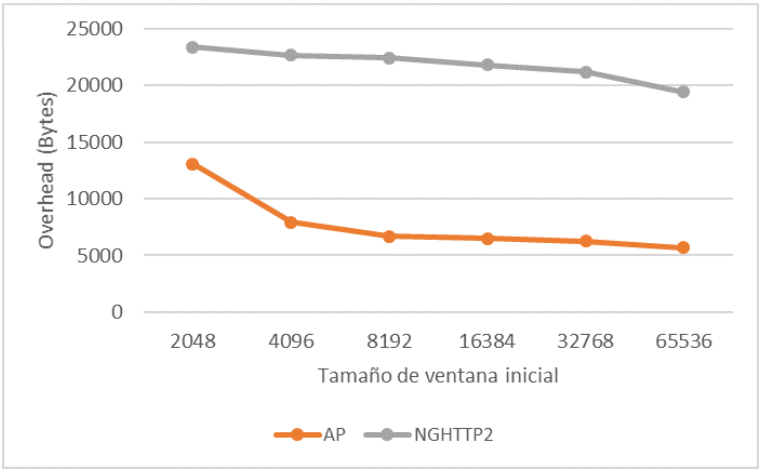
8. En el algoritmo de NGHTTP2 implementado en la simulación, el receptor revisa



(a) *Overhead* 100 kbps - 10 kbps - *Delay* 100 ms



(b) *Overhead* 500 kbps - 10 kbps - *Delay* 100 ms



(c) *Overhead* 500 kbps - 100 kbps - *Delay* 100 ms

Figura 5.6: *Overhead* cuando el receptor es más lento en relación al emisor

el estado del *buffer* después de recibir cada paquete, por lo que enviará varios paquetes WINDOW_UPDATE que provocarán el descarte de paquetes.

El problema anteriormente narrado es temporal. Posteriormente, el sistema se estabiliza y las ventanas anunciadas empiezan a tener valores que no llevan al descarte de paquetes, ya que están regulados por los tiempos de propagación y lectura del *buffer*.

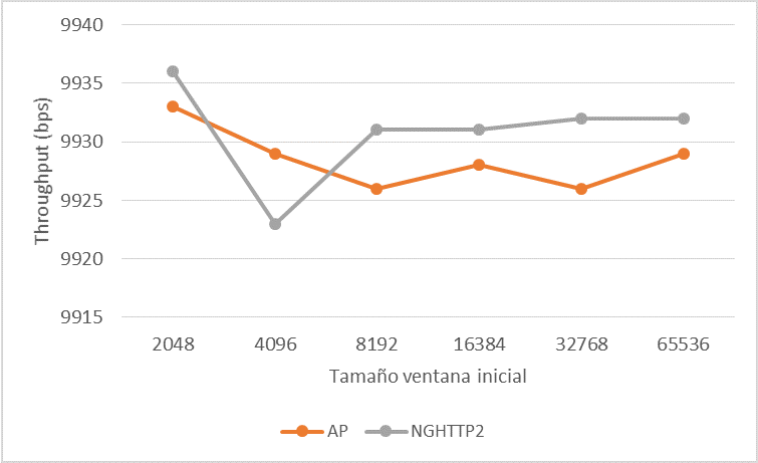
En el escenario recopilado en las Figuras 5.7 y 5.8, se muestra que un aumento de los tiempos de propagación disminuye el *overhead* debido a que los mayores tiempos de propagación provocan que el receptor disponga a su vez de más tiempo para leer el *buffer*. Por otro lado, en términos de *throughput*, el aumento del *delay* no lleva a diferencias significativas. La explicación es que, a pesar del aumento de los tiempos de propagación, el cuello de botella aún no es la red, sino que continúa siendo la velocidad de lectura del receptor.

Escenario 2: Emisor y receptor con velocidades similares

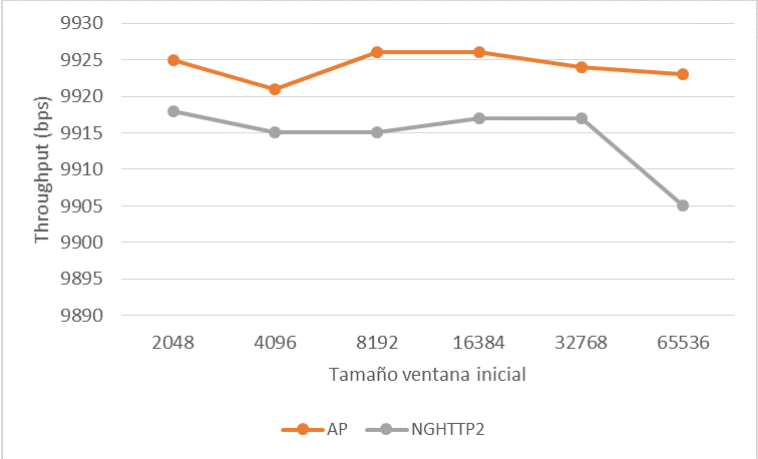
En la Figura 5.10 se muestra que el tráfico de control disminuye significativamente cuando se usa el algoritmo propuesto. En el caso de 10 kbps - 10 kbps disminuye en promedio un 69 %, en el caso 100 kbps - 100 kbps la disminución es de un 49 % y 48 % para el caso 500 kbps - 500 kbps. Los volúmenes de tráfico, al usar el algoritmo de NGHTTP2, tienden a ser mayores cuando se usan bajas velocidades, debido a que las ventanas anunciadas son más pequeñas. Por otro lado, los volúmenes de tráfico con el algoritmo propuesto son similares para todos los casos, excepto cuando la ventana inicial es 2048 Bytes.

En cuanto al *throughput*, en la Figura 5.9 el algoritmo de NGHTTP2 tiene un mejor desempeño en la mayoría de los escenarios. Cuando las velocidades son 10 kbps - 10 kbps, el algoritmo de NGHTTP2 es en promedio un 2 % más rápido, para el caso 100 kbps - 100 kbps es un 13 % más rápido y un 19 % más rápido en el escenario 500 kbps - 500 kbps. Sin embargo, hay que destacar que cuando las ventanas se agrandan las diferencias en throughput se reducen, e incluso el algoritmo propuesto logra un mejor desempeño. Esto debido a la baja cantidad de paquetes WINDOW_UPDATE utilizados por el algoritmo propuesto, ya que las ventanas anunciadas son cercanas a su valor máximo, permitiendo ráfagas de tráfico de datos más largas.

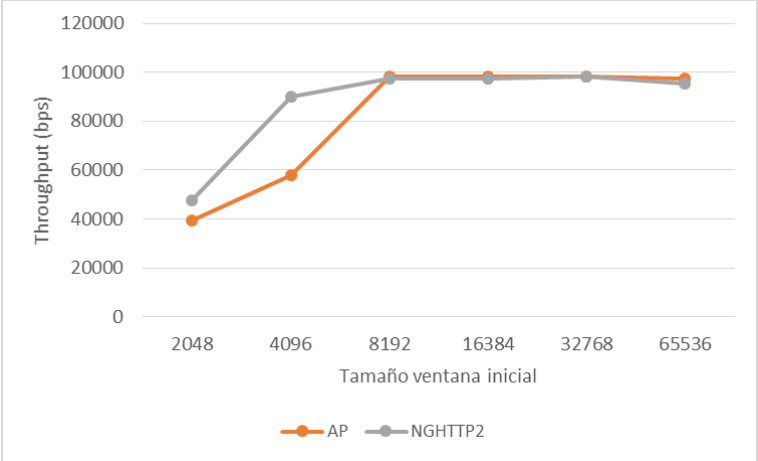
Las Figuras 5.11 y 5.12 muestran el caso cuando las velocidades son similares, pero el tiempo de propagación aumenta a 200 ms. En este escenario, el patrón de comportamiento es similar al de menor *delay*, pero en este caso el aumento en los tiempos de propagación sí afecta negativamente el *throughput* para ambos algoritmos, especialmente cuando las ventanas iniciales son pequeñas. El algoritmo propuesto es más sensible al aumento de los tiempos de propagación, debido a que hace pausas para esperar los paquetes WINDOW_UPDATE y a que en este caso el emisor y el receptor no actúan como cuellos de botella, por lo que las condiciones de red toman mayor relevancia.



(a) *Throughput* 100 kbps - 10 kbps - *Delay* 200 ms

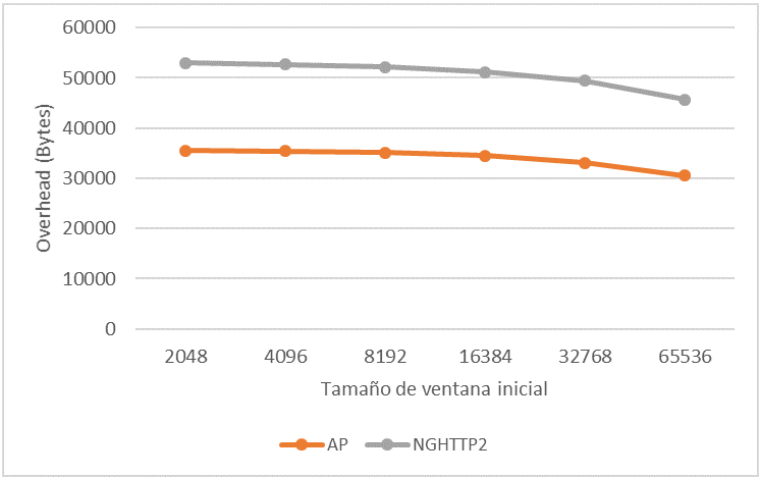


(b) *Throughput* 500 kbps - 10 kbps - *Delay* 200 ms

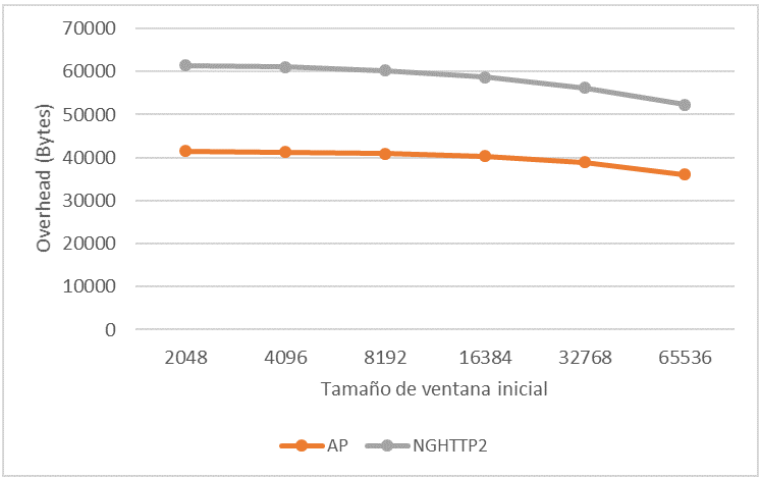


(c) *Throughput* 500 kbps - 100 kbps - *Delay* 200 ms

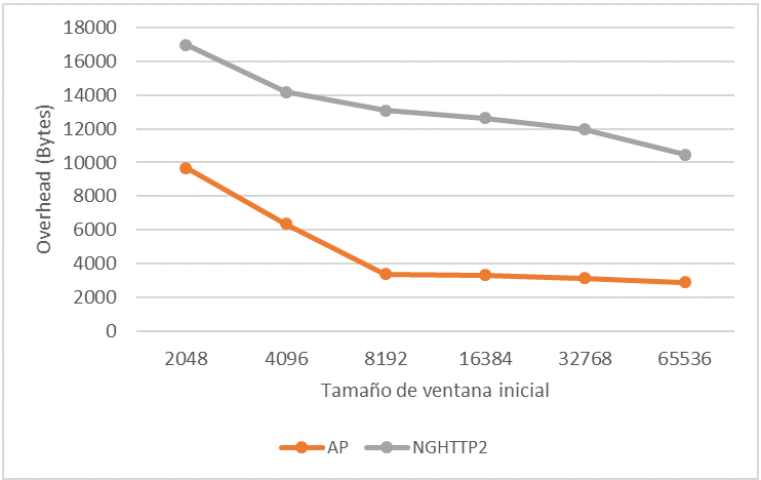
Figura 5.7: *Throughput* cuando el receptor es más lento en relación al emisor y el tiempo de propagación es 200 ms



(a) *Overhead* 100 kbps - 10 kbps - *Delay* 200 ms

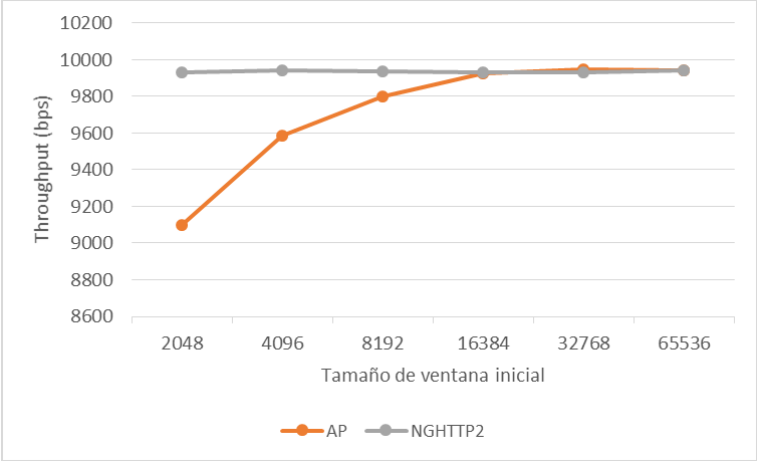


(b) *Overhead* 500 kbps - 10 kbps - *Delay* 200 ms

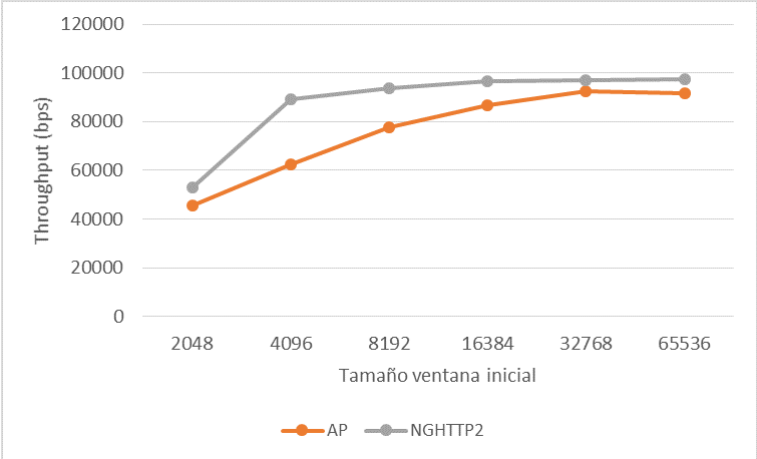


(c) *Overhead* 500 kbps - 100 kbps - *Delay* 200 ms

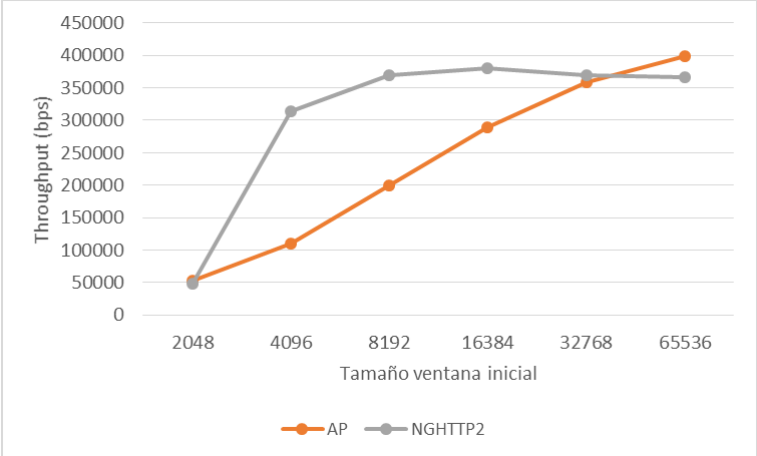
Figura 5.8: *Overhead* cuando el receptor es más lento en relación al emisor y el tiempo de propagación es 200 ms



(a) *Throughput* 10 kbps - 10 kbps - *Delay* 100 ms

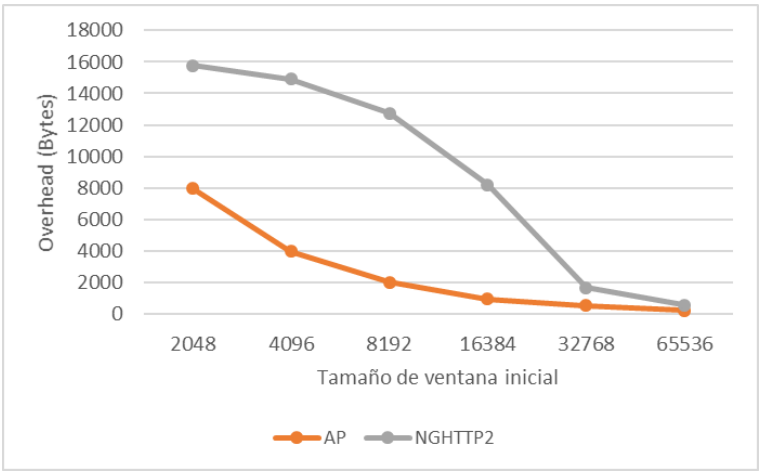


(b) *Throughput* 100 kbps - 100 kbps - *Delay* 100 ms

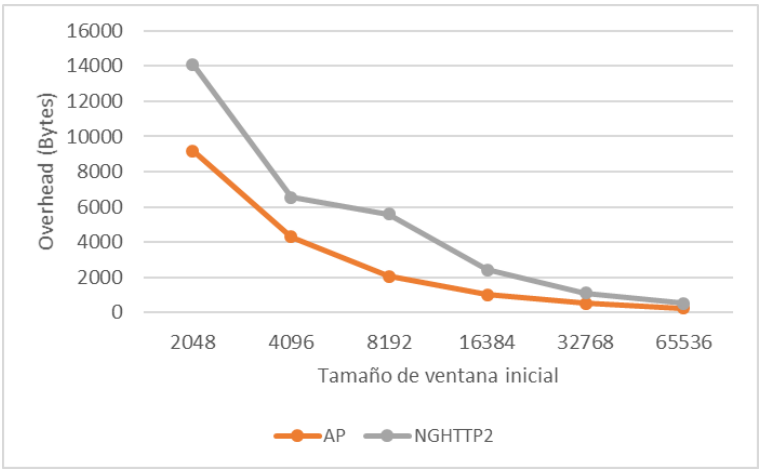


(c) *Throughput* 500 kbps - 500 kbps - *Delay* 100 ms

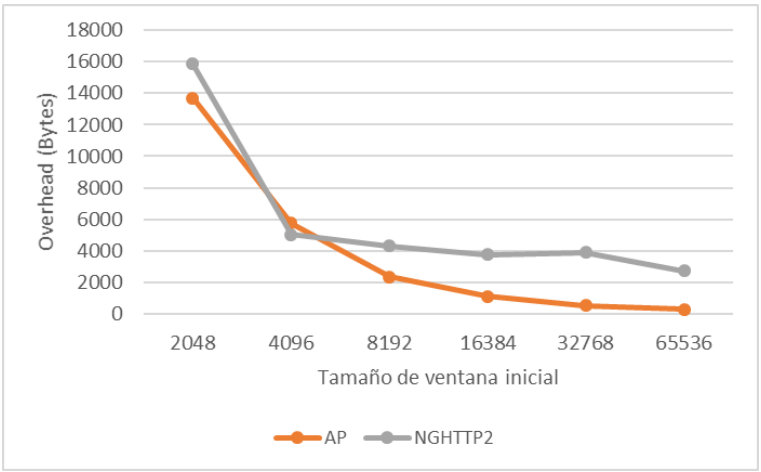
Figura 5.9: *Throughput* cuando el receptor y emisor tienen velocidades similares



(a) *Overhead* 10 kbps - 10 kbps - *Delay* 100 ms



(b) *Overhead* 100 kbps - 100 kbps - *Delay* 100 ms



(c) *Overhead* 500 kbps - 500 kbps - *Delay* 100 ms

Figura 5.10: *Overhead* cuando el receptor y emisor tienen velocidades similares

El *overhead* utilizado sigue siendo menor en el algoritmo propuesto, manteniendo los mismos patrones. En cuanto a los volúmenes de este tráfico, hay una ligera disminución respecto al escenario de 100 ms, especialmente en el algoritmo NGHTTP2; mientras que, el algoritmo propuesto mantiene valores similares a causa de la baja ocupación del *buffer* durante la transmisión para ambos *delays*.

Escenario 3: El receptor es más rápido que el emisor

En el escenario mostrado en la Figura 5.13 el transmisor actúa como un cuello de botella. La gran velocidad de lectura del receptor en comparación a la velocidad con la que le llegan datos va a permitir que el *buffer* se mantenga casi desocupado, por lo que las ventanas anunciadas van a ser valores cercanos al máximo. En este escenario, NGHTTP2 es un 3 % más rápido en promedio en el escenario 10 kbps - 100 kbps y 8 % más rápido en el escenario 100 kbps - 500 kbps, aunque en las ventanas más grandes las diferencias se reducen.

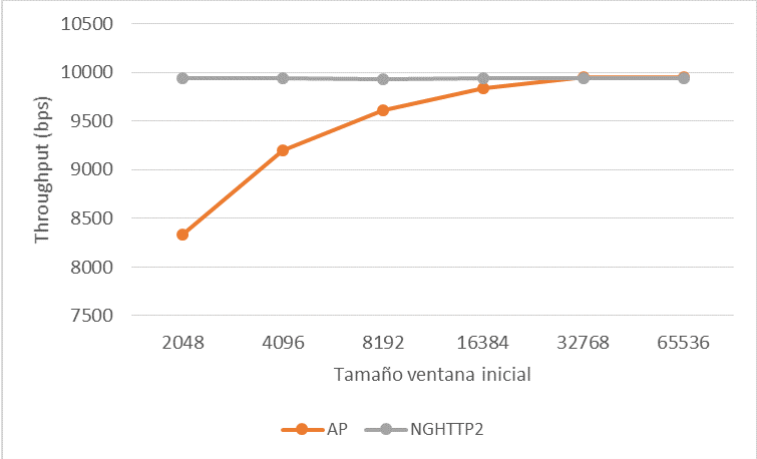
Por el lado del tráfico de control, el algoritmo propuesto disminuye el tráfico de control en promedio un 46 % para el escenario 10 kbps - 100 kbps y 55 % para el escenario 100 kbps - 500 kbps.

Cuando se aumentan los tiempos de propagación, como en el caso de la Figura 5.14, el tráfico de control adquiere valores similares al caso de 100 ms, ya que para ambos escenarios la relación entre velocidades de lectura y llegada de paquetes mantiene el *buffer* casi vacío, por lo que el número de paquetes WINDOW_UPDATE son parecidos. El algoritmo de NGHTTP2 logra mejor *throughput* que el algoritmo propuesto en todos los casos, pese a que las diferencias se acortan a medida que crecen las ventanas iniciales. Nuevamente, el algoritmo propuesto es más sensible al incremento de los tiempos de propagación, particularmente para ventanas iniciales pequeñas que son las que requieren un mayor número de tráfico de control.

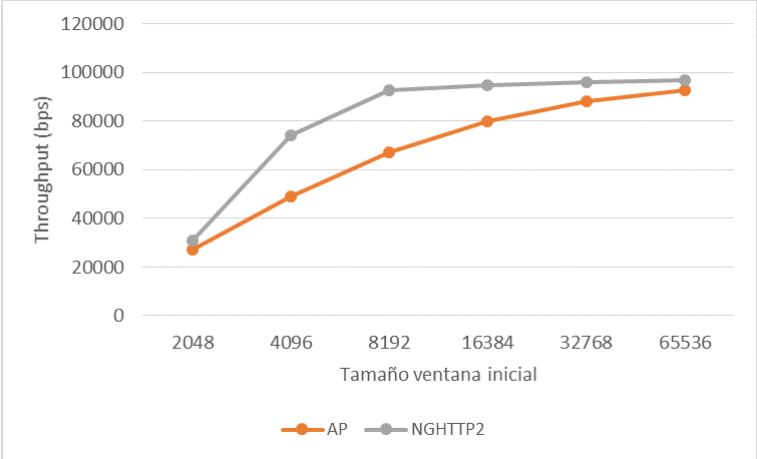
5.4.4. Comparación de resultados teóricos y simulados

A partir de las ecuaciones planteadas en las secciones 5.2 y 5.4.1, se obtienen los resultados de algunos de los escenarios más representativos y se comparan con sus respectivos resultados simulados. Esto con el fin de determinar qué tan ajustados están el simulador y el modelamiento matemático de los algoritmos.

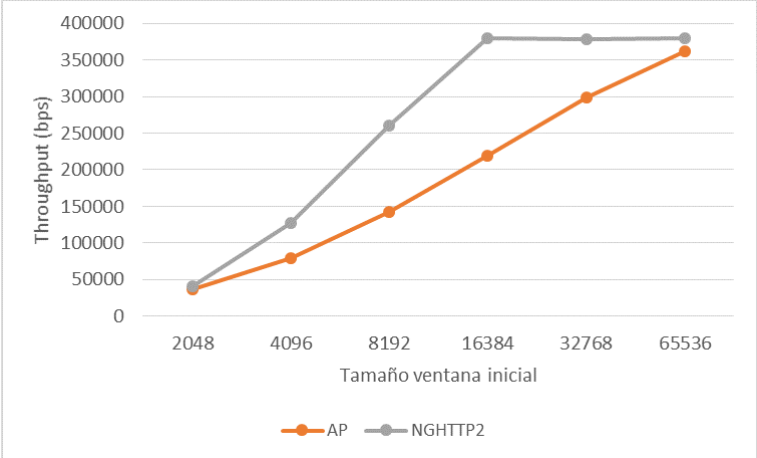
En la Figura 5.15 se muestra el *throughput* obtenido por medio del modelamiento matemático y de la simulación. Es posible afirmar que, en general, las diferencias no son muy grandes. En el caso del algoritmo de NGHTTP2, cuando las ventanas iniciales son pequeñas, los resultados simulados son menores a los teóricos. Esto se debe a que el planteamiento matemático considera una comunicación fluída, donde el cuello de botella es el nodo de menor velocidad; sin embargo, una ventana muy pequeña combinada con unos tiempos de propagación largos puede afectar la velocidad tal



(a) *Throughput* 10 kbps - 10 kbps - *Delay* 200 ms

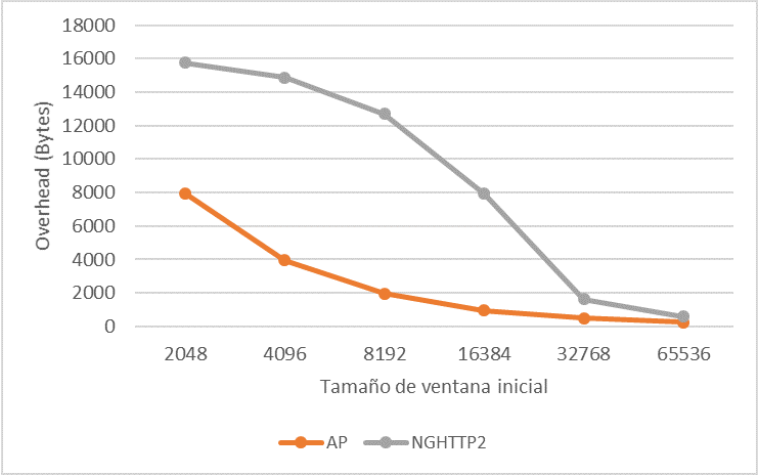


(b) *Throughput* 100 kbps - 100 kbps - *Delay* 200 ms

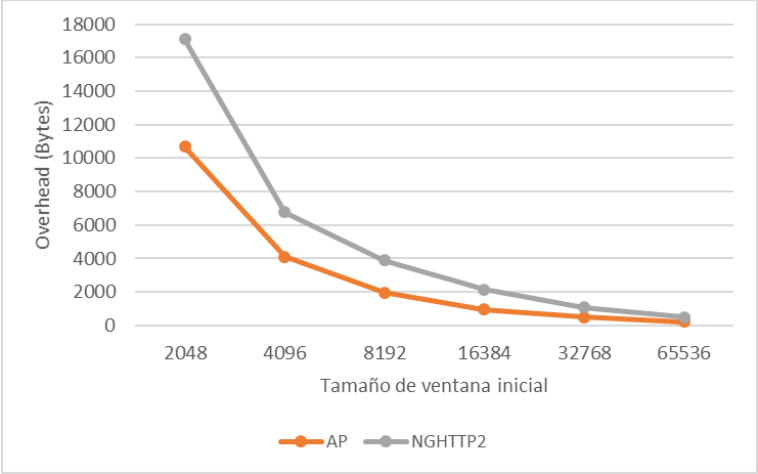


(c) *Throughput* 500 kbps - 500 kbps - *Delay* 200 ms

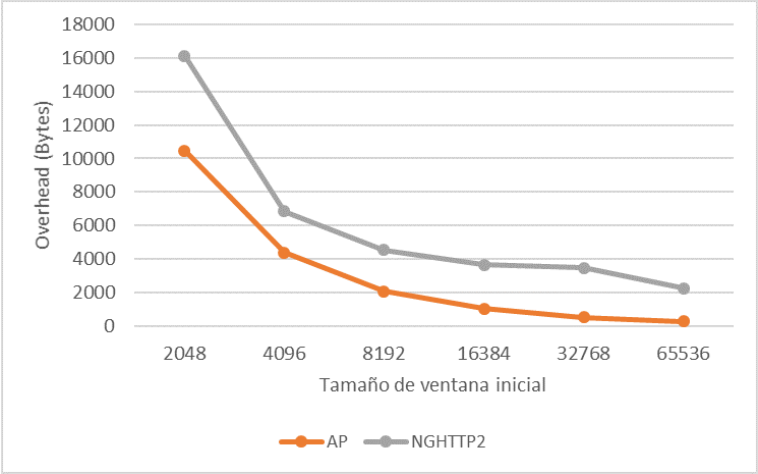
Figura 5.11: *Throughput* cuando el receptor y emisor tienen velocidades similares y el tiempo de propagación es 200 ms



(a) *Overhead* 10 kbps - 10 kbps - *Delay* 200 ms

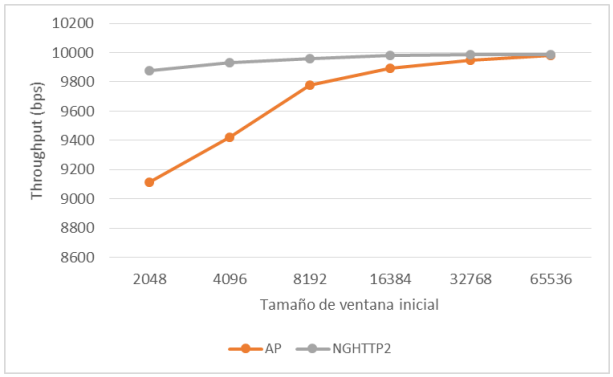


(b) *Overhead* 100 kbps - 100 kbps - *Delay* 200 ms

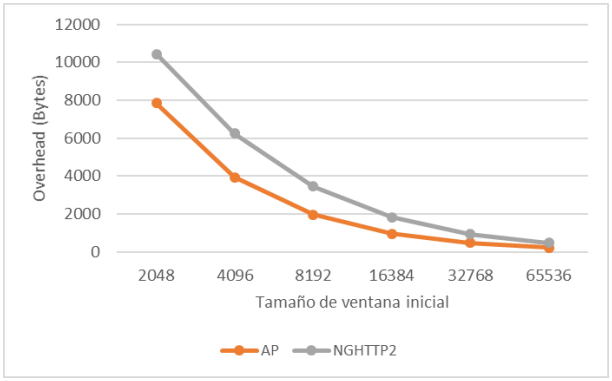


(c) *Overhead* 500 kbps - 500 kbps - *Delay* 200 ms

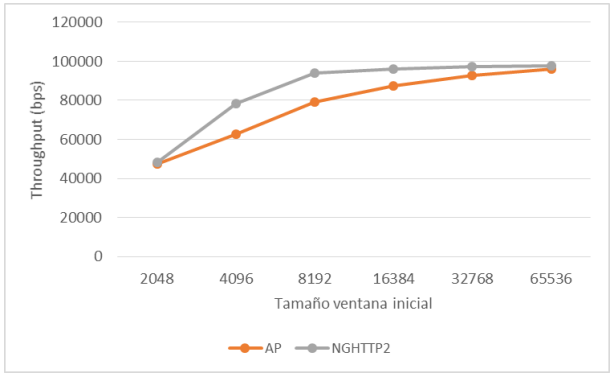
Figura 5.12: *Overhead* cuando el receptor y emisor tienen velocidades similares y el tiempo de propagación es 200 ms



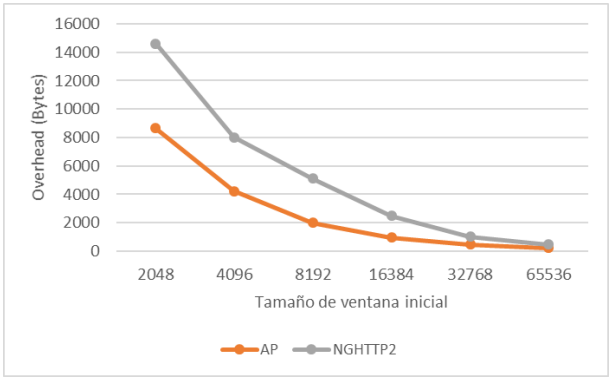
(a) *Throughput* 10 kbps - 100 kbps - *Delay* 100 ms



(b) *Overhead* 10 kbps - 100 kbps - *Delay* 100 ms

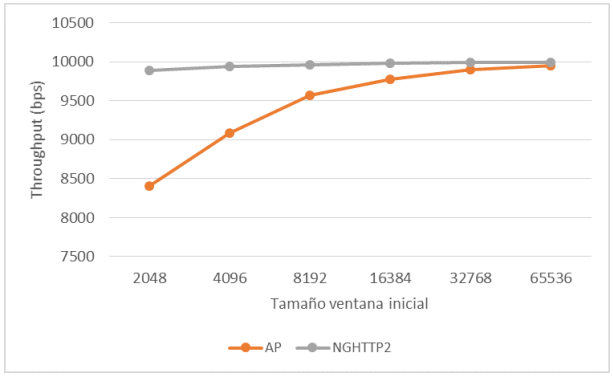


(c) *Throughput* 100 kbps - 500 kbps - *Delay* 100 ms

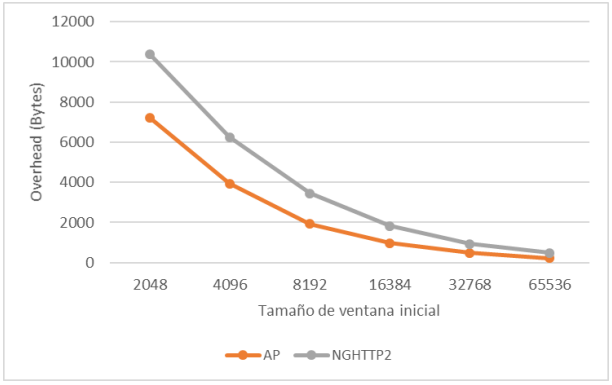


(d) *Overhead* 100 kbps - 500 kbps - *Delay* 100 ms

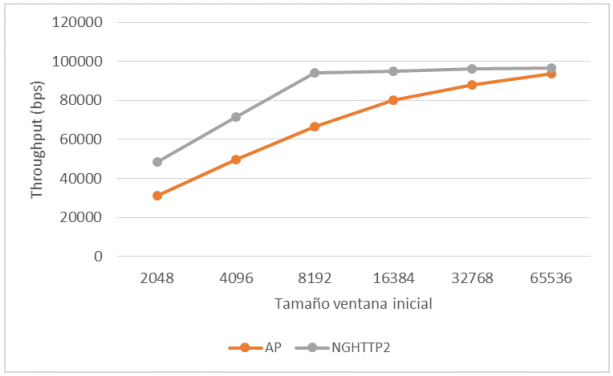
Figura 5.13: Resultados cuando el receptor es más rápido que el emisor



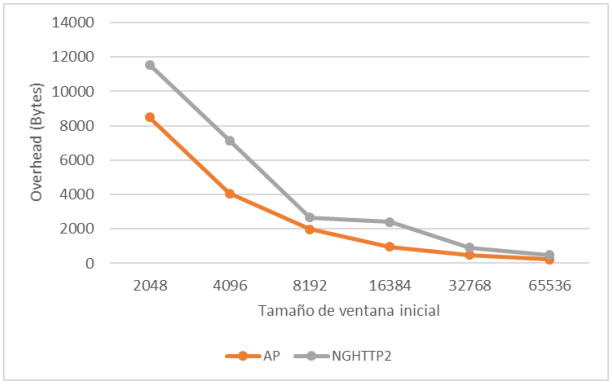
(a) *Throughput* 10 kbps - 100 kbps - *Delay* 200 ms



(b) *Overhead* 10 kbps - 100 kbps - *Delay* 200 ms

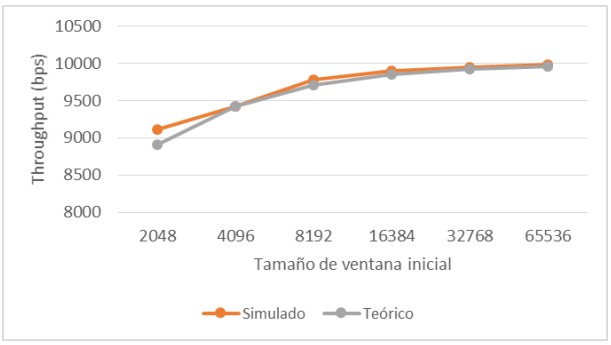


(c) *Throughput* 100 kbps - 500 kbps - *Delay* 200 ms

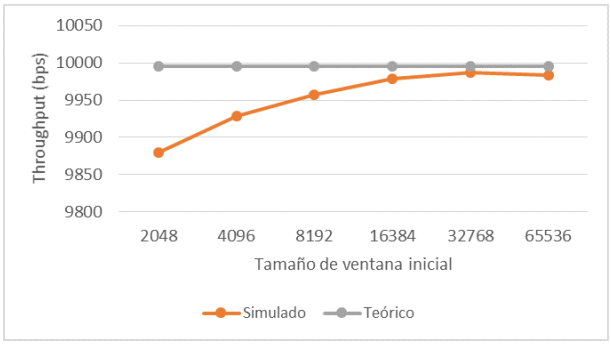


(d) *Overhead* 100 kbps - 500 kbps - *Delay* 200 ms

Figura 5.14: Resultados cuando el receptor es más rápido que el emisor y el tiempo de propagación es 200 ms.



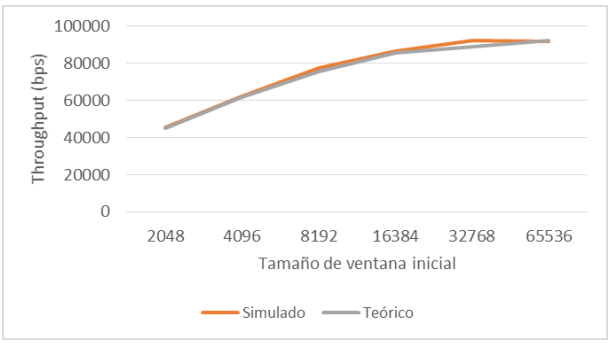
(a) *Throughput* de algoritmo propuesto para 10 kbps - 100 kbps - *Delay* 100 ms



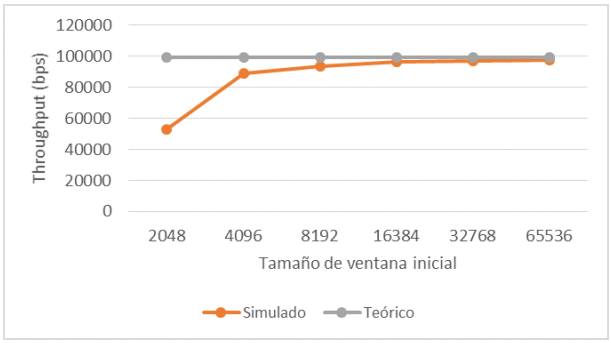
(b) *Throughput* de algoritmo de referencia para 10 kbps - 100 kbps - *Delay* 100 ms.

Figura 5.15: Comparación de resultados simulados y teóricos cuando el receptor es más rápido

como se ve en las gráficas. Cuando las ventanas iniciales son grandes, la fluidez es mayor y los resultados son muy similares. En el caso del algoritmo propuesto, tanto los resultados como el patrón que sigue el *throughput* a medida que aumenta el tamaño de la ventana son cercanos.



(a) *Throughput* de algoritmo propuesto para 100 kbps - 100 kbps - *Delay* 100 ms

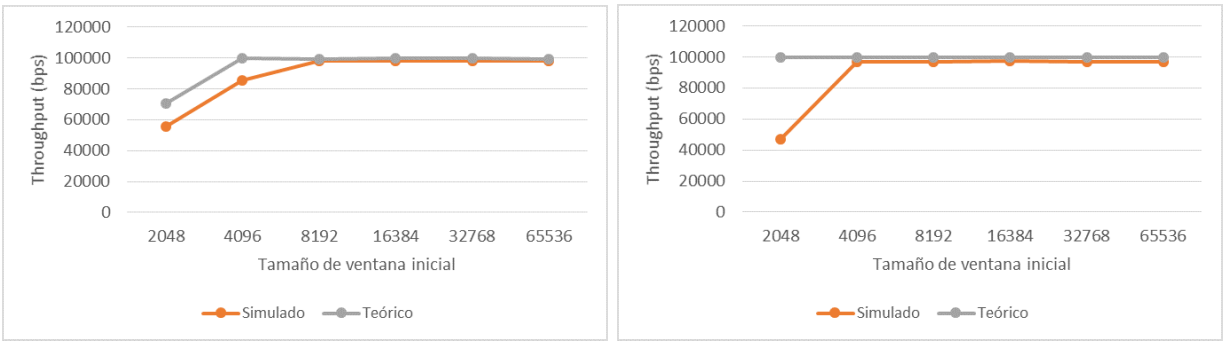


(b) *Throughput* de algoritmo de referencia para 100 kbps - 100 kbps - *Delay* 100 ms

Figura 5.16: Comparación de resultados simulados y teóricos cuando el receptor y emisor tienen velocidades similares.

En la Figura 5.16 se muestra el *throughput* cuando las velocidades de ambos nodos son iguales. En el caso del algoritmo propuesto, los resultados son muy similares; no obstante, en el caso del algoritmo de NGHTTP2 se muestra nuevamente una gran diferencia para la ventana de 2048 Bytes. Esto demuestra que una ventana muy pequeña y un *delay* alto afectan la transmisión, ya que obligan al algoritmo a hacer pausas. Cabe destacar que, esta es una situación que no se considera en el planteamiento matemático. Para las demás ventanas los resultados son acertados.

En la Figura 5.17 se muestra el *throghput* cuando el emisor es más rápido que el receptor. Como en los escenarios anteriores, para las ventanas más grandes las diferencias son mínimas. De nuevo, hay una marcada desigualdad en la ventana de 2048 Bytes del algoritmo de NGHTTP2 por los motivos ya mencionados. En cuanto al algoritmo propuesto, también se notan diferencias en las ventanas pequeñas, en las



(a) *Throughput* de algoritmo propuesto para 500 kbps - 100 kbps - *Delay* 100 ms

(b) *Throughput* de algoritmo de referencia para 500 kbps - 100 kbps - *Delay* 100 ms

Figura 5.17: Comparación de resultados simulados y teóricos cuando el emisor es más rápido.

que el *throughput* simulado es menor que el teórico. Los motivos pueden ser semejantes a los que se han explicado para el algoritmo de NGHTTP2, aunque, también se podría sumar que las ventanas pequeñas implican un procesamiento mayor en los nodos, ya que involucran más tráfico de control.

Conclusiones

El algoritmo propuesto cumple con su objetivo de disminuir el tráfico de control, pues dicha disminución fue entre el 25 % y 69 %, que equivale a un valor significativo para la simplificación del protocolo. Por otro lado, los resultados del *throughput* fueron mixtos, porque para la mayoría de los casos el algoritmo de NGHTTP2 fue ligeramente más rápido, a pesar de que, en el escenario en el que el receptor es muy lento en relación al emisor, el algoritmo propuesto tuvo un mejor desempeño. Este comportamiento es relevante para IoT, en el que receptores restringidos pueden ser comunes.

Cuando las velocidades de envío y lectura son similares, las condiciones de la red toman mayor relevancia, especialmente para el algoritmo propuesto que es más sensible al aumento en el *delay*. El incremento en los tiempos de propagación hace que, en general, se utilice menos paquetes WINDOW_UPDATE, pero disminuya el *throughput*.

Con base en lo anterior, se puede concluir que, el algoritmo propuesto es una buena opción para IoT, ya que simplifica el tráfico intercambiado en porcentajes importantes, sin sacrificar demasiado el *throughput*. Siendo este último un factor que en las comunicaciones *Machine to Machine*, típicas en IoT, no es tan crucial como en las comunicaciones que implican humanos.

Finalmente, en cuanto a la comparación de resultados teóricos y simulados, se puede decir que, en relación a los resultados simulados, el planteamiento matemático es acertado para la mayoría de los casos. Se presentan diferencias particularmente cuando la ventana inicial es 2048 Bytes. Para todos los casos, las ventanas muy pequeñas implican mayor procesamiento debido a un mayor *overhead*. En NGHTTP2,

en particular, además del procesamiento, las ventanas muy pequeñas reducen el *throughput* porque implica que el algoritmo haga pausas y esto no fue considerado en el planteamiento matemático.

5.4.5. Otros parámetros a considerar

En el desarrollo de los experimentos, se exploró la funcionalidad *Server Push*, la cual busca ahorrar peticiones enviando por adelantado recursos dada una solicitud determinada. Lo anterior suena bien en teoría, sobre todo cuando se transfieren muchos recursos; sin embargo, algunos autores ya han hablado del cuidado y las estrategias que hay que tener al usar *Server Push* para que no resulte contraproducente [101]. De hecho, los experimentos realizados mostraron un incremento considerable en el tiempo de carga de una página web al usar *Server Push*. En vista de lo anterior, y sumado a la simplicidad del tráfico de IoT y de los protocolos orientados a escenarios restringidos, es razonable pensar que esta funcionalidad debería estar deshabilitada en una implementación orientada a IoT.

Las prioridades de *streams* es otra funcionalidad que proporciona HTTP/2. Esta permite que el cliente le manifieste al servidor qué recursos requiere con mayor urgencia. De esta manera, el servidor podría priorizarlos. Una vez más, esta funcionalidad está orientada a aplicaciones que requieren transmitir un gran número de recursos, contrario a lo que sucede en IoT. Acudiendo a la simplicidad, esta funcionalidad no es necesaria en IoT y podría ser deshabilitada de la manera que se viene discutiendo en el grupo de httpbis de IETF mediante el *Internet-draft* [59].

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

En este trabajo se hacen una serie de pruebas y simulaciones que buscan evaluar la idoneidad de HTTP/2 para escenarios restringidos de IoT. Se parte de la hipótesis de que HTTP/2 es un protocolo que se puede adaptar a un escenario de IoT sin sacrificar demasiado el desempeño del mismo, ni comprometer la compatibilidad con el estándar. El amplio uso de HTTP y HTTP/2 en IoT y en la Internet en general, la interoperabilidad que ofrece (evita el uso de servidores Proxy) y la seguridad, son las principales motivaciones de este trabajo.

En un primer acercamiento experimental al protocolo, se evalúa HTTP/2 frente a CoAP en una red con distintos niveles de congestión (véase Sección 4.1). Los resultados muestran que, para una red confiable, CoAP tiene un mejor desempeño en términos de tiempos de carga de recursos, debido a que es un protocolo significativamente más ligero que HTTP/2. Por otro lado, este último tiene mejores resultados cuando la red se congestiona, puesto que ante la necesidad de retransmitir algunos paquetes, HTTP/2 lo hace más rápido. En IoT, las redes en ocasiones no son confiables, por lo que este es un resultado favorable para HTTP/2.

En la sección 4 se evalúa el tiempo de carga de los recursos y el trabajo realizado por una Raspberry Pi 3 ante una solicitud, variando el tamaño de la ventana de control de flujo en NGHTTP2. Los resultados de las dos métricas evaluadas llevan a concluir que 4096 Bytes es el valor adecuado como tamaño inicial de la ventana de control de flujo. Resultado que está acorde a lo considerado en el *Draft* [69].

HTTP/2, en la RFC 7540 [25], define el control de flujo, sin embargo, da libertad a los desarrolladores para utilizar esta funcionalidad como mecanismo para mejorar el desempeño de sus aplicaciones, por lo que no establece un algoritmo que determine el detalle de su funcionamiento. En la sección 5.2 se propone un algoritmo de control de flujo basado en la simplicidad que se debe tener en escenarios restringidos, cumpliéndose así el tercer objetivo de la tesis. El algoritmo propuesto es comparado mediante

simulaciones con un algoritmo de referencia, similar al usado por NGHTTP2. Los resultados muestran que el algoritmo propuesto tiene buen desempeño en términos de *throughput* y *overhead*, sobre todo cuando el dispositivo receptor es más lento que el emisor. La reducción de *overhead* es del 25 % al 65 %, verificándose la hipótesis para la mayoria de los casos, sobre todo cuando el receptor es lento y cuando se aumentan los tiempos de propagación. Por lo que se concluye que, el algoritmo propuesto es una buena opción para las implementaciones de HTTP/2 orientadas a escenarios restringidos.

La sección 5.4.5 muestra algunos otros parámetros a considerar en la simplificación del protocolo HTTP/2. Se considera que funcionalidades como *Server Push* y la priorización de *streams* no son necesarias en escenarios restringidos, por lo que en las implementaciones las pueden deshabilitar.

6.2. Trabajo futuro

En el trabajo futuro se espera poder construir una implementación de HTTP/2 para IoT con las recomendaciones mencionadas en el presente trabajo. Dicha implementación deberá ser probada sobre dispositivos restringidos e interactuar con implementaciones ya hechas y que sigan el estándar de la RFC 7540, con el fin de probar la interoperabilidad.

Por último, los parámetros considerados en la Sección 5.4.5 podrían ser evaluados para aplicaciones de IoT, especialmente el *Server Push*, el cual, en principio, ahorra algunos paquetes en la transferencia de datos. O en su defecto, se confirme que dichas funcionalidades no son necesarias en IoT y que su omisión aligeraría la implementación, como ha sido planteado en este trabajo.

Apéndice A

Anexos

A.1. Códigos de MATLAB para algoritmo propuesto

A.1.1. Código del algoritmo propuesto usado cuando el receptor es más rápido

```
clc
clear all

Benv=500000;                % Bytes transmitidos
Vi=10000;                   % Velocidad de envio en bps
Vo=100000;                  % Velocidad de lectura en bps
Ttrans_rec=0.1;             % Ttrans-rec
Trec_trans=0.1;             % Trec-trans
K=65536;                    % Tamano inicial de la ventana (2048 4096 8192
    16384 32768 65536)

Ttrans=1/(Vi/8);            %  Ttrans
Trec=1/(Vo/8);              % Trec

 %% Hallar W
p= Vi/Vo;                   % Relacion entrada/salida de la cola (rho)
N=(p/(1-p))-((K+1)*p^(K+1))/(1-p^(K+1)); % Ocupacion media de la cola de
     tamano K
W=floor(K-N);                 % Tamano promedio de la ventana disponible

 %% Hallar el tiempo total de la transmision

 % Primero se halla el numero de ventanas anunciadas
numV=Benv/W;
numVent=(numV);

Ttotal=numVent*((W*Ttrans)+Ttrans_rec+Trec_trans);

```

```
%% Hallar throughput promedio total

ThTotal=Benv*8/Ttotal;
```

A.1.2. Codigo del algoritmo popuesto usado cuando el receptor es más lento

```
clc
clear all

Benv=500000;                % Bytes transmitidos, ej: 500 KB
Vt=500000;                  % Velocidad del transmisor en bps, ej: 500 kbps
Vr=100000;                  % Velocidad de lectura del receptor en bps, ej:
    100 kbps
W0=4096;                    % Tamano inicial de la ventana
Tt_r=0.1;                   % Tiempo de propagacion transmisor-receptor
Tr_t=0.1;                   % Tiempo de propagacion receptor-transmisor
Btrans=0;                   % Contador de Bytes transmitidos
W=[];                       % Vector de ventanas anunciadas
Tt=[];                       % Vector de tiempos de transmision de cada
    rafaga

%%Transmision inicial con Wo (Ventana inicial (valor maximo))
Tt(1)=8*W0/Vt;              % Tiempo de transmision de la primera rafaga
    de valor W0
W(1)=floor(Vr*Tt(1)/8);     % Primera ventana anunciada
Btrans=W0;                  % Ya se transmitio la primera rafaga

%%Transmision de las siguientes rafagas
i=2;                        % indice de ventanas
while Btrans<Benv
    Tt(i)=(8*W(i-1)/Vt)+Tt_r+Tr_t;   % Tiempo de transmision de la rafaga
        i
    W(i)=floor((Vr*(Tt(i-1))/8));  % Ventana Wi
    if W(i)>W0
        W(i)=W0;            % Validacion de cota superior
    end
    if W(i)<0
        W(i)=0;             % Validacion de cota inferior
    end
    Btrans=Btrans+W(i);
    i=i+1;                  % Se aumenta el indice
end

Wprom=floor(mean(W));       % Tamano promedio de la ventana
numVent=floor(Benv/Wprom);  % Numero aproximado de ventanas necesarias
    para la transmision
Ttotalv=sum(Tt);            % Sumatoria de los tiempos de transmision
Tad=(W0-Wprom)*8/Vr;        % Tiempo de lectura adicional (final de
    transmision)

%% Hallar throughput proemdio total

```

```
41 ThTotalv=Benv*8/(Ttotalv+Tad+Tt_r);      % Throughput promedio de la
       transmision
```

A.1.3. Código del algoritmo de NGHTTP2 cuando hay cuello de botella

```
1  clc
2  clear all
3
4  Benv=500000;                % Bytes transmitidos
5  Vi=500000;                  % Velocidad de envio en bps **Para 0.1 de delay
       Bps/1.1 aprox
6  Vo=100000;                  % Velocidad de lectura en bps
7  Ttrans_rec=0.1;             % Ttrans-rec
8  Trec_trans=0.1;             % Trec-trans
9  K=32768;                    % Tamano inicial de la ventana 2048 4096 8192
       16384 32768 65536
10
11 Ttrans=1/(Vi/8);            % Ttrans
12 Trec=1/(Vo/8);              % Trec
13
14 %% Hallar Tiempo total de transmision
15 % Hallar el cuello de botella
16 Vb=min(Vi,Vo);              % Velocidad del cuello de botella (emisor o
       receptor)
17
18 Tideal=8*Benv/Vb;           % Tiempo ideal de transmision o tiempo total de
       lectura
19 Treal=Tideal+Ttrans_rec+Trec_trans; % Tiempo real de la transmision (
       tiempo de lectura mas propagacion inicial)
20
21 %% Hallar el throughput
22 ThTotal=8*Benv/Treal;
```

A.2. Código de MATLAB para el algoritmo de NGHTTP

```
1  clc
2  clear all
3
4  Benv=500000;                % Bytes transmitidos
5  Vi=99999;                   % Velocidad de envio en bps **Para 0.1 de delay
       Bps/1.1 aprox
6  Vo=100000;                  % Velocidad de lectura en bps
7  Ttrans_rec=0.1;             % Ttrans-rec
8  Trec_trans=0.1;             % Trec-trans
9  K=2048;                     % Tamano inicial de la ventana 2048 4096 8192
       16384 32768 65536
10
```

```
11 Ttrans=1/(Vi/8);              % Ttrans
12 Trec=1/(Vo/8);                % Trec
13
14 %% Hallar W
15 p= Vi/Vo;                      % Relacion entrada/salida de la cola (rho)
16 N=(p/(1-p))-((K+1)*p^(K+1))/(1-p^(K+1)); % Ocupacion media de la cola de
       tamano K
17 W=floor(K-N);                  % Tamano promedio de la ventana disponible
18
19 %% Hallar el tiempo total de la transmision
20
21 % Primero se halla el numero de ventanas anunciadas
22 numV=Benv/W;
23 numVent=(numV);
24
25 Ttotal=numVent*((W*Ttrans))+Ttrans-rec+Trec-trans;
26
27 %% Hallar throughput promedio total
28
29 ThTotal=Benv*8/Ttotal;
```

A.3. Resultados de simulaciones

	delay	Vetana inicial	10K-10K	10K-100K	100K-10K	100K-100K	100K-500K	500K-10K	500K-100K	500K-500K
Algoritmo propuesto	100ms	2048	9100	9112	9930	45379	47478	9915	55425	52295
		4096	9584	9422	9932	62394	62822	9928	85129	109439
		8192	9798	9778	9931	77563	79281	9923	97952	199730
		16384	9928	9893	9941	86578	87502	9924	98130	289582
		32768	9945	9949	9941	92385	92904	9925	98058	359034
		65536	9943	9979	9934	91743	96036	9924	98128	398644
	200ms	2048	8332	8403	9933	27262	30956	9925	39441	37051
		4096	9193	9080	9929	49120	49577	9921	57885	79495
		8192	9604	9565	9926	66890	66469	9926	98292	142242
		16384	9835	9773	9928	79703	80218	9926	98183	219346
		32768	9946	9893	9926	87906	87991	9924	98080	298173
		65536	9946	9946	9929	92699	93622	9923	97361	362384
NGHTTP2	100ms	2048	9930	9879	9893	52766	48429	9916	46688	47968
		4096	9940	9929	9905	89156	78157	9921	96819	314366
		8192	9937	9957	9872	93815	94060	9921	97026	368833
		16384	9932	9979	9920	96387	95865	9911	97608	380711
		32768	9930	9987	9920	97167	97283	9908	96913	369480
		65536	9940	9984	9922	97330	97538	9933	96772	366401
	200ms	2048	9938	9882	9936	30971	48302	9918	47437	41199
		4096	9940	9939	9923	74027	71612	9915	89966	127000
		8192	9931	9959	9931	92714	93931	9915	97354	260247
		16384	9940	9981	9931	94741	94919	9917	97252	379146
		32768	9937	9990	9932	96045	96084	9917	98335	378895
		65536	9940	9989	9932	96639	96546	9905	95533	380191

Tabla A.1: Throughput en Bytes variando tamaño velocidades de envío y lectura, tamaño de ventanas y delay.

Delay	Tamaño de ventana	Algoritmo	10K-10K	10K-100K	100K-10K	100K-100K	100K-500K	500K-10K	500K-100K	500K-500K
100 ms	2048	AP	249	245	2084	287	270	2424	408	428
		NGH2	493	326	2820	440	456	3669	730	495
	4096	AP	124	123	2073	134	132	2423	247	180
		NGH2	466	195	2806	204	250	3647	708	157
	8192	AP	63	61	2054	64	62	2422	208	74
		NGH2	398	108	2779	175	160	3603	701	135
	16384	AP	30	30	2014	32	30	2417	203	35
		NGH2	257	57	2732	76	78	3532	681	118
	32768	AP	17	15	1937	16	15	2354	195	17
		NGH2	53	29	2631	35	32	3401	662	122
	65536	AP	8	7	1785	8	7	2185	177	9
		NGH2	18	15	2435	16	15	3153	607	86
200 ms	2048	AP	249	226	1111	334	265	1297	302	327
		NGH2	492	325	1656	534	361	1918	531	504
	4096	AP	123	123	1106	128	127	1291	199	137
		NGH2	465	195	1645	211	223	1907	443	214
	8192	AP	61	60	1097	61	62	1280	105	64
		NGH2	397	108	1630	121	83	1883	409	141
	16384	AP	30	30	1077	30	30	1259	103	32
		NGH2	248	57	1600	67	75	1835	395	114
	32768	AP	15	15	1035	15	15	1216	98	16
		NGH2	51	29	1543	34	28	1757	374	108
	65536	AP	8	7	954	7	7	1128	90	8
		NGH2	18	15	1428	15	15	1632	327	70

Tabla A.2: Overhead en Bytes usado al variar velocidades de envío y lectura, tamaño de la ventana y delay.

Bibliografía

[1]] HTTP/2, High Performance Browser Network. https://hpbn.co/http2/. [Accedido: 06-sept-2017].

[2] Erbium (Er) REST Engine - CoAP Implementation. http://people.inf.ethz.ch/mkovatsc/erbium.php. [Accedido: 12-feb-2017].

[3] Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST). https://www.ics.uci.edu/{~}fielding/pubs/dissertation/rest{_}arch{_}style.htm. [Accedido: 06-feb-2017].

[4] Future internet of things (fit) iot-lab. https://www.iot-lab.info/. [Accedido: 17-Ago-2019].

[5] Intel® IoT Platform Reference Architecture. https://www.intel.com/content/www/us/en/internet-of-things/white-papers/iot-platform-reference-architecture-paper.html. [Accedido: 06-feb-2017].

[6] Internet of Things: Overview - Architecture Center - IBM Cloud Garage Method. https://www.ibm.com/devops/method/content/architecture/iotArchitecture/. [Accedido: 06-feb-2017].

[7] Microsoft Azure IoT reference architecture available. https://azure.microsoft.com/en-us/updates/microsoft-azure-iot-reference-architecture-available/. [Accedido: 06-feb-2017].

[8] SPDY Protocol - The Chromium Projects. https://www.chromium.org/spdy/spdy-protocol. [Accedido: 06-sept-2017].

[9] XMPP | An Overview of XMPP. https://xmpp.org/about/technology-overview.html. [Accedido: 05-ene-2017].

[10] X.805 : Security architecture for systems providing end-to-end communications. https://www.itu.int/rec/T-REC-X.805-200310-I/en,

2004. [Accedido: 12-feb-2017].

[11] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.xml, 2012. [Accedido: 10-feb-2017].

[12] A technical overview of LoRa ® and LoRaWAN ™. https://docs.wixstatic.com/ugd/eccc1a{_}ed71ea1cd969417493c74e4a13c55685.pdf, 2015. [Accedido: 10-feb-2017].

[13] Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709, April 2016.

[14] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015.

[15] T. A. Alghamdi, A. Lasebae, and M. Aiash. Security analysis of the constrained application protocol in the internet of things. In *Second International Conference on Future Generation Communication Technologies (FGCT 2013)*, pages 163–168, Nov 2013.

[16] E. Aljarrah, M. B. Yassein, and S. Aljawarneh. Routing protocol of low-power and lossy network: Survey and open issues. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–6, Sept 2016.

[17] Arnold O Allen. *Probability, statistics, and queueing theory*. Academic press, 2014.

[18] Apache. HTTP Server Project. https://httpd.apache.org/. [Accedido: 04-sept-2017].

[19] E. Arthurs, G. Chesson, and B. Stuck. Theoretical performance analysis of sliding window flow control. *IEEE Journal on Selected Areas in Communications*, 1(5):947–959, November 1983.

[20] E. Baccelli and D. Raggett. The promise of the internet of things and the web of things. *ERCIM News Special Issue on The Internet of Things and Web of Things*, pages 8–10, April 2015.

[21] E. Balandina, Y. Sheynin, Y. Koucheryavy, and S. Balandin. Protocol design for wireless extension of embedded networks: Overview of requirements and challenges. In *2014 International SpaceWire Conference (SpaceWire)*, pages 1–4, Sept 2014.

[22] A. Bassi, P. Giacomin, K. Koutsopoulos, A. Oliverau, M. Rossi, and J. Steffa. Guidelines for design. http://www.meet-iot.eu/deliverables-IOTA/D3_4.pdf, Sept 2013.

[Accedido: ene-2017].

[23] D. Basu, G. Sen Gupta, G. Moretti, and X. Gui. Investigation into the impact of protocol design on energy consumption of low power wireless sensors. In *2014 IEEE Sensors Applications Symposium (SAS)*, pages 69–74, Feb 2014.

[24] B. Bellalta. Ieee 802.11ax: High-efficiency wlans. *IEEE Wireless Communications*, 23(1):38–46, February 2016.

[25] M. Belshe, R. Peon, and M. Thomson. Hypertext transfer protocol version 2 (http/2). RFC 7540, RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7540.txt`.

[26] G. Bochmann and C. Sunshine. Formal Methods in Communication Protocol Design. *IEEE Transactions on Communications*, 28(4):624–631, 1980.

[27] C. Bormann, M. Ersue, and A. Keranen. Terminology for constrained-node networks. RFC 7228, RFC Editor, May 2014. `http://www.rfc-editor.org/rfc/rfc7228.txt`.

[28] Carsten Bormann. Using CoAP with IPsec. Internet-Draft draft-bormann-core-ipsec-for-coap-00, Internet Engineering Task Force, December 2012. Work in Progress.

[29] Carsten Bormann, August Betzler, Carles Gomez, and Ilker Demirkol. CoAP Simple Congestion Control/Advanced. Internet-Draft draft-ietf-core-cocoa-01, Internet Engineering Task Force, March 2017. Work in Progress.

[30] Carsten Bormann, Simon Lemay, Hannes Tschofenig, Klaus Hartke, Bill Silverajan, and Brian Raymor. CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets. Internet-Draft draft-ietf-core-coap-tcp-tls-09, Internet Engineering Task Force, May 2017. Work in Progress.

[31] R. Bush and D. Meyer. Some internet architectural guidelines and philosophy. RFC 3439, RFC Editor, December 2002. `http://www.rfc-editor.org/rfc/rfc3439.txt`.

[32] B. Carpenter, B. Aboba, and S. Cheshire. Design considerations for protocol extensions. RFC 6709, RFC Editor, September 2012.

[33] Brian E. Carpenter. Architectural principles of the internet. RFC 1958, RFC Editor, June 1996. `http://www.rfc-editor.org/rfc/rfc1958.txt`.

[34] A. Castellani, S. Loreto, A. Rahman, T. Fossati, and E. Dijk. Guidelines for mapping implementations: Http to the constrained application protocol (coap). RFC 8075, RFC Editor, February 2017.

[35] Y. Chen and T. Kunz. Performance evaluation of iot protocols under a constrained wireless access network. In *2016 International Conference on*

Selected Topics in Mobile Wireless Networking (MoWNeT), pages 1–7, April 2016.

[36] S. A. Chowdhury, V. Sapra, and A. Hindle. Client-side energy efficiency of http/2 for web and mobile app developers. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 529–540, March 2016.

[37] M. Collina, M. Bartolucci, A. Vanelli-Coralli, and G. E. Corazza. Internet of things application layer protocol analysis over error and delay prone links. In *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, pages 398–404, Sept 2014.

[38] R. d. J. Martins, V. G. Schaurich, L. A. D. Knob, J. A. Wickboldt, A. S. Filho, L. Z. Granville, and M. Pias. Performance analysis of 6lowpan and coap for secure communications in smart homes. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 1027–1034, March 2016.

[39] H. de Saxcé, I. Oprescu, and Y. Chen. Is http/2 really faster than http/1.1? In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 293–299, April 2015.

[40] D. J. Deng, K. C. Chen, and R. S. Cheng. Ieee 802.11ax: Next generation wireless local area networks. In *10th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 77–82, Aug 2014.

[41] diegold91. Simuladores de control de flujo - github. `https://bit.ly/2kuuheB`, Sep 2019. [Accedido: 10-sept-2019].

[42] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. `http://www.rfc-editor.org/rfc/rfc5246.txt`.

[43] Eclipse IoT WG, IEEE IoT, AGILE IoT, and IoT Council. IoT Developer Survey 2019. `https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2019.pdf`, 2019. [Accedido: 18-ago-2019].

[44] Asma Elmangoush. Evaluating the features of http/2 for the internet of things. `http://cit.edu.ly/wp-content/uploads/2018/02/10-037.pdf`, May 2017.

[45] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, June 2003.

[46] Stephen Farrell. LPWAN Overview. Internet-Draft

draft-ietf-lpwan-overview-06, Internet Engineering Task Force, July 2017. Work in Progress.

[47] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999. `http://www.rfc-editor.org/rfc/rfc2616.txt`.

[48] Giovanni Giambene. *Queuing Theory and Telecommunications: Networks and Applications. Springer NY, 2nd edition, 2014*. 04 2014.

[49] N. K. Giang, M. Ha, and D. Kim. Scoap: An integration of coap protocol with web-based application. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 2648–2653, Dec 2013.

[50] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. *Sensors*, 12(9):11734–11753, 2012.

[51] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660, 2013. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond.

[52] G. Held. *Server Management*. Best Practices. CRC Press, 2000.

[53] Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell, and Stephen Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 12–21, New York, NY, USA, 2014. ACM.

[54] P. Hynninen and M. Kauppinen. A/b testing: A promising tool for customer value evaluation. In *2014 IEEE 1st International Workshop on Requirements Engineering and Testing (RET)*, pages 16–17, Aug 2014.

[55] ITU-T. Internet of Things Global Standards Initiative. `http://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx`. [Accedido: 18-ago-2017].

[56] R. Karmakar, S. Chattopadhyay, and S. Chakraborty. Impact of ieee 802.11n/ac phy/mac high throughput enhancements on transport and application protocols x2013; a survey. *IEEE Communications Surveys Tutorials*, PP(99):1–1, 2017.

[57] R. Khan, A. K. Srivastava, and D. Pandey. Agile approach for software testing process. In *2016 International Conference System Modeling Advancement in*

Research Trends (SMART), pages 3–6, Nov 2016.

[58] Michael Koster, Ari Keränen, and Jaime Jimenez. Publish-Subscribe Broker for the Constrained Application Protocol (CoAP). Internet-Draft draft-koster-core-coap-pubsub-05, Internet Engineering Task Force, July 2016. Work in Progress.

[59] Bradford Lassey and Lucas Pardue. Declaring Support for HTTP/2 Priorities. Internet-Draft draft-lassey-priority-setting-00, Internet Engineering Task Force, July 2019. Work in Progress.

[60] C. Lerche, N. Laum, F. Golatowski, D. Timmermann, and C. Niedermeier. Connecting the web with the web of things: lessons learned from implementing a coap-http proxy. In *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)*, volume Supplement, pages 1–8, Oct 2012.

[61] Y. Liu, Y. Ma, X. Liu, and G. Huang. Can http/2 really help web performance on smartphones? In *2016 IEEE International Conference on Services Computing (SCC)*, pages 219–226, June 2016.

[62] Diego Londoño and Sandra Céspedes. Performance Evaluation of CoAP and HTTP/2 in Web Applications. `http://ceur-ws.org/Vol-1727/ssn16-final5.pdf`.

[63] Diego Londoño, Maite Gonzalez, Sandra Céspedes, Javier Bustos, and Gabriel Montenegro. Performance Evaluation of HTTP/2 Window Size in the Internet of Things. `http://ceur-ws.org/Vol-1950/paper8.pdf`, 2017.

[64] K. Lynn, J. Martocci, C. Neilson, and S. Donaldson. Transmission of ipv6 over master-slave/token-passing (ms/tp) networks. RFC 8163, RFC Editor, May 2017.

[65] Y. Ma, L. Wang, J. Cui, and B. Zheng. Design of p2p application layer protocol. In *2016 12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN)*, pages 175–178, Dec 2016.

[66] Q. Mamun and M. Kaosar. What is the first step in designing an application protocol for wireless sensor networks (wsns)? In *2014 IEEE Sensors Applications Symposium (SAS)*, pages 333–338, Feb 2014.

[67] P. Mariager, J. Petersen, Z. Shelby, M. Van de Logt, and D. Barthel. Transmission of ipv6 packets over digital enhanced cordless telecommunications (dect) ultra low energy (ule). RFC 8105, RFC Editor, May 2017.

[68] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of ipv6 packets over ieee 802.15.4 networks. RFC 4944, RFC Editor, September 2007. `http://www.rfc-editor.org/rfc/rfc4944.txt`.

[69] Gabriel Montenegro, Sandra Cespedes, Salvatore Loreto, and Robby Simpson. HTTP/2 Configuration Profile for the Internet of Things. Internet-Draft draft-montenegro-httpbis-h2ot-profile-00, Internet Engineering Task Force, March 2017. Work in Progress.

[70] N. Naik and P. Jenkins. Web protocols and challenges of web latency in the web of things. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 845–850, July 2016.

[71] Nghttp2. Nghttp2: HTTP/2 C Library and tools. `https://github.com/nghttp2/nghttp2`. [Accedido: 04-sept-2017].

[72] J. Nieminen, T. Savolainen, M. Isomaki, B. Patil, Z. Shelby, and C. Gomez. Ipv6 over bluetooth(r) low energy. RFC 7668, RFC Editor, October 2015.

[73] Amy Nordrum. Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated. `https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outda` September 2016.

[74] Oasis. MQTT Version 3.1.1. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf`, 2014. [Accedido: 10-feb-2017].

[75] Object Management Group. DDS 1.4. `http://www.omg.org/spec/DDS/1.4/`, 2015. [Accedido: 08-feb-2017].

[76] Gerardo Pardo-Castellote. OMG Data Distribution Service: Real-Time Publish/Subscribe Becomes a Standard. *Industry Insight*, 2005.

[77] J. Postel. User datagram protocol. STD 6, RFC Editor, August 1980. `http://www.rfc-editor.org/rfc/rfc768.txt`.

[78] Jon Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. `http://www.rfc-editor.org/rfc/rfc793.txt`.

[79] R. A. Rahman and B. Shah. Security analysis of iot protocols: A focus in coap. In *2016 3rd MEC International Conference on Big Data and Smart City (ICBDSC)*, pages 1–7, March 2016.

[80] E. Rescorla. Http over tls. RFC 2818, RFC Editor, May 2000. `http://www.rfc-editor.org/rfc/rfc2818.txt`.

[81] E. Rescorla and N. Modadugu. Datagram transport layer security version 1.2. RFC 6347, RFC Editor, January 2012. `http://www.rfc-editor.org/rfc/rfc6347.txt`.

[82] D. Ruz. Evaluación del protocolo http/2 para internet de las cosas. *Memoria*

de Título, Universidad de Chile, 2019.

[83] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. RFC 6120, RFC Editor, March 2011. http://www.rfc-editor.org/rfc/rfc6120.txt.

[84] J. Schaad. Cbor object signing and encryption (cose). RFC 8152, RFC Editor, July 2017.

[85] Göran Selander, John Mattsson, Francesca Palombini, and Ludwig Seitz. Object Security of CoAP (OSCOAP). Internet-Draft draft-ietf-core-object-security-04, Internet Engineering Task Force, July 2017. Work in Progress.

[86] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7252.txt.

[87] William Stallings. Queuing analysis. *WilliamStallings. com/StudentSupport. html*, 2000.

[88] Statista. Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/, 2020. [Accedido: 12-jan-2020].

[89] A. B. Sulaeman, F. A. Ekadiyanto, and R. F. Sari. Performance evaluation of http-coap proxy for wireless sensor and actuator networks. In *2016 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob)*, pages 68–73, Sept 2016.

[90] D. Thaler and B. Aboba. What makes for a successful protocol? RFC 5218, RFC Editor, July 2008.

[91] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan. Performance evaluation of mqtt and coap via a common middleware. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6, April 2014.

[92] The Chromium Projects. SPDY: An experimental protocol for a faster web - The Chromium Projects. https://www.chromium.org/spdy/spdy-whitepaper. [Accedido: 12-feb-2017].

[93] S. Thombre, R. Ul Islam, K. Andersson, and M. S. Hossain. Performance analysis of an ip based protocol stack for wsns. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 360–365, April 2016.

[94] P. van der Stok, C. Bormann, and A. Sehgal. Patch and fetch methods for the constrained application protocol (coap). RFC 8132, RFC Editor, April 2017.

[95] Wi-SUN Alliance. Comparing IoT Networks at a Glance. `https://www.wi-sun.org/images/assets/docs/Wi-SUN-Alliance-Comparing{_}IoT{_}Networks-r1.pdf`, 2016. [Accedido: 10-feb-2017].

[96] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, and R. Alexander. Rpl: Ipv6 routing protocol for low-power and lossy networks. RFC 6550, RFC Editor, March 2012. `http://www.rfc-editor.org/rfc/rfc6550.txt`.

[97] T. Yokotani and Y. Sasaki. Comparison with http and mqtt on required network resources for iot. In *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pages 1–6, Sept 2016.

[98] Deze Zeng, Song Guo, and Zixue Cheng. The web of things: A survey (invited paper). *JCM*, 6:424–438, 09 2011.

[99] J. Zhou and G. Wei. Two patterns in conversion between http2 and coap : Request-reponse and publish-subscribe. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1178–1184, March 2019.

[100] Zigbee Alliance. Standards: ZigBee Specification. `http://www.zigbee.org/download/standards-zigbee-specification/`, 2014. [Accedido: 04-sept-2017].

[101] T. Zimmermann, J. Rüth, B. Wolters, and O. Hohlfeld. How http/2 pushes the web: An empirical study of http/2 server push. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9, June 2017.